TAXI EVENT EXTRACTOR

USER MANUAL

Tomasz Gradzki
Osama Alsalous
Mia Li
Nancy Kaur
Navid Mirmohammadsadeghi
Dr. Susan Hotle
Dr. Antonio Trani

Version 8
March 2023

Air Transportation Systems Laboratory

TABLE OF CONTENTS

TAXI EVENT EXTRACTOR (TEE) OVERVIEW

As airport traffic continues to increase, so does the need for a way to effectively and efficiently manage it. A large component of traffic management is the ability to direct and instruct flights from their origin to their destination so that no two flights interrupt one another. If one were able to approximate how long and what route a given flight would take to get from point A to point B, then it would make scheduling and routing the flights much simpler.

It was for that reason that the Taxi Event Extractor (TEE) was developed. With the implementation of Airport Surface Detection Equipment, Model X (ASDE-X) at multiple commercial airports came a trove of location data for all flights at those airports. When combined with identifying information for individual flights and airport layout data, it would allow analysts to determine the routes, times, and origin-destination pairs of all flights that occurred at a given airport, greatly expanding the possibilities of traffic prediction and management.

While the program's primary purpose was to determine the time and location of the Gate-Out, Wheels-Off, Wheels-On, Gate-In (OOOI) events, it quickly became clear that the data could be used for so much more. Delay is an unfortunate reality of many flights, and it can be tricky to determine exactly what or who was responsible for the delay. This is further made difficult by the primary source of information for these delays having come from ASPM, a database that is generated through voluntary carrier information. By using the data provided by ASDE-X and the program's inherent layout information, the TEE can determine not only where a given flight is at any given time, but also how long it has stayed there. This being the case, knowing how many flights are moving around and exactly where the aircraft is idling makes it possible to determine where delay occurs and potentially what or who is responsible.

Additionally, deviations in a flight's movements could be the result of an unexpected, rare event. Visualizing the flight's route, as well as how that route interacts with the routes of other flights, can offer a clue as to where and when these events occur. The TEE's purpose is to utilize the ASDE-X data to the greatest degree possible and there are many beneficial tools and data that can be extrapolated from ASDE-X going forward, especially for estimation of operational metrics and safety metrics.

1 QUICK GUIDE

This section is a guide to a quick start of the Taxi Event Extractor. There are seven main capabilities, extracting taxi events using Source_Code.py is the primary function of the extractor. The remaining capabilities are tools for validation and visualization.

Taxi Event Extraction (Source_Code.py)

This script executes the main function of the taxi event extractor. It takes ASDE-X data input as IFF files in csv format. It accepts airport layout input as mat files (generated by Airport Layout Extractor) or shape files. The main output is a csv files with all parsed events.

User input:

- **AirportName:** List of FAA three character identifier (ex: ['LGA'])
- **Dates:** The date(s) in YYYYMMDD format (ex: 20200931)
- **Unrestricted:** If True, will process all flights regardless of successful takeoff/landing. If False, will only process flights that have successfully taken off/landed
- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder\\data_surface_tracks\\LGA+ASDEX)
- **FileType:** Airport layout file type ('mat' or 'shp')
- **OutputDir:** Output directory (rootfolder\\output\\ParsedEvents)
- **Check_Deicing:** Turn on/off deicing algorithm (True or False)
- **Deicing_Profiles:** Save deicing speed profiles in output (True or False)
- **Deicing_Demand:** Collect deicing start/end of demand (True or False)
- **T_List:** list of taxiways used as a staging area for deicing start of demand. Only used when Check_Deicing and Deicing_Demand are True (can be left empty [])

Validation Plot (SourceCode_ValidationPlot.py)

This script saves a figure of flight tracks with parsed events locations overlaying the airport layout. It can plot multiple flights and multiple dates for an airport.

Note: Source_Code.py must be run for the dataset for SourceCode_ValidationPlot.py to work correctly. If this is not possible or the parsed event locations (gate in/out, takeoff/landing, etc.) are not required, then use GraphFlightTracks.py instead.

User input:

- **AirportName:** FAA three character identifier (ex: 'LGA')
- **Dates:** The date(s) in YYYYMMDD format (ex: 20200931)
- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder\\data_surface_tracks\\LGA+ASDEX)
- **Result_File_Path:** Directory containing the Source_Code output for the input date(s) (ex: rootfolder\\output\\ParsedEvents)
- **Flight_Keys:** flight key(s) to be plotted (ex: [40337, 40345])

Flight Track Visualization (GraphFlightTracks.py)

This script saves a figure of flight tracks overlaying the airport layout. It can plot multiple flights and multiple dates for an airport.

User input:

- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Dates:** The date(s) in YYYYMMDD format (ex: [20201201])
- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder\\data_surface_tracks\\LGA+ASDEX)
- **Flight_Keys:** flight key(s) to be plotted (ex: [40337, 40345])

Animation Videos (Animation Batch.py)

This script saves videos of all flights within a specific time period and date in MP4 format.

User input:

- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Date:** The date in YYYYMMDD format (ex: 20201201)
- **Start_Time:** UTC start time of animation in HH24:MM:SS (ex: '08:35:00')
- **End_Time:** UTC end time of animation in HH24:MM:SS (ex: '12:08:00')
- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')
- **Unrestricted:** If True, will process all flights regardless of successful takeoff/landing. If False, will only process flights that have successfully taken off/landed
- **FileType:** Airport layout file type ('mat' or 'shp')
- **Check_Deicing:** display deicing pad area shaded in red (True or False)
- **Speed_Factor:** video frame rate in frames per second (fps), 1 fps = real time (ex: 10)

Heat maps (Heatmaps.py)

This script saves heat map figures of four taxiway metrics; average speed, average weight, total weight, total frequency. Three figures are created per metric to display departures, arrivals, and total operations. The output is a total of 12 figures, each set of figures can be generated for a specific period of time for a range of dates.

User input:

- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')
- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Start_Date:** The date in YYYYMMDD format (ex: 20201201)
- **End_Date:** The date in YYYYMMDD format (ex: 20201201)
- **Start_Time:** UTC start time in HH24:MM (ex: '09:00)
- **End_Time:** UTC end time in HH24:MM (ex: '16:00')
- **FileType:** Airport layout file type ('mat' or 'shp')
- **DecD:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.9)
- **DecA:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.7)
- **Colorbar:** Maximum value of the figure color bar (Default='vmax' for maximum value found from data. Otherwise it accepts an integer value provided by the user)
- **Output_Folder:** The file path where the output should be stored

Multiple Heat maps (Multimaps.py)

This script saves multiple heat map figures of four taxiway metrics; average speed, average weight, total weight, total frequency and compiles them into an animation. The output is a a set of animations that show how these characteristics change over time. Animations can be generated based off a user-defined time period and time increment.

User input:

- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')
- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Start_Date:** The date in YYYYMMDD format (ex: 20201201)
- **End_Date:** The date in YYYYMMDD format (ex: 20201201)
- **Start_Time:** UTC start time in HH24:MM (ex: '09:00)
- **End_Time:** UTC end time in HH24:MM (ex: '16:00')
- **Incrementor:** specifies the period of time covered by each individual heatmap in minutes (the higher the data, the more generalized the heatmaps become)
- **AverageSpeedON:** specifies if user wants to analyze for average speed (True or False)
- **AverageWeightON:** specifies if user wants to analyze for average weight (True or False)
- **TotalWeightON:** specifies if user wants to analyze for total weight (True or False)
- **FrequencyON:** specifies if user wants to analyze for frequency (True or False)
- **ArrivalsOnly:** specifies if user wants to analyze only arrival data (True or False)
- **DeparturesOnly:** specifies if user wants to analyze only departure data (True or False)
- **Both:** specifies if user wants to analyze only arrival data (True or False)
- **FileType:** Airport layout file type ('mat' or 'shp')
- **DecD:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.9)
- **DecA:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.7)
- **Colorbar1-Colorbar4:** Maximum value of each figure color bar (Default='vmax' for maximum value found from data. Otherwise it accepts an integer value provided by the user)
- **Output_Folder:** The file path where the output should be stored

ASPM Data Validation (ValidationASPM.py)

This script takes data from the Extractor results file and ASPM data file associated with the chosen airport and searches for matches between the two. The results are a series of Excel files that give the list of matched and unmatched flights, a time analysis between matched Extractor and ASPM flights, and a summary of the validation iteration process.

Note: Source_Code.py must be run for the dataset for ValidationASPM.py to work correctly.

User input:

- **AirportName:** FAA three character identifier (ex: 'ORD')
- **ASPM_Folder:** Folder containing ASPM Excel data files with AirportName in file name
- **TEE_Folder:** Folder containing Extractor results file with AirportName in file name
- **Results_Folder:** specifies the output folder for the validation results files
- **TZModifier:** specifies the time zone difference between ASPM data and GMT in hours
- **Dates:** The date(s) in YYYYMMDD format (ex: [20201201])
- **Start_Time:** UTC start time in HH24:MM (ex: '09:00)
- **End_Time:** UTC end time in HH24:MM (ex: '16:00')

2 DEPENDENCIES

This version of the Taxi Event Extractor was tested on Python 3.9. The program requires Python packages other than the standard to be installed in order to run. These packages are found in requirements.txt in "docs" folder. Below is a list of each required package along with the installed version at the time of testing the Taxi Event Extractor.

geopandas==0.6.2
matplotlib==3.3.4
numpy==1.21.2
openpyxl==3.0.7
pandas==1.3.4
scipy==1.6.2
Shapely==1.7.1

Notes on Geopandas

The installation of Geopandas may not be straightforward for all users depending on the Python environment on the machine. Geopandas is a package that makes working with geospatial data easier in Python by extending the functionality of Pandas. Geopandas relies on other packages to perform certain functions such as reading files, calculations, and plotting. Therefore, Geopandas has its own dependencies and installing it may not be simple. Below are two methods of installation that were successful on different machines during the development of the Taxi Event Extractor.

Method 1: Using pipwin

Execute the following command on the command prompt in order:

pip install pipwin
pipwin install numpy
pipwin install pandas
pipwin install shapely
pipwin install gdal
pipwin install fiona
pipwin install pyproj
pipwin install six
pipwin install rtree
pipwin install geopandas

You may need to copy DLLs (geos_c.dll) from "C:\ProgramData\Anaconda3\Lib\site-packages\shapely\DLLs" To "C:\ProgramData\Anaconda3\Library\bin"

Method 2: Using Anaconda's command prompt

1. Update the current Anaconda installation by typing the command `conda update --all -c conda-forge`, press enter when it says procced
2. Install the geopandas module using the command `conda install geopandas -c conda-forge`, press enter when it says procced
3. Check the installation using Spyder by typing "import geopandas" in the console window

For more information on installing Geopandas use the following URL:
https://wilcoxen.maxwell.insightworks.com/pages/6373.html

3 AIRPORT LAYOUT FILES

When using the Taxi Event Extractor program, the Layout_Loader.py function is responsible for loading the airport layout to meet the required format for the program to run. This version of the Taxi Event Extractor accepts two types of input files that the user can specify when running it.

The first type is the Aeronautical Information Exchange Model (AIXM) which is converted to a .mat file using the Airport Layout Extractor. For this option, two files containing airport layout information should be generated to be used as inputs to the program, specifically, the Airport Layout File and Runway Coordinates File. The second layout input type is shapefiles. This section outlines how to use each of the two file types.

3.1 Airport Layout File

To create the Airport Layout file in the correct format for the Taxi Event Extractor, one must use the Airport Layout Extractor. The Airport Layout Extractor is a standalone MATLAB GUI program that Virginia Tech created in a previous project for the FAA. It allows the user to input airport layouts as xml files that contain layout data in the Aeronautical Information Exchange Model (AIXM) format and output them as MAT files that contain parsed layout data that can be used as an input into the Taxi Event Extractor. The Airport Layout Extractor was completed by former Master's student Yang Zhang and Dr. Antonio Trani.

To install the Airport_Layout_Extractor.exe, click on agree for the MATHWORKS terms of use and let the installer automatically download the required MATLAB runtime package. This program does not require MATLAB to be installed on the computer. Figure 3.1 below shows the user interface of the application.

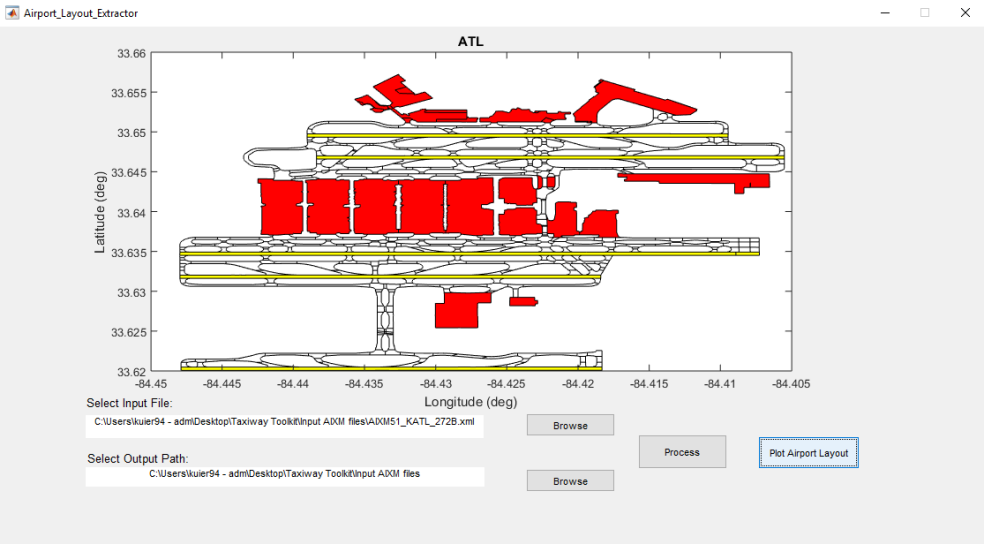


Figure 3.1- Airport Layout Extractor Application User Interface

As shown in Figure 1 above, once the input AIXM file and the output directory are specified, the extractor can start the data extraction and save the parsed data in the specified output directory. The application can also plot the parsed airport layout file for visualization and validation. The resulting Airport Layout File is one of the required inputs into the Taxi Event Extractor Python code. This Airport Layout File contains geographical information for the apron areas, runways, and taxiways.

The user needs to input ‘mat’ as file type in the source code to run this option.

3.2 Runway Coordinates File

Another airport data file that is required to run the Taxi Event Extractor is the Runway Coordinates File, the output of the Python script named Runway Coordinates Extractor. The Runway Coordinates Extractor reads the Layout Data for Airports and Other Landing Facilities (APT.txt), available on the following FAA website:

https://www.faa.gov/air_traffic/flight_info/aeronav/aero_data/NASR_Subscription/

The Python script extracts the runway threshold information and calculates the bearing of the runways using the runway thresholds’ coordinates. Then, using the width of the runway, the threshold coordinates, and the calculated bearing, the four runway vertices for each runway are calculated in the Python script.

The output file contains latitudes and longitudes of important runway points, and the runway names for all the runways at each of the forty-two airports for which we have AIXM files. This script allows one to specify the airport(s) to be updated from the APT data file, updated every 28 days, to ensure the taxi event extractor is using accurate runway information.

3.3 Shapefiles

Alternatively, the airport layout data may be inputted as a number of shapefiles. In this case, the Layout_Loader.py function looks for a group of shapefiles (.shp) and supporting files such as .dbf and .prj that contain data tables and projection information, respectively. These files are expected to be inside a folder named as the airport code inside a “shape_files” folder located in airport layout data folder. The naming convention for files is expected to follow the Feature and Attribute Coding Catalogue (FACC) Data Dictionary which is a part of the Digital Geographic Information Exchange Standard (DIGEST). FACC codes are 5 characters long, like how GB075 denotes taxiways information. More information is included within the Layout_Loader.py function. In order to run this option, the user needs to input ‘shp’ as file type in the source code.

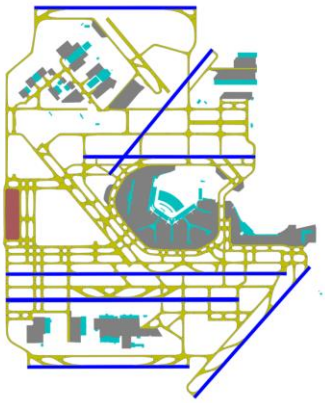


Figure 3.2 – Plot of shapefiles layout data for Chicago O'Hare (ORD)

4 MAIN EVENT EXTRACTOR PROGRAM: Source_Code.py

The main event extractor is run by Source_Code.py which calls supporting functions to execute the extraction capabilities. All Python files, airport layout files, and runway coordinates file should be saved in their appropriate folder in order to run the Taxi Event Extractor smoothly. The folder structure is shown in Figure 4.1 below and a flowchart of the code structure is available in Appendix C.

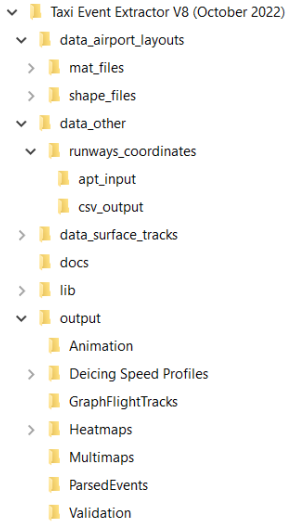


Figure 4.1 – Program Folder Structure

To execute the main Taxi Event Extractor program, one needs to run the Python file Souce_Code.py, which has the following required input arguments and optional deicing arguments:

```
# User input:

# Airport Name (FAA code)
AirportName = 'EWR' # ['CLT', 'EWR', 'JFK', 'LGA', 'PHL']
# Date - Leave blank for all dates. Otherwise format YYYYMMDD
Dates = []
# If only flights that successfully takeoff/land are desired, put False. Default is True
Unrestricted = True
# Input Flight Data Directory
Flight_Data_Path = os.path.join(workingdir, 'data_surface_tracks', AirportName +'+ASDEX') #'C:\\Taxi Event Extractor V#'
# Layout File type ['mat', 'shp']
FileType = 'mat'
# Output directory
OutputDir = os.path.join(workingdir, 'output', 'ParsedEvents')

# Deicing
Check_Deicing = False # Select True/False to turn on/off
Deicing_Profiles = False # Generate deicing speed profiles figures in output
Deicing_Demand = False # Select True or False to collect deicing start/end of demand
# t_list: list of taxiways to be used as staging area for deicing start of demand (otherwise leave empty [])
T_List = ['AA','BB','BB2','U','J1_U','AA_U','BB_U','J1','K1','J1_K1_BB1','AA_BB1','BB_BB1','BB2_K1','BB2_AA','BB2_BB']
```

Figure 4.2 – Screenshot of Input Arguments Example for Source Code

- **AirportName**: List of FAA three-character identifier (ex: ['LGA'])
- **Dates**: The date(s) in YYYYMMDD format (ex: 20200931)
- **Unrestricted**: If True, will process all flights regardless of successful takeoff/landing. If False, will only process flights that have successfully taken off/landed
- **Flight_Data_Path**: ASDE-X IFF input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')

- **FileType**: Airport layout file type ('mat' or 'shp')
- **OutputDir**: Output directory (rootfolder+'\\output\\ParsedEvents')
- **Check_Deicing**: Turn on/off deicing algorithm (True or False)
- **Deicing_Profiles**: Save deicing speed profiles in output (True or False)
- **Deicing_Demand**: Collect deicing start/end of demand (True or False)
- **T_List**: list of taxiways used as a staging area for deicing start of demand. Only used when Check_Deicing and Deicing_Demand are True (can be left empty [])

The output file will be saved in the specified output directory in .csv format with a row for each flight, in addition to any deicing fields if applicable. All output fields are listed in Appendix A.

4.1 Arrivals Methodology

The following figure shows the critical events for each arrival flight with points: 1) Enter Runway, 2) Wheels-On, 3) End of Landing Roll, 4) Exit Runway, 5) Exit Movement Area and 6) Enter Gate. For each event, the time, latitude, and longitude are noted in the data dictionary:

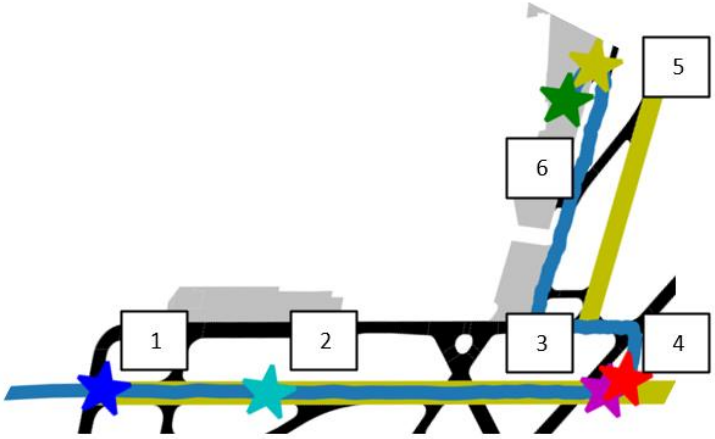


Figure 4.3 - Sample of an Arrival Flight at PHL and the Corresponding Events

***Operational Runway**:* The criteria for assigning an arrival runway is to have at least three consecutive points inside the runway polygon. Additionally, the program will calculate the current bearing of the flight and choose the runway end that matched the bearing. If the flight is heading approximately 90 degrees east of north, it will list 9R as the operational runway and not 27L. The procedure of checking for operational runway starts from the first point in arrival track and it stops as soon as a runway is assigned to the flight.

***Enter Runway (Labeled 1)*:** This is the first point of the arrival track detected as within the operational runway polygon. The program calculates the Enter Runway event by interpolating between the first point in runway polygon and the last point prior to the runway end. If interpolation is not possible (due to the unusual location of flight tracks), the Enter Runway time/location is the first point inside the runway.

Touchdown Point (Labeled 2): The algorithm checks a combination of ground speed loss of 5% and a continuous deceleration profile to define this event. This event could not be estimated using altitude due inaccurate variable readings. The 5% speed loss was chosen based on findings from

Dr. Toni Trani's study which uses landing/takeoff videos at ORD[1]. It is worth noting that this event specifically looks at a time period of 15 to 30 seconds after the Enter Runway event.

End of Landing Roll (Labeled 3): This point defines when the flight transitions from landing deceleration/braking to taxiing to the gate. Specifically, the program looks for the first point where the average of the aircraft's acceleration over the last 5 seconds falls below -0.5 m/s^2. This was done in case the acceleration only briefly falls below the threshold while braking. The program will look for this point in the time between when it first touches down to 30 seconds after exiting the runway. As a result, this point may fall before or after Exit Runway depending on the use of high-speed exit taxiways.

***Exit Runway (Labeled 4)*:** First point found outside of the runway polygon after touchdown.

Exit Movement Area (Labeled 5): This point marks where the flight moves from taxiways to the apron around the gate. For this point, the program will use the last moment where the flight transitions from a taxiway (black polygon) or runway (yellow polygon) to the apron (silver polygon)

Gate Point (Labeled 6): Estimated point where plane stops at gate. The program defines this point as the first point inside the apron where the plane does not move for 5 consecutive seconds. If no suitable point can be determined, then the program defaults to using the last point in the track.

Enter Runway On/Off Duration: Time difference between points 1 and 2.

Enter Runway On/Off Distance: Distance between points 1 and 2.

Enter Runway Exit Runway Duration: Time difference between points 1 and 4.

Taxi In/Out Duration: Time difference between points 4 and 6.

Taxiing Distance: Cumulative distance that airplane travels from the runway exit point to the near gate point. This distance is the travel distance from point 4 to point 6.

Average Taxi Speed: Smoothed taxiing speed from the runway exit point to the near gate point. This speed is the average of all the instant speeds from the point 4 to point 6.

Wait Time: All the moments that the airplane was taxiing with a speed lower than 3 m/s. This assumption was made to detect the places where airplanes were stuck in queues, when they had small changes in their instant location.

End of Landing Roll Speed: Speed at which the flight is travelling at the end of landing roll. It is worth noting that this value only represents the speed when acceleration falls below -0.5 m/s^2 and is not indicative of the aircraft's overall taxi speed. Refer both to this value and to the Average Taxi Speed to understand the flight's overall speed.

Altitude did not provide a good basis for the touchdown point due to its accuracy. Figure 4.4 shows how using the PDARS data the airplane goes below the reference altitude of the runway, which is not credible. This noisy behavior repeats for all of the other operations and more extreme situations were observed than the one shown in the graph.

[1] https://atsl.cee.vt.edu/content/dam/atsl_cee_vt_edu/redim/NoseGearDown_Dist_Aircraft.pdf

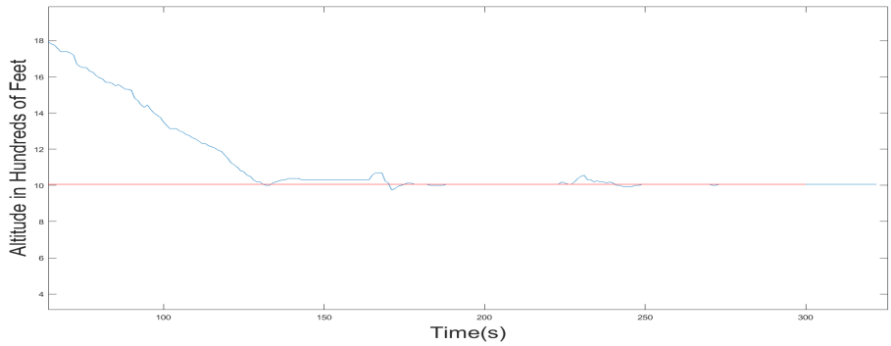


Figure 4.4 – Fuzzy Altitude Profile Example (Red Line is the Arrival Runway Altitude)

In order to extract the touchdown point, we implemented a more robust method instead of relying on altitude data. First, the speed at each track point was calculated using the information on location and time. While the data did include information on the speed of the plane, the data did not seem to be reliable as shown in Figure 4.5. Figure 4.6 shows the new calculated speed. Although Figure 4.6 shows a reasonable trend in speed change for an arrival flight, it still has noise that could affect the analysis. To reduce the amount of noise in the speed profile, the program utilizes the following procedure:

- Performs linear convolution on the speed profile using a kernel of [0.2, 0.2, 0.2, 0.2, 0.2]. What this does is that it effectively takes the average of 5-point clusters centered around a given speed.
    - So for the set of points [1, 3, 5, 3, 7, 3, 5], it would be the averages of the sets [1, 3, 5, 3, 7], [3, 5, 3, 7, 3], and [5, 3, 7, 3, 5] to get [4.8, 5.2, 5.6]
- The program then comes up with a "start" and "stop" pair of speeds to make sure the smoothed profile has the same number of points.
    - The first point in the "start" set is simply the initial speed value while the last point in the "stop" set is the final speed value (1 for the initial speed, 5 for the final speed).
    - The second point in the "start" set is the average of the first three points in the speed profile while the first point in the "stop" set is the average of the last three points in the speed profile ([1, 3, 5] to 3 for the second point, [7, 3, 5] to 5 for the second to last point.
    - This gives us a "start" set of [1, 3] and a "stop" set of [5, 5]
- The final step is to concatenate the vectors in the following order: "start", linear convolution, "stop."
    - For [1, 3], [4.8, 5.2, 5.6], and [5, 5], we would end with [1, 3, 4.8, 5.2, 5.6, 5, 5]

Therefore, the speed profiles were smoothed, and the results can be seen in Figure 4.7.

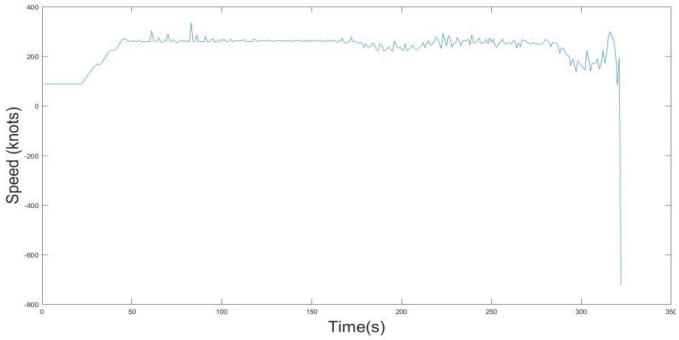


Figure 4.5 – Sample of Reported Speed for an Arrival Flight in Data

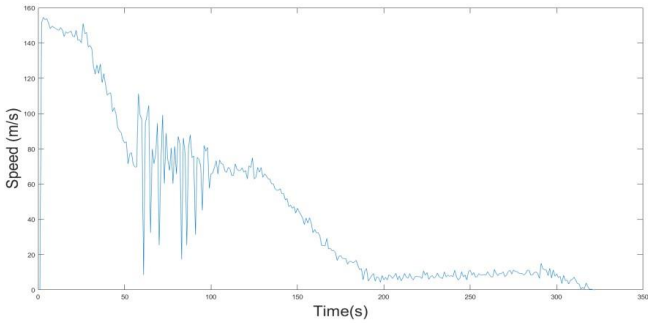


Figure 4.6 – Calculated Speed Profile in m/s

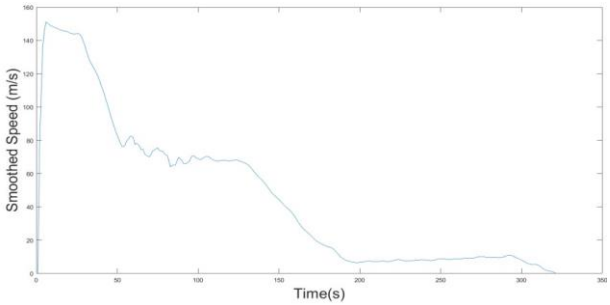


Figure 4.7 – Smoothed Speed Profile in m/s

After smoothing the speed profile, the acceleration was calculated. When the airplane touches the ground in a successful landing, the pilot applies the brakes in order to reduce the airplane speed (shown in Figure 4.8). Therefore, in the acceleration profile of the landing move, there should be a point with minimum acceleration (maximum deceleration) which helps the plane reduce its speed and vacate the runway. That moment of minimum acceleration happens after all the wheels are on the ground and the brakes are applied. Therefore, the acceleration profile should be continuously decreasing after the touchdown point.

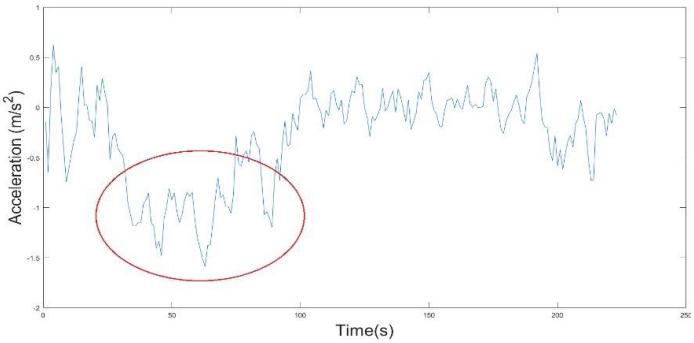


Figure 4.8 – Sample of an Acceleration Profile of an Arrival Flight

Although the event extractor was successful in parsing a high percentage of flights and assigning them to the operational runways, there were a few flights which could not be analyzed easily due to data drops or imprecise data. An additional check for gaps in recording time is performed for arrivals that have data drops before entering or inside the operational runway.

If there are data drops near runway threshold, defined as a gap greater than 10 seconds in recording time and distance to runway threshold greater than 150 meters, the event parse will use linear interpolation to estimate enter runway location instead of using the location of the first point inside the operational runway (red dot in Figure 4.9). Due to missing data, the speed of the airplane at runway threshold cannot be defined and it's difficult to estimate. Therefore, the methodology of 5% speed loss for defining touchdown location cannot be used in this case.

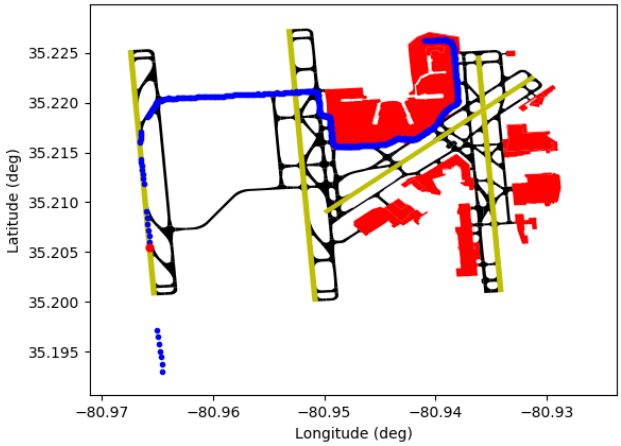


Figure 4.9 – Example of Arrival with Data Drops near Runway Threshold (Figure made during TEE V5)

Another situation occurs when there are data drops within the operational runway, defined as a gap greater than 10 seconds in recording time within 25 track points after entering the operational runway. In this case, the track point of 5% speed loss is likely missing from the data.

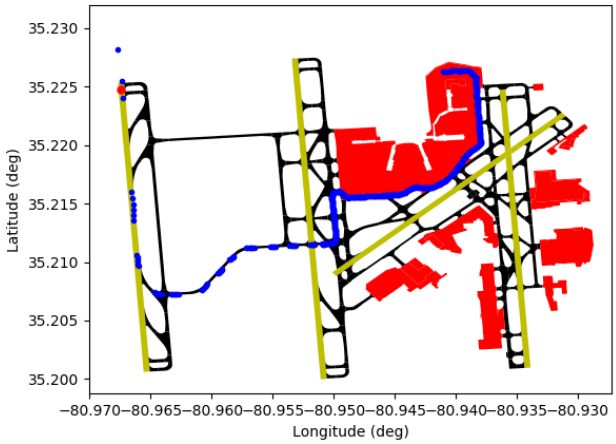


Figure 4.10 – Example of Arrival with Data Drops inside Operational Runway (Figure made during TEE V5)

For both cases (shown in Figures 4.9 and 4.10), "EnterRwy_On_Off_Dur" will be listed as "N/A Missing data" and other information relate to touchdown will be listed as "N/A" in the output table. The event parse will still be able to extract information related to taxiing if the operational runway can be defined.

Besides data drops, the track points occasionally showed unusual behavior and exited the runway at strange points, making the analysis more difficult. If the flight tracks did not meet the requirements of the event parser, i.e. cannot define operational runway or have runway occupation time less than 10 seconds, their operational runway will be listed as "No Lat/ Long Found" in the final output table, and the rest of the fields will be empty.

The Event Extractor is capable of recognizing go around movements for arrivals as well. The methodology for defining them is to check the average taxi speed. Because go around operations enter the runway polygon, spend time inside that polygon, and then leave, they have similar behavior to successful landings. However, they vacate the runway and fly away instead of taking an exit like a normal arrival. Hence, if an arrival flight has an average taxi speed of more than 30 m/s (~60 knots) it shows that it was a go around operation. Those flights are shown as "Go Around" in the final output table.

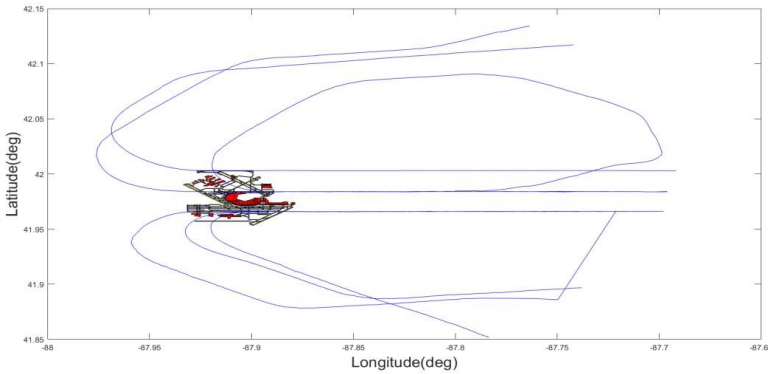


Figure 4.11 – Example of Defined Go Around Operations at ORD (Figure made during TEE V5)

4.2 Departures Methodology

The events extracted are shown in Figure 4.12, which includes: 1) Exit Gate, 2) Enter Movement

Area, 3) Enter Runway, 4) Start of Takeoff Roll, 5) Takeoff Point and 6) Exit Runway. All the fields extracted for arrivals and departures have similar naming in the final output table and occupy the same column. The criteria for finding an operational runway for departures is a little bit more complicated than arrivals, since the departure tracks pass more runways prior to the takeoff operation. Figure 4.12 shows a sample of a departure flight.

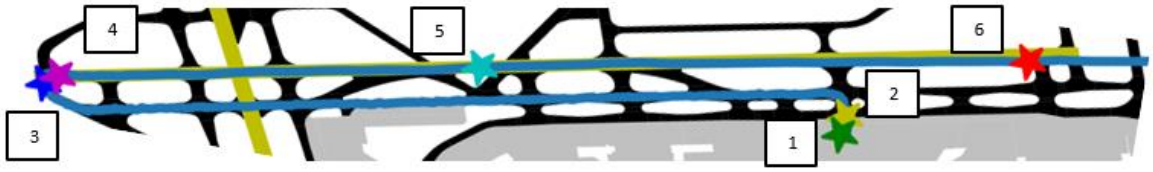


Figure 4.12 – Sample of a Departure Flight and Important Events

Operational Runway: The event parser searches for a track point known as the **Departure Index** in a departure flight which meets all of the following conditions:

1- Difference between the chosen point's altitude and the first point of the track's altitude should be equal or less than 150 ft.
2- The chosen point should be inside one of the runway polygons in the airport.
3- The acceleration of the chosen point should be a local maximum among the other points of the departure track on the same runway.
4- There must be 10 more points after the chosen point in the same runway polygon.
5- There must be 6 points prior to the chosen point in the same runway polygon.
6- The altitude difference between the chosen point and the 25th point ahead of that in the departure track should be equal or greater than 200 ft.

Exit Gate Point (Labeled 1): Estimated point where plane departs from gate. The program defines this point as the first point inside the apron that meets the following conditions:

- The flight has not moved for 5 consecutive seconds before the chosen point.
- The flight begins to move, and continues to move farther than 50 ft. after 5 seconds from the chosen point.

If no suitable point can be determined, then the program defaults to using the first (earliest) point in the track.

Enter Movement Area (Labeled 2): This is the point when the flight transitions from Apron area to Taxi-Way area and into FAA control. This boundary is provided by the FAA ASDE/RWSL Operations Engineering Subteam (AJW-1441). To avoid instances where the program detects a flight briefly re-entering the apron, this point must meet the following circumstances:

- The flight must have been in the apron (silver polygon) for 10 seconds or greater.
- The flight must have been outside the apron for 60 seconds or greater after leaving it.

If neither condition is met, the flight is not assigned an Enter Movement Area event and a Warning flag is assigned to the flight?

Enter Runway Point (Labeled 3): This is the point when the flight entered the operational runway. It is defined at the point a flight is detected as exiting an Apron/Taxiway polygon and entering a Runway polygon. Is the case of complex airports with multiple runways, the last event is taken as the Enter Runway thus marking the end of Taxiout.

Start of Takeoff Roll (Labeled 4): This is the point where the flight has begun accelerating to achieve takeoff velocity. The program looks for the first point where the current acceleration is 0.5 m/s^2 or greater. This threshold was determined by looking at the values of acceleration for several different flights and accounts for location uncertainty when the flight is not moving.

Wheels-Off Point (Labeled 5): First, the departure index is identified consistent with the description previously discussed (Departures Methodology section- Operational Runway subsection). After the departure index is found on the runway, 5 seconds are added to that moment to estimate the wheels-off location and time. The time of 5 seconds was chosen based on findings from Dr. Toni Trani's study based on landing/takeoff videos at ORD.

Exit Runway Point (Labeled 6): For this point, the program looks at all points past the Departure Index to ensure that it selects a Wheels-Off point just prior to taking off (within 25 seconds of the flight ascending 200 feet). The Exit Runway Point is defined as the first point in the set of points which is touching the boundaries of the runway polygon or entirely outside the runway. This point is not always at the other end of the runway since some flights diverge from the runway heading after taking off.

Enter Runway On/Off Duration: Time difference between points 3 and 5.

Enter Runway On/Off Distance: Distance between points 3 and 5.

Enter Runway Exit Runway Duration: Time difference between points 3 and 6.

Taxi In/Out Duration: Time difference between points 1 and 3.

Taxiing Distance: Cumulative distance that the airplane travels from the near gate point to the runway enter point. This distance is the travel distance from point 1 to point 3.

Average Taxi Speed: Smoothed taxiing speed from the near gate point to the enter runway point. This speed is the average of all the instant speeds from the point 1 to point 3.

Wait Time: All the moments that the airplane was taxiing with a speed lower than 3 m/s. This assumption was made to detect the places where airplanes were stuck in queues, when they had small changes in their instant location.

Start of Takeoff Roll Speed: Speed at which the flight is travelling at the beginning of takeoff roll.

Operational runway was complicated to parse. For departures, a maximum acceleration threshold was added to the procedure in order to prevent wrong runway assignment for situations where the flight crosses other runways to get to its departure runway, as shown in Figure 4.13.

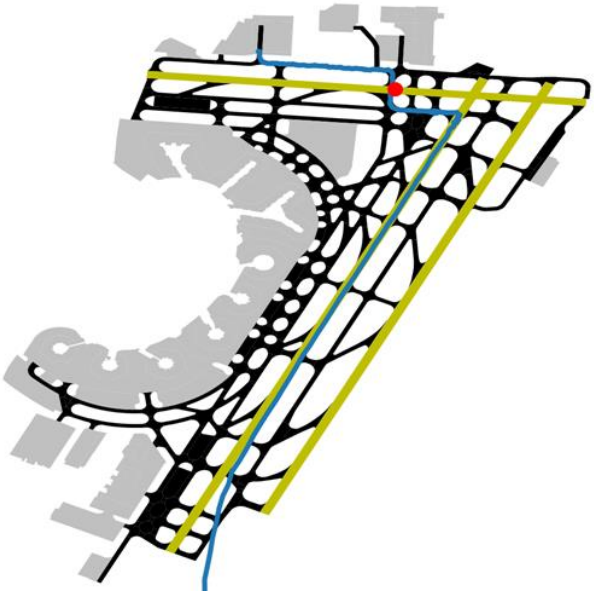


Figure 4.13 – Wrong Runway Assignment Example Based Only on Having Few Points on Runway

Unlike arrival flights that typically are lined up with the operation runway when entering the runway, departure flights usually turn around near the runway end after entering runway (See Figure 4.16 for example of turnaround) and the surface flight paths can be more complex. To extract the correct runway end name of the operational runway, the program calculates the bearing of the flight using the track point that meets all the criteria when defining operational runway (Departure Index) and the point 10 seconds afterwards and compares to the calculated bearing to the runway names. An example flight bearing calculation is shown in Figure 4.14 below, using these two indices, the program can calculate the flight bearing accurately and extract the runway end correctly.

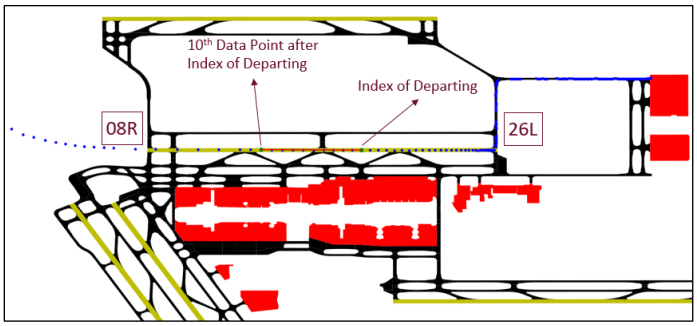


Figure 4.14 – Departure Flight Operational Runway End Extraction Example (Figure made during TEE V5)

To extract the enter runway point for more complex paths (shown in Figures 4.15 and 4.16), the event parser will make an additional check for continuous track points inside runway. The methodology is to find all indices of track points before takeoff that are inside operational runway and calculate the difference between each two indices. For reference, an index simply refers to the order in which the points in a flight are arranged. The first track point in a flight would have an index of 0, the second would have an index of 1, and so on. The difference in indices is the number of points that exist between any two given points. A gap (difference in indices greater than 1) indicates the airplane crosses the operational runway since there are track points where the flight is briefly outside of the runway. The average of the differences is calculated to define the enter runway point.

When the average of difference is greater than 1.2, representing a large gap in indices exists (Figure

4.15), the enter runway point is therefore defined as the first track point after the gap that's inside the operational runway (red dot in Figure 4.15).

Figure 4.15 – Example of Defining Enter Runway Point (Red Dot) for Flight Path Crosses Operational Runway

Another example flight track shown in Figure 4.16 below shows a flight path that has an average of differences in indices closer to value of 1. In this example, the airplane enters the operational runway and turns around near the runway threshold and then takes off. There are a couple of track points that are outside the operational runway (indicated in the red box), but the airplane occupies the runway continuously. Therefore, the enter runway point is defined as the first track point that is inside the operational runway (red dot).

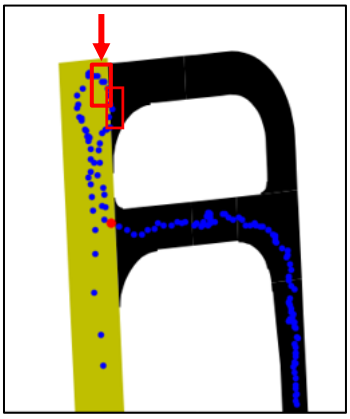


Figure 4.16 – Example of Defining Enter Runway Point (Red Dot) for Flight Path with Small Gap (Red Box) (Figure made during TEE V5)

4.3 Taxi Route Readout

The Taxi Event Extractor outputs the taxi route readout for each flight by listing the names of all taxiway elements used. This works by checking to see when the flight enters a new taxiway polygon and adding the name of the new taxiway to an ordered list.

It should be noted that taxi points that do not fall within any taxiway polygon are ignored in the algorithm when the taxi route readout is summarized. While these cases could be observed for a

variety of reasons, the typical reason is that flights use a new taxiway that is not included in the existing layout files. This case can be resolved by updating the layout AIXM file mentioned in section 3.1 or by using a different source such as updated shapefiles if available. In addition to these points, if the program sees that the flight enters a taxiway polygon that it has recently entered (last 5 polygons), then that polygon will be ignored. This is to simplify and prevent the Taxi Route Readout from generating an overly long entry.

Additionally, the program is set to work with the exact taxiway polygon boundaries without any tolerance to ensure the highest accuracy level in the algorithm output. If the user wishes to apply tolerance in the detection algorithm, they can uncomment the tolerance level section in ArrivalFlightsCheck.py and DepartureFlightsCheck.py functions. This tolerance feature, if uncommented, is activated when no taxiway is detected and applies an adjustable tolerance level to the detection process to help find the nearest taxiway. The default tolerance is set to 0.000075 decimal degrees which is approximately 20 feet, it is advised that advanced users only use this feature. In this example case, points not on a taxiway would report any taxiway polygon within 20 feet of the point.

4.4 Deicing Detection (Optional)

The Taxi Event Extractor has an optional feature to detect deicing activities in stationary locations. The deicing algorithm is contained in the Deicing function that is called by DepartureFlightsCheck.py function when applicable. These areas can be defined in the taxi event extractor by two ways that are coded into the Layout_Loader.py function:

- A csv file that contains coordinates of deicing areas. This is the default method when AIXM files are the airport layout source. The deicing areas data format in the csv file is organized in two columns and one row per deicing pad. The first column contains the airport FAA identifier, the second column contains a series of coordinates in the format $[[Lon_1,Lat_1],[Lon_2,Lat_2], \dots , [Lon_n,Lat_n]]$. Note that there is no limit to the number of deicing areas per airport or to the number of vertices per polygon. Figure 4.17 shows an example of a few departure flights stopping inside the user-defined deicing polygons before they continue taxiing to their departure runway at LaGuardia (LGA).

Figure 4.17 – Deicing Areas from user-defined coordinates at LaGuardia (LGA)

- Use any areas within the airport layout data that contains the keyword “DEICING” in their label. This is the default method when using shapefiles as the airport layout source. The program searches for deicing pads within the apron areas by default.

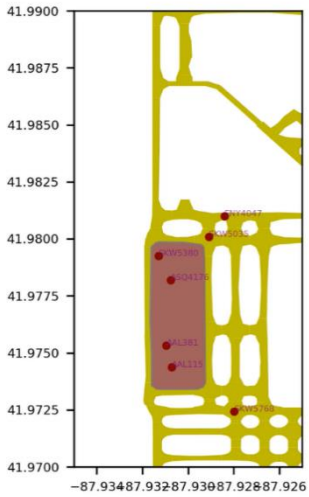


Figure 4.18 – Central Deicing Facility being used at Chicago O'Hare (ORD)

When the deicing detection feature is turned on in the source code, the Deicing function calculates the entry time, exit time, and time spent inside each deicing polygon. The program checks for multiple occurrences and returns the most recent usage of the deicing area for each flight. The deicing algorithm creates speed profiles for the time the aircraft is inside the deicing areas. These speed profiles are used to validate the deicing usage and are then analyzed to detect segments of traveling, waiting, and deicing based on patterns in the data. Figure 4.19 shows the resulting speed profile for a flight that experienced a period of time waiting before getting deiced. The figure contains circles at the end of each segment that denote the type of the segment as identified by the algorithm.

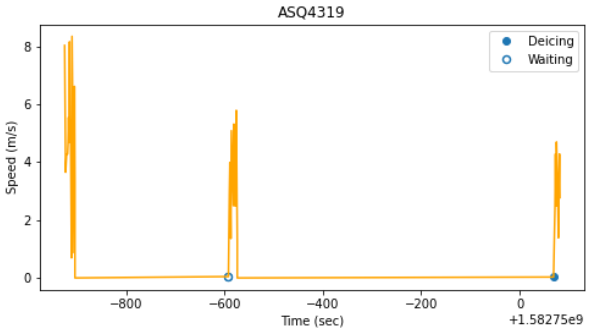


Figure 4.19 – Example Speed Profile

The algorithm saves each speed profile into a directory named "Deicing Speed Profiles" within the working directory. Additionally, the Deicing function returns the entry of deicing pad, exit of deicing pad, deicing time, travel time, and queue time within the deicing pad. The general deicing algorithm procedure is shown in Figure 4.20.

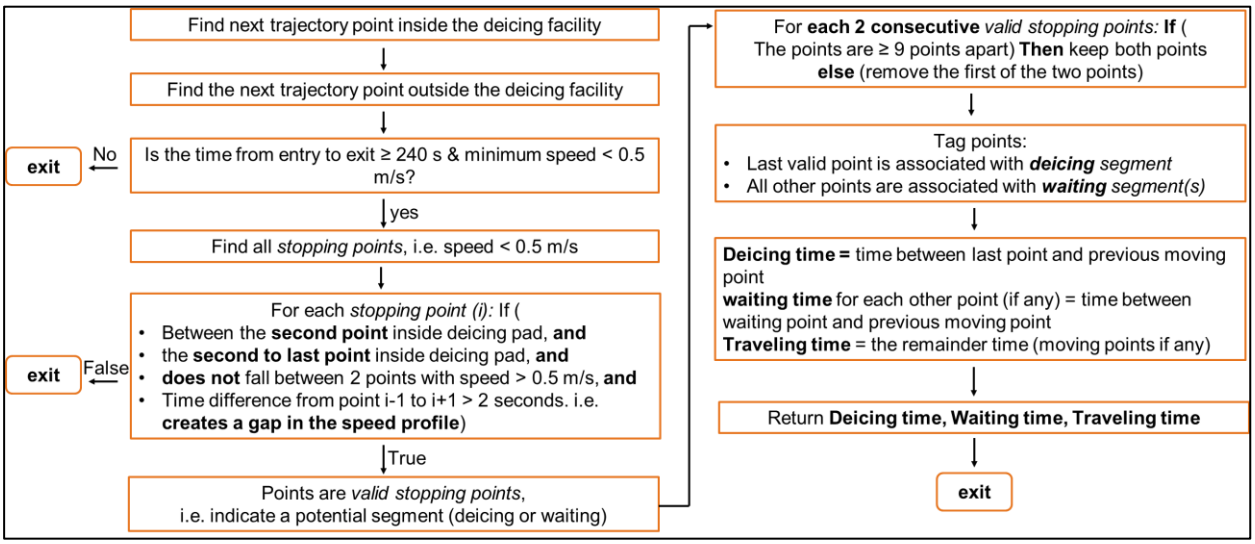


Figure 4.20 – Overview of Deicing Detection Algorithm

The criteria for deicing detection and segment identification were mainly based on observations in the data. We generated and analyzed speed profiles for 1,495 flights that used the central deicing facility at Chicago O'Hare (ORD) during the month of February 2020. We refined the criteria described in the flowchart above by analyzing individual profiles that did not conform to the typical expected cases. Additionally, we used video animations created by AnimateFlightTracks.py to validate our assumptions. Figure 4.21 Shows data for flights that used deicing areas at ORD, it was used to make assumptions of the characteristics of a reasonable deicing event whenever an aircraft enters the deicing facility. Speed requirement of less than 0.5 m/s and a minimum of 240 seconds were used to filter out false detections.

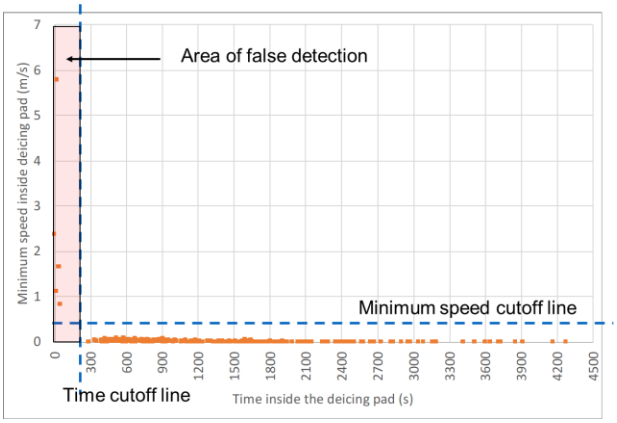


Figure 4.21 – Deicing Facility Usage for Data from ORD

Other cases that the algorithm can handle are irregularities in the speed profile due to noise in the ASDE-X position data. Figure 4.22 shows an example of spikes in the speed profile due to the change in coordinates, however, sometimes that change in position is caused by jitter in radar data and does not correspond to actual movement. Therefore, the algorithm requires at least nine consecutive points to consider the movement valid and assign a new segment. Otherwise, the algorithm ignores those spikes. Another example is having dips in the speed to zero for a very short period of time. The algorithm will ignore these cases when the dip is surrounded by two moving points. An example of the dip that is considered a false stopping point is shown in Figure 4.23

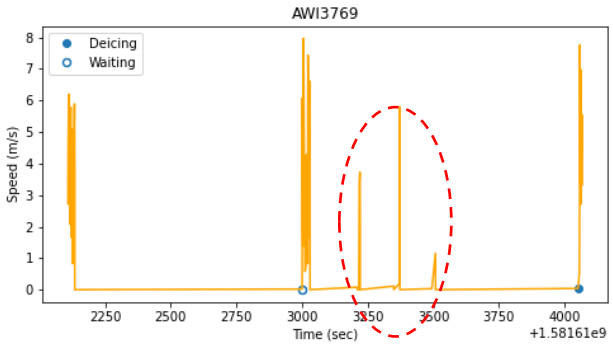


Figure 4.22 – Example of a Speed Profile with Spikes Caused by Noise in ASDE-X Data

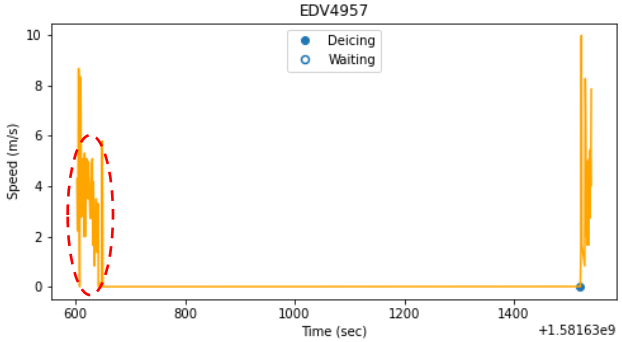


Figure 4.23 – Example of Speed Profile with False Stopping Point Caused by Noise in ASDE-X Data

Two additional output results are the start and end of deicing demand. The calculation of deicing demand is optional, it calculates the time aircraft enter a staging area that is used for waiting prior to entering the deicing facilities. The staging area needs to be defined by the user as a series of taxiway labels. Deicing demand ends when an aircraft exits the deicing facility.

4.5 Anomalies Variable

The Event Extractor provides the ability to determine and mark when certain flights exhibit specific errors or inconsistencies (i.e. Anomalies). . Using several parameters calculated during the DepartureFlightsCheck.py function, the AnomaliesCheck.py function checks the flight data to see if certain conditions for anomalies were found and adds the applicable conditions to a list that is included in the Excel results under the "Anomalies" header. If a flight has more than 1 attributed anomalies, the entry in the "Anomalies" field will include a comma-separated list of all anomalies for that flight. The following is a list of the different anomalies that the program checks for and their respective conditions:

Table 4.1 – Example Flights with Each Type of Anomaly

Flight_ID	Operation	Aircraft_Type	Airport	Anomalies	
DAL2125	D	B739	EWR	No Anomalies	1
MTN8311	D	C208	EWR	['False Gate Out']	2
GJS4425	D	CRJ7	EWR	['Start outside Apron/Taxiway', 'Incorrect Gate Out']	3
RPA3617	D	E75L	EWR	['Missing/Incorrect Airport']	4
MTN8310	D	C208	EWR	['No Wheels-Off/On']	5
GJS4411	D	CRJ7	EWR	['Gate Return', 'No Wheels-Off/On']	6

UAL1078	D	B739	EWR	['Disabled Aircraft', 'Gate Return', 'No Wheels-Off/On']	7
UAL1062	A	B739	ORD	[‘Missing Runway Info’]	8
				[‘Go Around’]	9

No Anomalies:

If no anomalies are detected by AnomaliesCheck.py, then the flight receives the status of “No Anomalies” in the results file.

False Gate In/Out:

A flight receives this anomaly if the following condition is met:

- The difference between the Gate Point time and either the last point in the track (arrivals) or the first point in the track (departures) is equal to 0.

In other words, if the code was unable to find a point that satisfied the requirements to be used as the Gate Point, then it receives the “False Gate In/Out” anomaly. A flight with this anomaly might not necessarily have the correct time that the flight left/arrived at the gate. Figure 4.24 below shows an example where the first ping in a flight occurred outside of the apron (lower left where blue trail starts).

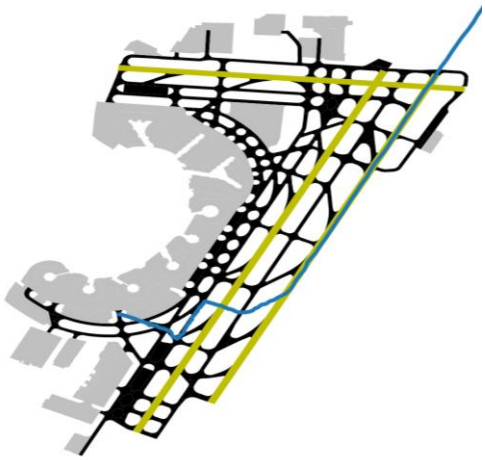


Figure 4.24: Example of Flight that Received “False Gate In/Out” Anomaly

Started/Ended Outside Taxiway/Apron:

A flight receives this anomaly if the following condition is met:

- The flight either starts (arrivals) or stops (departures) outside of any of the taxiway or apron polygons.

If this condition is met, then the flight receives the “Started/Ended Outside Taxiway/Apron” anomaly. This anomaly exists to indicate when a substantial number of flights either begin or end outside of the defined asphalt surfaces, meaning that the given airport layout may need to be updated. Figure 4.25 below shows an example where the flight ended outside of the defined apron area (green star).

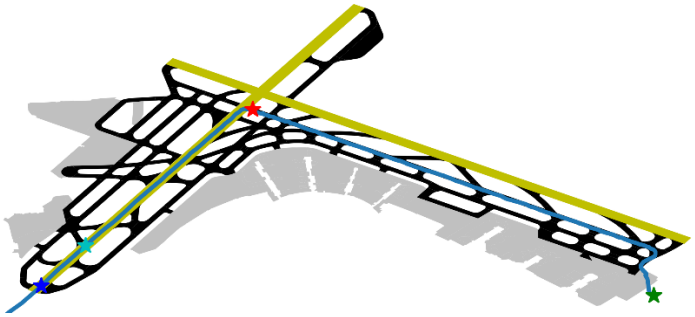


Figure 4.25: Example of Flight that Received "Started/Ended Outside Taxiway/Apron" Anomaly

Missing/Incorrect Airport:

One of two conditions must be met for this anomaly to apply:

- Neither the departure nor arrival airport fields contains the three-letter code for the chosen airport
- The flight's on-ground coordinates are further than 1 whole degree of latitude from the chosen airport's runway coordinates

If either of these conditions are met, then the "Missing/Incorrect Airport" anomaly is added to that flight's anomaly list. These flights typically appear in flight logs because the specified airport is either the arrival or departure airport and the query simply picks the wrong section of flight data (from the non-specified airport). Figure 4.26 below shows an example of a flight where the Gate coordinates are substantially further than the other flights in the log. The coordinates in the red box are the flight's coordinates at the gate and are significantly far away from other flights' gate points and the coordinates of the specified airport ((35.10, -89.69) compared to (40.69, -74.17)).

Flight_ID	Operation	Near_Gate_Lat	Near_Gate_Long	Near_Gate_Time	Anomalies
JBU890	A	40.68453	-74.18131	2/8/2022 14:59	['Near_Gate_Point is Last Point']
UAL496	D	40.69726	-74.17337	2/8/2022 14:54	No Anomalies
EDV5464	D	40.68837	-74.17464	2/8/2022 14:53	['Near_Gate_Point is First Point']
UAL700	A	40.69976	-74.18152	2/8/2022 15:00	['Near_Gate_Point is Last Point']
RPA3617	D	35.09882	-89.69212	2/8/2022 13:59	['Missing/Incorrect Airport']

Figure 4.26: Example of Flight that Received "Missing/Incorrect Airport" Anomaly

No Wheels Off/On:

One of two conditions must be met for this anomaly to apply:

- The flight must not reach the necessary elevation difference of 200 ft. when comparing the first and last points in the track
- The flight's aircraft type must change at some point during the flight track

If either of these conditions are met, then the "No Wheels Off/On" anomaly is added to that flight's anomaly list. Note that these entries do not have any geographical or time data entered in the Source

Code Results file due to never reaching the runway.

Gate Return:

The following conditions must be met for some point in the flight track for this anomaly to apply:

- The flight must have been outside the apron for 7.5 minutes before the current ping
- The flight must be in the apron for 45 seconds after the current ping

For the last 60 pings in a given flight track, the time parameters are slightly different:

- The flight must have been outside the apron for 7.5 minutes before the current ping
- The flight must be in the apron for 7 seconds after the current ping

The reason for this distinction is that flight tracks typically stop very soon after the flight enters the apron. In order to catch these flights, the time gate must be lowered for the final 60 points in the track in the case the flight does return to the gate. If these conditions are met, then the "Gate Return" anomaly is added to that flight's anomaly list. Figure 4.27 below shows an example of a flight that departed from the gate, returned to the gate, and then successfully departed.

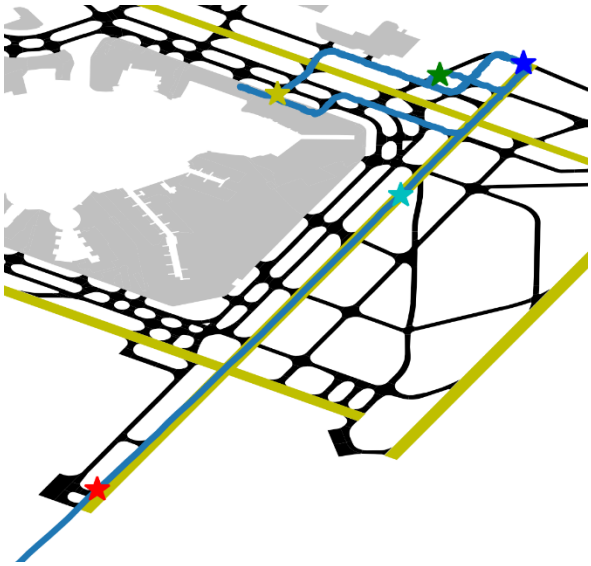


Figure 4.27: Example of Flight that Received "Gate Return" Anomaly

Disabled Aircraft:

All of the following conditions must be met for this anomaly to apply:

- The flight must have already received the "No Wheels Off/On" anomaly
- The flight must have already received the "Gate Return" anomaly
- The flight must not move more than 10 ft. over a span of 10 minutes while on the runway or taxiway

If all of these conditions are met, then the "Disabled Aircraft" anomaly is added to that flight's anomaly list.

Missing Runway Info:

2 of the 3 following conditions must be met for this anomaly to apply:

- A required elevation difference between the first and last points of 200 feet or more
- A Runway Occupancy Time (ROT) of less than 10 seconds.
- The flight spends less than 3 seconds in any runway.

If the flight meets these conditions, then the "Go Around" anomaly is added to that flight's anomaly list. An example of a flight that meets these criteria can be found in Figure 4.28.

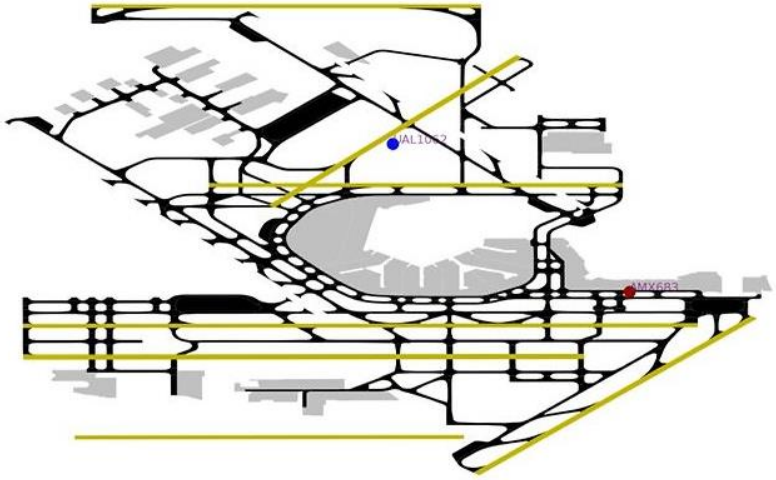


Figure 4.28: Example of Flight that Received "Missing Runway Info" Anomaly

Go Around:

In the case that a flight needs to perform a go-around, the Source Code is able to recognize this. This particular anomaly is triggered under the following circumstances:

- The flight must have been inside a given runway for at least 10 seconds.
- The flight's average speed must be 30 m/s (58.3 knots) or greater.

If all of these conditions are met, then the "Go Around" anomaly is added to that flight's anomaly list.

5 FLIGHT TRAJECTORY PLOTS:

5.1 SourceCode_ValidationPlot.py

After running the taxi event extractor, one can also execute the ValidationPlot.py code for visualization and validation of specific flight trajectories. The five required input arguments are:

```
# Set user defined input arguments:

AirportName = 'CLT' # Airport Name (FAA code)

Dates = [20220222]  # Date - format YYYYMMDD

Flight_Data_Path = os.path.join(workingdir, 'data_surface_tracks', AirportName + '+ASDEX') # Input flight data directory

Result_File_Path = os.path.join(workingdir, 'output', 'ParsedEvents') # Source Code Results directory

Flight_Keys = [6876, 5799, 7525] # From Source Code Results, input "Flight_Key" field for specified flights here
```

Figure 5.1 - Screenshot of Input Arguments Example for SourceCode_ValidatioPlot.py

- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Dates:** The date(s) in YYYYMMDD format (ex: 20220222)
- **Flight_Data_Path:** ASDE-X IFF input file(s) directory (rootfolder\\data_surface_tracks\\CLT+ASDEX)
- **Result_File_Path:** Directory containing the Source_Code output for the input date(s) (ex: rootfolder\\output\\ParsedEvents)
- **Flight_Keys:** flight key(s) to be plotted (ex: [6457])

The resulting plot in Figure 5.2 shows the airport layout and specified flight tracks with critical events including enter runway point, touchdown/wheels off point, and exit runway point.

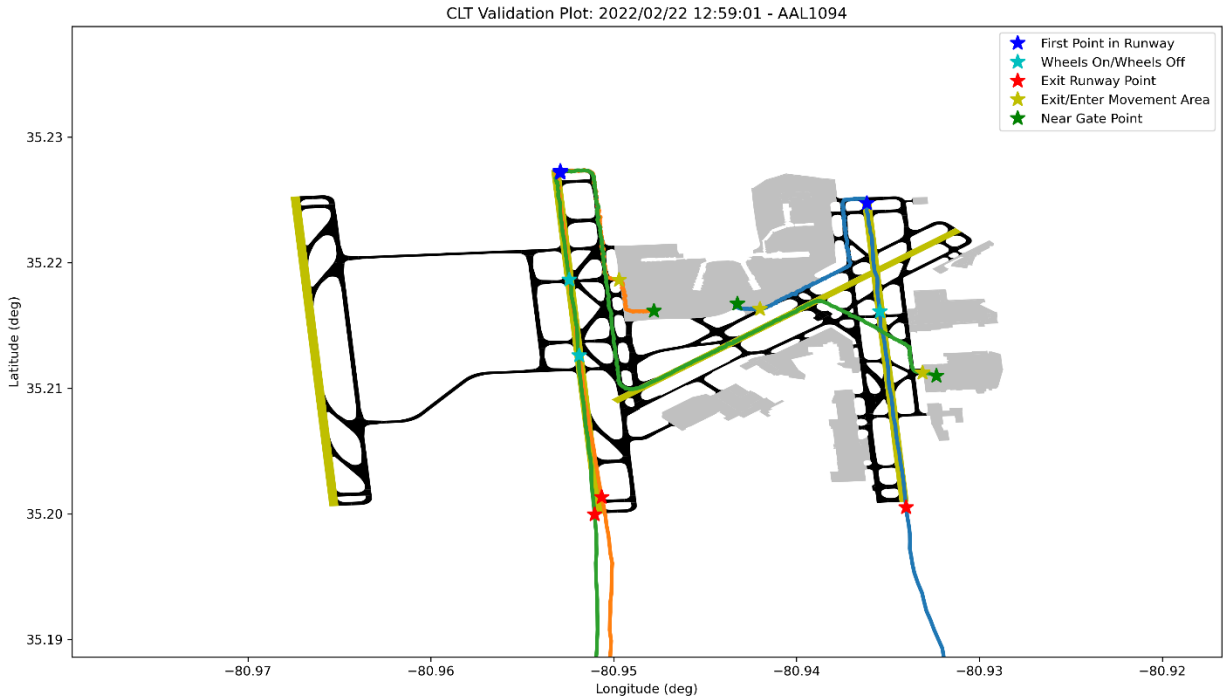


Figure 5.2 - Example Validation Plot

5.2 GraphFlightTracks.py

This script can be used to plot one or more flight tracks over the airport layout. The difference between this function and SourceCode_ValidationPlot.py is that it only plots the tracks and does not require the results file. The default behavior of the script is to focus on the airport area rather than show the entire track that extends up to five miles out. This was done to make sure the taxiways and apron areas are large enough in the plot area since they are of greater interest to the user than the rest of the trajectory.

GraphFlightTracks.py requires the user to input the following arguments:

```
# Set user defined input arguments:

AirportName = 'JFK' # Airport Name (FAA code)

Dates = [20220222]  # Date - format YYYYMMDD

Flight_Data_Path = os.path.join(workingdir, 'data_surface_tracks', AirportName + '+ASDEX') # Input flight data directory

Flight_Keys = [501022] # From Source Code Results, input "Flight_Key" field for specified flights here
```

Figure 5.3 - Screenshot of Input Arguments Example for GraphFlightTracks.py

- **AirportName:** FAA three character identifier (ex: 'ATL')
- **Dates:** The date(s) in YYYYMMDD format (ex: [20150704])
- **Flight_Data_Path:** ASDE-X IFF input file(s) directory (rootfolder\\data_surface_tracks\\ATL+ASDEX)
- **Flight_Keys:** flight key(s) to be plotted (ex: [40337, 40345])

Figure 5.4 shows an example of the output figure for an arrival flight into JFK.

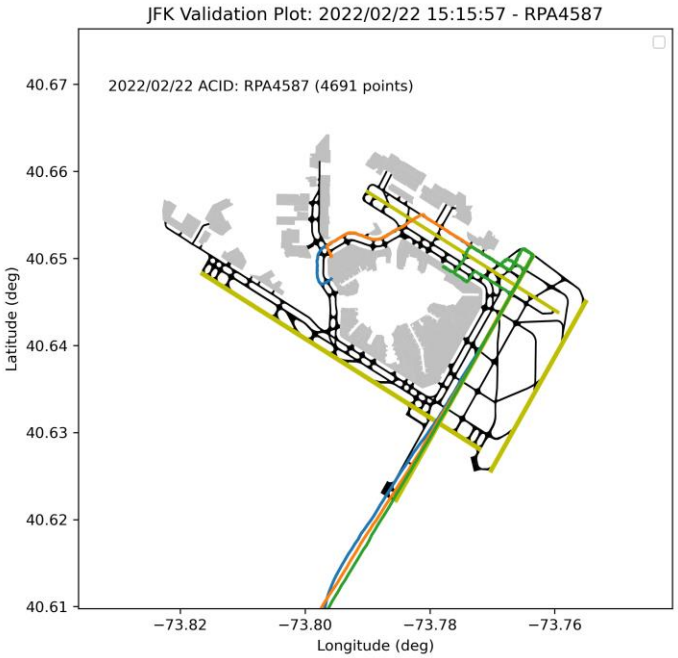


Figure 5.4 Example Flight Trajectory Plot

6 VIDEO ANIMATIONS: AnimateFlightTracks.py

After running the Taxi Extractor Source Code, the parsed data along with the raw track data can be used to produce a video animation of the airport for a user-defined time period. Animations can be created by running the Animation Batch.py code. The following input parameters are required by the user:

```
## USER INPUT ##
AirportName = 'EWR' # Airport Name (FAA code)
Date = [20220222] # Date - format YYYYMMDD
Start_Time = '13:00:00'
End_Time = '14:00:00'
Flight_Data_Path = os.path.join(workingdir, 'data_surface_tracks', AirportName+ '+ASDEX') # Input flight data directory
FileType = 'mat' # Layout File type ['mat', 'shp']
Unrestricted = True # If only flights that successfully takeoff/land are desired, put False. Default is True
Check_Deicing = False # Select True/False to turn on/off
Speed_Factor = 10 # Ratio of seconds of flight data per second of animation
```

Figure 6.1 - Input parameters for the video animation function

- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Date:** The date in YYYYMMDD format (ex: 20201201)
- **Start_Time:** UTC start time of animation in HH24:MM:SS (ex: ‘08:35:00’)
- **End_Time:** UTC end time of animation in HH24:MM:SS (ex: ‘12:08:00’)
- **Flight_Data_Path:** ASDE-X IFF input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')
- **FileType:** Airport layout file type ('mat' or 'shp')
- **Check_Deicing:** display deicing pad area shaded in red (True or False)
- **Speed_Factor:** video frame rate in frames per second (fps), 1 fps = real time (ex: 10)

The output of this code is a video in MP4 format with a resolution of 1800 x 1200 pixels. Videos are saved in a subfolder in the Outputs folder called "Animation". If the Animation folder does not currently exist, the code will create it. There are no special video codecs required for this code to run. Operations are color coded, departures are shown in red, arrivals in blue, and deicing areas are shaded in semi-transparent red. See the figure for a screenshot of video output. Animations can also be run as a batch using the “Animation Batch.py” file.

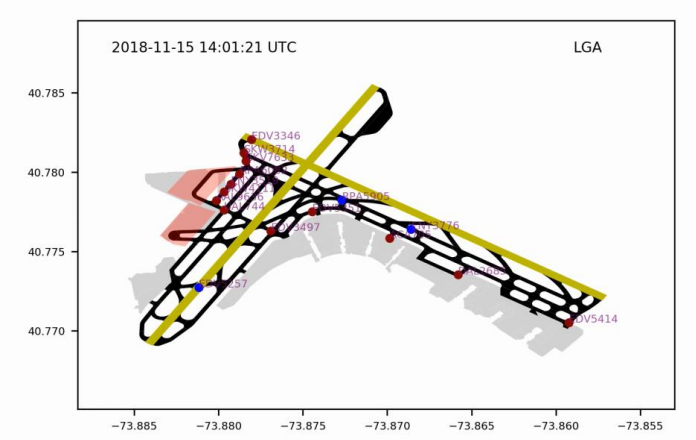


Figure 6.2 - Screenshot of a sample video

7 HEATMAPS

Another possible capability for the taxi event extractor is creating heatmaps that show four different characteristics: average speed, average weight, total weight, and frequency.
The heat maps can be created for a user-defined time period to show the characteristics of an airport's taxiways for that input time. The Heatmaps.py file can be used to create the four different heatmaps.

Before running the Heatmaps.py file, the geopandas module must be downloaded using the following instructions.

4. Open Anaconda's command line prompt
5. Update the current Anaconda installation by typing the command `conda update --all -c conda-forge`, press enter when it says procced
6. Install the geopandas module using the command `conda install geopandas -c conda-forge`, press enter when it says procced
7. Check the installation using Spyder by typing "import geopandas" in the console window

For more information on installing Geopandas use the following URL:

https://wilcoxen.maxwell.insightworks.com/pages/6373.html

In order to successfully run the main Heatmaps.py file the user needs to have 7 other .py files that contain the necessary functions, 3 CSV files with important aircraft and airport information, and 1 mat or shapefiles folder for that airport layout. There are also 5 functions that are reused from the main TEE extractor.

Once these files have been downloaded, the main Heatmaps.py file can be run with the following user inputs.

```
# Set user defined input arguments:

AirportName = 'JFK' #e.g. 'ORD'

Flight_Data_Path = os.path.join(workingdir, 'data_surface_tracks', AirportName + '+ASDEX') #

Start_Date = [20220222] #e.g. 20150701

End_Date = [20220222]  #e.g. 20150701

Start_Time = '12:00' #e.g. '17:25'

End_Time = '15:00' #e.g. '18:00'

FileType = 'mat' #e.g. 'mat'

DecD = 0.9 #e.g. 0.9 , the MTOW factor for departures, for 0.9 the MTOW will be multiplied b

DecA = 0.7 #e.g. 0.5 , the MTOW factor for arrivals,for 0.5 the MTOW will be multiplied by 0

Colorbar = 'vmax' #can be set to a integer of your choosing or 'vmax' (the highest average s

OutputFolder = 'default' #specify different output folder for Heatmaps and CSV file if neede
```

Figure 7.1 - Screenshot of Input Arguments Example for Heatmaps.py

- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')
- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Start_Date:** The date in YYYYMMDD format (ex: 20201201)
- **End_Date:** The date in YYYYMMDD format (ex: 20201201)
- **Start_Time:** UTC start time in HH24:MM (ex: '09:00)
- **End_Time:** UTC end time in HH24:MM (ex: '16:00')
- **FileType:** Airport layout file type ('mat' or 'shp')
- **DecD:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.9)
- **DecA:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.7)
- **Colorbar:** Maximum value of the figure color bar (Default='vmax' for maximum value found from data. Otherwise it accepts an integer value provided by the user)
- **Output_Folder:** The file path where the output should be stored

For each run, 12 plots will be created as well as one CSV file that shows the mapped number for each taxiway polygon. Each of the four characteristics can be seen for both arrivals and departures, just arrivals and just departures, which is why 12 plots are created.

7.1 Average Speed Heat Map

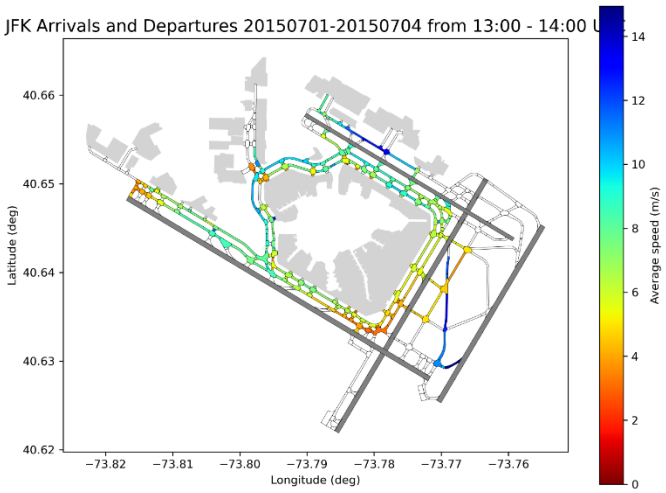


Figure 7.2 Average Speed Heat Map, arrivals and departures

The Average Speed Heatmaps shows the average speeds for each taxiway polygon. The speed is calculated using the same methodology as the Source Code where speed is the speed from one ping to the next. For each ping within the input time period, the speeds for every flight are sorted into a different list depending on which taxiway polygon the flight was in at that ping. The result is a list of speeds for every taxiway polygon. This list is then averaged to find the average speed in each taxiway polygon. Then the average for each taxiway polygon is mapped to the airport layout using the chosen color bar to create a heatmap effect. The other two plots created in this function are a heatmap of just arrivals and a heatmap of just departures.

7.2 Average Weight Heat Map

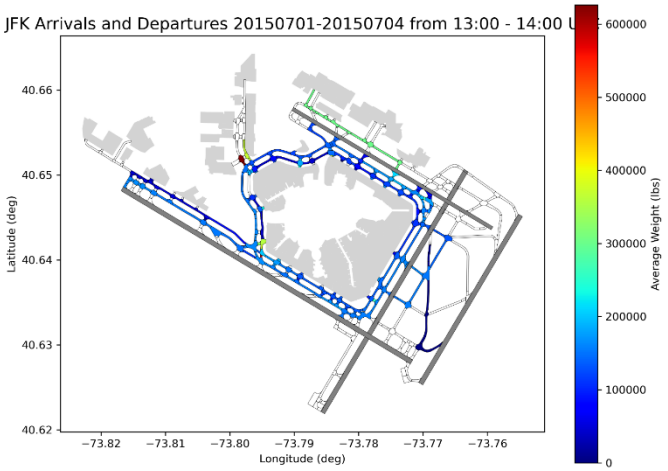


Figure 7.3 – Average Weight Heat Map, arrivals and departures

The Average Weight Heatmaps shows the average weight in each taxiway polygon. Each ping from every flight is analyzed and every time the ping enters a new taxiway polygon the weight associated with that ping is added to a list of weights for that particular taxiway polygon. Weights are pulled from an aircraft data Excel file located in the data_other folder. Only the first ping in each new taxiway polygon is recorded so that weights are not double counted. The average weight is found using the list of weight in each polygon. Then the average for each taxiway polygon is mapped to the airport layout using the color bar to create a heatmap effect. The other two plots created in this function are a heatmap of just arrivals and just departures.

7.3 Total Weight Heat Map

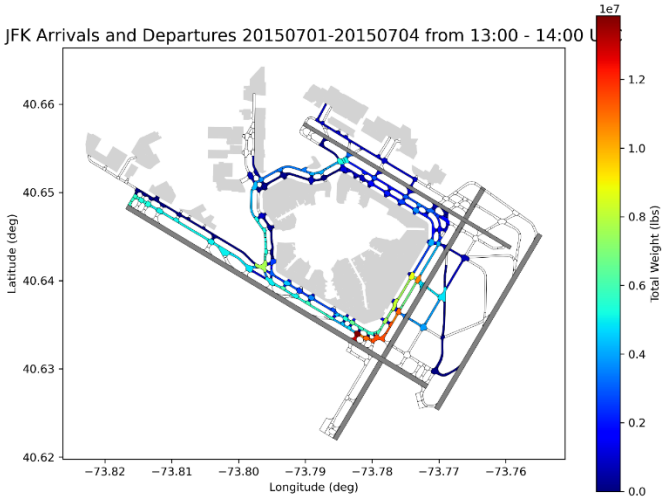


Figure 7.4 – Total Weight Heat Map, arrivals and departures

The Total Weight Heatmaps shows the total weight in each taxiway polygon. Each ping from every flight is analyzed and every time the ping enters a new taxiway polygon the weight associated with that ping is added to a list of weights for that particular taxiway polygon. Weights are pulled from an aircraft data Excel file located in the data_other folder. Only the first ping in each new taxiway polygon is recorded so that the weights are not double counted. Then the total weight is found using the list of weight in each polygon. Then the total for each taxiway polygon is mapped to the airport

layout using the color bar to create a heatmap effect. The other two plots created in this function are a heatmap of just arrivals and a heatmap of just departures.

7.4 Total Frequency Heat Map

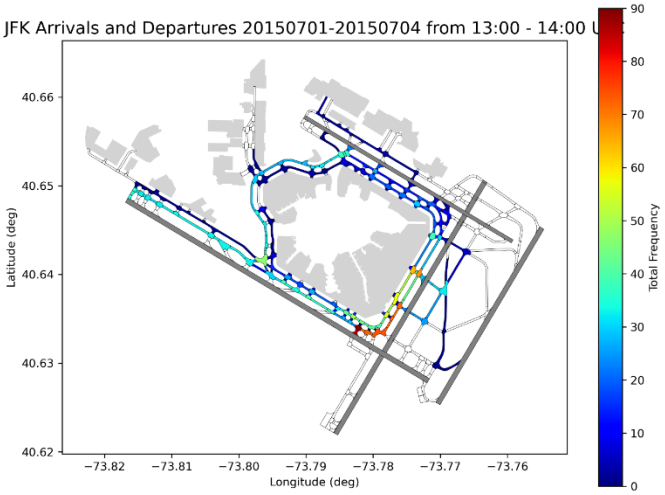


Figure 7.5 – Total Frequency Heat Map, arrivals and departures

The Total Frequency Heatmaps shows the total number of times each taxiway polygon was used. Each ping from every flight is analyzed and every time the ping enters a new taxiway polygon the count for that particular taxiway polygon is increased. Only the first ping in each new taxiway polygon is recorded so that the taxiway polygon frequency is not double counted. Then the total frequency is found once all the flights have been analyzed. Then the total for each taxiway polygon is mapped to the airport layout using the color bar to create a heatmap effect. The other two plots created in this function are a heatmap of just arrivals and a heatmap of just departures.

8 MULTIMAPS

Multimaps is an extension of Heatmaps.py that was developed during V6. Similar to Heatmaps.py, Multimaps.py generates heatmaps that show four different characteristics: average speed, average weight, total weight, and frequency for a user-specified time period. Multimaps is different in that it generates a series of heatmaps that cover a user-specified time increment (1, 5, 10 minutes) over the specified time period before creating an animation. The end results are animations that show how each taxiway's characteristics change during the time period specified (For example, the user could set the time period to 12:00-15:00 and the time increment to 5 minutes to see how a characteristic changes every 5 minutes).

Multimaps.py has the same system and package requirements as Heatmaps.py but uses a modified set of functions and inputs. The user-defined inputs for Multimaps are as follows:

```
# Set user defined input arguments:

AirportName = 'ATL' #e.g. 'ORD'

Flight_Data_Path = os.path.join(workingdir, 'data_surface_tracks', AirportName + '+ASDEX') #'C:\\Data\\LGA+ASDEX\\201811'  #C:\\Data\\CLT+A

Start_Date = [20150704] #e.g. 20150701

End_Date = [20150704]  #e.g. 20150701

Start_Time = '20:00' #e.g. '17:25'

End_Time = '23:00' #e.g. '18:00'

Incrementor = '30'

AverageSpeedON = True # Set to True for Average Speed animation/heatmaps

AverageWeightON = True # Set to True for Average Weight animation/heatmaps

TotalWeightON = True # Set to True for Total Weight animation/heatmaps

FrequencyON = True # Set to True for Frequency animation/heatmaps

Both = True # Set to True for All Flights animation/heatmaps

ArrivalsOnly = False # Set to True for Arrivals Only animation/heatmaps

DeparturesOnly = False # Set to True for Departures Only animation/heatmaps

FileType = 'mat' #e.g. 'mat'

DecD = 0.9 #e.g. 0.9 , the MTOW factor for departures, for 0.9 the MTOW will be multiplied by 0.9 and then 90% of the MTOW will be used

DecA = 0.7 #e.g. 0.5 , the MTOW factor for arrivals,for 0.5 the MTOW will be multiplied by 0.9 and then 50% of the MTOW will be used

OutputFolder = 'default' #specify different output folder for Heatmaps and CSV file if needed, otherwise default is Output/Heatmaps

Colorbar1 = 17.0000 # Maximum Average Speed
Colorbar2 = 900000.0000 # Maximum Average Weight
Colorbar3 = 2000000.0000 # Maximum Total Weight
Colorbar4 = 8.0000 # Maximum Frequency

OutputFolder = 'default' #specify different output folder for Heatmaps and CSV file if needed, otherwise default is Output/Heatmaps
```

Figure 8.1 - Screenshot of Input Arguments Example for Multimaps.py

- **Flight_Data_Path:** ASDE-X IFF or SWIM Feed input file(s) directory (rootfolder+'\\data_surface_tracks'+'\\LGA+ASDEX')
- **AirportName:** FAA three character identifier (ex: 'ORD')
- **Start_Date:** The date in YYYYMMDD format (ex: 20201201)
- **End_Date:** The date in YYYYMMDD format (ex: 20201201)

- **Start_Time:** UTC start time in HH24:MM (ex: ’09:00)
- **End_Time:** UTC end time in HH24:MM (ex: ‘16:00’)
- **Incrementor:** specifies the period of time covered by each individual heatmap in minutes (the higher the data, the more generalized the heatmaps become)
- **AverageSpeedON:** specifies if user wants to analyze for average speed (True or False)
- **AverageWeightON:** specifies if user wants to analyze for average weight (True or False)
- **TotalWeightON:** specifies if user wants to analyze for total weight (True or False)
- **FrequencyON:** specifies if user wants to analyze for frequency (True or False)
- **ArrivalsOnly:** specifies if user wants to analyze only arrival data (True or False)
- **DeparturesOnly:** specifies if user wants to analyze only departure data (True or False)
- **Both:** specifies if user wants to analyze only arrival data (True or False)
- **FileType:** Airport layout file type ('mat' or 'shp')
- **DecD:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.9)
- **DecA:** Maximum takeoff weight (MTOW) reduction factor for departures (ex: 0.7)
- **Colorbar1-Colorbar4:** Maximum value of each figure color bar (Default=‘vmax’ for maximum value found from data. Otherwise it accepts an integer value provided by the user)
- **Output_Folder:** The file path where the output should be stored

The number of heatmap images and animations generated depends on the user-defined settings and range from 1 image and 1 animation to multiple hundred images and 12 animations. It is advised to start with longer Incrementor values when searching for specific taxiway behavior and then narrowing down the time frame and desired animations. This will significantly reduce the time needed to find the desired Multimaps animation.

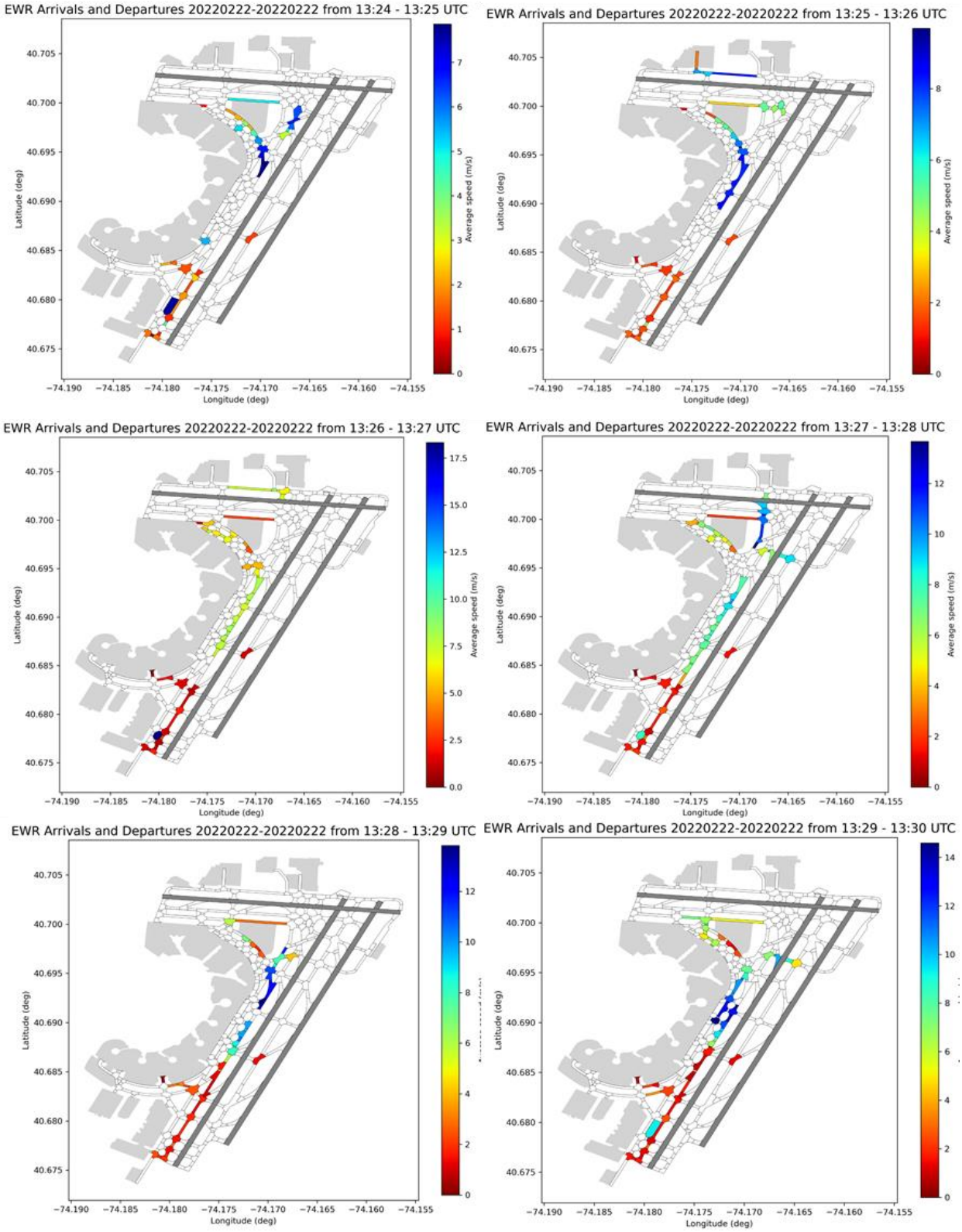
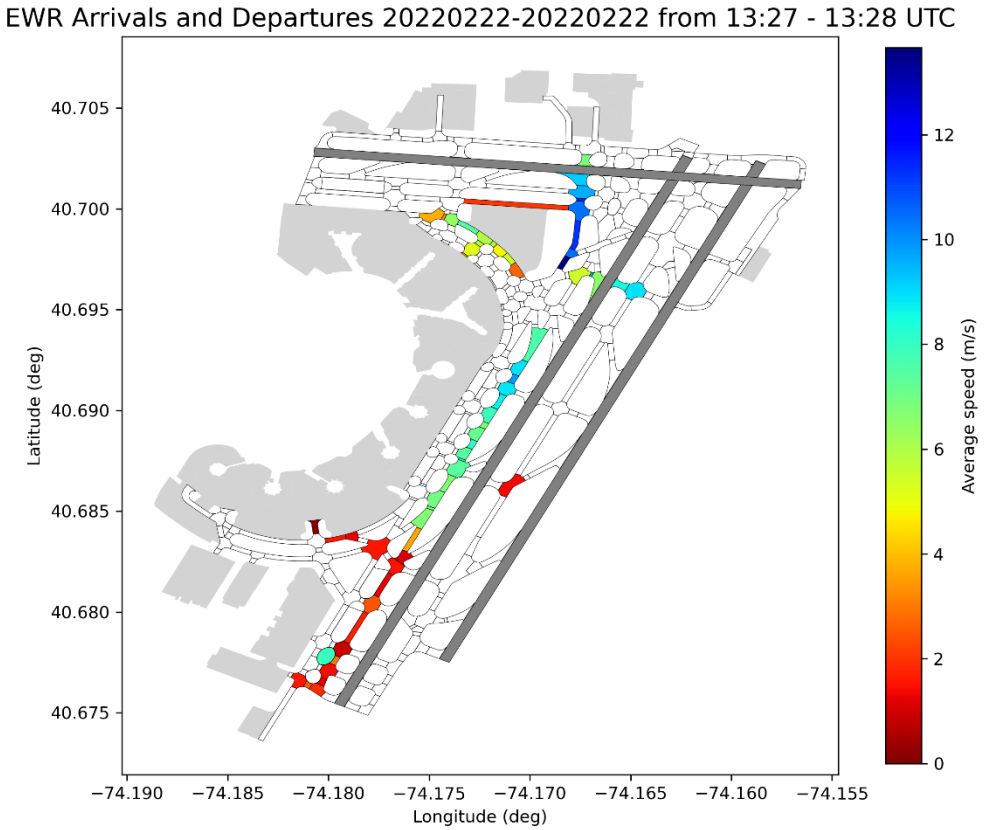
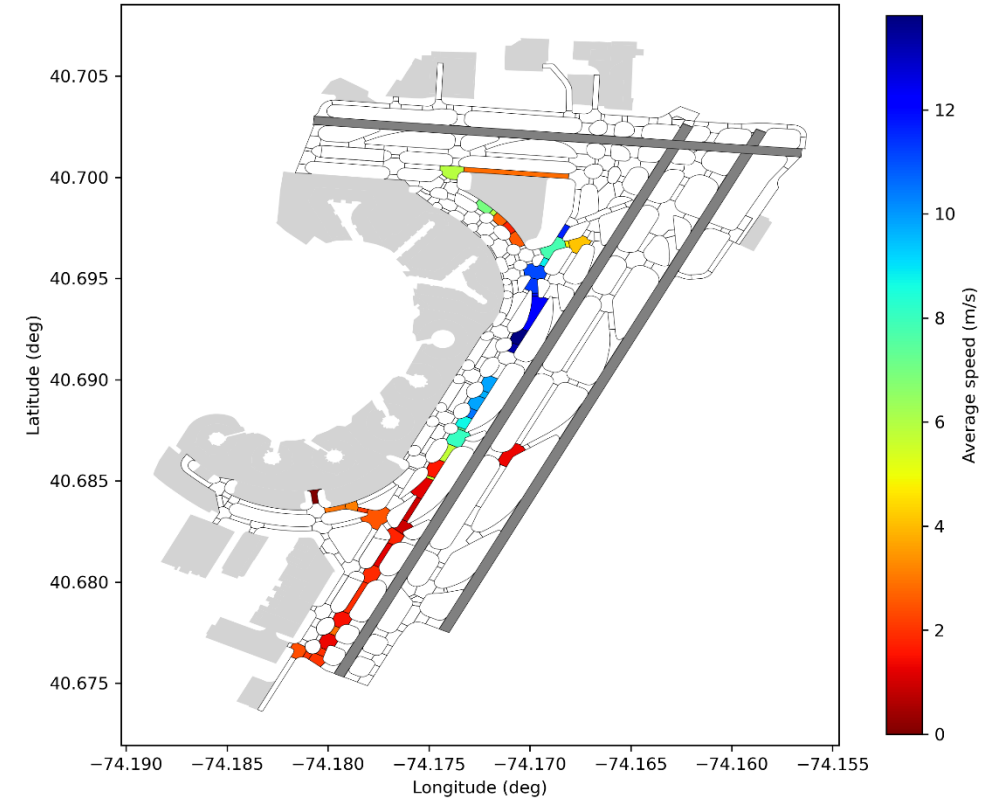


Figure 8.2 – Isolated Heatmap Frames from Average Speed Animation for EWR from 12:00-15:00 with 1 min Intervals


EWR Arrivals and Departures 20220222-20220222 from 13:27 - 13:28 UTC
Latitude (deg)
40.705
40.700
40.695
40.690
40.685
40.680
40.675
−74.190
−74.185
−74.180
−74.175
−74.170
−74.165
−74.160
−74.155
Longitude (deg)
Average speed (m/s)
0
2
4
6
8
10
12


EWR Arrivals and Departures 20220222-20220222 from 13:28 - 13:29 UTC
Latitude (deg)
40.705
40.700
40.695
40.690
40.685
40.680
40.675
−74.190
−74.185
−74.180
−74.175
−74.170
−74.165
−74.160
−74.155
Longitude (deg)
Average speed (m/s)
0
2
4
6
8
10
12

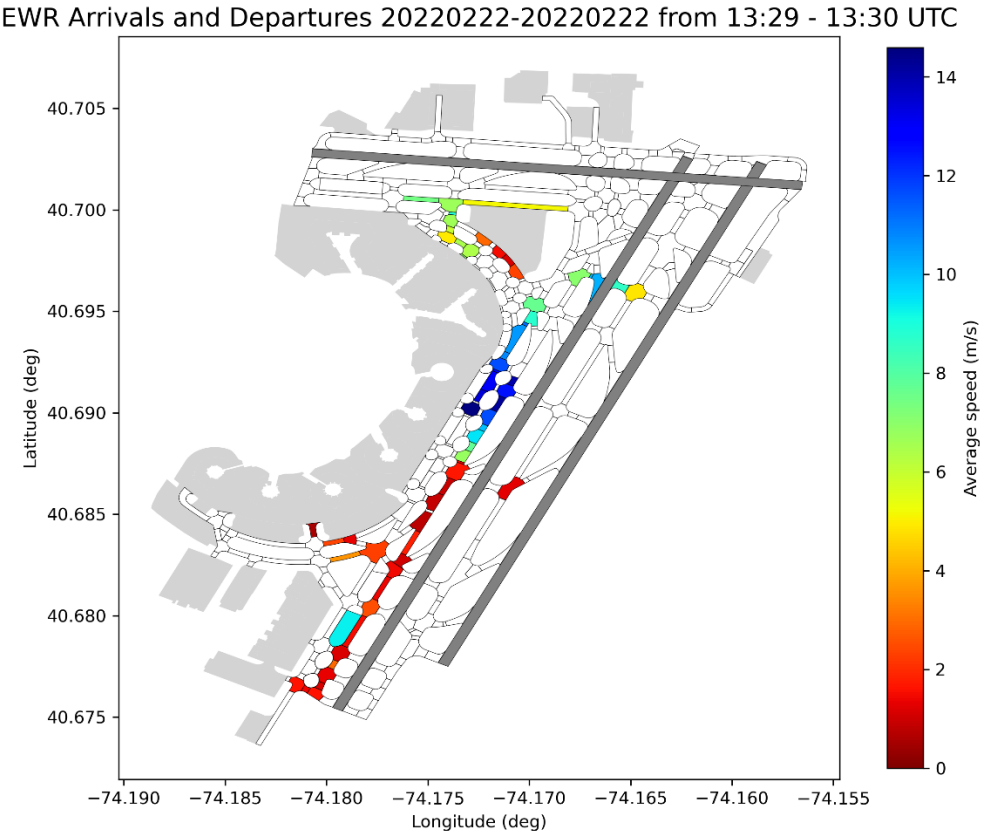


Figure 8.2 – Isolated Heatmap Frames from Average Speed Animation for EWR from 12:00-15:00 with 1 min Intervals

9 ASPM VALIDATION

This section describes the program, ValidationASPM.py, which streamlines the process of comparing the output of the surface event extractor with flight data from the FAA ASPM flight tables. ASPM flights counts are used in FAA performance reporting tools, contain key event times for Gate-Out, Wheels-Off, Wheels-on, and Gate-In. These times can then be compared to similar times calculated from processing the ASDE-X surface times. In this application, ASPM flight data is downloaded from the online ASPM Individual Flights module. The data downloaded includes all of the field sections (Flight Information, Departure Data, Arrival Data, and ASQP & OPSNET Delay Information) for all flights at a given airport within a given time period. Once the data is ordered, it is downloaded and then stored in an Excel .csv file in the "data_surface_tracks\\ASPM" folder using the default exported format.
t

The program takes the following user-defined inputs:

```
AirportName = 'JFK' # Indicates the irport of interest
ASPM_Folder = os.path.join(workingdir, "data_surface_tracks", "ASPM") # Folder that contains ASPM data
TEE_Folder = os.path.join(workingdir, 'output', 'ParsedEvents')
Results_Folder = os.path.join(workingdir, 'output', 'Validation') # file directory for the Validation results files
TZModifier = 0 # If ASPM data is not in GMT, this is the time difference behind GMT in hours
Dates = ['20220222'] # List of dates to be analyzed
Start_Time = '12:00'
End_Time = '15:00'
```

Figure 9.1 - Screenshot of Input Arguments Example for ValidationASPM.py

- **AirportName:** FAA three character identifier (ex: 'ORD')
- **ASPM_Folder:** Folder containing ASPM Excel data files with AirportName in file name
- **TEE_Folder:** Folder containing Extractor results file with AirportName in file name
- **Results_Folder:** specifies the output folder for the validation results files
- **TZModifier:** specifies the time zone difference between ASPM data and GMT in hours
- **Dates:** The date(s) in YYYYMMDD format (ex: [20201201])
- **Start_Time:** UTC start time in HH24:MM (ex: '09:00)
- **End_Time:** UTC end time in HH24:MM (ex: '16:00')

The program begins by aggregating all flight data that contains the Airport Name and at least one of the dates in Dates. Afterwards, there is an initial screening that isolates the ASPM flights that fall within the time period specified by the user with a 1-hour buffer in both directions. Once these initial screens are done, the program utilizes a function called FilterASPM.py to match flights from the Extractor and ASPM datasets. FilterASPM.py is run multiple times using different iterations with preset parameters that need to be met. These parameters include takeoff/landing times that are suitably close to one another, checks to see if the airline carrier and number match, and checks to see of the operational runways match. The parameters in each iteration are outlined in Table 9.1 below.

Table 9.1 – Matching Parameters for Each Filter Iteration in ValidationASPM

Iteration	Allowable Time Difference (seconds)	Carrier Check	Runway Check
0	0	0	0
1	120	Strict	TRUE
2	300	Strict	TRUE
3	600	Strict	TRUE
4	120	Strict	FALSE
5	120	Carrier	TRUE
6	60	None	TRUE

Once the code has run through all the set filter iterations, it will perform a time analysis on the flights that were matched. This includes comparing the operational runways, Gate Point times, the Takeoff/Landing times, and the Taxi times (difference between Gate Point and Takeoff/Landing times). Afterwards, the code generates the following 4 Excel files in the specified outputs folder:

9.1 Validation Complete

This file contains the flight data results from the merge iteration process. Flights are separated into those that were within the specified time period, but were only found in the ASPM dataset, flights that were only found in the Extractor dataset, and flights that were found in both datasets and were successfully merged. This file allows the user to see exactly which flights were not found by either database and allows them to compare those flights to those that were successfully merged. An outline of a Validation Complete file can be found below.

Table 9.2 – Validation Complete File Outline

ASPM Only Flights						
Flight	TEE Field 1	TEE Field 2	TEE Field 3	ASPM Field 1	ASPM Field 2	ASPM Field 3
Flight 1				##########	##########	##########
Flight 2				##########	##########	##########
TEE Only Flights						
Flight	TEE Field 1	TEE Field 2	TEE Field 3	ASPM Field 1	ASPM Field 2	ASPM Field 2
Flight 3	##########	##########	##########			
Flight 4	##########	##########	##########			
Shared Flights						
Flight	TEE Field 1	TEE Field 2	TEE Field 3	ASPM Field 1	ASPM Field 2	ASPM Field 2
Flight 5	##########	##########	##########	##########	##########	##########
Flight 6	##########	##########	##########	##########	##########	##########

9.2 Validation Merged

This file contains only the data for merged flights from the merge iteration process. This subset is useful for doing comparisons between the merged flights if the unmerged flights are not required or otherwise not preferred.

9.3 Validation Analysis

This file contains the results of the time analysis between the matched Extractor and ASPM flights. It contains each flight's operational runway, gate in/out time, takeoff/landing time, and the differences between each respective Extractor and ASPM time. This file allows the user to see how the Extractor-generated time data compares to ASPM time data. The table below gives a breakdown of every field in the Validation Analysis result file.

Table 9.3 – Field Breakdown in Validation Analysis Results File

Column Number	Field	Description
1	Date	Date at which flight occurred in YYYYMMDD format
2	Flight_ID	Flight's call sign
3	Operation_Type	Type of operation. A: Arrival, D: Departure
4	Aircraft_Type	Model of the airplane.
5	TEE_Track_Start_End_Loc	Type of area where the track starts or ends for departures or arrivals, respectively. Ex: Apron, Taxiway… etc.
6	TEE_Operational_Runway	Defines the runway end at which the operation happened in the TEE.
7	ASPM_Operational_Runway	Defines the runway end at which the operation happened in ASPM.
8	Runway_Same	Compares to see the TEE and ASPM runways are the same. Ex: Same or Different
9	TEE_Gate_Time	TEE Time of the estimated gate point
10	ASPM_Gate_Time	ASPM Time of the gate point
11	Diff_Gate_Time	Difference between the ASPM and TEE gate point times
12	TEE_On_Off_Time	TEE Time of the estimated on/off point
13	ASPM_On_Off_Time	ASPM Time of the on/off point
14	Diff_On_Off_Time	Difference between the ASPM and TEE on/off point times
15	TEE_Taxi_In_Out_Duration	Difference between TEE Gate time and On/Off time in seconds
16	ASPM_Taxi_In_Out_Duration	Difference between ASPM Gate time and On/Off time in seconds
17	Diff_Taxi_In_Out_Duration_Sec	Difference between ASPM and TEE taxi times in seconds
18	Diff_Taxi_In_Out_Duration_Min	Difference between ASPM and TEE taxi times in minutes

9.4 Validation Summary

This file contains a summary of the merge iteration process and shows how many flights were merged during each iteration. This gives the user an idea of how accurate the overall merge was as the later the iteration, the more lenient the conditions to be merged are. An example of Validation Summary can be found below.

Table 9.4 – Matching Parameters for Each Filter Iteration in ValidationASPM.py

CLT ASPM Data Validation Summary						
Iteration	Allowable_Time_ Difference	Carrier_ Check	Runway_ Check	TEE_Only_ Flights	ASPM_Only_ Flights	Merged_ Flights
0	0	0	0	304	367	0
1	120	Strict	TRUE	70	133	234
2	300	Strict	TRUE	68	131	236
3	600	Strict	TRUE	68	131	236
4	120	Strict	FALSE	18	81	286
5	120	Carrier	TRUE	18	81	286
6	60	None	TRUE	17	80	287

APPENDIX A: OUTPUT DATA DICTIONARY

Column Number	Field Name	Description
1	Flight_ID	Flight's call sign
2	Flight_Key	A unique integer representing each flight
3	Operation	Type of operation. A: Arrival, D: Departure
4	Aircraft_Type	Model of the airplane. Ex: B777
5	Airport	Airport at which the operation happened. Ex: ORD
6	Operational_Runway	Defines the runway end at which the operation happened. Ex: 27R
7	Enter_Rwy_Time	Time which airplanes entered the operational runway. For arrivals: time when they first passed the threshold For departures: time when they entered the departing runway. Format: yyyy-mm-dd HH:MM:SS (UTC)
8	Enter_Rwy_Lat	Latitude of first point inside operational runway
9	Enter_Rwy_Long	Longitude of first point inside operational runway
10	EnterRwy_On_Off_Dur	Time the plane took to touchdown (arrival) or wheels-off (departure) from Enter_Rwy_Time in seconds
11	On_Off_Lat	Latitude of Touchdown (arrivals) or Wheels-off (departures)
12	On_Off_Long	Longitude of Touchdown (arrivals) or Wheels-off (departures)
13	EnterRwy_On_Off_Dist	Distance from Operational_Runway threshold to touchdown (arrivals) or wheels-off (departures) location in meters
14	EnterRwy_ExitRwy_Dur	Runway occupancy time in seconds
15	Exit_Rwy_Time	Time which airplane vacates the runway. For arrivals: taking exits For departures: passing the opposite threshold or the first point outside of the departing runway Format: yyyy-mm-dd HH:MM:SS (UTC)
16	Exit_Rwy_Lat	Latitude of the exiting point
17	Exit_Rwy_Long	Longitude of the exiting point
18	Taxiway_Label	Label of taxiway used to enter or exit the operational runway
19	Enter_Exit_Movement_Area_Time	Time the movement area was entered or exited for departures or arrivals, respectively.
20	Enter_Exit_Movement_Area_Lat	Latitude the movement area was entered or exited for departures or arrivals, respectively.
21	Enter_Exit_Movement_Area_Lon	Longitude the movement area was entered or exited for departures or arrivals, respectively.
22	Gate_Lat	Latitude of the estimated gate point. If one cannot be found, then the last track point in the data is used for arrivals and the first track point in the data is used for departures
23	Gate_Long	Longitude of the estimated gate point. If one cannot be found, then the last track point in the data is used for arrivals and the first track point in the data is used for departures
24	Gate_Time	Time of the estimated gate point
25	Gate_Label	Label of gate (apron) area from FAA airports layout files
26	Track_Start/End_Loc	Type of area where the track starts or ends for departures or arrivals, respectively. Ex: Apron, Taxiway… etc.
27	Taxi_In_Out_Dur	Taxi in (arrivals) or Taxi out (departures) time in second Arrivals: time from Exit_Rwy point to the Near_Gate point Departures: time from Near_Gate point to the Enter_Rwy point
28	Avg_Taxi_Speed	Smoothed average taxiing speed (m/s)
29	Taxi_Dist	Taxiing distance (meters)
30	Wait_Time	Times that airplanes had taxiing speed lower than 3 m/s during taxiing
31	Taxi_Time_Minus_Wait	Taxi in (arrivals) or Taxi out (departures) time minus Wait_Time
32	Num_Runways_Crossed	The number of runways crossed during taxiing

33	Twy_Route	Taxi in/out route used in chronological order separated by commas
34	On_Off_Time	Time of Touchdown (arrivals) or Wheels-off (departures)
35	Gate_Diff_Time	Time difference between the Gate_Point and last point in track (arrivals) or first point in track (departures) in minutes. If this value is 0, then the code was unable to determine the exact gate point.
36	Takeoff/Landing_Roll_Lat	Latitude at the end of landing roll (arrivals) or start of takeoff roll (departures).
37	Takeoff/Landing_Roll_Long	Longitude at the end of landing roll (arrivals) or start of takeoff roll (departures).
38	Takeoff/Landing_Roll_Time	Time at the end of landing roll (arrivals) or start of takeoff roll (departures).
39	Takeoff/Landing_Roll_Dur	Time between Touchdown and end of landing roll (arrivals) or Enter Runway and start of takeoff roll (departures).
40	Takeoff/Landing_Roll_Speed	Speed at the end of landing roll (arrivals) or start of takeoff roll (departures).
41	Anomalies	List of anomalies associated with each flight if any exist
42	DeiceStart	Time of entry into the deicing polygon determined by the first track point inside.
43	DeiceEnd	Time of exit from the deicing polygon determined by the first track point outside.
44	DeiceTime	The deicing time from deicing polygon as determined by the deicing algorithm (seconds)
45	DeiceQueue	The waiting time inside deicing polygon as determined by the deicing algorithm (seconds)
46	DeiceTravel	The traveling time inside deicing polygon as determined by the deicing algorithm (seconds)
47	DeicePad	The number of the deicing pad that was used (if there were multiple pads)
48	DeiceStartDemand	The start of deicing demand as determined by the deicing algorithm based on the given list of staging taxiways (Seconds since 1-1-1970)
49	DeiceEndDemand	The end of deicing demand as determined by the deicing algorithm based on the given list of staging taxiways (Seconds since 1-1-1970)

APPENDIX B: TEE/ASPM VALIDATION REPORT

In the course of developing and testing ValidationASPM.py, approximately 24.5 hours of flight data for flights across 5 airports in the TEE and in ASPM were analyzed and merged. This amounted to a grand total of 1,524 flights being analyzed. The results of the merge iteration process for each airport date are summarized in the table below.

Table B.1 – ValidationASPM.py Merge Results for Approximately 24 Hours of Data

Airport	Date	Time	Total Flights	TEE-Only Flights		ASPM-Only Flights		Merged Flights	
				#	Out of Total	#	Out of Total	#	Out of Total
CLT	2/22/2022	12:00-15:30	320	17	5.3%	15	4.7%	288	90.0%
EWR	2/8/2022	12:00-15:30	190	0	0%	28	14.7%	162	85.3%
EWR	2/15/2022	12:00-15:30	195	4	2.1%	43	22.1%	148	75.8%
EWR	2/22/2022	12:00-15:30	209	1	0.5%	52	24.9%	156	74.6%
JFK	2/22/2022	12:00-15:30	212	6	2.8%	12	5.7%	194	90.5%
LGA	2/22/2022	12:00-15:30	247	3	1.2%	14	5.7%	230	93.1%
PHL	2/22/2022	12:00-15:30	151	10	6.6%	27	17.9%	114	75.5%
Total			1524	41	2.7%	191	12.5%	1292	84.8%

As shown, an overall merge rate of just below 85% was observed when accounting for all flights analyzed. However, the individual merge rates across airports and even across days varied significantly. See how the merge rate for EWR fell from roughly 85% to 75%, despite the date being the same day of the week. It's also worth noting that there is a large difference between airports with high merge rates (CLT, JFK, LGA) and airports with low merge rates (EWR and PHL). This indicates that individual circumstances surrounding the airport play a major role in the successful merge rate for a given day.

In addition to seeing how effective ValidationASPM.py was at successfully merging flights between TEE and ASPM, part of the purpose of this study was to see how closely the time data generated by the TEE compared to the data in ASPM. The difference between taxi times was of particular interest. During testing, it was noted that most flights that were successfully merged had small differences between the ASPM and TEE takeoff/landing times (99.8% were less than 2 minutes apart). That was the main reason why takeoff/landing time was used as one of the primary merge criteria.

B.1 – CLT Validation

Of the 5 airports in the analysis, CLT had the distribution that most resembled the overall distribution of the analysis. While there were a few examples where there was a larger difference between the ASPM and TEE taxi times for a given flight, most flights exhibit a small, positive difference between ASPM and TEE taxi times. The distribution of time differences can be found in the figure below.

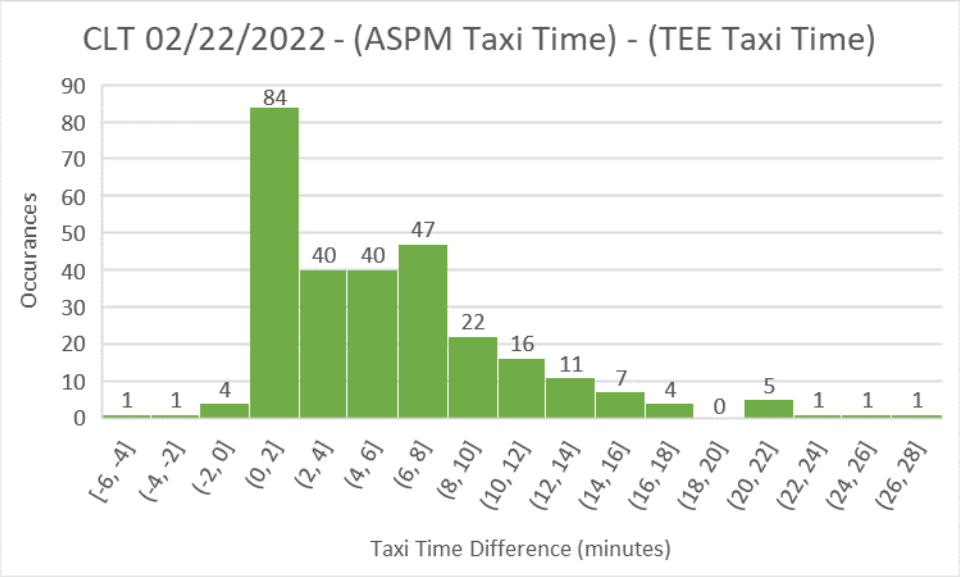


Figure B.1 – Final Difference between ASPM and TEE Taxi Time for Flights on 02/22/2022 CLT

While there were initially a number of flights that would not properly merge on CLT, many of these issues were addressed between versions V7 and V8. As it stands, there are no real outliers in the flight list for CLT.

Despite having the highest number of TEE-Only flights, CLT manages to have only the second-worst rate of TEE-Only flights. Even then, 16 of the 17 TEE-Only flights were not affiliated with any airline. We know this because these specific flights were identified by their tail number (Usually formatted as N###XX) instead of their airline code and flight number. Additionally, validation plots for these flights revealed that these flights would go hangers that were not typically used by the major cargo companies at the airport, so there is good reason to believe that these planes did not belong to a specific airport. An example trajectory for this type of flight can be found below.

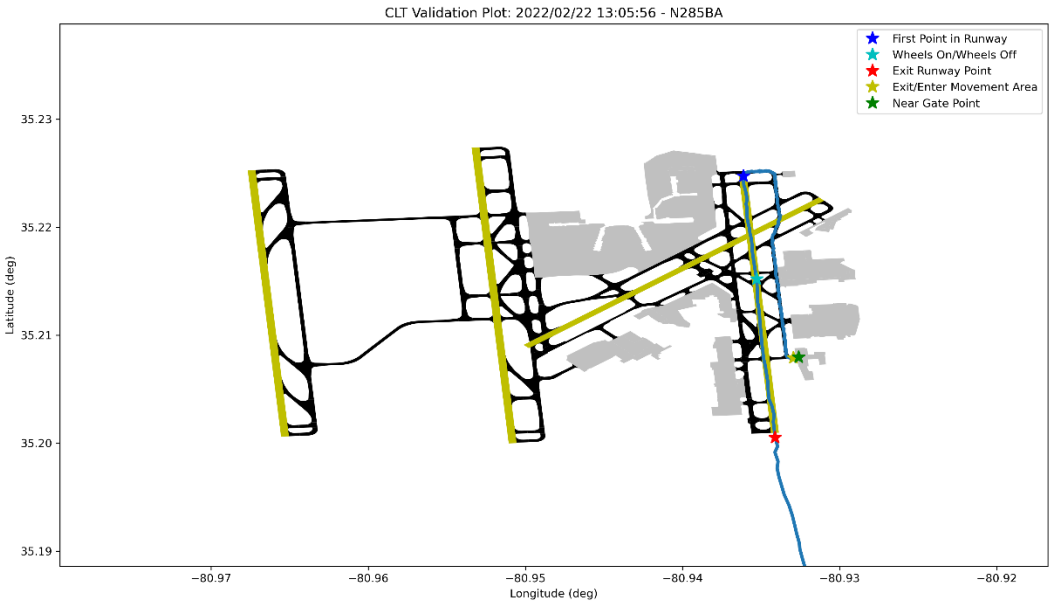


Figure B.2 – Example Flight Trajectory of a Non-Airline-Affiliated Flight

The single TEE-Only flight that was identified with an airline code was PAT322 or Priority Air Transport 322. Considering that this flight is owned by the Department of the Army, it makes sense that ASPM would not have flight data for this aircraft. In other words, despite having the most TEE-Only flights among the airports analyzed in this report, there is a logical reason why ValidationASPM.py was not able to find a match in the ASPM dataset.

B.2 – EWR Validation

Of the 5 airports, EWR displayed the most range in the difference between ASPM and TEE Taxi times. EWR was the only airport to have instances where the difference between taxi times exceeded 50 minutes. It should be stated that all but 0.2% flights that were successfully merged in ValidationASPM.py had ASPM and TEE takeoff/landing times that were less than 2 minutes apart, so it is quite likely that these are valid merges. The figure below shows the distribution for EWR.

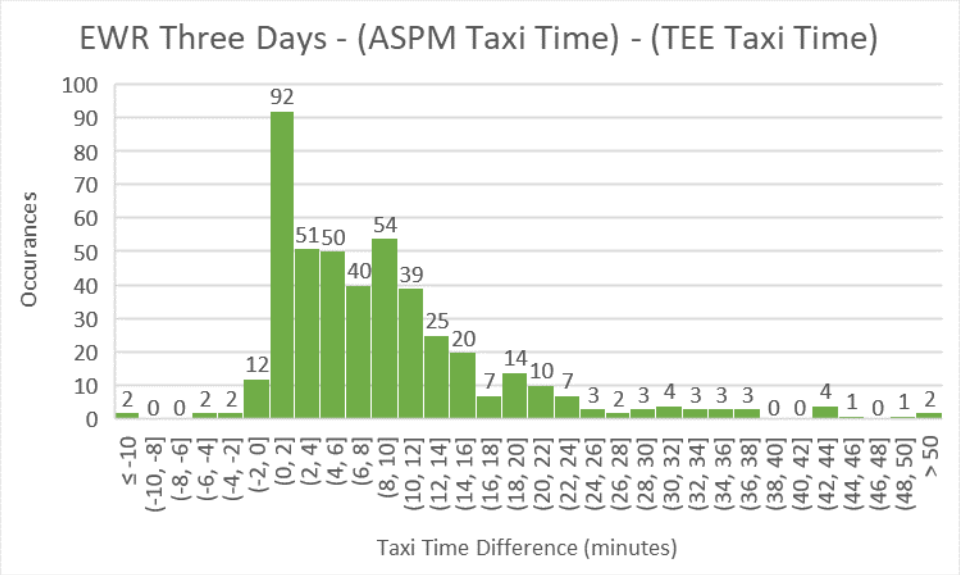
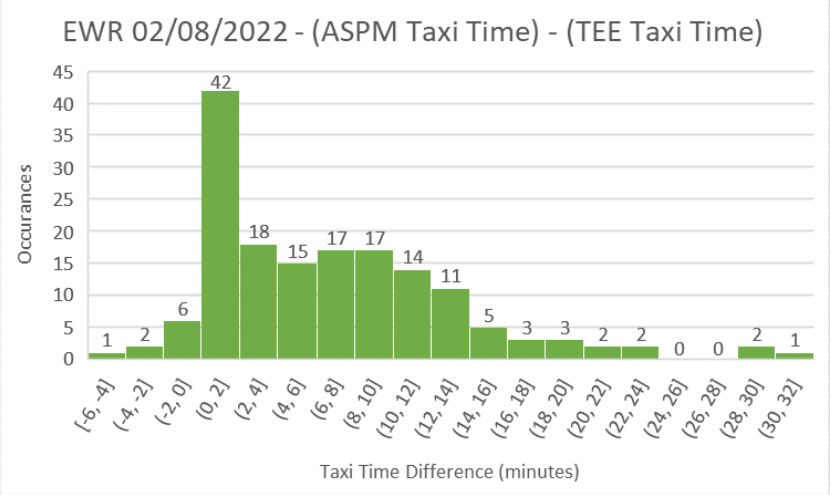


Figure B.3 – Final Difference between ASPM and TEE Taxi Time for Flights on 02/08, 02/15, and 02/22/2022 EWR

Unlike the other airports, the data for EWR spans three days, being the latter 3 Tuesdays of February 2022. Because of that, the team was able to see how merge rates changed across different days. In the case of EWR, the merge statistics for 02/08 were notably higher than for the remaining days (+~10%). Additionally, the distribution of taxi time differences was also much closer for 02/08 when compared to those of the remaining days, as shown by the figure below.



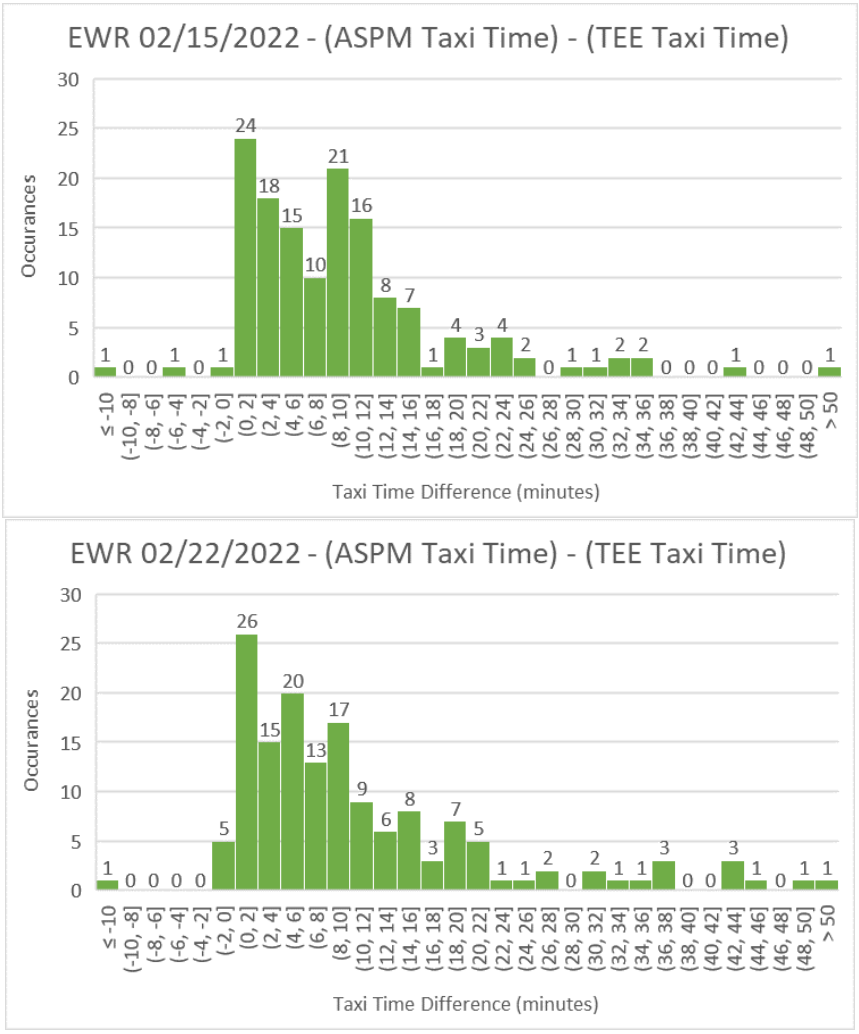


Figure B.4a, b, and c – Individual Differences between ASPM and TEE Taxi Time for Flights on 02/08, 02/15, and 02/22/2022 EWR

Notably, larger queues formed on 02/15 and 02/22 compared to 02/08. As a result, many planes would likely turn their transponders on, but remain idle as a spot in the queue opened up. As a result, these planes would still be recorded through the SWIM Feed as having begun departure but would not have actually left the apron area. While the Extractor determines the gate point s the point that the plane begins to move continuously away from the apron, it's possible that ASPM would record the gate point earlier. This would make it much more difficult for the Extractor to correctly determine the exact point that the plane would be considered to have left/entered the gate, which would account for the larger time differences on later days.

Despite having the highest total amount of flights and the highest rates of ASPM-Only flights, EWR managed to have the lowest overall TEE-Only rate. Part of this seems to be that there were relatively few non-airline affiliated flights that would inflate the number of TEE-Only Flights. However, one of these flights, AIC105, is particularly strange. While there were no records of an AIC105 in the ASPM dataset, there were records of an AIC106. A summary table of the gate time, takeoff/landing time, and taxi time for TEE and ASPM for both of these flights is included below.

Table B.2 – Time Breakdown of Flights AIC 105 and AIC 106 for EWR

Row	Event	ASPM (Scheduled)	ASPM (Actual)	TEE
	AIC 105 - Arrival			
1	Wheels On	N/A	N/A	12:38:50
2	Gate In/Nearest Point	N/A	N/A	12:49:08
3	Taxi Time (Row 2 – Row 1)	N/A	N/A	10 min
	AIC 106 - Departure			
5	Gate Out/Nearest Point	16:15:00	19:07:00	N/A
6	Wheels Off	16:27:00	19:31:00	N/A
7	Taxi Time (Row 6 – Row 5)	12 min	24 min	N/A

This was especially strange as these two flights used the same aircraft type and had the same pair of airports, so it stands to reason that this was the same plane. It also makes sense that the AIC106 data wouldn't appear in the TEE as it was outside of the time period (12:00-15:30). It's difficult to say why AIC105 did not appear in the ASPM dataset since there did not seem to be any anomalies in the TEE data. However, this would indicate that the TEE is capable of tracking flights that should appear in the ASPM reports, yet do not.

B.3 – JFK Validation

Out of the 5 airports available for validation, JFK displayed the narrowest distribution for taxi time differences with relatively few outliers. JFK also had the second-highest merge rate and relatively low database-specific rates, making it one of the more accurate datasets. The distribution for the differences between ASPM and TEE taxi times can be found in the image below.

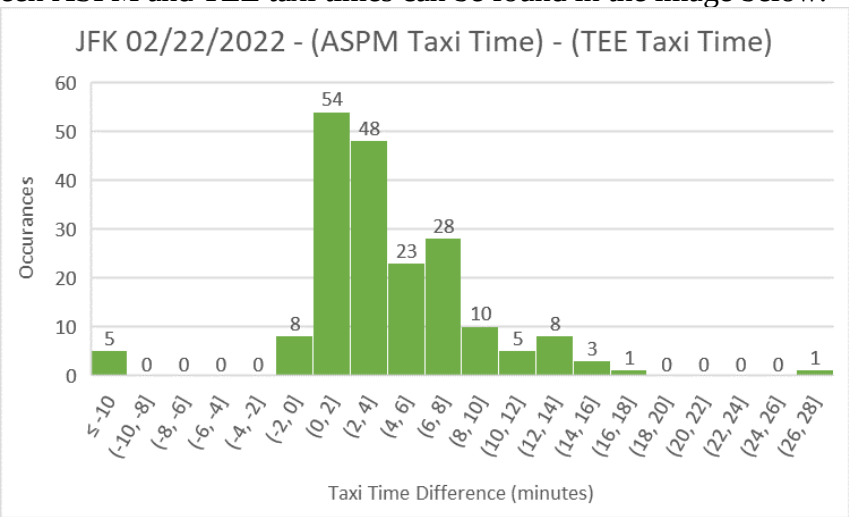


Figure B.5 – Final Difference between ASPM and TEE Taxi Time for Flights on 02/22/2022 JFK

While most of the data is centered around the 0-to-4-minute mark, JFK manages to have the most instances where the TEE taxi time is much larger than the ASPM taxi time (difference < -10 minutes). The main reason for this seems to be that a number of these flights turned on their transponders and then sat in the apron for a long period of time. If the plane had moved out of the gate only to stop within the apron at least 50 feet away, the TEE would count this as the point that the plane left the gate. If this point is determined differently in ASPM, then it would explain why there are flights with substantially larger TEE taxi times.

The one exception to this is JBU2637. Unlike the other flights with larger TEE taxi times, JBU2637 was not idle in the apron for a long period of time. Instead, JBU2637 originally departed around 12:46, but would return to the gate at approximately 13:30. It would then depart again at about 14:30 and then successfully take off at 14:38. The path JBU2637 took is illustrated below.

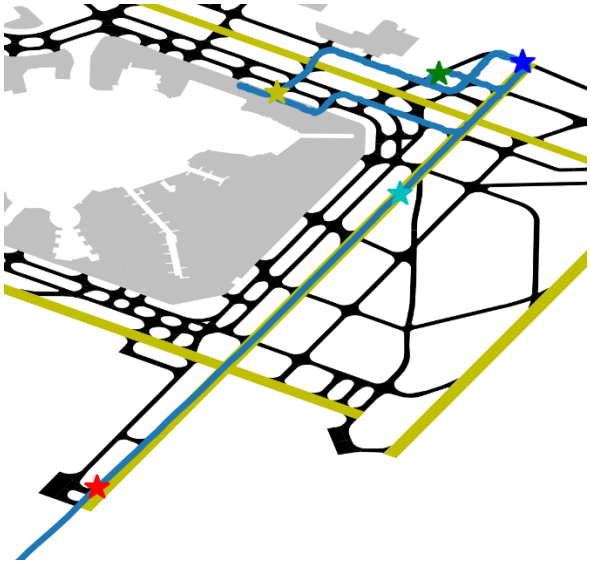


Figure B.6 – Overall Flight Trajectory of JBU 2637

While the second departure time would be far closer to the ASPM gate departure time of 14:25, it would not be truly accurate to say that the plane originally departed at 14:30. As a result, the TEE gate time uses the first gate time, resulting what looks like a taxi time difference of over 90 minutes.

Regarding non-merged flights, JFK had a number of TEE-Only flights that had proper flight numbers. Even though these flights clearly departed/arrived from the terminal as shown in the validation plot below, the program could not find a match in the ASPM dataset.

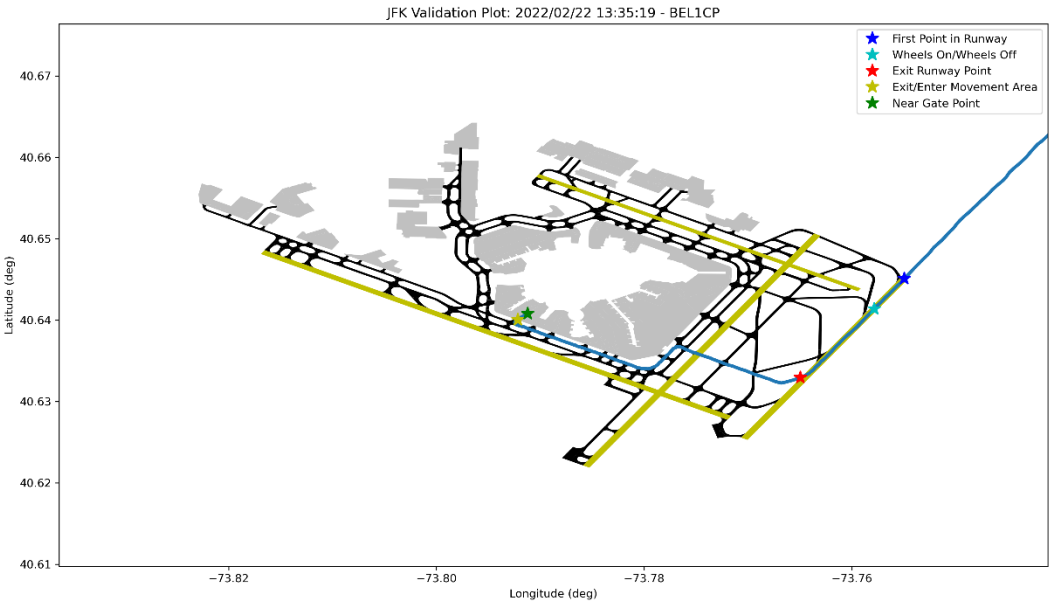


Figure B.7 – Example Flight Trajectory of Non-ASPM Carrier Flight

Upon looking in the ASPM database, there were no flights from the following airlines: CargoJet (CJT), Philippine Air Lines (PAL), and Brussels Airlines (BEL). After checking the list of ASPM carriers, none of the above-mentioned carriers were part of the list. As such, it makes sense that they would not be successfully merged. This also shows that the SWIM Feed is capable of monitoring carriers outside of those that report to ASPM.

B.4 – LGA Validation

Of the 5 airports analyzed, LGA displayed the most normal distribution of taxi time differences. This amounts to a significant number of instances where the TEE taxi time was greater than the ASPM taxi time for a merged flight. The distribution of taxi time differences for LGA can be found in the figure below.

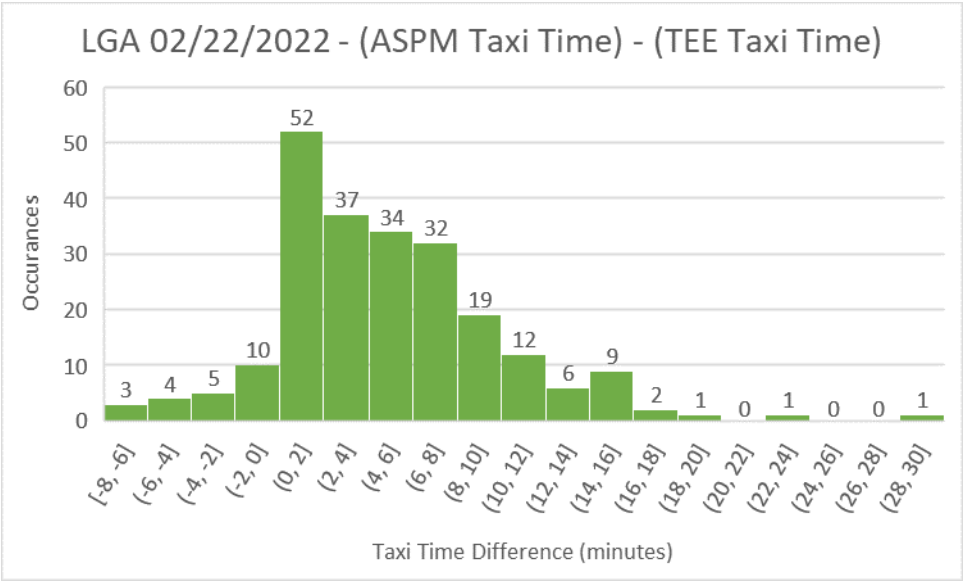


Figure B.8 – Final Difference between ASPM and TEE Taxi Time for Flights on 02/22/2022 LGA

The reason there are so many flights with longer ASPM taxi times is that the airport layout data for LGA is slightly outdated. Compare the most recent FAA diagram of LGA with the current map used in the TEE.

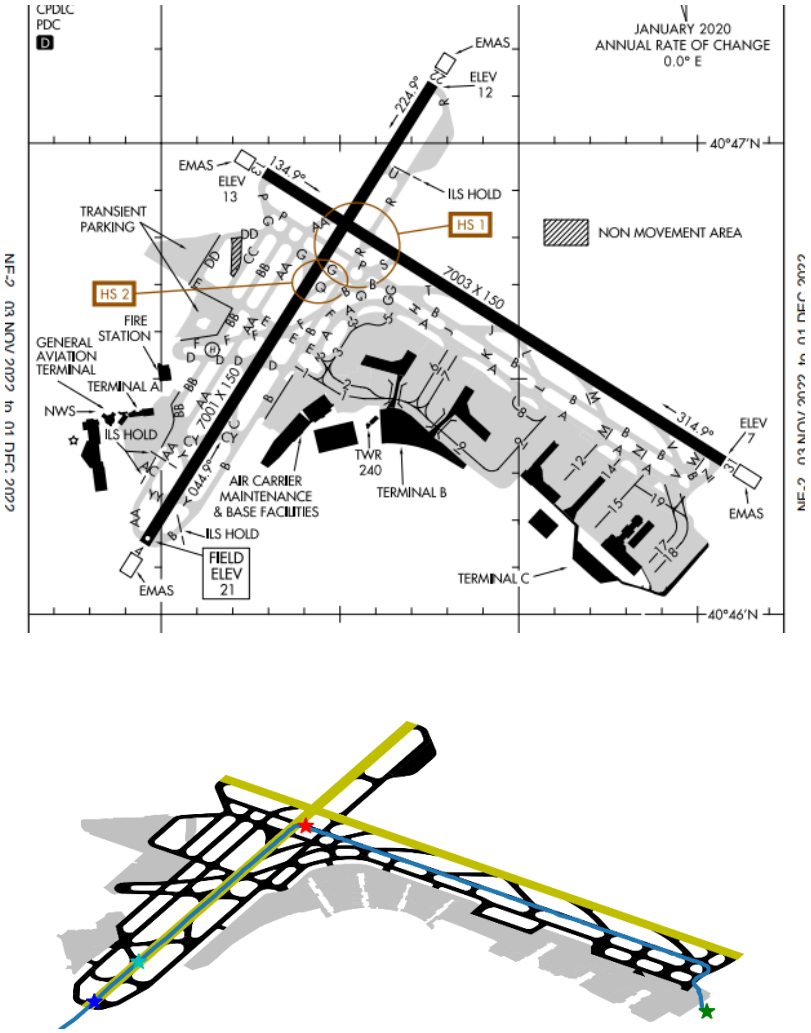


Figure B.9a and b – FAA Diagram of LGA and TEE Airport Layout Data for LGA

Notice that the recent addition to Terminal C is not reflected in the TEE data as of this point. As a result, any flights that depart or arrive from this extension are not within the apron, making it more difficult for the program to determine a valid gate point. If the program cannot determine a gate point, it will default to the last point in the track, inflating the overall TEE taxi time since the gate point is later. In other words, the incomplete layout data is what is primarily causing the TEE taxi time to be greater than the ASPM taxi time.

While LGA displayed the highest overall merge rate at 92.8%, there were still instances where the Extractor could not merge flights. One particular instance was the result of duplicate flight records within ASPM. In this instance, there were two separate flights that had the same aircraft ID, takeoff time, and arrival airport. The chart below shows the similarities and differences between these flights.

Table B.3 – Differences between LGA Duplicate Flights

Flight_ID	Flight_Key	Operation	FAACARRIER	FLTNO	DEP_LOCID	ARR_LOCID	Time
			AAL	456	LGA	ORD	12:26
AAL456	112864	D	AAL	456	LGA	ORD	12:26

There were similar instances that occurred in EWR, JFK, and PHL, however those instances had different arrival airports. This indicates that these duplicate flights exist as a result of the airline companies changing the specific route and aircraft used. Yet the takeoff time is the same for both flights, and ValidationASPM.py will only merge the first ASPM flight in the list with the corresponding TEE flight. As a result, these duplicate ASPM flights are considered ASPM-Only flights in the results. As such, it must be said that the accuracy of the merge process depends on the accuracy and thoroughness of the information provided in ASPM.

B.5 – PHL Validation

Of the 5 airports analyzed, PHL was the only instance where there were no flights where the TEE taxi time was greater than the ASPM taxi time (difference < 0). The overall distribution was also fairly concentrated, resulting in the graph below.

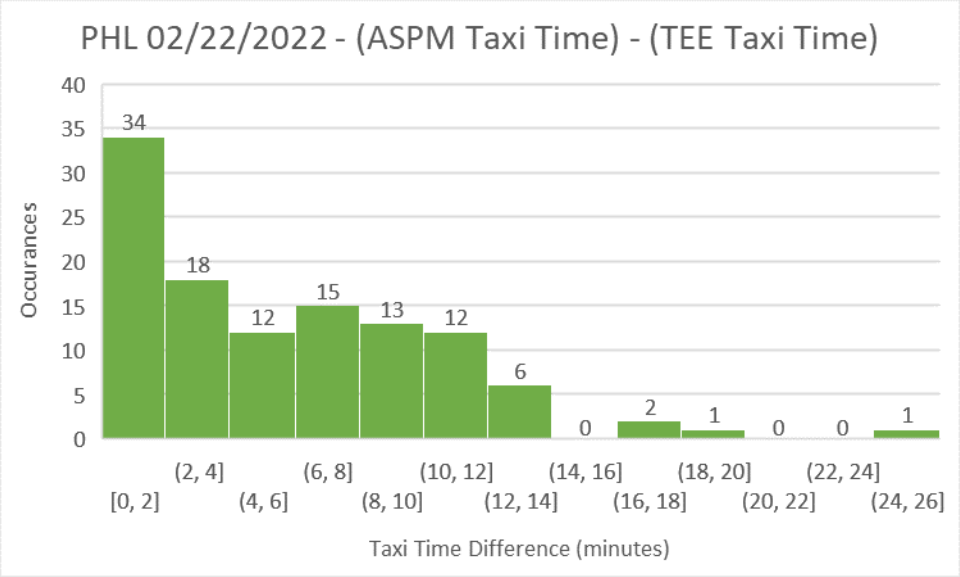


Figure B.10 – Final Difference between ASPM and TEE Taxi Time for Flights on 02/22/2022 PHL

Despite this, Philadelphia had one of the worst successful merge rates out of the 7 days analyzed, second only to EWR on February 22. Both the rate of TEE-Only flights and the rate of ASPM-Only flights was higher for PHL than typical. Every TEE-Only flight during this period was an unaffiliated flight (no airline code) and several flights both landed and departed within the time period (showed as 2 results in results file).

Perhaps more interestingly, are the ASPM-Only flights. Of the 27 ASPM-Only flights, 25 of them were arrival flights and 20 of them were United Airlines flights. This is a remarkable degree of

consistency for flights that were missed by the SWIM Feed and indicates that this combination of characteristics (arrival and airline) resulted in them being overlooked by the SWIM Feed. Further supporting this is that there were two specific time periods, between 12:00-12:25 and between 14:00-14:10, where there were several consecutive missed flights. This rarely occurred for the other airports and lasted for much shorter time frames (5 minutes and only 2-3 consecutive flights). Below is a chart showing the flights that occurred in the described time periods and whether they were successfully merged. Flights with missing fields are ASPM-Only.

Table B.4 – Excerpt of Arrivals That Occurred on 02/22/2022 at PHL

Flight_ID	Operation	Aircraft_Type	Airport	FLTNO	TAILNO	ACTONTM
RPA3419	A	E170	PHL	RPA	3419	11:59
SWA1751	A	B738	PHL	SWA	1751	12:01
				AAL	835	12:03
				RPA	4749	12:09
PDT6209	A	E145	PHL	PDT	6209	12:12
				RPA	4614	12:16
				AAL	1684	12:19
DAL2441	A	B739	PHL	DAL	2441	12:21
				AAL	1687	12:24
				AAL	2717	12:26
EJA781	A	CL35	PHL	EJA	781	12:26
JIA5542	A	CRJ9	PHL	JIA	5542	12:28
PDT6193	A	E145	PHL	PDT	6193	13:57
				AAL	593	14:01
				AAL	699	14:03
				AAL	2908	14:06
RPA4773	A	E75L	PHL	RPA	4773	14:06
				AAL	2384	14:07
VTE526	A	E135	PHL	VTE	526	14:08
UAL1507	A	A320	PHL	UAL	1507	14:10
				AAL	1333	14:12
JIA5411	A	CRJ7	PHL	JIA	5411	14:15
ASH6051	A	E75L	PHL	ASH	6051	14:21

There did not appear to be any consistency between the missed flights beyond operation type, time, or airline. Neither aircraft type nor runway entrance seemed to affect whether a flight was picked up by the SWIM Feed. Even so, it's especially strange that these missed flights were so densely clustered and shared any similarities. As such, it stands to reason that this combination of factors is the reason why these particular flights were not able to be merged: the SWIM Feed was not able to detect flights with these characteristics.

B.6 – All Flights Validation

The final histogram of the difference between ASPM and TEE taxi times after implementing these changes has been included below.

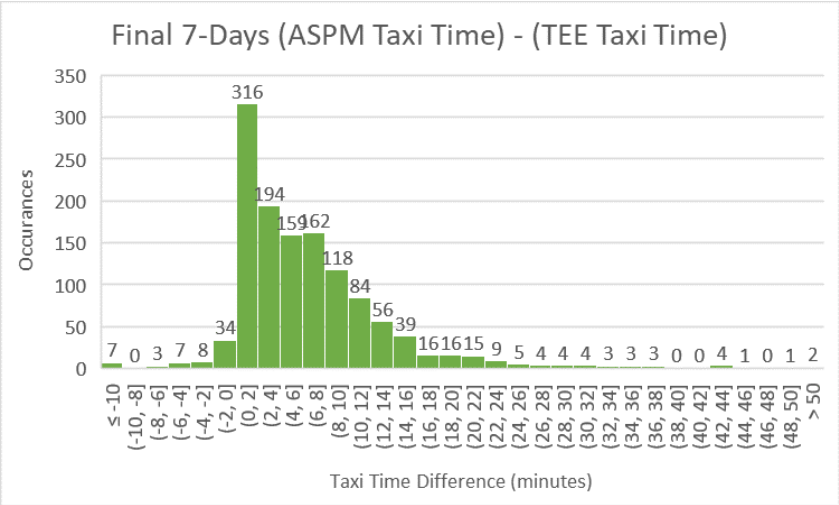


Figure B.11 – Final Difference between ASPM and TEE Taxi Time for 1,290 Flights

Here, the data displays better adherence to the right-skewing distribution centered slightly more than 0 minutes difference. Despite the changes made, a number of flights still displayed large taxi time differences. Interestingly enough, most of the flights with high differences between ASPM and TEE Gate times come from EWR as shown when you remove the ER flights from the histogram.

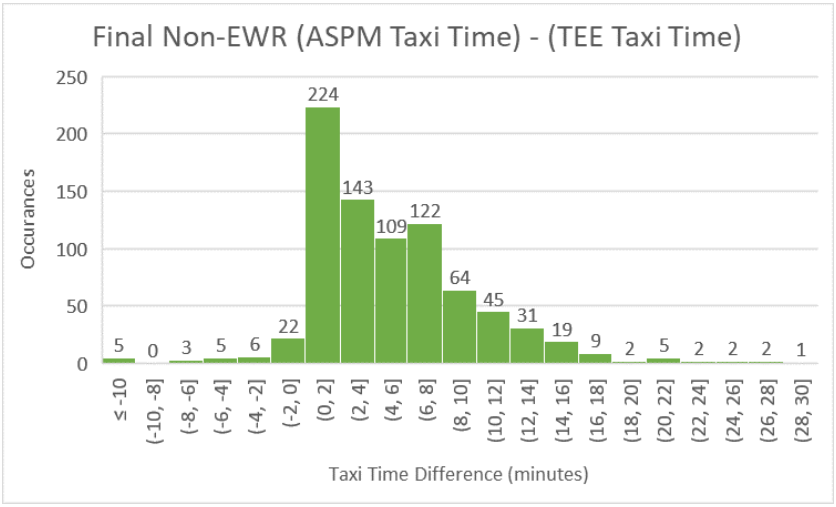


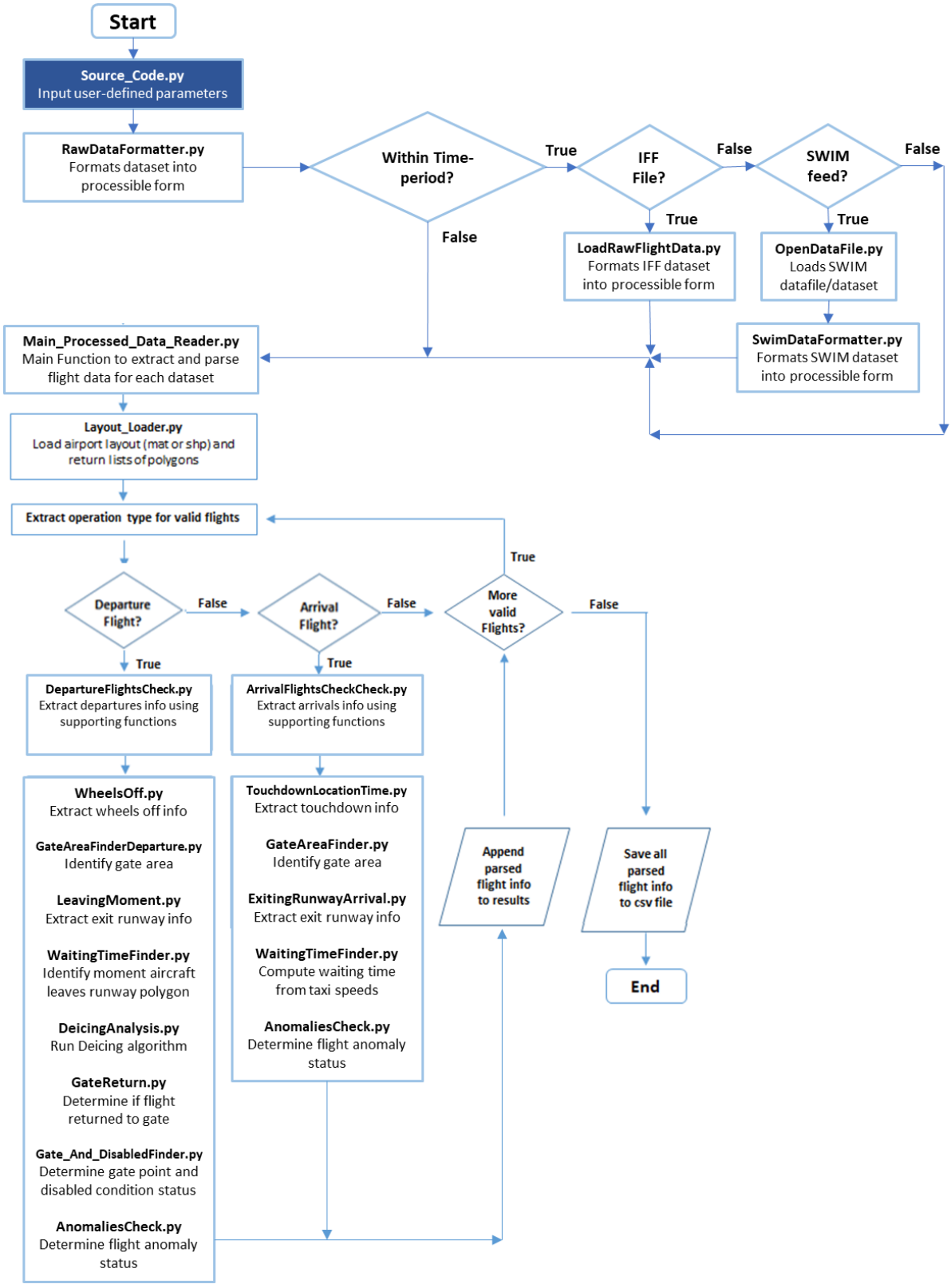
Figure B.3 – Final Difference between ASPM and TEE Taxi Time for 825 flights (EWR Excluded)

Here, the variance declines significantly with the maximum difference being 30 minutes. Considering that the majority of airports seem to follow this general distribution with minor variations, it seems reasonable to say that most airports would show similar differences between ASPM and TEE taxi times on an average day.

In regard to the flights that were not able to be merged, explanation has been given for the majority of TEE-Only flights among the flights that were studied. With that in mind, that would mean that there would almost no TEE flights that could be matched with ASPM flights. In other words, it would be impossible to merge the ASPM-Only flights since there would be no data in the TEE to merge them with. It is still not certain why specific flights were not picked up by the SWIM Feed, the source of flight data for the TEE. The results of the EWR study would indicate that busier conditions at an airport would result in some flights being missed, while the PHL study showed that flights with very specific conditions (AAL Arrivals) could be missed.

This study covered slightly more than 24 hours of flight data within a 3.5-hour window and 5 airports. Considering that merge results seem to be dependent on both time period and individual airport conditions, more studies should be done before this tool is used to any professional capacity. Even so, considering that the program is able to successfully pair the TEE results with ASPM counterparts nearly 85% of the time, it seems reasonable to say that the TEE and the data that come from the SWIM Feed are able to accurately recreate and evaluate the important events and routes of flights that operate within the 42 airports covered by the current system.

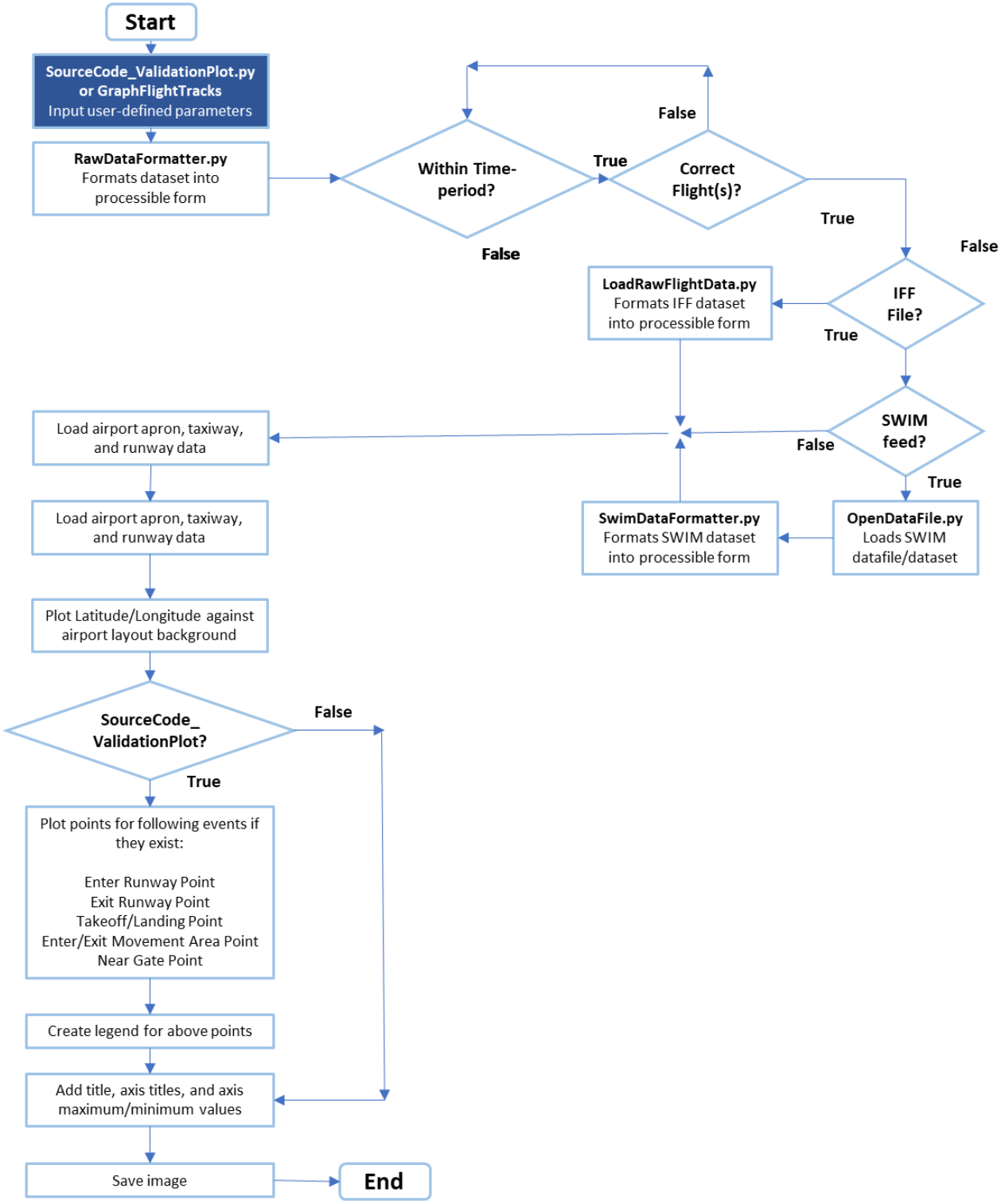
APPENDIX C: TAXI EVENT EXTRACTOR SOURCE CODE FLOW CHART



Source_Code.py: This program processes flights using either "ArrivalFlightsCheck.py" or "DepartureFlightsCheck.py." These functions extract various key parameters, including flight identification information, routing details, takeoff/landing times and coordinates, and overall taxi times. This data is saved and outputted as an Excel file.

- RawDataFormatter.py: This is the primary formatting program used for each of 6 codes used to generate an output. This includes ***Source_Code.py, SourceCode_ValidationPlot.py, GraphFlightTracks.py, Animation Batch.py, Heatmaps.py,*** and ***Multimaps.py***. It takes flight data from either a legacy IFF file or from the SWIM feed and reformats it into a Python dataset using "LoadRawFlightData.py" or "SWIMDataFormatter.py" respectively. This program outputs the dataset, date, and output destination.
    - LoadRawFlightData.py: This function takes the data from an IFF flight data file and converts it into a format readable by "Main_Processed_Data_Reader.py." This program is designed to remove any non-successful flights that do not achieve the necessary elevation difference.
    - SWIMDataFormatter.py: This function takes the data from the SWIM feed and converts it into a format readable by "Main_Processed_Data_Reader.py." This function accounts for the unique way flight data is identified and split by the SWIM feed.
- Main_Processed_Data_Reader.py: This is the primary function in the "Source_Code." It takes the formatted dataset and processes it using either the "ArrivalFlightsCheck.py" or "DepartureFlightsCheck.py" function.
    - Layout_Loader.py: This function loads the necessary airport layout data from the appropriate files. This function accepts both .mat and .shp files.
    - ArrivalFlightsCheck.py: This function takes arrival flight data and extracts the information on the important events
    - DepartureFlightsCheck.py: This function takes departure flight data and extracts the information on the important events
    - GateReturn.py: Determines whether a departure has left and returned to the gate based on its location. Supports AnomaliesCheck.py
    - Gate_And_DisabledFinder.py: Determines the probable point at which the departure left the gate as well as if it can be consider disabled based the amount of time spent idle on taxiways. Supports AnomaliesCheck.py
    - AnomaliesCheck.py: Checks each flight for different anomaly conditions and appends them to a list.
    - WaitingTimeFinder.py: Calculates the time spent not moving.
    - TouchdownLocationTime.py: Calculates the landing time and location. (A)
    - GateAreaFinder.py: Determines the general terminal of a flight. (A)
    - ExitingRunwayArrival.py: Determines runway exit time/location values. (A)
    - DeicingAnalysis.py: Determines deicing pathway statistics (D)
    - WheelsOff.py: Calculates the takeoff time and location (D)
    - GateAreaFinderDeparture.py: Determines the general terminal of a flight. (D)
    - LeavingMoment.py: Determines runway exit time/location values. (D)
- SaveOutput: This file takes a dataset and outputs it as an Excel file

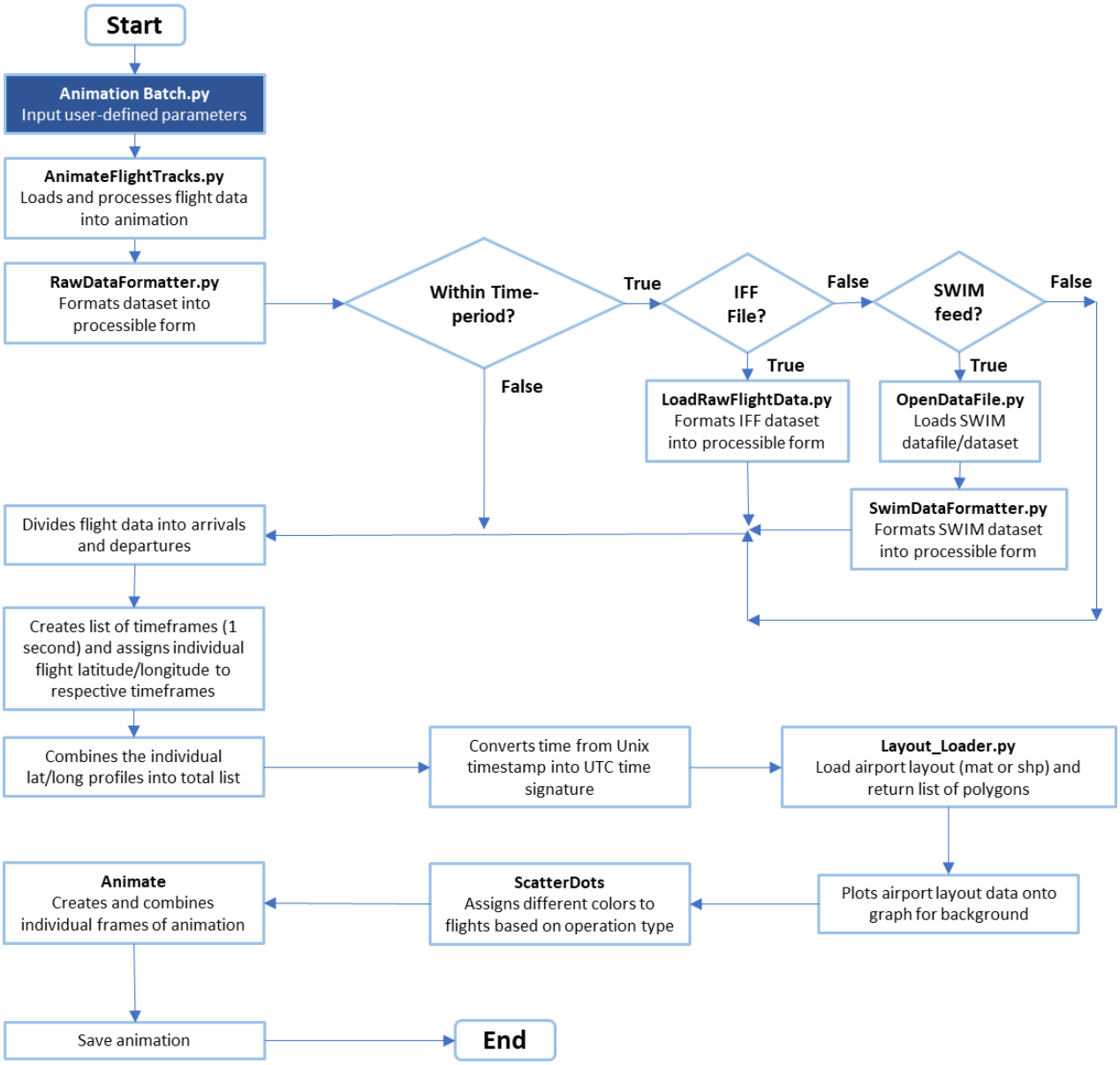
APPENDIX D: TAXI EVENT EXTRACTOR FLIGHT TRAJECTORY PLOT FLOW CHART



SourceCode_ValidationPlot.py: This program takes the flight data for a specified flight and maps it on top of the airport layout, recreating the flight's trajectory. Once mapped, the program will also note the location of several important events, such as near-gate point and takeoff/landing and mark them as a star on the map. This image is outputted as a .png.
GraphFlightTracks.py: This program takes the flight data for a specified flight and maps it on top of the airport layout, recreating the flight's trajectory. Unlike "SourceCode_ValidationPlot," this program does not mark the location of important events.

- RawDataFormatter.py: This is the primary formatting program used for each of 6 codes used to generate an output. This includes ***Source_Code.py, SourceCode_ValidationPlot.py, GraphFlightTracks.py, Animation Batch.py, Heatmaps.py,*** and ***Multimaps.py***. It takes flight data from either a legacy IFF file or from the SWIM feed and reformats it into a Python dataset using "LoadRawFlightData.py" or "SWIMDataFormatter.py" respectively. This program outputs the dataset, date, and output destination.
    - LoadRawFlightData.py: This function takes the data from an IFF flight data file and converts it into a format readable by "Main_Processed_Data_Reader.py." This program is designed to remove any non-successful flights that do not achieve the necessary elevation difference.
    - SWIMDataFormatter.py: This function takes the data from the SWIM feed and converts it into a format readable by "Main_Processed_Data_Reader.py." This function accounts for the unique way flight data is identified and split by the SWIM feed.

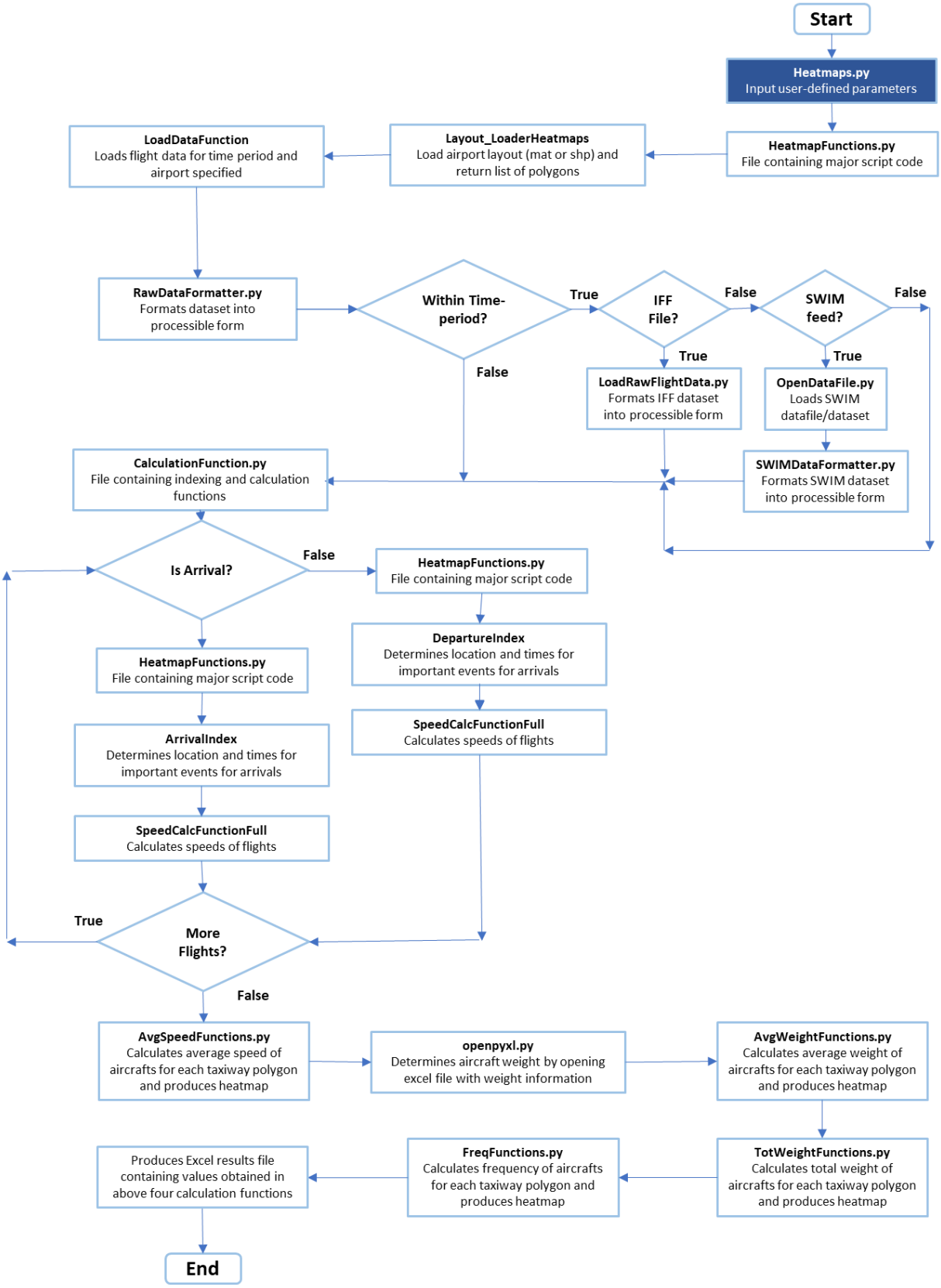
APPENDIX E: TAXI EVENT EXTRACTOR ANIMATION FLOW CHART



Animation Batch.py: This program takes the flight data and splits it into a list of departures and arrivals, which are mapped separately. For every second within the specified time period, the program will generate a snapshot of every plane's location at that point in time and then combine the resulting images. The result is an .mp4 showing planes' movements during the time period.

- RawDataFormatter.py: This is the primary formatting program used for each of 6 codes used to generate an output. It takes flight data from either a legacy IFF file or from the SWIM feed and reformats it into a Python dataset using "LoadRawFlightData.py" or "SWIMDataFormatter.py" respectively. This program outputs the dataset, date, and output destination.
    - LoadRawFlightData.py: This function takes the data from an IFF flight data file and converts it into a format readable by "Main_Processed_Data_Reader.py." This program is designed to remove any non-successful flights that do not achieve the necessary elevation difference.
    - SWIMDataFormatter.py: This function takes the data from the SWIM feed and converts it into a format readable by "Main_Processed_Data_Reader.py." This function accounts for the unique way flight data is identified and split by the SWIM feed.

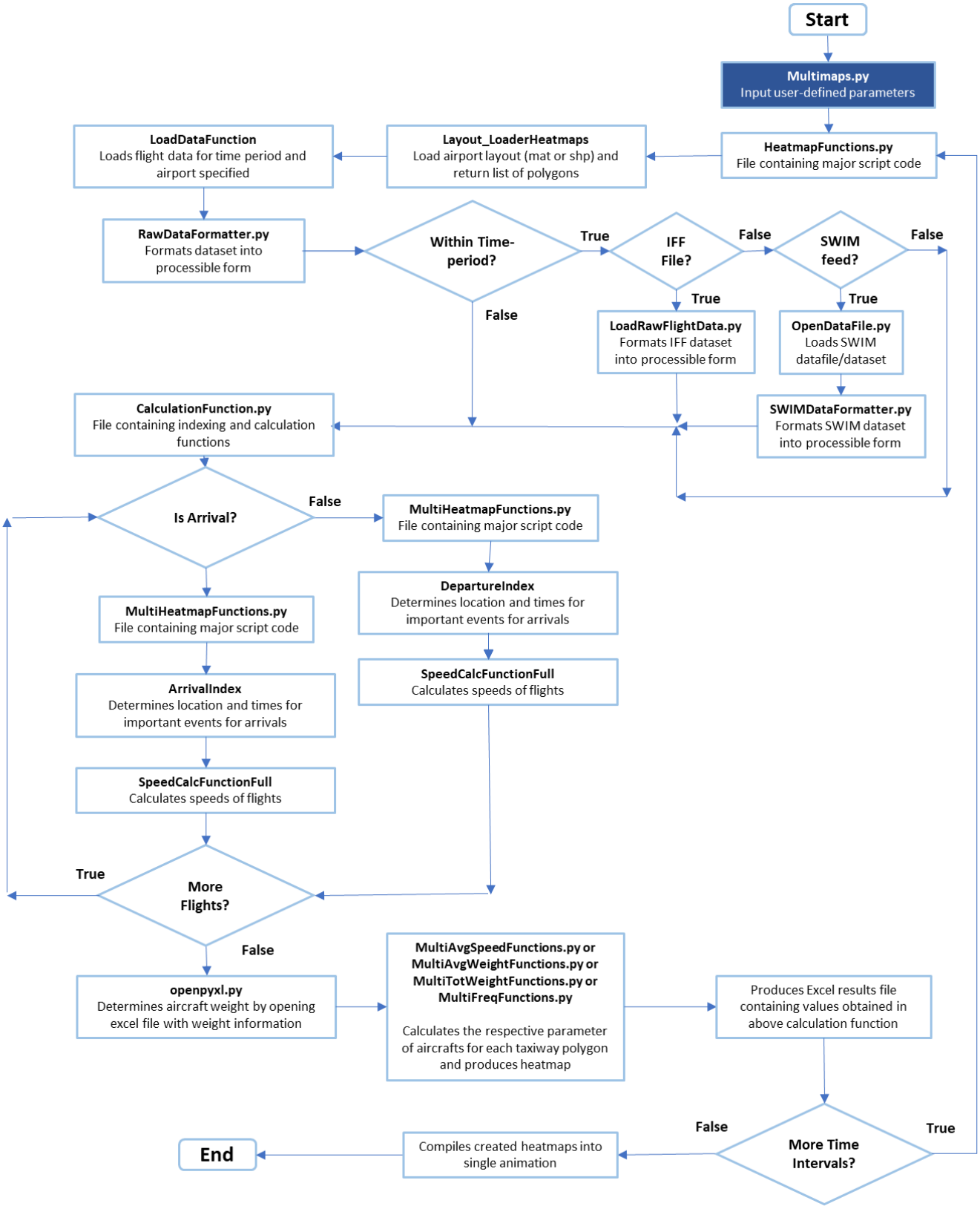
APPENDIX F: TAXI EVENT EXTRACTOR HEATMAPS FLOW CHART



Heatmaps.py: This program compiles the flight data to obtain the average and total speed and weight values for each taxiway during the time period. These values are recorded in color-graduated 'heatmaps,' which are then outputted as .png files. An Excel file containing the taxiway values is also created, with each row noting a single polygon's operational information.

- RawDataFormatter.py: This is the primary formatting program used for each of 6 codes used to generate an output. It takes flight data from either a legacy IFF file or from the SWIM feed and reformats it into a Python dataset using "LoadRawFlightData.py" or "SWIMDataFormatter.py" respectively. This program outputs the dataset, date, and output destination.
    - LoadRawFlightData.py: This function takes the data from an IFF flight data file and converts it into a format readable by "Main_Processed_Data_Reader.py." This program is designed to remove any non-successful flights that do not achieve the necessary elevation difference.
    - SWIMDataFormatter.py: This function takes the data from the SWIM feed and converts it into a format readable by "Main_Processed_Data_Reader.py." This function accounts for the unique way flight data is identified and split by the SWIM feed.
- HeatmapFunctions.py/MultiHeatmapFunctions.py: These larger functions containing function definitions for ***Heatmaps.py/Multimaps.py***. These include airport layout data imports, flight data imports, and functions to determine route and speed.
    - Layout_LoaderHeatmaps.py: This function loads the necessary airport layout data from the appropriate files. This function accepts both .mat and .shp files.
    - LoadDataFuntion: This function uses "RawDataFormatter" to convert and import a formatted flight dataset. Additionally, this function will isolate the flight data that falls within the specified time period.
    - ArrivalIndex: This function determines if an arrival successfully landed and at what point the flight exited the runway.
    - DepartureIndex: This function determines if a departure successfully took off and at what point the flight entered the runway.
    - SpeedCalcFunctionFull: This function creates a log of the distance traveled, speed, and acceleration of a flight as calculated by the flight's location over time.
- CalculationFunction.py: This function uses ArrivalIndex, DepartureIndex, and SpeedCalcFunctionFull to determine the time period when each flight was using the taxiways and the distance, speed, and acceleration profiles of each flight.
- AvgSpeedFunctions.py/MultAvgSpeedFunctions.py: This larger function contains 3 smaller functions that determine the average speed of all flights in a dataset over each taxiway for the following datasets: all flights, arrivals, and departures.
- AvgWeightFunctions.py/MultAvgWeightFunctions.py: This larger function contains 3 smaller functions that determine the average weight of all flights in a dataset over each taxiway for the following datasets: all flights, arrivals, and departures.
- TotWeightFunctions.py/MultTotWeightFunctions.py: This larger function contains 3 smaller functions that determine the total weight of all flights in a dataset over each taxiway for the following datasets: all flights, arrivals, and departures.
- FreqFunctions.py/MultFreqFunctions.py: This larger function contains 3 smaller functions that determine the amount of flights that pass over each taxiway for the following datasets: all flights, arrivals, and departures.

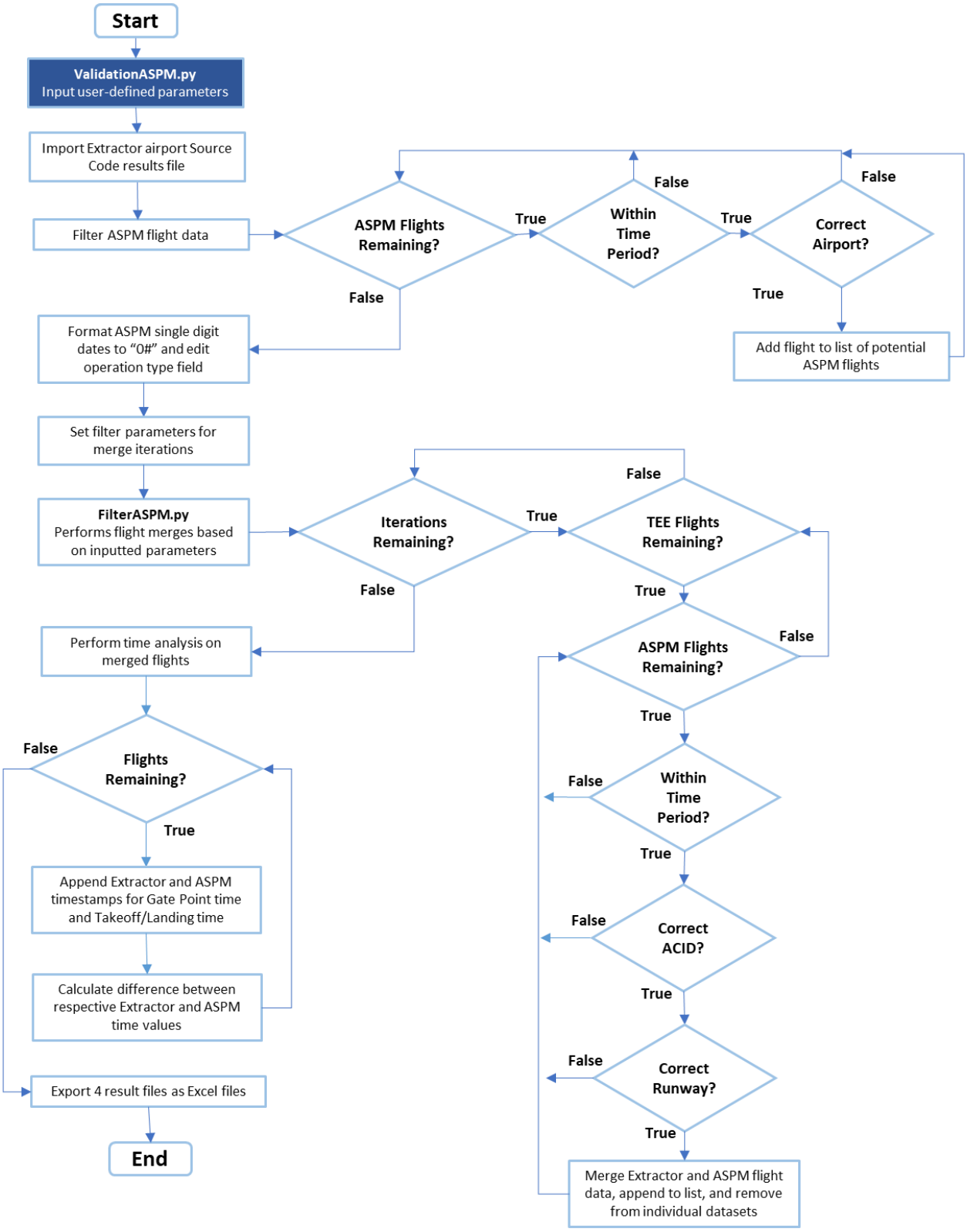
APPENDIX G: TAXI EVENT EXTRACTOR MULTIMAPS FLOW CHART



Multimaps: This program is similar to the Heatmaps program, but differs in that it splits the specified time frame into several smaller intervals where only a single parameter is analyzed. Instead of a series of different heatmaps showing different properties for the same time period, the result is a series of similar heatmaps showing the same property for concurrent time periods. These images are then combined into an .mp4.

- For supporting functions, see the ***Heatmaps.py*** page

APPENDIX H: TAXI EVENT EXTRACTOR ASPM DATA VALIDATION FLOW CHART



ValidationASPM.py: This program allows users to input a set of desired TEE and ASPM flight data based on airport, date, and time period and compare the flights to see which flights in the TEE data have a corresponding flight in ASPM. This function also sets parameters for the iteration process used to match flights, as well as performs time analysis of matched flights

FilterASPM.py: This function performs the merge iterations in ***ValidationASPM.py*** by taking the input iteration parameters and using them to determine if specific parameters of TEE and ASPM flights match

APPENDIX I: LIST OF AIRPORTS WITH AVAILABLE AIXM FILES

The table below shows a list of FAA identifiers for all 42 airports that have AIXM files available with the extractor, layouts were obtained in 2012.

ABQ	EWR	MSP
ATL	FLL	MSY
AUS	IAD	OKC
BDL	IAH	ORD
BNA	IND	PHL
BOS	JFK	PIT
BWI	LAS	SAN
CLT	LAX	SEA
CMH	LGA	SFO
CPS	MCO	SLC
DCA	MDW	TPA
DEN	MEM	AJN
DFW	MIA	HNL
DTW	MKE	HOG

APPENDIX J: LIST OF ALL RELEVANT FILES

File name	Description
Source_Code.py	Main Script
ArrivalFlightsCheck.py	Supporting Source_Code.py
DeicingAnalysis.py	
DepartureFlightsCheck.py	
ExitingRunwayArrival.py	
GateAreaFinder.py	
GateAreaFinderDeparture.py	
Layout_Loader.py	
LeavingMoment.py	
RawDataFormatter	
LoadRawFlightData.py	
SWIMDataFormatter	
SmoothTime	
Main_Processed_Data_Reader.py	
TouchdownLocationTime.py	
WaitingTimeFinder.py	
WheelsOff.py	
GateReturn.py	
Gate_And_DisabledFinder.py	
AnomaliesCheck.py	
RunwayCoordinatesExtractor.py	Generate runway coordinates file from APT.txt
APT File\APT.txt	Runway information
RunwayCoordinates.csv	Runway coordinates file
DeicingCoordinates.csv	User-defined deicing coordinates
ATL.mat	Layout file produced from AIXM data by Airport Layout Extractor
CLT.mat	
DEN.mat	
EWR.mat	
IAH.mat	
JFK.mat	
LGA.mat	
ORD.mat	
PHL.mat	
Shapefiles\ORD	Available shapefiles folder for ORD
AnimateFlightTracks.py	Animation Function
Animation Batch.py	Animation Batch
SourceCode_ValidationPlot.py	Plots for validation
GraphFlightTracks.py	Plot raw flight tracks over airport layout
Heatmaps.py	Heatmaps Main Script
HeatmapFunctions.py	Supporting Heatmaps
CalculationFunction.py	

AvgSpeedfunctions.py	
AvgWeightfunctions.py	
TotWeightfunctions.py	
FreqFunctions.py	
FilterDataFunctions.py	
Multimaps	
MultiHeatmapFunctions.py	Supporting Multimaps
MultiCalculationFunction.py	
MultiAvgSpeedfunctions.py	
MultiAvgWeightfunctions.py	
MultiTotWeightfunctions.py	
MultiFreqFunctions.py	
MultiFilterDataFunctions.py	
ValidationASPM	Allows user to setup and perform data validation against ASPM files
FilterASPM	Performs validation iterations based on parameters established in ValidationASPM
FAA-Aircraft-Char-Database-v2-201810.xlsx	FAA Database used for MTOW (Heatmaps)
Shapefile_Reader.py	Read and plot shapefiles