A METHODOLOGY FOR NOWCASTING UNSTABLE APPROACHES

by

Zhenming Wang
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Systems Engineering and Operations Research

Committee:

_____	Dr. John Shortle, Dissertation Director
_____	Dr. Lance Sherry, Dissertation Co-Director
_____	Dr. George Donohue, Committee Member
_____	Dr. Fei Li, Committee Member
_____	Dr. Ariela Sofer, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: _____	Spring Semester 2016 George Mason University Fairfax, VA

A Methodology for Nowcasting Unstable Approaches

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Zhenming Wang
Master of Science
George Mason University, 2012

Director: John Shortle, Professor
Department of Systems Engineering and Operations Research

Spring Semester 2016
George Mason University
Fairfax, VA

DEDICATION

This dissertation is dedicated to my parents.

ACKNOWLEDGEMENTS

I would like to thank all my committee members for their valuable advices and sustaining efforts in supporting my research. I would like to thank Dr. Shortle, my dissertation director, for guiding my research and providing valuable and rigorous thoughts on the mathematical nature of the problem. I would like to thank Dr. Sherry, my dissertation co-director, for identifying the key problem and providing fresh domain knowledge of the air transportation, making my research more meaningful for real-world application. I would like to thank Dr. Donohue. The deep insights from your past experience opened my minds and profoundly impacted my work, reminding me not to get lost in the big picture. I would like to thank Dr. Li for providing important and insightful feedbacks to my research with knowledge and expertise from computer science. Your help are highly appreciated.

I would like to thank SEOR department chair Prof. Ariela Sofer and associate chair Prof. Andrew Leorch for their support in my doctoral study. I would also like to thank SEOR department staff members Angel Manzo and Josefine Wiecks. Thank you all for helping me through all these years.

I would like to thank Dr. Chang and Dr. Sun for their support during my first year of graduate study.

I would like to thank all my colleagues at Center for Air Transportation Systems Research. I would like to give thanks to Akshay, Jianfeng, John, Kara, Sanja, Victor, Yimin, and all other graduated CATSR members. Thank you my friends Young, Seungwon, Houda, Anvard, Bahram, Saba, Saby, Azin, and Kevin. Wish you all success!

Finally, I want to thank my parents for their concern, support, and encouragement during my graduate study abroad. I would not have this achievement without you. I love you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

ADS-B.....Automatic Dependent Surveillance - Broadcast
AGL .....Above Ground Level
ASDE-X.....Airport Surface Detection Equipment, Model X
ATC.....Air Traffic Control
CFIT.....Controlled Flight Into Terrain
EWR.....Newark Liberty International Airport
FAA..... Federal Aviation Administration
FAF ..... Final Approach Fix
FDR.....Flight Data Recorder
FOQA..... Flight Operational Quality Assurance
FMS..... Flight Management System
FSF.....Flight Safety Foundation
GMT..... Greenwich Mean Time
IAF .....Initial Approach Fix
ICAO..... International Civil Aviation Organization
IFR .....Instrument Flight Rules
ILS..... Instrument Landing System
IMC..... Instrument Meteorological Conditions
MTOW..... Maximum Takeoff Weight
NAS.....National Airspace System
NTSB ..... National Transportation Safety Board
ORD .....Chicago O'Hare International Airport
SFO .....San Francisco International Airport
SOP .....Standard Operating Procedures
TRACON .....Terminal Radar Approach Control Facilities
UTM..... Universal Transverse Mercator Coordinate System
VFR..... Visual Flight Rules
VMC ..... Visual Meteorological Conditions

ABSTRACT

A METHODOLOGY FOR NOWCASTING UNSTABLE APPROACHES

Zhenming Wang, Ph.D.

George Mason University, 2016

Dissertation Director: Dr. John Shortle

The approach-and-landing is one of the most complex procedures in airline operations. To mitigate the potential risks in this phase of flight, airlines and regulators have published stabilized approach criteria that require the flight crew to check a set of conditions at 1000’ and 500’ above ground level (AGL). If the stabilization criteria are not met, the flight crew is required to abort the approach by executing a go-around, which adds operational costs to the flight and reduces runway efficiency.

Nowcasting unstable approaches prior to the stabilization altitudes (e.g. at 6 nm from the landing runway) could provide lead time for the flight crew to make adjustments to the trajectory being flown to avoid a potential unstable approach. Kinematic models used in the Flight Management System (FMS) to predict future aircraft state are not practical in nowcasting unstable approaches as these models do not account for events that will occur during flight progress (e.g. flap/slat and extension, Air Traffic Control (ATC) clearances, etc.).

This research develops a methodology for identifying and nowcasting unstable approach events through the analysis of massive amounts of surveillance track data for a given approach. The nowcasting algorithms are implemented in conditional probability methods and supervised learning methods. The performances of predictions are evaluated. The results indicate that at 3.5 nm, 75.3% of actual unstable approaches can be correctly nowcasted; with probability of 85.9%, a nowcast of unstable approach is correct. The implications, limitations, and future work of this research are discussed.

CHAPTER 1: INTRODUCTION

The approach-and-landing is one of the most complex procedures in airline flight operations. According to the International Civil Aviation Organization (ICAO), 47% of fatal accidents and 40% of onboard fatalities appear in the final approach and landing phase for worldwide operations (ICAO 2014). To avoid flying an unstable approach, the Standard Operating Procedures (SOPs) of airlines and regulators have established specific criteria for stabilized approaches. In general, these stabilized approach criteria require aircraft to be on the correct track (i.e., on the runway centerline and on the approach glide-path), within an appropriate range of a reference airspeed (e.g. ± 10 knots), and within an appropriate range of a required rate of descent (e.g. < 1000 ft/min). If any of these criteria is not satisfied at the stabilization altitude of 1000' AGL under Instrument Meteorological Condition (IMC) or 500' AGL under Visual Meteorological Conditions (VMC), the procedure requires the flight crews to perform a go-around to abort the approach. Failing to execute this procedure may lead to potential risk events such as controlled flight into terrain (CFIT) or a runway excursion. Due to the complex nature of the approach phase, there is a need to nowcast unstable approach events *prior to* reaching the stabilization altitude, e.g., at 10 nm, 6 nm, or 3.5 nm from runway threshold. Flight crews knowing a potential unstable approach may occur will have lead time to

check and correct the aircraft trajectory to reduce the risk. This research focuses on developing methods for nowcasting unstable approaches prior to the stabilization altitude.

1.1 Unstable Approaches and Their Risks

As one of the most complex procedures, the approach-and-landing phase serves a key function in bringing aircraft from airborne to runway landing. Flight crews must configure the aircraft appropriately for the maneuvers required by each trajectory segment to maintain a lift-generating energy state, coordinate the required trajectory to achieve the flight plan, remain within the constraints of the navigation procedure, maintain separation with other flights, and coordinate immediate and future trajectories with ATC. An example approach chart is shown in Figure 1, which shows the complexity of the approach and landing procedures in detail.

A stabilized approach is a key feature to a safe approach and landing operation. The Flight Safety Foundation (FSF) indicates that an unstable approach is a causal factor in 66% of 76 approach-and-landing accidents and serious incidents worldwide (FSF 1999). Several well-known accidents related to an unstable approach include the Asiana Airlines Flight 214 in 2013, the Turkish Airlines Flight 1951 in 2009, and the Southwest Airlines Flight 1455 in 2000.

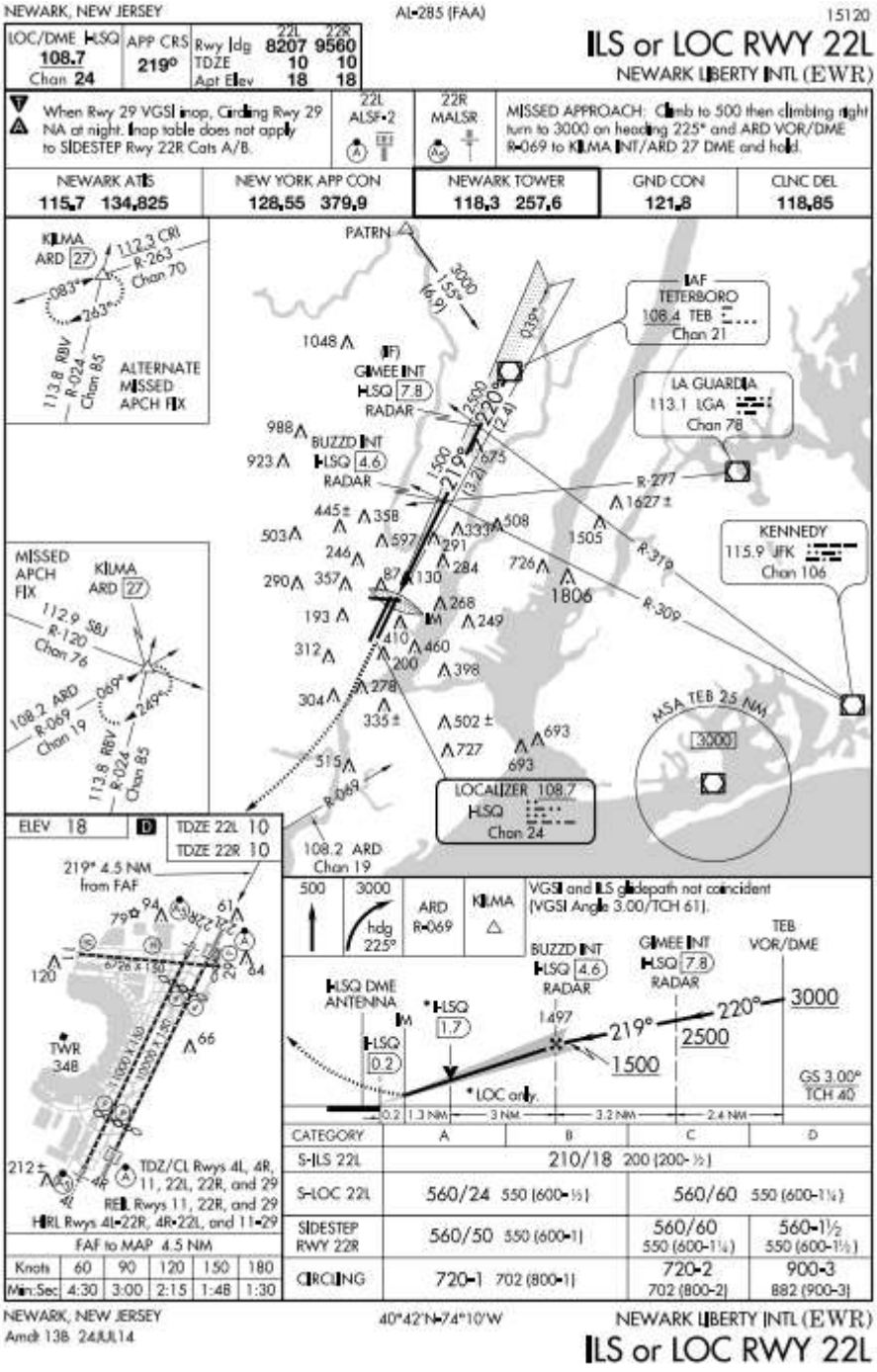

Figure 1 A Sample Approach Chart

In the Asiana Airlines Flight 214 accident, a Boeing 777-200ER crashed on final approach into San Francisco International Airport (SFO) on July 6, 2013. According to the accident report by the National Transportation Safety Board (NTSB), the approach of the flight was not stabilized and the airplane did not initiate a go-around (NTSB 2014). When the airplane reached 5 nm, it was above the desired 3-degree glide-path. This situation continued during the final approach. Asiana's SOP procedures dictates that an approach must be stabilized when an aircraft reaches 500' AGL. However, at 500', the airplane was still above the desired glide-path, and the airspeed had been decreasing rapidly to reach the proper approach speed of 137 knots. The rate of descent was about 1,200 ft/min which is significantly higher than the baseline value of 700 ft/min to maintain the desired glide-path. As the unstable approach continued, the airplane kept descending at an excessive rate, causing the airplane to fly below the glide-path. At about 200' AGL, the flight crews became aware of the situation. However, they did not initiate a go-around until the airplane was below 100', at which point the airplane did not have the performance capability to execute a go-around. This resulted in a contact with the sea wall prior to runway.

In another accident, Turkish Airlines Flight 1951 crashed into a field approximately 0.8 nm from the runway while landing at Amsterdam Schiphol Airport on February 25, 2009. The airplane flew high and fast, intercepting the glide-path from above. Also, the flight captured the runway centerline late at 5.5 nm. Due to the automated reaction triggered by a faulty radio altimeter, the engine power was decreased to idle during approach. The flight crew did not realize the problem until it was too late to

recover from the low speed. The aircraft stalled and crashed. In this accident, flying an unstable approach was a contributing factor.

For the runway overrun accident of Southwest Airlines Flight 1455 on March 5, 2000, the leading factors included the steep glide-path angle and the high approach speed.

These accidents illustrate the importance of a stabilized approach. Unstable behaviors such as steep glide-path angle, short glide-path interception, high approach speed and improper speed change can cause unstable approaches and lead to potential landing risks. It is necessary to have a methodology to identify these potential risks and alert the flight crews before they can occur.

1.2 Stabilized Approach Criteria

To help flight crews establish stabilized approaches, it is necessary to specify a list of requirements that are clearly defined as standard operation procedures to follow. In general, the concept of a stabilized approach is characterized by flying on the correct track (i.e., runway centerline and glide-path), within an appropriate range of reference airspeed, and within an appropriate range of required rate of descent. The criteria in SOPs proposed by different airlines and regulators all follow this basic concept but are different in the details. The following is an example of the stabilized approach criteria given by the FAA (FAA 2003). To be stabilized, all of the following criteria should be maintained:

- The aircraft is flying on the correct track (lateral & vertical).
- The aircraft is in the proper landing configuration.
- The air speed is within the acceptable range as specified in the operating manual.

- The rate of descent is no greater than 1000 feet per minute (fpm).
- After intercepting the glide-path intercept, the pilot flying requires no more than normal bracketing corrections to maintain the correct track and desired profile to land within the touchdown zone, e.g. bank angle < 30°, ± 300 ft/min deviation from the target rate of descent.
- The power setting is appropriate for the landing configuration selected, and is within the permissible power range.

Flights are required to meet these criteria by 1000’ in IMC or 500’ in VMC with all appropriate briefings and checklists accomplished. An unstable approach is characterized by a failure to meet one or more of the previous requirements. An aircraft not meeting all the criteria by the stabilization altitude is required to abort the landing and execute the published missed approach procedure.

A typical vertical profile for the approach is shown in Figure 2, with the stabilization check points at 1000’/500’ shown. According to the stabilized approach criteria, a landing aircraft should be stabilized by reaching 1000’ AGL (at about 3 nm from the runway threshold for a 3-degree glide-path) under IMC or 500’ AGL (at about 1.5 nm) under VMC; otherwise a go-around is mandated.

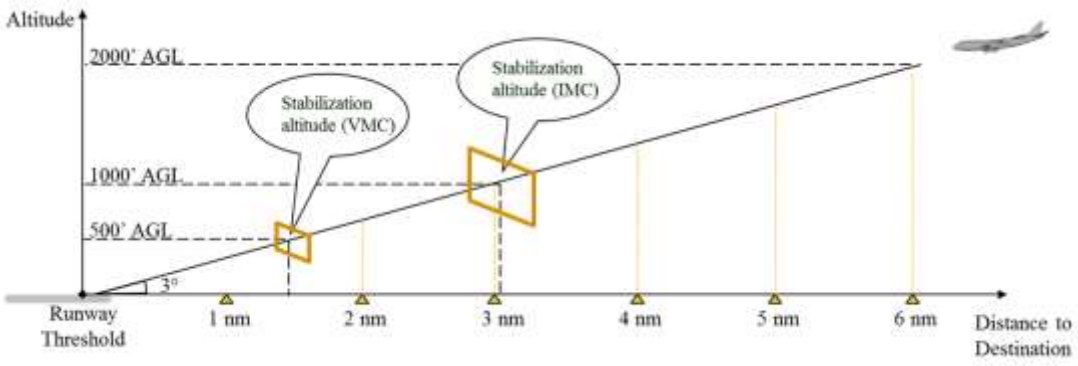

Figure 2 A Typical Approach Profile

1.3 Concept of Operations

During approach, the flight crews are required to monitor a rapidly evolving situation during the approach phase and to make split second decisions based on a large number of factors. Given the complexity in the coordination of trajectory and aircraft energy, checking the stabilization criteria at 1000'/500' is too late. Alerting the flight crew of a potential unstable approach *before* the aircraft reaches the stabilization altitude can give the crew the necessary lead time to check and correct the aircraft trajectory to avoid an unstable approach. However, currently used indicators in the FMS are for monitoring the *current* state of the flight. Kinematic models, although used in the FMS to predict the future aircraft state, are not practical in nowcasting unstable approaches as these models cannot account for various discrete events that occur during flight progress, for example, flap/slat and ATC clearances. Therefore, there is a need for developing a nowcasting methodology to identify potential unstable approaches.

The proposed solution in this research is to use historical flight track data, weather data, navigation procedure data, and aircraft performance data to build a prediction model

for nowcasting the risk of unstable approaches before the stabilization altitude. This concept of operations is demonstrated in Figure 3. The model training phase is offline using historical data. The output of the nowcast algorithm is the prediction model with trained parameters, which is specific to each landing runway. The models with trained parameters of target runways are uploaded to the avionics. With the real-time sensor data collected onboard, the nowcasting system calculates the probability of experiencing unstable approach events after 1000'/500' AGL. If the nowcast result is positive (i.e. unstable), an indicator on the Primary Flight Display (PFD) is activated with the corresponding information of a potential unstable approach. Flight crews with this message on the PFD are expected to take actions to adjust the aircraft trajectory accordingly to avoid a potential unstable approach.

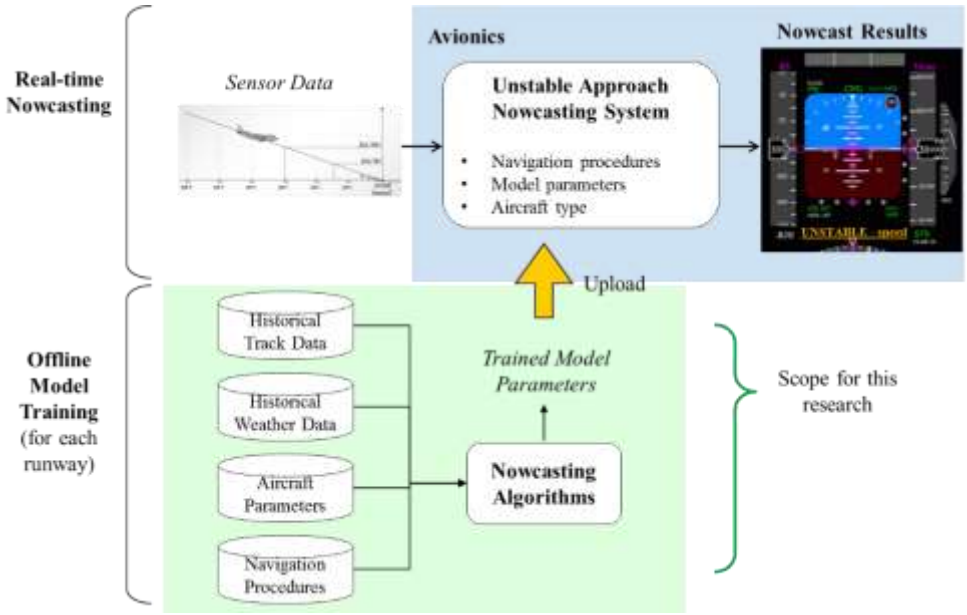

Figure 3 Concept of Operations

1.4 Research Objective

The objective of this research is to develop a methodology for nowcasting unstable approaches *prior to* the stabilization altitude. Key questions this research will address are listed below:

- How can unstable approaches be defined and detected with publicly available data sets to best reflect the stabilized approach criteria?
- What methods can be applied for nowcasting unstable approach events?
- What features can be derived from the data and used for model inputs?
- How well do the nowcasting models perform?

This research focuses on a proof of concept by using surveillance track data for analysis. This approach is "worst-case" in the sense that there are better (but not publicly available) data sets with higher fidelity and more fields (e.g., Flight Data Recorder (FDR) data) that would be expected to yield better results based on the proposed.

This research focuses on the module of off-line model training. The future design and integration of the nowcasting system on the FMS avionics is important but beyond the scope of this research.

1.5 Gaps in the Literature and Unique Contributions

In summary, there has not been an existing methodology for nowcasting unstable approach events. Related literature on this topic includes trajectory prediction, anomalous flight detection, and approach performance analysis. Trajectory prediction methods are based on point-mass models or kinematic models, which do not consider external factors such as ATC instructions. Anomaly detection methods can detect potential unstable approaches but real-time identification of actual unstable approaches cannot be achieved.

Methods for approach performance analysis assess the risks of related unstable events but do not directly apply to the task of nowcasting.

Given these gaps in nowcasting unstable approach events, the unique contributions of this research are summarized below:

- Development of a method to identify unstable approaches from integrated data of flight tracks, wind conditions, navigation procedures, and aircraft parameters,
- Development of a nowcasting method via a conditional probability method,
- Development of a nowcasting method via supervised learning,
- Evaluation and comparison of the nowcasting methods.

1.6 Summary of the Methodology

This research uses archived historical data to develop models for predicting unstable approach events at locations prior to the stabilization altitude. Figure 4 summarizes the framework of the research. The first step is to process and integrate historical flight tracks, weather conditions, navigation procedures, and aircraft parameters. From these processed and integrated data, aircraft state variables at key locations (e.g. 6 nm, 1000', 500', and landing) are calculated. Stabilized approach criteria are applied to identify unstable approaches from this integrated data set.

Then, to nowcast these detected unstable approach events, two methods are applied. The first model is based on historically observed probabilities of unstable approaches under different conditions. If the probability of an unstable approach under a

given set of conditions is significantly higher than the overall aggregate probability, then an alert of a potential unstable approach is sent to the flight crew. This method does not perform well with a limited amount of historical data. The second method uses supervised learning. The method starts with selecting features and training the parameters of the models to minimize the cost function. To evaluate the performance of the nowcasting models, several measures are studied to quantify their prediction performances.

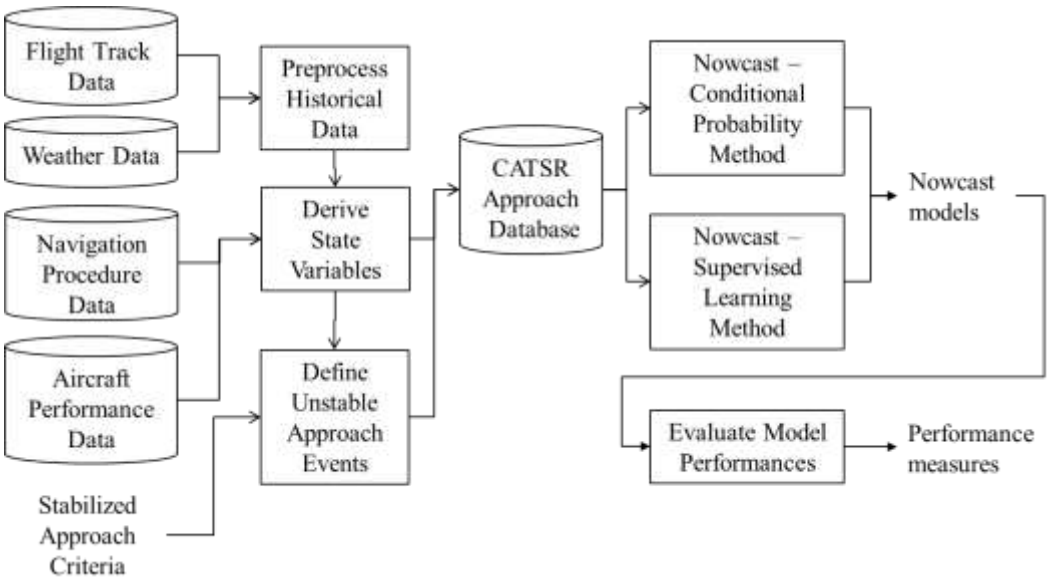

Figure 4 Methodology Overview

1.7 Summary of Results

A case study is conducted at Newwark International Airport (EWR) Runway 22L. First, the frequencies of unstable approach events are observed. Some key observations are summarized below:

- 47.3% of landing flights experienced unstable approaches after reaching 1000’ AGL. This percentage drops to 17.5% for 500’ AGL.
- 26.8% of landing flights experienced an excessive decrease in airspeed from 1000’ AGL to runway threshold. This percentage drops to 4.1% for 500’ AGL.
- 19.5% of landing flights acquired the runway centerline from the right side after 1000’ AGL. This percentage drops to 8.5% for 500’ AGL.

In terms of nowcasting unstable approaches, the best performance achieved is by the supervised learning method with all developed features. The nowcasting performances for unstable approaches after 1000’ at 10 nm, 6 nm, and 3.5 nm are summarized below.

- At 10 nm, 53.9% of unstable approaches can be correctly nowcasted; 62.5% of nowcasts of an unstable approach are correct.
- At 6 nm, 64.7% of unstable approaches can be correctly nowcasted; 78.4% of nowcasts of an unstable approach are correct.
- At 3.5 nm, 75.3% of actual unstable approaches can be correctly nowcasted; 85.9% of nowcasts of an unstable approach are correct.

Generally the nowcasting performance improves as the nowcasting location gets closer to the stabilization altitudes. Nowcasting by the conditional probability method is

less powerful. Larger amount of data sets will help in improving the performance of the conditional probability method. The supervised learning method with fewer features also lowers the performance measures, indicating that adding appropriate features may further improve the current model.

By studying the sensitivity of features in the trained models, key features are identified which have a large impact on unstable approaches. For example, at 6 nm, these features include lateral deviation, airspeed, and crosswind speed.

1.8 Potential Applications of This Research

This research proposes a methodology which is a starting point for further research on potential applications of the nowcasting system. This system is to be integrated in the FMS. The preliminary concept of operations is for the pilot monitoring to monitor the nowcast outputs of this system as the aircraft starts the approach process. If there is a nowcast message indicating a potential unstable event, the pilot will notify the pilot flying to consider an adjustment of the current trajectory to decrease the probability of an unstable approach.

Many tasks should be completed before this system can be realized. First, system validations are needed. One way of testing the system is to embed the nowcast system into a flight simulator. The purpose is to test the hypothesis that fewer unstable approaches are experienced by a flight crew using the nowcasting system compared to flight crews not using the system. Even if the performance of the system is validated, it will be necessary to carefully study the interface design with the current FMS. Many

human-in-the-loop experiments need to be conducted before the realization of this nowcasting system.

Finally, the concept of this research needs to be approved and certified by related stakeholders such as regulators and aircraft manufacturers. Although the system is expected to effectively reduce the chance of an unstable approach for a landing flight, the potential ripple effect must be considered. Introducing a solution to an existing problem may lead to other unexpected problems. For example, the trade-off between distracting the flight crews and the need for monitoring the nowcast results should be considered. The ultimate goal is to improve the *overall* safety margin of the landing flights during the approach phase.

CHAPTER 2: LITERATURE REVIEW

This chapter summarizes existing literature related to this research. The first section covers an overview of stabilized approaches including concepts and definitions as well as field studies to estimate the fraction of approaches that are unstable. The next sections describe literature relevant to identifying and predicting unstable approaches including algorithms for aircraft trajectory prediction, anomaly detection, and approach performance analysis. These papers use flight track data to study different perspectives of unstable events during the approach. Finally, the limitations of these methods for nowcasting unstable approaches are discussed.

2.1 Stabilized Approach

The FAA Safety Team defines the stabilized approach concept as follows (FAA 2006): "*A stabilized approach is one in which the pilot establishes and maintains a constant angle glide- path towards a predetermined point on the landing runway.*" For flight crews, a stabilized approach requires that the airplane descend on final approach at a constant rate and airspeed, and travel in a straight line toward a fixed point on the ground ahead.

In the flying lessons by Turner (2011), the concept of a stabilized approach is elaborated with instructions prior to and after the final approach fix (FAF). Prior to reaching the FAF, the aircraft is put into landing configuration (gear down and flaps set),

and airspeed is reduced to V_{REF} or some target just above V_{REF}. At the start of the descent, the pilot should adjust attitude and power to establish a descent while maintaining airspeed, flying in this configuration and attitude all the way to touchdown.

The definitions above for a stabilized approach are from the technical perspective of flight crews. A more rigorous way to define the concept is by setting a set of well-defined criteria. Airlines and regulators have specified minimum acceptable criteria for stabilized approaches. The criteria can be different in the details but generally share the same spirit. An example set of stabilized approach criteria proposed by the Flight Safety Foundation (FSF 2010) is summarized below:

- The aircraft is on the correct flight path.
- Only small changes in heading/pitch are required to maintain the flight path.
- The aircraft speed is not more than V_{REF} + 20 knots indicated airspeed and not less than V_{REF}.
- The aircraft is in the correct landing configurations.
- The rate of descent is no greater than 1,000 fpm.
- The power setting is appropriate.
- All briefings and checklists have been conducted.
- For an ILS approach, the aircraft must capture the glideslope and localizer.
- The stabilized approach should be achieved after reaching 1,000 feet above airport elevation in IMC, 500 feet above airport elevation in VMC. An approach failing to meet the criteria requires an immediate go-around.

If any of the criteria is not met by the stabilization altitude, the approach to land should be discontinued and a go-around is executed.

The main goals of the stabilized approach concept include the following (Turner 2011):

- Predicting the performance of different aircraft using the same technique,
- Increasing situational awareness,
- Detecting and correcting glide-path deviations,
- Increasing the ability to establish crosswind correction, and
- Landing in the touchdown zone at the proper speed.

Currently, there have not been agreements achieved on the proportion of unstable approaches. A survey (DGAC 2006) suggests that 3% of approaches are unstable. By analyzing the flight data collected from FSF's corporate flight operational quality assurance program, Darby (2010) indicates that the incidence of unstable approaches is 4.5% in 2009, down from 12.8% in 2006. During line-operation safety audits, observers on the flight decks of 4,532 commercial flights between 2002 and 2006 found that 5% of approaches were unstable (Merritt 2006). It was also noted that only 5% of these observed unstable approaches initiated a go-around.

Possible causal factors leading to unstable approaches include fatigue, pressure of the flight schedule, improper ATC instructions, a late runway change, a late takeover from the autopilot, inadequate awareness of wind conditions, etc. (Turner 2011). Moriarty et al. (2014) further analyze the potential causes for unstable approaches from a system perspective. The authors indicate that the major cause of unstable approaches lies

in the mismatched goals from different stakeholders. Rather than explain unstable approaches as being solely due to the actions of the pilots, the study identifies a collection of factors *external* to the flight deck that have significant impacts on the safety of an approach. Human decision-making in a high-workload situation tends to continue unstable approaches to landing, which further increases the risk. A published speed profile is recommended to unify the goals of pilots and air traffic controllers.

An effective method to detect unstable approaches is to examine flight track data at different locations, e.g. speed, position, etc. In recent years, the availability of surveillance track data has provided such convenience. For example, Wang et al. (2015) study the statistics of unstable events using surveillance track data. It showed that 27.8% of the approaches exhibited more than 10 knots change in groundspeed after reaching 1000’ AGL.

The following sections will cover some existing studies that apply flight track data to predict aircraft trajectories, detect anomalous flights, identify specific types of unstable events, and conduct stochastic simulations. All these methods are related to the research in this dissertation in two ways:

1. The methods use flight track data.
2. The methods focus on risk events that are the causes or results of unstable approaches.

2.2 Aircraft Trajectory Prediction

Ideally, if the future trajectory of an aircraft can be predicted to a satisfactory accuracy level based on the current trajectory, then unstable approach events can be

successfully predicted. However, the mechanism of prediction (i.e. based on aircraft performance models) becomes the major limit for their use in nowcasting unstable approaches.

Existing prediction methods include point-mass model, kinematic model, and others. A large amount of literature exists on the topic of trajectory prediction (Musialek et al. 2010). This section introduces a few of them.

A basic idea is to apply aircraft performance models to predict aircraft trajectories. The point-mass model applies aircraft motion equations. The inputs of the control system are thrust, flight path angle, and bank angle. The outputs of the model are the aircraft states containing the position of the aircraft mass point, speed vector, heading, and mass. Using point-mass model to predict aircraft trajectory is a major approach. For example, Rodriguez et al (2007) apply point-mass model for a 4D trajectory guidance model for descent phase of flight. Thipphavong et al. (2013) apply point-mass models to predict climbing trajectories using real-time radar track update with an adaptive weight algorithm which dynamically adjusts modeled aircraft weights on a per-flight basis to improve the predictions. The look-ahead time can be up to 12 minutes with the altitude profiles tested. With an update rate of 12 seconds, the quality of track data is a major limiting factor on the amount of improvement that can be achieved with the algorithm. Higher quality track data such as Automatic Dependent Surveillance - Broadcast (ADS-B) is expected to enhance the performance.

In kinematic models, only position and time rate of changes are modeled. The aircraft performance parameters are not included in the model. Uncertainties are

introduced by adding pertubations following assumed stochastic distributions, e.g. human pilot flying parameters (roll rate, rate of descent, etc.). Literatures using kinematic models include Miquel et al. (2006) and McGovern et al. (2007).

The major limitation of these trajectory prediction methods is that they do not account for external factors which can affect the approach trajectory. Examples of such factors include air traffic control (ATC) clearances and changes in aircraft configurations (flap/slat extension). These factors are discrete in nature and do not directly appear in the models. Some regulated approach patterns such as short curve-in landing also belong to those external factors. Due to the complex nature of approach and landing procedures, the trajectory of an aircraft cannot be purely explained by a set of aircraft movement equations.

There are other models for trajectory prediction. For example, Richards (2002) applies a neural-network-based fuzzy inference system for improving the landing signal officer's decision making on an aircraft carrier. The goal is to project 2 seconds ahead of the current flight position. This method outperforms the polynomial extrapolation in predicting aircraft positions. Limitation of using this model in predicting unstable approaches is the short look-ahead time. The time from beginning of descent to final touchdown can take several minutes.

Table 1 Summary of Aircraft Trajectory Literatures

Method	**Literature**	**Description**
Point-mass model	Rodriguez et al. 2007, Thipphavong et al. 2013, etc.	Apply aircraft performance parameters toward realistic modeling of flight.

Kinematic model	Miquel et al. 2006, McGovern et al. 2007, etc.	Focus only on position, course, speed, and their time rate of changes.
Other model	Richards et al. 2002	Apply neural network based fuzzy inference system to project 2 seconds ahead of current position.

Table 1 summarizes several papers in the area of trajectory prediction. In summary, existing models for aircraft trajectory prediction have the drawback of missing the external uncertain factors in predicting unstable approach events. These external factors include ATC instructions, onboard configurations, or specific approach patterns regulated in the approach diagram. Therefore, it is not appropriate to use these models to predict unstable approaches at an early stage.

2.3 Anomalous Flights Detection

Anomaly detection is an active research area in data mining (Chandola et al. 2009). It refers to the process of detecting inconsistent observations from the majority of a dataset. There are three major methods for addressing anomaly detection problems – the statistical method, the classification method, and the clustering method.

The statistical method assumes statistical distributions of data. Whether a data instance is anomalous depends on how well it fits the distribution. Due to the complexity and dependencies of unknown data, this method is not always applicable. The classification method is based on supervised learning. In the training phase, the classes are separated with boundaries created from learning the labeled data. Then, the trained classifier is used to assign test data into one of the specified classes. The clustering

method identifies groups of data so that data in one group are more similar to each other than those in other groups. Anomalies are detected which do not belong to any group.

The air transportation system provides a rich field for anomaly detection. For example, the Flight Operational Quality Assurance (FOQA) program is implemented in many US airlines, which contains the main source of onboard recorded data (FAA 2004). The main purpose of FOQA is to improve airline operations and safety by analyzing detailed flight data that are daily recorded. Many flight parameters are recorded with different sampling rates. The FOQA data is evaluated through exceedance analysis using statistical method. The exceedance analysis identifies instances in which parameters exceed predefined limits under different conditions (Treder et al. 2004). A flight with a particular parameter value exceeding the predefined limit is labeled as anomaly.

Li et al. (2011) have used cluster analysis techniques to develop an unsupervised method to detect anomalous flights based on continuous flight parameters. Digital Flight Data Recorder (DFDR) is used as the data source for this research. The work does not need a predefined threshold for flight parameters and detects anomalous flights differing from the majority of flights by considering all available flight parameters. The time series of multiple flight parameter data are converted to a high dimensional vector. Cluster analysis of these vectors is conducted to identify anomalous flights which do not belong to any cluster. Anomalies are then reviewed by domain experts to determine their significance. Detected anomalous behaviors include high and low energy states, unusual pitch excursions, and abnormal flap settings.

Matthews et al. (2013) apply classification method for anomaly detection in FOQA data using algorithms of Multiple Kernel Anomaly Detection (MKAD). The work demonstrates the anomaly detection process on a large commercial aviation data set. Discoveries of anomalous flight behaviors include fleet level anomalies and flight level anomalies.

Onboard measured data are not publicly available. Researchers also use external flight track data to do anomalous flight detection. For example, Matthews et al. (2014) use surveillance track data to identify operationally significant flights. The work takes the features of latitude, longitude, altitude and the separation to closest aircraft for each trajectory. After detection of anomalous flights, subject matter experts are needed to determine which of them are significant. Additional supportive data such as ATC voice data is used to help identify the issues. Detected anomalous events with operational significance include runway switches, go-arounds, S-turns, too fast, too high, and intercept angle excessive etc.

However, anomaly detection methods are not practical in nowcasting unstable approaches for several reasons:

- There is a significant need for manual inspection after the fact. The methods cannot be used for real-time prediction.
- Only a small portion of detected anomalies turn out to be significant.

2.4 Approach Performance Analysis Methods

Other methods dealing with risk analysis in approach processes include statistical analysis, stochastic simulation, and heuristic methods. These methods do not directly

apply to the task of nowcasting, but provide useful information for integrating the data sets, characterizing aircraft state variables, and defining specific risk events (e.g. go-around).

2.4.1 Statistical Analysis

Statistical properties of aircraft trajectories are the basis for track modeling and approach simulations. State variables such as lateral/vertical position, speed, and separation are fitted with probability distributions and treated as random variables. Of particular importance are the tail behaviors of the distributions which correspond to the extreme values of the flight track features. For example, an extraordinary small separation time between two successive arrivals or an extremely large lateral deviation from a predefined glide-path lead to higher probability of risks. By analyzing the track data, the distributions of the key state variables can be fitted and the probability of extreme behaviors can be estimated.

A key feature of arrival flight tracks is the position. Lateral and vertical position distributions can vary at different distances from the runway threshold along the glide-path for different approach types. The deviation and dispersion of flight tracks from the prescribed flight path can cause unstable approaches and lead to potential safety risk issues.

Hall et al. (2008) quantified the flight technical error of landing aircraft along the final approach segment under IMC at Lambert-St. Louis International Airport (STL) using Airport Surface Detection Equipment, Model X (ASDE-X) data and other sources of surveillance track data. Data are analyzed in terms of the distribution of the lateral and

vertical deviations from the ideal glide-path as a function of distance from the runway. Mean and standard deviations are summarized and compared to the ICAO ILS specification tolerance, proved to be tighter than prescribed standards.

Zhang et al. (2010) investigated the cross-sectional position distributions of arrival flight tracks along glide-path at Hartsfield–Jackson Atlanta International Airport and O'Hare International Airport. The paper fits several types of probability density functions to lateral and vertical position distributions, among which, the normal distribution generally provides a good fit. The results show similar distributions between two airports under IMC for both lateral and vertical positions. And the results are similar to results at STL in Hall et al. (2008), which has provided some indication that the distributions observed under IMC may be generalized and extrapolated to other airports.

Another important statistical feature of arrival tracks is the separation between consecutive arrivals. Separation values can be obtained by measuring time intervals between landing flights to the same runway at fixed distances from the runway threshold. For example, Haynie (2002) and Xie (2005) study the time separation by direct observation of aircraft operations data. Andres et al. (2001) and Rakas et al. (2005) use radar data to study the statistical distributions of separations. Jeddi et al. (2006) use multilateration data to study the landing operations at Detroit Metropolitan Wayne County airport. Algorithms are developed to extract landing process variables including Landing Time Intervals (LTI), Inter-Arrival Distance (IAD) and Runway Occupancy Time (ROT) from the data. LTI and IAD are fit by Erlang distributions, and ROT is fit by a beta distribution. Shortle et al. (2010) study the probability distributions for time-

separations of landing aircraft at Detroit Metropolitan Wayne County Airport using multilateration data. The tail behaviors of separation distribution are studied. The left tail of the separation distribution decays faster than a normal distribution, which is positive from a safety perspective.

2.4.2 Stochastic Simulation

Stochastic simulation methods can be applied to quantitatively study the probability of some risk events that are the causes or results of unstable approaches. These methods, based on the statistical analysis, assume specific probability density functions for key features of flight tracks. Well-defined dependency relationships between variables are also needed.

Simulation of the approach and landing process can be conducted by generating hypothetical flight tracks with statistical features derived from surveillance track data. The safety metrics are expressed in probabilities of specific risk events which may lead to or result from unstable approaches.

An example application of simulation methods is in studying the separation violations. For example, Jeddi et al. (2007) have analyzed the optimal level of landing operations on a single independent runway with fitted distributions for ROT and LTI obtained from previous research. The two types of risks with landing procedures considered are the wake vortex encounter and the simultaneous runway occupancy. The modeling of the landing operations assumes that the go-around procedures are strictly enforced when separation distances fall below a specified value. Models are developed to maximize the number of successful landing operations while mitigating the risk factors.

In Shortle et al. (2013), an interaction model between go-arounds and runway throughput is further proposed for modeling of runway landing operations with go-around process as an extension to the previous work. The stochastic model is based on the updated go-around rates (Sherry et al. 2013).

Another application of stochastic simulation methods are for wake vortex encounter. Wake vortices are a major potential safety hazard for landing aircraft. Minimum separation requirements between landing aircraft must be ensured. Shortle et al. (2007) have developed a probabilistic method to assess the potential wake encounter probability for landing aircraft using different wake models and flight tracks data. The method to estimate the probability of a wake encounter is via a hybrid simulation method. The method directly uses real flight track data, while simulating the wake behavior using wake-evolution models. The frequency of wake events were estimated using one week of sample flight tracks. The results show how the probabilities of the risk events depend on atmospheric conditions and other factors. Based on this work, Wang et al (2012) conducted sensitivity analysis of potential wake encounters to stochastic flight track parameters, atmospheric parameters and parallel runway parameters. Aircraft locations were simulated using hypothetical probability distributions in dimensions of lateral position, vertical position, and separation time based on previous studies of the statistical characteristics of these variables. Wake behaviors were simulated using wake models with input parameters of airspeed, weight, and wingspan of aircraft, and atmospheric parameters etc. By integrating these two components together, it is possible to estimate the probability of a potential wake encounter. The results showed that the mean and

variance of separation time and the variance in vertical position have big impacts on wake encounter.

2.4.3 Heuristic Methods

Heuristic methods, or rule-based method, can be developed according to specific rules to detect unstable risk events. For example, as a consequence event of unstable approaches, go-around is a standardized flight procedure to abort the approach when stabilized approach criteria are not met. Sherry et al. (2013) developed a heuristic method to study statistics and potential causes of go-arounds at Chicago O'Hare International Airport (ORD). The go-arounds are identified from surveillance track data by using the key feature of cumulative turning angles. Potential leading factors are identified from voluntary pilot and air traffic controller reports, including airplane issues, traffic separation issues, weather issues, runway availability issues, and flightcrew-ATC interaction issues.

Rule-based methods can be good at detecting and discovering some risk events in a heuristic way. But these methods is not applicable in establishing relations between factors and results which is the key to the task of this research.

2.5 Summary of Literature Review

In summary, current literature on stabilized approach analysis using flight track data generally cover the following aspects: trajectory prediction, anomaly detection, and approach performance analysis. Their limitations or contributions to the task of this research are summarized in Table 2.

Table 2 Summary of Literatures

Method	Description	Contribution/limitation for nowcasting
Systematic overview, survey	Obtained statistics of unstable approach events. Analyzed potential causes for unstable approaches.	Provided concept, definition, reference statistics and potential causes for unstable approach events.
Trajectory Prediction	Applied point-mass or kinematic models to predict trajectories.	External factors were not included in the model; Limited look-ahead time and accuracy level.
Anomaly detection	Detect anomalous flights which experienced potential unstable approaches.	Need manual inspection of significance afterwards, not for real-time.
Approach Performance Analysis	Detect, analyze and simulate approach performances.	Provide information for integrating the data sets, characterizing aircraft state variables, and defining specific risk events.

The major gap for research is in developing methods to predict unstable approach events at an earlier stage before stabilization heights, for example, at 3.5 nm, 6 nm, or 10 nm prior to the runway threshold. Also, there is a need to study what types of features and unstable events can be identified using pubicly available data such as surveillance track data, since the onboard flight data are not widely available. A methodology is needed for deriving and analyzing key information contained in flight tracks and other related information sources to effectively predict unstable approaches given the state of aircraft before the stabilization altitude.

Specifically, to fill the gap for nowcasting unstable approaches, this research focuses on:

- Developing a methodology for identifying unstable approach events by processing and integrating publicly available surveillance track data, weather data, aircraft performance data, and navigation procedures.
- Developing a methodology for nowcasting unstable approach events, including a conditional probability method and a supervised learning method.

With the developed nowcasting models, the goal is to provide flight crews or other stakeholders with real-time unstable approach annunciations for their decision making before reaching 1000’/500’ AGL.

CHAPTER 3: METHODOLOGY

This chapter describes the methodology for nowcasting unstable approaches. The general approach is to use archived historical data to first obtain key state variables and identify unstable approaches, and then to develop models for predicting unstable approach events using the derived information. This methodology includes the following steps: obtaining and preprocessing data sources, deriving state variables, detecting unstable approach events, developing nowcasting models, and evaluating the performances of developed models. A methodology overview is presented in Figure 5.

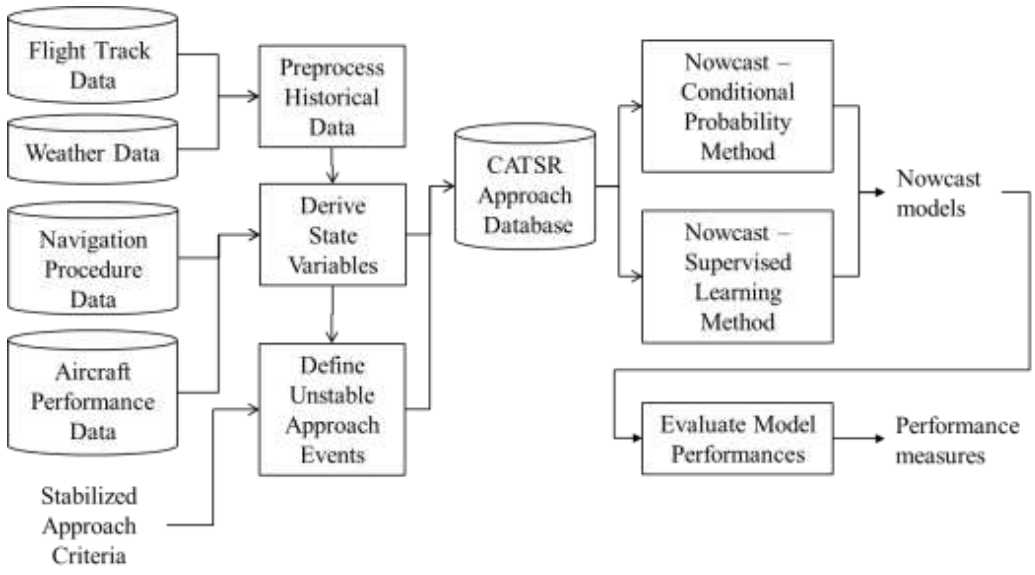

Figure 5 Methodology Overview

3.1 Data Sources

Four sources of data are used in this research. The primary source is surveillance track data from Terminal Radar Approach Control Facilities (TRACON), which contain the key information of landing aircraft trajectories. Historical weather data are also obtained for wind information and airspeed derivation. Furthermore, navigation procedures and an aircraft performance database are included. Preprocessing and integrating these data sets becomes the first step in nowcasting unstable approaches.

3.1.1 Flight Track Data

Flight track information is contained in many sources, for example, FDR data, FOQA data, ADS-B, and surveillance radar, etc. Some of these data sources contain additional information such as aircraft configuration. However, not all of the data sources are publicly available. The main requirement for the methodology in this research is that the flight data contain the aircraft type as well as a set of track points consisting of time, latitude, longitude, and altitude. Since the research focuses on evaluating the *feasibility* of nowcasting unstable approaches using flight tracks, the fidelity of the dataset is not a primary concern, although higher fidelity and additional fields can help in improving the prediction performance.

An example of such a data source is the surveillance track data from the N90 TRACON. The raw format of this data contains more than ten fields. Each row is a record of a radar hit. Key fields are extracted for basic flight track information, including the track index, aircraft type, seconds past midnight (GMT), latitude, longitude and altitude (Table 3). The track index defines each flight track with a unique code. The ICAO code for the aircraft used by each flight is stored in the aircraft type field. The time

field contains the cumulative number of seconds past midnight (GMT). The time steps can be in integer or fractional values in seconds. For surveillance radar, the sampling rate is every 4 to 5 seconds. The longitude and latitude provides the horizontal position of an aircraft at each time stamp. These values are in decimals and the precision level is to the fifth digit after the decimal point. For this dataset, the altitude readings are in numbers of flight levels, i.e., multiples of 100 feet. The integer readings in this column indicate that the precision level of the altitude information is no better than 100 feet.

Table 3 Summary of Key Fields in Surveillance Track Data

Field	**Description**	**Sample value**
Track index	Uniquely identify a track	20070111000339N902242BTA2087
Aircraft type	Type of aircraft	A310
Seconds past midnight (GMT)	Time in seconds past midnight (GMT)	297
Latitude	Latitude in decimal	40.70711
Longitude	Longitude in decimal	-74.14420
Altitude	Altitude in 100 ft.	15

Some preprocessing procedures are needed to clean up and standardized the data. The implementation of these steps will be introduced in Section 3.2.

3.1.2 Weather Data

Historical weather conditions are obtained from the METAR database, in which the raw measures are from the National Oceanic and Atmospheric Administration. The

available fields include temperature, wind speed, wind direction, gust, dew point, humidity, pressure, visibility etc. The key information used in this research is the wind condition. The METAR data is sampled hourly. The preprocessing of wind data in introduced in Section 3.2.

3.1.3 Airport Data and Navigation Procedures

To evaluate how well the landing tracks follow a prescribed path, the basic navigational information of a landing runway is needed, e.g., runway centerline heading, glide path angle, and the position of the FAF. Airport and runway information is obtained from the National Flight Data Center (NFDC) of the FAA, which is the official repository responsible for the collection, validation and quality control of aeronautical information in the NAS. For this research, the major fields obtained include the name, latitude, longitude, elevation, and runway information of the airport. For each runway, details are collected including the runway name, true alignment, glide path angle, threshold crossing height (above ground level), latitude, longitude, and elevation of the runway threshold. Table 4 summarizes the collected airport and runway data.

Table 4 Airport and Runway Data

Field	Description	Sample value
Airport name	Three-letter airport identifier	EWR
Airport location	Latitude and longitude of the center of airport	latitude 40.639751, longitude -73.778925
Airport elevation	Elevation of airport above mean sea level	17.6 ft.

Runway name	Name of runway	13R
True alignment	Runway heading angle, north = 0, clockwise	121 degrees
Visual glide-path angle	Visual glide-path angle	3 degrees
Runway elevation	Elevation of runway threshold	12.5 ft.
Threshold crossing height	The prescribed crossing height at runway threshold for landing aircraft, above ground level	73 ft.
Runway threshold location	Latitude and longitude of runway threshold	latitude 40.645339, longitude -73.810057

In addition to runway parameters, approach charts are also studied to extract the approach procedure. The approach chart is specific for a given approach type and landing runway. Figure 6 shows an example approach chart specifying the approach procedures for ILS approaches to Runway 22L at Newark Liberty International Airport (EWR). Key locations such as the initial approach fix are specified. Further extracting procedures are discussed in Section 3.2.

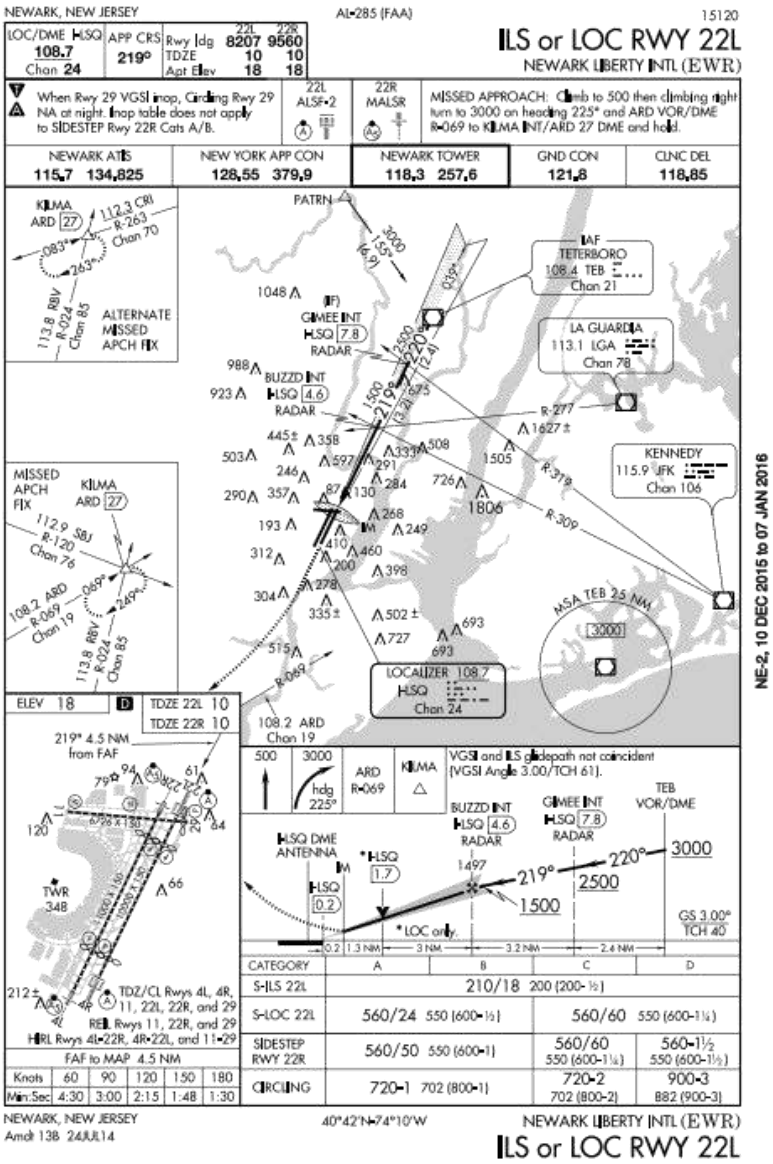

Figure 6 Sample Approach Chart

3.1.4 Aircraft Performance Data

Aircraft type information is contained in the flight track data. Aircraft parameters of a landing flight can provide useful information. The parameters of each aircraft model can be obtained from the Base of Aircraft Data (BADA), which is developed and

maintained by EUROCONTROL (2012) through active cooperation with aircraft manufacturers and airlines. For this study, the key parameters used are the aircraft type name, weight class, and maximum takeoff weight (MTOW). Five weight classes are defined based on the maximum takeoff weight: Super, Heavy, B757, Large, and Small. Table 5 shows the sample input aircraft parameters for this research.

Table 5 Sample Aircraft Parameters

Aircraft Type	Wake Class	Maximum Takeoff Weight (kg.)
A300	Heavy	165002
A306	Heavy	165002
A30B	Heavy	142002
A310	Heavy	164022
A318	Large	68001
A319	Large	75501
A320	Large	77001
A321	Large	93002
A322	Large	93002
…	…	…

3.2 Data Processing and Integration

Processing and integrating data are the necessary steps for deriving key state variables (features), for detecting unstable approach events, and building the nowcasting models.

3.2.1 Preprocessing Flight Tracks

The raw data from different sources may contain formats and features which are incompatible. A cleaning procedure is needed to solve the issue. Example cleaning steps are as below:

1. Convert time format in raw data (e.g. “2011-01-17 19:08:09.300”) to GMT cumulative seconds past midnight.
2. Convert GMT to local time.
3. Sort all flights first by track index, then by time (ascending).
4. Filter target airport arrivals by checking the coordinates of the last point in a flight track (i.e., to identify that the latitude and longitude are within a predefined range for a target airport).
5. If needed, filter target airline flights (airline information is contained in the track index).
6. Adjust tracks with ending time less than 0 seconds or greater than or equal to 86400 seconds (24 hours) past local midnight for each day.
7. Trim unused columns and rows.
8. Convert all units to the metric system.
9. Convert longitude and latitude to the Universal Transverse Mercator (UTM) Coordinate System.
10. Generate input flight data files and track data files with standardized format.
11.

After these cleaning steps, the formats of the input flight data and track data are the same over all data sources. For Step 4, the raw data may have contained the

destination airport, but for some sources this information can be incomplete. Therefore, a predefined range of latitude and longitude is needed to filter landing aircraft to the target airport. For example, the latitude range (41.773, 41.799) and longitude range (-87.768, -87.733) are used for Chicago Midway International Airport. As stated in Step 8, for convenience, all units in the data files of this research are converted to the metric system for computation. The outputs of the preprocessing are a flight data file and a track data file. The flight data file contains the flight track index, flight ID, and aircraft type used by each flight. The track data file stores all track-point information for each flight.

The original time series in raw data can have different sampling rates. For N90 TRACON data, the sampling rate is every 4 to 5 seconds. For C90 TRACON data, the data are sampled at a much lower and irregular rate. To unify the sampling rates, linear interpolation is conducted between successive data points to obtain updated tracks. For example, let x_i represent the lateral position of the aircraft at time i. Suppose that i and j represent successive time stamps from the original data, with $i < j$ (both i and j are integers, in seconds). To estimate the lateral position at an intermediate point, $i + n < j$, the following formula is applied:

Equation 1 Linear Interpolation

$$x_{i+n} = x_i + \frac{n}{j-i}(x_j - x_i)$$

A similar process is used to obtain the interpolated points for vertical position.

The updated sampling interval is set at 1 second. Figure 7 shows a comparison between the original track points and the interpolated track points, where the circled points are the original points and the other points are linearly interpolated.

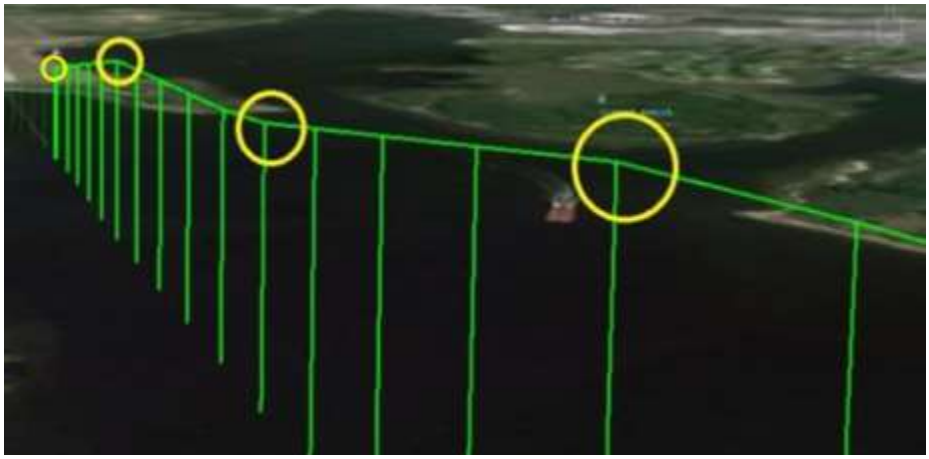

Figure 7 Track Points Linear Interpolation

To reduce noise in the flight tracks, a smoothing procedure is applied in the vertical and horizontal dimensions. The smoothing procedure is applied via a running average with the following formula:

Equation 2 Track Smoothing

$$x_i = \sum_{j=i-m}^{i+m} x_j$$

where x_i represents the horizontal coordinate or altitude for track point at time stamp i. The parameter m indicates the half width of the averaging interval. This study uses m = 5 seconds for track smoothing. Smoothing in the vertical profile is demonstrated in Figure 8. The stepwise vertical profile in the original data is due to low altitude precision (multiples of 100 feet). Smoothing in the horizontal dimension is shown in Figure 9. The lateral noise is reduced by smoothing.

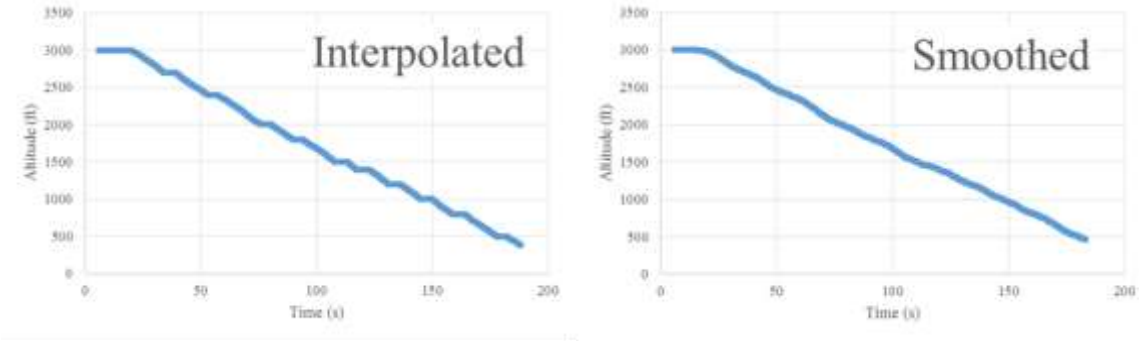

Figure 8 Track Smoothing – Vertical Effect

Figure 9 Track Smoothing – Horizontal Effect

To study unstable approaches, additional variables are needed for analysis besides time and position information. The following three variables are derived from raw positions data: ground speed, vertical speed, and heading angle.

The ground speed of an aircraft at some track point is estimated by simply averaging the ground speed of its two adjacent segments. For example, Equation 3 shows

the calculation of the ground speed at P_2. In this equation, v_{12} is the ground speed of adjacent segment connecting points P_1 and P_2, which is calculated in Equation 4. In Equation 4, t_i is the time in seconds at point P_i. x_i and y_i are the Cartesian coordinates for Point i. Similarly, the vertical speed (i.e. rate of descent) at a track point can be estimated by averaging the vertical speed values of the two adjacent segments.

Equation 3 Ground Speed for Track Point

$$v_2 = \frac{(v_{12} + v_{23})}{2}$$

Equation 4 Ground Speed for Segment

$$v_{12} = \frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}{t_2 - t_1}$$

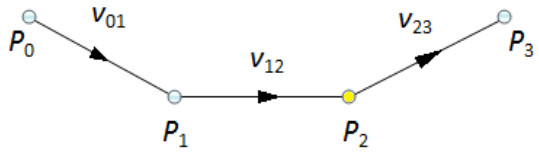

Figure 10 Calculation of Ground Speed

Similarly, the vertical speed (i.e. rate of descent) at a track point can be estimated by averaging the vertical speed values of the two adjacent segments.

To reduce noise in estimating ground speed and vertical speed, we use an averaging interval similar to the process for smoothing the track data (Equation 2). The method of calculating averaged ground speed at point P_j is summarized in the following formula:

Equation 5 Averaged Ground Speed

$$GS(P_j) = \sum_{i=j-\frac{n}{2}}^{j+\frac{n}{2}} v_i \,/(n+1)$$

Figure 11 shows the effect of using different averaging intervals for the smoothing, applied to the derived ground speed for an example track. As the length of the averaging interval increases, the noise is reduced. However, the ability to distinguish speed changes over short time intervals is also reduced. To preserve the original information and reduce the noise as much as possible, an averaging interval of n = 30 seconds is chosen. The same noise reduction technique is applied to derive the averaged vertical speed at target track points.

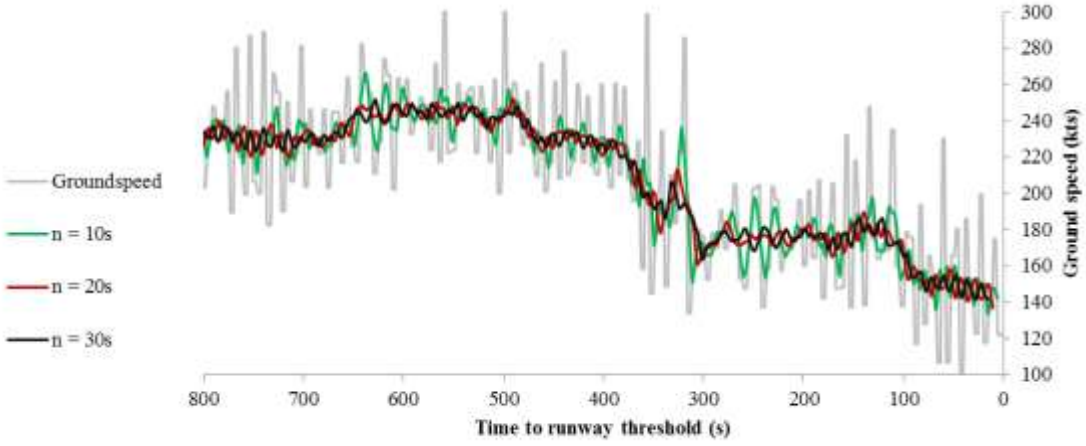

Figure 11 Averaging Time Intervals and Their Effects on Noise Reduction for Ground Speed

To derive the heading angle θ of a track point, define $\theta = 0$ for heading true north. Clockwise from the north, the value of θ (in radians) grows from 0 to 2π (Figure 12). To calculate θ, we average the heading angles formed by the two adjacent segments. The heading angle of P_1 is either $(\theta_{01}+\theta_{12})/2$ or $(\theta_{01}+\theta_{12})/2 \pm\pi$ depending on the situation, where θ_{01} and θ_{12} are the heading angles of segment connecting P_0 to P_1 and P_1 to P_2 respectively. The estimated heading direction at Point 1 is indicated by the dashed arrow in the right chart of Figure 12.

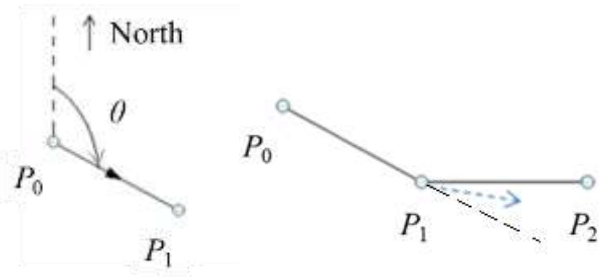

Figure 12 Calculation of Heading Angle

3.2.2 Preprocessing Wind Data

Wind speed, direction, and gust speed are updated hourly. A sample wind record for one day is shown in Table 6. To use the wind information at an arbitrary point in time, interpolation is needed. The linear interpolation for wind speed, direction, and gust follows the same form in Equation 1. A sample interpolated wind speed is demonstrated in Figure 13.

Table 6 Sample METAR Wind Records

Time (EDT)	Wind Dir	Wind Speed	Gust Speed
12:51 AM	NNE	4.6 mph	-
1:51 AM	NNE	6.9 mph	-
2:51 AM	NE	8.1 mph	-
3:51 AM	NE	6.9 mph	-
4:51 AM	NNE	11.5 mph	-
5:51 AM	NE	10.4 mph	-
6:51 AM	NE	10.4 mph	-
7:51 AM	NE	8.1 mph	-
8:51 AM	ENE	9.2 mph	-
9:51 AM	East	8.1 mph	-
10:51 AM	East	8.1 mph	-
11:51 AM	SE	12.7 mph	20.7 mph
12:51 PM	East	12.7 mph	-
1:51 PM	ESE	13.8 mph	-
2:51 PM	SE	10.4 mph	18.4 mph
3:51 PM	ESE	15.0 mph	-
4:51 PM	SSE	12.7 mph	-
5:51 PM	South	9.2 mph	-
6:51 PM	SSE	8.1 mph	-
7:51 PM	South	8.1 mph	-
8:51 PM	ENE	6.9 mph	-
9:51 PM	SE	3.5 mph	-
10:51 PM	Calm	Calm	-
11:39 PM	SSW	5.8 mph	-
11:51 PM	Calm	Calm	-

Figure 13 Interpolated Wind Speed

One major purpose of including wind data is to derive airspeed instead of using groundspeed to better represent the aircraft state. Airspeed is obtained by using Equation 6. The relationship between airspeed, groundspeed, and windspeed vectors is presented in Figure 14.

Equation 6 Deriving Airspeed

$$\overrightarrow{airspeed} = \overrightarrow{groundspeed} - \overrightarrow{windspeed}$$

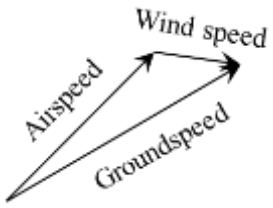

Figure 14 Relationships between Airspeed, Wind Speed, and Ground Speed

Also, given the alignment of a specific landing runway, the crosswind and headwind for a landing aircraft can be derived. An example decomposition of wind speed is shown in Figure 15.

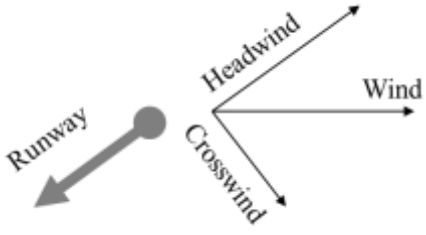

Figure 15 Deriving Crosswind and Headwind

3.2.3 Building Wireframe Approach Zone

Airport data and the navigation information are processed to build the approach zone. The navigation procedures can be obtained from the airport approach charts. An example specification is shown in Figure 16, part of an approach diagram.

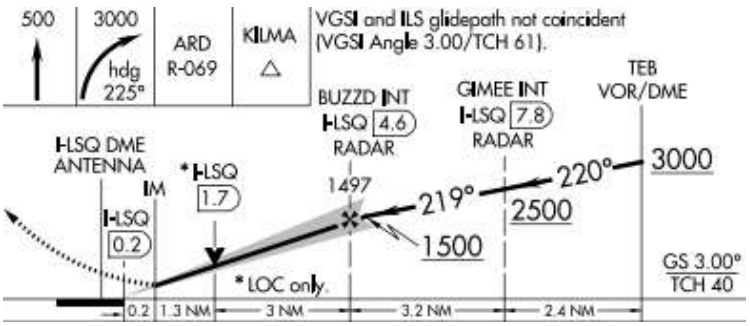

Figure 16 Instrument Approach Procedure for Runway 22L at EWR

With the airport data and the navigation procedures, a conceptual approach zone can be built for each runway. The approach zone can be represented in a wireframe structure. The wireframe starts at the level flight segment before descending (e.g. 12 nm from runway threshold for Runway 22L at EWR). Before reaching the Initial Approach Fix (IAF), the wireframe zone is level, representing the level flight segment before descending. After passing the IAF, the approach path follows the prescribed glide-path angle, which is usually 3 degrees. The landing aircraft is expected to follow the glide-path and cross the runway threshold at a specified crossing height. Figure 17 and 18 show the idea of the wireframe approach zone. The three-dimensional approach wireframe model is built for each runway for each approach type (e.g. Visual and ILS). The modeled zone is defined by the specific heading, elevation, threshold crossing height,

threshold position, and glide-path angle at each runway. The beginning segment of the approach starts from the IAF to the reference fix (e.g. FAF) at about 6 nm; the final approach segment starts from FAF to the runway threshold.

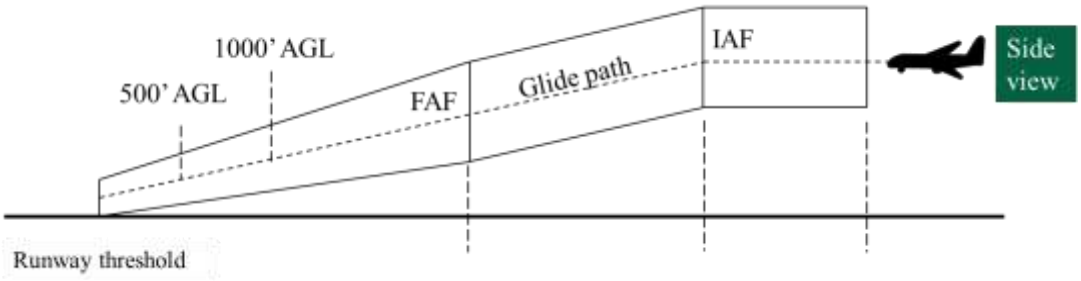

Figure 17 Wireframe Approach Zone – Side View

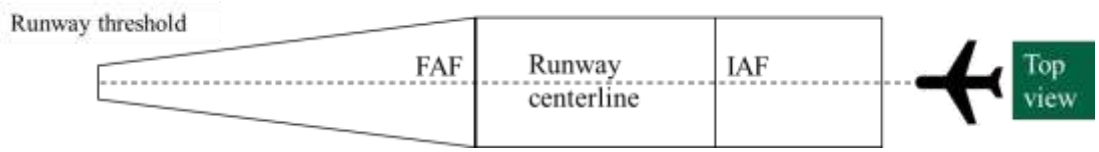

Figure 18 Wireframe Approach Zone – Top View

The lateral and vertical sizes of the wireframe structure are determined by the cross-sectional position distributions of landing flights. This study determines the wireframe sizes so that 95% of flights are within the wireframe boundaries at the FAF (about 6 nm) and the runway threshold (i.e. 0 nm). In between the FAF and runway threshold, the approach zone is determined by linearly connecting the wireframe coordinates. The wireframe size beyond the FAF is the same as at the FAF.

This wireframe region is a region containing the majority of landing trajectories. The purpose of building the wireframe is to:

- Determine whether a flight is on track (hence determine whether the stabilized approach criteria are violated),
- Identify glide-path/centerline acquisition positions,
- Identify boundaries for binning state variables of lateral/vertical position for nowcasting methods.

3.2.4 Landing Runway Identification

Since there is no landing runway information recorded in the original data, a method to automatically identify the landing runway from a flight track is needed. A heuristic algorithm is developed to detect the landing runway for each arrival flight. The approach is to search backward from the last track point (on or near runway threshold). If the track point is inside any modeled wireframe for a particular runway, the following variables are calculated:

- Distance traveled inside the wireframe d.
- Time spent inside a wireframe t.
- Angle difference with runway centerline α.
- Distance from the corresponding runway threshold D.

The criteria are summarized in Table 7. If all of these conditions are satisfied by the track points from the time the aircraft enters the wireframe until the last point, then the landing runway of the flight is the runway corresponding to the modeled wireframe.

Table 7 Parameters and Criteria for Landing Runway Identification

Parameters	Description	Criteria	Thresholds
d	Distance traveled inside a wireframe	$d > d_{threshold}$	50 meters
t	time spent inside a wireframe	$t > t_{threshold}$	3 seconds
α	angle difference from runway centerline	$\alpha < \alpha_{threshold}$	$\pi/4$
D	distance from runway threshold	$D < D_{threshold}$	3 nautical miles

This method has been validated by visual checking the landing runways of hundreds of sample arrival flights. In future work, supervised learning techniques can be used to optimized the threshold parameters and improve the degree of automation and performance of this rule-based method.

3.2.5 Flight Track Qualification

The surveillance track data may contain flights with incomplete track data. A qualification procedure is a final step in preprocessing the raw data sets. To filter tracks, the prediction model focuses only on the qualified flight tracks which satisfy all the following criteria:

- The landing runway is identified.
- The aircraft type information is known.
- The flight track has entered a wireframe approach zone.
- The altitude of the first track point is greater than 3000 ft. AGL.
- The altitude of the last track point is less than 500 ft. AGL.
- The distance from the first available track point to the landing runway threshold is larger than 12 nm.

- The distance from the last track point to the landing runway threshold is smaller than 3 nm.
- The time duration from first entrance of the wireframe zone to the final landing is less than 400 seconds.

The case study in Chapter 4 shows that most of the flights (98.6%) in the surveillance track data have their tracks satisfying these qualification conditions. The last condition is used to filter out the flights with multiple approaches. A go-around is an example of such a case. These abnormal flight track patterns are included in the summary, e.g. go-around statistics, but not for prediction of unstable approaches. One reason is that some state variables, such as rate of descent, for these abnormal flights can be inconsistent with those of qualified tracks. As a result, the performance measures (e.g. speed change from 1000' AGL to runway threshold) for these flights can be misleading.

3.3 Extracting State Variables at Key Locations

To define the risk events related to an unstable approach using flight track data, it is essential to first identify the track points which mark key events along the approach path. Such events include lateral/vertical acquisition of wireframe, acquiring 1000'/750'/500' AGL, and acquiring 10 nm/6 nm/3.5 nm, etc. For example, a type of key track points is featured by being at key horizontal distances from runway threshold. The algorithm to identify the track point which is at a specific distance from some key location is presented in pseudocode below:

```
dist_difference = M;
for (i = 1; i <= track_size; i = i + 1)
{
```

```
        current_dist = distance(P(i), KEY);
        if abs(current_dist - target_dist) < dist_difference
        {
              dist_difference = distance(P(i), KEY);
              i_dist = i;
        }
  }
  return P(i_dist)
```

In this script, M is an arbitrarily big number (e.g. 10^5 m). "P(i)" is the track point with index i. KEY is any key location (e.g. threshold of the landing runway). "abs()" calculates the absolute value. "distance()" calculates the two-dimentional or three-dimentional distance between two points depending on the type of key events (e.g. for detecting acquisition of FAF, 3-D distance is applied; for detecting where an aircraft reaches some specific horizontal distance from runway threshold, 2-D distance is used). The index i_dist is for the track points at key horizontal distances. The "target_dist" is the indicated distance from the key location. If the task is to find the track point closest to the key location, then target_dist is set zero. The algorithm chooses the last track point at key distances from some key locations if there are multiple qualified points. The state variables of identified points will be used as features for nowcasting unstable approaches.

Another class of points are the points corresponding to the fixes along the approach path. These points are detected in a way that they are closest in proximity to the target fixes. The target fixes include the runway threshold, and 1 to 12 nm from the runway threshold along the approach path. For example, for EWR Runway 22L, the flight state at the track point that is closest to 10 nm from the runway threshold is used to approximate its performance at the FAF.

Other types of key track points include the wireframe entrance points and the target altitude points. The wireframe entrance points correspond to the moment when an aircraft first enters the modeled wireframe approach zone. The target altitude track points indicate where the flights capture the target altitude levels such as 1000'/500' AGL. Examples of these points are shown in Figure 19.

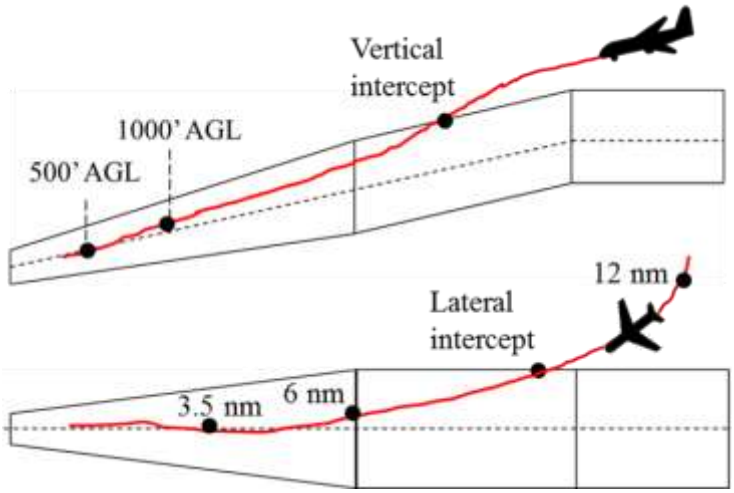

Figure 19 Key Locations and Corresponding Track Points

With all these track points at key locations identified, the aircraft state variables at these locations can be obtained, for example, the airspeed at the moment when a landing aircraft captures 1000 ft. AGL. These state variables quantitatively determine whether a landing flight satisfy the stabilized approach criteria.

3.4 Identification of Unstable Approach Events

The purpose of processing flight track data, wind data, and building the wireframe model is to form the quantitative criteria for determining a stabilized approach. With the state variables of a landing aircraft at key locations along the final approach path extracted, the identification of unstable approach events can be achieved.

3.4.1 Defining Unstable Approach Events

Based on the original stabilized approach criteria given by different airlines/regulators, the criteria used in this research are developed, which focus on applying processed flight track data and navigation procedures. Specifically, the criteria used in this research are as follows:

- The aircraft should acquire the glide-path by maintaining its position within the defined vertical boundary of the wireframe after reaching 1000'/500' AGL.
- The aircraft should acquire the runway centerline by maintaining its position within the defined lateral boundary of the wireframe after reaching 1000'/500' AGL.
- At 1000'/500' AGL the aircraft airspeed should remain within ±10 knots of the airspeed at the runway threshold (i.e. landing speed).
- From 1000'/500' AGL to the runway threshold, the rate of descent should not be greater than 1000 feet per minute for more than 10 seconds;

If any of these four conditions is not satisfied, the stabilized approach criteria are not met, and an overall unstable approach event occurs for the corresponding stabilization altitude.

It should be noted that these derived stabilized approach criteria are a *subset* of the original criteria used in airlines or defined by organizations. For example, criteria related to aircraft configuration are not included in the previous definition, since aircraft configuration is not available in the surveillance track data.

3.4.2 Defining Unstable Approach Sub-events

Following the criteria developed for identifying unstable approaches, a list of sub-events of unstable approaches can be defined. These sub-events are introduced in Table 8. These sub-events are the necessary events for the overall event. If any sub-event occurs, the approach is unstable.

Table 8 Unstable Approach Events

Unstable Event	**Definition**
Unstable approach	Aircraft is classified as unstable if any following sub-event occurs
Unstable speed (deceleration)	Difference of averaged ground speed $v_{1000'/500'} - v_{THR} > 10$ knots
Unstable speed (acceleration)	Difference of averaged ground speed $v_{1000'/500'} - v_{THR} < -10$ knots
Excessive rate of descent	Rate of descent > 1000 feet per minute for more than 10 seconds
Short acquisition from above	Acquisition of glide-path from above after stabilization point
Short acquisition from below	Acquisition of glide-path from below after stabilization point
Short acquisition from left	Acquisition of runway centerline from left

side	side after stabilization point
Short acquisition from right side	Acquisition of runway centerline from right side after stabilization point

In the definition of Unstable Speed events, $v_{1000'/500'}$ represents the airspeed when aircraft captures 1000'/500' AGL and v_{THR} represents the aircraft airspeed at runway threshold. In defining the excessive rate of descent event, a time duration is used to filter out possible noises. For example, Figure 20 shows a sample flight's altitude and rate of descent profile during final approach. If it captures 1000' AGL at 70 seconds from touchdown, it can be seen that its rate of descent is higher than 1000 feet per minute for more than 10 seconds after capturing 1000' AGL, then the event of excessive rate of descent occurs.

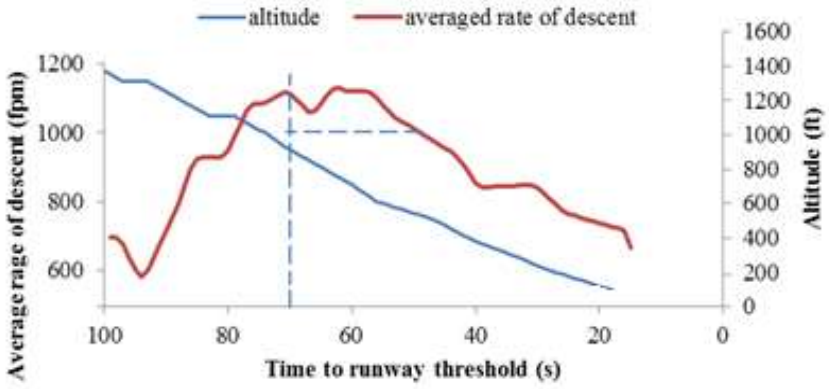

Figure 20 Excessive Rate of Descent

3.5 Nowcasting by Conditional Probability Method

A basic question is: historically, given the similar conditions for current flight, what is the probability that an unstable approach occurred after acquiring stabilization altitude? The idea of conditional probability method is to answer such a question. With the probability obtained, it can be used as a criterion for nowcasting whether an approach is stable or not.

To answer the question above, a table of conditional probabilities is needed. The conditions are expressed in combination of binned state variables. The corresponding probabilities are estimated by obtaining the proportion of flights which fall into this condition (i.e. combination of state variable categories) that experienced unstable approaches.

This method starts with determining and binning the state variables. First, the number of bins for each variable needs to be specified, for example, 3 categories for airspeed (“low”, “normal”, “high”). Then the boundaries for each variable need to be defined for binning. For example, for lateral/vertical position, the wireframe boundary can be used.

A condition mean a combination of binned state variables. The output format of this method is hence a table of conditional probabilities. The contents of each cell such table include: number of samples, observed proportion of target unstable event (i.e. the estimated probabilities under corresponding condition), and derived confidence interval. The confidence interval for proportions depends not only on the number of samples, but also on the probability of target event. Equation 7 gives the form of confidence interval for estimating mean of proportions. In this formula, *CI* represents the confidence interval,

$\bar{p}$ is the observed probability of target event, *n* is the sample size, and $z_{\alpha/2}$is the value for the standard normal random variable corresponding to an area of $\alpha/2$ in the upper tail of the distribution, where α is the prescribed significance level. With a probability of (1- α) the mean proportion is within the confidence interval.

Equation 7 Confidence Interval for Proportion Mean

$$CI = \bar{p} \pm z_{\alpha/2}\sqrt{\frac{\bar{p}(1-\bar{p})}{n-1}}$$

For some cells, the results may be less reliable due to limited number of samples and the value of conditional probability. To deal with the various confidence levels of calculated probabilities in each cell, a threshold for minimum confidence level is needed. If the confidence interval computed in a cell exceeds the minimum range, the probability calculated from this cell will not be reported. For example, to filter out results with inadequate confidence, a confidence threshold can be set (e.g. ±10%). Then, the results will not be reported if there is insufficient confidence level.

To apply Equation 7, a basic assumption is needed that the sampling distribution of proportion mean follows normal distribution. This assumption approximately holds when $np \geq 5$ and $n(1-p) \geq 5$. An alternative way is to set a threshold for number of samples *n*.

In summary, the criteria for reporting the conditional probability is as below:

1. Minimum number of samples is reached. (e.g. > 20 in each combination)
2. Confidence level is sufficient. (e.g. *CI* within [-10%, 10%])

For cells with enough samples and sufficient confidence levels, the following criteria are applied to nowcast the unstable approach events:

- Conditional probability is relatively higher than the unconditional probability P (e.g. 1.2*P)

If the ratio (conditional probability over unconditional probability) is less than the predefined level, flights under this condition are predicted stable. If the ratio is higher than the predefined level, then the flights are nowcasted unstable.

To apply this method of nowcast, basic information such as landing runway, aircraft weight class, distance to destination etc. need to be set first. Different runways can have different landing procedures, wind conditions, etc. Each scenario corresponds to a different probability table.

This method needs a large amount of data, which is a major limitation. As the number of variables and categories increase, the number of cells which contains not enough sample data to compute the probability of unstable approaches increases dramatically. Cells with insufficient confidence levels are not applicable for reporting. In fact, for the case study of this research, an implementation of conditional probability method has shown that the possibility of adding more than four features is prevented given current amount of available data.

3.6 Nowcasting by Supervised Learning

Supervised Learning is a classification approach in the field of Machine Learning. Supervised learning models are trained by first feeding the model with samples, each having their features and outcome known. After the model parameters are trained with sufficient samples, the model can be used to automatically classify the outcomes given the features of new samples.

In this research, the features are the various state variables, and the outcomes are the unstable approach events. The classification problem can also be seen as prediction or nowcasting problem, where the output of the model indicate whether a flight will experience unstable approach events or not given features at the nowcasting location.

Logistic regression is one of these models which is often used for classification and prediction. A logistic regression model is developed to establish the relationship between these features and the outcome of stable/unstable approaches. The mathematical expression is given in the following equation:

Equation 8 Hypothesis Function

$$h_\theta(x) = \frac{1}{1 + e^{-\theta x}}$$

In this equation, $h_\theta(x)$ is the probability that a flight with features x ends up in an unstable approach, where x is a vector containing the feature values and θ is a vector containing the regression coefficients.

The optimal coefficients can be found by minimizing the cost function $J(\theta)$ given below:

Equation 9 Cost Function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

In this formula, m is the number of training samples and i is the index for samples. $y^{(i)}$ is the actual outcome of sample i, which is equal to 1 if sample flight i ends up with unstable approach, otherwise $y^{(i)} = 0$. *Log* is the natural logarithm. The Gradient Descent method is applied to find the optimal coefficient θ. The coefficients updating procedure is

shown in Equation 10, where *j* is the index for features and parameter α controls the step length. The term following α is the gradient $\frac{\partial J(\theta)}{\partial \theta_j}$.

Equation 10 Gradient Descent Algorithm

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

As inputs to the model, all features are mean normalized using Equation 11, where *x* is the original feature value, μ is the mean value of the feature for all samples of training data, and σ is the standard deviation. The purpose of normalization is to keep the efficiency of the algorithm.

Equation 11 Mean Normalization

$$z = \frac{x - \mu}{\sigma}$$

In application, the degree of the polynomial θx can be set to integer value greater than one to derive more features based on the basic input features so the model can provide better fits. For example, if the degree of polynomial is set 2, then the potential interactions between features are considered in the model. To be specific, assuming the number of basic features is *m*, the number of total features will be the *m* basic features + *m* squared features + *m*-choose-2 interaction features.

3.7 Evaluating Performances of Prediction Models

Several measures are applied to quantify the performances of the models. They are accuracy, recall, prediction precision, and F1 Score.

The accuracy is an overall measure which reflects the model performance for all samples of stable and unstable flights. When the classes are skewed, this measure can be misleading. For example, when a very small portion of flights experienced unstable

approach, even if the model performs very poor on predicting unstable approaches (but good in predicting stable approaches), the total accuracy can still be high due to large numbers of stable flights. To overcome this issue, additional measures are needed.

The recall is the proportion of actual unstable approaches that have been correctly predicted. The precision represents the proportion of actual unstable approaches within predicted unstable approaches.

F1 Score is an aggregate measure of recall and precision. The decision makers can choose to accept the results if the F1 Score is above some predefined level. The definition of F1 Score is given below in Equation 12, where *P* and *R* stand for precision and recall.

Equation 12 F1 Score

$$F1\ Score = \frac{2PR}{P + R}$$

The relationships and the calculations of these measures are shown in Figure 21 and Table 9.

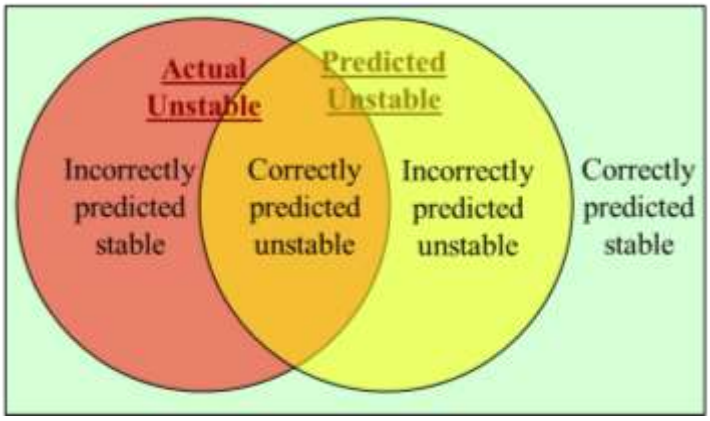

Figure 21 Relationships between Predicted and Actual Stable/Unstable Flights

Table 9 Performance Measures

Measure	Definition
Accuracy	$\frac{\#correct\ predictions}{\#all\ approaches}$
Precision	$\frac{\#correctly\ predicted\ unstable}{\#predicted\ unstable}$
Recall	$\frac{\#correctly\ predicted\ unstable}{\#actual\ unstable}$
F1 Score	$\frac{2 * Precision * Recall}{Precision + Recall}$

These measures are directly applicable to conditional probability method. For supervised learning method, the data set are randomly split into training set and test set. Training set is used to train the model parameters (i.e. the coefficients for features) by minimizing the cost function. The measures of performance are applicable only toward the test set.

CHAPTER 4 EWR RUNWAY 22L CASE STUDY

This chapter introduces a case study of 8,158 approaches to Runway 22L at EWR using the methods introduced in Chapter 3. This chapter first presents the processed and integrated data sets for the current case study. Then some key observations of unstable approach events are identified and effects of several key factors are analyzed. The implementations of the conditional probability method and the supervised learning method are presented, with their performance evaluated and compared.

4.1 Processing and Integrating EWR 22L Data

The target runway for the case study is EWR 22L, highlighted in the airport diagram in Figure 22. This study uses 28 days of operations in 2007 at the N90 TRACON. In total, there are 18,426 arrival flights to EWR during the study period. Of these, 8,275 (44.9%) are detected to land on Runway 22L. Within these 8,275 22L landings, 8,158 (98.6%) of them are qualified tracks (satisfying the criteria in Section 3.2.5). The fleet mix is Large-dominant, with 73.6% of the flights as Large aircraft. Table 10 summarizes the key parameters for Runway 22L.

The key parameters for Runway 22L is summarized in Table 10.

Table 10 Runway 22L Parameters

True alignment	Glide-path angle	Elevation	Threshold crossing height

206 degrees	3 degrees	9.1 ft	61 ft

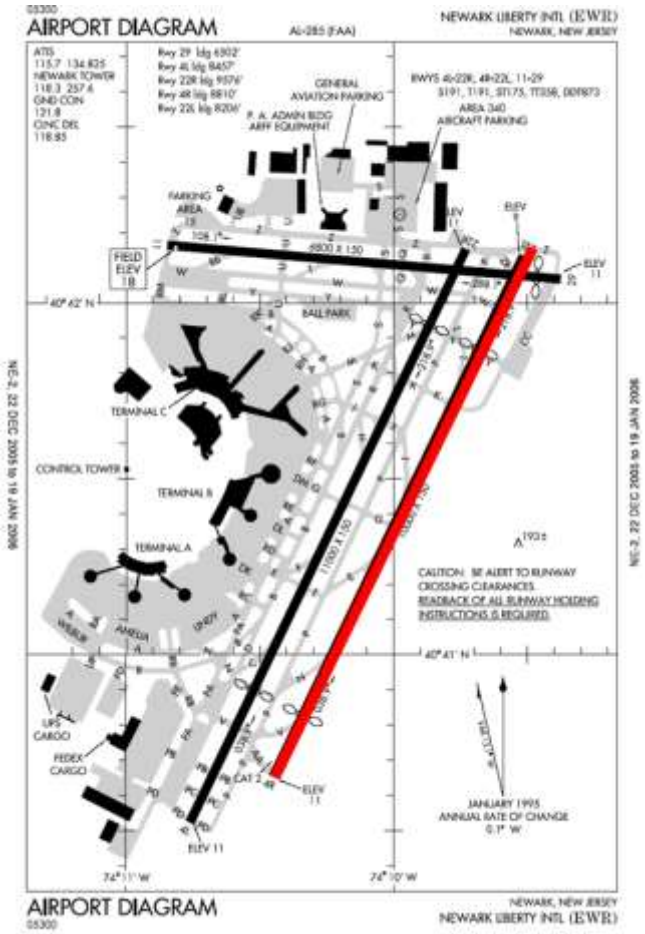

Figure 22 EWR Runway 22L

For EWR Runway 22L, according to the published approach diagrams, the difference between the ILS and visual approach procedures is not significant (Figure 23) and cannot precisely be differentiated purely from surveillance track data. The beginning of the final approach is 9.4 nm from the runway threshold for the visual approach and is

10.1 nm for the ILS approach. For this reason, a single approach zone is built for all approaches.

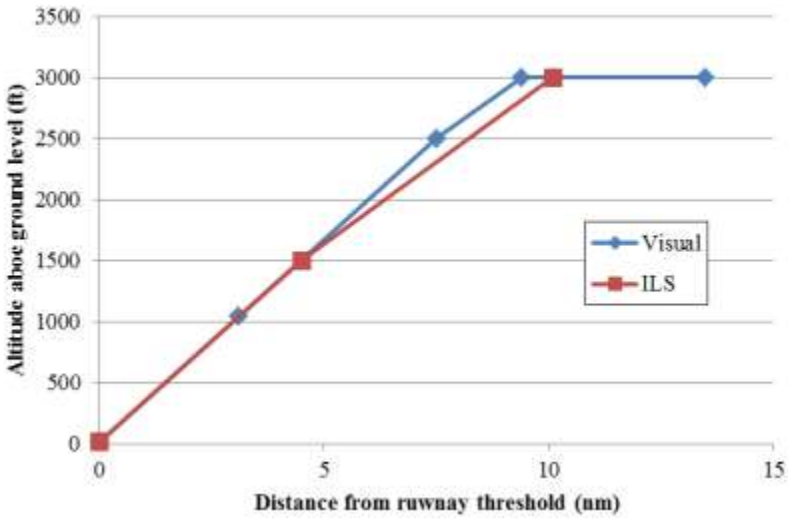

Figure 23 Vertical Profile of Different Approach Procedures at EWR 22L

To determine the approach zone, four basic fixes are identified along the approach path. They are the 12 nm fix, the IAF, the FAF at 6nm, and the runway threshold. According to Figure 23, the beginning of the final approach is at 9.4 nm for the visual approach or at 10.1 nm for the ILS approach. As an averaged approximation, the IAF for 22L is set at 10 nm. Therefore, the wireframe starts at 12 nm from the runway threshold. Before reaching the IAF at 10 nm, the wireframe zone is level, representing the level flight segment before descending. After passing the IAF, the approach path follows a 3-degree glide-path angle, flying along the runway centerline with a heading of 206 degrees.

The lateral and vertical sizes of this zone are determined from actual cross-sectional positions of landing tracks at different distances from the runway threshold. Figure 24 shows the scatterplots for crossing positions of arrival tracks at 6 nm from the runway threshold. These crossing position distributions are used to determine the dimensions of the approach zone. Specifically, each dimension of the zone is sized to contain approximately 95% of all arrival flight trajectories in that dimension. For Runway 22L, given the crossing positions, the estimated cross section sizes (half width) at 6 to 12 nm from runway threshold are 475 ft lateral and 275 ft vertical. At the runway threshold, the lateral and vertical sizes are set to 135 ft and 80 ft. These boundaries are shown as red boxes in Figure 24. For simplicity, the cross-sectional sizes between the runway threshold and 6 nm can be linearly determined. Beyond 6 nm, the wireframe boundary dimensions remain of fixed size.

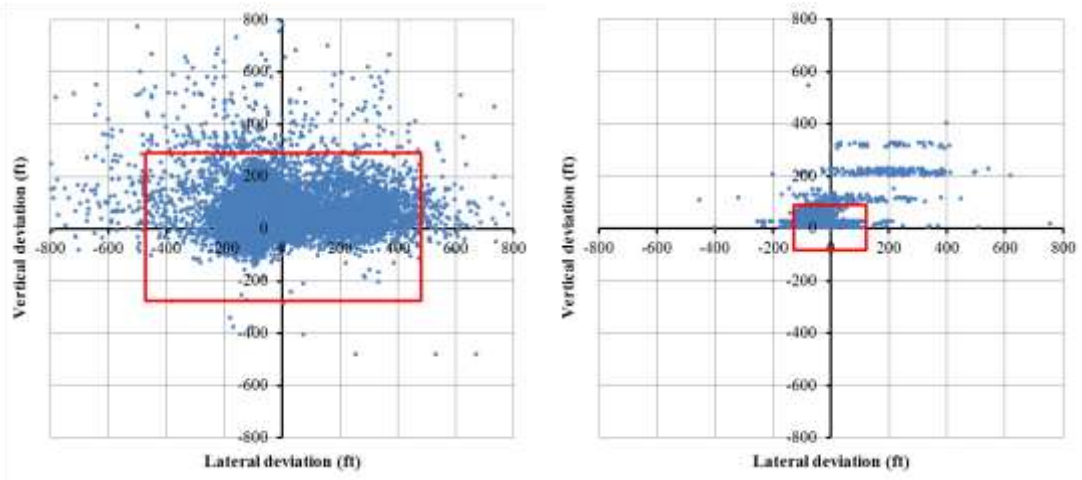

Figure 24 Scatterplot of Cross-sectional positions at 6 nm and 0 nm from Runway Threshold

4.2 Key Observations

4.2.1 Statistics of Unstable Approach Events

Table 11 summarizes the statistics for unstable approach risk events. The first row shows the overall unstable events and the following rows shows sub-events. Overall, according to the stabilized approach criteria defined in Section 3.4.1:

- 47.3% of all flights landing to 22L were not stable after reaching 1000'.
- 33.6% of all flights landing to 22L were not stable after reaching 750'.
- 17.5% of all flights landing to 22L were not stable after reaching 500'.

Table 11 Statistics of Unstable Approach Events

Risk Event	1000' AGL		750' AGL		500' AGL	
	Count	**Proportion**	**Count**	**Proportion**	**Count**	**Proportion**
Unstable approach	3858	47.3%	2737	33.6%	1425	17.5%
Unstable speed (decelerate)	2188	26.8%	1123	13.8%	334	4.1%
Unstable speed (accelerate)	479	5.9%	291	3.6%	140	1.7%
Excessive rate of descent	152	1.9%	61	0.8%	16	0.2%
Short acquisition from left	727	8.9%	573	7.0%	178	2.2%
Short acquisition from right	1592	19.5%	1311	16.1%	697	8.5%

Short acquisition from above	423	5.2%	352	4.3%	232	2.8%
Short acquisition from below	84	1.0%	84	1.0%	83	1.0%

Within the table, the following sub-events have occurrence probabilities greater than 10%:

- Unstable speed (excessive deceleration) between 1000’ and touchdown.
- Unstable speed (excessive deceleration) between 750’ and touchdown.
- Short acquisition (capturing from right) after reaching 1000’.
- Short acquisition (capturing from right) after reaching 750’.

These events indicate that being too fast and not acquiring the runway centerline at or before reaching 1000’ or 750’ AGL are the major causes of an unstable approach for landing flights at EWR 22L. Due to the curve-in pattern of landing tracks at 22L, most flights penetrate the wireframe from the right side. This explains the asymmetry between the proportion of short-left and short-right acquisition of runway centerline. Flights are more prone to short acquisition from right side. (Figure 25)

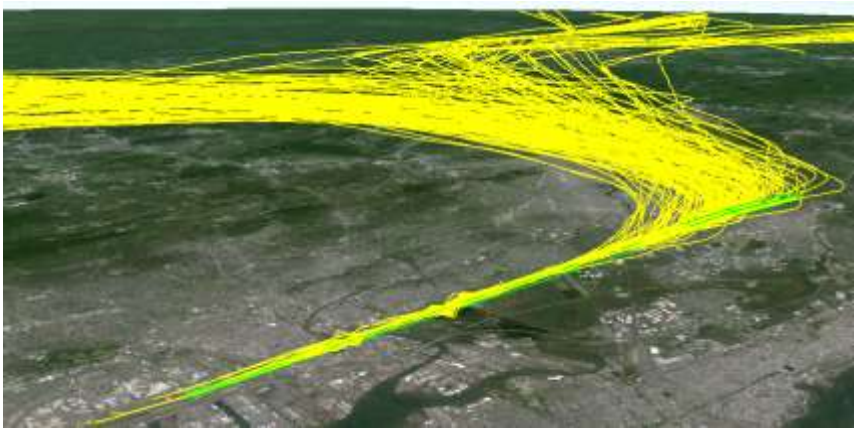

Figure 25 A Snapshot of Landing Tracks to EWR 22L

Table 11 also shows that the approaches are more stable as the flights get closer to the runway. The probability of experiencing unstable approaches after 500' AGL is 17.5%, 63% lower than that for 1000' AGL (47.3%).

The statistics for unstable approaches reported here are significantly higher than previous estimates in public literatures. The potential reasons for this difference are:

1. The criteria used in this research are stricter than the criteria proposed by airlines/regulators. The criteria in this research focus on the process *between stabilization altitude and touchdown*, while other criteria indicate checking the criteria *only at the stabilization altitude*.
2. Due to lack of relevant information in the raw surveillance data, there is no information of the approach types these approaches experienced. Considering there is a mix of VMC and IMC, the real proportion of unstable approaches is between 47.3% and 17.5%.

4.2.2 Observations – Weight Classes and Unstable Approaches

Table 12 summarizes the composition of the landing fleet. The majority of landing aircraft at 22L belong to the large category. The table shows the probability of experiencing an unstable approach by weight class. Small aircraft are more prone to unstable approaches. Specifically more than 80% of small landing aircraft experience unstable approaches after 1000', and 40.3% after 500'.

Table 12 Breakdown by Aircraft Weight Classes

Weight Class	Count (proportion)	%Unstable Approaches 1000'	%Unstable Approaches 500'
Heavy	968 (11.9%)	62.3%	30.8%
B757	1,041 (12.8%)	52.4%	18.3%
Large	6,000 (73.6%)	43.2%	14.6%
Small	149 (1.8%)	80.5%	40.3%

The reason for this requires a detailed look at the sub-events, shown in Table 13. Hypothesis tests are conducted, with the null hypothesis assuming equal proportions of target sub-event between the two groups. The null hypothesis is rejected for subevent 1, 2, 3, and 6. For small aircraft, the biggest contributor to unstable approach events is the first sub-event, i.e. unstable speed change (deceleration). In fact, to maintain runway utilization and runway throughput, flights are instructed to hold a similar speed leading up to the FAF. Small aircraft must decelerate further to achieve their designated landing speed. As shown in Figure 26, small aircraft have a steeper slope in their average speed profile, which indicates a higher deceleration rate.

From the table, it is also shown than small aircraft flies a higher trajectory, which leads to an excessive rate of decent and short acquisition from above.

Table 13 Sub-events of Unstable Approach for Small Aircraft

Sub-events 1000' AGL	Small aircraft	Non-Small aircraft	H_0: $p_1 = p_2$ ($\alpha = 0.05$)
Unstable speed (decelerate)	67.1%	26.1%	Rejected
Unstable speed (accelerate)	2.7%	5.9%	Rejected
Excessive rate of descent	6.7%	1.8%	Rejected
Short acquisition (from left)	8.7%	8.9%	
Short acquisition (from right)	17.4%	19.6%	
Short acquisition (from above)	20.8%	4.9%	Rejected

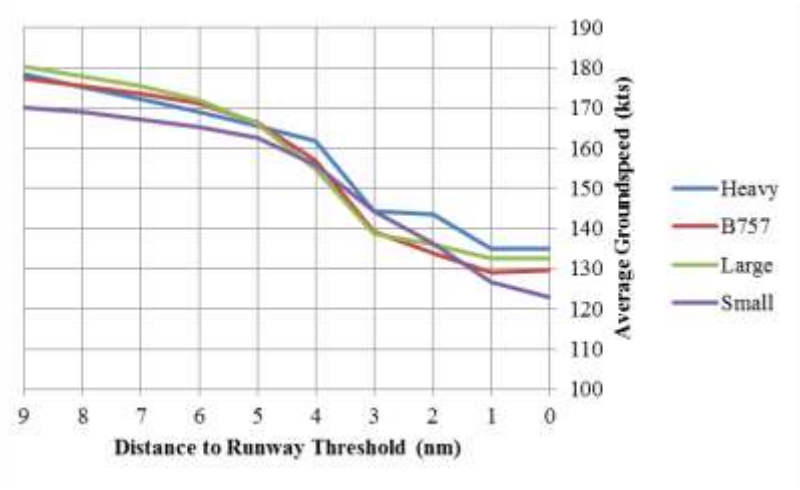

Figure 26 Speed Profiles for Aircraft in Different Weight Classes

4.2.3 Observations – Wind Conditions and Unstable Approaches

Among the qualified 8,158 arrival tracks, the wind conditions at 6 nm from runway threshold are derived and summarized below:

- 1121 (13.7%) flights experienced a tailwind (negative headwind);

- 442 (5.4%) flights experienced gust;
- 242 (2.8%) flights experienced crosswind with magnitude > 10 knots;
- 18 (0.2%) flights experienced crosswind with magnitude > 15 knots;
- The crosswind speed of the southeast winds had a higher variance compared to the northwest winds.

Figure 27 shows the distribution of crosswind speed experienced by all landing flights to 22L. Positive values represent winds with a southeast component, i.e. from left to right across the runway centerline when facing the runway threshold. All crosswinds greater than 15 knots were in the direction or with a component in the direction from the southeast.

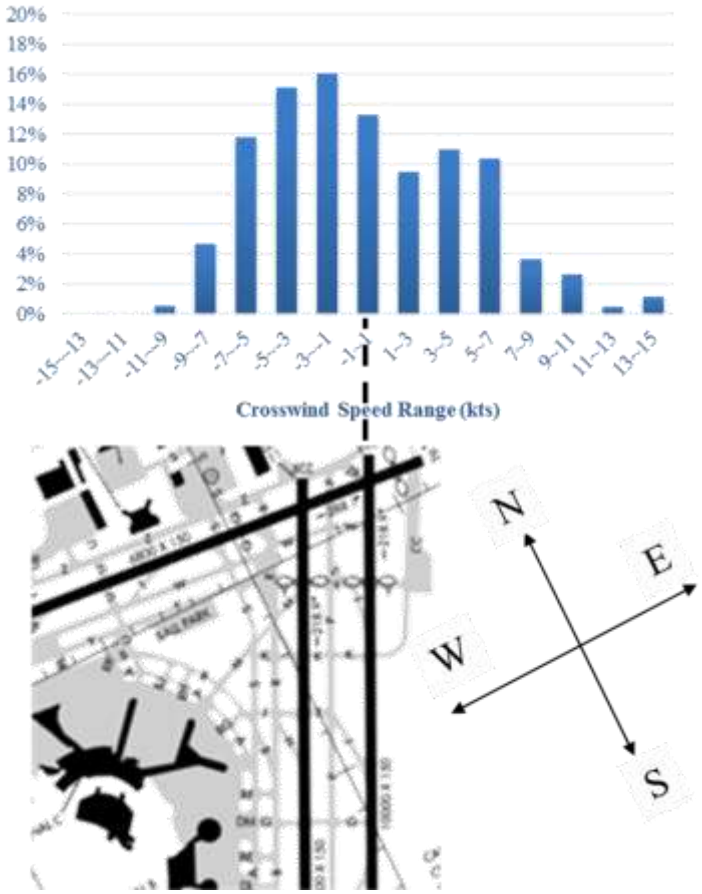

Figure 27 Distribution of Crosswind Speed at EWR 22L

To study the impact of crosswind speed on unstable approaches, the crosswind speeds are divided into bins with length of 2 knots. For each bin, the proportion of unstable approaches is calculated, with the 95% confidence intervals constructed as indicators of certainty. The results are shown in Figure 28. The figure indicates that a higher crosswind magnitude increases the probability of experiencing unstable approaches after acquiring the stabilization altitude.

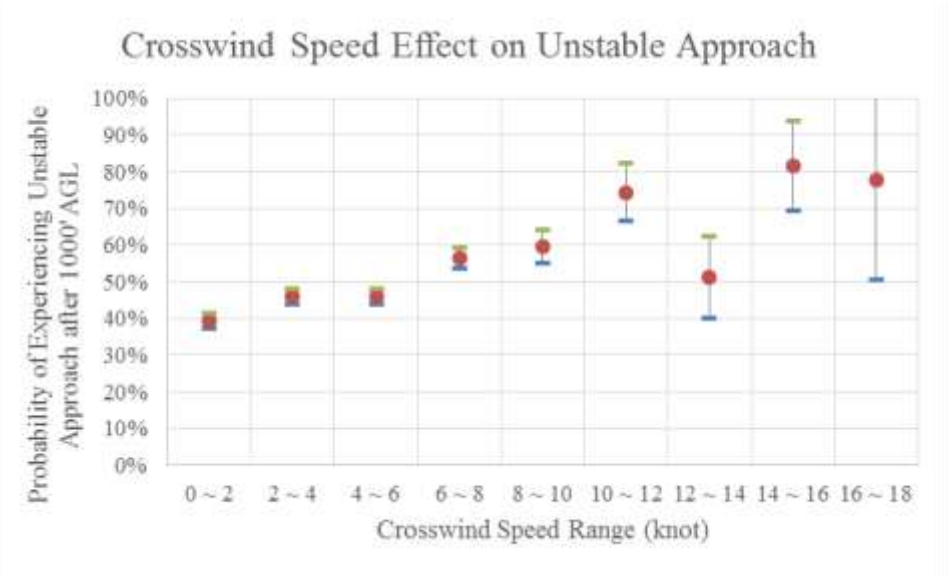

Figure 28 Crosswind Speed Effect on Unstable Approach

4.2.4 Observations – Speed and Altitude and Unstable Approaches

Flying high and fast is a key causal factor for unstable approaches. Figure 29 shows the distributions of airspeed and altitude observed at 6 nm from runway threshold. The mean airspeed at 6 nm is 174.0 knots with a standard deviation of 17.8 knots. The mean altitude at 6 nm is 1988.2 ft AGL with a standard deviation of 120.2 ft.

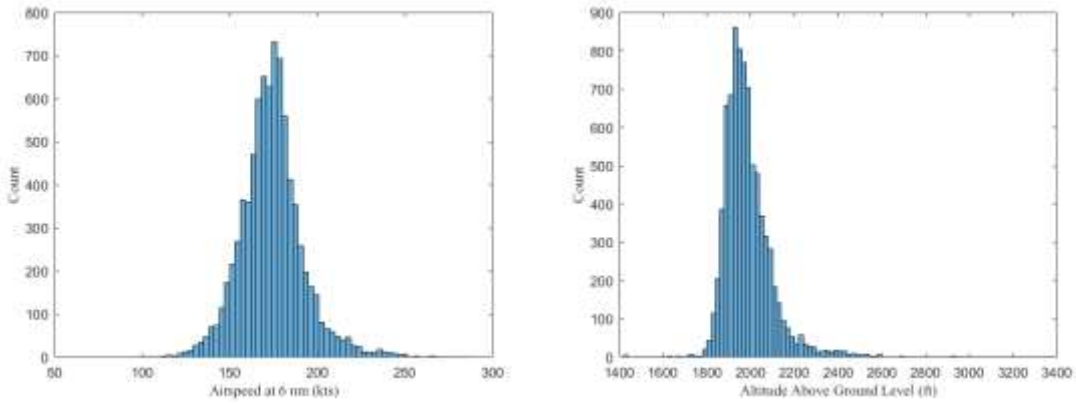

Figure 29 Distribution of Airspeed and Altitude at 6 nm

Figure 30 shows the effect of airspeed. It shows that both high and low speeds lead to an increased probability of experiencing unstable approach below 1000’ AGL. Flying too fast at 6 nm leads to excessive deceleration, while flying too slow requires excessive acceleration to achieve the designated landing speed. Both have the potential to violate the stabilized approach criterion of less than 10 knots of speed change from the reference landing speed.

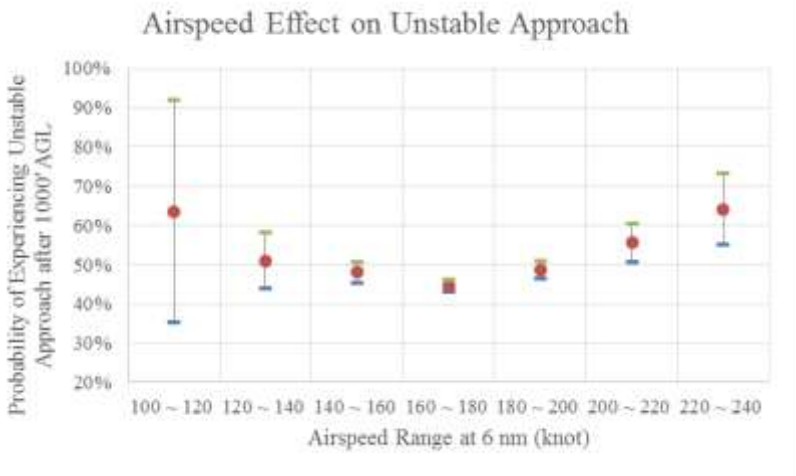

Figure 30 Airspeed Effect on Unstable Approach

Figure 31 shows the effect of altitude. Generally, a higher altitude at 6 nm increases the probability of incurring an unstable approach below 1000' AGL. Flying too high may lead to a short acquisition of the glide-path from above. Flying high can also lead to a higher approach speed. From the chart, if a flight is 400 ft above the prescribed glide path at 6 nm (which is at about 2000' AGL), the observed proportion of flights experiencing unstable approaches after 1000' AGL increases by approximately 15%.

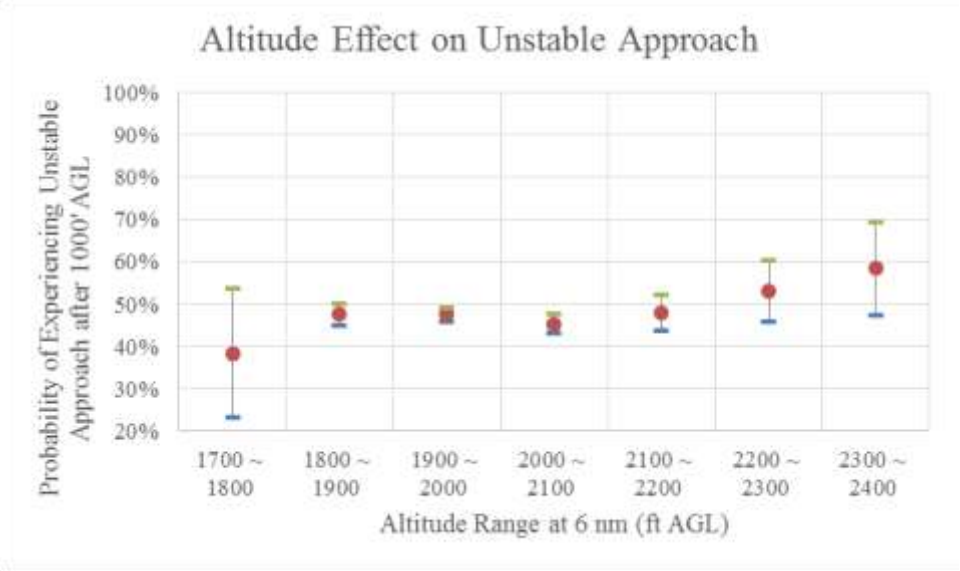

Figure 31 Altitude Effect on Unstable Approach

Flying high and fast bring high risks of an unstable approach. Fortunately, the proportion of flights flying at a higher speed (10 knots higher than the mean value) and a higher altitude (250 ft above glide-path) is only 2.4% (199 out of 8,158 flights). However, the probability of experiencing an unstable approach after 1000' AGL for these high and fast flights is 75.4% (150 out of 199 flights), which is significantly higher than the average probability introduced in Section 4.1 (47.3%). This has validated the fact that flying high and fast can bring much higher risk to approach and landing.

Because of correlations between factors, it may not be possible to adjust a single factor in isolation in order to improve the probability of an unstable approach. Correlations between factors must be taken into consideration when correcting the current trajectory. Table 14 summarizes the correlation coefficients between several key factors. For example, a positive correlation between rate of descent and airspeed (0.38)

implies that a higher airspeed is usually associated with a higher rate of descent. Aggregated effects from all factors must be considered in reducing the probability of an unstable approach.

Table 14 Correlations between Key Factors

	Lateral deviation	**Altitude**	**Airspeed**	**Rate of descent**
Lateral deviation	-	0.07	0.12	-0.11
Altitude	-	-	0.04	0.10
Airspeed	-	-		0.38
Rate of descent	-	-	-	-

4.2.5 Observations – Go-around

Go-arounds can result from unstable approaches. Here, several heuristics are applied to detect go-around events. Results are compared with published results. Go-arounds are detected using the following criteria:

- The time from first penetrating the wireframe to the time of the last track point is greater than 400 seconds, and
- The cumulative change of course angle is greater than 330 degrees after first penetrating the approach wireframe zone.

If a flight track satisfies both conditions, it is considered a go-around flight. The two criteria work together to correctly detect go-arounds. For example, with only the first

criterion, some non-go-around flights are identified which simply have a large number of recorded track points on the airport surface.

Figure 32 shows the time from penetrating the wireframe to the final track point of each flight. The peaks shows the outliers of this measure which are candidates of go-around flights. The threshold of the measure (400 seconds) is indicated in the chart.

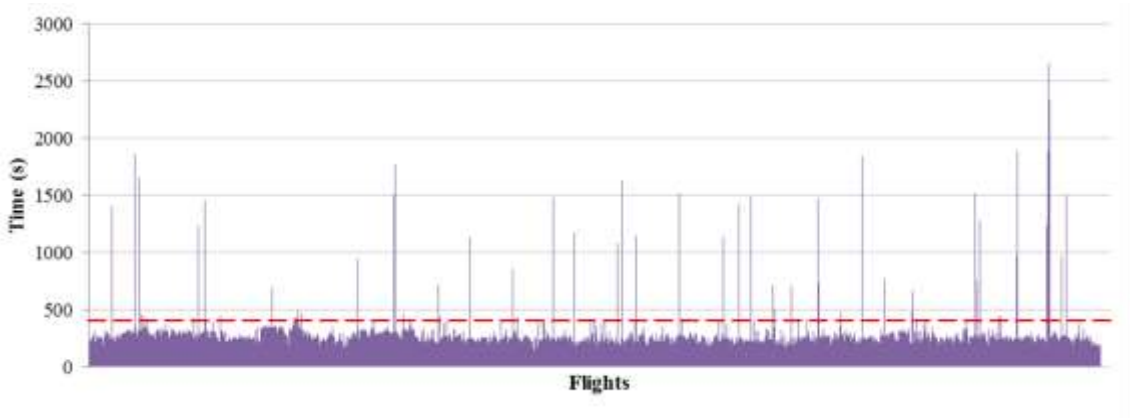

Figure 32 Time from Penetrating the Wireframe to Landing

Applying these rules, 35 go-arounds are detected using the criteria, which is an average of 4.3 per 1,000 flights. A screenshot for one of them is shown in Figure 33. The rate of go-arounds is much less than the rate of unstable approaches detected based on criteria and existing data. According to the standard operation procedures, a flight not stabilized at 1000'/500' AGL should abort the approach and execute a go-around. There are many possible explanations for this difference. One is that the aircraft stabilized at 1000'/500' AGL can turn back to unstable state during the course, for example, flying outside of the wireframe zone. The stabilization criteria used in this research capture the

performance *between* the stabilization altitude and the runway threshold. Nonetheless, the rate of go-arounds is still very small relative to the rate of unstable approaches. A similar finding was observed in Merritt et al. (2006) who found that only 5% of unstable approaches executed go-arounds. Similar overall go-around statistics appear in Sherry et al. (2013), where the rate of go-around is about 7 per 1,000 tracks.

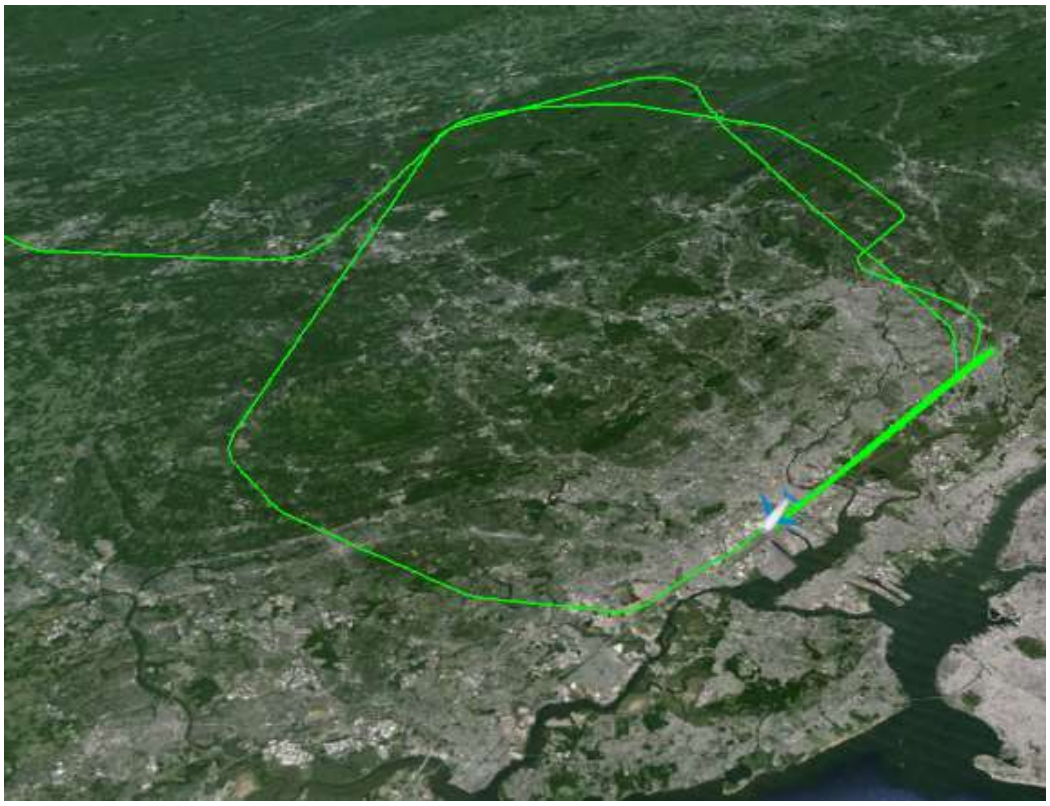

Figure 33 Go-around Track

4.3 Nowcasting Results – Conditional Probability Method

4.3.1 Nowcasting with Single State Variable

The nowcasting method of using conditional probability starts with using a single state variable, which is airspeed. Airspeed is divided into 8 bins, each bin having a width of 20 knots. If there are a sufficient number of samples for a specific condition (range of airspeed), and the confidence level is satisfactory, and $P\{U|v\} > 1.05*P\{U\}$ (where U represents an unstable approach event and v is the airspeed of a landing aircraft), then the condition is marked unstable. Table 15 presents example parameters for this model.

Table 15 Configurations for Example of Conditional Probability Method

Nowcasting location	6 nm
Target event	Unstable after 1000' AGL
Minimum number of samples for each condition	20
Confidence level threshold	±10%
Alerting ratio	1.05

The table of conditional probabilities are shown in Table 16. The red numbers indicate that the rows are not reliable for reporting due to lack of samples or confidence levels. The yellow-shaded cells represents the model output of unstable events, and green-shaded for stable events. Using this model, the performance measures (see Table 9 for definitions) are summarized below:

- Accuracy = 55.0%.
- Recall = 10.1%.

- Precision = 65.7%.
- F1 Score = 17.4%.
- 99.3% of all approaches can be nowcasted.

Table 16 Implementation of Single Variable Conditional Probability Table

Range (kts)	*n*	*CI*	*P*{U|*v*}	*P*{U}	Ratio
100 ~ 120	11	±28.4%	63.6%	47.3%	1.35
120 ~ 140	188	±7.1%	51.1%		1.08
140 ~ 160	1353	±2.7%	48.1%		1.02
160 ~ 180	4020	±1.5%	44.6%		0.94
180 ~ 200	2037	±2.2%	48.7%		1.03
200 ~ 220	403	±4.9%	55.6%		1.18
220 ~ 240	106	±9.1%	64.2%		1.36
240 ~ 260	33	±15.2%	72.1%		1.52

The performance of this model implies that only a small portion of unstable approaches can be nowcasted. And less than two-thirds of time the nowcast is correct. A model with more state variables are needed for better performances.

4.3.2 Nowcasting with Multiple State Variable

Building conditional probability tables with multiple state variables is now investigated. Table 17 gives an example lookup table focusing on large aircraft, with state variables collected at 6 nm from the runway threshold. Four state variables are selected: Airspeed, rate of descent, lateral deviation, and vertical deviation. Each state variable is categorized into three levels (e.g. low, normal, and high). The boundaries for categorizing

lateral and vertical deviations are based on the corresponding wireframe dimensions introduced in Section 4.1, specifically, ±475 ft for lateral deviation and ±275 for vertical deviation. The airspeed boundary for categorization is set as ±10 knots from the baseline value for large aircraft airspeed at 6 nm (174.9 knots). The boundary for rate of descent is set as ±300 ft/min from the baseline value (949.4 ft/min). The significance level α is set as 0.05 and the threshold for the confidence interval is set as ±15%.

Table 17 Conditional Probability Table for Nowcasting Unstable Approaches for Large Aircraft (at 6nm)

State Variables				1000' AGL		500' AGL	
Lateral Deviation	**Vertical Deviation**	**Airspeed**	**Rate of Descent**	**Probability**	**Confidence**	**Probability**	**Confidence**
left	low	low	low	N/A	N/A	N/A	N/A
left	low	low	normal	N/A	N/A	N/A	N/A
left	low	low	high	N/A	N/A	N/A	N/A
left	low	normal	low	N/A	N/A	N/A	N/A
left	low	normal	normal	N/A	N/A	N/A	N/A
left	low	normal	high	N/A	N/A	N/A	N/A
left	low	high	low	N/A	N/A	N/A	N/A
left	low	high	normal	N/A	N/A	N/A	N/A
left	low	high	high	N/A	N/A	N/A	N/A
left	normal	low	low	N/A	N/A	N/A	N/A
left	normal	low	normal	28.6%	16.7%	10.7%	11.5%
left	normal	low	high	N/A	N/A	N/A	N/A
left	normal	normal	low	N/A	N/A	N/A	N/A
left	normal	normal	normal	33.3%	14.8%	7.7%	8.4%
left	normal	normal	high	N/A	N/A	N/A	N/A
left	normal	high	low	N/A	N/A	N/A	N/A
left	normal	high	normal	51.4%	16.6%	17.1%	12.5%
left	normal	high	high	N/A	N/A	N/A	N/A
left	high	low	low	N/A	N/A	N/A	N/A
left	high	low	normal	66.7%	53.3%	33.3%	53.3%
left	high	low	high	N/A	N/A	N/A	N/A
left	high	normal	low	N/A	N/A	N/A	N/A
left	high	normal	normal	69.2%	25.1%	15.4%	19.6%

left	high	normal	high	N/A	N/A	N/A	N/A
left	high	high	low	N/A	N/A	N/A	N/A
left	high	high	normal	83.3%	17.2%	22.2%	19.2%
left	high	high	high	N/A	N/A	N/A	N/A
normal	low	low	low	N/A	N/A	N/A	N/A
normal	low	low	normal	0.0%	0.0%	0.0%	0.0%
normal	low	low	high	N/A	N/A	N/A	N/A
normal	low	normal	low	N/A	N/A	N/A	N/A
normal	low	normal	normal	N/A	N/A	N/A	N/A
normal	low	normal	high	N/A	N/A	N/A	N/A
normal	low	high	low	N/A	N/A	N/A	N/A
normal	low	high	normal	N/A	N/A	N/A	N/A
normal	low	high	high	N/A	N/A	N/A	N/A
normal	normal	low	low	N/A	N/A	N/A	N/A
normal	normal	low	normal	39.8%	2.5%	13.3%	1.7%
normal	normal	low	high	N/A	N/A	N/A	N/A
normal	normal	normal	low	N/A	N/A	N/A	N/A
normal	normal	normal	normal	35.6%	1.8%	13.3%	1.3%
normal	normal	normal	high	N/A	N/A	N/A	N/A
normal	normal	high	low	N/A	N/A	N/A	N/A
normal	normal	high	normal	56.7%	2.8%	14.7%	2.0%
normal	normal	high	high	N/A	N/A	N/A	N/A
normal	high	low	low	N/A	N/A	N/A	N/A
normal	high	low	normal	40.0%	15.2%	17.5%	11.8%
normal	high	low	high	N/A	N/A	N/A	N/A
normal	high	normal	low	N/A	N/A	N/A	N/A
normal	high	normal	normal	54.9%	9.7%	22.5%	8.1%
normal	high	normal	high	N/A	N/A	N/A	N/A
normal	high	high	low	N/A	N/A	N/A	N/A
normal	high	high	normal	78.5%	7.8%	28.0%	8.5%
normal	high	high	high	N/A	N/A	N/A	N/A
right	low	low	low	N/A	N/A	N/A	N/A
right	low	low	normal	0.0%	0.0%	0.0%	0.0%
right	low	low	high	N/A	N/A	N/A	N/A
right	low	normal	low	N/A	N/A	N/A	N/A
right	low	normal	normal	50.0%	49.0%	0.0%	0.0%
right	low	normal	high	N/A	N/A	N/A	N/A
right	low	high	low	N/A	N/A	N/A	N/A
right	low	high	normal	N/A	N/A	N/A	N/A
right	low	high	high	N/A	N/A	N/A	N/A
right	normal	low	low	N/A	N/A	N/A	N/A
right	normal	low	normal	62.5%	19.4%	20.8%	16.2%

right	normal	low	high	N/A	N/A	N/A	N/A
right	normal	normal	low	N/A	N/A	N/A	N/A
right	normal	normal	normal	78.9%	10.6%	42.1%	12.8%
right	normal	normal	high	N/A	N/A	N/A	N/A
right	normal	high	low	N/A	N/A	N/A	N/A
right	normal	high	normal	80.6%	12.9%	44.4%	16.2%
right	normal	high	high	N/A	N/A	N/A	N/A
right	high	low	low	N/A	N/A	N/A	N/A
right	high	low	normal	100.0%	0.0%	40.0%	42.9%
right	high	low	high	N/A	N/A	N/A	N/A
right	high	normal	low	N/A	N/A	N/A	N/A
right	high	normal	normal	100.0%	0.0%	20.0%	35.1%
right	high	normal	high	N/A	N/A	N/A	N/A
right	high	high	low	N/A	N/A	N/A	N/A
right	high	high	normal	77.8%	19.2%	33.3%	21.8%
right	high	high	high	N/A	N/A	N/A	N/A

For all these contents of the table, only the shaded cells contain sufficient and reliable results, i.e. with the normal sampling distribution criterion satisfied and with confidence intervals within ±15%. The large number of "N/A" cells in the table illustrates that few state-variable combinations can be nowcasted based on the current amount of data (about 8,000 tracks).

To use the information in the lookup table, it is necessary to refer to the analogous *unconditional* probability of an unstable approach. From results in Section 4.2.1, the unconditional probabilities of incurring unstable approach after 1000' and 500' AGL are 47.3%, and 17.5%. Compared with these baseline values, the indicated probability in each shaded cell (i.e., cells producing reliable results) can be evaluated for their significance. For example, in the case where every factor is within "normal" range (a row near the middle of the table), the probability of experiencing an unstable approach after

1000’ AGL has dropped from 47.3% to 35.6%. In another case, for the combination of states “normal, high, high, normal” (i.e. the flight is flying high and fast), the corresponding probability has increased from 47.3% to 78.5%, which indicates that this is a risky situation which could significantly increase the probability of an unstable approach. The flight crew who receives an alert message based on this data should take actions to adjust flight trajectories to reduce the potential risk of an unstable approach.

The performance of this model is presented below:

- Accuracy = 53.7%
- Recall = 24.9%
- Precision = 55.8%
- F1 Score = 34.4%
- 87.2% of all (Large) approaches can be nowcasted

The drawback of this method is obvious. To have enough information in each cell, the method needs a large amount of historical data. With th current limited amount of data, lookup tables have been generated with sparse cells containing reliable predicted probabilities. Also, the number of state-variable combinations grows dramatically in the number of state variables and the number of categories per variable. Even four variables with three categories yield a large fraction of non-reportable cells. Although more state variables may be helpful for better characterizing the situation of the aircraft (such as adding a wind variable), adding more variables and categories only makes this table

sparser. To overcome this limitation, nowcasting models using supervised learning are introduced in the next section.

4.4 Nowcasting Results – Supervised Learning Method

Nowcasting models based on supervised learning are developed. Features are carefully selected to represent the current state of the aircraft at the nowcasting location, the recent historical state of aircraft (from 12 nm to nowcasting location), and wind conditions. The performance of models trained with all features is discussed, which demonstrate better results over the conditional probability method. Also nowcasting using different feature sets are studied, showing the contributions of different information sets in predicting unstable approach events. The performance on sub-events is also studied. Finally, a sensitivity analysis is conducted based on the trained model using all samples to study the impact of each key feature on the probability of an unstable approach. Key factors with large impacts are identified.

4.4.1 Feature Selection

A total number of 22 basic features are selected for this research as the inputs to the logistic regression model. These basic features, summarized in Table 18, fall into four categories: aircraft parameters, current aircraft state, historical performance, and wind conditions. These features are designed to reflect information that is available to the landing aircraft at the nowcasting location.

Table 18 Selected Basic Features

Feature	Category	Description

x_1	Aircraft parameter	Maximum takeoff weight
x_2	Aircraft state prior to nowcast	Average airspeed at 12 nm
x_3	Aircraft state prior to nowcast	Altitude AGL at 12 nm
x_4	Aircraft state prior to nowcast	Angle difference between course and centerline at 12nm
x_5	Aircraft state prior to nowcast	Time spent in level flight segment
x_6	Aircraft state at nowcast	Current lateral deviation (negative if left of centerline)
x_7	Aircraft state at nowcast	Current lateral deviation (absolute value)
x_8	Aircraft state at nowcast	Current vertical deviation (negative if below glide-path)
x_9	Aircraft state at nowcast	Current vertical deviation (absolute value)
x_{10}	Aircraft state at nowcast	Current airspeed
x_{11}	Aircraft state at nowcast	Current airspeed deviation from baseline of corresponding weight class (absolute value)
x_{12}	Aircraft state at nowcast	Current rate of descent
x_{13}	Aircraft state at nowcast	Current rate of descent deviation from baseline of corresponding weight class (absolute value)
x_{14}	Aircraft state at nowcast	Current angle difference between course and centerline
x_{15}	Aircraft state prior to nowcast	Number of times penetrating wireframe before current location
x_{16}	Aircraft state prior to nowcast	Distance from lateral acquisition position to current position (nonnegative)
x_{17}	Aircraft state prior to nowcast	Distance from vertical acquisition position to current position (nonnegative)
x_{18}	Aircraft state prior to nowcast	Distance from deceleration point to current position (nonnegative)
x_{19}	Wind condition	Crosswind speed (negative for northwest wind)

x_{20}	Wind condition	Headwind
x_{21}	Wind condition	Gust
x_{22}	Wind condition	Crosswind speed magnitude (absolute value)

To capture the state of the aircraft at the current location, its speed, position, course, and rate of descent are measured. Also, the deviation of the descent rate from a baseline value based on the aircraft's weight class is also included. The former feature reflects the magnitude of descent rate, while the latter features represent a deviation from a baseline. A similar deviation is measured for airspeed. For positional features, the spirit is the same. Both lateral and vertical positions relative to the centerline and the absolute value of positional deviations are considered. Finally, the current angle difference between course and the runway centerline is included.

Historical behavior can also provide useful information of a target flight in addition to its current state. To reflect the recent historical behavior of a flight, the following features are extracted from processed track data: Level flight time before descending, average airspeed at 12 nm, altitude at 12 nm, course at 12 nm, number of times penetrating the wireframe, distance traveled between the lateral/vertical acquisition point and the current location, and distance traveled between the deceleration point and the current location. Level flight time is defined as the time a flight is between 2700' and 3300' AGL between 12 nm and 10 nm from the runway threshold. The number of times penetrating the wireframe before the current nowcasting location is a measure of variability of the flight trajectory. The lateral/vertical acquisition point is defined to be the first point the flight is laterally/vertically inside the wireframe zone. The deceleration

point is the point where the aircraft airspeed first drops 10 knots or more from its airspeed at 12 nm. For lateral/vertical acquisition and deceleration points, if the flight has not acquired these locations, the relative distances between the current location and these points are set to zero.

Finally, there are a set of features to reflect wind conditions. The wind data collected from METAR are processed and the crosswind, head wind, and gust are derived for the current runway. Four wind features are included: crosswind speed (with direction), crosswind speed magnitude, headwind, and gust.

In this research, the regression model introduced in next section sets the degree of the polynomial to be 2, i.e. the squared terms of basic features and potential interactions between basic features are derived based on the 22 basic features. This results in a total number of 275 features (22 basic features + 22 squared features + 22-choose-2 interaction features).

4.4.2 Nowcasting Performance – All Features

To achieve the best possible performance, all 22 basic features and their derived features are used to train the logistic regression model. Ten nowcasting locations are chosen for the study (from 3.5 nm to 10 nm). The outputs focus on three stabilization altitudes (1000', 750', and 500'). Seven unstable events are studied (one overall event, plus six sub-events). These result in 10*3*7 = 210 different models. To increase the reliability of results, ten replicates are conducted for each model (i.e. ten different random sets for training/testing). For each replicate of an experiment, 5,000 flights among the

8,158 flights are randomly selected to train the model. The remaining flights are used for testing.

A sample performance result from a single replicate is shown in Table 19, which is for nowcasting unstable approaches after 1000' AGL at 6 nm. For this single replicate, the performance measures for nowcasting unstable approaches are shown below:

- Total accuracy of the model: (962+1397)/(962+281+518+1397) = 74.7%;
- The recall of predicting unstable approaches: 962/(962+518) = 65.0%;
- The precision of predicting unstable approaches: 962/(962+281) = 77.4%;
- F1 score: 2*0.65*0.774/(0.65+0.774) = 70.7%.

Table 19 Sample Prediction Performance Summary

	Actual Unstable	Actual Stable
Predicted Unstable	962	281
Predicted Stable	518	1397

The averaged performance measures for all ten replicates are shown in Figure 34 (nowcasting unstable approaches after 1000'). The figure shows the mean values of accuracy, precision, recall, and F1 Score at each nowcasting location. The nowcast performance improve as the flights progress from 10 nm to 3.5 nm, getting closer to the stabilization altitude 1000' AGL. The maximum measures are achieved at 3.5 nm, with the average F1 Score equal to 80.3%, recall equal to 75.3% and precision equal to 85.9%.

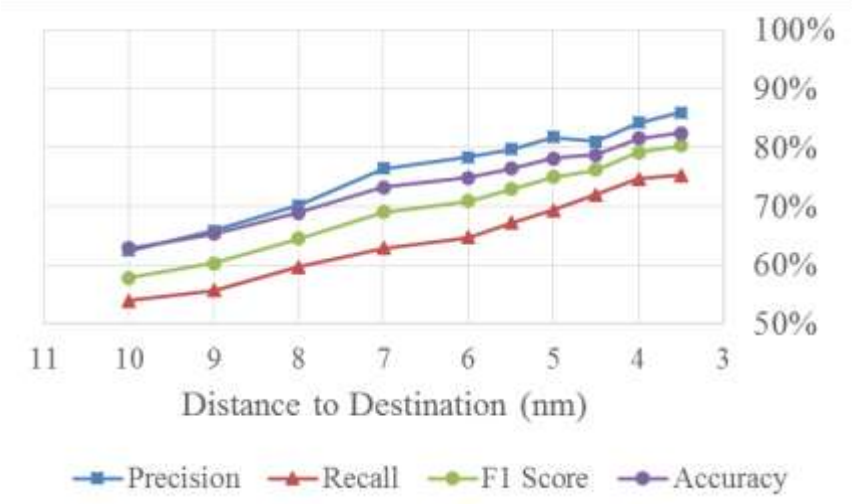

Figure 34 Performance Measures at Different Nowcast Locations

The nowcast performance at 10 nm is less powerful. Specifically, the nowcast at 10 nm can correctly identify 53.9% of unstable approaches below 1000' AGL. The predicted results are correct 62.5% of time. The low values in the performance measures are due to the uncertainties between 10 nm and 1000' AGL. The nowcast performance improves as the flights progress from 10 nm to 6 nm to 3 nm. At 6 nm, the proportion of correctly predicted unstable approaches improves from 53.9% to 64.7%. The probability of giving an incorrect nowcast decreases from 37.5% to 21.6%. At 3.5 nm, where the flight is very close to the stabilization altitude of 1000' AGL, the proportion of unstable flights which can be correctly predicted rises to 75.3%, and the correct prediction rate for unstable approaches increases to 85.9%.

The prediction performance for unstable approaches below 500' AGL is less satisfactory compared to prediction of the 1000' AGL event. The performance measures for predicting sub-events such as short acquisition and unstable speed is also lower

compared to the overall unstable event. This is because for the 500' AGL and sub-events there are fewer positive samples (i.e. unstable flights) for training the model. Figure 35 shows the performance of nowcasting the sub-event of unstable speed (deceleration). To improve the performance, more historical data are needed to include sufficient positive samples for model training.

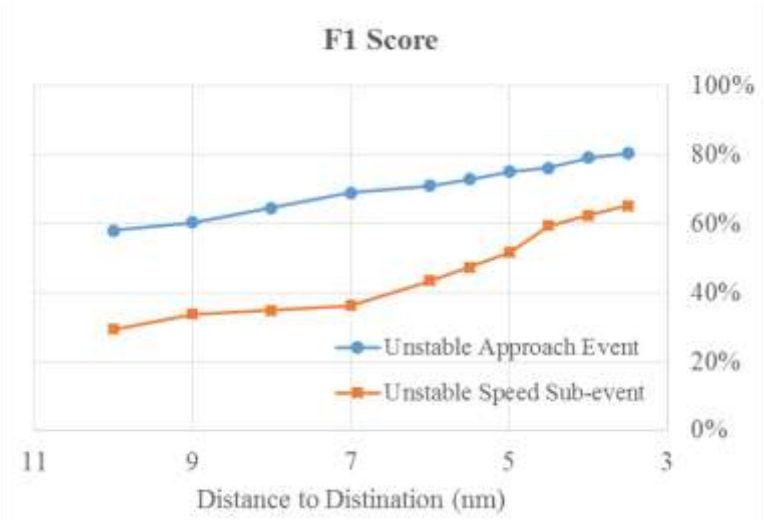

Figure 35 Nowcasting Performance on Sub-event

Finally, to accept the nowcast results, decision makers need to set a threshold for the F1 Score, e.g., 75%, in order to consider use of the nowcast. With a nowcast of a potential unstable approach, the flight crew will have lead time to correct the aircraft trajectory.

4.4.3 Contributions of Wind and Aircraft Historical Performance Information

In the previous section, all features are used for training the nowcasting model. If only a subset of the features are used, their effects on predicting unstable approaches can

be shown. Figure 36 shows the F1 Score of models trained using the current aircraft state (10 features), current state + wind conditions (14 features), and current state + wind conditions + historical performance (22 features). The contribution of wind information and the aircraft historical performance information can be observed. At further distances (e.g. beyond 7 nm), adding wind information can improve the F1 Score by about 5 percentage points. Adding historical features can further improve the performance by 5 to 10 percentage points. The improvements are more significant at further distances.

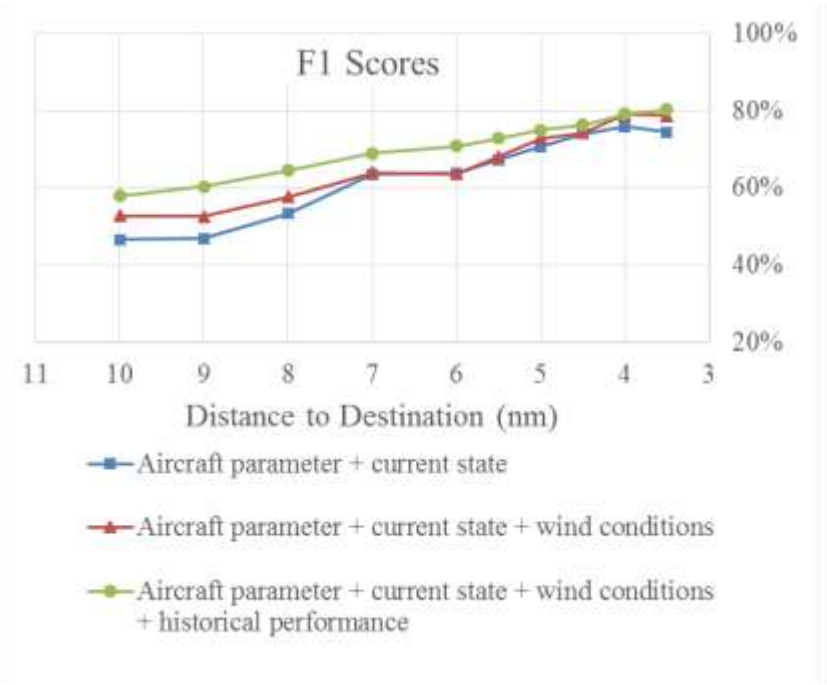

Figure 36 Contributions from Wind Information and Aircraft Historical Performance Information

4.4.4 Feature Sensitivity

In this section, each factor's impact on the probability of an unstable approach is studied using the trained logistic regression model using the entire samples of 8,158 flights. Among the basic 22 features, 5 of them are dependent on the other features, and the other 17 basic features can be treated as independent features. Among these 17 features, each one is varied from -1 to 1 on a normalized scale (i.e., changing from μ - σ to μ + σ, where μ is the sample mean of the feature based on the 8,158 sample flights and σ is the standard deviation of the feature), while the other 16 features are kept at their mean values. All other dependent features including squared terms and interaction terms are updated accordingly. The corresponding change in the probability of an unstable approach after 1000' AGL (given in Equation 8) is recorded and used as a measure for the impact of the factor. A larger magnitude means a higher impact. A positive value means that increasing the feature value will increase the probability of an unstable approach.

Figure 37 shows the results of the factor impacts. From this chart, the features with impact higher than 10% (in magnitude) are airspeed at 12 nm, current lateral deviation, current airspeed, current crosswind speed, and current angle difference between course and runway centerline. Specifically, varying the current airspeed by increasing from its mean minus one standard deviation (174.0-17.8 knots) to its mean plus one standard deviation (174.0+17.8 knots) increases the probability of experiencing an unstable approach below 1000' AGL from 22.1% to 66.6%.

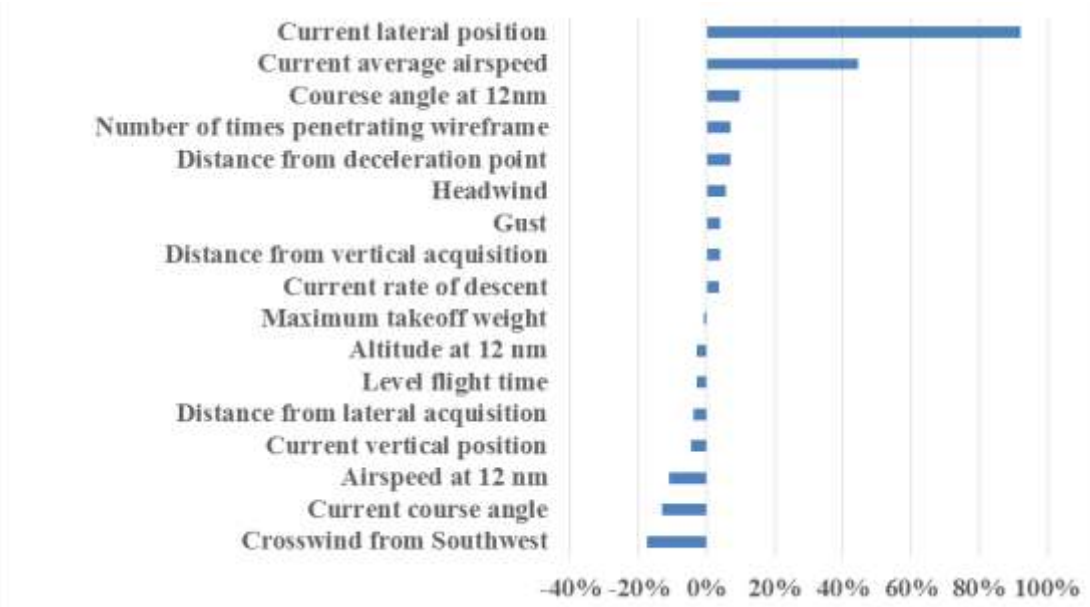

Figure 37 Sensitivity Analysis for Quantifying Factor Impacts

A limitation of this sensitivity analysis is the need for the assumption of independence between the basic features which are tuned. In reality this is not true (e.g., changing the airspeed at 12 nm may affect the airspeed at the nowcasting location). However, based on the trained model, the results still can provide important information on how each factor could affect the probability of an unstable approach.

4.4.5 Trade-off Analysis

In previous models, the discrimination threshold is all set to 0.5, i.e. a test sample is predicted to be positive if and only if the hypothesis function $h_\theta(x)$ returns a value higher than 0.5. This parameter can be tuned for different considerations of modeling and hence further improve the model. A higher discrimination usually leads to higher precision of prediction, but may reduce the recall, i.e. the portion of actual unstable approaches that can be nowcasted. A lower threshold can lead to higher recall with lower

precision. In summary, in this study, a lower precision indicates an increased number of false alarms, while a lower recall indicates an increased amount of missed detections. The trade-off between these results is one of the major concerns in future deployment of this nowcasting system. The precision and recall values for nowcasting at 10 nm using different discrimination threshold is shown in Figure 38, and the trade-off of precision and recall is shown in Figure 39. The purpose is to move the curves in a direction towards the up-right corner.

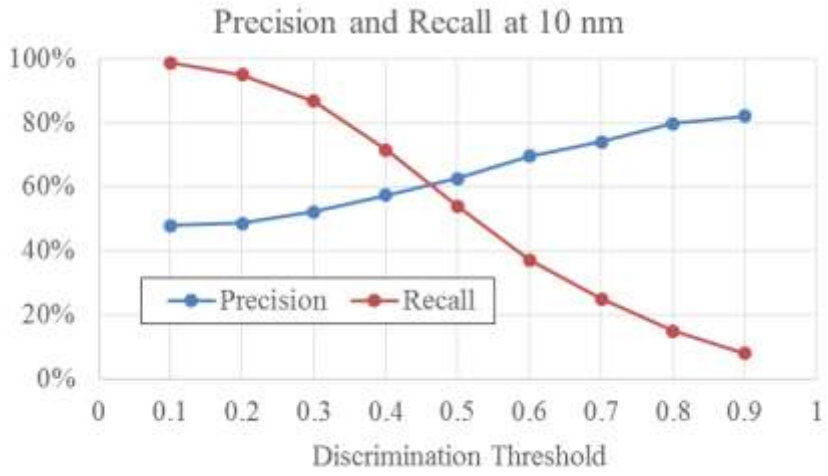

Figure 38 Precisoin and Recall at 10 nm Given Different Discrimination Thresholds

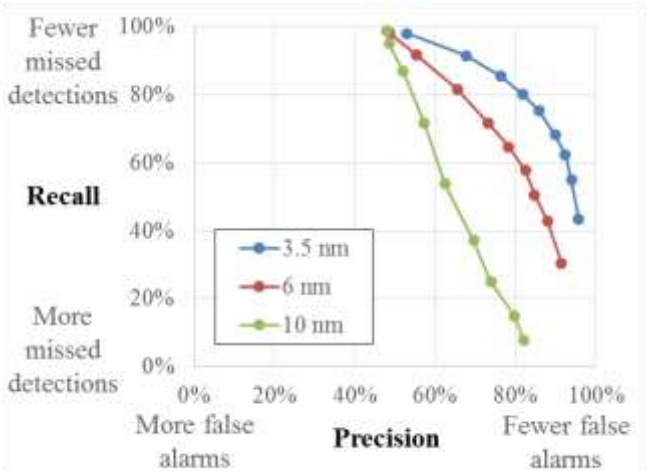

Figure 39 Trade-off between Precision and Recall

4.5 Computational Performance

This research uses a computer with 2.5GHz CPU, 16G RAM, and 64-bit OS. The track processing, event labeling, and conditional probability method are implemented in C++ with about 5,500+ lines of codes (Appendix A) to fulfill the tasks of processing 28 days' flights. It takes approximately 2.5 hours to finish processing tracks and identifying unstable events.

The supervised learning method is implemented in MATLAB. The code for this method is attached in Appendix B. For each replicate of an experiment, the code randomly chooses 5,000 flights for training and the remaining 3,158 flights are used for testing. For each replicate of an experiment, it takes approximately 2 hours of training/testing with all features at all locations.

Although the training phase of the model can take time, execution of the model is quick, so real-time prediction can be achieved. Aircraft with the nowcasting system

onboard can use real-time sensor data to feed the model and the nowcasting results can be instantly available.

CHAPTER 5: CONCLUSIONS

5.1 Summary

This dissertation developed methods for nowcasting unstable approaches prior to the stabilization altitude using historical flight track data, wind data, navigation procedures, and aircraft parameters. The algorithms for detecting unstable approaches focused on applying the stabilized approach criteria: runway centerline alignment, glide-path alignment, rate of decent range, and change in airspeed. The nowcast was able to predict unstable approaches prior to 1000' AGL based on information available to the aircraft at its current location.

This dissertation first addressed the question of how to define and detect unstable approaches from publicly available data sets. First, preprocessing procedures were conducted on surveillance track data and wind data to work with limited precision levels/sampling rates in the raw data. Navigation procedures were extracted from published diagrams/runway parameters and wireframe structures were built to model the approach region of landing aircraft. Algorithms for filtering qualified tracks were developed, which were critical in developing accurate statistics. Stabilized approach criteria were applied to identify unstable approach events using integrated data sets.

Then, the statistics of detected unstable approach events were obtained using the developed algorithms. The results indicated that 47.3% of flights did not meet all the stabilized approach criteria after reaching 1000' AGL. This percentage of unstable

approaches dropped to 33.6% for flights after reaching 750’ AGL, and to 17.5% for flights after reaching 500’ AGL. The proportion of detected go-around was much smaller than that of unstable approaches. This difference may come from the fact that this research uses a much stricter set of criteria, for example, looking at *all* states between the stabilization altitude and the runway threshold crossing point, instead of looking just at the state at the stabilization altitude.

The research focused on nowcasting the potential unstable approaches by calculating the probabilities of target events for similar historical states of aircraft. The first method of nowcasting applied conditional probability tables for outputting the corresponding probability of unstable approach events under different input conditions. If the output probability is significantly higher than the unconditional probability, the result will be reported to alert the user. With a large number of different combinations of input variables, this method needs a lot of historical data to generate predicted probability values with a higher confidence level, which is impractical for this study. To improve the nowcast, supervised learning methods were adopted. With selected features, the models were trained using a portion of historical data. The remaining data were used for testing the performance of the models. For both methods, measures of prediction performance were studied. The findings are summarized in the next section.

In summary, this research presents a proof-of-concept analysis. The focus is on the feasibility of using historical data to identify and predict unstable approaches. With larger data sets and better quality data (e.g., FOQA data), the results may improve significantly over the results reported here. More information will help in identifying

unstable approaches and improving the current nowcasting performance, making the model more applicable in the real world.

5.2 Key Findings and the Implications of the Case Study

Key findings in detecting unstable approach events are summarized below:

- 47.3% of landing flights experienced unstable approach after reaching 1000’ AGL, and this percentage drops to 17.5% after 500’ AGL.
- 26.8% of landing flights experienced an excessive decrease in airspeed from 1000’ AGL to runway threshold, and this percentage drops to 4.1% after 500’ AGL.
- 19.5% of landing flights acquired runway centerline from right side after 1000’ AGL, and this percentage drops to 8.5% after 500’ AGL.

These percentages are higher than previous estimates in the literature. This can be explained as follows:

1. The criteria applied focused on the behavior of aircraft *between* the stabilization altitude and the runway threshold crossing point. In the literature, the estimated results are based on published criteria which are only checked at or before the stabilization altitudes.
2. This research did not differentiate between approach types (ILS vs. Visual approach) and used 1000’ AGL as a baseline for all results. In a real case, under VMC, 500’ AGL is the check point instead of 1000’ AGL.
3. This research used publicly available data which is somewhat noisy and has a low sampling rate. The objective was to demonstrate the feasibility

of detecting and nowcasting unstable approaches. Results may improve with better data quality.

An example summary of nowcasting performances at 6 nm using different methods is shown in Table 20.

Table 20 Summary of Nowcasting Performances at 6 nm

Method	Accuracy	Precision	Recall	F1 Score	Applicable flights
Conditional Probability (single state variable)	55.0%	65.7%	10.1%	17.4%	99.3%
Conditional Probability (four state variables, Large)	53.7%	55.8%	24.9%	34.4%	87.2%
Supervised Learning (current aircraft state)	70.5%	75.4%	55.2%	63.8%	100%
Supervised Learning (current aircraft state + historical aircraft state + wind conditions)	74.8%	78.4%	64.7%	70.9%	100%

The prediction performance for 500’ AGL is relatively lower due to uncertainties between 1000’ AGL and 500’ AGL, and fewer samples for training and testing. The nowcasting performance for sub-events were also weak compared to predicting the overall unstable approach event. This problem is also due to the limited number of samples. With a larger amount of data, the performance of the model is expected to improve.

The key factors that have a relatively large impact on the unstable approach probability were identified. At 6 nm, these factors include airspeed at 12 nm, current

lateral deviation, current airspeed, and current crosswind speed. For example, varying the current airspeed by increasing it from 156.2 knots to 191.8 knots will increase the probability of experiencing an unstable approach after 1000' AGL from 22.1% to 66.6% (as predicted by the model).

The contributions of this research include:

- Developing a methodology for identifying unstable approaches from integrated data of surveillance tracks, wind conditions, and navigation procedures.
- Developing nowcasting models for real-time prediction of unstable approaches at locations prior to the stabilization altitudes.

The advantages of applying the methodology in this research include:

- The use of historical flight track data for prediction. Surveillance tracks implicitly account for external factors, like aircraft configurations and ATC instructions, that are challenging to model directly using standard aircraft peroformance models.
- Real-time prediction.
- Scalable. More features and other data sources can be included to enrich the input information and improve the performance of current models.

5.3 Limitations of Current Methodology

There are inherent limitations of the current approach for studying unstable approaches. First, the nowcasting is not designed for a key causal factor analysis. Current model outputs give the predicted results based on aircraft states for which the deviations

of the variables are not too far from what is observed. (because the prediction models are trained using only observed data). The models are not for “manipulation”, i.e. manually adjusting the input factor values to “achieve” some level of probability. Correlations exist between input variables. In summary, the model outputs are based on what have been observed during the model training phase.

Second, in preprocessing the tracks, the ground speed and rate of descent are smoothed using averaging windows extending both before and after the current track point. This means that some future information is used for prediction. Currently the averaging interval is set to ±15 seconds for both the ground speed and rate of descent. In the future, if data sets of higher quality are used, this problem can be solved.

Third, the causes of unstable approaches can be complex. Not all of them originate prior to the stabilization altitudes. There can be uncertainties which cannot be predicted at an earlier stage as in the current model. Also, not all factors can be reflected in the data used, for example, problems in aircraft configuration cannot be revealed using current surveillance track data with a limited number of fields.

Fourth, the current nowcasting model does not provide a recommendation to the flight crew to adjust the trajectory to avoid an unstable approach. The correlations between factors must be considered. This can be done in future research. Bayesian Network models can be applied to identify the key factors and to study the action needed.

Finally, the limitations of the current methodology lie in the data quality. For example, the surveillance track data used did not provide sufficient data to assess the flap/slat and landing configuration of the aircraft. Therefore, the aircraft configuration

requirements in the stabilized approach criteria were temporarily not included in this research. This quality issue can be solved in the future work by using onboard recorded data which have higher fidelity and more fields. The results from this research using flight track data with basic fields formed a baseline for the nowcast prediction.

5.4 Future Work

This research is a starting point to make the nowcasting system practical in a real world application (Figure 40). Several future research topics are summarized below.

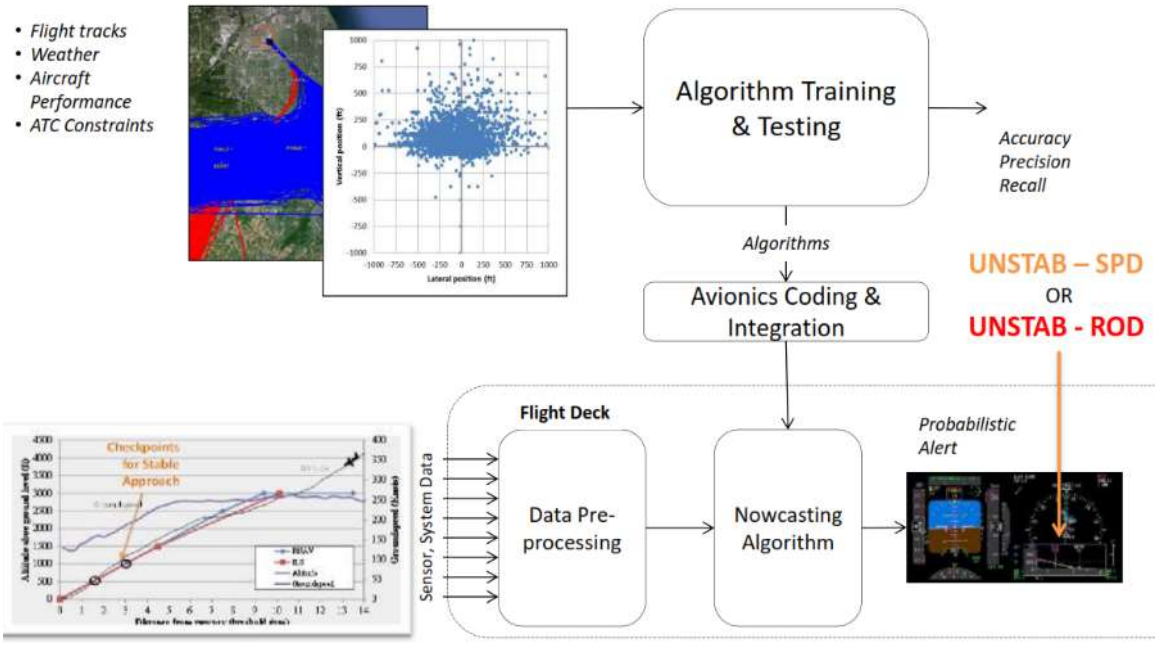

Figure 40 Framework for Nowcasting Operations

First, the performance of prediction can be improved. Current nowcasting performance is a baseline for future improvements. The prediction power of unstable approaches is weak at distances beyond 6 nm. Also, the performance is less satisfactory

for nowcasting the sub-events due to the limited number of training samples. To improve the performance, the following actions can be taken:

- Using data with higher accuracy and precision levels.
- Using data with more fields (e.g. onboard data) to add more features.
- Increasing the amount of samples for training and testing the model.

A key effort is to reduce the Type I and Type II errors, which are the probabilities of a false alert and a missed detection. Key questions are about accuracy and precision levels required for input flight track data to achieve a satisfactory outcome. What should the baseline sampling rate of the raw data be? How do the precision levels affect the prediction performance? These questions need to be answered in future study.

Second, it will be useful to study the application of the methodology at other runways at EWR and other airports. With different approach procedures and flight track patterns, the model parameters to be trained can vary. It will be insightful to compare the performances of the nowcast models at different runways and airports. Factors such as different approach types can be studied for their impact on unstable approaches.

Third, it is necessary to establish a decision threshold to accept or deny a nowcast result. An example of such threshold is a minimum value of the F1 Score (e.g. 70%). The value of this decision boundary needs to be determined through further studies. Constantly receiving insignificant alerts will distract the attention of flight crew, while neglecting important alert messages may lead to unstable approach which could have been avoided by accepting the nowcast results.

Furthermore, the system should be validated. One form of validation is to embed the nowcast system into some flight simulator to test the system by manually flying approach flights. The purpose is to test the hypothesis that fewer unstable approaches are experienced by flight crews using the nowcasting system compared to flight crews without the system. Even if the system performance for aiding the flight crew is validated and the concept of this research is approved by stakeholders, there will still be a need to study the interface design with the current FMS. Human factors need to be considered to guarantee the efficiency of using this embedded system. Although the system is expected to effectively reduce the chance of an unstable approach for a landing flight, the ultimate goal is to improve the *overall* safety margin of the flights during the approach and landing phase. Introducing a solution to an existing problem may lead to other unexpected problems. Interactions between components and integration of the systems must be studied to improve the overall safety.

APPENDIX A: C++ CODE FOR DATA PROCESSING AND UNSTABLE EVENTS IDENTIFICATION

Header file

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <list>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
//Constants
static const double PI = 3.14159265;
static const double ft2m = 0.3048;
static const double m2ft = 3.28084;
static const double nm2m = 1852.0;
static const double m2nm = 1.0/1852.0;
static const double nm2ft = 6076.11549;
static const double ft2nm = 1.0/6076.11549;
static const double kg2lb = 2.205;
static const double lb2kg = 1.0/2.205;
static const double kts2mps = 0.51444;
static const double mps2kts = 1.0/0.51444;
static const double mps2fpm = m2ft * 60.0;
static const double fpm2mps = ft2m/60.0;
static const double deg2rad = 3.14159265/180.0;
static const double rad2deg = 180.0/3.14159265;

const int regionID = 2; //2 for NYC, 6 for Chicago
const string CURRENTDATA = "N90"; //data sources: "N90", "PDARS", "C90",
"C90_Southwest"
const string CURRENTAIRPORT = "EWR"; //"EWR","JFK","LGA", "MDW"
const int N_DAYS = 1; //N90: 28 days, C90: 21 days;
const string DATES[] = {
       //N90
       "20070111"//,"20070112","20070113","20070114","20070115","20070116","2007
0117",
       //"20070323","20070324","20070325","20070326","20070327","20070328","2007
0329",
       //"20070721","20070722","20070723","20070724","20070725","20070726","2007
0727",
       //"20071001","20071002","20071003","20071004","20071005","20071006","2007
1007"
       //PDARS
       //"20080205", "20080319", "20080331", "20080410", "20080623", "20080723"
       //C90
```

```
      //"20110117", "20110118", "20110124", "20110125", "20110126", "20110127",
"20110215",
      //"20110322", "20110323", "20110426", "20110509", "20110510", "20110529",
"20110610",
      //"20110611", "20110615", "20110620", "20110701", "20111214", "20120122",
"20120123"
};
//wireframe dimensions
const double REF2THR_DIST = 6.0*nm2m;
const double FAF2THR_DIST = 10.0*nm2m;
const double LATERALHALFWIDTH_THR = 135.0*ft2m;
const double VERTICALHALFWIDTH_THR = 80.0*ft2m;
const double LATERALHALFWIDTH_REF = 475.0*ft2m;
const double VERTICALHALFWIDTH_REF = 275.0*ft2m;
const double LATERALHALFWIDTH_12nm = LATERALHALFWIDTH_REF;
const double VERTICALHALFWIDTH_12nm = VERTICALHALFWIDTH_REF;
const double RELAXEDHALFWIDTH = 5000.0*ft2m;
const double LEVEL_FLIGHT_ALTITUDE = 3000.0*ft2m; //for EWR 22L

const bool PROJECTING = false;
const double TIMESTEP = 1.0; //time step for linear interpolation
const int AVGBIN = 30; //averaging time interval for smoothing groundspeed and
rate of descent; even number
const double CUMUTHRESH = 330*deg2rad; //cumulative turn angle threshold
const int j_default = -1; //default track point index
const double BIG_DIST_default = 99999.0;
const double TIMELIMIT = 400.0;

const double dist_traveled_inside_threshold = 50.0;
const double time_traveled_inside_threshold = 3.0;
const double angle_diff_threshold = PI/4.0;
const double dist2THR_threshold = 3.0*nm2m;

//parameters for track-qualification
const double starting_min_altitude_AGL = 3000.0 * ft2m;
const double ending_max_altitude_AGL = 500.0 * ft2m;
const double starting_min_dist_to_THR = 12.0 * nm2m;
const double ending_max_dist_to_THR = 3.0 * nm2m;

//unstable approach criteria parameters
const double time_period_for_ROD_event = 10.0;
const double ROD_event_threshold = -1000.0*fpm2mps;
const double unstablespeed_toofast_thresohld = 10.0*kts2mps;
const double unstablespeed_tooslow_thresohld = -10.0*kts2mps;
const string runwayname_default = "not identified";
const string flow_default = "default flow"; //e.g. from east/west
const string procedure_default = "default procedure"; //VFR, RNP, ILS

const string STATECATEGORY_lateraldev_left = "left";
const string STATECATEGORY_lateraldev_normal = "normal";
const string STATECATEGORY_lateraldev_right = "right";
const string STATECATEGORY_verticaldev_high = "high";
const string STATECATEGORY_verticaldev_normal = "normal";
const string STATECATEGORY_verticaldev_low = "low";
const string STATECATEGORY_avgspeed_high = "high";
const string STATECATEGORY_avgspeed_normal = "normal";
const string STATECATEGORY_avgspeed_low = "low";
const string STATECATEGORY_avgROD_high = "high";
const string STATECATEGORY_avgROD_normal = "normal";
const string STATECATEGORY_avgROD_low = "low";
```

```
const string STATECATEGORY_headingdiff_clockwise = "clockwise";
const string STATECATEGORY_headingdiff_normal = "normal";
const string STATECATEGORY_headingdiff_counterclockwise = "counterclockwise";
const string AIRCRAFT_ICON =
"https://dl.dropboxusercontent.com/u/22140462/plane-icon.png";

//(+-)boundaries for state variable category; hardcoded for REF at 6nm, update
boundaries when REF changes
const double LAT_DEV_BOUNDARY_10nm = LATERALHALFWIDTH_REF;
const double LAT_DEV_BOUNDARY_9nm = LATERALHALFWIDTH_REF;
const double LAT_DEV_BOUNDARY_8nm = LATERALHALFWIDTH_REF;
const double LAT_DEV_BOUNDARY_7nm = LATERALHALFWIDTH_REF;
const double LAT_DEV_BOUNDARY_6nm = LATERALHALFWIDTH_REF;
const double LAT_DEV_BOUNDARY_5dot5nm = LATERALHALFWIDTH_THR +
(5.5*nm2m/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR);
const double LAT_DEV_BOUNDARY_5nm = LATERALHALFWIDTH_THR +
(5.0*nm2m/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR);
const double LAT_DEV_BOUNDARY_4dot5nm = LATERALHALFWIDTH_THR +
(4.5*nm2m/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR);
const double LAT_DEV_BOUNDARY_4nm = LATERALHALFWIDTH_THR +
(4.0*nm2m/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR);
const double LAT_DEV_BOUNDARY_3dot5nm = LATERALHALFWIDTH_THR +
(3.5*nm2m/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR);
const double LAT_DEV_BOUNDARY_3nm = LATERALHALFWIDTH_THR +
(3.0*nm2m/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR);
const double VERT_DEV_BOUNDARY_10nm = VERTICALHALFWIDTH_REF;
const double VERT_DEV_BOUNDARY_9nm = VERTICALHALFWIDTH_REF;
const double VERT_DEV_BOUNDARY_8nm = VERTICALHALFWIDTH_REF;
const double VERT_DEV_BOUNDARY_7nm = VERTICALHALFWIDTH_REF;
const double VERT_DEV_BOUNDARY_6nm = VERTICALHALFWIDTH_REF;
const double VERT_DEV_BOUNDARY_5dot5nm = VERTICALHALFWIDTH_THR +
(5.5*nm2m/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR);
const double VERT_DEV_BOUNDARY_5nm = VERTICALHALFWIDTH_THR +
(5.0*nm2m/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR);
const double VERT_DEV_BOUNDARY_4dot5nm = VERTICALHALFWIDTH_THR +
(4.5*nm2m/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR);
const double VERT_DEV_BOUNDARY_4nm = VERTICALHALFWIDTH_THR +
(4.0*nm2m/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR);
const double VERT_DEV_BOUNDARY_3dot5nm = VERTICALHALFWIDTH_THR +
(3.5*nm2m/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR);
const double VERT_DEV_BOUNDARY_3nm = VERTICALHALFWIDTH_THR +
(3.0*nm2m/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR);
const double AVG_AIRSPEED_BOUNDARY = 10*kts2mps;
const double AVG_ROD_BOUNDARY = 300*kts2mps;
const double HEADING_DIFF_BOUNDARY = 20*deg2rad; //currently not in lookup
table

//baseline avg_v and avg_ROD of aircraft in different weight classes //values
are based on observation of 28days' approaches to EWR 22L
//Heavy
const double BASELINE_AVG_AIRSPEED_10nm_Heavy = 189.0482903*kts2mps;
const double BASELINE_AVG_AIRSPEED_9nm_Heavy = 182.5649132*kts2mps;
const double BASELINE_AVG_AIRSPEED_8nm_Heavy = 177.9601436*kts2mps;
const double BASELINE_AVG_AIRSPEED_7nm_Heavy = 174.1863833*kts2mps;
const double BASELINE_AVG_AIRSPEED_6nm_Heavy = 170.6060506*kts2mps;
const double BASELINE_AVG_AIRSPEED_5dot5nm_Heavy = 169.2260537*kts2mps;
const double BASELINE_AVG_AIRSPEED_5nm_Heavy = 165.1870847*kts2mps;
const double BASELINE_AVG_AIRSPEED_4dot5nm_Heavy = 161.4124267*kts2mps;
const double BASELINE_AVG_AIRSPEED_4nm_Heavy = 154.7638347*kts2mps;
const double BASELINE_AVG_AIRSPEED_3dot5nm_Heavy = 149.9245258*kts2mps;
```

```
const double BASELINE_AVG_AIRSPEED_3nm_Heavy = 143.3753461*kts2mps;
const double BASELINE_AVG_AIRSPEED_1000ftAGL_Heavy = 140.2110658*kts2mps;
const double BASELINE_AVG_AIRSPEED_750ftAGL_Heavy = 138.0675971*kts2mps;
const double BASELINE_AVG_AIRSPEED_500ftAGL_Heavy = 134.9320217*kts2mps;
const double BASELINE_AVG_ROD_10nm_Heavy = -247.09*fpm2mps;
const double BASELINE_AVG_ROD_9nm_Heavy = -477.88*fpm2mps;
const double BASELINE_AVG_ROD_8nm_Heavy = -738.66*fpm2mps;
const double BASELINE_AVG_ROD_7nm_Heavy = -882.65*fpm2mps;
const double BASELINE_AVG_ROD_6nm_Heavy = -922.14*fpm2mps;
const double BASELINE_AVG_ROD_5dot5nm_Heavy = -894.06*fpm2mps;
const double BASELINE_AVG_ROD_5nm_Heavy = -876.76*fpm2mps;
const double BASELINE_AVG_ROD_4dot5nm_Heavy = -857.65*fpm2mps;
const double BASELINE_AVG_ROD_4nm_Heavy = -825.69*fpm2mps;
const double BASELINE_AVG_ROD_3dot5nm_Heavy = -785.31*fpm2mps;
const double BASELINE_AVG_ROD_3nm_Heavy = -757.81*fpm2mps;
const double BASELINE_AVG_ROD_1000ftAGL_Heavy = -754.62*fpm2mps;
const double BASELINE_AVG_ROD_750ftAGL_Heavy = -741.23*fpm2mps;
const double BASELINE_AVG_ROD_500ftAGL_Heavy = -753.82*fpm2mps;
//B757
const double BASELINE_AVG_AIRSPEED_10nm_B757 = 184.3648396*kts2mps;
const double BASELINE_AVG_AIRSPEED_9nm_B757 = 180.3046609*kts2mps;
const double BASELINE_AVG_AIRSPEED_8nm_B757 = 177.9726657*kts2mps;
const double BASELINE_AVG_AIRSPEED_7nm_B757 = 175.6919318*kts2mps;
const double BASELINE_AVG_AIRSPEED_6nm_B757 = 172.9729731*kts2mps;
const double BASELINE_AVG_AIRSPEED_5dot5nm_B757 = 171.2753948*kts2mps;
const double BASELINE_AVG_AIRSPEED_5nm_B757 = 167.1680826*kts2mps;
const double BASELINE_AVG_AIRSPEED_4dot5nm_B757 = 162.5272622*kts2mps;
const double BASELINE_AVG_AIRSPEED_4nm_B757 = 154.9754707*kts2mps;
const double BASELINE_AVG_AIRSPEED_3dot5nm_B757 = 147.9847224*kts2mps;
const double BASELINE_AVG_AIRSPEED_3nm_B757 = 140.0977925*kts2mps;
const double BASELINE_AVG_AIRSPEED_1000ftAGL_B757 = 135.3356798*kts2mps;
const double BASELINE_AVG_AIRSPEED_750ftAGL_B757 = 131.2019039*kts2mps;
const double BASELINE_AVG_AIRSPEED_500ftAGL_B757 = 128.763931*kts2mps;
const double BASELINE_AVG_ROD_10nm_B757 = -270.32*fpm2mps;
const double BASELINE_AVG_ROD_9nm_B757 = -455.46*fpm2mps;
const double BASELINE_AVG_ROD_8nm_B757 = -726.54*fpm2mps;
const double BASELINE_AVG_ROD_7nm_B757 = -895.03*fpm2mps;
const double BASELINE_AVG_ROD_6nm_B757 = -939.46*fpm2mps;
const double BASELINE_AVG_ROD_5dot5nm_B757 = -908.93*fpm2mps;
const double BASELINE_AVG_ROD_5nm_B757 = -894.11*fpm2mps;
const double BASELINE_AVG_ROD_4dot5nm_B757 = -871.46*fpm2mps;
const double BASELINE_AVG_ROD_4nm_B757 = -843.30*fpm2mps;
const double BASELINE_AVG_ROD_3dot5nm_B757 = -792.96*fpm2mps;
const double BASELINE_AVG_ROD_3nm_B757 = -747.07*fpm2mps;
const double BASELINE_AVG_ROD_1000ftAGL_B757 = -731.31*fpm2mps;
const double BASELINE_AVG_ROD_750ftAGL_B757 = -710.38*fpm2mps;
const double BASELINE_AVG_ROD_500ftAGL_B757 = -722.62*fpm2mps;
//Large
const double BASELINE_AVG_AIRSPEED_10nm_Large = 188.1501885*kts2mps;
const double BASELINE_AVG_AIRSPEED_9nm_Large = 183.8682213*kts2mps;
const double BASELINE_AVG_AIRSPEED_8nm_Large = 180.9792332*kts2mps;
const double BASELINE_AVG_AIRSPEED_7nm_Large = 178.2615607*kts2mps;
const double BASELINE_AVG_AIRSPEED_6nm_Large = 174.9038329*kts2mps;
const double BASELINE_AVG_AIRSPEED_5dot5nm_Large = 172.6490606*kts2mps;
const double BASELINE_AVG_AIRSPEED_5nm_Large = 168.2347767*kts2mps;
const double BASELINE_AVG_AIRSPEED_4dot5nm_Large = 162.4972751*kts2mps;
const double BASELINE_AVG_AIRSPEED_4nm_Large = 154.5541553*kts2mps;
const double BASELINE_AVG_AIRSPEED_3dot5nm_Large = 147.2779017*kts2mps;
const double BASELINE_AVG_AIRSPEED_3nm_Large = 140.7090882*kts2mps;
const double BASELINE_AVG_AIRSPEED_1000ftAGL_Large = 136.357891*kts2mps;
```

```
const double BASELINE_AVG_AIRSPEED_750ftAGL_Large = 133.7094905*kts2mps;
const double BASELINE_AVG_AIRSPEED_500ftAGL_Large = 132.1625477*kts2mps;
const double BASELINE_AVG_ROD_10nm_Large = -270.99*fpm2mps;
const double BASELINE_AVG_ROD_9nm_Large = -452.27*fpm2mps;
const double BASELINE_AVG_ROD_8nm_Large = -731.66*fpm2mps;
const double BASELINE_AVG_ROD_7nm_Large = -907.72*fpm2mps;
const double BASELINE_AVG_ROD_6nm_Large = -949.15*fpm2mps;
const double BASELINE_AVG_ROD_5dot5nm_Large = -926.22*fpm2mps;
const double BASELINE_AVG_ROD_5nm_Large = -898.78*fpm2mps;
const double BASELINE_AVG_ROD_4dot5nm_Large = -870.67*fpm2mps;
const double BASELINE_AVG_ROD_4nm_Large = -832.64*fpm2mps;
const double BASELINE_AVG_ROD_3dot5nm_Large = -786.46*fpm2mps;
const double BASELINE_AVG_ROD_3nm_Large = -752.20*fpm2mps;
const double BASELINE_AVG_ROD_1000ftAGL_Large = -771.25*fpm2mps;
const double BASELINE_AVG_ROD_750ftAGL_Large = -755.89*fpm2mps;
const double BASELINE_AVG_ROD_500ftAGL_Large = -765.57*fpm2mps;
//Small
const double BASELINE_AVG_AIRSPEED_10nm_Small = 177.8834295*kts2mps;
const double BASELINE_AVG_AIRSPEED_9nm_Small = 173.7126577*kts2mps;
const double BASELINE_AVG_AIRSPEED_8nm_Small = 172.1243557*kts2mps;
const double BASELINE_AVG_AIRSPEED_7nm_Small = 170.2593557*kts2mps;
const double BASELINE_AVG_AIRSPEED_6nm_Small = 168.4190268*kts2mps;
const double BASELINE_AVG_AIRSPEED_5dot5nm_Small = 167.0271812*kts2mps;
const double BASELINE_AVG_AIRSPEED_5nm_Small = 164.3659866*kts2mps;
const double BASELINE_AVG_AIRSPEED_4dot5nm_Small = 160.8483423*kts2mps;
const double BASELINE_AVG_AIRSPEED_4nm_Small = 156.0155235*kts2mps;
const double BASELINE_AVG_AIRSPEED_3dot5nm_Small = 151.4032215*kts2mps;
const double BASELINE_AVG_AIRSPEED_3nm_Small = 146.3830893*kts2mps;
const double BASELINE_AVG_AIRSPEED_1000ftAGL_Small = 140.1401456*kts2mps;
const double BASELINE_AVG_AIRSPEED_750ftAGL_Small = 134.3621174*kts2mps;
const double BASELINE_AVG_AIRSPEED_500ftAGL_Small = 127.9504564*kts2mps;
const double BASELINE_AVG_ROD_10nm_Small = -184.98*fpm2mps;
const double BASELINE_AVG_ROD_9nm_Small = -345.59*fpm2mps;
const double BASELINE_AVG_ROD_8nm_Small = -655.36*fpm2mps;
const double BASELINE_AVG_ROD_7nm_Small = -822.68*fpm2mps;
const double BASELINE_AVG_ROD_6nm_Small = -871.82*fpm2mps;
const double BASELINE_AVG_ROD_5dot5nm_Small = -885.96*fpm2mps;
const double BASELINE_AVG_ROD_5nm_Small = -887.42*fpm2mps;
const double BASELINE_AVG_ROD_4dot5nm_Small = -881.73*fpm2mps;
const double BASELINE_AVG_ROD_4nm_Small = -862.84*fpm2mps;
const double BASELINE_AVG_ROD_3dot5nm_Small = -837.34*fpm2mps;
const double BASELINE_AVG_ROD_3nm_Small = -803.83*fpm2mps;
const double BASELINE_AVG_ROD_1000ftAGL_Small = -904.46*fpm2mps;
const double BASELINE_AVG_ROD_750ftAGL_Small = -872.42*fpm2mps;
const double BASELINE_AVG_ROD_500ftAGL_Small = -858.94*fpm2mps;

//file location
string AIRCRAFT_DATA_FILE = "../Data/DATA _ AIRCRAFT/[input] aircraft
data.csv";
string FINALAPPROACHWIREFRAMEFILE_JFK = "../Data/[output]
finalapproachsegment_JFK.kml";
string FINALAPPROACHWIREFRAMEFILE_LGA = "../Data/[output]
finalapproachsegment_LGA.kml";
string FINALAPPROACHWIREFRAMEFILE_EWR = "../Data/[output]
finalapproachsegment_EWR.kml";
string FINALAPPROACHWIREFRAMEFILE_MDW = "../Data/[output]
finalapproachsegment_MDW.kml";
string CURRENTTRACKSKMLOUTPUT = "../Data/[output] current_tracks.kml";
string LATERALVERTICAL_DEVIATION_ADDRESS = "../Data/(output) lateral & vertical
position distribution.csv";
```

```
//data structures
struct Point //3D fix
{
	double lat; //decimal
	double lng; //decimal
	double x; //m
	double y; //m
	double z; //m, above mean sea level
};
struct TRACKPOINT //time series of Point, with derived features
{
	double t; //sec past midnight
	double lat; //decimal
	double lng; //decimal
	double x; //m
	double y; //m
	double z; //m, above mean sea level
	double groundspeed; //mps, derived
	double airspeed; //mps, derived from groundspeed and wind speed
	double heading; //rad, derived heading angle, north = 0, clockwise,
0~2*PI
	double verticalspeed; //mps, derived, POSITIVE for CLIMBING UP
};
struct WINDPOINT
{
	double t; //sec past midnight
	double heading; //heading, in rad (0 = from north)
	double speed; //knot
	double gust; //knot, gust speed, 0 if no gust
};
struct RUNWAY
{
	string name; //e.g. "31L"
	double true_alignment; //true north = 0, clockwise, same with "heading",
FAF->THR
	double vgpa; //visual glide path angle, usually 3 degrees
	double elevation; //m, above mean sea level
	double threshcrossheight; //m, relative height above runway elevation
	double angle; //convert from north-clockwise to standard east-
counterclockwise FAF->THR
	Point FAF; //final approach fix
	Point THR; //runway threshold
	Point REF; //reference location (after REF wireframe lat/vert dimensions
keep constant)
	//wireframe vertices at THR; "left" "right" is in terms of FAF->THR
	Point THR_leftup;
	Point THR_leftdown;
	Point THR_rightup;
	Point THR_rightdown;
	Point THR_leftup_horizontalrelax;
	Point THR_leftdown_horizontalrelax;
	Point THR_rightup_horizontalrelax;
	Point THR_rightdown_horizontalrelax;
	Point THR_leftup_verticalrelax;
	Point THR_leftdown_verticalrelax;
	Point THR_rightup_verticalrelax;
	Point THR_rightdown_verticalrelax;
	//wireframe vertices at REF
	Point REF_leftup;
```

```
	Point REF_leftdown;
	Point REF_rightup;
	Point REF_rightdown;
	Point REF_leftup_horizontalrelax;
	Point REF_leftdown_horizontalrelax;
	Point REF_rightup_horizontalrelax;
	Point REF_rightdown_horizontalrelax;
	Point REF_leftup_verticalrelax;
	Point REF_leftdown_verticalrelax;
	Point REF_rightup_verticalrelax;
	Point REF_rightdown_verticalrelax;
	//wireframe vertices at FAF
	Point FAF_leftup;
	Point FAF_leftdown;
	Point FAF_rightup;
	Point FAF_rightdown;
	Point FAF_leftup_horizontalrelax;
	Point FAF_leftdown_horizontalrelax;
	Point FAF_rightup_horizontalrelax;
	Point FAF_rightdown_horizontalrelax;
	Point FAF_leftup_verticalrelax;
	Point FAF_leftdown_verticalrelax;
	Point FAF_rightup_verticalrelax;
	Point FAF_rightdown_verticalrelax;
	//wireframe vertices at 12nm
	Point LOC_12nm_leftup;
	Point LOC_12nm_leftdown;
	Point LOC_12nm_rightup;
	Point LOC_12nm_rightdown;
	Point LOC_12nm_leftup_horizontalrelax;
	Point LOC_12nm_leftdown_horizontalrelax;
	Point LOC_12nm_rightup_horizontalrelax;
	Point LOC_12nm_rightdown_horizontalrelax;
	Point LOC_12nm_leftup_verticalrelax;
	Point LOC_12nm_leftdown_verticalrelax;
	Point LOC_12nm_rightup_verticalrelax;
	Point LOC_12nm_rightdown_verticalrelax;
	//3D fixes along final approach path with specific distance to runway
threshold
	Point location_1dot5nm; //for determining j_1dot5nm, as a substitude of
j_500ftAGL
	Point location_2dot25nm; //for determining j_2dot25nm, as a substitude of
j_750ftAGL
	Point location_3nm; //for determining j_3nm, can be a substitude of
j_1000ftAGL
	Point location_4nm;
	Point location_5nm;
	Point location_6nm;
	Point location_7nm;
	Point location_8nm;
	Point location_9nm;
	Point location_10nm;
	Point location_11nm;
	Point location_12nm;
};
struct AIRCRAFT {
	string model; //ICAO code
	string weight_class; //superheavy, heavy, B757, large, small
	double MTOW; //kg, maximum takeoff weight
};
```

```
struct FLIGHT
{
	int flightID; //e.g. 10001, loaded from flight data file
	string trackindex; //unique index for each track
	string date; //"20070111"
	AIRCRAFT aircraft; //search aircraft database for target aircraft
	RUNWAY landingrunway;
	vector<TRACKPOINT> Track; //load from track data file
	string procedure; //temp for MDW 13C only: ILS, Visual, RNP
	string flow; //temp for MDW 13C only: east, west

	//key track point index
	int j_3nm; // approx. 1000 ft AGL (an alternative for j_altitude for
flights with multiple approaches)
	int j_2dot25nm; //approx. 750 ft AGL (an alternative for j_altitude for
flights with multiple approaches)
	int j_1dot5nm; //approx. 500 ft AGL (an alternative for j_altitude for
flights with multiple approaches)
	int j_THR; //the track point of a standardized track with closest 3D
distance to landing runway threshold, first attempt
	//points with specific 2D-distance (radius) to THR, prediction location
	int j_10nm_r;
	int j_9nm_r;
	int j_8nm_r;
	int j_7nm_r;
	int j_6nm_r;
	int j_5dot5nm_r;
	int j_5nm_r;
	int j_4dot5nm_r;
	int j_4nm_r;
	int j_3dot5nm_r;
	int j_3nm_r;
	//at key altitude levels (the moment when the aircaft ACTUALLY reached
prescribed altitude level), first attempt
	int j_1000ftAGL;
	int j_750ftAGL;
	int j_500ftAGL;
	//wireframe entrance points and lateral/vertical acquisition points
	int j_firstentrance; //first entrance to the wireframed zone (combined
zone from 12nm~FAF & FAF~REF & REF~THR)
	int j_beforeentrance; //right before first entrance
	int j_acquisition_lateral; //"acquisition": use relaxed boundary in one
dimension and focus on the other
	int j_beforelateralintercept;
	int j_acquisition_vertical;
	int j_beforeverticalintercept;
	//deceleration point (within final 12nm, the approx. point where the
decent starts)
	int j_deceleration;
	double dist_deceleration;
//track qualification
	bool entrance;
	bool track_qualified; //if all qualification criteria are satisfied
//go-around
	bool go_around;
//aircraft state variables
	double v_12nm; //avg airspeed at 12 nm radius
	double z_AGL_12nm; //avg altitude at 12 nm radius
	double headingdiff_12nm; //heading difference at 12 nm from runway
centerline
```

```
	//at 10nm
	double avg_groundspeed_10nm;
	double avg_airspeed_10nm;
	double v_dev_10nm; //avg ground speed deviation from baseline at the
location for current weight class
	double avg_rateofdescent_10nm;
	double rod_dev_10nm; //avg rate of descent deviation from baseline at the
location for current weight class
	double lateral_deviation_10nm;
	double vertical_deviation_10nm; //relative to glide slope line
	double lateral_deviation_abs_10nm;
	double vertical_deviation_abs_10nm; //absolute deviation
	double heading_diff_10nm;
	string avg_airspeedcategory_10nm; //state categories (e.g. "low" "medium"
"high")
	string avg_rateofdescent_category_10nm;
	string lateral_deviation_category_10nm;
	string vertical_deviation_category_10nm;
	string heading_diff_category_10nm;
	int n_entrance_before_10nm; //number of times aircraft fly into wireframe
zone
	double rel_dist_to_10nm_lateral_acquisition_prj; //if less than
prediction location, set to 0
	double rel_dist_to_10nm_vertical_acquisition_prj;
	//at 9nm
	double avg_groundspeed_9nm;
	double avg_airspeed_9nm;
	double v_dev_9nm;
	double avg_rateofdescent_9nm;
	double rod_dev_9nm; //avg rate of descent deviation from baseline at the
location for current weight class
	double lateral_deviation_9nm;
	double vertical_deviation_9nm;
	double lateral_deviation_abs_9nm;
	double vertical_deviation_abs_9nm; //absolute deviation
	double heading_diff_9nm;
	string avg_airspeedcategory_9nm;
	string avg_rateofdescent_category_9nm;
	string lateral_deviation_category_9nm;
	string vertical_deviation_category_9nm;
	string heading_diff_category_9nm;
	int n_entrance_before_9nm;
	double rel_dist_to_9nm_lateral_acquisition_prj;
	double rel_dist_to_9nm_vertical_acquisition_prj;
	//at 8nm
	double avg_groundspeed_8nm;
	double avg_airspeed_8nm;
	double v_dev_8nm;
	double avg_rateofdescent_8nm;
	double rod_dev_8nm; //avg rate of descent deviation from baseline at the
location for current weight class
	double lateral_deviation_8nm;
	double vertical_deviation_8nm;
	double lateral_deviation_abs_8nm;
	double vertical_deviation_abs_8nm; //absolute deviation
	double heading_diff_8nm;
	string avg_airspeedcategory_8nm;
	string avg_rateofdescent_category_8nm;
	string lateral_deviation_category_8nm;
	string vertical_deviation_category_8nm;
```

```
        string heading_diff_category_8nm;
        int n_entrance_before_8nm;
        double rel_dist_to_8nm_lateral_acquisition_prj;
        double rel_dist_to_8nm_vertical_acquisition_prj;
        //at 7nm
        double avg_groundspeed_7nm;
        double avg_airspeed_7nm;
        double v_dev_7nm;
        double avg_rateofdescent_7nm;
        double rod_dev_7nm; //avg rate of descent deviation from baseline at the
location for current weight class
        double lateral_deviation_7nm;
        double vertical_deviation_7nm;
        double lateral_deviation_abs_7nm;
        double vertical_deviation_abs_7nm; //absolute deviation
        double heading_diff_7nm;
        string avg_airspeedcategory_7nm;
        string avg_rateofdescent_category_7nm;
        string lateral_deviation_category_7nm;
        string vertical_deviation_category_7nm;
        string heading_diff_category_7nm;
        int n_entrance_before_7nm;
        double rel_dist_to_7nm_lateral_acquisition_prj;
        double rel_dist_to_7nm_vertical_acquisition_prj;
        //at 6nm
        double avg_groundspeed_6nm;
        double avg_airspeed_6nm;
        double v_dev_6nm;
        double avg_rateofdescent_6nm;
        double rod_dev_6nm; //avg rate of descent deviation from baseline at the
location for current weight class
        double lateral_deviation_6nm;
        double vertical_deviation_6nm;
        double lateral_deviation_abs_6nm;
        double vertical_deviation_abs_6nm; //absolute deviation
        double heading_diff_6nm;
        string avg_airspeedcategory_6nm;
        string avg_rateofdescent_category_6nm;
        string lateral_deviation_category_6nm;
        string vertical_deviation_category_6nm;
        string heading_diff_category_6nm;
        int n_entrance_before_6nm;
        double rel_dist_to_6nm_lateral_acquisition_prj;
        double rel_dist_to_6nm_vertical_acquisition_prj;
        //at 5.5nm
        double avg_groundspeed_5dot5nm;
        double avg_airspeed_5dot5nm;
        double v_dev_5dot5nm;
        double avg_rateofdescent_5dot5nm;
        double rod_dev_5dot5nm; //avg rate of descent deviation from baseline at
the location for current weight class
        double lateral_deviation_5dot5nm;
        double vertical_deviation_5dot5nm;
        double lateral_deviation_abs_5dot5nm;
        double vertical_deviation_abs_5dot5nm; //absolute deviation
        double heading_diff_5dot5nm;
        string avg_airspeedcategory_5dot5nm;
        string avg_rateofdescent_category_5dot5nm;
        string lateral_deviation_category_5dot5nm;
        string vertical_deviation_category_5dot5nm;
```

```
        string heading_diff_category_5dot5nm;
        int n_entrance_before_5dot5nm;
        double rel_dist_to_5dot5nm_lateral_acquisition_prj;
        double rel_dist_to_5dot5nm_vertical_acquisition_prj;
        //at 5nm
        double avg_groundspeed_5nm;
        double avg_airspeed_5nm;
        double v_dev_5nm;
        double avg_rateofdescent_5nm;
        double rod_dev_5nm; //avg rate of descent deviation from baseline at the
location for current weight class
        double lateral_deviation_5nm;
        double vertical_deviation_5nm;
        double lateral_deviation_abs_5nm;
        double vertical_deviation_abs_5nm; //absolute deviation
        double heading_diff_5nm;
        string avg_airspeedcategory_5nm;
        string avg_rateofdescent_category_5nm;
        string lateral_deviation_category_5nm;
        string vertical_deviation_category_5nm;
        string heading_diff_category_5nm;
        int n_entrance_before_5nm;
        double rel_dist_to_5nm_lateral_acquisition_prj;
        double rel_dist_to_5nm_vertical_acquisition_prj;
        //at 4.5nm
        double avg_groundspeed_4dot5nm;
        double avg_airspeed_4dot5nm;
        double v_dev_4dot5nm;
        double avg_rateofdescent_4dot5nm;
        double rod_dev_4dot5nm; //avg rate of descent deviation from baseline at
the location for current weight class
        double lateral_deviation_4dot5nm;
        double vertical_deviation_4dot5nm;
        double lateral_deviation_abs_4dot5nm;
        double vertical_deviation_abs_4dot5nm; //absolute deviation
        double heading_diff_4dot5nm;
        string avg_airspeedcategory_4dot5nm;
        string avg_rateofdescent_category_4dot5nm;
        string lateral_deviation_category_4dot5nm;
        string vertical_deviation_category_4dot5nm;
        string heading_diff_category_4dot5nm;
        int n_entrance_before_4dot5nm;
        double rel_dist_to_4dot5nm_lateral_acquisition_prj;
        double rel_dist_to_4dot5nm_vertical_acquisition_prj;
        //at 4nm
        double avg_groundspeed_4nm;
        double avg_airspeed_4nm;
        double v_dev_4nm;
        double avg_rateofdescent_4nm;
        double rod_dev_4nm; //avg rate of descent deviation from baseline at the
location for current weight class
        double lateral_deviation_4nm;
        double vertical_deviation_4nm;
        double lateral_deviation_abs_4nm;
        double vertical_deviation_abs_4nm; //absolute deviation
        double heading_diff_4nm;
        string avg_airspeedcategory_4nm;
        string avg_rateofdescent_category_4nm;
        string lateral_deviation_category_4nm;
        string vertical_deviation_category_4nm;
```

```
	string heading_diff_category_4nm;
	int n_entrance_before_4nm;
	double rel_dist_to_4nm_lateral_acquisition_prj;
	double rel_dist_to_4nm_vertical_acquisition_prj;
	//at 3.5nm
	double avg_groundspeed_3dot5nm;
	double avg_airspeed_3dot5nm;
	double v_dev_3dot5nm;
	double avg_rateofdescent_3dot5nm;
	double rod_dev_3dot5nm; //avg rate of descent deviation from baseline at
the location for current weight class
	double lateral_deviation_3dot5nm;
	double vertical_deviation_3dot5nm;
	double lateral_deviation_abs_3dot5nm;
	double vertical_deviation_abs_3dot5nm; //absolute deviation
	double heading_diff_3dot5nm;
	string avg_airspeedcategory_3dot5nm;
	string avg_rateofdescent_category_3dot5nm;
	string lateral_deviation_category_3dot5nm;
	string vertical_deviation_category_3dot5nm;
	string heading_diff_category_3dot5nm;
	int n_entrance_before_3dot5nm;
	double rel_dist_to_3dot5nm_lateral_acquisition_prj;
	double rel_dist_to_3dot5nm_vertical_acquisition_prj;
	//at 3nm
	double avg_groundspeed_3nm;
	double avg_airspeed_3nm;
	double v_dev_3nm;
	double avg_rateofdescent_3nm;
	double rod_dev_3nm; //avg rate of descent deviation from baseline at the
location for current weight class
	double lateral_deviation_3nm;
	double vertical_deviation_3nm;
	double lateral_deviation_abs_3nm;
	double vertical_deviation_abs_3nm; //absolute deviation
	double heading_diff_3nm;
	string avg_airspeedcategory_3nm;
	string avg_rateofdescent_category_3nm;
	string lateral_deviation_category_3nm;
	string vertical_deviation_category_3nm;
	string heading_diff_category_3nm;
	int n_entrance_before_3nm;
	double rel_dist_to_3nm_lateral_acquisition_prj;
	double rel_dist_to_3nm_vertical_acquisition_prj;
	//state variables at 1000' AGL
	double avg_groundspeed_1000ftAGL;
	double avg_airspeed_1000ftAGL;
	double avg_rateofdescent_1000ftAGL;
	double lateral_deviation_1000ftAGL;
	double vertical_deviation_1000ftAGL;
	string avg_airspeedcategory_1000ftAGL;
	string avg_rateofdescent_category_1000ftAGL;
	string lateral_deviation_category_1000ftAGL;
	string vertical_deviation_category_1000ftAGL;
	//state variables at 750' AGL
	double avg_groundspeed_750ftAGL;
	double avg_airspeed_750ftAGL;
	double avg_rateofdescent_750ftAGL;
	double lateral_deviation_750ftAGL;
	double vertical_deviation_750ftAGL;
```

```
	string avg_airspeedcategory_750ftAGL;
	string avg_rateofdescent_category_750ftAGL;
	string lateral_deviation_category_750ftAGL;
	string vertical_deviation_category_750ftAGL;
	//state variables at 500' AGL
	double avg_groundspeed_500ftAGL;
	double avg_airspeed_500ftAGL;
	double avg_rateofdescent_500ftAGL;
	double lateral_deviation_500ftAGL;
	double vertical_deviation_500ftAGL;
	string avg_airspeedcategory_500ftAGL;
	string avg_rateofdescent_category_500ftAGL;
	string lateral_deviation_category_500ftAGL;
	string vertical_deviation_category_500ftAGL;
	//state variables at THR
	double avg_groundspeed_THR; //backward traced
	double avg_airspeed_THR;
//entrance and acquisition related variables
	double time2landing_entrance; //time from first entrance to touchdown
	double dist2THR_lateral_acquisition_prj; //projected distance from
lateral acquisition point to rw threshold
	double dist2THR_vertical_acquisition_prj; //projected distance from
vertical acquisition point to rw threshold
	double levelsegmentflighttime; //time spent in the FAF-to-12nm wireframe
segment
//unstable approach events and sub-events
	bool USD1000; //Unstable final approach speed, deceleration:
avg_speed_1000ft - avg_speed_THR > 10 kts
	bool USD750; //Unstable final approach speed, deceleration:
avg_speed_750ft - avg_speed_THR > 10 kts
	bool USD500; //Unstable final approach speed, deceleration:
avg_speed_500ft - avg_speed_THR > 10 kts
	bool USA1000; //Unstable final approach speed, acceleration:
avg_speed_1000ft - avg_speed_THR > 10 kts
	bool USA750; //Unstable final approach speed, acceleration:
avg_speed_750ft - avg_speed_THR > 10 kts
	bool USA500; //Unstable final approach speed, acceleration:
avg_speed_500ft - avg_speed_THR > 10 kts
	bool ERD1000; //Excessive rate of decent: avg_ROD  > 1000ft/min for more
than 10s from 1000ft to runway threshold
	bool ERD750; //Excessive rate of decent: avg_ROD  > 1000ft/min for more
than 10s from 750ft to runway threshold
	bool ERD500; //Excessive rate of decent: avg_ROD  > 1000ft/min for more
than 10s from 500ft to runway threshold
	bool SAA1000; //Short vertical acquisition of glide path from above from
1000ft AGL to landing
	bool SAA750;
	bool SAA500;
	bool SAB1000; //Short vertical acquisition of glide path from below from
1000ft AGL to landing
	bool SAB750;
	bool SAB500;
	bool SAL1000; //Short lateral acquisition of runway centerline from left
side from 1000ft AGL to landing
	bool SAL750;
	bool SAL500;
	bool SAR1000; //Short lateral acquisition of runway centerline from right
side from 1000ft AGL to landing
	bool SAR750;
	bool SAR500;
```

```
	//if stable1000 is true then SA_750/500 must be true
	bool stable1000; //if none of the above events occur, the flight is
stable
	bool stable750;
	bool stable500;
	bool unstable1000; //if any sub-event occurs, including SAB, unstable =
true
	bool unstable750;
	bool unstable500;
};

vector<RUNWAY> JFKrunways;
RUNWAY JFK13R, JFK31L, JFK04L, JFK22R, JFK13L, JFK31R, JFK04R, JFK22L;
vector<RUNWAY> LGArunways;
RUNWAY LGA13, LGA31, LGA22, LGA04;
vector<RUNWAY> EWRrunways;
RUNWAY EWR11, EWR29, EWR22L, EWR04R, EWR04L, EWR22R;
vector<RUNWAY> MDWrunways;
RUNWAY MDW04L, MDW22R, MDW04R, MDW22L, MDW13C, MDW31C, MDW13L, MDW31R, MDW13R,
MDW31L;
RUNWAY RWpending;

vector<WINDPOINT> wind20070111;
vector<WINDPOINT> wind20070112;
vector<WINDPOINT> wind20070113;
vector<WINDPOINT> wind20070114;
vector<WINDPOINT> wind20070115;
vector<WINDPOINT> wind20070116;
vector<WINDPOINT> wind20070117;
vector<WINDPOINT> wind20070323;
vector<WINDPOINT> wind20070324;
vector<WINDPOINT> wind20070325;
vector<WINDPOINT> wind20070326;
vector<WINDPOINT> wind20070327;
vector<WINDPOINT> wind20070328;
vector<WINDPOINT> wind20070329;
vector<WINDPOINT> wind20070721;
vector<WINDPOINT> wind20070722;
vector<WINDPOINT> wind20070723;
vector<WINDPOINT> wind20070724;
vector<WINDPOINT> wind20070725;
vector<WINDPOINT> wind20070726;
vector<WINDPOINT> wind20070727;
vector<WINDPOINT> wind20071001;
vector<WINDPOINT> wind20071002;
vector<WINDPOINT> wind20071003;
vector<WINDPOINT> wind20071004;
vector<WINDPOINT> wind20071005;
vector<WINDPOINT> wind20071006;
vector<WINDPOINT> wind20071007;
```

Cpp file

```
#include "Header.h";
void split(const string &str, const string &delim, vector<string> &results, int
flag)
{
	int n;
	string temp = str;

	if (flag != -1 && flag != 0)
		throw "Split: illegal flag (should be -1 or 0).";

	results.clear();
	while ((n = temp.find_first_of(delim)) != temp.npos) {			//
npos means no match found
		if (n > flag)
			results.push_back(temp.substr(0, n));
		temp = temp.substr(n+1);
	}
	if ((int)temp.length() > flag)
		results.push_back(temp);
}
class Ellipsoid
{
public:
	Ellipsoid(){};
	Ellipsoid(int Id, char* name, double radius, double ecc)
	{
		id = Id; ellipsoidName = name;
		EquatorialRadius = radius; eccentricitySquared = ecc;
	}
	int id;
	char* ellipsoidName;
	double EquatorialRadius;
	double eccentricitySquared;
};
static Ellipsoid ellipsoid[] = { //  id, Ellipsoid name, Equatorial Radius,
square of eccentricity
	Ellipsoid( -1, "Placeholder", 0, 0),//placeholder only, To allow array
indices to match id numbers
	Ellipsoid( 1, "Airy", 6377563, 0.00667054),
	Ellipsoid( 2, "Australian National", 6378160, 0.006694542),
	Ellipsoid( 3, "Bessel 1841", 6377397, 0.006674372),
	Ellipsoid( 4, "Bessel 1841 (Nambia) ", 6377484, 0.006674372),
	Ellipsoid( 5, "Clarke 1866", 6378206, 0.006768658),
	Ellipsoid( 6, "Clarke 1880", 6378249, 0.006803511),
	Ellipsoid( 7, "Everest", 6377276, 0.006637847),
	Ellipsoid( 8, "Fischer 1960 (Mercury) ", 6378166, 0.006693422),
	Ellipsoid( 9, "Fischer 1968", 6378150, 0.006693422),
	Ellipsoid( 10, "GRS 1967", 6378160, 0.006694605),
	Ellipsoid( 11, "GRS 1980", 6378137, 0.00669438),
	Ellipsoid( 12, "Helmert 1906", 6378200, 0.006693422),
	Ellipsoid( 13, "Hough", 6378270, 0.00672267),
	Ellipsoid( 14, "International", 6378388, 0.00672267),
	Ellipsoid( 15, "Krassovsky", 6378245, 0.006693422),
	Ellipsoid( 16, "Modified Airy", 6377340, 0.00667054),
	Ellipsoid( 17, "Modified Everest", 6377304, 0.006637847),
	Ellipsoid( 18, "Modified Fischer 1960", 6378155, 0.006693422),
	Ellipsoid( 19, "South American 1969", 6378160, 0.006694542),
	Ellipsoid( 20, "WGS 60", 6378165, 0.006693422),
```

```
	Ellipsoid( 21, "WGS 66", 6378145, 0.006694542),
	Ellipsoid( 22, "WGS-72", 6378135, 0.006694318),
	Ellipsoid( 23, "WGS-84", 6378137, 0.00669438)
};
#define REFELLIPSOID 23    //Lat/Long to UTM conversion. WGS-84
// longitudes of main airport
const double regionLongitude[7] = {0, -118.2, -74.0, -84.445, 0, -89.6, -
87.7518}; // xxx, LAX, NY area, ATL, DC, MEM, MDW (adjusted)
// Approximate middle longitude of UTM zone containing a given airport
const double UTM_MidLongitude[7] = {0, -117.0, -75.0, -87.0, 0, -87.0, -87.0};
	// xxx, LAX, NY area, ATL, DC, MEM, Chicago area
// Approximate XY coordinates of main airport, after offsetting the longitude
const double OriginX[7] = {0, 0, 486366, 0, 0, 500006.019, 437266};
const double OriginY[7] = {0, 0, 4505781, 0, 0, 3877958.77, 462695};

string kmlTime(double time)
{
	int hrInt, mnInt, scInt;
	double hrDbl, mnDbl, scDbl;
	stringstream tm;
	hrDbl = time / 3600.0; hrInt = (int) hrDbl;
	mnDbl = (time - 3600.0 * (double) hrInt) / 60.0; mnInt = (int) mnDbl;
	scDbl = time - 3600.0 * (double) hrInt - 60.0 * (double) mnInt; scInt =
(int) scDbl;
	tm.fill('0');
	tm << "2000-01-01T";
	tm.width(2);
	tm << hrInt;
	tm << ":";
	tm.width(2);
	tm << mnInt;
	tm << ":";
	tm.width(2);
	tm << scInt;
	tm << "Z";
	return tm.str();
}
char UTMLetterDesignator(double Lat)
{
//This routine determines the correct UTM letter designator for the given
latitude
	char LetterDesignator;

	if((84 >= Lat) && (Lat >= 72)) LetterDesignator = 'X';
	else if((72 > Lat) && (Lat >= 64)) LetterDesignator = 'W';
	else if((64 > Lat) && (Lat >= 56)) LetterDesignator = 'V';
	else if((56 > Lat) && (Lat >= 48)) LetterDesignator = 'U';
	else if((48 > Lat) && (Lat >= 40)) LetterDesignator = 'T';
	else if((40 > Lat) && (Lat >= 32)) LetterDesignator = 'S';
	else if((32 > Lat) && (Lat >= 24)) LetterDesignator = 'R';
	else if((24 > Lat) && (Lat >= 16)) LetterDesignator = 'Q';
	else if((16 > Lat) && (Lat >= 8)) LetterDesignator = 'P';
	else if(( 8 > Lat) && (Lat >= 0)) LetterDesignator = 'N';
	else if(( 0 > Lat) && (Lat >= -8)) LetterDesignator = 'M';
	else if((-8> Lat) && (Lat >= -16)) LetterDesignator = 'L';
	else if((-16 > Lat) && (Lat >= -24)) LetterDesignator = 'K';
	else if((-24 > Lat) && (Lat >= -32)) LetterDesignator = 'J';
	else if((-32 > Lat) && (Lat >= -40)) LetterDesignator = 'H';
	else if((-40 > Lat) && (Lat >= -48)) LetterDesignator = 'G';
	else if((-48 > Lat) && (Lat >= -56)) LetterDesignator = 'F';
```

```
	else if((-56 > Lat) && (Lat >= -64)) LetterDesignator = 'E';
	else if((-64 > Lat) && (Lat >= -72)) LetterDesignator = 'D';
	else if((-72 > Lat) && (Lat >= -80)) LetterDesignator = 'C';
	else LetterDesignator = 'Z'; //an error flag
	return LetterDesignator;
}
void LLtoUTM(int ReferenceEllipsoid, const double Lat, const double Long,
double &UTMNorthing, double &UTMEasting, char* UTMZone)
{
//converts lat/long to UTM coords.
	double a = ellipsoid[ReferenceEllipsoid].EquatorialRadius;
	double eccSquared = ellipsoid[ReferenceEllipsoid].eccentricitySquared;
	double k0 = 0.9996;
	double LongOrigin;
	double eccPrimeSquared;
	double N, T, C, A, M;
	double LongTemp = (Long+180)-int((Long+180)/360)*360-180;
	double LatRad = Lat*deg2rad;
	double LongRad = LongTemp*deg2rad;
	double LongOriginRad;
	int    ZoneNumber;
	ZoneNumber = int((LongTemp + 180)/6) + 1;
	if( Lat >= 56.0 && Lat < 64.0 && LongTemp >= 3.0 && LongTemp < 12.0 )
		ZoneNumber = 32;
	if( Lat >= 72.0 && Lat < 84.0 )
	{
	  if(      LongTemp >= 0.0  && LongTemp <  9.0 ) ZoneNumber = 31;
	  else if( LongTemp >= 9.0  && LongTemp < 21.0 ) ZoneNumber = 33;
	  else if( LongTemp >= 21.0 && LongTemp < 33.0 ) ZoneNumber = 35;
	  else if( LongTemp >= 33.0 && LongTemp < 42.0 ) ZoneNumber = 37;
	 }
	LongOrigin = (ZoneNumber - 1)*6 - 180 + 3;
	LongOriginRad = LongOrigin * deg2rad;

	sprintf(UTMZone, "%d%c", ZoneNumber, UTMLetterDesignator(Lat));
	eccPrimeSquared = (eccSquared)/(1-eccSquared);
	N = a/sqrt(1-eccSquared*sin(LatRad)*sin(LatRad));
	T = tan(LatRad)*tan(LatRad);
	C = eccPrimeSquared*cos(LatRad)*cos(LatRad);
	A = cos(LatRad)*(LongRad-LongOriginRad);
	M = a*((1    - eccSquared/4             - 3*eccSquared*eccSquared/64
	- 5*eccSquared*eccSquared*eccSquared/256)*LatRad
	                    - (3*eccSquared/8   + 3*eccSquared*eccSquared/32
	+ 45*eccSquared*eccSquared*eccSquared/1024)*sin(2*LatRad)
	                                                    +
(15*eccSquared*eccSquared/256 +
45*eccSquared*eccSquared*eccSquared/1024)*sin(4*LatRad)
	                                                    -
(35*eccSquared*eccSquared*eccSquared/3072)*sin(6*LatRad));
	UTMEasting = (double)(k0*N*(A+(1-T+C)*A*A*A/6
	                           + (5-18*T+T*T+72*C-
58*eccPrimeSquared)*A*A*A*A*A/120)
	                           + 500000.0);
	UTMNorthing = (double)(k0*(M+N*tan(LatRad)*(A*A/2+(5-
T+9*C+4*C*C)*A*A*A*A/24
	                       + (61-58*T+T*T+600*C-
330*eccPrimeSquared)*A*A*A*A*A*A/720)));
	if(Lat < 0)
		UTMNorthing += 10000000.0;
}
```

```
void UTMtoLL(int ReferenceEllipsoid, const double UTMNorthing, const double
UTMEasting, const char* UTMZone,double& Lat,  double& Long )
{
//converts UTM coords to lat/long.  Equations from USGS Bulletin 1532
       double k0 = 0.9996;
       double a = ellipsoid[ReferenceEllipsoid].EquatorialRadius;
       double eccSquared = ellipsoid[ReferenceEllipsoid].eccentricitySquared;
       double eccPrimeSquared;
       double e1 = (1-sqrt(1-eccSquared))/(1+sqrt(1-eccSquared));
       double N1, T1, C1, R1, D, M;
       double LongOrigin;
       double mu, phi1, phi1Rad;
       double x, y;
       int ZoneNumber;
       char* ZoneLetter;
       int NorthernHemisphere; //1 for northern hemispher, 0 for southern

       x = UTMEasting - 500000.0; //remove 500,000 meter offset for longitude
       y = UTMNorthing;
       ZoneNumber = strtoul(UTMZone, &ZoneLetter, 10);
       if((*ZoneLetter - 'N') >= 0)
              NorthernHemisphere = 1;
       else
       {
              NorthernHemisphere = 0;
              y -= 10000000.0;
       }
       LongOrigin = (ZoneNumber - 1)*6 - 180 + 3;
       eccPrimeSquared = (eccSquared)/(1-eccSquared);
       M = y / k0;
       mu = M/(a*(1-eccSquared/4-3*eccSquared*eccSquared/64-
5*eccSquared*eccSquared*eccSquared/256));
       phi1Rad = mu  + (3*e1/2-27*e1*e1*e1/32)*sin(2*mu)
                           + (21*e1*e1/16-55*e1*e1*e1*e1/32)*sin(4*mu)
                           +(151*e1*e1*e1/96)*sin(6*mu);
       phi1 = phi1Rad*rad2deg;
       N1 = a/sqrt(1-eccSquared*sin(phi1Rad)*sin(phi1Rad));
       T1 = tan(phi1Rad)*tan(phi1Rad);
       C1 = eccPrimeSquared*cos(phi1Rad)*cos(phi1Rad);
       R1 = a*(1-eccSquared)/pow(1-eccSquared*sin(phi1Rad)*sin(phi1Rad), 1.5);
       D = x/(N1*k0);
       Lat = phi1Rad - (N1*tan(phi1Rad)/R1)*(D*D/2-(5+3*T1+10*C1-4*C1*C1-
9*eccPrimeSquared)*D*D*D*D/24
                                    +(61+90*T1+298*C1+45*T1*T1-
252*eccPrimeSquared-3*C1*C1)*D*D*D*D*D*D/720);
       Lat = Lat * rad2deg;
       Long = (D-(1+2*T1+C1)*D*D*D/6+(5-2*C1+28*T1-
3*C1*C1+8*eccPrimeSquared+24*T1*T1)
                                    *D*D*D*D*D/120)/cos(phi1Rad);
       Long = LongOrigin + Long * rad2deg;
}
void LatLong2XY(int regionID, const double lat, const double lng, double &x,
double &y)
{
       double lng2, x2, y2;
       char UTMZone[4];
       // Offset longitude so that the airport is in the middle of the UTM zone
       double delta_lng = lng - regionLongitude[regionID]; //the difference in
longitude from target location to predefined region (i.e. airport) loacation
       lng2 = UTM_MidLongitude[regionID] + delta_lng;
```

```
	// Convert lat-long to UTM
	LLtoUTM(REFELLIPSOID, lat, lng2, y2, x2, UTMZone);
	x = x2 - OriginX[regionID];
	y = y2 - OriginY[regionID];
}
void XY2LatLong(int regionID, const double x, const double y, double &lat,
double &lng)
{	// Converts x-y to Lat-Long coordinates.
	double lng2 = 0, x2, y2;
	char UTMZone[3];
	if (regionID == 0) {			// Temporary: UTM zone for NYC =
"18T"
		UTMZone[0] = '1'; UTMZone[1] = '8'; UTMZone[2] = 'T';
	}
	else if (regionID == 1) {		// UTM zone for LA area = "11S"
		UTMZone[0] = '1'; UTMZone[1] = '1'; UTMZone[2] = 'S';
	}
	else if (regionID == 2) {		// UTM zone for NY area = "18T"
		UTMZone[0] = '1'; UTMZone[1] = '8'; UTMZone[2] = 'T';
	}
	else if (regionID == 3) {		// UTM zone for ATL = "16S"
		UTMZone[0] = '1'; UTMZone[1] = '6'; UTMZone[2] = 'S';
	}
	else if (regionID == 4) {		// UTM zone for DC area = "18S"
		UTMZone[0] = '1'; UTMZone[1] = '8'; UTMZone[2] = 'S';
	}
	else if (regionID == 5) {		// UTM zone for MEM = "16S"
		UTMZone[0] = '1'; UTMZone[1] = '6'; UTMZone[2] = 'S';
	}
	else if (regionID == 6) {		// for Chicago area ("16T"), added by
Zhenming Wang
		UTMZone[0] = '1'; UTMZone[1] = '6'; UTMZone[2] = 'T';
	}
	else {
		cout << "Error in XY2LatLong: Airport code not defined\n";
		exit(-1);
	}
	x2 = x + OriginX[regionID];
	y2 = y + OriginY[regionID];
	UTMtoLL(REFELLIPSOID, y2, x2, UTMZone, lat, lng2);
	// Offset longitude so that airport is in the middle of the
	lng = lng2 + regionLongitude[regionID] - UTM_MidLongitude[regionID];
}
void LatLong2UTM(int regionID, const double lat, const double lng, double &x,
double &y)
{	// Convert Lat-Long to UTM coordinates (does not offset longitude)
	char UTMZone[4];
	LLtoUTM(REFELLIPSOID, lat, lng, y, x, UTMZone);
}

double maxof(double x1, double x2) {
	double themax = 0.0;
	if (x1 > x2)
		themax = x1;
	else
		themax = x2;
	return themax;
}
double distance_2D(double x1, double y1, double x2, double y2) { //in m
	double d;
```

```
	d = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
	return d;
}
double distance_2D(Point p1, Point p2) {
	double d;
	d = sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
	return d;
}
double distance_2D(TRACKPOINT p1, Point p2) {
	double d;
	d = sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
	return d;
}
double distance_2D(TRACKPOINT tp1, TRACKPOINT tp2) {
	double d;
	d = sqrt((tp1.x-tp2.x)*(tp1.x-tp2.x)+(tp1.y-tp2.y)*(tp1.y-tp2.y));
	return d;
}
double distance_3D(double x1, double y1, double z1, double x2, double y2,
double z2) {
	double d;
	d = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
	return d;
}
double distance_3D(Point p1, Point p2) {
	double d;
	d = sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-
p2.z)*(p1.z-p2.z));
	return d;
}
double distance_3D(TRACKPOINT tp1, TRACKPOINT tp2) {
	double d;
	d = sqrt((tp1.x-tp2.x)*(tp1.x-tp2.x)+(tp1.y-tp2.y)*(tp1.y-tp2.y)+(tp1.z-
tp2.z)*(tp1.z-tp2.z));
	return d;
}
double distance_3D(TRACKPOINT tp, Point p) {
	double d;
	d = sqrt((tp.x-p.x)*(tp.x-p.x)+(tp.y-p.y)*(tp.y-p.y)+(tp.z-p.z)*(tp.z-
p.z));
	return d;
}
double dist_point2plane(Point p, Point p1, Point p2, Point p3)
{
	double x = p.x;
	double y = p.y;
	double z = p.z;
	double x1 = p1.x;
	double y1 = p1.y;
	double z1 = p1.z;
	double x2 = p2.x;
	double y2 = p2.y;
	double z2 = p2.z;
	double x3 = p3.x;
	double y3 = p3.y;
	double z3 = p3.z;
	double A = (y2-y1)*(z3-z1)-(z2-z1)*(y3-y1);
	double B = (z2-z1)*(x3-x1)-(x2-x1)*(z3-z1);
	double C = (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
```

```
	double D = -x1*(y2-y1)*(z3-z1)-y1*(z2-z1)*(x3-x1)-z1*(x2-x1)*(y3-
y1)+x1*(z2-z1)*(y3-y1)+y1*(x2-x1)*(z3-z1)+z1*(y2-y1)*(x3-x1);
	double dist = (A*x + B*y + C*z + D)/sqrt(A*A+B*B+C*C); // signed value
	return dist;
}
double dist_point2plane(TRACKPOINT p, Point p1, Point p2, Point p3)
{
	double x = p.x;
	double y = p.y;
	double z = p.z;
	double x1 = p1.x;
	double y1 = p1.y;
	double z1 = p1.z;
	double x2 = p2.x;
	double y2 = p2.y;
	double z2 = p2.z;
	double x3 = p3.x;
	double y3 = p3.y;
	double z3 = p3.z;
	double A = (y2-y1)*(z3-z1)-(z2-z1)*(y3-y1);
	double B = (z2-z1)*(x3-x1)-(x2-x1)*(z3-z1);
	double C = (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
	double D = -x1*(y2-y1)*(z3-z1)-y1*(z2-z1)*(x3-x1)-z1*(x2-x1)*(y3-
y1)+x1*(z2-z1)*(y3-y1)+y1*(x2-x1)*(z3-z1)+z1*(y2-y1)*(x3-x1);
	double dist = (A*x + B*y + C*z + D)/sqrt(A*A+B*B+C*C); // signed value
	return dist;
}
double dist_point2line_2D(Point p, Point p1, Point p2) {
	//note: always return positive value, focusing on horizontal distance
only
	double d, A, B, C;
	A = p2.y - p1.y;
	B = p1.x - p2.x;
	C = p2.x*p1.y - p1.x*p2.y;
	d = abs(A*p.x + B*p.y + C)/sqrt(A*A + B*B);
	return d;
}
double dist_point2line_2D(TRACKPOINT p, Point p1, Point p2) {
	double d, A, B, C;
	A = p2.y - p1.y;
	B = p1.x - p2.x;
	C = p2.x*p1.y - p1.x*p2.y;
	d = abs(A*p.x + B*p.y + C)/sqrt(A*A + B*B);
	return d;
}
bool inbox(Point p, Point p1, Point p2, Point p3, Point p4, Point p5, Point p6,
Point p7, Point p8)
{	// p1 up-front-left, p2 up-front-right, p3 up-back-left, p4 up-back-right
	// p5 down-front-left, p6 down-front-right, p7 down-back-left, p8 down-
back-right
	double dist_up = dist_point2plane(p,p1,p2,p3); // to plane_up
	double dist_down = dist_point2plane(p,p5,p6,p7); // to plane_down
	double dist_front = dist_point2plane(p,p1,p5,p6); // to plane_front
	double dist_back = dist_point2plane(p,p3,p7,p8); // to plane_back
	double dist_left = dist_point2plane(p,p1,p3,p5); // to plane_left
	double dist_right = dist_point2plane(p,p2,p4,p6); // to plane_right
	bool in_box;
	if (dist_up * dist_down < 0 && dist_front * dist_back < 0 && dist_left *
dist_right < 0) in_box = true;
	else in_box = false;
```

```
	return in_box;
}
void plot_tracks(vector<FLIGHT> Flights, string kmlfile, string linecolor, int
linewidth, string iconlink)
{
	// show tracks in kml animation using the track data in Flights
	cout << "plotting flight tracks...\n";
	ofstream out;
	out.precision(10);//set precision, otherwise only keep four digits
following decimal point
	out.open(kmlfile.c_str());
	out << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
	out << "<kml xmlns=\"http://www.opengis.net/kml/2.2\"
xmlns:gx=\"http://www.google.com/kml/ext/2.2\"
xmlns:kml=\"http://www.opengis.net/kml/2.2\"
xmlns:atom=\"http://www.w3.org/2005/Atom\">\n";
	out << "<Document id=\"doc1\">";
	out << "<name>" << "tracks" << "</name>\n";
	out << "<visibility>0</visibility><open>0</open>" << "\n";
	out << "<Style id=\"" << "bbb" << "\">\n";
	out << "<LineStyle>\n";
	out << "<color>" << linecolor << "</color>\n";
	out << "<width>" << linewidth << "</width>\n";
	out << "</LineStyle>\n";
	out << "<IconStyle>
	<color>ffffffff</color><scale>1.0</scale><heading>1</heading><Icon><href>
"+ iconlink +"</href></Icon>	</IconStyle>" << "\n";
	out << "<LabelStyle><color>0000ffff</color>
	<scale>0.8</scale></LabelStyle>" << "\n";
	out << "<PolyStyle> <color>ff00ffff</color>
	<fill>0</fill></PolyStyle>" << "\n";
	out << "</Style>\n";
	for(int i = 1; i <= Flights.size(); i++)
	{
		out << "<Placemark id=\" " << Flights[i-1].flightID << "\"><name>"
			<< Flights[i-1].flightID << " Aircraft type " + Flights[i-
1].aircraft.model << "</name><visibility>1</visibility><description>"
			<< "ID: "<<Flights[i-1].flightID <<","<<Flights[i-
1].trackindex << "\n"
			<< "Aircraft: " << Flights[i-1].aircraft.model <<"\n"
			<< "Runway: " << Flights[i-1].landingrunway.name << "\n"
			<< "dt_entrance2landing: "<<Flights[i-
1].time2landing_entrance
			<< "</description>  <styleUrl>bbb</styleUrl><gx:Track
kml:id=\"null\"><extrude>0</extrude><altitudeMode>absolute</altitudeMode>" <<
"\n";
		for(int j = 1; j <= Flights[i-1].Track.size(); j++)
		{
			out << "<when>" << kmlTime(Flights[i-1].Track[j-1].t) <<
"</when>" <<"\n";
			out << "<gx:coord>" << Flights[i-1].Track[j-1].lng << ","
<< Flights[i-1].Track[j-1].lat << "," << Flights[i-1].Track[j-1].z <<
"</gx:coord>" << "\n";
		}
		out << "</gx:Track></Placemark>" << "\n";
	}
	out << "</Document>\n";
	out << "</kml>\n";
	out.close();
}
```

```
void plot_LGAcylinder(string outputfile, double radius, double z_sep)
{
	ofstream out(outputfile);
	out.precision(10);
	out << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
	out << "<kml xmlns=\"http://www.opengis.net/kml/2.2\">\n";
	out << "<Document>\n";
	out << "<name>" << "cylinder_test" << "</name>\n";

	out << "<Style id=\"" << "default" << "\">\n";
	out << "    <LineStyle>\n";
	out << "        <color>" << "fffff00f" << "</color>\n";
	out << "        <width>" << 2 << "</width>\n";
	out << "    </LineStyle>\n";
	out << "</Style>\n";
	out << "<Placemark>\n";
	out << "<styleUrl>#default</styleUrl>\n";
	out << "<LineString>\n";
	out << "<altitudeMode>absolute</altitudeMode>\n";// absolute values above
see level
	out << "<coordinates>\n";
	double LGAx, LGAy;
	LatLong2XY(regionID, 40.77725, -73.87261111, LGAx, LGAy); //LGA_lat,
LGA_lng, LGAx, LGAy);
	int nSeg = 20;
	double tempLAT, tempLNG, tempLAT1;
	for (int k = 1; k <= nSeg+1; ++k) { //clockwise
		XY2LatLong(regionID, LGAx +
radius*sin(k*360.0*deg2rad/(double)nSeg), LGAy +
radius*cos(k*360.0*deg2rad/(double)nSeg), tempLAT, tempLNG);
		out<<tempLNG<<","<<tempLAT<<","<<z_sep<<"\n";
		out<<tempLNG<<","<<tempLAT<<","<<0<<"\n";
		out<<tempLNG<<","<<tempLAT<<","<<z_sep<<"\n";
	}
	out << "</coordinates>\n";
	out << "</LineString>\n</Placemark>\n";
	out << "</Document>\n";
	out << "</kml>\n";
	out.close();
}
Point airport_location(string currentairport) {
	Point apt_pt;
	if (currentairport == "JFK") {apt_pt.lat = 40.63975111; apt_pt.lng = -
73.77892556; apt_pt.z = 12.6*ft2m;}
	else if (currentairport == "LGA") {apt_pt.lat = 40.77725; apt_pt.lng = -
73.87261111; apt_pt.z = 20.6*ft2m;}
	else if (currentairport == "EWR") {apt_pt.lat = 40.69247983; apt_pt.lng =
-74.16868678; apt_pt.z = 17.6*ft2m;}
	else if (currentairport == "MDW") {apt_pt.lat = 41.78583; apt_pt.lng = -
87.7518; apt_pt.z = 619.8*ft2m;}
	else {apt_pt.lat = 0; apt_pt.lng = 0; apt_pt.z = 0;} //default
	LatLong2XY(regionID, apt_pt.lat, apt_pt.lng, apt_pt.x, apt_pt.y);
	return apt_pt;
}
double getangle(double x1, double y1, double x2, double y2) {
	//calculate the counterclockwise angle formed between the x-axis and the
segment (from P(x1,y1) to P(x2,y2)), [0, 2*PI)
	double angle; // in rad
	if (x2 - x1 > 0 && y2 - y1 > 0) {angle = atan((y2-y1)/(x2-x1));}
```

```
	else if (x2 - x1 < 0 && y2 - y1 > 0) {angle = atan((y2-y1)/(x2-x1)) +
PI;}
	else if (x2 - x1 < 0 && y2 - y1 < 0) {angle = atan((y2-y1)/(x2-x1)) +
PI;}
	else if (x2 - x1 > 0 && y2 - y1 < 0) {angle = atan((y2-y1)/(x2-x1)) +
2*PI;}
	else if (x2 - x1 == 0 && y2 - y1 > 0) {angle = 0.5*PI;}
	else if (x2 - x1 == 0 && y2 - y1 < 0) {angle = 1.5*PI;}
	else if (x2 - x1 > 0 && y2 - y1 == 0) {angle = 0;}
	else if (x2 - x1 < 0 && y2 - y1 == 0) {angle = PI;}
	else angle = -1; //error for segment with no length
	return angle;
}
double getprocessedangle(double angle_original) {
	//convert standard [0, 2*PI) angle (output from getangle() function) to
processed angle with range [0, 0.5*PI)
	double angle_processed;
	if (angle_original < 0.5* PI) {angle_processed = angle_original;}
	else if (angle_original < PI) {angle_processed = PI - angle_original;}
	else if (angle_original < 1.5*PI) {angle_processed = angle_original -
PI;}
	else {angle_processed = 2*PI - angle_original;}
	return angle_processed;
}
void getPerpendicularPositions(double r, double angle, double x, double y,
double &x1, double &y1, double &x2, double &y2) {
	//a segment from some point to end point (x, y), with heading angle
[0,2*PI) counterclockwise
	//x1, y1 is the point (relatively, not horizontally in x) on the left
side (if moving from (0,0) to (x,y)) of point(x,y), which has a distance r to
original segment,
	double angle_processed = getprocessedangle(angle);
	if (angle < 0.5*PI) {
		x1 = x - r*sin(angle_processed);
		y1 = y + r*cos(angle_processed);
		x2 = x + r*sin(angle_processed);
		y2 = y - r*cos(angle_processed);
	}
	else if (angle < PI) {
		x1 = x - r*sin(angle_processed);
		y1 = y - r*cos(angle_processed);
		x2 = x + r*sin(angle_processed);
		y2 = y + r*cos(angle_processed);
	}
	else if (angle < 1.5*PI) {
		x1 = x + r*sin(angle_processed);
		y1 = y - r*cos(angle_processed);
		x2 = x - r*sin(angle_processed);
		y2 = y + r*cos(angle_processed);
	}
	else {
		x1 = x + r*sin(angle_processed);
		y1 = y + r*cos(angle_processed);
		x2 = x - r*sin(angle_processed);
		y2 = y - r*cos(angle_processed);
	}
}
void kmlPoint(ofstream &out, Point p) {
	out << p.lng << "," << p.lat << "," << p.z << "\n";
}
```

```
double heading2angle(double heading) {
	//from heading ([0, 2*PI) clockwise from north) to standard angle (east =
0, counterclockwise)
	double angle;
	angle = 0.5*PI - heading;
	if (angle < 0) {angle = angle + 2*PI;}
	return angle;
}
double angle2heading(double angle) {
	double heading;
	heading = 0.5*PI - angle;
	if (heading < 0) {heading = heading + 2*PI;}
	return heading;
}
Point define_runwaylocation(double distfromTHR, Point thr, Point ref) {
	Point runwaylocation;
	runwaylocation.x = (distfromTHR/REF2THR_DIST)*ref.x + ((REF2THR_DIST-
distfromTHR)/REF2THR_DIST)*thr.x;
	runwaylocation.y = (distfromTHR/REF2THR_DIST)*ref.y + ((REF2THR_DIST-
distfromTHR)/REF2THR_DIST)*thr.y;
	runwaylocation.z = (distfromTHR/REF2THR_DIST)*ref.z + ((REF2THR_DIST-
distfromTHR)/REF2THR_DIST)*thr.z;
	XY2LatLong(regionID, runwaylocation.x, runwaylocation.y,
runwaylocation.lat, runwaylocation.lng);
	return runwaylocation;
}
void runway_initialize(RUNWAY &rw, string name, double true_alignment, double
vgpa, double elevation, double threshcrossheight, double THR_lat, double
THR_lng, double dist_FAF2THR) {
	rw.name = name;
	rw.true_alignment = true_alignment;
	rw.angle = heading2angle(true_alignment);
	rw.vgpa = vgpa;
	rw.elevation = elevation;
	rw.threshcrossheight = threshcrossheight;
//runway THR location
	rw.THR.lat = THR_lat;
	rw.THR.lng = THR_lng;
	LatLong2XY(regionID, rw.THR.lat, rw.THR.lng, rw.THR.x, rw.THR.y);
	rw.THR.z = rw.elevation + rw.threshcrossheight;
//runway FAF location
	rw.FAF.x = rw.THR.x - dist_FAF2THR*cos(rw.angle);
	rw.FAF.y = rw.THR.y - dist_FAF2THR*sin(rw.angle);
	XY2LatLong(regionID, rw.FAF.x, rw.FAF.y, rw.FAF.lat, rw.FAF.lng);
	rw.FAF.z = rw.THR.z + dist_FAF2THR*tan(rw.vgpa);
//runway REF location
	rw.REF.x = rw.THR.x - REF2THR_DIST*cos(rw.angle);
	rw.REF.y = rw.THR.y - REF2THR_DIST*sin(rw.angle);
	XY2LatLong(regionID, rw.REF.x, rw.REF.y, rw.REF.lat, rw.REF.lng);
	rw.REF.z = rw.THR.z + REF2THR_DIST*tan(rw.vgpa);
//Key locations along final path
	rw.location_1dot5nm = define_runwaylocation(1.5*nm2m, rw.THR, rw.REF);
	rw.location_2dot25nm = define_runwaylocation(2.25*nm2m, rw.THR, rw.REF);
	rw.location_3nm = define_runwaylocation(3.0*nm2m, rw.THR, rw.REF);
	rw.location_4nm = define_runwaylocation(4.0*nm2m, rw.THR, rw.REF);
	rw.location_5nm = define_runwaylocation(5.0*nm2m, rw.THR, rw.REF);
	rw.location_6nm = define_runwaylocation(6.0*nm2m, rw.THR, rw.REF);
	rw.location_7nm = define_runwaylocation(7.0*nm2m, rw.THR, rw.REF);
	rw.location_8nm = define_runwaylocation(8.0*nm2m, rw.THR, rw.REF);
	rw.location_9nm = define_runwaylocation(9.0*nm2m, rw.THR, rw.REF);
```

```
	rw.location_10nm = define_runwaylocation(10.0*nm2m, rw.THR, rw.REF);
	rw.location_11nm = define_runwaylocation(11.0*nm2m, rw.THR, rw.REF);
	rw.location_12nm = define_runwaylocation(12.0*nm2m, rw.THR, rw.REF);
	///hardcoded, for different FAF value, need to reconfigure the altitudes
of key rw.locations, so that locations after FAF will form level flight segment
	rw.location_11nm.z = rw.location_10nm.z;
	rw.location_12nm.z = rw.location_10nm.z;
	//LOC_12nm surrounding positions
	double out12nmx1, out12nmy1, out12nmx2, out12nmy2, out12nmlat1,
out12nmlat2, out12nmlng1, out12nmlng2;
	getPerpendicularPositions(LATERALHALFWIDTH_12nm, rw.angle,
rw.location_12nm.x, rw.location_12nm.y, out12nmx1, out12nmy1, out12nmx2,
out12nmy2);
	XY2LatLong(regionID, out12nmx1, out12nmy1, out12nmlat1, out12nmlng1);
	XY2LatLong(regionID, out12nmx2, out12nmy2, out12nmlat2, out12nmlng2);
	rw.LOC_12nm_leftup.lat = out12nmlat1;
	rw.LOC_12nm_leftup.lng = out12nmlng1;
	rw.LOC_12nm_leftup.z = rw.location_12nm.z + VERTICALHALFWIDTH_12nm;
	LatLong2XY(regionID, rw.LOC_12nm_leftup.lat, rw.LOC_12nm_leftup.lng,
rw.LOC_12nm_leftup.x, rw.LOC_12nm_leftup.y);
	rw.LOC_12nm_leftdown.lat = out12nmlat1;
	rw.LOC_12nm_leftdown.lng = out12nmlng1;
	rw.LOC_12nm_leftdown.z = rw.location_12nm.z - VERTICALHALFWIDTH_12nm;
	LatLong2XY(regionID, rw.LOC_12nm_leftdown.lat, rw.LOC_12nm_leftdown.lng,
rw.LOC_12nm_leftdown.x, rw.LOC_12nm_leftdown.y);
	rw.LOC_12nm_rightup.lat = out12nmlat2;
	rw.LOC_12nm_rightup.lng = out12nmlng2;
	rw.LOC_12nm_rightup.z = rw.location_12nm.z + VERTICALHALFWIDTH_12nm;
	LatLong2XY(regionID, rw.LOC_12nm_rightup.lat, rw.LOC_12nm_rightup.lng,
rw.LOC_12nm_rightup.x, rw.LOC_12nm_rightup.y);
	rw.LOC_12nm_rightdown.lat = out12nmlat2;
	rw.LOC_12nm_rightdown.lng = out12nmlng2;
	rw.LOC_12nm_rightdown.z = rw.location_12nm.z - VERTICALHALFWIDTH_12nm;
	LatLong2XY(regionID, rw.LOC_12nm_rightdown.lat,
rw.LOC_12nm_rightdown.lng, rw.LOC_12nm_rightdown.x, rw.LOC_12nm_rightdown.y);
		//horizontally relaxed
	double horizonalrelax_out12nmx1, horizonalrelax_out12nmy1,
horizonalrelax_out12nmx2, horizonalrelax_out12nmy2, horizonalrelax_out12nmlat1,
horizonalrelax_out12nmlat2, horizonalrelax_out12nmlng1,
horizonalrelax_out12nmlng2;
	getPerpendicularPositions(RELAXEDHALFWIDTH, rw.angle, rw.location_12nm.x,
rw.location_12nm.y, horizonalrelax_out12nmx1, horizonalrelax_out12nmy1,
horizonalrelax_out12nmx2, horizonalrelax_out12nmy2);
	XY2LatLong(regionID, horizonalrelax_out12nmx1, horizonalrelax_out12nmy1,
horizonalrelax_out12nmlat1, horizonalrelax_out12nmlng1);
	XY2LatLong(regionID, horizonalrelax_out12nmx2, horizonalrelax_out12nmy2,
horizonalrelax_out12nmlat2, horizonalrelax_out12nmlng2);
	rw.LOC_12nm_leftup_horizontalrelax.lat = horizonalrelax_out12nmlat1;
	rw.LOC_12nm_leftup_horizontalrelax.lng = horizonalrelax_out12nmlng1;
	rw.LOC_12nm_leftup_horizontalrelax.z = rw.location_12nm.z +
VERTICALHALFWIDTH_12nm;
	LatLong2XY(regionID, rw.LOC_12nm_leftup_horizontalrelax.lat,
rw.LOC_12nm_leftup_horizontalrelax.lng, rw.LOC_12nm_leftup_horizontalrelax.x,
rw.LOC_12nm_leftup_horizontalrelax.y);
	rw.LOC_12nm_leftdown_horizontalrelax.lat = horizonalrelax_out12nmlat1;
	rw.LOC_12nm_leftdown_horizontalrelax.lng = horizonalrelax_out12nmlng1;
	rw.LOC_12nm_leftdown_horizontalrelax.z = rw.location_12nm.z -
VERTICALHALFWIDTH_12nm;
	LatLong2XY(regionID, rw.LOC_12nm_leftdown_horizontalrelax.lat,
rw.LOC_12nm_leftdown_horizontalrelax.lng,
```

```
rw.LOC_12nm_leftdown_horizontalrelax.x,
rw.LOC_12nm_leftdown_horizontalrelax.y);
       rw.LOC_12nm_rightup_horizontalrelax.lat = horizonalrelax_out12nmlat2;
       rw.LOC_12nm_rightup_horizontalrelax.lng = horizonalrelax_out12nmlng2;
       rw.LOC_12nm_rightup_horizontalrelax.z = rw.location_12nm.z +
VERTICALHALFWIDTH_12nm;
       LatLong2XY(regionID, rw.LOC_12nm_rightup_horizontalrelax.lat,
rw.LOC_12nm_rightup_horizontalrelax.lng, rw.LOC_12nm_rightup_horizontalrelax.x,
rw.LOC_12nm_rightup_horizontalrelax.y);
       rw.LOC_12nm_rightdown_horizontalrelax.lat = horizonalrelax_out12nmlat2;
       rw.LOC_12nm_rightdown_horizontalrelax.lng = horizonalrelax_out12nmlng2;
       rw.LOC_12nm_rightdown_horizontalrelax.z = rw.location_12nm.z -
VERTICALHALFWIDTH_12nm;
       LatLong2XY(regionID, rw.LOC_12nm_rightdown_horizontalrelax.lat,
rw.LOC_12nm_rightdown_horizontalrelax.lng,
rw.LOC_12nm_rightdown_horizontalrelax.x,
rw.LOC_12nm_rightdown_horizontalrelax.y);
             //vertically relaxed
       rw.LOC_12nm_leftup_verticalrelax.lat = rw.LOC_12nm_leftup.lat;
       rw.LOC_12nm_leftup_verticalrelax.lng = rw.LOC_12nm_leftup.lng;
       rw.LOC_12nm_leftup_verticalrelax.z = rw.location_12nm.z +
RELAXEDHALFWIDTH;
       LatLong2XY(regionID, rw.LOC_12nm_leftup_verticalrelax.lat,
rw.LOC_12nm_leftup_verticalrelax.lng, rw.LOC_12nm_leftup_verticalrelax.x,
rw.LOC_12nm_leftup_verticalrelax.y);
       rw.LOC_12nm_leftdown_verticalrelax.lat = rw.LOC_12nm_leftdown.lat;
       rw.LOC_12nm_leftdown_verticalrelax.lng = rw.LOC_12nm_leftdown.lng;
       rw.LOC_12nm_leftdown_verticalrelax.z = 0.0;
       LatLong2XY(regionID, rw.LOC_12nm_leftdown_verticalrelax.lat,
rw.LOC_12nm_leftdown_verticalrelax.lng, rw.LOC_12nm_leftdown_verticalrelax.x,
rw.LOC_12nm_leftdown_verticalrelax.y);
       rw.LOC_12nm_rightup_verticalrelax.lat = rw.LOC_12nm_rightup.lat;
       rw.LOC_12nm_rightup_verticalrelax.lng = rw.LOC_12nm_rightup.lng;
       rw.LOC_12nm_rightup_verticalrelax.z = rw.location_12nm.z +
RELAXEDHALFWIDTH;
       LatLong2XY(regionID, rw.LOC_12nm_rightup_verticalrelax.lat,
rw.LOC_12nm_rightup_verticalrelax.lng, rw.LOC_12nm_rightup_verticalrelax.x,
rw.LOC_12nm_rightup_verticalrelax.y);
       rw.LOC_12nm_rightdown_verticalrelax.lat = rw.LOC_12nm_rightdown.lat;
       rw.LOC_12nm_rightdown_verticalrelax.lng = rw.LOC_12nm_rightdown.lng;
       rw.LOC_12nm_rightdown_verticalrelax.z = 0.0;
       LatLong2XY(regionID, rw.LOC_12nm_rightdown_verticalrelax.lat,
rw.LOC_12nm_rightdown_verticalrelax.lng, rw.LOC_12nm_rightdown_verticalrelax.x,
rw.LOC_12nm_rightdown_verticalrelax.y);
       //FAF surrounding positions
       double outframex1, outframey1, outframex2, outframey2, outframelat1,
outframelat2, outframelng1, outframelng2;
       getPerpendicularPositions(LATERALHALFWIDTH_REF, rw.angle, rw.FAF.x,
rw.FAF.y, outframex1, outframey1, outframex2, outframey2);
       XY2LatLong(regionID, outframex1, outframey1, outframelat1, outframelng1);
       XY2LatLong(regionID, outframex2, outframey2, outframelat2, outframelng2);
       rw.FAF_leftup.lat = outframelat1;
       rw.FAF_leftup.lng = outframelng1;
       rw.FAF_leftup.z = rw.FAF.z + VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_leftup.lat, rw.FAF_leftup.lng,
rw.FAF_leftup.x, rw.FAF_leftup.y);
       rw.FAF_leftdown.lat = outframelat1;
       rw.FAF_leftdown.lng = outframelng1;
       rw.FAF_leftdown.z = rw.FAF.z - VERTICALHALFWIDTH_REF;
```

```
       LatLong2XY(regionID, rw.FAF_leftdown.lat, rw.FAF_leftdown.lng,
rw.FAF_leftdown.x, rw.FAF_leftdown.y);
       rw.FAF_rightup.lat = outframelat2;
       rw.FAF_rightup.lng = outframelng2;
       rw.FAF_rightup.z = rw.FAF.z + VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_rightup.lat, rw.FAF_rightup.lng,
rw.FAF_rightup.x, rw.FAF_rightup.y);
       rw.FAF_rightdown.lat = outframelat2;
       rw.FAF_rightdown.lng = outframelng2;
       rw.FAF_rightdown.z = rw.FAF.z - VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_rightdown.lat, rw.FAF_rightdown.lng,
rw.FAF_rightdown.x, rw.FAF_rightdown.y);
             //horizontally relaxed
       double horizontalrelax_outframex1, horizontalrelax_outframey1,
horizontalrelax_outframex2, horizontalrelax_outframey2,
horizontalrelax_outframelat1, horizontalrelax_outframelat2,
horizontalrelax_outframelng1, horizontalrelax_outframelng2;
       getPerpendicularPositions(RELAXEDHALFWIDTH, rw.angle, rw.FAF.x, rw.FAF.y,
horizontalrelax_outframex1, horizontalrelax_outframey1,
horizontalrelax_outframex2, horizontalrelax_outframey2);
       XY2LatLong(regionID, horizontalrelax_outframex1,
horizontalrelax_outframey1, horizontalrelax_outframelat1,
horizontalrelax_outframelng1);
       XY2LatLong(regionID, horizontalrelax_outframex2,
horizontalrelax_outframey2, horizontalrelax_outframelat2,
horizontalrelax_outframelng2);
       rw.FAF_leftup_horizontalrelax.lat = horizontalrelax_outframelat1;
       rw.FAF_leftup_horizontalrelax.lng = horizontalrelax_outframelng1;
       rw.FAF_leftup_horizontalrelax.z = rw.FAF.z + VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_leftup_horizontalrelax.lat,
rw.FAF_leftup_horizontalrelax.lng, rw.FAF_leftup_horizontalrelax.x,
rw.FAF_leftup_horizontalrelax.y);
       rw.FAF_leftdown_horizontalrelax.lat = horizontalrelax_outframelat1;
       rw.FAF_leftdown_horizontalrelax.lng = horizontalrelax_outframelng1;
       rw.FAF_leftdown_horizontalrelax.z = rw.FAF.z - VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_leftdown_horizontalrelax.lat,
rw.FAF_leftdown_horizontalrelax.lng, rw.FAF_leftdown_horizontalrelax.x,
rw.FAF_leftdown_horizontalrelax.y);
       rw.FAF_rightup_horizontalrelax.lat = horizontalrelax_outframelat2;
       rw.FAF_rightup_horizontalrelax.lng = horizontalrelax_outframelng2;
       rw.FAF_rightup_horizontalrelax.z = rw.FAF.z + VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_rightup_horizontalrelax.lat,
rw.FAF_rightup_horizontalrelax.lng, rw.FAF_rightup_horizontalrelax.x,
rw.FAF_rightup_horizontalrelax.y);
       rw.FAF_rightdown_horizontalrelax.lat = horizontalrelax_outframelat2;
       rw.FAF_rightdown_horizontalrelax.lng = horizontalrelax_outframelng2;
       rw.FAF_rightdown_horizontalrelax.z = rw.FAF.z - VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.FAF_rightdown_horizontalrelax.lat,
rw.FAF_rightdown_horizontalrelax.lng, rw.FAF_rightdown_horizontalrelax.x,
rw.FAF_rightdown_horizontalrelax.y);
             //vertically relaxed
       rw.FAF_leftup_verticalrelax.lat = rw.FAF_leftup.lat;
       rw.FAF_leftup_verticalrelax.lng = rw.FAF_leftup.lng;
       rw.FAF_leftup_verticalrelax.z = rw.FAF_leftup.z + RELAXEDHALFWIDTH;
       LatLong2XY(regionID, rw.FAF_leftup_verticalrelax.lat,
rw.FAF_leftup_verticalrelax.lng, rw.FAF_leftup_verticalrelax.x,
rw.FAF_leftup_verticalrelax.y);
       rw.FAF_leftdown_verticalrelax.lat = rw.FAF_leftdown.lat;
       rw.FAF_leftdown_verticalrelax.lng = rw.FAF_leftdown.lng;
       rw.FAF_leftdown_verticalrelax.z = 0.0;
```

```
        LatLong2XY(regionID, rw.FAF_leftdown_verticalrelax.lat,
rw.FAF_leftdown_verticalrelax.lng, rw.FAF_leftdown_verticalrelax.x,
rw.FAF_leftdown_verticalrelax.y);
        rw.FAF_rightup_verticalrelax.lat = rw.FAF_rightup.lat;
        rw.FAF_rightup_verticalrelax.lng = rw.FAF_rightup.lng;
        rw.FAF_rightup_verticalrelax.z = rw.FAF_rightup.z + RELAXEDHALFWIDTH;
        LatLong2XY(regionID, rw.FAF_rightup_verticalrelax.lat,
rw.FAF_rightup_verticalrelax.lng, rw.FAF_rightup_verticalrelax.x,
rw.FAF_rightup_verticalrelax.y);
        rw.FAF_rightdown_verticalrelax.lat = rw.FAF_rightdown.lat;
        rw.FAF_rightdown_verticalrelax.lng = rw.FAF_rightdown.lng;
        rw.FAF_rightdown_verticalrelax.z = 0.0;
        LatLong2XY(regionID, rw.FAF_rightdown_verticalrelax.lat,
rw.FAF_rightdown_verticalrelax.lng, rw.FAF_rightdown_verticalrelax.x,
rw.FAF_rightdown_verticalrelax.y);
        //THR surrounding positions
        double inframex1, inframey1, inframex2, inframey2, inframelat1,
inframelat2, inframelng1, inframelng2;
        getPerpendicularPositions(LATERALHALFWIDTH_THR, rw.angle, rw.THR.x,
rw.THR.y, inframex1, inframey1, inframex2, inframey2);
        XY2LatLong(regionID, inframex1, inframey1, inframelat1, inframelng1);
        XY2LatLong(regionID, inframex2, inframey2, inframelat2, inframelng2);
        rw.THR_leftup.lat = inframelat1;
        rw.THR_leftup.lng = inframelng1;
        rw.THR_leftup.z = rw.THR.z + VERTICALHALFWIDTH_THR;
        LatLong2XY(regionID, rw.THR_leftup.lat, rw.THR_leftup.lng,
rw.THR_leftup.x, rw.THR_leftup.y);
        rw.THR_leftdown.lat = inframelat1;
        rw.THR_leftdown.lng = inframelng1;
        rw.THR_leftdown.z = rw.THR.z - VERTICALHALFWIDTH_THR;
        LatLong2XY(regionID, rw.THR_leftdown.lat, rw.THR_leftdown.lng,
rw.THR_leftdown.x, rw.THR_leftdown.y);
        rw.THR_rightup.lat = inframelat2;
        rw.THR_rightup.lng = inframelng2;
        rw.THR_rightup.z = rw.THR.z + VERTICALHALFWIDTH_THR;
        LatLong2XY(regionID, rw.THR_rightup.lat, rw.THR_rightup.lng,
rw.THR_rightup.x, rw.THR_rightup.y);
        rw.THR_rightdown.lat = inframelat2;
        rw.THR_rightdown.lng = inframelng2;
        rw.THR_rightdown.z = rw.THR.z - VERTICALHALFWIDTH_THR;
        LatLong2XY(regionID, rw.THR_rightdown.lat, rw.THR_rightdown.lng,
rw.THR_rightdown.x, rw.THR_rightdown.y);
            //horizontally relaxed
        double horizontalrelax_inframex1, horizontalrelax_inframey1,
horizontalrelax_inframex2, horizontalrelax_inframey2,
horizontalrelax_inframelat1, horizontalrelax_inframelat2,
horizontalrelax_inframelng1, horizontalrelax_inframelng2;
        getPerpendicularPositions(RELAXEDHALFWIDTH, rw.angle, rw.THR.x, rw.THR.y,
horizontalrelax_inframex1, horizontalrelax_inframey1,
horizontalrelax_inframex2, horizontalrelax_inframey2);
        XY2LatLong(regionID, horizontalrelax_inframex1,
horizontalrelax_inframey1, horizontalrelax_inframelat1,
horizontalrelax_inframelng1);
        XY2LatLong(regionID, horizontalrelax_inframex2,
horizontalrelax_inframey2, horizontalrelax_inframelat2,
horizontalrelax_inframelng2);
        rw.THR_leftup_horizontalrelax.lat = horizontalrelax_inframelat1;
        rw.THR_leftup_horizontalrelax.lng = horizontalrelax_inframelng1;
        rw.THR_leftup_horizontalrelax.z = rw.THR.z + VERTICALHALFWIDTH_THR;
```

```
       LatLong2XY(regionID, rw.THR_leftup_horizontalrelax.lat,
rw.THR_leftup_horizontalrelax.lng, rw.THR_leftup_horizontalrelax.x,
rw.THR_leftup_horizontalrelax.y);
       rw.THR_leftdown_horizontalrelax.lat = horizontalrelax_inframelat1;
       rw.THR_leftdown_horizontalrelax.lng = horizontalrelax_inframelng1;
       rw.THR_leftdown_horizontalrelax.z = rw.THR.z - VERTICALHALFWIDTH_THR;
       LatLong2XY(regionID, rw.THR_leftdown_horizontalrelax.lat,
rw.THR_leftdown_horizontalrelax.lng, rw.THR_leftdown_horizontalrelax.x,
rw.THR_leftdown_horizontalrelax.y);
       rw.THR_rightup_horizontalrelax.lat = horizontalrelax_inframelat2;
       rw.THR_rightup_horizontalrelax.lng = horizontalrelax_inframelng2;
       rw.THR_rightup_horizontalrelax.z = rw.THR.z + VERTICALHALFWIDTH_THR;
       LatLong2XY(regionID, rw.THR_rightup_horizontalrelax.lat,
rw.THR_rightup_horizontalrelax.lng, rw.THR_rightup_horizontalrelax.x,
rw.THR_rightup_horizontalrelax.y);
       rw.THR_rightdown_horizontalrelax.lat = horizontalrelax_inframelat2;
       rw.THR_rightdown_horizontalrelax.lng = horizontalrelax_inframelng2;
       rw.THR_rightdown_horizontalrelax.z = rw.THR.z - VERTICALHALFWIDTH_THR;
       LatLong2XY(regionID, rw.THR_rightdown_horizontalrelax.lat,
rw.THR_rightdown_horizontalrelax.lng, rw.THR_rightdown_horizontalrelax.x,
rw.THR_rightdown_horizontalrelax.y);
              //vertically relaxed
       rw.THR_leftup_verticalrelax.lat = rw.THR_leftup.lat;
       rw.THR_leftup_verticalrelax.lng = rw.THR_leftup.lng;
       rw.THR_leftup_verticalrelax.z = rw.THR_leftup.z + RELAXEDHALFWIDTH;
       LatLong2XY(regionID, rw.THR_leftup_verticalrelax.lat,
rw.THR_leftup_verticalrelax.lng, rw.THR_leftup_verticalrelax.x,
rw.THR_leftup_verticalrelax.y);
       rw.THR_leftdown_verticalrelax.lat = rw.THR_leftdown.lat;
       rw.THR_leftdown_verticalrelax.lng = rw.THR_leftdown.lng;
       rw.THR_leftdown_verticalrelax.z = 0.0;
       LatLong2XY(regionID, rw.THR_leftdown_verticalrelax.lat,
rw.THR_leftdown_verticalrelax.lng, rw.THR_leftdown_verticalrelax.x,
rw.THR_leftdown_verticalrelax.y);
       rw.THR_rightup_verticalrelax.lat = rw.THR_rightup.lat;
       rw.THR_rightup_verticalrelax.lng = rw.THR_rightup.lng;
       rw.THR_rightup_verticalrelax.z = rw.THR_rightup.z + RELAXEDHALFWIDTH;
       LatLong2XY(regionID, rw.THR_rightup_verticalrelax.lat,
rw.THR_rightup_verticalrelax.lng, rw.THR_rightup_verticalrelax.x,
rw.THR_rightup_verticalrelax.y);
       rw.THR_rightdown_verticalrelax.lat = rw.THR_rightdown.lat;
       rw.THR_rightdown_verticalrelax.lng = rw.THR_rightdown.lng;
       rw.THR_rightdown_verticalrelax.z = 0.0;
       LatLong2XY(regionID, rw.THR_rightdown_verticalrelax.lat,
rw.THR_rightdown_verticalrelax.lng, rw.THR_rightdown_verticalrelax.x,
rw.THR_rightdown_verticalrelax.y);
       //REF surrounding positions
       double REFframex1, REFframey1, REFframex2, REFframey2, REFframelat1,
REFframelat2, REFframelng1, REFframelng2; //the left and right side 2D location
(each corresponds to 2 vertical points)
       getPerpendicularPositions(LATERALHALFWIDTH_REF, rw.angle, rw.REF.x,
rw.REF.y, REFframex1, REFframey1, REFframex2, REFframey2);
       XY2LatLong(regionID, REFframex1, REFframey1, REFframelat1, REFframelng1);
       XY2LatLong(regionID, REFframex2, REFframey2, REFframelat2, REFframelng2);
       rw.REF_leftup.lat = REFframelat1;
       rw.REF_leftup.lng = REFframelng1;
       rw.REF_leftup.z = rw.REF.z + VERTICALHALFWIDTH_REF;
       LatLong2XY(regionID, rw.REF_leftup.lat, rw.REF_leftup.lng,
rw.REF_leftup.x, rw.REF_leftup.y);
       rw.REF_leftdown.lat = REFframelat1;
```

```
        rw.REF_leftdown.lng = REFframelng1;
        rw.REF_leftdown.z = rw.REF.z - VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_leftdown.lat, rw.REF_leftdown.lng,
rw.REF_leftdown.x, rw.REF_leftdown.y);
        rw.REF_rightup.lat = REFframelat2;
        rw.REF_rightup.lng = REFframelng2;
        rw.REF_rightup.z = rw.REF.z + VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_rightup.lat, rw.REF_rightup.lng,
rw.REF_rightup.x, rw.REF_rightup.y);
        rw.REF_rightdown.lat = REFframelat2;
        rw.REF_rightdown.lng = REFframelng2;
        rw.REF_rightdown.z = rw.REF.z - VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_rightdown.lat, rw.REF_rightdown.lng,
rw.REF_rightdown.x, rw.REF_rightdown.y);
              //horizontally relaxed
        double horizontalrelax_REFframex1, horizontalrelax_REFframey1,
horizontalrelax_REFframex2, horizontalrelax_REFframey2,
horizontalrelax_REFframelat1, horizontalrelax_REFframelat2,
horizontalrelax_REFframelng1, horizontalrelax_REFframelng2;
        getPerpendicularPositions(RELAXEDHALFWIDTH, rw.angle, rw.REF.x, rw.REF.y,
horizontalrelax_REFframex1, horizontalrelax_REFframey1,
horizontalrelax_REFframex2, horizontalrelax_REFframey2);
        XY2LatLong(regionID, horizontalrelax_REFframex1,
horizontalrelax_REFframey1, horizontalrelax_REFframelat1,
horizontalrelax_REFframelng1);
        XY2LatLong(regionID, horizontalrelax_REFframex2,
horizontalrelax_REFframey2, horizontalrelax_REFframelat2,
horizontalrelax_REFframelng2);
        rw.REF_leftup_horizontalrelax.lat = horizontalrelax_REFframelat1;
        rw.REF_leftup_horizontalrelax.lng = horizontalrelax_REFframelng1;
        rw.REF_leftup_horizontalrelax.z = rw.REF.z + VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_leftup_horizontalrelax.lat,
rw.REF_leftup_horizontalrelax.lng, rw.REF_leftup_horizontalrelax.x,
rw.REF_leftup_horizontalrelax.y);
        rw.REF_leftdown_horizontalrelax.lat = horizontalrelax_REFframelat1;
        rw.REF_leftdown_horizontalrelax.lng = horizontalrelax_REFframelng1;
        rw.REF_leftdown_horizontalrelax.z = rw.REF.z - VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_leftdown_horizontalrelax.lat,
rw.REF_leftdown_horizontalrelax.lng, rw.REF_leftdown_horizontalrelax.x,
rw.REF_leftdown_horizontalrelax.y);
        rw.REF_rightup_horizontalrelax.lat = horizontalrelax_REFframelat2;
        rw.REF_rightup_horizontalrelax.lng = horizontalrelax_REFframelng2;
        rw.REF_rightup_horizontalrelax.z = rw.REF.z + VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_rightup_horizontalrelax.lat,
rw.REF_rightup_horizontalrelax.lng, rw.REF_rightup_horizontalrelax.x,
rw.REF_rightup_horizontalrelax.y);
        rw.REF_rightdown_horizontalrelax.lat = horizontalrelax_REFframelat2;
        rw.REF_rightdown_horizontalrelax.lng = horizontalrelax_REFframelng2;
        rw.REF_rightdown_horizontalrelax.z = rw.REF.z - VERTICALHALFWIDTH_REF;
        LatLong2XY(regionID, rw.REF_rightdown_horizontalrelax.lat,
rw.REF_rightdown_horizontalrelax.lng, rw.REF_rightdown_horizontalrelax.x,
rw.REF_rightdown_horizontalrelax.y);
              //vertically relaxed
        rw.REF_leftup_verticalrelax.lat = rw.REF_leftup.lat;
        rw.REF_leftup_verticalrelax.lng = rw.REF_leftup.lng;
        rw.REF_leftup_verticalrelax.z = rw.REF_leftup.z + RELAXEDHALFWIDTH;
        LatLong2XY(regionID, rw.REF_leftup_verticalrelax.lat,
rw.REF_leftup_verticalrelax.lng, rw.REF_leftup_verticalrelax.x,
rw.REF_leftup_verticalrelax.y);
        rw.REF_leftdown_verticalrelax.lat = rw.REF_leftdown.lat;
```

```
	rw.REF_leftdown_verticalrelax.lng = rw.REF_leftdown.lng;
	rw.REF_leftdown_verticalrelax.z = 0.0;
	LatLong2XY(regionID, rw.REF_leftdown_verticalrelax.lat,
rw.REF_leftdown_verticalrelax.lng, rw.REF_leftdown_verticalrelax.x,
rw.REF_leftdown_verticalrelax.y);
	rw.REF_rightup_verticalrelax.lat = rw.REF_rightup.lat;
	rw.REF_rightup_verticalrelax.lng = rw.REF_rightup.lng;
	rw.REF_rightup_verticalrelax.z = rw.REF_leftup.z + RELAXEDHALFWIDTH;
	LatLong2XY(regionID, rw.REF_rightup_verticalrelax.lat,
rw.REF_rightup_verticalrelax.lng, rw.REF_rightup_verticalrelax.x,
rw.REF_rightup_verticalrelax.y);
	rw.REF_rightdown_verticalrelax.lat = rw.REF_rightdown.lat;
	rw.REF_rightdown_verticalrelax.lng = rw.REF_rightdown.lng;
	rw.REF_rightdown_verticalrelax.z = 0.0;
	LatLong2XY(regionID, rw.REF_rightdown_verticalrelax.lat,
rw.REF_rightdown_verticalrelax.lng, rw.REF_rightdown_verticalrelax.x,
rw.REF_rightdown_verticalrelax.y);
}
void loadAirportRunwaydata() {
	//runway_initialize(rw,   name,  true_alignment, vgpa,         elevation,
THRcrossheight, THR_lat, THR_lng, dist_FAF2THR)
	//JFK runways
	runway_initialize(JFK13R, "13R", 121*deg2rad, 3.0*deg2rad, 12.5*ft2m,
73*ft2m, 40.645339, -73.810057, FAF2THR_DIST);
	runway_initialize(JFK31L, "31L", 301*deg2rad, 3.0*deg2rad, 11.8*ft2m,
67*ft2m, 40.632726, -73.782203, FAF2THR_DIST);
	runway_initialize(JFK04L, "04L", 31*deg2rad, 3.0*deg2rad, 11.5*ft2m,
72*ft2m, 40.622276, -73.785389, FAF2THR_DIST);
	runway_initialize(JFK22R, "22R", 211*deg2rad, 3.0*deg2rad, 11.5*ft2m,
72*ft2m, 40.64219, -73.769836, FAF2THR_DIST);
	runway_initialize(JFK13L, "13L", 121*deg2rad, 3.0*deg2rad, 11.3*ft2m,
65*ft2m, 40.656344, -73.78714, FAF2THR_DIST);///2.75*deg2rad
	runway_initialize(JFK31R, "31R", 301*deg2rad, 3.0*deg2rad, 11.8*ft2m,
65*ft2m, 40.645306, -73.762775, FAF2THR_DIST);
	runway_initialize(JFK04R, "04R", 31*deg2rad, 3.0*deg2rad, 11.8*ft2m,
66*ft2m, 40.625664, -73.770168, FAF2THR_DIST);
	runway_initialize(JFK22L, "22L", 211*deg2rad, 3.0*deg2rad, 11.8*ft2m,
66*ft2m, 40.645027, -73.755045, FAF2THR_DIST);
	JFKrunways.push_back(JFK13R); JFKrunways.push_back(JFK31L);
JFKrunways.push_back(JFK04L); JFKrunways.push_back(JFK22R);
JFKrunways.push_back(JFK13L); JFKrunways.push_back(JFK31R);
JFKrunways.push_back(JFK04R); JFKrunways.push_back(JFK22L);
	//LGA runways
	runway_initialize(LGA13, "13", 122*deg2rad, 3.0*deg2rad, 11.6*ft2m,
55*ft2m, 40.782166, -73.878242, FAF2THR_DIST);///3.1*deg2rad
	runway_initialize(LGA31, "31", 302*deg2rad, 3.0*deg2rad, 6.7*ft2m,
71*ft2m, 40.772216, -73.857406, FAF2THR_DIST);
	runway_initialize(LGA04, "04", 32*deg2rad, 3.0*deg2rad, 20.6*ft2m,
76*ft2m, 40.769411, -73.883913, FAF2THR_DIST);///3.1*deg2rad
	runway_initialize(LGA22, "22", 212*deg2rad, 3.0*deg2rad, 11.5*ft2m,
67*ft2m, 40.785188, -73.87088, FAF2THR_DIST);
	LGArunways.push_back(LGA13); LGArunways.push_back(LGA31);
LGArunways.push_back(LGA04); LGArunways.push_back(LGA22);
	//EWR runways
	runway_initialize(EWR11, "11", 95*deg2rad, 3.0*deg2rad, 17.6*ft2m,
53*ft2m, 40.702774, -74.180353, FAF2THR_DIST);///3.08*deg2rad
	runway_initialize(EWR29, "29", 275*deg2rad, 3.0*deg2rad, 9.9*ft2m,
60*ft2m, 40.701273, -74.157704, FAF2THR_DIST);
	runway_initialize(EWR04R, "04R", 26*deg2rad, 3.0*deg2rad, 10.8*ft2m,
72*ft2m, 40.680783, -74.172216, FAF2THR_DIST);
```

```
	runway_initialize(EWR22L, "22L", 206*deg2rad, 3.0*deg2rad, 9.1*ft2m,
61*ft2m, 40.697602, -74.161525, FAF2THR_DIST);
	runway_initialize(EWR04L, "04L", 26*deg2rad, 3.0*deg2rad, 9.9*ft2m,
77*ft2m, 40.681925, -74.175291, FAF2THR_DIST);///3.1*deg2rad
	runway_initialize(EWR22R, "22R", 206*deg2rad, 3.0*deg2rad, 8.6*ft2m,
70*ft2m, 40.698785, -74.164568, FAF2THR_DIST);///3.1*deg2rad
	EWRrunways.push_back(EWR11); EWRrunways.push_back(EWR29);
EWRrunways.push_back(EWR04R); EWRrunways.push_back(EWR22L);
EWRrunways.push_back(EWR04L); EWRrunways.push_back(EWR22R);
	//MDW runways
	runway_initialize(MDW04L, "04L", 43*deg2rad, 3.0*deg2rad, 618.5*ft2m,
39*ft2m, 41.783079, -87.759042, FAF2THR_DIST);
	runway_initialize(MDW22R, "22R", 223*deg2rad, 3.0*deg2rad, 605.2*ft2m,
35*ft2m, 41.790365, -87.749823, FAF2THR_DIST);
	runway_initialize(MDW04R, "04R", 43*deg2rad, 3.0*deg2rad, 619.7*ft2m,
66*ft2m, 41.780396, -87.757765, FAF2THR_DIST);
	runway_initialize(MDW22L, "22L", 223*deg2rad, 3.0*deg2rad, 605.2*ft2m,
43*ft2m, 41.790518, -87.744949, FAF2THR_DIST);
	runway_initialize(MDW13C, "13C", 134*deg2rad, 3.0*deg2rad, 607*ft2m,
46*ft2m, 41.790523, -87.759593, FAF2THR_DIST);
	runway_initialize(MDW31C, "31C", 314*deg2rad, 3.0*deg2rad, 611.6*ft2m,
63*ft2m, 41.780779, -87.745896, FAF2THR_DIST);
	runway_initialize(MDW13L, "13L", 134*deg2rad, 3.0*deg2rad, 606.1*ft2m,
44*ft2m, 41.790743, -87.755717, FAF2THR_DIST); //no records for crossing height
	runway_initialize(MDW31R, "31R", 314*deg2rad, 3.0*deg2rad, 608.3*ft2m,
44*ft2m, 41.782803, -87.744561, FAF2THR_DIST);
	runway_initialize(MDW13R, "13R", 134*deg2rad, 3.0*deg2rad, 606.9*ft2m,
44*ft2m, 41.787952, -87.758615, FAF2THR_DIST); //no records for crossing height
	runway_initialize(MDW31L, "31L", 314*deg2rad, 3.0*deg2rad, 612.9*ft2m,
44*ft2m, 41.781193, -87.749123, FAF2THR_DIST); //no records for crossing height
	MDWrunways.push_back(MDW04L); MDWrunways.push_back(MDW22R);
MDWrunways.push_back(MDW04R); MDWrunways.push_back(MDW22L);
MDWrunways.push_back(MDW13C);
	MDWrunways.push_back(MDW31C); MDWrunways.push_back(MDW13L);
MDWrunways.push_back(MDW31R); MDWrunways.push_back(MDW13R);
MDWrunways.push_back(MDW31L);
	//default runway
	runway_initialize(RWpending, runwayname_default, 0, 0, 0, 0, 0, 0, 0);
}
void load_flights(vector<FLIGHT> *p_Flights, string flightfile, AIRCRAFT
*p_aircraft, int aircraft_data_size) {
	//load flightID, ACFT_TYPE, TRACK_INDEX from flight data file
	AIRCRAFT *p_initial = p_aircraft;
	ifstream in(flightfile);
	int ncolumns = 3;
	vector<string> cluster(ncolumns);
	char chars[200];
	in.getline(chars, 200); //skip header line
	bool aircarftfound;
	while(in.getline(chars, 200)) {
		FLIGHT tempflight;
		p_aircraft = p_initial;
		split(chars, ",", cluster, 0);
		tempflight.flightID = atol(cluster[0].c_str());	//flightID
		aircarftfound = false;
		for (int kk = 1; kk <= aircraft_data_size; ++kk) { //search for
aircraft info in aircraft data
			if (p_aircraft->model == cluster[1].c_str()) {
				tempflight.aircraft = *p_aircraft; //aircraft type
				aircarftfound = true;
```

```
                break;
        }
        p_aircraft++;
}
if (aircarftfound == false) {
        tempflight.aircraft.model = cluster[1].c_str();
        tempflight.aircraft.MTOW = 0.0;
        tempflight.aircraft.weight_class = "Unknown...";
}
tempflight.trackindex = cluster[2].c_str(); //track index
tempflight.date = tempflight.trackindex.substr(0,8); //"20070111"
//initialize all other attributes
tempflight.landingrunway = RWpending;
tempflight.j_3nm = j_default;
tempflight.headingdiff_12nm = 0.0;
tempflight.j_2dot25nm = j_default;
tempflight.j_1dot5nm = j_default;
tempflight.j_THR = j_default;
tempflight.j_10nm_r = j_default;
tempflight.j_9nm_r = j_default;
tempflight.j_8nm_r = j_default;
tempflight.j_7nm_r = j_default;
tempflight.j_6nm_r = j_default;
tempflight.j_5dot5nm_r = j_default;
tempflight.j_5nm_r = j_default;
tempflight.j_4dot5nm_r = j_default;
tempflight.j_4nm_r = j_default;
tempflight.j_3dot5nm_r = j_default;
tempflight.j_3nm_r = j_default;
tempflight.j_1000ftAGL = j_default;
tempflight.j_750ftAGL = j_default;
tempflight.j_500ftAGL = j_default;
tempflight.j_firstentrance = j_default;
tempflight.j_beforeentrance = j_default;
tempflight.j_acquisition_lateral = j_default;
tempflight.j_acquisition_vertical = j_default;
tempflight.j_beforelateralintercept = j_default;
tempflight.j_beforeverticalintercept = j_default;
tempflight.j_deceleration = j_default;
tempflight.dist_deceleration = -1.0;
tempflight.entrance = false;
tempflight.track_qualified = false;
tempflight.go_around = false;
tempflight.v_dev_10nm = 0.0;
tempflight.v_dev_9nm = 0.0;
tempflight.v_dev_8nm = 0.0;
tempflight.v_dev_7nm = 0.0;
tempflight.v_dev_6nm = 0.0;
tempflight.v_dev_5dot5nm = 0.0;
tempflight.v_dev_5nm = 0.0;
tempflight.v_dev_4dot5nm = 0.0;
tempflight.v_dev_4nm = 0.0;
tempflight.v_dev_3dot5nm = 0.0;
tempflight.v_dev_3nm = 0.0;
tempflight.rod_dev_10nm = 0.0;
tempflight.rod_dev_9nm = 0.0;
tempflight.rod_dev_8nm = 0.0;
tempflight.rod_dev_7nm = 0.0;
tempflight.rod_dev_6nm = 0.0;
tempflight.rod_dev_5dot5nm = 0.0;
```

```
tempflight.rod_dev_5nm = 0.0;
tempflight.rod_dev_4dot5nm = 0.0;
tempflight.rod_dev_4nm = 0.0;
tempflight.rod_dev_3dot5nm = 0.0;
tempflight.rod_dev_3nm = 0.0;
tempflight.lateral_deviation_abs_10nm = 0.0;
tempflight.vertical_deviation_abs_10nm = 0.0;
tempflight.lateral_deviation_abs_9nm = 0.0;
tempflight.vertical_deviation_abs_9nm = 0.0;
tempflight.lateral_deviation_abs_8nm = 0.0;
tempflight.vertical_deviation_abs_8nm = 0.0;
tempflight.lateral_deviation_abs_7nm = 0.0;
tempflight.vertical_deviation_abs_7nm = 0.0;
tempflight.lateral_deviation_abs_6nm = 0.0;
tempflight.vertical_deviation_abs_6nm = 0.0;
tempflight.lateral_deviation_abs_5dot5nm = 0.0;
tempflight.vertical_deviation_abs_5dot5nm = 0.0;
tempflight.lateral_deviation_abs_5nm = 0.0;
tempflight.vertical_deviation_abs_5nm = 0.0;
tempflight.lateral_deviation_abs_4dot5nm = 0.0;
tempflight.vertical_deviation_abs_4dot5nm = 0.0;
tempflight.lateral_deviation_abs_4nm = 0.0;
tempflight.vertical_deviation_abs_4nm = 0.0;
tempflight.lateral_deviation_abs_3dot5nm = 0.0;
tempflight.vertical_deviation_abs_3dot5nm = 0.0;
tempflight.lateral_deviation_abs_3nm = 0.0;
tempflight.vertical_deviation_abs_3nm = 0.0;
tempflight.avg_groundspeed_10nm = 0.0;
tempflight.avg_rateofdescent_10nm = 0.0;
tempflight.lateral_deviation_10nm = 0.0;
tempflight.vertical_deviation_10nm = 0.0;
tempflight.heading_diff_10nm = 0.0;
tempflight.avg_airspeedcategory_10nm = "";
tempflight.avg_rateofdescent_category_10nm = "";
tempflight.lateral_deviation_category_10nm = "";
tempflight.vertical_deviation_category_10nm = "";
tempflight.heading_diff_category_10nm = "";
tempflight.n_entrance_before_10nm = 0;
tempflight.rel_dist_to_10nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_10nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_9nm = 0.0;
tempflight.avg_rateofdescent_9nm = 0.0;
tempflight.lateral_deviation_9nm = 0.0;
tempflight.vertical_deviation_9nm = 0.0;
tempflight.heading_diff_9nm = 0.0;
tempflight.avg_airspeedcategory_9nm = "";
tempflight.avg_rateofdescent_category_9nm = "";
tempflight.lateral_deviation_category_9nm = "";
tempflight.vertical_deviation_category_9nm = "";
tempflight.heading_diff_category_9nm = "";
tempflight.n_entrance_before_9nm = 0;
tempflight.rel_dist_to_9nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_9nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_8nm = 0.0;
tempflight.avg_rateofdescent_8nm = 0.0;
tempflight.lateral_deviation_8nm = 0.0;
tempflight.vertical_deviation_8nm = 0.0;
tempflight.heading_diff_8nm = 0.0;
tempflight.avg_airspeedcategory_8nm = "";
tempflight.avg_rateofdescent_category_8nm = "";
```

```
tempflight.lateral_deviation_category_8nm = "";
tempflight.vertical_deviation_category_8nm = "";
tempflight.heading_diff_category_8nm = "";
tempflight.n_entrance_before_8nm = 0;
tempflight.rel_dist_to_8nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_8nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_7nm = 0.0;
tempflight.avg_rateofdescent_7nm = 0.0;
tempflight.lateral_deviation_7nm = 0.0;
tempflight.vertical_deviation_7nm = 0.0;
tempflight.heading_diff_7nm = 0.0;
tempflight.avg_airspeedcategory_7nm = "";
tempflight.avg_rateofdescent_category_7nm = "";
tempflight.lateral_deviation_category_7nm = "";
tempflight.vertical_deviation_category_7nm = "";
tempflight.heading_diff_category_7nm = "";
tempflight.n_entrance_before_7nm = 0;
tempflight.rel_dist_to_7nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_7nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_6nm = 0.0;
tempflight.avg_rateofdescent_6nm = 0.0;
tempflight.lateral_deviation_6nm = 0.0;
tempflight.vertical_deviation_6nm = 0.0;
tempflight.heading_diff_6nm = 0.0;
tempflight.avg_airspeedcategory_6nm = "";
tempflight.avg_rateofdescent_category_6nm = "";
tempflight.lateral_deviation_category_6nm = "";
tempflight.vertical_deviation_category_6nm = "";
tempflight.heading_diff_category_6nm = "";
tempflight.n_entrance_before_6nm = 0;
tempflight.rel_dist_to_6nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_6nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_5dot5nm = 0.0;
tempflight.avg_rateofdescent_5dot5nm = 0.0;
tempflight.lateral_deviation_5dot5nm = 0.0;
tempflight.vertical_deviation_5dot5nm = 0.0;
tempflight.heading_diff_5dot5nm = 0.0;
tempflight.avg_airspeedcategory_5dot5nm = "";
tempflight.avg_rateofdescent_category_5dot5nm = "";
tempflight.lateral_deviation_category_5dot5nm = "";
tempflight.vertical_deviation_category_5dot5nm = "";
tempflight.heading_diff_category_5dot5nm = "";
tempflight.n_entrance_before_5dot5nm = 0;
tempflight.rel_dist_to_5dot5nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_5dot5nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_5nm = 0.0;
tempflight.avg_rateofdescent_5nm = 0.0;
tempflight.lateral_deviation_5nm = 0.0;
tempflight.vertical_deviation_5nm = 0.0;
tempflight.heading_diff_5nm = 0.0;
tempflight.avg_airspeedcategory_5nm = "";
tempflight.avg_rateofdescent_category_5nm = "";
tempflight.lateral_deviation_category_5nm = "";
tempflight.vertical_deviation_category_5nm = "";
tempflight.heading_diff_category_5nm = "";
tempflight.n_entrance_before_5nm = 0;
tempflight.rel_dist_to_5nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_5nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_4dot5nm = 0.0;
tempflight.avg_rateofdescent_4dot5nm = 0.0;
```

```
tempflight.lateral_deviation_4dot5nm = 0.0;
tempflight.vertical_deviation_4dot5nm = 0.0;
tempflight.heading_diff_4dot5nm = 0.0;
tempflight.avg_airspeedcategory_4dot5nm = "";
tempflight.avg_rateofdescent_category_4dot5nm = "";
tempflight.lateral_deviation_category_4dot5nm = "";
tempflight.vertical_deviation_category_4dot5nm = "";
tempflight.heading_diff_category_4dot5nm = "";
tempflight.n_entrance_before_4dot5nm = 0;
tempflight.rel_dist_to_4dot5nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_4dot5nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_4nm = 0.0;
tempflight.avg_rateofdescent_4nm = 0.0;
tempflight.lateral_deviation_4nm = 0.0;
tempflight.vertical_deviation_4nm = 0.0;
tempflight.heading_diff_4nm = 0.0;
tempflight.avg_airspeedcategory_4nm = "";
tempflight.avg_rateofdescent_category_4nm = "";
tempflight.lateral_deviation_category_4nm = "";
tempflight.vertical_deviation_category_4nm = "";
tempflight.heading_diff_category_4nm = "";
tempflight.n_entrance_before_4nm = 0;
tempflight.rel_dist_to_4nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_4nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_3dot5nm = 0.0;
tempflight.avg_rateofdescent_3dot5nm = 0.0;
tempflight.lateral_deviation_3dot5nm = 0.0;
tempflight.vertical_deviation_3dot5nm = 0.0;
tempflight.heading_diff_3dot5nm = 0.0;
tempflight.avg_airspeedcategory_3dot5nm = "";
tempflight.avg_rateofdescent_category_3dot5nm = "";
tempflight.lateral_deviation_category_3dot5nm = "";
tempflight.vertical_deviation_category_3dot5nm = "";
tempflight.heading_diff_category_3dot5nm = "";
tempflight.n_entrance_before_3dot5nm = 0;
tempflight.rel_dist_to_3dot5nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_3dot5nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_3nm = 0.0;
tempflight.avg_rateofdescent_3nm = 0.0;
tempflight.lateral_deviation_3nm = 0.0;
tempflight.vertical_deviation_3nm = 0.0;
tempflight.heading_diff_3nm = 0.0;
tempflight.avg_airspeedcategory_3nm = "";
tempflight.avg_rateofdescent_category_3nm = "";
tempflight.lateral_deviation_category_3nm = "";
tempflight.vertical_deviation_category_3nm = "";
tempflight.heading_diff_category_3nm = "";
tempflight.n_entrance_before_3nm = 0;
tempflight.rel_dist_to_3nm_lateral_acquisition_prj = 0.0;
tempflight.rel_dist_to_3nm_vertical_acquisition_prj = 0.0;
tempflight.avg_groundspeed_1000ftAGL = 0.0;
tempflight.avg_rateofdescent_1000ftAGL = 0.0;
tempflight.lateral_deviation_1000ftAGL = 0.0;
tempflight.vertical_deviation_1000ftAGL = 0.0;
tempflight.avg_airspeedcategory_1000ftAGL = "";
tempflight.avg_rateofdescent_category_1000ftAGL = "";
tempflight.lateral_deviation_category_1000ftAGL = "";
tempflight.vertical_deviation_category_1000ftAGL = "";
tempflight.avg_groundspeed_750ftAGL = 0.0;
tempflight.avg_rateofdescent_750ftAGL = 0.0;
```

```
                tempflight.lateral_deviation_750ftAGL = 0.0;
                tempflight.vertical_deviation_750ftAGL = 0.0;
                tempflight.avg_airspeedcategory_750ftAGL = "";
                tempflight.avg_rateofdescent_category_750ftAGL = "";
                tempflight.lateral_deviation_category_750ftAGL = "";
                tempflight.vertical_deviation_category_750ftAGL = "";
                tempflight.avg_groundspeed_500ftAGL = 0.0;
                tempflight.avg_rateofdescent_500ftAGL = 0.0;
                tempflight.lateral_deviation_500ftAGL = 0.0;
                tempflight.vertical_deviation_500ftAGL = 0.0;
                tempflight.avg_airspeedcategory_500ftAGL = "";
                tempflight.avg_rateofdescent_category_500ftAGL = "";
                tempflight.lateral_deviation_category_500ftAGL = "";
                tempflight.vertical_deviation_category_500ftAGL = "";
                tempflight.avg_groundspeed_THR = 0.0;
                tempflight.time2landing_entrance = 0.0;
                tempflight.dist2THR_lateral_acquisition_prj = 0.0;
                tempflight.dist2THR_vertical_acquisition_prj = 0.0;
                tempflight.USD1000 = false;
                tempflight.USD750 = false;
                tempflight.USD500 = false;
                tempflight.USA1000 = false;
                tempflight.USA750 = false;
                tempflight.USA500 = false;
                tempflight.ERD1000 = false;
                tempflight.ERD750 = false;
                tempflight.ERD500 = false;
                tempflight.SAA1000 = false;
                tempflight.SAA750 = false;
                tempflight.SAA500 = false;
                tempflight.SAB1000 = false;
                tempflight.SAB750 = false;
                tempflight.SAB500 = false;
                tempflight.SAL1000 = false;
                tempflight.SAL750 = false;
                tempflight.SAL500 = false;
                tempflight.SAR1000 = false;
                tempflight.SAR750 = false;
                tempflight.SAR500 = false;
                tempflight.stable1000 = false;
                tempflight.stable750 = false;
                tempflight.stable500 = false;
                tempflight.unstable1000 = false;
                tempflight.unstable750 = false;
                tempflight.unstable500 = false;
                tempflight.levelsegmentflighttime = 0.0;
                tempflight.procedure = procedure_default;
                tempflight.flow = flow_default;
                (*p_Flights).push_back(tempflight);
        }
}
void load_tracks(vector<FLIGHT> *p_Flights, string trackfile, int i_last) {
        //read preprocessed format track data, load into Flights, starting from
Flights[(i_last+1)-1], if no previous flights with tracks initialized, i_last =
0
        //for PDARS data, need first to use the preprocessed track files
generated by function "preprocessPDARStracks()",
        //  which extracts flights flying to the target destination from a
mixture of flights by checking the flight id...
        //  and also converts original format to preprocessed format:
```

```
	//	(time converted to single value (double), altitude converted to
unit in meters, eliminate heading and speed columns)
	//the preprocessed file should share same format with preprocessed N90
data

	int pre_flightID;
	TRACKPOINT temptrackpoint;
	ifstream in(trackfile);
	//in.precision(10);
	int ncolumns = 5;
	vector<string> cluster(ncolumns);
	char chars[200];		// maximum characters in a row
	in.getline(chars, 200);		// skip header line

	// for first flight (initialize)
	in.getline(chars, 200);
	split(chars, ",", cluster, 0);
	pre_flightID = atol(cluster[0].c_str());
	//flightID, first column
	temptrackpoint.t = atof(cluster[1].c_str());		//seconds
past midnight, second column
	temptrackpoint.lat = atof(cluster[2].c_str());		//lat, third
column
	temptrackpoint.lng = atof(cluster[3].c_str());		//lng, fourth
column
	temptrackpoint.z = atof(cluster[4].c_str());		//altitude
AMSL, in metrics in data file, fifth column
	LatLong2XY(regionID, temptrackpoint.lat, temptrackpoint.lng,
temptrackpoint.x, temptrackpoint.y); // x, y
	//default values for derived info
	temptrackpoint.groundspeed = 0.0;
	temptrackpoint.verticalspeed = 0.0;
	temptrackpoint.heading = 0.0;
	(*p_Flights)[(i_last+1)-1].Track.push_back(temptrackpoint);  //starting
from Flights[(i_last+1)-1]

	int i = i_last + 1; //the first new one
	while(in.getline(chars, 200)) {
		// flightID, ACFT_TYPE, SECONDS_PAST_MIDNIGHT_GMT, LATITUDE,
LONGITUDE, ALTITUDE, TRACK_INDEX
		split(chars, ",", cluster, 0);
		if(atol(cluster[0].c_str()) != pre_flightID) {
	// if new flight
//			cout << "reading track " << cluster[0] << "\n";
			i = i + 1;
			pre_flightID = (*p_Flights)[i-1].flightID;
		//update pre_
		}
		temptrackpoint.t = atof(cluster[1].c_str());
	// SECONDS_PAST_MIDNIGHT_GMT
		temptrackpoint.lat = atof(cluster[2].c_str());
	// LATITUDE
		temptrackpoint.lng = atof(cluster[3].c_str());
	// LONGITUDE
		temptrackpoint.z = atof(cluster[4].c_str());
	// ALTITUDE (converted to metrics in input data file, assume in
FightLevel in raw format)
		LatLong2XY(regionID, temptrackpoint.lat, temptrackpoint.lng,
temptrackpoint.x, temptrackpoint.y);
		//default values for derived info
```

```
                temptrackpoint.groundspeed = 0.0;
                temptrackpoint.verticalspeed = 0.0;
                temptrackpoint.heading = 0.0;
                (*p_Flights)[i-1].Track.push_back(temptrackpoint);
        }
}
vector<WINDPOINT> load_wind(string windfile) {
        vector<WINDPOINT> wind;
        WINDPOINT tempwindpoint;
        ifstream in(windfile);
        int ncolumns = 4;
        vector<string> cluster(ncolumns);
        char chars[200];                  // maximum characters in a row
        in.getline(chars, 200);           // skip header line
        while(in.getline(chars, 200)) {
                split(chars, ",", cluster, 0);
                tempwindpoint.t = atof(cluster[0].c_str());
                tempwindpoint.heading = atof(cluster[1].c_str());
                tempwindpoint.speed = atof(cluster[2].c_str());

                tempwindpoint.gust = atof(cluster[3].c_str());
                wind.push_back(tempwindpoint);
        }
        return wind;
}
WINDPOINT get_current_wind(string date, double t) {
        WINDPOINT outputwind; outputwind.gust = 0.0; outputwind.heading = 0.0;
outputwind.speed = 0.0; outputwind.t = 0.0;
        vector<WINDPOINT> currentwindvector;
        //hardcodded for EWR 22L
        if (date == "20070111") currentwindvector = wind20070111;
        else if (date == "20070112") currentwindvector = wind20070112;
        else if (date == "20070113") currentwindvector = wind20070113;
        else if (date == "20070114") currentwindvector = wind20070114;
        else if (date == "20070115") currentwindvector = wind20070115;
        else if (date == "20070116") currentwindvector = wind20070116;
        else if (date == "20070117") currentwindvector = wind20070117;
        else if (date == "20070323") currentwindvector = wind20070323;
        else if (date == "20070324") currentwindvector = wind20070324;
        else if (date == "20070325") currentwindvector = wind20070325;
        else if (date == "20070326") currentwindvector = wind20070326;
        else if (date == "20070327") currentwindvector = wind20070327;
        else if (date == "20070328") currentwindvector = wind20070328;
        else if (date == "20070329") currentwindvector = wind20070329;
        else if (date == "20070721") currentwindvector = wind20070721;
        else if (date == "20070722") currentwindvector = wind20070722;
        else if (date == "20070723") currentwindvector = wind20070723;
        else if (date == "20070724") currentwindvector = wind20070724;
        else if (date == "20070725") currentwindvector = wind20070725;
        else if (date == "20070726") currentwindvector = wind20070726;
        else if (date == "20070727") currentwindvector = wind20070727;
        else if (date == "20071001") currentwindvector = wind20071001;
        else if (date == "20071002") currentwindvector = wind20071002;
        else if (date == "20071003") currentwindvector = wind20071003;
        else if (date == "20071004") currentwindvector = wind20071004;
        else if (date == "20071005") currentwindvector = wind20071005;
        else if (date == "20071006") currentwindvector = wind20071006;
        else if (date == "20071007") currentwindvector = wind20071007;

        if (t < currentwindvector[0].t)
```

```
			outputwind = currentwindvector[0];
		else if (t > currentwindvector[currentwindvector.size()-1].t)
			outputwind = currentwindvector[currentwindvector.size()-1];
		else {
			double anglediff1, anglediff2, anglediff3;
			for (int i = 1; i <= currentwindvector.size()-1; ++i) {
				if (t >= currentwindvector[i-1].t && t <
currentwindvector[i].t) {
					outputwind.t = t;
					outputwind.speed = currentwindvector[i-1].speed + ((t
- currentwindvector[i-1].t)/(currentwindvector[i].t - currentwindvector[i-
1].t))*(currentwindvector[i].speed - currentwindvector[i-1].speed);
					outputwind.gust = currentwindvector[i-1].gust + ((t -
currentwindvector[i-1].t)/(currentwindvector[i].t - currentwindvector[i-
1].t))*(currentwindvector[i].gust - currentwindvector[i-1].gust);
					anglediff1 = currentwindvector[i-1].heading -
currentwindvector[i].heading;
					anglediff2 = currentwindvector[i-1].heading -
currentwindvector[i].heading + 2*PI;
					anglediff3 = currentwindvector[i-1].heading -
currentwindvector[i].heading - 2*PI;
					if (abs(anglediff1) < abs(anglediff2) &&
abs(anglediff1) < abs(anglediff3)) {
						outputwind.heading = currentwindvector[i-
1].heading + ((t - currentwindvector[i-1].t)/(currentwindvector[i].t -
currentwindvector[i-1].t))*(currentwindvector[i].heading - currentwindvector[i-
1].heading);
					}
					if (abs(anglediff2) < abs(anglediff1) &&
abs(anglediff2) < abs(anglediff3)) {
						outputwind.heading = currentwindvector[i-
1].heading + 2*PI + ((t - currentwindvector[i-1].t)/(currentwindvector[i].t -
currentwindvector[i-1].t))*(currentwindvector[i].heading - currentwindvector[i-
1].heading);
					}
					if (abs(anglediff3) < abs(anglediff1) &&
abs(anglediff3) < abs(anglediff2)) {
						outputwind.heading = currentwindvector[i-
1].heading - 2*PI + ((t - currentwindvector[i-1].t)/(currentwindvector[i].t -
currentwindvector[i-1].t))*(currentwindvector[i].heading - currentwindvector[i-
1].heading);
					}
					break;
				}
			}
		}

		return outputwind;
}
void plot_finalapproachsegment(string outputfile, vector<RUNWAY> Runways) {
		ofstream out(outputfile);
		out.precision(10);
		out << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";			// XML
header
		out << "<kml xmlns=\"http://www.opengis.net/kml/2.2\">\n";  // KML
namespace declaration
		out << "<Document>\n";
		out << "<name>" << "wireframes_test" << "</name>\n";

		out << "<Style id=\"" << "default" << "\">\n";
```

```
	out << "    <LineStyle>\n";
	out << "        <color>" << "ff00ff00" << "</color>\n";
	out << "        <width>" << 2 << "</width>\n";
	out << "    </LineStyle>\n";
	out << "</Style>\n";

	for (int i = 1; i <= Runways.size(); ++i) {
	//centerline
		out << "<Placemark>\n";
		out << "<styleUrl>#default</styleUrl>\n";
		out << "<LineString>\n";
		out << "<altitudeMode>absolute</altitudeMode>\n";// absolute
values above sea level
		out << "<coordinates>\n";
		kmlPoint(out, Runways[i-1].THR);
		kmlPoint(out, Runways[i-1].FAF);
		kmlPoint(out, Runways[i-1].location_12nm);
		out << "</coordinates>\n";
		out << "</LineString>\n</Placemark>\n";

	//wireframes
		out << "<Placemark>\n";
		out << "<styleUrl>#default</styleUrl>\n";
		out << "<LineString>\n";
		out << "<altitudeMode>absolute</altitudeMode>\n";// absolute
values above see level
		out << "<coordinates>\n";
		//REF square
		kmlPoint(out, Runways[i-1].REF_leftup);
		kmlPoint(out, Runways[i-1].REF_leftdown);
		kmlPoint(out, Runways[i-1].REF_rightdown);
		kmlPoint(out, Runways[i-1].REF_rightup);
		kmlPoint(out, Runways[i-1].REF_leftup);
		//THR square
		kmlPoint(out, Runways[i-1].THR_leftup);
		kmlPoint(out, Runways[i-1].THR_rightup);
		kmlPoint(out, Runways[i-1].THR_rightdown);
		kmlPoint(out, Runways[i-1].THR_leftdown);
		kmlPoint(out, Runways[i-1].THR_leftup);
		//connect THR and REF
		kmlPoint(out, Runways[i-1].THR_leftdown);
		kmlPoint(out, Runways[i-1].REF_leftdown);
		kmlPoint(out, Runways[i-1].REF_rightdown);
		kmlPoint(out, Runways[i-1].THR_rightdown);
		kmlPoint(out, Runways[i-1].THR_rightup);
		kmlPoint(out, Runways[i-1].REF_rightup);
		//connect REF and FAF
		kmlPoint(out, Runways[i-1].FAF_rightup);
		kmlPoint(out, Runways[i-1].FAF_leftup);
		kmlPoint(out, Runways[i-1].REF_leftup);
		kmlPoint(out, Runways[i-1].FAF_leftup);
		kmlPoint(out, Runways[i-1].FAF_leftdown);
		kmlPoint(out, Runways[i-1].REF_leftdown);
		kmlPoint(out, Runways[i-1].FAF_leftdown);
		kmlPoint(out, Runways[i-1].FAF_rightdown);
		kmlPoint(out, Runways[i-1].REF_rightdown);
		kmlPoint(out, Runways[i-1].FAF_rightdown);
		kmlPoint(out, Runways[i-1].FAF_rightup);
		//connect FAF and 12nm
		kmlPoint(out, Runways[i-1].LOC_12nm_rightup);
```

```
			kmlPoint(out, Runways[i-1].LOC_12nm_rightdown);
			kmlPoint(out, Runways[i-1].FAF_rightdown);
			kmlPoint(out, Runways[i-1].LOC_12nm_rightdown);
			kmlPoint(out, Runways[i-1].LOC_12nm_leftdown);
			kmlPoint(out, Runways[i-1].FAF_leftdown);
			kmlPoint(out, Runways[i-1].LOC_12nm_leftdown);
			kmlPoint(out, Runways[i-1].LOC_12nm_leftup);
			kmlPoint(out, Runways[i-1].FAF_leftup);
			kmlPoint(out, Runways[i-1].LOC_12nm_leftup);
			kmlPoint(out, Runways[i-1].LOC_12nm_rightup);

			out << "</coordinates>\n";
			out << "</LineString>\n</Placemark>\n";
		}
		out << "</Document>\n";
		out << "</kml>\n";
		out.close();
}
void identify_runway(vector<FLIGHT> &Flights, string airportname) {
		///currently include EWR,JFK,LGA,MDW only
		for (int i = 1; i <= Flights.size(); i++) {
			//find the last point the flight left within a wireframe of some
runway
			Flights[i-1].landingrunway = RWpending; // default initialization
			double dist_traveled_inside = 0.0;
			double time_traveled_inside = 0.0;
			RUNWAY lastrunway = RWpending;
			for (int j = Flights[i-1].Track.size(); j >= 1; j--) {
				Point p;
				p.lat = Flights[i-1].Track[j-1].lat;
				p.lng = Flights[i-1].Track[j-1].lng;
				p.x = Flights[i-1].Track[j-1].x;
				p.y = Flights[i-1].Track[j-1].y;
				p.z =Flights[i-1].Track[j-1].z;
				if (airportname == "JFK") {
					bool in_wireframe_13R = inbox(p, JFK13R.REF_leftup,
JFK13R.REF_rightup, JFK13R.THR_leftup, JFK13R.THR_rightup, JFK13R.REF_leftdown,
JFK13R.REF_rightdown, JFK13R.THR_leftdown, JFK13R.THR_rightdown);
					bool in_wireframe_31L = inbox(p, JFK31L.REF_leftup,
JFK31L.REF_rightup, JFK31L.THR_leftup, JFK31L.THR_rightup, JFK31L.REF_leftdown,
JFK31L.REF_rightdown, JFK31L.THR_leftdown, JFK31L.THR_rightdown);
					bool in_wireframe_04L = inbox(p, JFK04L.REF_leftup,
JFK04L.REF_rightup, JFK04L.THR_leftup, JFK04L.THR_rightup, JFK04L.REF_leftdown,
JFK04L.REF_rightdown, JFK04L.THR_leftdown, JFK04L.THR_rightdown);
					bool in_wireframe_22R = inbox(p, JFK22R.REF_leftup,
JFK22R.REF_rightup, JFK22R.THR_leftup, JFK22R.THR_rightup, JFK22R.REF_leftdown,
JFK22R.REF_rightdown, JFK22R.THR_leftdown, JFK22R.THR_rightdown);
					bool in_wireframe_13L = inbox(p, JFK13L.REF_leftup,
JFK13L.REF_rightup, JFK13L.THR_leftup, JFK13L.THR_rightup, JFK13L.REF_leftdown,
JFK13L.REF_rightdown, JFK13L.THR_leftdown, JFK13L.THR_rightdown);
					bool in_wireframe_31R = inbox(p, JFK31R.REF_leftup,
JFK31R.REF_rightup, JFK31R.THR_leftup, JFK31R.THR_rightup, JFK31R.REF_leftdown,
JFK31R.REF_rightdown, JFK31R.THR_leftdown, JFK31R.THR_rightdown);
					bool in_wireframe_04R = inbox(p, JFK04R.REF_leftup,
JFK04R.REF_rightup, JFK04R.THR_leftup, JFK04R.THR_rightup, JFK04R.REF_leftdown,
JFK04R.REF_rightdown, JFK04R.THR_leftdown, JFK04R.THR_rightdown);
					bool in_wireframe_22L = inbox(p, JFK22L.REF_leftup,
JFK22L.REF_rightup, JFK22L.THR_leftup, JFK22L.THR_rightup, JFK22L.REF_leftdown,
JFK22L.REF_rightdown, JFK22L.THR_leftdown, JFK22L.THR_rightdown);
					if (in_wireframe_13R == true) {
```

```
                                    if (JFK13R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = JFK13R;
                                    }
                              }
                              if (in_wireframe_31L == true) {
                                    if (JFK31L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = JFK31L;
                                    }
                              }
                              if (in_wireframe_04L == true) {
                                    if (JFK04L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = JFK04L;
                                    }
                              }
                              if (in_wireframe_22R == true) {
                                    if (JFK22R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
```

```
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = JFK22R;
                                }
                        }
                        if (in_wireframe_13L == true) {
                                if (JFK13L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = JFK13L;
                                }
                        }
                        if (in_wireframe_31R == true) {
                                if (JFK31R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = JFK31R;
                                }
                        }
                        if (in_wireframe_04R == true) {
                                if (JFK04R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
```

```
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = JFK04R;
                                    }
                              }
                              if (in_wireframe_22L == true) {
                                    if (JFK22L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = JFK22L;
                                    }
                              }
                        }
                        if (airportname == "LGA") {
                              bool in_wireframe_13 = inbox(p, LGA13.REF_leftup,
LGA13.REF_rightup, LGA13.THR_leftup, LGA13.THR_rightup, LGA13.REF_leftdown,
LGA13.REF_rightdown, LGA13.THR_leftdown, LGA13.THR_rightdown);
                              bool in_wireframe_31 = inbox(p, LGA31.REF_leftup,
LGA31.REF_rightup, LGA31.THR_leftup, LGA31.THR_rightup, LGA31.REF_leftdown,
LGA31.REF_rightdown, LGA31.THR_leftdown, LGA31.THR_rightdown);
                              bool in_wireframe_22 = inbox(p, LGA22.REF_leftup,
LGA22.REF_rightup, LGA22.THR_leftup, LGA22.THR_rightup, LGA22.REF_leftdown,
LGA22.REF_rightdown, LGA22.THR_leftdown, LGA22.THR_rightdown);
                              bool in_wireframe_04 = inbox(p, LGA04.REF_leftup,
LGA04.REF_rightup, LGA04.THR_leftup, LGA04.THR_rightup, LGA04.REF_leftdown,
LGA04.REF_rightdown, LGA04.THR_leftdown, LGA04.THR_rightdown);
                              if (in_wireframe_13 == true) {
                                    if (LGA13.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = LGA13;
                                    }
                              }
                              if (in_wireframe_31 == true) {
                                    if (LGA31.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
```

```
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = LGA31;
                                }
                        }
                        if (in_wireframe_22 == true) {
                                if (LGA22.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = LGA22;
                                }
                        }
                        if (in_wireframe_04 == true) {
                                if (LGA04.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = LGA04;
                                }
                        }
                }
                if (airportname == "EWR") {
                        bool in_wireframe_11 = inbox(p, EWR11.REF_leftup,
EWR11.REF_rightup, EWR11.THR_leftup, EWR11.THR_rightup, EWR11.REF_leftdown,
EWR11.REF_rightdown, EWR11.THR_leftdown, EWR11.THR_rightdown);
                        bool in_wireframe_29 = inbox(p, EWR29.REF_leftup,
EWR29.REF_rightup, EWR29.THR_leftup, EWR29.THR_rightup, EWR29.REF_leftdown,
EWR29.REF_rightdown, EWR29.THR_leftdown, EWR29.THR_rightdown);
```

```
                        bool in_wireframe_22L = inbox(p, EWR22L.REF_leftup,
EWR22L.REF_rightup, EWR22L.THR_leftup, EWR22L.THR_rightup, EWR22L.REF_leftdown,
EWR22L.REF_rightdown, EWR22L.THR_leftdown, EWR22L.THR_rightdown);
                        bool in_wireframe_04R = inbox(p, EWR04R.REF_leftup,
EWR04R.REF_rightup, EWR04R.THR_leftup, EWR04R.THR_rightup, EWR04R.REF_leftdown,
EWR04R.REF_rightdown, EWR04R.THR_leftdown, EWR04R.THR_rightdown);
                        bool in_wireframe_22R = inbox(p, EWR22R.REF_leftup,
EWR22R.REF_rightup, EWR22R.THR_leftup, EWR22R.THR_rightup, EWR22R.REF_leftdown,
EWR22R.REF_rightdown, EWR22R.THR_leftdown, EWR22R.THR_rightdown);
                        bool in_wireframe_04L = inbox(p, EWR04L.REF_leftup,
EWR04L.REF_rightup, EWR04L.THR_leftup, EWR04L.THR_rightup, EWR04L.REF_leftdown,
EWR04L.REF_rightdown, EWR04L.THR_leftdown, EWR04L.THR_rightdown);
                        if (in_wireframe_11 == true) {
                                if (EWR11.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = EWR11;
                                }
                        }
                        if (in_wireframe_29 == true) {
                                if (EWR29.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = EWR29;
                                }
                        }
                        if (in_wireframe_22L == true) {
                                if (EWR22L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
```

```
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = EWR22L;
                                    }
                              }
                              if (in_wireframe_04R == true) {
                                    if (EWR04R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = EWR04R;
                                    }
                              }
                              if (in_wireframe_22R == true) {
                                    if (EWR22R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = EWR22R;
                                    }
                              }
                              if (in_wireframe_04L == true) {
                                    if (EWR04L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                          dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                          time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                    }
                                    else {
                                          dist_traveled_inside = 0.0;
                                          time_traveled_inside = 0.0;
                                          lastrunway = EWR04L;
                                    }
                              }
```

```
                    }
                    if (airportname == "MDW") {
                           bool in_wireframe_04L = inbox(p, MDW04L.REF_leftup,
MDW04L.REF_rightup, MDW04L.THR_leftup, MDW04L.THR_rightup, MDW04L.REF_leftdown,
MDW04L.REF_rightdown, MDW04L.THR_leftdown, MDW04L.THR_rightdown);
                           bool in_wireframe_22R = inbox(p, MDW22R.REF_leftup,
MDW22R.REF_rightup, MDW22R.THR_leftup, MDW22R.THR_rightup, MDW22R.REF_leftdown,
MDW22R.REF_rightdown, MDW22R.THR_leftdown, MDW22R.THR_rightdown);
                           bool in_wireframe_04R = inbox(p, MDW04R.REF_leftup,
MDW04R.REF_rightup, MDW04R.THR_leftup, MDW04R.THR_rightup, MDW04R.REF_leftdown,
MDW04R.REF_rightdown, MDW04R.THR_leftdown, MDW04R.THR_rightdown);
                           bool in_wireframe_22L = inbox(p, MDW22L.REF_leftup,
MDW22L.REF_rightup, MDW22L.THR_leftup, MDW22L.THR_rightup, MDW22L.REF_leftdown,
MDW22L.REF_rightdown, MDW22L.THR_leftdown, MDW22L.THR_rightdown);
                           bool in_wireframe_13C = inbox(p, MDW13C.REF_leftup,
MDW13C.REF_rightup, MDW13C.THR_leftup, MDW13C.THR_rightup, MDW13C.REF_leftdown,
MDW13C.REF_rightdown, MDW13C.THR_leftdown, MDW13C.THR_rightdown);
                           bool in_wireframe_31C = inbox(p, MDW31C.REF_leftup,
MDW31C.REF_rightup, MDW31C.THR_leftup, MDW31C.THR_rightup, MDW31C.REF_leftdown,
MDW31C.REF_rightdown, MDW31C.THR_leftdown, MDW31C.THR_rightdown);
                           bool in_wireframe_13L = inbox(p, MDW13L.REF_leftup,
MDW13L.REF_rightup, MDW13L.THR_leftup, MDW13L.THR_rightup, MDW13L.REF_leftdown,
MDW13L.REF_rightdown, MDW13L.THR_leftdown, MDW13L.THR_rightdown);
                           bool in_wireframe_31R = inbox(p, MDW31R.REF_leftup,
MDW31R.REF_rightup, MDW31R.THR_leftup, MDW31R.THR_rightup, MDW31R.REF_leftdown,
MDW31R.REF_rightdown, MDW31R.THR_leftdown, MDW31R.THR_rightdown);
                           bool in_wireframe_13R = inbox(p, MDW13R.REF_leftup,
MDW13R.REF_rightup, MDW13R.THR_leftup, MDW13R.THR_rightup, MDW13R.REF_leftdown,
MDW13R.REF_rightdown, MDW13R.THR_leftdown, MDW13R.THR_rightdown);
                           bool in_wireframe_31L = inbox(p, MDW31L.REF_leftup,
MDW31L.REF_rightup, MDW31L.THR_leftup, MDW31L.THR_rightup, MDW31L.REF_leftdown,
MDW31L.REF_rightdown, MDW31L.THR_leftdown, MDW31L.THR_rightdown);
                           if (in_wireframe_04L == true) {
                                  if (MDW04L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                         dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                         time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                  }
                                  else {
                                         dist_traveled_inside = 0.0;
                                         time_traveled_inside = 0.0;
                                         lastrunway = MDW04L;
                                  }
                           }
                           if (in_wireframe_22R == true) {
                                  if (MDW22R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                         dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
```

```
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = MDW22R;
                                }
                        }
                        if (in_wireframe_04R == true) {
                                if (MDW04R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = MDW04R;
                                }
                        }
                        if (in_wireframe_22L == true) {
                                if (MDW22L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = MDW22L;
                                }
                        }
                        if (in_wireframe_13C == true) {
                                if (MDW13C.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                }
                                else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
```

```
                                    lastrunway = MDW13C;
                              }
                        }
                        if (in_wireframe_31C == true) {
                              if (MDW31C.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                    dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                    time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                              }
                              else {
                                    dist_traveled_inside = 0.0;
                                    time_traveled_inside = 0.0;
                                    lastrunway = MDW31C;
                              }
                        }
                        if (in_wireframe_13L == true) {
                              if (MDW13L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                    dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                    time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                              }
                              else {
                                    dist_traveled_inside = 0.0;
                                    time_traveled_inside = 0.0;
                                    lastrunway = MDW13L;
                              }
                        }
                        if (in_wireframe_31R == true) {
                              if (MDW31R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                    dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                    time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                              }
                              else {
                                    dist_traveled_inside = 0.0;
                                    time_traveled_inside = 0.0;
                                    lastrunway = MDW31R;
                              }
                        }
                        if (in_wireframe_13R == true) {
                              if (MDW13R.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
```

```
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                  }
                                  else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = MDW13R;
                                  }
                            }
                            if (in_wireframe_31L == true) {
                                  if (MDW31L.name == lastrunway.name &&
abs(getangle(p.x, p.y, Flights[i-1].Track[j-2].x, Flights[i-1].Track[j-2].y) -
getangle(lastrunway.THR.x, lastrunway.THR.y, lastrunway.FAF.x,
lastrunway.FAF.y)) < angle_diff_threshold && distance_2D(p.x, p.y,
lastrunway.THR.x, lastrunway.THR.y) < dist2THR_threshold) {
                                        dist_traveled_inside +=
distance_2D(Flights[i-1].Track[j-1].x, Flights[i-1].Track[j-1].y, Flights[i-
1].Track[j-2].x, Flights[i-1].Track[j-2].y);
                                        time_traveled_inside += Flights[i-
1].Track[j-1].t - Flights[i-1].Track[j-2].t;
                                  }
                                  else {
                                        dist_traveled_inside = 0.0;
                                        time_traveled_inside = 0.0;
                                        lastrunway = MDW31L;
                                  }
                            }

                      }

                      if (dist_traveled_inside > dist_traveled_inside_threshold
&& time_traveled_inside > time_traveled_inside_threshold) {
                            Flights[i-1].landingrunway = lastrunway;
                            break;
                      }
                }
        }
}
double gettrueairspeed(double heading_ac, double groundspeed_ac, double
heading_wind, double speed_wind) {
        double v_TAS_x, v_TAS_y;
        v_TAS_x = groundspeed_ac * cos(heading_ac) - speed_wind *
sin(heading_wind);
        v_TAS_y = groundspeed_ac * sin(heading_ac) - speed_wind *
sin(heading_wind);
        double v_TAS = sqrt(v_TAS_x*v_TAS_x + v_TAS_y*v_TAS_y);
        return v_TAS;
}
double interpolate(double start, double end, double fraction) {
        //e.g. f(2,3,0.1) = 2.1; f(2,3,-0.1) = 1.9; end must > start
        double output;
        output = start + fraction*(end - start);
        return output;
}
```

```
vector<TRACKPOINT> linear_interpolate(vector<TRACKPOINT> *p_Track_old) {
	//linear interploate
	vector<TRACKPOINT> Track_new;
	if ((*p_Track_old).size() > 1) {
		TRACKPOINT start, end; //keep updating the bin formed by these two
points
		TRACKPOINT temp; //new interpolated point
		//set initial values for start and end
		start = (*p_Track_old).front();
		end = (*p_Track_old)[1];
		temp.t = int(start.t + 0.5); //rounded to closest integer for
first point
		double r = (temp.t - start.t)/(end.t - start.t); // can be
negative if temp.t is rounded to an integer smaller than start.t
		temp.x = interpolate(start.x, end.x, r);
		temp.y = interpolate(start.y, end.y, r);
		temp.z = interpolate(start.z, end.z, r);
		XY2LatLong(regionID, temp.x, temp.y, temp.lat, temp.lng);
		temp.groundspeed = 0.0;
		temp.verticalspeed = 0.0;
		temp.heading = 0.0;
		Track_new.push_back(temp);

		int i = 1;//for first point: bin is formed between track[i-1] and
track[i+1-1]
		bool endreached = false;
		while (temp.t <= (*p_Track_old).back().t) {
			temp.t = temp.t + TIMESTEP; // the last interpolated point
could not be greater than (Track_old.back().t + TIMESTEP)
			// update bin (moving forward) as long as temp.t > end.t
			while (temp.t > end.t && endreached == false) {
				i = i + 1;
				if (i <= (*p_Track_old).size() - 1) {
					start = (*p_Track_old)[i-1];
					end = (*p_Track_old)[(i+1)-1];
				}
				else {//if the last point is beyond the end of track,
use last bin to exterpolate (r > 1)
					start = (*p_Track_old)[((*p_Track_old).size()-
1)-1];
					end = (*p_Track_old).back();
					endreached = true;
				}
			}
			r = (temp.t - start.t)/(end.t - start.t);
			temp.x = interpolate(start.x, end.x, r);
			temp.y = interpolate(start.y, end.y, r);
			temp.z = interpolate(start.z, end.z, r);
			XY2LatLong(regionID, temp.x, temp.y, temp.lat, temp.lng);
			temp.groundspeed = 0.0;
			temp.verticalspeed = 0.0;
			temp.heading = 0.0;
			Track_new.push_back(temp);
		}
	}
	else {
		Track_new = *p_Track_old;
	}
	return Track_new;
}
```

```
vector<TRACKPOINT> smooth_track(vector<TRACKPOINT> *p_track) {
	//hardcoded smoothbin as 10s; require that the input track has been
interpolated with even time intervals between consecutive points
	vector<TRACKPOINT> smoothed_track;
	int n = (*p_track).size();
	if (n >= 11) {
		smoothed_track.push_back((*p_track).front()); //first point same
		TRACKPOINT tp1;
		tp1.x = ((*p_track)[0].x + (*p_track)[1].x + (*p_track)[2].x)/3.0;
		tp1.y = ((*p_track)[0].y + (*p_track)[1].y + (*p_track)[2].y)/3.0;
		tp1.z = ((*p_track)[0].z + (*p_track)[1].z + (*p_track)[2].z)/3.0;
		XY2LatLong(regionID, tp1.x, tp1.y, tp1.lat, tp1.lng);
		tp1.t = (*p_track)[1].t;
		smoothed_track.push_back(tp1);
		TRACKPOINT tp2;
		tp2.x = ((*p_track)[0].x + (*p_track)[1].x + (*p_track)[2].x +
(*p_track)[3].x + (*p_track)[4].x)/5.0;
		tp2.y = ((*p_track)[0].y + (*p_track)[1].y + (*p_track)[2].y +
(*p_track)[3].y + (*p_track)[4].y)/5.0;
		tp2.z = ((*p_track)[0].z + (*p_track)[1].z + (*p_track)[2].z +
(*p_track)[3].z + (*p_track)[4].z)/5.0;
		XY2LatLong(regionID, tp2.x, tp2.y, tp2.lat, tp2.lng);
		tp2.t = (*p_track)[2].t;
		smoothed_track.push_back(tp2);
		TRACKPOINT tp3;
		tp3.x = ((*p_track)[0].x + (*p_track)[1].x + (*p_track)[2].x +
(*p_track)[3].x + (*p_track)[4].x + (*p_track)[5].x + (*p_track)[6].x)/7.0;
		tp3.y = ((*p_track)[0].y + (*p_track)[1].y + (*p_track)[2].y +
(*p_track)[3].y + (*p_track)[4].y + (*p_track)[5].y + (*p_track)[6].y)/7.0;
		tp3.z = ((*p_track)[0].z + (*p_track)[1].z + (*p_track)[2].z +
(*p_track)[3].z + (*p_track)[4].z + (*p_track)[5].z + (*p_track)[6].z)/7.0;
		XY2LatLong(regionID, tp3.x, tp3.y, tp3.lat, tp3.lng);
		tp3.t = (*p_track)[3].t;
		smoothed_track.push_back(tp3);
		TRACKPOINT tp4;
		tp4.x = ((*p_track)[0].x + (*p_track)[1].x + (*p_track)[2].x +
(*p_track)[3].x + (*p_track)[4].x + (*p_track)[5].x + (*p_track)[6].x +
(*p_track)[7].x + (*p_track)[8].x)/9.0;
		tp4.y = ((*p_track)[0].y + (*p_track)[1].y + (*p_track)[2].y +
(*p_track)[3].y + (*p_track)[4].y + (*p_track)[5].y + (*p_track)[6].y +
(*p_track)[7].y + (*p_track)[8].y)/9.0;
		tp4.z = ((*p_track)[0].z + (*p_track)[1].z + (*p_track)[2].z +
(*p_track)[3].z + (*p_track)[4].z + (*p_track)[5].z + (*p_track)[6].z +
(*p_track)[7].z + (*p_track)[8].z)/9.0;
		XY2LatLong(regionID, tp4.x, tp4.y, tp4.lat, tp4.lng);
		tp4.t = (*p_track)[4].t;
		smoothed_track.push_back(tp4);
		int k = 5;
		while (smoothed_track.size() < n-5) {
			TRACKPOINT tp;
			tp.x = ((*p_track)[k-5].x + (*p_track)[k-4].x +
(*p_track)[k-3].x + (*p_track)[k-2].x + (*p_track)[k-1].x + (*p_track)[k].x +
(*p_track)[k+1].x + (*p_track)[k+2].x + (*p_track)[k+3].x + (*p_track)[k+4].x +
(*p_track)[k+5].x)/11.0;
			tp.y = ((*p_track)[k-5].y + (*p_track)[k-4].y +
(*p_track)[k-3].y + (*p_track)[k-2].y + (*p_track)[k-1].y + (*p_track)[k].y +
(*p_track)[k+1].y + (*p_track)[k+2].y + (*p_track)[k+3].y + (*p_track)[k+4].y +
(*p_track)[k+5].y)/11.0;
			tp.z = ((*p_track)[k-5].z + (*p_track)[k-4].z +
(*p_track)[k-3].z + (*p_track)[k-2].z + (*p_track)[k-1].z + (*p_track)[k].z +
```

```
(*p_track)[k+1].z + (*p_track)[k+2].z + (*p_track)[k+3].z + (*p_track)[k+4].z +
(*p_track)[k+5].z)/11.0;
                    XY2LatLong(regionID, tp.x, tp.y, tp.lat, tp.lng);
                    tp.t = (*p_track)[k].t;
                    smoothed_track.push_back(tp);
                    k++;
             }
             TRACKPOINT tpn_4;
             tpn_4.x = ((*p_track)[k-4].x + (*p_track)[k-3].x + (*p_track)[k-
2].x + (*p_track)[k-1].x + (*p_track)[k].x + (*p_track)[k+1].x +
(*p_track)[k+2].x + (*p_track)[k+3].x + (*p_track)[k+4].x)/9.0;
             tpn_4.y = ((*p_track)[k-4].y + (*p_track)[k-3].y + (*p_track)[k-
2].y + (*p_track)[k-1].y + (*p_track)[k].y + (*p_track)[k+1].y +
(*p_track)[k+2].y + (*p_track)[k+3].y + (*p_track)[k+4].y)/9.0;
             tpn_4.z = ((*p_track)[k-4].z + (*p_track)[k-3].z + (*p_track)[k-
2].z + (*p_track)[k-1].z + (*p_track)[k].z + (*p_track)[k+1].z +
(*p_track)[k+2].z + (*p_track)[k+3].z + (*p_track)[k+4].z)/9.0;
             XY2LatLong(regionID, tpn_4.x, tpn_4.y, tpn_4.lat, tpn_4.lng);
             tpn_4.t = (*p_track)[k].t;
             smoothed_track.push_back(tpn_4);
             k++;
             TRACKPOINT tpn_3;
             tpn_3.x = ((*p_track)[k-3].x + (*p_track)[k-2].x + (*p_track)[k-
1].x + (*p_track)[k].x + (*p_track)[k+1].x + (*p_track)[k+2].x +
(*p_track)[k+3].x)/7.0;
             tpn_3.y = ((*p_track)[k-3].y + (*p_track)[k-2].y + (*p_track)[k-
1].y + (*p_track)[k].y + (*p_track)[k+1].y + (*p_track)[k+2].y +
(*p_track)[k+3].y)/7.0;
             tpn_3.z = ((*p_track)[k-3].z + (*p_track)[k-2].z + (*p_track)[k-
1].z + (*p_track)[k].z + (*p_track)[k+1].z + (*p_track)[k+2].z +
(*p_track)[k+3].z)/7.0;
             XY2LatLong(regionID, tpn_3.x, tpn_3.y, tpn_3.lat, tpn_3.lng);
             tpn_3.t = (*p_track)[k].t;
             smoothed_track.push_back(tpn_3);
             k++;
             TRACKPOINT tpn_2;
             tpn_2.x = ((*p_track)[k-2].x + (*p_track)[k-1].x + (*p_track)[k].x
+ (*p_track)[k+1].x + (*p_track)[k+2].x)/5.0;
             tpn_2.y = ((*p_track)[k-2].y + (*p_track)[k-1].y + (*p_track)[k].y
+ (*p_track)[k+1].y + (*p_track)[k+2].y)/5.0;
             tpn_2.z = ((*p_track)[k-2].z + (*p_track)[k-1].z + (*p_track)[k].z
+ (*p_track)[k+1].z + (*p_track)[k+2].z)/5.0;
             XY2LatLong(regionID, tpn_2.x, tpn_2.y, tpn_2.lat, tpn_2.lng);
             tpn_2.t = (*p_track)[k].t;
             smoothed_track.push_back(tpn_2);
             k++;
             TRACKPOINT tpn_1;
             tpn_1.x = ((*p_track)[k-1].x + (*p_track)[k].x +
(*p_track)[k+1].x)/3.0;
             tpn_1.y = ((*p_track)[k-1].y + (*p_track)[k].y +
(*p_track)[k+1].y)/3.0;
             tpn_1.z = ((*p_track)[k-1].z + (*p_track)[k].z +
(*p_track)[k+1].z)/3.0;
             XY2LatLong(regionID, tpn_1.x, tpn_1.y, tpn_1.lat, tpn_1.lng);
             tpn_1.t = (*p_track)[k].t;
             smoothed_track.push_back(tpn_1);
             smoothed_track.push_back((*p_track).back());
       }
       else smoothed_track = *p_track;
       return smoothed_track;
```

```
}
void derived_track_data(vector<TRACKPOINT> *p_Track_new, string currentdate) {
	//derive groundspeed, airspeed, vertical speed and heading
	(*p_Track_new).front().groundspeed = distance_2D((*p_Track_new)[0].x,
(*p_Track_new)[0].y, (*p_Track_new)[1].x, (*p_Track_new)[1].y)/TIMESTEP;
	(*p_Track_new).front().verticalspeed = ((*p_Track_new)[0].z -
(*p_Track_new)[1].z)/TIMESTEP;
	(*p_Track_new).front().heading =
angle2heading(getangle((*p_Track_new)[0].x, (*p_Track_new)[0].y,
(*p_Track_new)[1].x, (*p_Track_new)[1].y));
	(*p_Track_new).front().airspeed =
gettrueairspeed((*p_Track_new).front().heading,
(*p_Track_new).front().groundspeed, get_current_wind(currentdate,
(*p_Track_new).front().t).heading, get_current_wind(currentdate,
(*p_Track_new).front().t).speed);
	for (int i = 2; i <= (*p_Track_new).size()-1; ++i) {
		(*p_Track_new)[i-1].groundspeed =
0.5*((distance_2D((*p_Track_new)[(i+1)-1], (*p_Track_new)[i-1])/TIMESTEP) +
(distance_2D((*p_Track_new)[(i-1)-1], (*p_Track_new)[i-1])/TIMESTEP));
		(*p_Track_new)[i-1].airspeed = gettrueairspeed((*p_Track_new)[i-
1].heading, (*p_Track_new)[i-1].groundspeed, get_current_wind(currentdate,
(*p_Track_new)[i-1].t).heading, get_current_wind(currentdate, (*p_Track_new)[i-
1].t).speed);
		(*p_Track_new)[i-1].verticalspeed = 0.5*(((*p_Track_new)[(i+1)-
1].z - (*p_Track_new)[i-1].z)/TIMESTEP + ((*p_Track_new)[i-1].z -
(*p_Track_new)[(i-1)-1].z)/TIMESTEP);
		//get average heading for each point
		double angle1 = getangle((*p_Track_new)[(i-1)-1].x,
(*p_Track_new)[(i-1)-1].y, (*p_Track_new)[i-1].x, (*p_Track_new)[i-1].y);
		double angle2 = getangle((*p_Track_new)[i-1].x, (*p_Track_new)[i-
1].y, (*p_Track_new)[(i+1)-1].x, (*p_Track_new)[(i+1)-1].y);
		double heading1 = angle2heading(angle1);
		double heading2 = angle2heading(angle2);
		bool condition1 = heading1 >= 0 && heading1 < PI && heading2 >= 0
&& heading2 < heading1 + PI;
		bool condition2 = heading1 >= PI && heading1 < 2*PI && heading2 >=
heading1 - PI && heading2 < 2*PI;
		bool condition3 = heading1 >= 0 && heading1 < 0.5*PI && heading2
>= heading1 + PI && heading2 < 2*PI - heading1;
		bool condition4 = heading1 >= PI && heading1 < 1.5*PI && heading2
>= 0 && heading2 < heading1 - PI;
		bool condition5 = heading1 >= 1.5*PI && heading1 < 2*PI &&
heading2 >= 0 && heading2 < 2*PI - heading1;
		bool condition6 = heading1 >= 0 && heading1 < 0.5*PI && heading2
>= 2*PI - heading1 && heading2 < 2*PI;
		bool condition7 = heading1 >= 0.5*PI && heading1 < PI && heading2
>= heading1 + PI && heading2 < 2*PI;
		bool condition8 = heading1 >= 1.5*PI && heading1 < 2*PI &&
heading2 >= 2*PI - heading1 && heading2 < heading1 - PI;
		if (condition1 == true || condition2 == true)
			(*p_Track_new)[i-1].heading = 0.5*(heading1 + heading2);
		else if (condition3 == true || condition4 == true || condition5 ==
true)
			(*p_Track_new)[i-1].heading = PI + 0.5*(heading1 +
heading2);
		else if (condition6 == true || condition7 == true || condition8 ==
true)
			(*p_Track_new)[i-1].heading = -PI + 0.5*(heading1 +
heading2);
	}
```

```
	(*p_Track_new).back().groundspeed =
distance_2D((*p_Track_new)[(*p_Track_new).size()-1].x,
(*p_Track_new)[(*p_Track_new).size()-1].y,
(*p_Track_new)[(*p_Track_new).size()-1-1].x,
(*p_Track_new)[(*p_Track_new).size()-1-1].y)/TIMESTEP;
	(*p_Track_new).back().verticalspeed =
((*p_Track_new)[(*p_Track_new).size()-1].z -
(*p_Track_new)[(*p_Track_new).size()-1-1].z)/TIMESTEP;
	(*p_Track_new).back().heading =
angle2heading(getangle((*p_Track_new)[(*p_Track_new).size()-1-1].x,
(*p_Track_new)[(*p_Track_new).size()-1-1].y,
(*p_Track_new)[(*p_Track_new).size()-1].x,
(*p_Track_new)[(*p_Track_new).size()-1].y));
	(*p_Track_new).back().airspeed =
gettrueairspeed((*p_Track_new).back().heading,
(*p_Track_new).back().groundspeed, get_current_wind(currentdate,
(*p_Track_new).back().t).heading, get_current_wind(currentdate,
(*p_Track_new).back().t).speed);
}
double colonformattime2double(string colonformat) {
	//convert format "00:00:01.346" to (double) seconds after midnight
	vector<string> cluster(3);
	split(colonformat, ":", cluster, 0);
	double hr = atof(cluster[0].c_str());
	double min = atof(cluster[1].c_str());
	double sec =atof(cluster[2].c_str());
	double timesincemidnight = hr*3600.0 + min*60.0 + sec;
	return timesincemidnight;
}
void preprocessPDARStracks(string oldtrackfile, string flightfile, string
newtrackfile) {
	//convert original PDARS track file to preprocessed format
	ifstream in_flight(flightfile);
	vector<int> FlightIDs;
	vector<string> cluster0(1);
	char chars0[200];
	in_flight.getline(chars0, 200);
	while (in_flight.getline(chars0,200)) {
		split(chars0, ",", cluster0, 0);
		FlightIDs.push_back(atol(cluster0[0].c_str()));
	}
	in_flight.close();

	ifstream in_track(oldtrackfile);
	vector<int> c1;//flight id
	vector<double> c2;
	vector<double> c3;//lat
	vector<double> c4;//lng
	vector<double> c5;//alt converted from original unit (e.g. 100*ft.)
	vector<string> cluster(5);//there can be more colomns in original file
	char chars[200];
	in_track.getline(chars, 200);
	int i = 1; //index for elements in vector FlightIDs
	cout<<"preprocessing PDARS track data for flight " << i <<"\n";
	bool flightfound = false;
	while (in_track.getline(chars,200)) {
		split(chars, ",", cluster, 0);
		if (flightfound == true && atol(cluster[0].c_str()) !=
FlightIDs[i-1]) {
			if (i <= FlightIDs.size()-1) {
```

```
                        i = i + 1; // update to next flight
                        cout<<"preprocessing PDARS track data for flight " <<
i <<"\n";
                  }
                  else {break;}
                  flightfound = false;
            }
            if (flightfound == false && atol(cluster[0].c_str()) !=
FlightIDs[i-1]) {
                  continue;
            }
            flightfound = true;
            //if same flightID found as in flights file
            c1.push_back(atol(cluster[0].c_str()));
            c2.push_back(colonformattime2double(cluster[1].c_str())); //PDARS
time format: 00:00:01.346, need to split using ":" to delimit, and convert to
single value by the function
            c3.push_back(atof(cluster[2].c_str()));
            c4.push_back(atof(cluster[3].c_str()));
            c5.push_back(atof(cluster[4].c_str())*100*ft2m);
      }
      in_track.close();
      ofstream out(newtrackfile);
      out<<"flight_ID,time,lat,lng,alt(m)"<<"\n";
      for (int i = 1; i <= c1.size(); ++i) {
            out<<c1[i-1]<<","<<c2[i-1]<<","<<c3[i-1]<<","<<c4[i-1]<<","<<c5[i-
1]<<"\n";
      }
      out.close();
}
void preprocessPDARStracksforall() {
      //for currently six days NYC PDARS data
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080205_Tracks.csv",
"../Data/DATA _ Preprocessed/EWR_PDARS/[input] flights EWR20080205
arrival.csv", "../Data/DATA _ Preprocessed/EWR_PDARS/[input] tracks EWR20080205
arrival.csv");
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080205_Tracks.csv",
"../Data/DATA _ Preprocessed/JFK_PDARS/[input] flights JFK20080205
arrival.csv", "../Data/DATA _ Preprocessed/JFK_PDARS/[input] tracks JFK20080205
arrival.csv");
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080205_Tracks.csv",
"../Data/DATA _ Preprocessed/LGA_PDARS/[input] flights LGA20080205
arrival.csv", "../Data/DATA _ Preprocessed/LGA_PDARS/[input] tracks LGA20080205
arrival.csv");
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080319_Tracks.csv",
"../Data/DATA _ Preprocessed/EWR_PDARS/[input] flights EWR20080319
arrival.csv", "../Data/DATA _ Preprocessed/EWR_PDARS/[input] tracks EWR20080319
arrival.csv");
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080319_Tracks.csv",
"../Data/DATA _ Preprocessed/JFK_PDARS/[input] flights JFK20080319
arrival.csv", "../Data/DATA _ Preprocessed/JFK_PDARS/[input] tracks JFK20080319
arrival.csv");
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080319_Tracks.csv",
"../Data/DATA _ Preprocessed/LGA_PDARS/[input] flights LGA20080319
arrival.csv", "../Data/DATA _ Preprocessed/LGA_PDARS/[input] tracks LGA20080319
arrival.csv");
      preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080331_Tracks.csv",
"../Data/DATA _ Preprocessed/EWR_PDARS/[input] flights EWR20080331
arrival.csv", "../Data/DATA _ Preprocessed/EWR_PDARS/[input] tracks EWR20080331
arrival.csv");
```

```
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080331_Tracks.csv",
"../Data/DATA _ Preprocessed/JFK_PDARS/[input] flights JFK20080331
arrival.csv", "../Data/DATA _ Preprocessed/JFK_PDARS/[input] tracks JFK20080331
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080331_Tracks.csv",
"../Data/DATA _ Preprocessed/LGA_PDARS/[input] flights LGA20080331
arrival.csv", "../Data/DATA _ Preprocessed/LGA_PDARS/[input] tracks LGA20080331
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080410_Tracks.csv",
"../Data/DATA _ Preprocessed/EWR_PDARS/[input] flights EWR20080410
arrival.csv", "../Data/DATA _ Preprocessed/EWR_PDARS/[input] tracks EWR20080410
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080410_Tracks.csv",
"../Data/DATA _ Preprocessed/JFK_PDARS/[input] flights JFK20080410
arrival.csv", "../Data/DATA _ Preprocessed/JFK_PDARS/[input] tracks JFK20080410
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080410_Tracks.csv",
"../Data/DATA _ Preprocessed/LGA_PDARS/[input] flights LGA20080410
arrival.csv", "../Data/DATA _ Preprocessed/LGA_PDARS/[input] tracks LGA20080410
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080623_Tracks.csv",
"../Data/DATA _ Preprocessed/EWR_PDARS/[input] flights EWR20080623
arrival.csv", "../Data/DATA _ Preprocessed/EWR_PDARS/[input] tracks EWR20080623
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080623_Tracks.csv",
"../Data/DATA _ Preprocessed/JFK_PDARS/[input] flights JFK20080623
arrival.csv", "../Data/DATA _ Preprocessed/JFK_PDARS/[input] tracks JFK20080623
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080623_Tracks.csv",
"../Data/DATA _ Preprocessed/LGA_PDARS/[input] flights LGA20080623
arrival.csv", "../Data/DATA _ Preprocessed/LGA_PDARS/[input] tracks LGA20080623
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080723_Tracks.csv",
"../Data/DATA _ Preprocessed/EWR_PDARS/[input] flights EWR20080723
arrival.csv", "../Data/DATA _ Preprocessed/EWR_PDARS/[input] tracks EWR20080723
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080723_Tracks.csv",
"../Data/DATA _ Preprocessed/JFK_PDARS/[input] flights JFK20080723
arrival.csv", "../Data/DATA _ Preprocessed/JFK_PDARS/[input] tracks JFK20080723
arrival.csv");
	preprocessPDARStracks("../Data/DATA _ PDARS/NYC_20080723_Tracks.csv",
"../Data/DATA _ Preprocessed/LGA_PDARS/[input] flights LGA20080723
arrival.csv", "../Data/DATA _ Preprocessed/LGA_PDARS/[input] tracks LGA20080723
arrival.csv");
}
void find_trackpoints_at_key_locations(FLIGHT &flight) {
	//all for first approach if multiple approaches exist
	double distto9nm = BIG_DIST_default;
	double distto8nm = BIG_DIST_default;
	double distto7nm = BIG_DIST_default;
	double distto6nm = BIG_DIST_default;
	double distto5nm = BIG_DIST_default;
	double distto4nm = BIG_DIST_default;
	double distto3nm = BIG_DIST_default;
	double distto2dot25nm = BIG_DIST_default;
	double distto1dot5nm = BIG_DIST_default;
	double disttoTHR = BIG_DIST_default;
	for (int j = 1; j <= flight.Track.size(); ++j) { //search forward
		double currentdist_3nm = distance_3D(flight.Track[j-1],
flight.landingrunway.location_3nm);
```

```
            if (currentdist_3nm < distto3nm &&
abs(flight.Track[flight.j_firstentrance-1].t - flight.Track[j-1].t) <
TIMELIMIT) {
                  distto3nm = currentdist_3nm;
                  flight.j_3nm = j;
            }
            double currentdist_2dot25nm = distance_3D(flight.Track[j-1],
flight.landingrunway.location_2dot25nm);
            if (currentdist_2dot25nm < distto2dot25nm &&
abs(flight.Track[flight.j_firstentrance-1].t - flight.Track[j-1].t) <
TIMELIMIT) {
                  distto2dot25nm = currentdist_2dot25nm;
                  flight.j_2dot25nm = j;
            }

            double currentdist2_1dot5nm = distance_3D(flight.Track[j-1],
flight.landingrunway.location_1dot5nm);
            if (currentdist2_1dot5nm < distto1dot5nm &&
abs(flight.Track[flight.j_firstentrance-1].t - flight.Track[j-1].t) <
TIMELIMIT) {
                  distto1dot5nm = currentdist2_1dot5nm;
                  flight.j_1dot5nm = j;
            }
            //j_THR, the closest point to runway threshold
            double currentdist2THR = distance_3D(flight.Track[j-1],
flight.landingrunway.THR);
            if (currentdist2THR < disttoTHR &&
abs(flight.Track[flight.j_firstentrance-1].t - flight.Track[j-1].t) <
TIMELIMIT) {
                  disttoTHR = currentdist2THR;
                  flight.j_THR = j;
            }
            //points with specific distances to runway threshold
            if (j + 1 <= flight.Track.size()) {
                  if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 10.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 10.0*nm2m) {
                        flight.j_10nm_r = j;
                  }
                  if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 9.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 9.0*nm2m) {
                        flight.j_9nm_r = j;
                  }
                  if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 8.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 8.0*nm2m) {
                        flight.j_8nm_r = j;
                  }
                  if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 7.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 7.0*nm2m) {
                        flight.j_7nm_r = j;
                  }
                  if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 6.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 6.0*nm2m) {
                        flight.j_6nm_r = j;
                  }
```

```
                    if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 5.5*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 5.5*nm2m) {
                           flight.j_5dot5nm_r = j;
                    }
                    if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 5.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 5.0*nm2m) {
                           flight.j_5nm_r = j;
                    }
                    if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 4.5*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 4.5*nm2m) {
                           flight.j_4dot5nm_r = j;
                    }
                    if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 4.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 4.0*nm2m) {
                           flight.j_4nm_r = j;
                    }
                    if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 3.5*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 3.5*nm2m) {
                           flight.j_3dot5nm_r = j;
                    }
                    if (distance_2D(flight.Track[j-1],
flight.landingrunway.THR) > 3.0*nm2m && distance_2D(flight.Track[(j+1)-1],
flight.landingrunway.THR) <= 3.0*nm2m) {
                           flight.j_3nm_r = j;
                    }
             }
       }
}
double get_avg_groundspeed(int avgbin, FLIGHT Flight, int j_location, bool
projection) {
       //get time-averaged groundspeed value
       double avg_groundspeed_location = 0.0;
       if (projection == true) {
             for (int jj = 0; jj <= avgbin; ++jj) {
                    double angle_pt; //angle of a track point
                    double angle_delta; //angle difference
                    if (j_location-0.5*avgbin+jj > Flight.Track.size()) {
                           //e.g. if not lined up well, the point closest to 3nm
is close to point at threshold, and beyond boundary
                           angle_delta = Flight.landingrunway.angle -
heading2angle(Flight.Track[Flight.Track.size()-1].heading);
                           avg_groundspeed_location +=
Flight.Track[Flight.Track.size()-1].groundspeed * cos(angle_delta);
                    }
                    else {
                           angle_pt = heading2angle(Flight.Track[(j_location-
0.5*avgbin+jj)-1].heading);
                           angle_delta = Flight.landingrunway.angle - angle_pt;
                           avg_groundspeed_location += Flight.Track[(j_location-
0.5*avgbin+jj)-1].groundspeed * cos(angle_delta);
                    }
             }
       }
       else {
             for (int jj = 0; jj <= avgbin; ++jj) {
```

```
                    if (j_location-0.5*avgbin+jj > Flight.Track.size()) {
                           avg_groundspeed_location +=
Flight.Track[Flight.Track.size()-1].groundspeed;
                    }
                    else {
                           avg_groundspeed_location += Flight.Track[(j_location-
0.5*avgbin+jj)-1].groundspeed;
                    }
             }
       }
       avg_groundspeed_location = avg_groundspeed_location/(double)(avgbin+1);
       return avg_groundspeed_location;
}
double get_avg_airspeed(int avgbin, FLIGHT Flight, int j_location, bool
projection) {
       double avg_airspeed_location = 0.0;
       if (projection == true) {
             for (int jj = 0; jj <= avgbin; ++jj) {
                    double angle_pt; //angle of a track point
                    double angle_delta; //angle difference
                    if (j_location-0.5*avgbin+jj > Flight.Track.size()) {
                           //e.g. if not lined up well, the point closest to 3nm
is close to point at threshold, and beyond boundary
                           angle_delta = Flight.landingrunway.angle -
heading2angle(Flight.Track[Flight.Track.size()-1].heading);
                           avg_airspeed_location +=
Flight.Track[Flight.Track.size()-1].airspeed * cos(angle_delta);
                    }
                    else {
                           angle_pt = heading2angle(Flight.Track[(j_location-
0.5*avgbin+jj)-1].heading);
                           angle_delta = Flight.landingrunway.angle - angle_pt;
                           avg_airspeed_location += Flight.Track[(j_location-
0.5*avgbin+jj)-1].airspeed * cos(angle_delta);
                    }
             }
       }
       else {
             for (int jj = 0; jj <= avgbin; ++jj) {
                    if (j_location-0.5*avgbin+jj > Flight.Track.size()) {
                           //e.g. if not lined up well, the point closest to 3nm
is close to point at threshold, and beyond boundary
                           avg_airspeed_location +=
Flight.Track[Flight.Track.size()-1].airspeed;
                    }
                    else {
                           avg_airspeed_location += Flight.Track[(j_location-
0.5*avgbin+jj)-1].airspeed;
                    }
             }
       }
       avg_airspeed_location = avg_airspeed_location/(double)(avgbin+1);
       return avg_airspeed_location;
}
double get_avg_groundspeed_THR(int avgbin, FLIGHT Flight, bool projection) {
       double avg_groundspeed_THR = 0.0; //initialization
       for (int jj = 0; jj <= avgbin; ++jj) {
             if (projection == true) {
```

```
                    double angle_pt = heading2angle(Flight.Track[(Flight.j_THR-
jj)-1].heading); //angle of previous point (in a direction toward landing
runway)
                    double angle_delta = Flight.landingrunway.angle - angle_pt;
//angle difference

                    avg_groundspeed_THR += Flight.Track[(Flight.j_THR-jj)-
1].groundspeed * cos(angle_delta);
             }
             else
                    avg_groundspeed_THR += Flight.Track[(Flight.j_THR-jj)-
1].groundspeed;
       }
       avg_groundspeed_THR = avg_groundspeed_THR/(double)(avgbin+1);
       return avg_groundspeed_THR;
}
double get_avg_airspeed_THR(int avgbin, FLIGHT Flight, bool projection) {
       double avg_airspeed_THR = 0.0; //initialization
       for (int jj = 0; jj <= avgbin; ++jj) {
             if (projection == true) {
                    double angle_pt = heading2angle(Flight.Track[(Flight.j_THR-
jj)-1].heading); //angle of previous point (in a direction toward landing
runway)
                    double angle_delta = Flight.landingrunway.angle - angle_pt;
//angle difference
                    avg_airspeed_THR += Flight.Track[(Flight.j_THR-jj)-
1].airspeed * cos(angle_delta);
             }
             else
                    avg_airspeed_THR += Flight.Track[(Flight.j_THR-jj)-
1].airspeed;
       }
       avg_airspeed_THR = avg_airspeed_THR/(double)(avgbin+1);
       return avg_airspeed_THR;
}
double get_avg_rateofdescent(int avgbin, FLIGHT Flight, int j_location) {
       double avg_rateofdescent = 0.0;
       for (int jj = 0; jj <= avgbin; ++jj) {
             if (j_location-0.5*avgbin+jj > Flight.Track.size()) {
                    avg_rateofdescent += Flight.Track[Flight.Track.size()-
1].verticalspeed; //if out of range, use last track point's vertical speed
             }
             else avg_rateofdescent += Flight.Track[(j_location-0.5*avgbin+jj)-
1].verticalspeed;
       }
       avg_rateofdescent = avg_rateofdescent/(double)(avgbin+1);
       return avg_rateofdescent; //negative for descent
}
void find_j_altitude(FLIGHT &Flight, int &j_altitude, double altitude, int
j_location_for_approx) {
       //the "altitude" in j index and in function input "altitude" are all
above ground level
       j_altitude = j_default;
       for (int j = Flight.Track.size() - 1; j >= 1; j = j - 1) {  // search
backward
             if (Flight.Track[(j+1)-1].z - Flight.landingrunway.elevation <=
altitude
                    && Flight.Track[j-1].z - Flight.landingrunway.elevation >
altitude
```

```
                    && Flight.Track[j-1].t -
Flight.Track[Flight.j_firstentrance-1].t < TIMELIMIT) { // for first approach
                    j_altitude = j + 1;  // the first one that breaks the limit
                    break;
              }
       }
       if (j_altitude == j_default)
              j_altitude = j_location_for_approx; //substitute
}
bool determine_ROD_event(FLIGHT flight, int j_starting) {
       bool ROD_event = false;
       int count_abnormal_ROD = 0;
       double last_ROD = 0.0;
       double current_ROD = 0.0;
       for (int j = j_starting; j <= flight.Track.size() - 0.5*AVGBIN; ++j) {
              if (abs(flight.Track[j-1].t - flight.Track[flight.j_firstentrance-
1].t) < TIMELIMIT) {
                    current_ROD = get_avg_rateofdescent(AVGBIN, flight, j);
                    if (current_ROD < ROD_event_threshold && last_ROD >=
ROD_event_threshold) {
                           count_abnormal_ROD = 1;
                    }
                    if (current_ROD < ROD_event_threshold && last_ROD <
ROD_event_threshold) {
                           count_abnormal_ROD++;
                    }
                    last_ROD = current_ROD;
                    if (count_abnormal_ROD * TIMESTEP >=
time_period_for_ROD_event) {
                           ROD_event = true;
                           break;
                    }
              }
       }
       return ROD_event;
}
void identify_ApproachProcedure(FLIGHT &flight) {
       Point p1_ILS, p2_ILS, p3_ILS, p4_ILS, p5_ILS, p6_ILS, p7_ILS, p8_ILS;
       p1_ILS.lat = 41.832424; p1_ILS.lng = -87.892060; p1_ILS.z = 3000*ft2m;
LatLong2XY(regionID, p1_ILS.lat, p1_ILS.lng, p1_ILS.x, p1_ILS.y);
       p2_ILS.lat = 41.835020; p2_ILS.lng = -87.887140; p2_ILS.z = 3000*ft2m;
LatLong2XY(regionID, p2_ILS.lat, p2_ILS.lng, p2_ILS.x, p2_ILS.y);
       p3_ILS.lat = 41.832424; p3_ILS.lng = -87.892060; p3_ILS.z = 0*ft2m;
LatLong2XY(regionID, p3_ILS.lat, p3_ILS.lng, p3_ILS.x, p3_ILS.y);
       p4_ILS.lat = 41.835020; p4_ILS.lng = -87.887140; p4_ILS.z = 0*ft2m;
LatLong2XY(regionID, p4_ILS.lat, p4_ILS.lng, p4_ILS.x, p4_ILS.y);
       p5_ILS.lat = 41.901889; p5_ILS.lng = -87.999773; p5_ILS.z = 3000*ft2m;
LatLong2XY(regionID, p5_ILS.lat, p5_ILS.lng, p5_ILS.x, p5_ILS.y);
       p6_ILS.lat = 41.909217; p6_ILS.lng = -87.988322; p6_ILS.z = 3000*ft2m;
LatLong2XY(regionID, p6_ILS.lat, p6_ILS.lng, p6_ILS.x, p6_ILS.y);
       p7_ILS.lat = 41.901889; p7_ILS.lng = -87.999773; p7_ILS.z = 0*ft2m;
LatLong2XY(regionID, p7_ILS.lat, p7_ILS.lng, p7_ILS.x, p7_ILS.y);
       p8_ILS.lat = 41.909217; p8_ILS.lng = -87.988322; p8_ILS.z = 0*ft2m;
LatLong2XY(regionID, p8_ILS.lat, p8_ILS.lng, p8_ILS.x, p8_ILS.y);
       Point p1_VFR, p2_VFR, p3_VFR, p4_VFR, p5_VFR, p6_VFR, p7_VFR, p8_VFR;
       p1_VFR.lat = 41.741682; p1_VFR.lng = -87.786101; p1_VFR.z = 3000*ft2m;
LatLong2XY(regionID, p1_VFR.lat, p1_VFR.lng, p1_VFR.x, p1_VFR.y);
       p2_VFR.lat = 41.746730; p2_VFR.lng = -87.778837; p2_VFR.z = 3000*ft2m;
LatLong2XY(regionID, p2_VFR.lat, p2_VFR.lng, p2_VFR.x, p2_VFR.y);
```

```
	p3_VFR.lat = 41.741682; p3_VFR.lng = -87.786101; p3_VFR.z = 0*ft2m;
LatLong2XY(regionID, p3_VFR.lat, p3_VFR.lng, p3_VFR.x, p3_VFR.y);
	p4_VFR.lat = 41.746730; p4_VFR.lng = -87.778837; p4_VFR.z = 0*ft2m;
LatLong2XY(regionID, p4_VFR.lat, p4_VFR.lng, p4_VFR.x, p4_VFR.y);
	p5_VFR.lat = 41.827939; p5_VFR.lng = -87.889794; p5_VFR.z = 3000*ft2m;
LatLong2XY(regionID, p5_VFR.lat, p5_VFR.lng, p5_VFR.x, p5_VFR.y);
	p6_VFR.lat = 41.830154; p6_VFR.lng = -87.883381; p6_VFR.z = 3000*ft2m;
LatLong2XY(regionID, p6_VFR.lat, p6_VFR.lng, p6_VFR.x, p6_VFR.y);
	p7_VFR.lat = 41.827939; p7_VFR.lng = -87.889794; p7_VFR.z = 0*ft2m;
LatLong2XY(regionID, p7_VFR.lat, p7_VFR.lng, p7_VFR.x, p7_VFR.y);
	p8_VFR.lat = 41.830154; p8_VFR.lng = -87.883381; p8_VFR.z = 0*ft2m;
LatLong2XY(regionID, p8_VFR.lat, p8_VFR.lng, p8_VFR.x, p8_VFR.y);
	for (int j = 1; j <= flight.Track.size(); ++j) {
		Point p;
		p.lat = flight.Track[j-1].lat;
		p.lng = flight.Track[j-1].lng;
		p.z = flight.Track[j-1].z;
		LatLong2XY(regionID, p.lat, p.lng, p.x, p.y);
		if (flight.procedure == procedure_default) {
			if (inbox(p, p1_ILS, p2_ILS, p3_ILS, p4_ILS, p5_ILS,
p6_ILS, p7_ILS, p8_ILS))
				flight.procedure = "ILS";
			else if (inbox(p, p1_VFR, p2_VFR, p3_VFR, p4_VFR, p5_VFR,
p6_VFR, p7_VFR, p8_VFR))
				flight.procedure = "VFR";
		}
	}
	if (flight.procedure == procedure_default) {
		flight.procedure = "RNP";
	}
}
void identify_Flow(FLIGHT &flight) {
	for (int j = 1; j <= flight.Track.size(); ++j) {
		if (flight.Track.front().lng > -87.7518) {flight.flow = "east";}
//MDW.lng
		if (flight.Track.front().lng < -87.7518) {flight.flow = "west";}
//MDW.lng
	}
}
bool determine_GoAround(FLIGHT flight) {
	bool GA = false;
	double cumuangle = 0.0;
	double headingchange = 0.0;
	for (int j = 2; j <= flight.Track.size(); ++j) {
		headingchange = flight.Track[j-1].heading - flight.Track[j-
2].heading;
		if (headingchange < -PI)
			headingchange = flight.Track[j-1].heading + 2*PI -
flight.Track[j-2].heading;
		if (headingchange > PI)
			headingchange = flight.Track[j-1].heading - 2*PI -
flight.Track[j-2].heading;
		cumuangle += headingchange;
	}
	if (abs(cumuangle) > CUMUTHRESH && flight.time2landing_entrance >
TIMELIMIT) {
		GA = true;
	}
	return GA;
}
```

```
void determine_vertical_acquisition_method_and_event(FLIGHT &flight) {
	double currentdist = 0.0;
	double previousdist = 0.0;
	double currentvertdev = 0.0;
	double previousvertdev = 0.0;
	for (int j = flight.j_1000ftAGL; j <= flight.Track.size(); ++j) {
		currentdist = distance_2D(flight.Track[j-
1],flight.landingrunway.THR);
		currentvertdev = flight.Track[j-1].z -
(currentdist*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
		previousdist = distance_2D(flight.Track[(j-1)-
1],flight.landingrunway.THR);
		previousvertdev = flight.Track[(j-1)-1].z -(previousdist
*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
		if (flight.SAA1000 == false && previousvertdev >
VERTICALHALFWIDTH_THR + (previousdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR) && currentvertdev < VERTICALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR)) {
			flight.SAA1000 = true; //SAA1000
		}
		if (flight.SAB1000 == false && previousvertdev < -
(VERTICALHALFWIDTH_THR + (previousdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR)) && currentvertdev > -(VERTICALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR))) {
			flight.SAB1000 = true; //SAB1000
		}
	}
	for (int j = flight.j_750ftAGL; j <= flight.Track.size(); ++j) {
		currentdist = distance_2D(flight.Track[j-
1],flight.landingrunway.THR);
		currentvertdev = flight.Track[j-1].z -
(currentdist*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
		previousdist = distance_2D(flight.Track[(j-1)-
1],flight.landingrunway.THR);
		previousvertdev = flight.Track[(j-1)-1].z -(previousdist
*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
		if (flight.SAA750 == false && previousvertdev >
VERTICALHALFWIDTH_THR + (previousdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR) && currentvertdev < VERTICALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR)) {
			flight.SAA750 = true;
		}
		if (flight.SAB750 == false && previousvertdev < -
(VERTICALHALFWIDTH_THR + (previousdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR)) && currentvertdev > -(VERTICALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR))) {
			flight.SAB750 = true;
		}
	}
	for (int j = flight.j_500ftAGL; j <= flight.Track.size(); ++j) {
		currentdist = distance_2D(flight.Track[j-
1],flight.landingrunway.THR);
		currentvertdev = flight.Track[j-1].z -
(currentdist*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
		previousdist = distance_2D(flight.Track[(j-1)-
1],flight.landingrunway.THR);
		previousvertdev = flight.Track[(j-1)-1].z -(previousdist
*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
		if (flight.SAA500 == false && previousvertdev >
VERTICALHALFWIDTH_THR + (previousdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
```

```
VERTICALHALFWIDTH_THR) && currentvertdev < VERTICALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR)) {
                        flight.SAA500 = true;
                }
                if (flight.SAB500 == false && previousvertdev < -
(VERTICALHALFWIDTH_THR + (previousdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR)) && currentvertdev > -(VERTICALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR))) {
                        flight.SAB500 = true;
                }
        }

}
void determine_lateral_acquisition_method_and_event(FLIGHT &flight) {
        double currentdist = 0.0;
        double currentlatdev = 0.0;
        double previousdist = 0.0;
        double previouslatdev = 0.0;
        for (int j = flight.j_1000ftAGL; j <= flight.j_THR; ++j) {
                currentdist = distance_2D(flight.Track[j-
1],flight.landingrunway.THR);
                currentlatdev = dist_point2line_2D(flight.Track[j-1],
flight.landingrunway.FAF, flight.landingrunway.THR); //always > 0
                previousdist = distance_2D(flight.Track[(j-1)-
1],flight.landingrunway.THR);
                previouslatdev = dist_point2line_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF, flight.landingrunway.THR);;
                if (previouslatdev > LATERALHALFWIDTH_THR +
(previousdist/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)
                        && currentlatdev < LATERALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)) {
                        if (distance_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF_leftup) < distance_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF_rightup))
                                flight.SAL1000 = true;
                        else
                                flight.SAR1000 = true;
                }
        }
        for (int j = flight.j_750ftAGL; j <= flight.j_THR; ++j) {
                currentdist = distance_2D(flight.Track[j-
1],flight.landingrunway.THR);
                currentlatdev = dist_point2line_2D(flight.Track[j-1],
flight.landingrunway.FAF, flight.landingrunway.THR); //always > 0
                previousdist = distance_2D(flight.Track[(j-1)-
1],flight.landingrunway.THR);
                previouslatdev = dist_point2line_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF, flight.landingrunway.THR);;
                if (previouslatdev > LATERALHALFWIDTH_THR +
(previousdist/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)
                        && currentlatdev < LATERALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)) {
                        if (distance_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF_leftup) < distance_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF_rightup))
                                flight.SAL750 = true;
                        else
                                flight.SAR750 = true;
                }
        }
```

```
	for (int j = flight.j_500ftAGL; j <= flight.j_THR; ++j) {
		currentdist = distance_2D(flight.Track[j-
1],flight.landingrunway.THR);
		currentlatdev = dist_point2line_2D(flight.Track[j-1],
flight.landingrunway.FAF, flight.landingrunway.THR); //always > 0
		previousdist = distance_2D(flight.Track[(j-1)-
1],flight.landingrunway.THR);
		previouslatdev = dist_point2line_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF, flight.landingrunway.THR);;
		if (previouslatdev > LATERALHALFWIDTH_THR +
(previousdist/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)
			&& currentlatdev < LATERALHALFWIDTH_THR +
(currentdist/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)) {
			if (distance_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF_leftup) < distance_2D(flight.Track[(j-1)-1],
flight.landingrunway.FAF_rightup))
				flight.SAL500 = true;
			else
				flight.SAR500 = true;
		}
	}
}
void find_lateral_vertical_acquisition_point(FLIGHT &flight) {
	for (int j = 1; j <= flight.Track.size(); ++j) {
		Point PP;
		PP.lat = flight.Track[j-1].lat;
		PP.lng = flight.Track[j-1].lng;
		PP.x = flight.Track[j-1].x;
		PP.y = flight.Track[j-1].y;
		PP.z = flight.Track[j-1].z;
		//lateral acquisition point
		if (flight.j_acquisition_lateral == j_default) {
			bool in_vertical_relaxed_12nm_to_FAF_zone = inbox(PP,
flight.landingrunway.LOC_12nm_leftup_verticalrelax,
flight.landingrunway.LOC_12nm_rightup_verticalrelax,
flight.landingrunway.FAF_leftup_verticalrelax,
flight.landingrunway.FAF_rightup_verticalrelax,
flight.landingrunway.LOC_12nm_leftdown_verticalrelax,
flight.landingrunway.LOC_12nm_rightdown_verticalrelax,
flight.landingrunway.FAF_leftdown_verticalrelax,
flight.landingrunway.FAF_rightdown_verticalrelax);
			bool in_vertical_relaxed_FAF_to_REF_zone = inbox(PP,
flight.landingrunway.FAF_leftup_verticalrelax,
flight.landingrunway.FAF_rightup_verticalrelax,
flight.landingrunway.REF_leftup_verticalrelax,
flight.landingrunway.REF_rightup_verticalrelax,
flight.landingrunway.FAF_leftdown_verticalrelax,
flight.landingrunway.FAF_rightdown_verticalrelax,
flight.landingrunway.REF_leftdown_verticalrelax,
flight.landingrunway.REF_rightdown_verticalrelax);
			bool in_vertical_relaxed_REF_to_THR_zone = inbox(PP,
flight.landingrunway.REF_leftup_verticalrelax,
flight.landingrunway.REF_rightup_verticalrelax,
flight.landingrunway.THR_leftup_verticalrelax,
flight.landingrunway.THR_rightup_verticalrelax,
flight.landingrunway.REF_leftdown_verticalrelax,
flight.landingrunway.REF_rightdown_verticalrelax,
flight.landingrunway.THR_leftdown_verticalrelax,
flight.landingrunway.THR_rightdown_verticalrelax);
```

```
                    if (in_vertical_relaxed_12nm_to_FAF_zone ||
in_vertical_relaxed_FAF_to_REF_zone || in_vertical_relaxed_REF_to_THR_zone) {
                           flight.j_acquisition_lateral = j;
                           flight.j_beforelateralintercept = j-1;
                    }
             }
             //vertical acquisition point
             if (flight.j_acquisition_vertical == j_default) {
                    bool in_lateral_relaxed_12nm_to_FAF_zone = inbox(PP,
flight.landingrunway.LOC_12nm_leftup_horizontalrelax,
flight.landingrunway.LOC_12nm_rightup_horizontalrelax,
flight.landingrunway.FAF_leftup_horizontalrelax,
flight.landingrunway.FAF_rightup_horizontalrelax,
flight.landingrunway.LOC_12nm_leftdown_horizontalrelax,
flight.landingrunway.LOC_12nm_rightdown_horizontalrelax,
flight.landingrunway.FAF_leftdown_horizontalrelax,
flight.landingrunway.FAF_rightdown_horizontalrelax);
                    bool in_lateral_relaxed_FAF_to_REF_zone = inbox(PP,
flight.landingrunway.FAF_leftup_horizontalrelax,
flight.landingrunway.FAF_rightup_horizontalrelax,
flight.landingrunway.REF_leftup_horizontalrelax,
flight.landingrunway.REF_rightup_horizontalrelax,
flight.landingrunway.FAF_leftdown_horizontalrelax,
flight.landingrunway.FAF_rightdown_horizontalrelax,
flight.landingrunway.REF_leftdown_horizontalrelax,
flight.landingrunway.REF_rightdown_horizontalrelax);
                    bool in_lateral_relaxed_REF_to_THR_zone = inbox(PP,
flight.landingrunway.REF_leftup_horizontalrelax,
flight.landingrunway.REF_rightup_horizontalrelax,
flight.landingrunway.THR_leftup_horizontalrelax,
flight.landingrunway.THR_rightup_horizontalrelax,
flight.landingrunway.REF_leftdown_horizontalrelax,
flight.landingrunway.REF_rightdown_horizontalrelax,
flight.landingrunway.THR_leftdown_horizontalrelax,
flight.landingrunway.THR_rightdown_horizontalrelax);
                    if (in_lateral_relaxed_12nm_to_FAF_zone ||
in_lateral_relaxed_FAF_to_REF_zone || in_lateral_relaxed_REF_to_THR_zone) {
                           flight.j_acquisition_vertical = j;
                           flight.j_beforeverticalintercept = j-1;
                    }
             }
       }
}
void determine_wireframe_entrance_related(FLIGHT &flight){
       for (int j = 1; j <= flight.Track.size(); ++j) {
             Point Pj;
             Pj.lat = flight.Track[j-1].lat;
             Pj.lng = flight.Track[j-1].lng;
             Pj.x = flight.Track[j-1].x;
             Pj.y = flight.Track[j-1].y;
             Pj.z = flight.Track[j-1].z;
             bool inside_REF_to_THR_zone = inbox(Pj,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
             bool inside_FAF_to_REF_zone = inbox(Pj,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
```

```
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nm_to_FAF_zone = inbox(Pj,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            if ((inside_REF_to_THR_zone || inside_FAF_to_REF_zone ||
inside_12nm_to_FAF_zone) && j > 1) {  // if the beginning point of a track is
not inside the zone already
                  flight.entrance = true;
                  flight.j_firstentrance = j;
                  flight.j_beforeentrance = flight.j_firstentrance - 1;
                  //determine the time to LANDING (not to j_THR), for anomaly
detection (e.g. go-around)
                  flight.time2landing_entrance = flight.Track.back().t -
flight.Track[flight.j_firstentrance-1].t; //contains anomaly information for
go-around...
                  break; //if found, break the loop
            }
      }
}
bool track_qualified(FLIGHT flight) {
      bool qualified = false;
      qualified =
            flight.landingrunway.name != runwayname_default &&
            flight.aircraft.weight_class != "Unknown..." &&
            flight.entrance &&
            flight.Track.front().z - flight.landingrunway.elevation >
starting_min_altitude_AGL &&
            flight.Track.back().z - flight.landingrunway.elevation <
ending_max_altitude_AGL &&
            distance_2D(flight.Track.front(), flight.landingrunway.THR) >
starting_min_dist_to_THR &&
            distance_2D(flight.Track.back(), flight.landingrunway.THR) <
ending_max_dist_to_THR &&
            flight.time2landing_entrance < TIMELIMIT; //single approach
      return qualified;
}
void load_aircraft_data(vector<AIRCRAFT> &aircraft, string aircraftfile) {
      AIRCRAFT tempaircraft;
      ifstream in(aircraftfile);
      vector<string> cluster(3);
      char chars[200];
      in.getline(chars, 200);
      while(in.getline(chars, 200)) {
            split(chars, ",", cluster, 0);
            tempaircraft.model = cluster[0].c_str();
            tempaircraft.weight_class = cluster[1].c_str();
            tempaircraft.MTOW = atof(cluster[2].c_str());
            aircraft.push_back(tempaircraft);
      }
}
void get_aircraft_states(FLIGHT &flight, string dist) {
      int j_key = j_default;
      double lateral_deviation = 0.0;
      double vertical_deviation = 0.0;
      double avg_groundspeed = 0.0;
      double avg_airspeed = 0.0;
```

```
	double v_dev = 0.0;
	double avg_rateofdescent = 0.0;
	double Distance = 0.0;
	if (dist == "10nm") {
		j_key = flight.j_10nm_r;
		Distance = 10*nm2m;
	}
	else if (dist == "9nm") {
		j_key = flight.j_9nm_r;
		Distance = 9*nm2m;
	}
	else if (dist == "8nm") {
		j_key = flight.j_8nm_r;
		Distance = 8*nm2m;
	}
	else if (dist == "7nm") {
		j_key = flight.j_7nm_r;
		Distance = 7*nm2m;
	}
	else if (dist == "6nm") {
		j_key = flight.j_6nm_r;
		Distance = 6*nm2m;
	}
	else if (dist == "5.5nm") {
		j_key = flight.j_5dot5nm_r;
		Distance = 5.5*nm2m;
	}
	else if (dist == "5nm") {
		j_key = flight.j_5nm_r;
		Distance = 5*nm2m;
	}
	else if (dist == "4.5nm") {
		j_key = flight.j_4dot5nm_r;
		Distance = 4.5*nm2m;
	}
	else if (dist == "4nm") {
		j_key = flight.j_4nm_r;
		Distance = 4*nm2m;
	}
	else if (dist == "3.5nm") {
		j_key = flight.j_3dot5nm_r;
		Distance = 3.5*nm2m;
	}
	else if (dist == "3nm") {
		j_key = flight.j_3nm_r;
		Distance = 3*nm2m;
	}
	else if (dist == "1000ftAGL") {
		j_key = flight.j_1000ftAGL;
		Distance = distance_2D(flight.Track[flight.j_1000ftAGL-1],
flight.landingrunway.THR);
	}
	else if (dist == "750ftAGL") {
		j_key = flight.j_750ftAGL;
		Distance = distance_2D(flight.Track[flight.j_750ftAGL-1],
flight.landingrunway.THR);
	}
	else if (dist == "500ftAGL") {
		j_key = flight.j_500ftAGL;
```

```
            Distance = distance_2D(flight.Track[flight.j_500ftAGL-1],
flight.landingrunway.THR);
      }
      double baseline_v_10nm, baseline_v_9nm, baseline_v_8nm, baseline_v_7nm,
baseline_v_6nm, baseline_v_5dot5nm, baseline_v_5nm, baseline_v_4dot5nm,
baseline_v_4nm, baseline_v_3dot5nm, baseline_v_3nm, baseline_v_1000ftAGL,
baseline_v_750ftAGL, baseline_v_500ftAGL;
      double baseline_ROD_10nm, baseline_ROD_9nm, baseline_ROD_8nm,
baseline_ROD_7nm, baseline_ROD_6nm, baseline_ROD_5dot5nm, baseline_ROD_5nm,
baseline_ROD_4dot5nm, baseline_ROD_4nm, baseline_ROD_3dot5nm, baseline_ROD_3nm,
baseline_ROD_1000ftAGL, baseline_ROD_750ftAGL, baseline_ROD_500ftAGL;
      if (flight.aircraft.weight_class == "Superheavy" ||
flight.aircraft.weight_class == "Heavy") {
            baseline_v_10nm = BASELINE_AVG_AIRSPEED_10nm_Heavy;
baseline_ROD_10nm = BASELINE_AVG_ROD_10nm_Heavy;
            baseline_v_9nm = BASELINE_AVG_AIRSPEED_9nm_Heavy; baseline_ROD_9nm
= BASELINE_AVG_ROD_9nm_Heavy;
            baseline_v_8nm = BASELINE_AVG_AIRSPEED_8nm_Heavy; baseline_ROD_8nm
= BASELINE_AVG_ROD_8nm_Heavy;
            baseline_v_7nm = BASELINE_AVG_AIRSPEED_7nm_Heavy; baseline_ROD_7nm
= BASELINE_AVG_ROD_7nm_Heavy;
            baseline_v_6nm = BASELINE_AVG_AIRSPEED_6nm_Heavy; baseline_ROD_6nm
= BASELINE_AVG_ROD_6nm_Heavy;
            baseline_v_5dot5nm = BASELINE_AVG_AIRSPEED_5dot5nm_Heavy;
baseline_ROD_5dot5nm = BASELINE_AVG_ROD_5dot5nm_Heavy;
            baseline_v_5nm = BASELINE_AVG_AIRSPEED_5nm_Heavy; baseline_ROD_5nm
= BASELINE_AVG_ROD_5nm_Heavy;
            baseline_v_4dot5nm = BASELINE_AVG_AIRSPEED_4dot5nm_Heavy;
baseline_ROD_4dot5nm = BASELINE_AVG_ROD_4dot5nm_Heavy;
            baseline_v_4nm = BASELINE_AVG_AIRSPEED_4nm_Heavy; baseline_ROD_4nm
= BASELINE_AVG_ROD_4nm_Heavy;
            baseline_v_3dot5nm = BASELINE_AVG_AIRSPEED_3dot5nm_Heavy;
baseline_ROD_3dot5nm = BASELINE_AVG_ROD_3dot5nm_Heavy;
            baseline_v_3nm = BASELINE_AVG_AIRSPEED_3nm_Heavy; baseline_ROD_3nm
= BASELINE_AVG_ROD_3nm_Heavy;
            baseline_v_1000ftAGL = BASELINE_AVG_AIRSPEED_1000ftAGL_Heavy;
baseline_ROD_1000ftAGL = BASELINE_AVG_ROD_1000ftAGL_Heavy;
            baseline_v_750ftAGL = BASELINE_AVG_AIRSPEED_750ftAGL_Heavy;
baseline_ROD_750ftAGL = BASELINE_AVG_ROD_750ftAGL_Heavy;
            baseline_v_500ftAGL = BASELINE_AVG_AIRSPEED_500ftAGL_Heavy;
baseline_ROD_500ftAGL = BASELINE_AVG_ROD_500ftAGL_Heavy;
      }
      else if (flight.aircraft.weight_class == "B757") {
            baseline_v_10nm = BASELINE_AVG_AIRSPEED_10nm_B757;
baseline_ROD_10nm = BASELINE_AVG_ROD_10nm_B757;
            baseline_v_9nm = BASELINE_AVG_AIRSPEED_9nm_B757; baseline_ROD_9nm
= BASELINE_AVG_ROD_9nm_B757;
            baseline_v_8nm = BASELINE_AVG_AIRSPEED_8nm_B757; baseline_ROD_8nm
= BASELINE_AVG_ROD_8nm_B757;
            baseline_v_7nm = BASELINE_AVG_AIRSPEED_7nm_B757; baseline_ROD_7nm
= BASELINE_AVG_ROD_7nm_B757;
            baseline_v_6nm = BASELINE_AVG_AIRSPEED_6nm_B757; baseline_ROD_6nm
= BASELINE_AVG_ROD_6nm_B757;
            baseline_v_5dot5nm = BASELINE_AVG_AIRSPEED_5dot5nm_B757;
baseline_ROD_5dot5nm = BASELINE_AVG_ROD_5dot5nm_B757;
            baseline_v_5nm = BASELINE_AVG_AIRSPEED_5nm_B757; baseline_ROD_5nm
= BASELINE_AVG_ROD_5nm_B757;
            baseline_v_4dot5nm = BASELINE_AVG_AIRSPEED_4dot5nm_B757;
baseline_ROD_4dot5nm = BASELINE_AVG_ROD_4dot5nm_B757;
```

```
            baseline_v_4nm = BASELINE_AVG_AIRSPEED_4nm_B757; baseline_ROD_4nm
= BASELINE_AVG_ROD_4nm_B757;
            baseline_v_3dot5nm = BASELINE_AVG_AIRSPEED_3dot5nm_B757;
baseline_ROD_3dot5nm = BASELINE_AVG_ROD_3dot5nm_B757;
            baseline_v_3nm = BASELINE_AVG_AIRSPEED_3nm_B757; baseline_ROD_3nm
= BASELINE_AVG_ROD_3nm_B757;
            baseline_v_1000ftAGL = BASELINE_AVG_AIRSPEED_1000ftAGL_B757;
baseline_ROD_1000ftAGL = BASELINE_AVG_ROD_1000ftAGL_B757;
            baseline_v_750ftAGL = BASELINE_AVG_AIRSPEED_750ftAGL_B757;
baseline_ROD_750ftAGL = BASELINE_AVG_ROD_750ftAGL_B757;
            baseline_v_500ftAGL = BASELINE_AVG_AIRSPEED_500ftAGL_B757;
baseline_ROD_500ftAGL = BASELINE_AVG_ROD_500ftAGL_B757;
      }
      else if (flight.aircraft.weight_class == "Large") {
            baseline_v_10nm = BASELINE_AVG_AIRSPEED_10nm_Large;
baseline_ROD_10nm = BASELINE_AVG_ROD_10nm_Large;
            baseline_v_9nm = BASELINE_AVG_AIRSPEED_9nm_Large; baseline_ROD_9nm
= BASELINE_AVG_ROD_9nm_Large;
            baseline_v_8nm = BASELINE_AVG_AIRSPEED_8nm_Large; baseline_ROD_8nm
= BASELINE_AVG_ROD_8nm_Large;
            baseline_v_7nm = BASELINE_AVG_AIRSPEED_7nm_Large; baseline_ROD_7nm
= BASELINE_AVG_ROD_7nm_Large;
            baseline_v_6nm = BASELINE_AVG_AIRSPEED_6nm_Large; baseline_ROD_6nm
= BASELINE_AVG_ROD_6nm_Large;
            baseline_v_5dot5nm = BASELINE_AVG_AIRSPEED_5dot5nm_Large;
baseline_ROD_5dot5nm = BASELINE_AVG_ROD_5dot5nm_Large;
            baseline_v_5nm = BASELINE_AVG_AIRSPEED_5nm_Large; baseline_ROD_5nm
= BASELINE_AVG_ROD_5nm_Large;
            baseline_v_4dot5nm = BASELINE_AVG_AIRSPEED_4dot5nm_Large;
baseline_ROD_4dot5nm = BASELINE_AVG_ROD_4dot5nm_Large;
            baseline_v_4nm = BASELINE_AVG_AIRSPEED_4nm_Large; baseline_ROD_4nm
= BASELINE_AVG_ROD_4nm_Large;
            baseline_v_3dot5nm = BASELINE_AVG_AIRSPEED_3dot5nm_Large;
baseline_ROD_3dot5nm = BASELINE_AVG_ROD_3dot5nm_Large;
            baseline_v_3nm = BASELINE_AVG_AIRSPEED_3nm_Large; baseline_ROD_3nm
= BASELINE_AVG_ROD_3nm_Large;
            baseline_v_1000ftAGL = BASELINE_AVG_AIRSPEED_1000ftAGL_Large;
baseline_ROD_1000ftAGL = BASELINE_AVG_ROD_1000ftAGL_Large;
            baseline_v_750ftAGL = BASELINE_AVG_AIRSPEED_750ftAGL_Large;
baseline_ROD_750ftAGL = BASELINE_AVG_ROD_750ftAGL_Large;
            baseline_v_500ftAGL = BASELINE_AVG_AIRSPEED_500ftAGL_Large;
baseline_ROD_500ftAGL = BASELINE_AVG_ROD_500ftAGL_Large;
      }
      else if (flight.aircraft.weight_class == "Small") {
            baseline_v_10nm = BASELINE_AVG_AIRSPEED_10nm_Small;
baseline_ROD_10nm = BASELINE_AVG_ROD_10nm_Small;
            baseline_v_9nm = BASELINE_AVG_AIRSPEED_9nm_Small; baseline_ROD_9nm
= BASELINE_AVG_ROD_9nm_Small;
            baseline_v_8nm = BASELINE_AVG_AIRSPEED_8nm_Small; baseline_ROD_8nm
= BASELINE_AVG_ROD_8nm_Small;
            baseline_v_7nm = BASELINE_AVG_AIRSPEED_7nm_Small; baseline_ROD_7nm
= BASELINE_AVG_ROD_7nm_Small;
            baseline_v_6nm = BASELINE_AVG_AIRSPEED_6nm_Small; baseline_ROD_6nm
= BASELINE_AVG_ROD_6nm_Small;
            baseline_v_5dot5nm = BASELINE_AVG_AIRSPEED_5dot5nm_Small;
baseline_ROD_5dot5nm = BASELINE_AVG_ROD_5dot5nm_Small;
            baseline_v_5nm = BASELINE_AVG_AIRSPEED_5nm_Small; baseline_ROD_5nm
= BASELINE_AVG_ROD_5nm_Small;
            baseline_v_4dot5nm = BASELINE_AVG_AIRSPEED_4dot5nm_Small;
baseline_ROD_4dot5nm = BASELINE_AVG_ROD_4dot5nm_Small;
```

```
            baseline_v_4nm = BASELINE_AVG_AIRSPEED_4nm_Small; baseline_ROD_4nm
= BASELINE_AVG_ROD_4nm_Small;
            baseline_v_3dot5nm = BASELINE_AVG_AIRSPEED_3dot5nm_Small;
baseline_ROD_3dot5nm = BASELINE_AVG_ROD_3dot5nm_Small;
            baseline_v_3nm = BASELINE_AVG_AIRSPEED_3nm_Small; baseline_ROD_3nm
= BASELINE_AVG_ROD_3nm_Small;
            baseline_v_1000ftAGL = BASELINE_AVG_AIRSPEED_1000ftAGL_Small;
baseline_ROD_1000ftAGL = BASELINE_AVG_ROD_1000ftAGL_Small;
            baseline_v_750ftAGL = BASELINE_AVG_AIRSPEED_750ftAGL_Small;
baseline_ROD_750ftAGL = BASELINE_AVG_ROD_750ftAGL_Small;
            baseline_v_500ftAGL = BASELINE_AVG_AIRSPEED_500ftAGL_Small;
baseline_ROD_500ftAGL = BASELINE_AVG_ROD_500ftAGL_Small;
      }
      lateral_deviation = dist_point2line_2D(flight.Track[j_key-1],
flight.landingrunway.FAF, flight.landingrunway.THR);
      double d_left = distance_3D(flight.Track[j_key-1],
flight.landingrunway.FAF_leftup);
      double d_right = distance_3D(flight.Track[j_key-1],
flight.landingrunway.FAF_rightup);
      if (d_left > d_right)
            lateral_deviation = abs(lateral_deviation);
      else
            lateral_deviation = -abs(lateral_deviation);
      vertical_deviation = flight.Track[j_key-1].z -
(Distance*tan(flight.landingrunway.vgpa) + flight.landingrunway.THR.z);
            if (CURRENTAIRPORT == "EWR" && flight.landingrunway.name == "22L"
&& Distance > 9.4*nm2m)
                  vertical_deviation = flight.Track[j_key-1].z - 3000*ft2m;
      avg_groundspeed = get_avg_groundspeed(AVGBIN, flight, j_key, PROJECTING);
      avg_airspeed = get_avg_airspeed(AVGBIN, flight, j_key, PROJECTING);
      avg_rateofdescent = get_avg_rateofdescent(AVGBIN, flight, j_key);
      double headingdiff = flight.Track[j_key-1].heading -
flight.landingrunway.true_alignment;
      if (headingdiff < -PI)
            headingdiff = headingdiff + 2*PI;
      else if (headingdiff > PI)
            headingdiff = headingdiff - 2*PI;
      int j_12nm_r = j_default;
      for (int j = flight.Track.size(); j >= 1; j=j-1) {
            if (distance_2D(flight.Track[j-1], flight.landingrunway.THR) >
12.0 * nm2m) {
                  j_12nm_r = j;
                  break;
            }
      }
      flight.v_12nm = get_avg_airspeed(AVGBIN, flight, j_12nm_r, PROJECTING);
      flight.z_AGL_12nm = flight.Track[j_12nm_r-1].z -
flight.landingrunway.THR.z;
      double headingdiff12nm = flight.Track[j_12nm_r-1].heading -
flight.landingrunway.true_alignment;
      if (headingdiff12nm < -PI) headingdiff12nm = headingdiff12nm + 2*PI;
      else if (headingdiff12nm > PI) headingdiff12nm = headingdiff12nm - 2*PI;
      flight.headingdiff_12nm = headingdiff12nm;
      if (dist == "10nm") {
            flight.lateral_deviation_10nm = lateral_deviation;
            flight.vertical_deviation_10nm = vertical_deviation;
            flight.lateral_deviation_abs_10nm = abs(lateral_deviation);
            flight.vertical_deviation_abs_10nm = abs(vertical_deviation);
            flight.avg_groundspeed_10nm = avg_groundspeed;
            flight.avg_airspeed_10nm = avg_airspeed;
```

```
		flight.v_dev_10nm = abs(avg_groundspeed - baseline_v_10nm);
		flight.avg_rateofdescent_10nm = avg_rateofdescent;
		flight.rod_dev_10nm = abs(avg_rateofdescent - baseline_ROD_10nm);
		flight.heading_diff_10nm = headingdiff;
		//lateral deviation category
		if (flight.lateral_deviation_10nm < -LAT_DEV_BOUNDARY_10nm)
			flight.lateral_deviation_category_10nm =
STATECATEGORY_lateraldev_left;
		else if (flight.lateral_deviation_10nm > LAT_DEV_BOUNDARY_10nm)
			flight.lateral_deviation_category_10nm =
STATECATEGORY_lateraldev_right;
		else
			flight.lateral_deviation_category_10nm =
STATECATEGORY_lateraldev_normal;
		//vertical deviation category
		if (flight.vertical_deviation_10nm < -VERT_DEV_BOUNDARY_10nm)
			flight.vertical_deviation_category_10nm =
STATECATEGORY_verticaldev_low;
		else if (flight.vertical_deviation_10nm > VERT_DEV_BOUNDARY_10nm)
			flight.vertical_deviation_category_10nm =
STATECATEGORY_verticaldev_high;
		else
			flight.vertical_deviation_category_10nm =
STATECATEGORY_verticaldev_normal;
		//avg_airspeed category
		if (flight.avg_airspeed_10nm < baseline_v_10nm -
AVG_AIRSPEED_BOUNDARY)
			flight.avg_airspeedcategory_10nm =
STATECATEGORY_avgspeed_low;
		else if (flight.avg_airspeed_10nm > baseline_v_10nm +
AVG_AIRSPEED_BOUNDARY)
			flight.avg_airspeedcategory_10nm =
STATECATEGORY_avgspeed_high;
		else
			flight.avg_airspeedcategory_10nm =
STATECATEGORY_avgspeed_normal;
		//avg_ROD category
		if (flight.avg_rateofdescent_10nm > baseline_ROD_10nm +
AVG_ROD_BOUNDARY) //"low" means less descent rate
			flight.avg_rateofdescent_category_10nm =
STATECATEGORY_avgROD_low;
		else if (flight.avg_rateofdescent_10nm < baseline_ROD_10nm -
AVG_ROD_BOUNDARY) //"high" means high descent rate, e.g. >1000ftpm
			flight.avg_rateofdescent_category_10nm =
STATECATEGORY_avgROD_high;
		else
			flight.avg_rateofdescent_category_10nm =
STATECATEGORY_avgROD_normal;
		//heading difference category (may not be used for now)
		if (flight.heading_diff_10nm < -HEADING_DIFF_BOUNDARY)
			flight.heading_diff_category_10nm =
STATECATEGORY_headingdiff_counterclockwise;
		else if (flight.heading_diff_10nm > HEADING_DIFF_BOUNDARY)
			flight.heading_diff_category_10nm =
STATECATEGORY_headingdiff_clockwise;
		else
			flight.heading_diff_category_10nm =
STATECATEGORY_headingdiff_normal;
	}
	else if (dist == "9nm") {
```

```
            flight.lateral_deviation_9nm = lateral_deviation;
            flight.vertical_deviation_9nm = vertical_deviation;
            flight.lateral_deviation_abs_9nm = abs(lateral_deviation);
            flight.vertical_deviation_abs_9nm = abs(vertical_deviation);
            flight.avg_groundspeed_9nm = avg_groundspeed;
            flight.avg_airspeed_9nm = avg_airspeed;
            flight.v_dev_9nm = abs(avg_groundspeed - baseline_v_9nm);
            flight.avg_rateofdescent_9nm = avg_rateofdescent;
            flight.rod_dev_9nm = abs(avg_rateofdescent - baseline_ROD_9nm);
            flight.heading_diff_9nm = headingdiff;
            //lateral deviation category
            if (flight.lateral_deviation_9nm < -LAT_DEV_BOUNDARY_9nm)
                  flight.lateral_deviation_category_9nm =
STATECATEGORY_lateraldev_left;
            else if (flight.lateral_deviation_9nm > LAT_DEV_BOUNDARY_9nm)
                  flight.lateral_deviation_category_9nm =
STATECATEGORY_lateraldev_right;
            else
                  flight.lateral_deviation_category_9nm =
STATECATEGORY_lateraldev_normal;
            //vertical deviation category
            if (flight.vertical_deviation_9nm < -VERT_DEV_BOUNDARY_9nm)
                  flight.vertical_deviation_category_9nm =
STATECATEGORY_verticaldev_low;
            else if (flight.vertical_deviation_9nm > VERT_DEV_BOUNDARY_9nm)
                  flight.vertical_deviation_category_9nm =
STATECATEGORY_verticaldev_high;
            else
                  flight.vertical_deviation_category_9nm =
STATECATEGORY_verticaldev_normal;
            //avg_airspeed category
            if (flight.avg_airspeed_9nm < baseline_v_9nm -
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_9nm =
STATECATEGORY_avgspeed_low;
            else if (flight.avg_airspeed_9nm > baseline_v_9nm +
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_9nm =
STATECATEGORY_avgspeed_high;
            else
                  flight.avg_airspeedcategory_9nm =
STATECATEGORY_avgspeed_normal;
            //avg_ROD category
            if (flight.avg_rateofdescent_9nm > baseline_ROD_9nm +
AVG_ROD_BOUNDARY) //"low" means less descent rate
                  flight.avg_rateofdescent_category_9nm =
STATECATEGORY_avgROD_low;
            else if (flight.avg_rateofdescent_9nm < baseline_ROD_9nm -
AVG_ROD_BOUNDARY) //"high" means high descent rate, e.g. >1000ftpm
                  flight.avg_rateofdescent_category_9nm =
STATECATEGORY_avgROD_high;
            else
                  flight.avg_rateofdescent_category_9nm =
STATECATEGORY_avgROD_normal;
            //heading difference category (may not be used for now)
            if (flight.heading_diff_9nm < -HEADING_DIFF_BOUNDARY)
                  flight.heading_diff_category_9nm =
STATECATEGORY_headingdiff_counterclockwise;
            else if (flight.heading_diff_9nm > HEADING_DIFF_BOUNDARY)
```

```
                    flight.heading_diff_category_9nm =
STATECATEGORY_headingdiff_clockwise;
             else
                    flight.heading_diff_category_9nm =
STATECATEGORY_headingdiff_normal;
       }
       else if (dist == "8nm") {
             flight.lateral_deviation_8nm = lateral_deviation;
             flight.vertical_deviation_8nm = vertical_deviation;
             flight.lateral_deviation_abs_8nm = abs(lateral_deviation);
             flight.vertical_deviation_abs_8nm = abs(vertical_deviation);
             flight.avg_groundspeed_8nm = avg_groundspeed;
             flight.avg_airspeed_8nm = avg_airspeed;
             flight.v_dev_8nm = abs(avg_groundspeed - baseline_v_8nm);
             flight.avg_rateofdescent_8nm = avg_rateofdescent;
             flight.rod_dev_8nm = abs(avg_rateofdescent - baseline_ROD_8nm);
             flight.heading_diff_8nm = headingdiff;
             //lateral deviation category
             if (flight.lateral_deviation_8nm < -LAT_DEV_BOUNDARY_8nm)
                    flight.lateral_deviation_category_8nm =
STATECATEGORY_lateraldev_left;
             else if (flight.lateral_deviation_8nm > LAT_DEV_BOUNDARY_8nm)
                    flight.lateral_deviation_category_8nm =
STATECATEGORY_lateraldev_right;
             else
                    flight.lateral_deviation_category_8nm =
STATECATEGORY_lateraldev_normal;
             //vertical deviation category
             if (flight.vertical_deviation_8nm < -VERT_DEV_BOUNDARY_8nm)
                    flight.vertical_deviation_category_8nm =
STATECATEGORY_verticaldev_low;
             else if (flight.vertical_deviation_8nm > VERT_DEV_BOUNDARY_8nm)
                    flight.vertical_deviation_category_8nm =
STATECATEGORY_verticaldev_high;
             else
                    flight.vertical_deviation_category_8nm =
STATECATEGORY_verticaldev_normal;
             //avg_airspeed category
             if (flight.avg_airspeed_8nm < baseline_v_8nm -
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_8nm =
STATECATEGORY_avgspeed_low;
             else if (flight.avg_airspeed_8nm > baseline_v_8nm +
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_8nm =
STATECATEGORY_avgspeed_high;
             else
                    flight.avg_airspeedcategory_8nm =
STATECATEGORY_avgspeed_normal;
             //avg_ROD category
             if (flight.avg_rateofdescent_8nm > baseline_ROD_8nm +
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_8nm =
STATECATEGORY_avgROD_low;
             else if (flight.avg_rateofdescent_8nm < baseline_ROD_8nm -
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_8nm =
STATECATEGORY_avgROD_high;
             else
```

```
                    flight.avg_rateofdescent_category_8nm =
STATECATEGORY_avgROD_normal;
             //heading difference category (may not be used for now)
             if (flight.heading_diff_8nm < -HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_8nm =
STATECATEGORY_headingdiff_counterclockwise;
             else if (flight.heading_diff_8nm > HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_8nm =
STATECATEGORY_headingdiff_clockwise;
             else
                    flight.heading_diff_category_8nm =
STATECATEGORY_headingdiff_normal;
       }
       else if (dist == "7nm") {
             flight.lateral_deviation_7nm = lateral_deviation;
             flight.vertical_deviation_7nm = vertical_deviation;
             flight.lateral_deviation_abs_7nm = abs(lateral_deviation);
             flight.vertical_deviation_abs_7nm = abs(vertical_deviation);
             flight.avg_groundspeed_7nm = avg_groundspeed;
             flight.avg_airspeed_7nm = avg_airspeed;
             flight.v_dev_7nm = abs(avg_groundspeed - baseline_v_7nm);
             flight.avg_rateofdescent_7nm = avg_rateofdescent;
             flight.rod_dev_7nm = abs(avg_rateofdescent - baseline_ROD_7nm);
             flight.heading_diff_7nm = headingdiff;
             //lateral deviation category
             if (flight.lateral_deviation_7nm < -LAT_DEV_BOUNDARY_7nm)
                    flight.lateral_deviation_category_7nm =
STATECATEGORY_lateraldev_left;
             else if (flight.lateral_deviation_7nm > LAT_DEV_BOUNDARY_7nm)
                    flight.lateral_deviation_category_7nm =
STATECATEGORY_lateraldev_right;
             else
                    flight.lateral_deviation_category_7nm =
STATECATEGORY_lateraldev_normal;
             //vertical deviation category
             if (flight.vertical_deviation_7nm < -VERT_DEV_BOUNDARY_7nm)
                    flight.vertical_deviation_category_7nm =
STATECATEGORY_verticaldev_low;
             else if (flight.vertical_deviation_7nm > VERT_DEV_BOUNDARY_7nm)
                    flight.vertical_deviation_category_7nm =
STATECATEGORY_verticaldev_high;
             else
                    flight.vertical_deviation_category_7nm =
STATECATEGORY_verticaldev_normal;
             //avg_airspeed category
             if (flight.avg_airspeed_7nm < baseline_v_7nm -
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_7nm =
STATECATEGORY_avgspeed_low;
             else if (flight.avg_airspeed_7nm > baseline_v_7nm +
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_7nm =
STATECATEGORY_avgspeed_high;
             else
                    flight.avg_airspeedcategory_7nm =
STATECATEGORY_avgspeed_normal;
             //avg_ROD category
             if (flight.avg_rateofdescent_7nm > baseline_ROD_7nm +
AVG_ROD_BOUNDARY)
```

```
                    flight.avg_rateofdescent_category_7nm =
STATECATEGORY_avgROD_low;
             else if (flight.avg_rateofdescent_7nm < baseline_ROD_7nm -
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_7nm =
STATECATEGORY_avgROD_high;
             else
                    flight.avg_rateofdescent_category_7nm =
STATECATEGORY_avgROD_normal;
             //heading difference category (may not be used for now)
             if (flight.heading_diff_7nm < -HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_7nm =
STATECATEGORY_headingdiff_counterclockwise;
             else if (flight.heading_diff_7nm > HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_7nm =
STATECATEGORY_headingdiff_clockwise;
             else
                    flight.heading_diff_category_7nm =
STATECATEGORY_headingdiff_normal;
       }
       else if (dist == "6nm") {
             flight.lateral_deviation_6nm = lateral_deviation;
             flight.vertical_deviation_6nm = vertical_deviation;
             flight.lateral_deviation_abs_6nm = abs(lateral_deviation);
             flight.vertical_deviation_abs_6nm = abs(vertical_deviation);
             flight.avg_groundspeed_6nm = avg_groundspeed;
             flight.avg_airspeed_6nm = avg_airspeed;
             flight.v_dev_6nm = abs(avg_groundspeed - baseline_v_6nm);
             flight.avg_rateofdescent_6nm = avg_rateofdescent;
             flight.rod_dev_6nm = abs(avg_rateofdescent - baseline_ROD_6nm);
             flight.heading_diff_6nm = headingdiff;
             //lateral deviation category
             if (flight.lateral_deviation_6nm < -LAT_DEV_BOUNDARY_6nm)
                    flight.lateral_deviation_category_6nm =
STATECATEGORY_lateraldev_left;
             else if (flight.lateral_deviation_6nm > LAT_DEV_BOUNDARY_6nm)
                    flight.lateral_deviation_category_6nm =
STATECATEGORY_lateraldev_right;
             else
                    flight.lateral_deviation_category_6nm =
STATECATEGORY_lateraldev_normal;
             //vertical deviation category
             if (flight.vertical_deviation_6nm < -VERT_DEV_BOUNDARY_6nm)
                    flight.vertical_deviation_category_6nm =
STATECATEGORY_verticaldev_low;
             else if (flight.vertical_deviation_6nm > VERT_DEV_BOUNDARY_6nm)
                    flight.vertical_deviation_category_6nm =
STATECATEGORY_verticaldev_high;
             else
                    flight.vertical_deviation_category_6nm =
STATECATEGORY_verticaldev_normal;
             //avg_airspeed category
             if (flight.avg_airspeed_6nm < baseline_v_6nm -
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_6nm =
STATECATEGORY_avgspeed_low;
             else if (flight.avg_airspeed_6nm > baseline_v_6nm +
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_6nm =
STATECATEGORY_avgspeed_high;
```

```
		else
			flight.avg_airspeedcategory_6nm =
STATECATEGORY_avgspeed_normal;
		//avg_ROD category
		if (flight.avg_rateofdescent_6nm > baseline_ROD_6nm +
AVG_ROD_BOUNDARY)
			flight.avg_rateofdescent_category_6nm =
STATECATEGORY_avgROD_low;
		else if (flight.avg_rateofdescent_6nm < baseline_ROD_6nm -
AVG_ROD_BOUNDARY)
			flight.avg_rateofdescent_category_6nm =
STATECATEGORY_avgROD_high;
		else
			flight.avg_rateofdescent_category_6nm =
STATECATEGORY_avgROD_normal;
		//heading difference category (may not be used for now)
		if (flight.heading_diff_6nm < -HEADING_DIFF_BOUNDARY)
			flight.heading_diff_category_6nm =
STATECATEGORY_headingdiff_counterclockwise;
		else if (flight.heading_diff_6nm > HEADING_DIFF_BOUNDARY)
			flight.heading_diff_category_6nm =
STATECATEGORY_headingdiff_clockwise;
		else
			flight.heading_diff_category_6nm =
STATECATEGORY_headingdiff_normal;
	}
	else if (dist == "5.5nm") {
		flight.lateral_deviation_5dot5nm = lateral_deviation;
		flight.vertical_deviation_5dot5nm = vertical_deviation;
		flight.lateral_deviation_abs_5dot5nm = abs(lateral_deviation);
		flight.vertical_deviation_abs_5dot5nm = abs(vertical_deviation);
		flight.avg_groundspeed_5dot5nm = avg_groundspeed;
		flight.avg_airspeed_5dot5nm = avg_airspeed;
		flight.v_dev_5dot5nm = abs(avg_groundspeed - baseline_v_5dot5nm);
		flight.avg_rateofdescent_5dot5nm = avg_rateofdescent;
		flight.rod_dev_5dot5nm = abs(avg_rateofdescent -
baseline_ROD_5dot5nm);
		flight.heading_diff_5dot5nm = headingdiff;
		//lateral deviation category
		if (flight.lateral_deviation_5dot5nm < -LAT_DEV_BOUNDARY_5dot5nm)
			flight.lateral_deviation_category_5dot5nm =
STATECATEGORY_lateraldev_left;
		else if (flight.lateral_deviation_5dot5nm >
LAT_DEV_BOUNDARY_5dot5nm)
			flight.lateral_deviation_category_5dot5nm =
STATECATEGORY_lateraldev_right;
		else
			flight.lateral_deviation_category_5dot5nm =
STATECATEGORY_lateraldev_normal;
		//vertical deviation category
		if (flight.vertical_deviation_5dot5nm < -
VERT_DEV_BOUNDARY_5dot5nm)
			flight.vertical_deviation_category_5dot5nm =
STATECATEGORY_verticaldev_low;
		else if (flight.vertical_deviation_5dot5nm >
VERT_DEV_BOUNDARY_5dot5nm)
			flight.vertical_deviation_category_5dot5nm =
STATECATEGORY_verticaldev_high;
		else
```

```
                    flight.vertical_deviation_category_5dot5nm =
STATECATEGORY_verticaldev_normal;
             //avg_airspeed category
             if (flight.avg_airspeed_5dot5nm < baseline_v_5dot5nm -
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_5dot5nm =
STATECATEGORY_avgspeed_low;
             else if (flight.avg_airspeed_5dot5nm > baseline_v_5dot5nm +
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_5dot5nm =
STATECATEGORY_avgspeed_high;
             else
                    flight.avg_airspeedcategory_5dot5nm =
STATECATEGORY_avgspeed_normal;
             //avg_ROD category
             if (flight.avg_rateofdescent_5dot5nm > baseline_ROD_5dot5nm +
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_5dot5nm =
STATECATEGORY_avgROD_low;
             else if (flight.avg_rateofdescent_5dot5nm < baseline_ROD_5dot5nm -
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_5dot5nm =
STATECATEGORY_avgROD_high;
             else
                    flight.avg_rateofdescent_category_5dot5nm =
STATECATEGORY_avgROD_normal;
             //heading difference category (may not be used for now)
             if (flight.heading_diff_5dot5nm < -HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_5dot5nm =
STATECATEGORY_headingdiff_counterclockwise;
             else if (flight.heading_diff_5dot5nm > HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_5dot5nm =
STATECATEGORY_headingdiff_clockwise;
             else
                    flight.heading_diff_category_5dot5nm =
STATECATEGORY_headingdiff_normal;
       }
       else if (dist == "5nm") {
             flight.lateral_deviation_5nm = lateral_deviation;
             flight.vertical_deviation_5nm = vertical_deviation;
             flight.lateral_deviation_abs_5nm = abs(lateral_deviation);
             flight.vertical_deviation_abs_5nm = abs(vertical_deviation);
             flight.avg_groundspeed_5nm = avg_groundspeed;
             flight.avg_airspeed_5nm = avg_airspeed;
             flight.v_dev_5nm = abs(avg_groundspeed - baseline_v_5nm);
             flight.avg_rateofdescent_5nm = avg_rateofdescent;
             flight.rod_dev_5nm = abs(avg_rateofdescent - baseline_ROD_5nm);
             flight.heading_diff_5nm = headingdiff;
             //lateral deviation category
             if (flight.lateral_deviation_5nm < -LAT_DEV_BOUNDARY_5nm)
                    flight.lateral_deviation_category_5nm =
STATECATEGORY_lateraldev_left;
             else if (flight.lateral_deviation_5nm > LAT_DEV_BOUNDARY_5nm)
                    flight.lateral_deviation_category_5nm =
STATECATEGORY_lateraldev_right;
             else
                    flight.lateral_deviation_category_5nm =
STATECATEGORY_lateraldev_normal;
             //vertical deviation category
             if (flight.vertical_deviation_5nm < -VERT_DEV_BOUNDARY_5nm)
```

```
                    flight.vertical_deviation_category_5nm =
STATECATEGORY_verticaldev_low;
             else if (flight.vertical_deviation_5nm > VERT_DEV_BOUNDARY_5nm)
                    flight.vertical_deviation_category_5nm =
STATECATEGORY_verticaldev_high;
             else
                    flight.vertical_deviation_category_5nm =
STATECATEGORY_verticaldev_normal;
             //avg_airspeed category
             if (flight.avg_airspeed_5nm < baseline_v_5nm -
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_5nm =
STATECATEGORY_avgspeed_low;
             else if (flight.avg_airspeed_5nm > baseline_v_5nm +
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_5nm =
STATECATEGORY_avgspeed_high;
             else
                    flight.avg_airspeedcategory_5nm =
STATECATEGORY_avgspeed_normal;
             //avg_ROD category
             if (flight.avg_rateofdescent_5nm > baseline_ROD_5nm +
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_5nm =
STATECATEGORY_avgROD_low;
             else if (flight.avg_rateofdescent_5nm < baseline_ROD_5nm -
AVG_ROD_BOUNDARY)
                    flight.avg_rateofdescent_category_5nm =
STATECATEGORY_avgROD_high;
             else
                    flight.avg_rateofdescent_category_5nm =
STATECATEGORY_avgROD_normal;
             //heading difference category (may not be used for now)
             if (flight.heading_diff_5nm < -HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_5nm =
STATECATEGORY_headingdiff_counterclockwise;
             else if (flight.heading_diff_5nm > HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_5nm =
STATECATEGORY_headingdiff_clockwise;
             else
                    flight.heading_diff_category_5nm =
STATECATEGORY_headingdiff_normal;
       }
       else if (dist == "4.5nm") {
             flight.lateral_deviation_4dot5nm = lateral_deviation;
             flight.vertical_deviation_4dot5nm = vertical_deviation;
             flight.lateral_deviation_abs_4dot5nm = abs(lateral_deviation);
             flight.vertical_deviation_abs_4dot5nm = abs(vertical_deviation);
             flight.avg_groundspeed_4dot5nm = avg_groundspeed;
             flight.avg_airspeed_4dot5nm = avg_airspeed;
             flight.v_dev_4dot5nm = abs(avg_groundspeed - baseline_v_4dot5nm);
             flight.avg_rateofdescent_4dot5nm = avg_rateofdescent;
             flight.rod_dev_4dot5nm = abs(avg_rateofdescent -
baseline_ROD_4dot5nm);
             flight.heading_diff_4dot5nm = headingdiff;
             //lateral deviation category
             if (flight.lateral_deviation_4dot5nm < -LAT_DEV_BOUNDARY_4dot5nm)
                    flight.lateral_deviation_category_4dot5nm =
STATECATEGORY_lateraldev_left;
```

```
            else if (flight.lateral_deviation_4dot5nm >
LAT_DEV_BOUNDARY_4dot5nm)
                  flight.lateral_deviation_category_4dot5nm =
STATECATEGORY_lateraldev_right;
            else
                  flight.lateral_deviation_category_4dot5nm =
STATECATEGORY_lateraldev_normal;
            //vertical deviation category
            if (flight.vertical_deviation_4dot5nm < -
VERT_DEV_BOUNDARY_4dot5nm)
                  flight.vertical_deviation_category_4dot5nm =
STATECATEGORY_verticaldev_low;
            else if (flight.vertical_deviation_4dot5nm >
VERT_DEV_BOUNDARY_4dot5nm)
                  flight.vertical_deviation_category_4dot5nm =
STATECATEGORY_verticaldev_high;
            else
                  flight.vertical_deviation_category_4dot5nm =
STATECATEGORY_verticaldev_normal;
            //avg_airspeed category
            if (flight.avg_airspeed_4dot5nm < baseline_v_4dot5nm -
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_4dot5nm =
STATECATEGORY_avgspeed_low;
            else if (flight.avg_airspeed_4dot5nm > baseline_v_4dot5nm +
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_4dot5nm =
STATECATEGORY_avgspeed_high;
            else
                  flight.avg_airspeedcategory_4dot5nm =
STATECATEGORY_avgspeed_normal;
            //avg_ROD category
            if (flight.avg_rateofdescent_4dot5nm > baseline_ROD_4dot5nm +
AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_4dot5nm =
STATECATEGORY_avgROD_low;
            else if (flight.avg_rateofdescent_4dot5nm < baseline_ROD_4dot5nm -
AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_4dot5nm =
STATECATEGORY_avgROD_high;
            else
                  flight.avg_rateofdescent_category_4dot5nm =
STATECATEGORY_avgROD_normal;
            //heading difference category (may not be used for now)
            if (flight.heading_diff_4dot5nm < -HEADING_DIFF_BOUNDARY)
                  flight.heading_diff_category_4dot5nm =
STATECATEGORY_headingdiff_counterclockwise;
            else if (flight.heading_diff_4dot5nm > HEADING_DIFF_BOUNDARY)
                  flight.heading_diff_category_4dot5nm =
STATECATEGORY_headingdiff_clockwise;
            else
                  flight.heading_diff_category_4dot5nm =
STATECATEGORY_headingdiff_normal;
      }
      else if (dist == "4nm") {
            flight.lateral_deviation_4nm = lateral_deviation;
            flight.vertical_deviation_4nm = vertical_deviation;
            flight.lateral_deviation_abs_4nm = abs(lateral_deviation);
            flight.vertical_deviation_abs_4nm = abs(vertical_deviation);
            flight.avg_groundspeed_4nm = avg_groundspeed;
```

```
		flight.avg_airspeed_4nm = avg_airspeed;
		flight.v_dev_4nm = abs(avg_groundspeed - baseline_v_4nm);
		flight.avg_rateofdescent_4nm = avg_rateofdescent;
		flight.rod_dev_4nm = abs(avg_rateofdescent - baseline_ROD_4nm);
		flight.heading_diff_4nm = headingdiff;
		//lateral deviation category
		if (flight.lateral_deviation_4nm < -LAT_DEV_BOUNDARY_4nm)
			flight.lateral_deviation_category_4nm =
STATECATEGORY_lateraldev_left;
		else if (flight.lateral_deviation_4nm > LAT_DEV_BOUNDARY_4nm)
			flight.lateral_deviation_category_4nm =
STATECATEGORY_lateraldev_right;
		else
			flight.lateral_deviation_category_4nm =
STATECATEGORY_lateraldev_normal;
		//vertical deviation category
		if (flight.vertical_deviation_4nm < -VERT_DEV_BOUNDARY_4nm)
			flight.vertical_deviation_category_4nm =
STATECATEGORY_verticaldev_low;
		else if (flight.vertical_deviation_4nm > VERT_DEV_BOUNDARY_4nm)
			flight.vertical_deviation_category_4nm =
STATECATEGORY_verticaldev_high;
		else
			flight.vertical_deviation_category_4nm =
STATECATEGORY_verticaldev_normal;
		//avg_airspeed category
		if (flight.avg_airspeed_4nm < baseline_v_4nm -
AVG_AIRSPEED_BOUNDARY)
			flight.avg_airspeedcategory_4nm =
STATECATEGORY_avgspeed_low;
		else if (flight.avg_airspeed_4nm > baseline_v_4nm +
AVG_AIRSPEED_BOUNDARY)
			flight.avg_airspeedcategory_4nm =
STATECATEGORY_avgspeed_high;
		else
			flight.avg_airspeedcategory_4nm =
STATECATEGORY_avgspeed_normal;
		//avg_ROD category
		if (flight.avg_rateofdescent_4nm > baseline_ROD_4nm +
AVG_ROD_BOUNDARY)
			flight.avg_rateofdescent_category_4nm =
STATECATEGORY_avgROD_low;
		else if (flight.avg_rateofdescent_4nm < baseline_ROD_4nm -
AVG_ROD_BOUNDARY)
			flight.avg_rateofdescent_category_4nm =
STATECATEGORY_avgROD_high;
		else
			flight.avg_rateofdescent_category_4nm =
STATECATEGORY_avgROD_normal;
		//heading difference category (may not be used for now)
		if (flight.heading_diff_4nm < -HEADING_DIFF_BOUNDARY)
			flight.heading_diff_category_4nm =
STATECATEGORY_headingdiff_counterclockwise;
		else if (flight.heading_diff_4nm > HEADING_DIFF_BOUNDARY)
			flight.heading_diff_category_4nm =
STATECATEGORY_headingdiff_clockwise;
		else
			flight.heading_diff_category_4nm =
STATECATEGORY_headingdiff_normal;
	}
```

```
	else if (dist == "3.5nm") {
		flight.lateral_deviation_3dot5nm = lateral_deviation;
		flight.vertical_deviation_3dot5nm = vertical_deviation;
		flight.lateral_deviation_abs_3dot5nm = abs(lateral_deviation);
		flight.vertical_deviation_abs_3dot5nm = abs(vertical_deviation);
		flight.avg_groundspeed_3dot5nm = avg_groundspeed;
		flight.avg_airspeed_3dot5nm = avg_airspeed;
		flight.v_dev_3dot5nm = abs(avg_groundspeed - baseline_v_3dot5nm);
		flight.avg_rateofdescent_3dot5nm = avg_rateofdescent;
		flight.rod_dev_3dot5nm = abs(avg_rateofdescent -
baseline_ROD_3dot5nm);
		flight.heading_diff_3dot5nm = headingdiff;
		//lateral deviation category
		if (flight.lateral_deviation_3dot5nm < -LAT_DEV_BOUNDARY_3dot5nm)
			flight.lateral_deviation_category_3dot5nm =
STATECATEGORY_lateraldev_left;
		else if (flight.lateral_deviation_3dot5nm >
LAT_DEV_BOUNDARY_3dot5nm)
			flight.lateral_deviation_category_3dot5nm =
STATECATEGORY_lateraldev_right;
		else
			flight.lateral_deviation_category_3dot5nm =
STATECATEGORY_lateraldev_normal;
		//vertical deviation category
		if (flight.vertical_deviation_3dot5nm < -
VERT_DEV_BOUNDARY_3dot5nm)
			flight.vertical_deviation_category_3dot5nm =
STATECATEGORY_verticaldev_low;
		else if (flight.vertical_deviation_3dot5nm >
VERT_DEV_BOUNDARY_3dot5nm)
			flight.vertical_deviation_category_3dot5nm =
STATECATEGORY_verticaldev_high;
		else
			flight.vertical_deviation_category_3dot5nm =
STATECATEGORY_verticaldev_normal;
		//avg_airspeed category
		if (flight.avg_airspeed_3dot5nm < baseline_v_3dot5nm -
AVG_AIRSPEED_BOUNDARY)
			flight.avg_airspeedcategory_3dot5nm =
STATECATEGORY_avgspeed_low;
		else if (flight.avg_airspeed_3dot5nm > baseline_v_3dot5nm +
AVG_AIRSPEED_BOUNDARY)
			flight.avg_airspeedcategory_3dot5nm =
STATECATEGORY_avgspeed_high;
		else
			flight.avg_airspeedcategory_3dot5nm =
STATECATEGORY_avgspeed_normal;
		//avg_ROD category
		if (flight.avg_rateofdescent_3dot5nm > baseline_ROD_3dot5nm +
AVG_ROD_BOUNDARY)
			flight.avg_rateofdescent_category_3dot5nm =
STATECATEGORY_avgROD_low;
		else if (flight.avg_rateofdescent_3dot5nm < baseline_ROD_3dot5nm -
AVG_ROD_BOUNDARY)
			flight.avg_rateofdescent_category_3dot5nm =
STATECATEGORY_avgROD_high;
		else
			flight.avg_rateofdescent_category_3dot5nm =
STATECATEGORY_avgROD_normal;
		//heading difference category (may not be used for now)
```

```
            if (flight.heading_diff_3dot5nm < -HEADING_DIFF_BOUNDARY)
                  flight.heading_diff_category_3dot5nm =
STATECATEGORY_headingdiff_counterclockwise;
            else if (flight.heading_diff_3dot5nm > HEADING_DIFF_BOUNDARY)
                  flight.heading_diff_category_3dot5nm =
STATECATEGORY_headingdiff_clockwise;
            else
                  flight.heading_diff_category_3dot5nm =
STATECATEGORY_headingdiff_normal;
      }
      else if (dist == "3nm") {
            flight.lateral_deviation_3nm = lateral_deviation;
            flight.vertical_deviation_3nm = vertical_deviation;
            flight.lateral_deviation_abs_3nm = abs(lateral_deviation);
            flight.vertical_deviation_abs_3nm = abs(vertical_deviation);
            flight.avg_groundspeed_3nm = avg_groundspeed;
            flight.avg_airspeed_3nm = avg_airspeed;
            flight.v_dev_3nm = abs(avg_groundspeed - baseline_v_3nm);
            flight.avg_rateofdescent_3nm = avg_rateofdescent;
            flight.rod_dev_3nm = abs(avg_rateofdescent - baseline_ROD_3nm);
            flight.heading_diff_3nm = headingdiff;
            //lateral deviation category
            if (flight.lateral_deviation_3nm < -LAT_DEV_BOUNDARY_3nm)
                  flight.lateral_deviation_category_3nm =
STATECATEGORY_lateraldev_left;
            else if (flight.lateral_deviation_3nm > LAT_DEV_BOUNDARY_3nm)
                  flight.lateral_deviation_category_3nm =
STATECATEGORY_lateraldev_right;
            else
                  flight.lateral_deviation_category_3nm =
STATECATEGORY_lateraldev_normal;
            //vertical deviation category
            if (flight.vertical_deviation_3nm < -VERT_DEV_BOUNDARY_3nm)
                  flight.vertical_deviation_category_3nm =
STATECATEGORY_verticaldev_low;
            else if (flight.vertical_deviation_3nm > VERT_DEV_BOUNDARY_3nm)
                  flight.vertical_deviation_category_3nm =
STATECATEGORY_verticaldev_high;
            else
                  flight.vertical_deviation_category_3nm =
STATECATEGORY_verticaldev_normal;
            //avg_airspeed category
            if (flight.avg_airspeed_3nm < baseline_v_3nm -
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_3nm =
STATECATEGORY_avgspeed_low;
            else if (flight.avg_airspeed_3nm > baseline_v_3nm +
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_3nm =
STATECATEGORY_avgspeed_high;
            else
                  flight.avg_airspeedcategory_3nm =
STATECATEGORY_avgspeed_normal;
            //avg_ROD category
            if (flight.avg_rateofdescent_3nm > baseline_ROD_3nm +
AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_3nm =
STATECATEGORY_avgROD_low;
            else if (flight.avg_rateofdescent_3nm < baseline_ROD_3nm -
AVG_ROD_BOUNDARY)
```

```
                    flight.avg_rateofdescent_category_3nm =
STATECATEGORY_avgROD_high;
             else
                    flight.avg_rateofdescent_category_3nm =
STATECATEGORY_avgROD_normal;
             //heading difference category (may not be used for now)
             if (flight.heading_diff_3nm < -HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_3nm =
STATECATEGORY_headingdiff_counterclockwise;
             else if (flight.heading_diff_3nm > HEADING_DIFF_BOUNDARY)
                    flight.heading_diff_category_3nm =
STATECATEGORY_headingdiff_clockwise;
             else
                    flight.heading_diff_category_3nm =
STATECATEGORY_headingdiff_normal;
       }
       else if (dist == "1000ftAGL") {
             flight.lateral_deviation_1000ftAGL = lateral_deviation;
             flight.vertical_deviation_1000ftAGL = vertical_deviation;
             flight.avg_groundspeed_1000ftAGL = avg_groundspeed;
             flight.avg_airspeed_1000ftAGL = avg_airspeed;
             flight.avg_rateofdescent_1000ftAGL = avg_rateofdescent;
             //lateral deviation category
             if (flight.lateral_deviation_1000ftAGL < -(LATERALHALFWIDTH_THR +
(Distance/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)))
                    flight.lateral_deviation_category_1000ftAGL =
STATECATEGORY_lateraldev_left;
             else if (flight.lateral_deviation_1000ftAGL > LATERALHALFWIDTH_THR
+ (Distance/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR))
                    flight.lateral_deviation_category_1000ftAGL =
STATECATEGORY_lateraldev_right;
             else
                    flight.lateral_deviation_category_1000ftAGL =
STATECATEGORY_lateraldev_normal;
             //vertical deviation category
             if (flight.vertical_deviation_1000ftAGL < -(VERTICALHALFWIDTH_THR
+ (Distance/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR)))
                    flight.vertical_deviation_category_1000ftAGL =
STATECATEGORY_verticaldev_low;
             else if (flight.vertical_deviation_1000ftAGL >
VERTICALHALFWIDTH_THR + (Distance/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR))
                    flight.vertical_deviation_category_1000ftAGL =
STATECATEGORY_verticaldev_high;
             else
                    flight.vertical_deviation_category_1000ftAGL =
STATECATEGORY_verticaldev_normal;
             //avg_airspeed category
             if (flight.avg_airspeed_1000ftAGL < baseline_v_1000ftAGL -
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_1000ftAGL =
STATECATEGORY_avgspeed_low;
             else if (flight.avg_airspeed_1000ftAGL > baseline_v_1000ftAGL +
AVG_AIRSPEED_BOUNDARY)
                    flight.avg_airspeedcategory_1000ftAGL =
STATECATEGORY_avgspeed_high;
             else
                    flight.avg_airspeedcategory_1000ftAGL =
STATECATEGORY_avgspeed_normal;
             //avg_ROD category
```

```
            if (flight.avg_rateofdescent_1000ftAGL > baseline_ROD_1000ftAGL +
AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_1000ftAGL =
STATECATEGORY_avgROD_low;
            else if (flight.avg_rateofdescent_1000ftAGL <
baseline_ROD_1000ftAGL - AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_1000ftAGL =
STATECATEGORY_avgROD_high;
            else
                  flight.avg_rateofdescent_category_1000ftAGL =
STATECATEGORY_avgROD_normal;
      }
      else if (dist == "750ftAGL") {
            flight.lateral_deviation_750ftAGL = lateral_deviation;
            flight.vertical_deviation_750ftAGL = vertical_deviation;
            flight.avg_groundspeed_750ftAGL = avg_groundspeed;
            flight.avg_airspeed_750ftAGL = avg_airspeed;
            flight.avg_rateofdescent_750ftAGL = avg_rateofdescent;
            //lateral deviation category
            if (flight.lateral_deviation_750ftAGL < -(LATERALHALFWIDTH_THR +
(Distance/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)))
                  flight.lateral_deviation_category_750ftAGL =
STATECATEGORY_lateraldev_left;
            else if (flight.lateral_deviation_750ftAGL > LATERALHALFWIDTH_THR
+ (Distance/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR))
                  flight.lateral_deviation_category_750ftAGL =
STATECATEGORY_lateraldev_right;
            else
                  flight.lateral_deviation_category_750ftAGL =
STATECATEGORY_lateraldev_normal;
            //vertical deviation category
            if (flight.vertical_deviation_750ftAGL < -(VERTICALHALFWIDTH_THR +
(Distance/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR)))
                  flight.vertical_deviation_category_750ftAGL =
STATECATEGORY_verticaldev_low;
            else if (flight.vertical_deviation_500ftAGL >
VERTICALHALFWIDTH_THR + (Distance/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR))
                  flight.vertical_deviation_category_750ftAGL =
STATECATEGORY_verticaldev_high;
            else
                  flight.vertical_deviation_category_750ftAGL =
STATECATEGORY_verticaldev_normal;
            //avg_airspeed category
            if (flight.avg_airspeed_750ftAGL < baseline_v_750ftAGL -
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_750ftAGL =
STATECATEGORY_avgspeed_low;
            else if (flight.avg_airspeed_750ftAGL > baseline_v_750ftAGL +
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_750ftAGL =
STATECATEGORY_avgspeed_high;
            else
                  flight.avg_airspeedcategory_750ftAGL =
STATECATEGORY_avgspeed_normal;
            //avg_ROD category
            if (flight.avg_rateofdescent_750ftAGL > baseline_ROD_750ftAGL +
AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_750ftAGL =
STATECATEGORY_avgROD_low;
```

```
            else if (flight.avg_rateofdescent_750ftAGL < baseline_ROD_750ftAGL
- AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_750ftAGL =
STATECATEGORY_avgROD_high;
            else
                  flight.avg_rateofdescent_category_750ftAGL =
STATECATEGORY_avgROD_normal;
      }
      else if (dist == "500ftAGL") {
            flight.lateral_deviation_500ftAGL = lateral_deviation;
            flight.vertical_deviation_500ftAGL = vertical_deviation;
            flight.avg_groundspeed_500ftAGL = avg_groundspeed;
            flight.avg_airspeed_500ftAGL = avg_airspeed;
            flight.avg_rateofdescent_500ftAGL = avg_rateofdescent;
            //lateral deviation category
            if (flight.lateral_deviation_500ftAGL < -(LATERALHALFWIDTH_THR +
(Distance/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR)))
                  flight.lateral_deviation_category_500ftAGL =
STATECATEGORY_lateraldev_left;
            else if (flight.lateral_deviation_500ftAGL > LATERALHALFWIDTH_THR
+ (Distance/REF2THR_DIST)*(LATERALHALFWIDTH_REF-LATERALHALFWIDTH_THR))
                  flight.lateral_deviation_category_500ftAGL =
STATECATEGORY_lateraldev_right;
            else
                  flight.lateral_deviation_category_500ftAGL =
STATECATEGORY_lateraldev_normal;
            //vertical deviation category
            if (flight.vertical_deviation_500ftAGL < -(VERTICALHALFWIDTH_THR +
(Distance/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-VERTICALHALFWIDTH_THR)))
                  flight.vertical_deviation_category_500ftAGL =
STATECATEGORY_verticaldev_low;
            else if (flight.vertical_deviation_500ftAGL >
VERTICALHALFWIDTH_THR + (Distance/REF2THR_DIST)*(VERTICALHALFWIDTH_REF-
VERTICALHALFWIDTH_THR))
                  flight.vertical_deviation_category_500ftAGL =
STATECATEGORY_verticaldev_high;
            else
                  flight.vertical_deviation_category_500ftAGL =
STATECATEGORY_verticaldev_normal;
            //avg_airspeed category
            if (flight.avg_airspeed_500ftAGL < baseline_v_500ftAGL -
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_500ftAGL =
STATECATEGORY_avgspeed_low;
            else if (flight.avg_airspeed_500ftAGL > baseline_v_500ftAGL +
AVG_AIRSPEED_BOUNDARY)
                  flight.avg_airspeedcategory_500ftAGL =
STATECATEGORY_avgspeed_high;
            else
                  flight.avg_airspeedcategory_500ftAGL =
STATECATEGORY_avgspeed_normal;
            //avg_ROD category
            if (flight.avg_rateofdescent_500ftAGL > baseline_ROD_500ftAGL +
AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_500ftAGL =
STATECATEGORY_avgROD_low;
            else if (flight.avg_rateofdescent_500ftAGL < baseline_ROD_500ftAGL
- AVG_ROD_BOUNDARY)
                  flight.avg_rateofdescent_category_500ftAGL =
STATECATEGORY_avgROD_high;
```

```
			else
				flight.avg_rateofdescent_category_500ftAGL =
STATECATEGORY_avgROD_normal;
		}
}
void find_n_entrance_before_xnm(FLIGHT &flight) {
		for (int j = 2; j <= flight.j_10nm_r; ++j) {
			Point PPP;
			PPP.lat = flight.Track[j-1].lat;
			PPP.lng = flight.Track[j-1].lng;
			PPP.x = flight.Track[j-1].x;
			PPP.y = flight.Track[j-1].y;
			PPP.z = flight.Track[j-1].z;
			Point PPP_prev;
			PPP_prev.lat = flight.Track[(j-1)-1].lat;
			PPP_prev.lng = flight.Track[(j-1)-1].lng;
			PPP_prev.x = flight.Track[(j-1)-1].x;
			PPP_prev.y = flight.Track[(j-1)-1].y;
			PPP_prev.z = flight.Track[(j-1)-1].z;
			bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
			bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
			bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
			bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
			bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
			bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

			bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
			if (entered == true)
				flight.n_entrance_before_10nm++;
		}
		for (int j = 2; j <= flight.j_9nm_r; ++j) {
			Point PPP;
```

```
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_9nm++;
      }
      for (int j = 2; j <= flight.j_8nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
```

```
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_8nm++;
      }
      for (int j = 2; j <= flight.j_7nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
```

```
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_7nm++;
      }
      for (int j = 2; j <= flight.j_6nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
```

```
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_6nm++;
      }
      for (int j = 2; j <= flight.j_5dot5nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
```

```
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_5dot5nm++;
      }
      for (int j = 2; j <= flight.j_5nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
```

```
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_5nm++;
      }
      for (int j = 2; j <= flight.j_4dot5nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
```

```
                flight.n_entrance_before_4dot5nm++;
        }
        for (int j = 2; j <= flight.j_4nm_r; ++j) {
                Point PPP;
                PPP.lat = flight.Track[j-1].lat;
                PPP.lng = flight.Track[j-1].lng;
                PPP.x = flight.Track[j-1].x;
                PPP.y = flight.Track[j-1].y;
                PPP.z = flight.Track[j-1].z;
                Point PPP_prev;
                PPP_prev.lat = flight.Track[(j-1)-1].lat;
                PPP_prev.lng = flight.Track[(j-1)-1].lng;
                PPP_prev.x = flight.Track[(j-1)-1].x;
                PPP_prev.y = flight.Track[(j-1)-1].y;
                PPP_prev.z = flight.Track[(j-1)-1].z;
                bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
                bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
                bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
                bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
                bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
                bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

                bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
                if (entered == true)
                        flight.n_entrance_before_4nm++;
        }
        for (int j = 2; j <= flight.j_3dot5nm_r; ++j) {
                Point PPP;
                PPP.lat = flight.Track[j-1].lat;
                PPP.lng = flight.Track[j-1].lng;
                PPP.x = flight.Track[j-1].x;
                PPP.y = flight.Track[j-1].y;
```

```
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_3dot5nm++;
      }
      for (int j = 2; j <= flight.j_3nm_r; ++j) {
            Point PPP;
            PPP.lat = flight.Track[j-1].lat;
            PPP.lng = flight.Track[j-1].lng;
            PPP.x = flight.Track[j-1].x;
            PPP.y = flight.Track[j-1].y;
            PPP.z = flight.Track[j-1].z;
            Point PPP_prev;
            PPP_prev.lat = flight.Track[(j-1)-1].lat;
            PPP_prev.lng = flight.Track[(j-1)-1].lng;
            PPP_prev.x = flight.Track[(j-1)-1].x;
            PPP_prev.y = flight.Track[(j-1)-1].y;
            PPP_prev.z = flight.Track[(j-1)-1].z;
```

```
            bool inside_REFtoTHR_zone = inbox(PPP,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone = inbox(PPP,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone = inbox(PPP,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);
            bool inside_REFtoTHR_zone_prev = inbox(PPP_prev,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.THR_leftup, flight.landingrunway.THR_rightup,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown,
flight.landingrunway.THR_leftdown, flight.landingrunway.THR_rightdown);
            bool inside_FAFtoREF_zone_prev = inbox(PPP_prev,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.REF_leftup, flight.landingrunway.REF_rightup,
flight.landingrunway.FAF_leftdown, flight.landingrunway.FAF_rightdown,
flight.landingrunway.REF_leftdown, flight.landingrunway.REF_rightdown);
            bool inside_12nmtoFAF_zone_prev = inbox(PPP_prev,
flight.landingrunway.LOC_12nm_leftup, flight.landingrunway.LOC_12nm_rightup,
flight.landingrunway.FAF_leftup, flight.landingrunway.FAF_rightup,
flight.landingrunway.LOC_12nm_leftdown,
flight.landingrunway.LOC_12nm_rightdown, flight.landingrunway.FAF_leftdown,
flight.landingrunway.FAF_rightdown);

            bool entered = (inside_REFtoTHR_zone || inside_FAFtoREF_zone ||
inside_12nmtoFAF_zone) && (!inside_REFtoTHR_zone_prev) &&
(!inside_FAFtoREF_zone_prev) && (!inside_12nmtoFAF_zone_prev);
            if (entered == true)
                  flight.n_entrance_before_3nm++;
      }
}
void find_level_segment_flight_time(FLIGHT &flight) {
      int j_12nmr = j_default;
      int j_10nmr = j_default;
      for (int j = flight.Track.size(); j >= 1; j=j-1) {
            if (distance_2D(flight.Track[j-1], flight.landingrunway.THR) >
12.0*nm2m) {
                  j_12nmr = j;
                  break;
            }
      }
      for (int j = flight.Track.size(); j >= 1; j=j-1) {
            if (distance_2D(flight.Track[j-1], flight.landingrunway.THR) >
10.0*nm2m) {
                  j_10nmr = j;
                  break;
            }
      }
      double lvltime = 0.0;
      for (int j = j_12nmr; j <= j_10nmr; j++) {
```

```
            if (flight.Track[j-1].z - flight.landingrunway.THR.z >=
LEVEL_FLIGHT_ALTITUDE - 300.0*ft2m && flight.Track[j-1].z -
flight.landingrunway.THR.z <= LEVEL_FLIGHT_ALTITUDE + 300.0*ft2m) { //hardcoded
                  lvltime = lvltime + TIMESTEP;
            }
      }
      flight.levelsegmentflighttime = lvltime;
}
void prediction_lookuptable(vector<FLIGHT> *p_Flights, string currentlocation,
string currentaircraftweightclass) {
      ofstream out("../Data/[output] Lookup table for prediction_" +
currentlocation + "_" + currentaircraftweightclass + ".csv");
      vector<string> lateral_dev_category;
      vector<string> vertical_dev_category;
      vector<string> avg_v_category;
      vector<string> avg_ROD_category;
      if (currentlocation == "10nm") {
            for (int i = 1; i <= (*p_Flights).size(); ++i) {
                  lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_10nm);
                  vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_10nm);
                  avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_10nm);
                  avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_10nm);
            }
      }
      if (currentlocation == "9nm") {
            for (int i = 1; i <= (*p_Flights).size(); ++i) {
                  lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_9nm);
                  vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_9nm);
                  avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_9nm);
                  avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_9nm);
            }
      }
      if (currentlocation == "8nm") {
            for (int i = 1; i <= (*p_Flights).size(); ++i) {
                  lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_8nm);
                  vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_8nm);
                  avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_8nm);
                  avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_8nm);
            }
      }
      if (currentlocation == "7nm") {
            for (int i = 1; i <= (*p_Flights).size(); ++i) {
                  lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_7nm);
                  vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_7nm);
                  avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_7nm);
```

```
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_7nm);
             }
       }
       if (currentlocation == "6nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_6nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_6nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_6nm);
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_6nm);
             }
       }
       if (currentlocation == "5dot5nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_5dot5nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_5dot5nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_5dot5nm);
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_5dot5nm);
             }
       }
       if (currentlocation == "5nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_5nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_5nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_5nm);
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_5nm);
             }
       }
       if (currentlocation == "4dot5nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_4dot5nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_4dot5nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_4dot5nm);
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_4dot5nm);
             }
       }
       if (currentlocation == "4nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_4nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_4nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_4nm);
```

```
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_4nm);
             }
       }
       if (currentlocation == "3dot5nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_3dot5nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_3dot5nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_3dot5nm);
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_3dot5nm);
             }
       }
       if (currentlocation == "3nm") {
             for (int i = 1; i <= (*p_Flights).size(); ++i) {
                    lateral_dev_category.push_back((*p_Flights)[i-
1].lateral_deviation_category_3nm);
                    vertical_dev_category.push_back((*p_Flights)[i-
1].vertical_deviation_category_3nm);
                    avg_v_category.push_back((*p_Flights)[i-
1].avg_airspeedcategory_3nm);
                    avg_ROD_category.push_back((*p_Flights)[i-
1].avg_rateofdescent_category_3nm);
             }
       }
       vector<string> categoryset_lateraldeviation;
       categoryset_lateraldeviation.push_back(STATECATEGORY_lateraldev_left);
       categoryset_lateraldeviation.push_back(STATECATEGORY_lateraldev_normal);
       categoryset_lateraldeviation.push_back(STATECATEGORY_lateraldev_right);
       vector<string> categoryset_verticaldeviation;
       categoryset_verticaldeviation.push_back(STATECATEGORY_verticaldev_low);
       categoryset_verticaldeviation.push_back(STATECATEGORY_verticaldev_normal)
;
       categoryset_verticaldeviation.push_back(STATECATEGORY_verticaldev_high);
       vector<string> categoryset_avgspeed;
       categoryset_avgspeed.push_back(STATECATEGORY_avgspeed_low);
       categoryset_avgspeed.push_back(STATECATEGORY_avgspeed_normal);
       categoryset_avgspeed.push_back(STATECATEGORY_avgspeed_high);
       vector<string> categoryset_avgROD;
       categoryset_avgROD.push_back(STATECATEGORY_avgROD_low);
       categoryset_avgROD.push_back(STATECATEGORY_avgROD_normal);
       categoryset_avgROD.push_back(STATECATEGORY_avgROD_high);
       out << "Current location: " + currentlocation + "\n";
       out << "Current aircraft weight class: " + currentaircraftweightclass +
"\n";
       out << "lat_dev,vert_dev,avg_v,avg_ROD,n_sample,"
             <<
"n_unstable1000,p_unstable1000,(half)CI_unstable1000,n_USD1000,p_USD1000,(half)
CI_USD1000,n_USA1000,p_USA1000,(half)CI_USA1000,n_ERD1000,p_ERD1000,(half)CI_ER
D1000,n_SAL1000,p_SAL1000,(half)CI_SAL1000,n_SAR1000,p_SAR1000,(half)CI_SAR1000
,n_SAA1000,p_SAA1000,(half)CI_SAA1000,n_SAB1000,p_SAB1000,(half)CI_SAB1000,"
             <<
"n_unstable750,p_unstable750,(half)CI_unstable750,n_USD750,p_USD750,(half)CI_US
D750,n_USA750,p_USA750,(half)CI_USA750,n_ERD750,p_ERD750,(half)CI_ERD750,n_SAL7
50,p_SAL750,(half)CI_SAL750,n_SAR750,p_SAR750,(half)CI_SAR750,n_SAA750,p_SAA750
,(half)CI_SAA750,n_SAB750,p_SAB750,(half)CI_SAB750,"
```

```
            <<
"n_unstable500,p_unstable500,(half)CI_unstable500,n_USD500,p_USD500,(half)CI_US
D500,n_USA500,p_USA500,(half)CI_USA500,n_ERD500,p_ERD500,(half)CI_ERD500,n_SAL5
00,p_SAL500,(half)CI_SAL500,n_SAR500,p_SAR500,(half)CI_SAR500,n_SAA500,p_SAA500
,(half)CI_SAA500,n_SAB500,p_SAB500,(half)CI_SAB500,"
            << "\n";
      for (int k1 = 1; k1 <= categoryset_lateraldeviation.size(); ++k1) {
            for (int k2 = 1; k2 <= categoryset_verticaldeviation.size(); ++k2)
{
                  for (int k3 = 1; k3 <= categoryset_avgspeed.size(); ++k3) {
                        for (int k4 = 1; k4 <= categoryset_avgROD.size();
++k4) {
                              int n_sample = 0;
                              int n_unstable1000 = 0;
                              int n_USD1000 = 0;
                              int n_USA1000 = 0;
                              int n_ERD1000 = 0;
                              int n_SAL1000 = 0;
                              int n_SAR1000 = 0;
                              int n_SAA1000 = 0;
                              int n_SAB1000 = 0;
                              int n_unstable750 = 0;
                              int n_USD750 = 0;
                              int n_USA750 = 0;
                              int n_ERD750 = 0;
                              int n_SAL750 = 0;
                              int n_SAR750 = 0;
                              int n_SAA750 = 0;
                              int n_SAB750 = 0;
                              int n_unstable500 = 0;
                              int n_USD500 = 0;
                              int n_USA500 = 0;
                              int n_ERD500 = 0;
                              int n_SAL500 = 0;
                              int n_SAR500 = 0;
                              int n_SAA500 = 0;
                              int n_SAB500 = 0;
                              for (int i = 1; i <= (*p_Flights).size(); ++i)
{
                                    if ((*p_Flights)[i-
1].aircraft.weight_class == currentaircraftweightclass &&
                                          lateral_dev_category[i-1] ==
categoryset_lateraldeviation[k1-1] &&
                                          vertical_dev_category[i-1] ==
categoryset_verticaldeviation[k2-1] &&
                                          avg_v_category[i-1] ==
categoryset_avgspeed[k3-1] &&
                                          avg_ROD_category[i-1] ==
categoryset_avgROD[k4-1]) {
                                          n_sample++;
                                          if ((*p_Flights)[i-
1].unstable1000) n_unstable1000++;
                                          if ((*p_Flights)[i-1].USD1000)
n_USD1000++;
                                          if ((*p_Flights)[i-1].USA1000)
n_USA1000++;
                                          if ((*p_Flights)[i-1].ERD1000)
n_ERD1000++;
                                          if ((*p_Flights)[i-1].SAL1000)
n_SAL1000++;
```

```
                                                if ((*p_Flights)[i-1].SAR1000)
n_SAR1000++;
                                                if ((*p_Flights)[i-1].SAA1000)
n_SAA1000++;
                                                if ((*p_Flights)[i-1].SAB1000)
n_SAB1000++;
                                                if ((*p_Flights)[i-
1].unstable750) n_unstable750++;
                                                if ((*p_Flights)[i-1].USD750)
n_USD750++;
                                                if ((*p_Flights)[i-1].USA750)
n_USA750++;
                                                if ((*p_Flights)[i-1].ERD750)
n_ERD750++;
                                                if ((*p_Flights)[i-1].SAL750)
n_SAL750++;
                                                if ((*p_Flights)[i-1].SAR750)
n_SAR750++;
                                                if ((*p_Flights)[i-1].SAA750)
n_SAA750++;
                                                if ((*p_Flights)[i-1].SAB750)
n_SAB750++;
                                                if ((*p_Flights)[i-
1].unstable500) n_unstable500++;
                                                if ((*p_Flights)[i-1].USD500)
n_USD500++;
                                                if ((*p_Flights)[i-1].USA500)
n_USA500++;
                                                if ((*p_Flights)[i-1].ERD500)
n_ERD500++;
                                                if ((*p_Flights)[i-1].SAL500)
n_SAL500++;
                                                if ((*p_Flights)[i-1].SAR500)
n_SAR500++;
                                                if ((*p_Flights)[i-1].SAA500)
n_SAA500++;
                                                if ((*p_Flights)[i-1].SAB500)
n_SAB500++;
                                        }
                                }
                                out
                                        << categoryset_lateraldeviation[k1-1]
<< ","
                                        << categoryset_verticaldeviation[k2-1]
<< ","
                                        << categoryset_avgspeed[k3-1] << ","
                                        << categoryset_avgROD[k4-1] << ","
                                        << n_sample <<",";
                                if (n_sample == 0)
                                        out <<
"N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/
A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N
/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,
N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,\n";
                                else {
                                        double p_unstable1000 =
(double)n_unstable1000/(double)n_sample;
                                        double p_unstable750 =
(double)n_unstable750/(double)n_sample;
```

```
                                        double p_unstable500 =
(double)n_unstable500/(double)n_sample;
                                        double p_USD1000 =
(double)n_USD1000/(double)n_sample;
                                        double p_USD750 =
(double)n_USD750/(double)n_sample;
                                        double p_USD500 =
(double)n_USD500/(double)n_sample;
                                        double p_USA1000 =
(double)n_USA1000/(double)n_sample;
                                        double p_USA750 =
(double)n_USA750/(double)n_sample;
                                        double p_USA500 =
(double)n_USA500/(double)n_sample;
                                        double p_ERD1000 =
(double)n_ERD1000/(double)n_sample;
                                        double p_ERD750 =
(double)n_ERD750/(double)n_sample;
                                        double p_ERD500 =
(double)n_ERD500/(double)n_sample;
                                        double p_SAL1000 =
(double)n_SAL1000/(double)n_sample;
                                        double p_SAL750 =
(double)n_SAL750/(double)n_sample;
                                        double p_SAL500 =
(double)n_SAL500/(double)n_sample;
                                        double p_SAR1000 =
(double)n_SAR1000/(double)n_sample;
                                        double p_SAR750 =
(double)n_SAR750/(double)n_sample;
                                        double p_SAR500 =
(double)n_SAR500/(double)n_sample;
                                        double p_SAA1000 =
(double)n_SAA1000/(double)n_sample;
                                        double p_SAA750 =
(double)n_SAA750/(double)n_sample;
                                        double p_SAA500 =
(double)n_SAA500/(double)n_sample;
                                        double p_SAB1000 =
(double)n_SAB1000/(double)n_sample;
                                        double p_SAB750 =
(double)n_SAB750/(double)n_sample;
                                        double p_SAB500 =
(double)n_SAB500/(double)n_sample;
                                        out //half width of 95% CI assuming
normal distribution
                                              << n_unstable1000 << "," <<
100.0*p_unstable1000 << "%," << 100.0 * 1.96 * sqrt(p_unstable1000 * (1.0 -
p_unstable1000)/(double)n_sample) << "%,"
                                              << n_USD1000 << ","<<
100.0*p_USD1000 << "%," << 100.0 * 1.96 * sqrt(p_USD1000 * (1.0 -
p_USD1000)/(double)n_sample) << "%,"
                                              << n_USA1000 << ","<<
100.0*p_USA1000 << "%," << 100.0 * 1.96 * sqrt(p_USA1000 * (1.0 -
p_USA1000)/(double)n_sample) << "%,"
                                              << n_ERD1000 << "," <<
100.0*p_ERD1000 << "%," << 100.0 * 1.96 * sqrt(p_ERD1000 * (1.0 -
p_ERD1000)/(double)n_sample) << "%,"
```

```
                                                << n_SAL1000 << "," <<
100.0*p_SAL1000 << "%," << 100.0 * 1.96 * sqrt(p_SAL1000 * (1.0 -
p_SAL1000)/(double)n_sample) << "%,"
                                                << n_SAR1000 << "," <<
100.0*p_SAR1000 << "%," << 100.0 * 1.96 * sqrt(p_SAR1000 * (1.0 -
p_SAR1000)/(double)n_sample) << "%,"
                                                << n_SAA1000 << "," <<
100.0*p_SAA1000 << "%," << 100.0 * 1.96 * sqrt(p_SAA1000 * (1.0 -
p_SAA1000)/(double)n_sample) << "%,"
                                                << n_SAB1000 << "," <<
100.0*p_SAB1000 << "%," << 100.0 * 1.96 * sqrt(p_SAB1000 * (1.0 -
p_SAB1000)/(double)n_sample) << "%,"
                                                << n_unstable750 << "," <<
100.0*p_unstable750 << "%," << 100.0 * 1.96 * sqrt(p_unstable750 * (1.0 -
p_unstable750)/(double)n_sample) << "%,"
                                                << n_USD750 << "," <<
100.0*p_USD750 << "%," << 100.0 * 1.96 * sqrt(p_USD750 * (1.0 -
p_USD750)/(double)n_sample) << "%,"
                                                << n_USA750 << "," <<
100.0*p_USA750 << "%," << 100.0 * 1.96 * sqrt(p_USA750 * (1.0 -
p_USA750)/(double)n_sample) << "%,"
                                                << n_ERD750 << "," <<
100.0*p_ERD750 << "%," << 100.0 * 1.96 * sqrt(p_ERD750 * (1.0 -
p_ERD750)/(double)n_sample) << "%,"
                                                << n_SAL750 << "," <<
100.0*p_SAL750 << "%," << 100.0 * 1.96 * sqrt(p_SAL750 * (1.0 -
p_SAL750)/(double)n_sample) << "%,"
                                                << n_SAR750 << "," <<
100.0*p_SAR750 << "%," << 100.0 * 1.96 * sqrt(p_SAR750 * (1.0 -
p_SAR750)/(double)n_sample) << "%,"
                                                << n_SAA750 << "," <<
100.0*p_SAA750 << "%," << 100.0 * 1.96 * sqrt(p_SAA750 * (1.0 -
p_SAA750)/(double)n_sample) << "%,"
                                                << n_SAB750 << "," <<
100.0*p_SAB750 << "%," << 100.0 * 1.96 * sqrt(p_SAB750 * (1.0 -
p_SAB750)/(double)n_sample) << "%,"
                                                << n_unstable500 << "," <<
100.0*p_unstable500 << "%," << 100.0 * 1.96 * sqrt(p_unstable500 * (1.0 -
p_unstable500)/(double)n_sample) << "%,"

                                                << n_USD500 << "," <<
100.0*p_USD500 << "%," << 100.0 * 1.96 * sqrt(p_USD500 * (1.0 -
p_USD500)/(double)n_sample) << "%,"
                                                << n_USA500 << "," <<
100.0*p_USA500 << "%," << 100.0 * 1.96 * sqrt(p_USA500 * (1.0 -
p_USA500)/(double)n_sample) << "%,"
                                                << n_ERD500 << "," <<
100.0*p_ERD500 << "%," << 100.0 * 1.96 * sqrt(p_ERD500 * (1.0 -
p_ERD500)/(double)n_sample) << "%,"
                                                << n_SAL500 << "," <<
100.0*p_SAL500 << "%," << 100.0 * 1.96 * sqrt(p_SAL500 * (1.0 -
p_SAL500)/(double)n_sample) << "%,"
                                                << n_SAR500 << "," <<
100.0*p_SAR500 << "%," << 100.0 * 1.96 * sqrt(p_SAR500 * (1.0 -
p_SAR500)/(double)n_sample) << "%,"
                                                << n_SAA500 << "," <<
100.0*p_SAA500 << "%," << 100.0 * 1.96 * sqrt(p_SAA500 * (1.0 -
p_SAA500)/(double)n_sample) << "%,"
```

```
                                                    << n_SAB500 << "," <<
100.0*p_SAB500 << "%," << 100.0 * 1.96 * sqrt(p_SAB500 * (1.0 -
p_SAB500)/(double)n_sample) << "%,"
                                                    << "\n";
                                    }
                                }
                            }
                    }
            }
      }
      out.close();
}
void get_rel_dist_from_acq_to_predictionlocations(FLIGHT &flight) {
      vector<double> locations;
      locations.push_back(10*nm2m);
      locations.push_back(9*nm2m);
      locations.push_back(8*nm2m);
      locations.push_back(7*nm2m);
      locations.push_back(6*nm2m);
      locations.push_back(5.5*nm2m);
      locations.push_back(5*nm2m);
      locations.push_back(4.5*nm2m);
      locations.push_back(4*nm2m);
      locations.push_back(3.5*nm2m);
      locations.push_back(3*nm2m);
      vector<double> rel_lat_dists;
      vector<double> rel_vert_dists;
      for (int k = 1; k <= locations.size(); ++k) {
            rel_lat_dists.push_back(maxof(0.0,
flight.dist2THR_lateral_acquisition_prj - locations[k-1]));
            rel_vert_dists.push_back(maxof(0.0,
flight.dist2THR_vertical_acquisition_prj - locations[k-1]));
      }
      flight.rel_dist_to_10nm_lateral_acquisition_prj = rel_lat_dists[0];
      flight.rel_dist_to_9nm_lateral_acquisition_prj = rel_lat_dists[1];
      flight.rel_dist_to_8nm_lateral_acquisition_prj = rel_lat_dists[2];
      flight.rel_dist_to_7nm_lateral_acquisition_prj = rel_lat_dists[3];
      flight.rel_dist_to_6nm_lateral_acquisition_prj = rel_lat_dists[4];
      flight.rel_dist_to_5dot5nm_lateral_acquisition_prj = rel_lat_dists[5];
      flight.rel_dist_to_5nm_lateral_acquisition_prj = rel_lat_dists[6];
      flight.rel_dist_to_4dot5nm_lateral_acquisition_prj = rel_lat_dists[7];
      flight.rel_dist_to_4nm_lateral_acquisition_prj = rel_lat_dists[8];
      flight.rel_dist_to_3dot5nm_lateral_acquisition_prj = rel_lat_dists[9];
      flight.rel_dist_to_3nm_lateral_acquisition_prj = rel_lat_dists[10];
      flight.rel_dist_to_10nm_vertical_acquisition_prj = rel_vert_dists[0];
      flight.rel_dist_to_9nm_vertical_acquisition_prj = rel_vert_dists[1];
      flight.rel_dist_to_8nm_vertical_acquisition_prj = rel_vert_dists[2];
      flight.rel_dist_to_7nm_vertical_acquisition_prj = rel_vert_dists[3];
      flight.rel_dist_to_6nm_vertical_acquisition_prj = rel_vert_dists[4];
      flight.rel_dist_to_5dot5nm_vertical_acquisition_prj = rel_vert_dists[5];
      flight.rel_dist_to_5nm_vertical_acquisition_prj = rel_vert_dists[6];
      flight.rel_dist_to_4dot5nm_vertical_acquisition_prj = rel_vert_dists[7];
      flight.rel_dist_to_4nm_vertical_acquisition_prj = rel_vert_dists[8];
      flight.rel_dist_to_3dot5nm_vertical_acquisition_prj = rel_vert_dists[9];
      flight.rel_dist_to_3nm_vertical_acquisition_prj = rel_vert_dists[10];
}
void get_deceleration_point_and_relative_distance(FLIGHT &flight) {
      int j_12nm = j_default;
      for (int j = flight.Track.size(); j >= 1; j = j - 1) {
            if (distance_2D(flight.Track[j-1], flight.landingrunway.THR) >
12.0 * nm2m) {
```

```
                        j_12nm = j;
                        break;
                }
        }
        double currentspeed = 0.0;
        for (int j = j_12nm; j <= flight.Track.size(); ++j) {
                currentspeed = get_avg_airspeed(AVGBIN, flight, j, PROJECTING);
                if (currentspeed < flight.v_12nm - 10.0*kts2mps) {
                        flight.j_deceleration = j;
                        break;
                }
        }
        if (flight.j_deceleration == j_default) {
                flight.j_deceleration = flight.Track.size();
        }
        flight.dist_deceleration =
distance_2D(flight.Track[flight.j_deceleration-1], flight.landingrunway.THR);
}
void process(string currentrunway, vector<FLIGHT> *p_Flights) {
        for (int i = 1; i <= (*p_Flights).size(); ++i) {
                if ((*p_Flights)[i-1].landingrunway.name != currentrunway) {
                        cout << i << "th flight " << (*p_Flights)[i-1].trackindex
<< " did not land on " + currentrunway << "\n";
                        continue;
                }
                cout << "Processing flight " << (*p_Flights)[i-1].flightID <<",
"<< (*p_Flights)[i-1].trackindex << " at runway " << (*p_Flights)[i-
1].landingrunway.name << "\n";
        //determine wireframe-entrance, j_firstentrance, j_beforeentrance
                determine_wireframe_entrance_related((*p_Flights)[i-1]);
        //determine go-around
                if ((*p_Flights)[i-1].Track.front().z - (*p_Flights)[i-
1].landingrunway.elevation > starting_min_altitude_AGL
                        && (*p_Flights)[i-1].Track.back().z - (*p_Flights)[i-
1].landingrunway.elevation < ending_max_altitude_AGL
                        && distance_2D((*p_Flights)[i-1].Track.front(),
(*p_Flights)[i-1].landingrunway.THR) > starting_min_dist_to_THR
                        && distance_2D((*p_Flights)[i-1].Track.back(),
(*p_Flights)[i-1].landingrunway.THR) < ending_max_dist_to_THR) {
                        (*p_Flights)[i-1].go_around =
determine_GoAround((*p_Flights)[i-1]);
                }
        //determine whether current flight has qualified track according to the
criteria
                (*p_Flights)[i-1].track_qualified =
track_qualified((*p_Flights)[i-1]);
                if ((*p_Flights)[i-1].track_qualified) {
        //find key track points
                        find_trackpoints_at_key_locations((*p_Flights)[i-1]);
                        find_j_altitude((*p_Flights)[i-1], (*p_Flights)[i-
1].j_1000ftAGL, 1000*ft2m, (*p_Flights)[i-1].j_3nm);
                        find_j_altitude((*p_Flights)[i-1], (*p_Flights)[i-
1].j_750ftAGL, 750*ft2m, (*p_Flights)[i-1].j_2dot25nm);
                        find_j_altitude((*p_Flights)[i-1], (*p_Flights)[i-
1].j_500ftAGL, 500*ft2m, (*p_Flights)[i-1].j_1dot5nm);
                        find_lateral_vertical_acquisition_point((*p_Flights)[i-1]);
//find the lateral/vertical acquisition point
        //find number of entering wireframes before each prediciton locations (as
an additional feature for prediction)
                        find_n_entrance_before_xnm((*p_Flights)[i-1]);
```

```
	//find level segment flight time
			find_level_segment_flight_time((*p_Flights)[i-1]);
	//aircraft state variables
		//at radii of 10~3nm from runway threshold
			get_aircraft_states((*p_Flights)[i-1], "10nm");
			get_aircraft_states((*p_Flights)[i-1], "9nm");
			get_aircraft_states((*p_Flights)[i-1], "8nm");
			get_aircraft_states((*p_Flights)[i-1], "7nm");
			get_aircraft_states((*p_Flights)[i-1], "6nm");
			get_aircraft_states((*p_Flights)[i-1], "5.5nm");
			get_aircraft_states((*p_Flights)[i-1], "5nm");
			get_aircraft_states((*p_Flights)[i-1], "4.5nm");
			get_aircraft_states((*p_Flights)[i-1], "4nm");
			get_aircraft_states((*p_Flights)[i-1], "3.5nm");
			get_aircraft_states((*p_Flights)[i-1], "3nm");
		//at 1000'/750'/500' AGL
			get_aircraft_states((*p_Flights)[i-1], "1000ftAGL");
			get_aircraft_states((*p_Flights)[i-1], "750ftAGL");
			get_aircraft_states((*p_Flights)[i-1], "500ftAGL");
	//determine the projected 2D distance from lateral/vertical acquisition
point to landing runway threshold
			double templatacq = distance_2D((*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_acquisition_lateral-1].x, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_acquisition_lateral-1].y, (*p_Flights)[i-
1].landingrunway.THR.x, (*p_Flights)[i-1].landingrunway.THR.y);
			double tempvertacq = distance_2D((*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_acquisition_vertical-1].x, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_acquisition_vertical-1].y, (*p_Flights)[i-
1].landingrunway.THR.x, (*p_Flights)[i-1].landingrunway.THR.y);
			double dd_lat = dist_point2line_2D((*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_acquisition_lateral-1], (*p_Flights)[i-
1].landingrunway.THR, (*p_Flights)[i-1].landingrunway.FAF);
			double dd_vert = dist_point2line_2D((*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_acquisition_vertical-1], (*p_Flights)[i-
1].landingrunway.THR, (*p_Flights)[i-1].landingrunway.FAF);
			(*p_Flights)[i-1].dist2THR_lateral_acquisition_prj =
sqrt(templatacq*templatacq - dd_lat*dd_lat);
			(*p_Flights)[i-1].dist2THR_vertical_acquisition_prj =
sqrt(tempvertacq*tempvertacq - dd_vert*dd_vert);
	//determine relative prj_distance from lat/vert acquisition point to each
prediction location, =0 if not yet acquired by the location

	get_rel_dist_from_acq_to_predictionlocations((*p_Flights)[i-1]);
	//determine distance from it decelerates below some groundpspeed
threshold (e.g. 200 knots)

	get_deceleration_point_and_relative_distance((*p_Flights)[i-1]);
	//determine unstable events
		//calculate average ground speed at THR
			(*p_Flights)[i-1].avg_groundspeed_THR =
get_avg_groundspeed_THR(AVGBIN, (*p_Flights)[i-1], PROJECTING);
			(*p_Flights)[i-1].avg_airspeed_THR =
get_avg_airspeed_THR(AVGBIN, (*p_Flights)[i-1], PROJECTING);
		//determine unstable speed events
			if ((*p_Flights)[i-1].avg_airspeed_1000ftAGL -
(*p_Flights)[i-1].avg_airspeed_THR > unstablespeed_toofast_thresohld)
				(*p_Flights)[i-1].USD1000 = true;
			if ((*p_Flights)[i-1].avg_airspeed_750ftAGL -
(*p_Flights)[i-1].avg_airspeed_THR > unstablespeed_toofast_thresohld)
				(*p_Flights)[i-1].USD750 = true;
```

```
                    if ((*p_Flights)[i-1].avg_airspeed_500ftAGL -
(*p_Flights)[i-1].avg_airspeed_THR > unstablespeed_toofast_thresohld)
                           (*p_Flights)[i-1].USD500 = true;
                    if ((*p_Flights)[i-1].avg_airspeed_1000ftAGL -
(*p_Flights)[i-1].avg_airspeed_THR < unstablespeed_tooslow_thresohld)
                           (*p_Flights)[i-1].USA1000 = true;
                    if ((*p_Flights)[i-1].avg_airspeed_750ftAGL -
(*p_Flights)[i-1].avg_airspeed_THR < unstablespeed_tooslow_thresohld)
                           (*p_Flights)[i-1].USA750 = true;
                    if ((*p_Flights)[i-1].avg_airspeed_500ftAGL -
(*p_Flights)[i-1].avg_airspeed_THR < unstablespeed_tooslow_thresohld)
                           (*p_Flights)[i-1].USA500 = true;
             //determine excessive ROD risk event (average ROD remains greater
than 1000fpm for longer than the time period threshold)
                    (*p_Flights)[i-1].ERD1000 =
determine_ROD_event((*p_Flights)[i-1], (*p_Flights)[i-1].j_1000ftAGL);
                    (*p_Flights)[i-1].ERD750 =
determine_ROD_event((*p_Flights)[i-1], (*p_Flights)[i-1].j_750ftAGL);
                    (*p_Flights)[i-1].ERD500 =
determine_ROD_event((*p_Flights)[i-1], (*p_Flights)[i-1].j_500ftAGL);
             //determine lateral/vertical entrance (acquisition) method and
events

       determine_lateral_acquisition_method_and_event((*p_Flights)[i-1]);

       determine_vertical_acquisition_method_and_event((*p_Flights)[i-1]);
             //determine whether current flight is stable (i.e. none of the
sub-events occur)
                    if ((*p_Flights)[i-1].USD1000 || (*p_Flights)[i-1].USA1000
|| (*p_Flights)[i-1].ERD1000 || (*p_Flights)[i-1].SAA1000 || (*p_Flights)[i-
1].SAB1000 || (*p_Flights)[i-1].SAL1000 || (*p_Flights)[i-1].SAR1000) {
                           (*p_Flights)[i-1].stable1000 = false;
                           (*p_Flights)[i-1].unstable1000 = true;
                    }
                    else {
                           (*p_Flights)[i-1].stable1000 = true;
                           (*p_Flights)[i-1].unstable1000 = false;
                    }
                    if ((*p_Flights)[i-1].USD750 || (*p_Flights)[i-1].USA750 ||
(*p_Flights)[i-1].ERD750 || (*p_Flights)[i-1].SAA750 || (*p_Flights)[i-
1].SAB750 || (*p_Flights)[i-1].SAL750 || (*p_Flights)[i-1].SAR750) {
                           (*p_Flights)[i-1].stable750 = false;
                           (*p_Flights)[i-1].unstable750 = true;
                    }
                    else {
                           (*p_Flights)[i-1].stable750 = true;
                           (*p_Flights)[i-1].unstable750 = false;
                    }
                    if ((*p_Flights)[i-1].USD500 || (*p_Flights)[i-1].USA500 ||
(*p_Flights)[i-1].ERD500 || (*p_Flights)[i-1].SAA500 || (*p_Flights)[i-
1].SAB500 || (*p_Flights)[i-1].SAL500 || (*p_Flights)[i-1].SAR500) {
                           (*p_Flights)[i-1].stable500 = false;
                           (*p_Flights)[i-1].unstable500 = true;
                    }
                    else {
                           (*p_Flights)[i-1].stable500 = true;
                           (*p_Flights)[i-1].unstable500 = false;
                    }
```

```
            //temp: determine approach procedure and flow identification for
MDW 13C only
                  if (CURRENTAIRPORT == "MDW" && (*p_Flights)[i-
1].landingrunway.name == MDW13C.name) {
                        identify_ApproachProcedure((*p_Flights)[i-1]);
                        identify_Flow((*p_Flights)[i-1]);
                  }
            }
      }

/*output results*/
      ofstream out_summary;
      if (PROJECTING)
            out_summary.open("../Data/[output] Summary " + CURRENTDATA + " " +
CURRENTAIRPORT + currentrunway + " projecting = true" + ".csv");
      else
            out_summary.open("../Data/[output] Summary " + CURRENTDATA + " " +
CURRENTAIRPORT + currentrunway + ".csv");
      out_summary
                  << "date,trackindex,flight_ID,go-
around,qualified,time_to_threshold(s),runway,weight_class,lateral_acq_dist_prj(
nm),vertical_acq_dist_prj(nm),deceleration_dist(nm),"
                  <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_10nm(ft), lat_dev_abs(ft), vertical_deviation_10nm(ft),
vert_dev_abs(ft), avg_v_10nm(kts), v_dev(kts), avg_ROD_10nm(fpm), rod_dev(fpm),
heading_diff_10nm(deg),
n_entrance_before10nm,reldist_to_10nm_latacq(nm),reldist_to_10nm_vertacq(nm),re
ldist_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(
kts),"
                  <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_9nm(ft), lat_dev_abs(ft), vertical_deviation_9nm(ft),
vert_dev_abs(ft), avg_v_9nm(kts), v_dev(kts), avg_ROD_9nm(fpm), rod_dev(fpm),
heading_diff_9nm(deg),
n_entrance_before9nm,reldist_to_9nm_latacq(nm),reldist_to_9nm_vertacq(nm),reldi
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                  <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_8nm(ft), lat_dev_abs(ft), vertical_deviation_8nm(ft),
vert_dev_abs(ft), avg_v_8nm(kts), v_dev(kts), avg_ROD_8nm(fpm), rod_dev(fpm),
heading_diff_8nm(deg),
n_entrance_before8nm,reldist_to_8nm_latacq(nm),reldist_to_8nm_vertacq(nm),reldi
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                  <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_7nm(ft), lat_dev_abs(ft), vertical_deviation_7nm(ft),
vert_dev_abs(ft), avg_v_7nm(kts), v_dev(kts), avg_ROD_7nm(fpm), rod_dev(fpm),
heading_diff_7nm(deg),
n_entrance_before7nm,reldist_to_7nm_latacq(nm),reldist_to_7nm_vertacq(nm),reldi
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                  <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_6nm(ft), lat_dev_abs(ft), vertical_deviation_6nm(ft),
vert_dev_abs(ft), avg_v_6nm(kts), v_dev(kts), avg_ROD_6nm(fpm), rod_dev(fpm),
heading_diff_6nm(deg),
n_entrance_before6nm,reldist_to_6nm_latacq(nm),reldist_to_6nm_vertacq(nm),reldi
```

```
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                    <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_5.5nm(ft), lat_dev_abs(ft), vertical_deviation_5.5nm(ft),
vert_dev_abs(ft), avg_v_5.5nm(kts), v_dev(kts), avg_ROD_5.5nm(fpm),
rod_dev(fpm), heading_diff_5.5nm(deg),
n_entrance_before5.5nm,reldist_to_5.5nm_latacq(nm),reldist_to_5.5nm_vertacq(nm)
,reldist_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswi
nd(kts),"
                    <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_5nm(ft), lat_dev_abs(ft), vertical_deviation_5nm(ft),
vert_dev_abs(ft), avg_v_5nm(kts), v_dev(kts), avg_ROD_5nm(fpm), rod_dev(fpm),
heading_diff_5nm(deg),
n_entrance_before5nm,reldist_to_5nm_latacq(nm),reldist_to_5nm_vertacq(nm),reldi
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                    <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_4.5nm(ft), lat_dev_abs(ft), vertical_deviation_4.5nm(ft),
vert_dev_abs(ft), avg_v_4.5nm(kts), v_dev(kts), avg_ROD_4.5nm(fpm),
rod_dev(fpm), heading_diff_4.5nm(deg),
n_entrance_before4.5nm,reldist_to_4.5nm_latacq(nm),reldist_to_4.5nm_vertacq(nm)
,reldist_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswi
nd(kts),"
                    <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_4nm(ft), lat_dev_abs(ft), vertical_deviation_4nm(ft),
vert_dev_abs(ft), avg_v_4nm(kts), v_dev(kts), avg_ROD_4nm(fpm), rod_dev(fpm),
heading_diff_4nm(deg),
n_entrance_before4nm,reldist_to_4nm_latacq(nm),reldist_to_4nm_vertacq(nm),reldi
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                    <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_3.5nm(ft), lat_dev_abs(ft), vertical_deviation_3.5nm(ft),
vert_dev_abs(ft), avg_v_3.5nm(kts), v_dev(kts), avg_ROD_3.5nm(fpm),
rod_dev(fpm), heading_diff_3.5nm(deg),
n_entrance_before3.5nm,reldist_to_3.5nm_latacq(nm),reldist_to_3.5nm_vertacq(nm)
,reldist_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswi
nd(kts),"
                    <<
"MTOW(lb.),avg_v_12nm,avg_zAGL_12nm,headingdiff12nm,levelsegmenttime(s),lateral
_deviation_3nm(ft), lat_dev_abs(ft), vertical_deviation_3nm(ft),
vert_dev_abs(ft), avg_v_3nm(kts), v_dev(kts), avg_ROD_3nm(fpm), rod_dev(fpm),
heading_diff_3nm(deg),
n_entrance_before3nm,reldist_to_3nm_latacq(nm),reldist_to_3nm_vertacq(nm),reldi
st_to_deceleration(nm),crosswind_abs(kts),headwind(kts),gust(kts),crosswind(kts
),"
                    << "lateral_deviation_1000ftAGL(ft),
vertical_deviation_1000ftAGL(ft), avg_v_1000ftAGL(kts),
avg_ROD_1000ftAGL(fpm),"
                    << "lateral_deviation_750ftAGL(ft),
vertical_deviation_750ftAGL(ft), avg_v_750ftAGL(kts), avg_ROD_750ftAGL(fpm),"
                    << "lateral_deviation_500ftAGL(ft),
vertical_deviation_500ftAGL(ft), avg_v_500ftAGL(kts), avg_ROD_500ftAGL(fpm),"
                    << "lateral_deviation_category_10nm,
vertical_deviation_category_10nm,avg_v_category_10nm,avg_ROD_category_10nm,head
ing_diff_category_10nm,"
```

```
                    << "lateral_deviation_category_9nm,
vertical_deviation_category_9nm,avg_v_category_9nm,avg_ROD_category_9nm,heading
_diff_category_9nm,"
                    << "lateral_deviation_category_8nm,
vertical_deviation_category_8nm,avg_v_category_8nm,avg_ROD_category_8nm,heading
_diff_category_8nm,"
                    << "lateral_deviation_category_7nm,
vertical_deviation_category_7nm,avg_v_category_7nm,avg_ROD_category_7nm,heading
_diff_category_7nm,"
                    << "lateral_deviation_category_6nm,
vertical_deviation_category_6nm,avg_v_category_6nm,avg_ROD_category_6nm,heading
_diff_category_6nm,"
                    << "lateral_deviation_category_5dot5nm,
vertical_deviation_category_5dot5nm,avg_v_category_5dot5nm,avg_ROD_category_5do
t5nm,heading_diff_category_5dot5nm,"
                    << "lateral_deviation_category_5nm,
vertical_deviation_category_5nm,avg_v_category_5nm,avg_ROD_category_5nm,heading
_diff_category_5nm,"
                    << "lateral_deviation_category_4dot5nm,
vertical_deviation_category_4dot5nm,avg_v_category_4dot5nm,avg_ROD_category_4do
t5nm,heading_diff_category_4dot5nm,"
                    << "lateral_deviation_category_4nm,
vertical_deviation_category_4nm,avg_v_category_4nm,avg_ROD_category_4nm,heading
_diff_category_4nm,"
                    << "lateral_deviation_category_3dot5nm,
vertical_deviation_category_3dot5nm,avg_v_category_3dot5nm,avg_ROD_category_3do
t5nm,heading_diff_category_3dot5nm,"
                    << "lateral_deviation_category_3nm,
vertical_deviation_category_3nm,avg_v_category_3nm,avg_ROD_category_3nm,heading
_diff_category_3nm,"
                    << "lateral_deviation_category_1000ftAGL,
vertical_deviation_category_1000ftAGL,avg_v_category_1000ftAGL,avg_ROD_category
_1000ftAGL,"
                    << "lateral_deviation_category_750ftAGL,
vertical_deviation_category_750ftAGL,avg_v_category_750ftAGL,avg_ROD_category_7
50ftAGL,"
                    << "lateral_deviation_category_500ftAGL,
vertical_deviation_category_500ftAGL,avg_v_category_500ftAGL,avg_ROD_category_5
00ftAGL,"
                    <<
"unstable1000,USD1000,USA1000,ERD1000,SAA1000,SAL1000,SAR1000,"
                    << "unstable750,USD750,USA750,ERD750,SAA750,SAL750,SAR750,"
                    << "unstable500,USD500,USA500,ERD500,SAA500,SAL500,SAR500,"
                    << "\n";
      int n_tracks = 0;
      int n_qualifiedTracks = 0;
      int n_unstable1000 = 0;
      int n_unstable750 = 0;
      int n_unstable500 = 0;
      int n_USD1000 = 0;
      int n_USD750 = 0;
      int n_USD500 = 0;
      int n_USA1000 = 0;
      int n_USA750 = 0;
      int n_USA500 = 0;
      int n_ERD1000 = 0;
      int n_ERD750 = 0;
      int n_ERD500 = 0;
      int n_SAA1000 = 0;
      int n_SAA750 = 0;
```

```
	int n_SAA500 = 0;
	int n_SAB1000 = 0;
	int n_SAB750 = 0;
	int n_SAB500 = 0;
	int n_SAL1000 = 0;
	int n_SAL750 = 0;
	int n_SAL500 = 0;
	int n_SAR1000 = 0;
	int n_SAR750 = 0;
	int n_SAR500 = 0;
	int n_go_around = 0;
	int n_Small = 0;
	int n_Large = 0;
	int n_B757 = 0;
	int n_Heavy = 0;
	int n_Superheavy = 0;
	for (int i = 1; i <= (*p_Flights).size(); ++i) {
		if ((*p_Flights)[i-1].landingrunway.name != currentrunway)
continue;
		n_tracks++;
		if ((*p_Flights)[i-1].go_around == true)	n_go_around++;
		if ((*p_Flights)[i-1].track_qualified == true) {
			n_qualifiedTracks++;
			if ((*p_Flights)[i-1].aircraft.weight_class ==
"Superheavy") n_Superheavy ++;
			else if ((*p_Flights)[i-1].aircraft.weight_class ==
"Heavy") n_Heavy++;
			else if ((*p_Flights)[i-1].aircraft.weight_class == "B757")
n_B757++;
			else if ((*p_Flights)[i-1].aircraft.weight_class ==
"Large") n_Large++;
			else if ((*p_Flights)[i-1].aircraft.weight_class ==
"Small") n_Small++;
			if ((*p_Flights)[i-1].unstable1000) n_unstable1000++;
			if ((*p_Flights)[i-1].unstable750) n_unstable750++;
			if ((*p_Flights)[i-1].unstable500) n_unstable500++;
			if ((*p_Flights)[i-1].USD1000) n_USD1000++;
			if ((*p_Flights)[i-1].USD750) n_USD750++;
			if ((*p_Flights)[i-1].USD500) n_USD500++;
			if ((*p_Flights)[i-1].USA1000) n_USA1000++;
			if ((*p_Flights)[i-1].USA750) n_USA750++;
			if ((*p_Flights)[i-1].USA500) n_USA500++;
			if ((*p_Flights)[i-1].ERD1000) n_ERD1000++;
			if ((*p_Flights)[i-1].ERD750) n_ERD750++;
			if ((*p_Flights)[i-1].ERD500) n_ERD500++;
			if ((*p_Flights)[i-1].SAA1000) n_SAA1000++;
			if ((*p_Flights)[i-1].SAA750) n_SAA750++;
			if ((*p_Flights)[i-1].SAA500) n_SAA500++;
			if ((*p_Flights)[i-1].SAB1000) n_SAB1000++;
			if ((*p_Flights)[i-1].SAB750) n_SAB750++;
			if ((*p_Flights)[i-1].SAB500) n_SAB500++;
			if ((*p_Flights)[i-1].SAL1000) n_SAL1000++;
			if ((*p_Flights)[i-1].SAL750) n_SAL750++;
			if ((*p_Flights)[i-1].SAL500) n_SAL500++;
			if ((*p_Flights)[i-1].SAR1000) n_SAR1000++;
			if ((*p_Flights)[i-1].SAR750) n_SAR750++;
			if ((*p_Flights)[i-1].SAR500) n_SAR500++;

			out_summary
				<< (*p_Flights)[i-1].date << ","
```

```
                            << (*p_Flights)[i-1].trackindex << ","
                            << (*p_Flights)[i-1].flightID << ","
                            << (*p_Flights)[i-1].go_around << ","
                            << (*p_Flights)[i-1].track_qualified << ","
                            << (*p_Flights)[i-1].time2landing_entrance << ","
                            << (*p_Flights)[i-1].landingrunway.name << ","
                            << (*p_Flights)[i-1].aircraft.weight_class << ","
                            << (*p_Flights)[i-1].dist2THR_lateral_acquisition_prj
* m2nm << ","
                            << (*p_Flights)[i-
1].dist2THR_vertical_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-1].dist_deceleration * m2nm << ","
                     // state variables
                            //10nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_10nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_10nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_10nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_10nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_10nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_10nm * mps2kts << ","
//kts
                            << (*p_Flights)[i-1].avg_rateofdescent_10nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_10nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_10nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_10nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_10nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_10nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 10.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).gust << ","
```

```
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_10nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //9nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_9nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_9nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_9nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_9nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_9nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_9nm * mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_9nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_9nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_9nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_9nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_9nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_9nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 9.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_9nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //8nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
```

```
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_8nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_8nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_8nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_8nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_8nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_8nm * mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_8nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_8nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_8nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_8nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_8nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_8nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 8.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_8nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //7nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_7nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_7nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_7nm * m2ft<<
"," //ft
```

```
                            << (*p_Flights)[i-1].vertical_deviation_abs_7nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_7nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_7nm * mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_7nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_7nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_7nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_7nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_7nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_7nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 7.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_7nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //6nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_6nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_6nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_6nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_6nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_6nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_6nm * mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_6nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_6nm * mps2fpm << ","
//fpm
```

```
                            << (*p_Flights)[i-1].heading_diff_6nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_6nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_6nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_6nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 6.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_6nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //5.5nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_5dot5nm * m2ft
<< "," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_5dot5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_5dot5nm *
m2ft<< "," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_5dot5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_5dot5nm * mps2kts
<< "," //kts
                            << (*p_Flights)[i-1].v_dev_5dot5nm * mps2kts << ","
//kts
                            << (*p_Flights)[i-1].avg_rateofdescent_5dot5nm *
mps2fpm << "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_5dot5nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_5dot5nm * rad2deg
<< "," //degree
                            << (*p_Flights)[i-1].n_entrance_before_5dot5nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_5dot5nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_5dot5nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 5.5) << "," //nm, rel_dist
```

```
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_5dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //5nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_5nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_5nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_5nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_5nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_5nm * mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_5nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_5nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_5nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_5nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_5nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_5nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 5.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
```

```
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //4.5nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_4dot5nm * m2ft
<< "," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_4dot5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_4dot5nm *
m2ft<< "," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_4dot5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_4dot5nm * mps2kts
<< "," //kts
                            << (*p_Flights)[i-1].v_dev_4dot5nm * mps2kts << ","
//kts
                            << (*p_Flights)[i-1].avg_rateofdescent_4dot5nm *
mps2fpm << "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_4dot5nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_4dot5nm * rad2deg
<< "," //degree
                            << (*p_Flights)[i-1].n_entrance_before_4dot5nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_4dot5nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_4dot5nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 4.5) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_4dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //4nm
```

```
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_4nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_4nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_4nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_4nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_4nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_4nm * mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_4nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_4nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_4nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_4nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_4nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_4nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 4.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_4nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //3.5nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_3dot5nm * m2ft
<< "," //ft
```

```
                            << (*p_Flights)[i-1].lateral_deviation_abs_3dot5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_3dot5nm *
m2ft<< "," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_3dot5nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_3dot5nm * mps2kts
<< "," //kts
                            << (*p_Flights)[i-1].v_dev_3dot5nm * mps2kts << ","
//kts
                            << (*p_Flights)[i-1].avg_rateofdescent_3dot5nm *
mps2fpm << "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_3dot5nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_3dot5nm * rad2deg
<< "," //degree
                            << (*p_Flights)[i-1].n_entrance_before_3dot5nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_3dot5nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_3dot5nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 3.5) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_3dot5nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //3nm
                            << (*p_Flights)[i-1].aircraft.MTOW * kg2lb << ","
//lb
                            << (*p_Flights)[i-1].v_12nm * mps2kts << ","
                            << (*p_Flights)[i-1].z_AGL_12nm * m2ft << ","
                            << (*p_Flights)[i-1].headingdiff_12nm * rad2deg <<
","
                            << (*p_Flights)[i-1].levelsegmentflighttime << ","
//s
                            << (*p_Flights)[i-1].lateral_deviation_3nm * m2ft <<
"," //ft
                            << (*p_Flights)[i-1].lateral_deviation_abs_3nm * m2ft
<< "," //ft, absolute value
                            << (*p_Flights)[i-1].vertical_deviation_3nm * m2ft<<
"," //ft
                            << (*p_Flights)[i-1].vertical_deviation_abs_3nm *
m2ft << "," //ft, absolute value
                            << (*p_Flights)[i-1].avg_airspeed_3nm * mps2kts <<
"," //kts
                            << (*p_Flights)[i-1].v_dev_3nm * mps2kts << "," //kts
```

```
                            << (*p_Flights)[i-1].avg_rateofdescent_3nm * mps2fpm
<< "," // fpm, negative value...
                            << (*p_Flights)[i-1].rod_dev_3nm * mps2fpm << ","
//fpm
                            << (*p_Flights)[i-1].heading_diff_3nm * rad2deg <<
"," //degree
                            << (*p_Flights)[i-1].n_entrance_before_3nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_3nm_lateral_acquisition_prj * m2nm << ","
                            << (*p_Flights)[i-
1].rel_dist_to_3nm_vertical_acquisition_prj * m2nm << ","
                            << maxof(0.0, (*p_Flights)[i-1].dist_deceleration *
m2nm - 3.0) << "," //nm, rel_dist
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).speed *
cos(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment) << ","
                            << get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).gust << ","
                            << abs(get_current_wind((*p_Flights)[i-1].date,
(*p_Flights)[i-1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).speed *
sin(get_current_wind((*p_Flights)[i-1].date, (*p_Flights)[i-
1].Track[(*p_Flights)[i-1].j_3nm_r-1].t).heading - (*p_Flights)[i-
1].landingrunway.true_alignment)) << ","
                            //1000',750',500'
                            << (*p_Flights)[i-1].lateral_deviation_1000ftAGL *
m2ft << "," //ft
                            << (*p_Flights)[i-1].vertical_deviation_1000ftAGL *
m2ft<< "," //ft
                            << (*p_Flights)[i-1].avg_groundspeed_1000ftAGL *
mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_1000ftAGL *
mps2fpm << "," // fpm, negative value...
                            << (*p_Flights)[i-1].lateral_deviation_750ftAGL *
m2ft << "," //ft
                            << (*p_Flights)[i-1].vertical_deviation_750ftAGL *
m2ft<< "," //ft
                            << (*p_Flights)[i-1].avg_groundspeed_750ftAGL *
mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_750ftAGL *
mps2fpm << "," // fpm, negative value...
                            << (*p_Flights)[i-1].lateral_deviation_500ftAGL *
m2ft << "," //ft
                            << (*p_Flights)[i-1].vertical_deviation_500ftAGL *
m2ft<< "," //ft
                            << (*p_Flights)[i-1].avg_groundspeed_500ftAGL *
mps2kts << "," //kts
                            << (*p_Flights)[i-1].avg_rateofdescent_500ftAGL *
mps2fpm << "," // fpm, negative value...
                            //categories
                            << (*p_Flights)[i-1].lateral_deviation_category_10nm
<< ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_10nm<< ","
```

```
                            << (*p_Flights)[i-1].avg_airspeedcategory_10nm << ","
                            << (*p_Flights)[i-1].avg_rateofdescent_category_10nm
<< ","
                            << (*p_Flights)[i-1].heading_diff_category_10nm <<
","
                            << (*p_Flights)[i-1].lateral_deviation_category_9nm
<< ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_9nm<< ","
                            << (*p_Flights)[i-1].avg_airspeedcategory_9nm << ","
                            << (*p_Flights)[i-1].avg_rateofdescent_category_9nm
<< ","
                            << (*p_Flights)[i-1].heading_diff_category_9nm << ","
                            << (*p_Flights)[i-1].lateral_deviation_category_8nm
<< ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_8nm<< ","
                            << (*p_Flights)[i-1].avg_airspeedcategory_8nm << ","
                            << (*p_Flights)[i-1].avg_rateofdescent_category_8nm
<< ","
                            << (*p_Flights)[i-1].heading_diff_category_8nm << ","
                            << (*p_Flights)[i-1].lateral_deviation_category_7nm
<< ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_7nm<< ","
                            << (*p_Flights)[i-1].avg_airspeedcategory_7nm << ","
                            << (*p_Flights)[i-1].avg_rateofdescent_category_7nm
<< ","
                            << (*p_Flights)[i-1].heading_diff_category_7nm << ","
                            << (*p_Flights)[i-1].lateral_deviation_category_6nm
<< ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_6nm<< ","
                            << (*p_Flights)[i-1].avg_airspeedcategory_6nm << ","
                            << (*p_Flights)[i-1].avg_rateofdescent_category_6nm
<< ","
                            << (*p_Flights)[i-1].heading_diff_category_6nm << ","
                            << (*p_Flights)[i-
1].lateral_deviation_category_5dot5nm << ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_5dot5nm<< ","
                            << (*p_Flights)[i-1].avg_airspeedcategory_5dot5nm <<
","
                            << (*p_Flights)[i-
1].avg_rateofdescent_category_5dot5nm << ","
                            << (*p_Flights)[i-1].heading_diff_category_5dot5nm <<
","
                            << (*p_Flights)[i-1].lateral_deviation_category_5nm
<< ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_5nm<< ","
                            << (*p_Flights)[i-1].avg_airspeedcategory_5nm << ","
                            << (*p_Flights)[i-1].avg_rateofdescent_category_5nm
<< ","
                            << (*p_Flights)[i-1].heading_diff_category_5nm << ","
                            << (*p_Flights)[i-
1].lateral_deviation_category_4dot5nm << ","
                            << (*p_Flights)[i-
1].vertical_deviation_category_4dot5nm<< ","
```

```
                        << (*p_Flights)[i-1].avg_airspeedcategory_4dot5nm <<
","
                        << (*p_Flights)[i-
1].avg_rateofdescent_category_4dot5nm << ","
                        << (*p_Flights)[i-1].heading_diff_category_4dot5nm <<
","
                        << (*p_Flights)[i-1].lateral_deviation_category_4nm
<< ","
                        << (*p_Flights)[i-
1].vertical_deviation_category_4nm<< ","
                        << (*p_Flights)[i-1].avg_airspeedcategory_4nm << ","
                        << (*p_Flights)[i-1].avg_rateofdescent_category_4nm
<< ","
                        << (*p_Flights)[i-1].heading_diff_category_4nm << ","
                        << (*p_Flights)[i-
1].lateral_deviation_category_3dot5nm << ","
                        << (*p_Flights)[i-
1].vertical_deviation_category_3dot5nm<< ","
                        << (*p_Flights)[i-1].avg_airspeedcategory_3dot5nm <<
","
                        << (*p_Flights)[i-
1].avg_rateofdescent_category_3dot5nm << ","
                        << (*p_Flights)[i-1].heading_diff_category_3dot5nm <<
","
                        << (*p_Flights)[i-1].lateral_deviation_category_3nm
<< ","
                        << (*p_Flights)[i-
1].vertical_deviation_category_3nm<< ","
                        << (*p_Flights)[i-1].avg_airspeedcategory_3nm << ","
                        << (*p_Flights)[i-1].avg_rateofdescent_category_3nm
<< ","
                        << (*p_Flights)[i-1].heading_diff_category_3nm << ","
                        << (*p_Flights)[i-
1].lateral_deviation_category_1000ftAGL << ","
                        << (*p_Flights)[i-
1].vertical_deviation_category_1000ftAGL<< ","
                        << (*p_Flights)[i-1].avg_airspeedcategory_1000ftAGL
<< ","
                        << (*p_Flights)[i-
1].avg_rateofdescent_category_1000ftAGL << ","
                        << (*p_Flights)[i-
1].lateral_deviation_category_750ftAGL << ","
                        << (*p_Flights)[i-
1].vertical_deviation_category_750ftAGL<< ","
                        << (*p_Flights)[i-1].avg_airspeedcategory_750ftAGL <<
","
                        << (*p_Flights)[i-
1].avg_rateofdescent_category_750ftAGL << ","
                        << (*p_Flights)[i-
1].lateral_deviation_category_500ftAGL << ","
                        << (*p_Flights)[i-
1].vertical_deviation_category_500ftAGL<< ","
                        << (*p_Flights)[i-1].avg_airspeedcategory_500ftAGL <<
","
                        << (*p_Flights)[i-
1].avg_rateofdescent_category_500ftAGL << ","
                        // events
                        << (*p_Flights)[i-1].unstable1000 << ","
                        << (*p_Flights)[i-1].USD1000 << ","
                        << (*p_Flights)[i-1].USA1000 << ","
```

```
                        << (*p_Flights)[i-1].ERD1000 <<","
                        << (*p_Flights)[i-1].SAA1000 << ","
                        << (*p_Flights)[i-1].SAL1000 << ","
                        << (*p_Flights)[i-1].SAR1000 << ","
                        << (*p_Flights)[i-1].unstable750 << ","
                        << (*p_Flights)[i-1].USD750 << ","
                        << (*p_Flights)[i-1].USA750 << ","
                        << (*p_Flights)[i-1].ERD750 <<","
                        << (*p_Flights)[i-1].SAA750 << ","
                        << (*p_Flights)[i-1].SAL750 << ","
                        << (*p_Flights)[i-1].SAR750 << ","
                        << (*p_Flights)[i-1].unstable500 << ","
                        << (*p_Flights)[i-1].USD500 << ","
                        << (*p_Flights)[i-1].USA500 << ","
                        << (*p_Flights)[i-1].ERD500 <<","
                        << (*p_Flights)[i-1].SAA500 << ","
                        << (*p_Flights)[i-1].SAL500 << ","
                        << (*p_Flights)[i-1].SAR500 << "," << "\n";
                }
                else {        // if not qualified track
                       out_summary
                        << (*p_Flights)[i-1].date << ","
                        << (*p_Flights)[i-1].trackindex << ","
                        << (*p_Flights)[i-1].flightID << ","
                        << (*p_Flights)[i-1].go_around << ","
                        << (*p_Flights)[i-1].track_qualified << ","
                        << (*p_Flights)[i-1].time2landing_entrance << ","
                        << (*p_Flights)[i-1].landingrunway.name << ","
                        << (*p_Flights)[i-1].aircraft.weight_class << ","
                        <<
"N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A,N/A"<<"\n"
;
                }
        }
        // output summary of key statistics
        out_summary << "total number of flights to airport " << CURRENTAIRPORT <<
"," << (*p_Flights).size() << "\n";
        out_summary << "total number of flights to target runway " <<
currentrunway << "," << n_tracks << "," <<
100.0*(double)n_tracks/(double)(*p_Flights).size()<< "%,"  << "\n";
        out_summary << "qualified tracks," << n_qualifiedTracks << "," <<
100.0*(double)n_qualifiedTracks/(double)n_tracks << "%," << "\n";
        out_summary << "Superheavy aircraft," << n_Superheavy << "," <<
100.0*(double)n_Superheavy/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "Heavy aircraft," << n_Heavy << "," <<
100.0*(double)n_Heavy/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "B757 aircraft," << n_B757 << "," <<
100.0*(double)n_B757/(double)n_qualifiedTracks<< "%,"  << "\n";
        out_summary << "Large aircraft," << n_Large << "," <<
100.0*(double)n_Large/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "Small aircraft," << n_Small << "," <<
100.0*(double)n_Small/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "Unstable approach 1000ftAGL," << n_unstable1000 << "," <<
100.0*(double)n_unstable1000/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "Unstable approach 750ftAGL," << n_unstable750 << "," <<
100.0*(double)n_unstable750/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "Unstable approach 500ftAGL," << n_unstable500 << "," <<
100.0*(double)n_unstable500/(double)n_qualifiedTracks << "%," << "\n";
        out_summary << "USD1000 event," << n_USD1000 << "," <<
100.0*(double)n_USD1000/(double)n_qualifiedTracks << "%," << "\n";
```

```
	out_summary << "USD750 event," << n_USD750 << "," <<
100.0*(double)n_USD750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "USD500 event," << n_USD500 << "," <<
100.0*(double)n_USD500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "USA1000 event," << n_USA1000 << "," <<
100.0*(double)n_USA1000/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "USA750 event," << n_USA750 << "," <<
100.0*(double)n_USA750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "USA500 event," << n_USA500 << "," <<
100.0*(double)n_USA500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "ERD1000 event," << n_ERD1000 << "," <<
100.0*(double)n_ERD1000/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "ERD750 event," << n_ERD750 << "," <<
100.0*(double)n_ERD750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "ERD500 event," << n_ERD500 << "," <<
100.0*(double)n_ERD500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAL1000 event," << n_SAL1000 << "," <<
100.0*(double)n_SAL1000/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAL750 event," << n_SAL750 << "," <<
100.0*(double)n_SAL750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAL500 event," << n_SAL500 << "," <<
100.0*(double)n_SAL500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAR1000 event," << n_SAR1000 << "," <<
100.0*(double)n_SAR1000/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAR750 event," << n_SAR750 << "," <<
100.0*(double)n_SAR750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAR500 event," << n_SAR500 << "," <<
100.0*(double)n_SAR500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAA1000 event," << n_SAA1000 << "," <<
100.0*(double)n_SAA1000/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAA750 event," << n_SAA750 << "," <<
100.0*(double)n_SAA750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAA500 event," << n_SAA500 << "," <<
100.0*(double)n_SAA500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAB1000 event," << n_SAB1000 << "," <<
100.0*(double)n_SAB1000/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAB750 event," << n_SAB750 << "," <<
100.0*(double)n_SAB750/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "SAB500 event," << n_SAB500 << "," <<
100.0*(double)n_SAB500/(double)n_qualifiedTracks << "%," << "\n";
	out_summary << "Go-around," << n_go_around << "," <<
100.0*(double)n_go_around/(double)n_qualifiedTracks << "%," << "\n";
	// output breakdown table (///obsolete?)
	int count_summary[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //for 1000ft
	int count_summary2[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //for 750ft
	int count_summary3[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //for 500ft
	for (int i = 1; i <= (*p_Flights).size(); ++i) {
		if ((*p_Flights)[i-1].landingrunway.name != currentrunway)
			continue;
		// 1000 ft AGL
		if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == false && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-
1].SAL1000 == false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[0]++;
		if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == false && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-
1].SAL1000 == true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[1]++;
		if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == false && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-
1].SAL1000 == false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[2]++;
```

```
            if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == false && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-
1].SAL1000 == true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[3]++;
            if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == true && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-
1].SAL1000 == false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[4]++;
            if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == true && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-
1].SAL1000 == true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[5]++;
            if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == true && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-
1].SAL1000 == false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[6]++;
            if ((*p_Flights)[i-1].USD1000 == false && (*p_Flights)[i-
1].ERD1000 == true && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-
1].SAL1000 == true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[7]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== false && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-1].SAL1000 ==
false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[8]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== false && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-1].SAL1000 ==
true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[9]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== false && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-1].SAL1000 ==
false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[10]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== false && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-1].SAL1000 ==
true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[11]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== true && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-1].SAL1000 ==
false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[12]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== true && (*p_Flights)[i-1].SAA1000 == false && ((*p_Flights)[i-1].SAL1000 ==
true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[13]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== true && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-1].SAL1000 ==
false && (*p_Flights)[i-1].SAR1000 == false)) count_summary[14]++;
            if ((*p_Flights)[i-1].USD1000 == true && (*p_Flights)[i-1].ERD1000
== true && (*p_Flights)[i-1].SAA1000 == true && ((*p_Flights)[i-1].SAL1000 ==
true || (*p_Flights)[i-1].SAR1000 == true)) count_summary[15]++;
            // 750 ft AGL
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[0]++;
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[1]++;
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[2]++;
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[3]++;
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == false &&((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[4]++;
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[5]++;
```

```
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[6]++;
            if ((*p_Flights)[i-1].USD750 == false && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[7]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[8]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[9]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[10]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== false && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[11]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[12]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == false && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[13]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
false && (*p_Flights)[i-1].SAR750 == false)) count_summary2[14]++;
            if ((*p_Flights)[i-1].USD750 == true && (*p_Flights)[i-1].ERD750
== true && (*p_Flights)[i-1].SAA750 == true && ((*p_Flights)[i-1].SAL750 ==
true || (*p_Flights)[i-1].SAR750 == true)) count_summary2[15]++;
            // 500 ft AGL
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[0]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[1]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[2]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[3]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[4]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[5]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[6]++;
            if ((*p_Flights)[i-1].USD500 == false && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[7]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[8]++;
```

```
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[9]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[10]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== false && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[11]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[12]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == false && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[13]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
false && (*p_Flights)[i-1].SAR500 == false)) count_summary3[14]++;
            if ((*p_Flights)[i-1].USD500 == true && (*p_Flights)[i-1].ERD500
== true && (*p_Flights)[i-1].SAA500 == true && ((*p_Flights)[i-1].SAL500 ==
true || (*p_Flights)[i-1].SAR500 == true)) count_summary3[15]++;
      }
      out_summary << "Breakdown summary _ at 1000ft AGL" << "\n";
      for (int i = 1; i <= 16; ++i) out_summary << count_summary[i-1] << "," <<
(double)count_summary[i-1]/(double)n_qualifiedTracks << "\n";
      out_summary << "Breakdown summary _ at 750ft AGL" << "\n";
      for (int i = 1; i <= 16; ++i) out_summary << count_summary2[i-1] << ","
<< (double)count_summary2[i-1]/(double)n_qualifiedTracks << "\n";
      out_summary << "Breakdown summary _ at 500ft AGL" << "\n";
      for (int i = 1; i <= 16; ++i) out_summary << count_summary3[i-1] << ","
<< (double)count_summary3[i-1]/(double)n_qualifiedTracks << "\n";
      out_summary.close();

      prediction_lookuptable(p_Flights, "10nm", "Large");
      prediction_lookuptable(p_Flights, "9nm", "Large");
      prediction_lookuptable(p_Flights, "8nm", "Large");
      prediction_lookuptable(p_Flights, "7nm", "Large");
      prediction_lookuptable(p_Flights, "6nm", "Large");
      prediction_lookuptable(p_Flights, "5dot5nm", "Large");
      prediction_lookuptable(p_Flights, "5nm", "Large");
      prediction_lookuptable(p_Flights, "4dot5nm", "Large");
      prediction_lookuptable(p_Flights, "4nm", "Large");
      prediction_lookuptable(p_Flights, "3dot5nm", "Large");
      prediction_lookuptable(p_Flights, "3nm", "Large");
}
//*************************************************
void main()
{
//load wind data
      wind20070111 = load_wind("../Data/DATA _ WIND/20070111 - processed.csv");
      wind20070112 = load_wind("../Data/DATA _ WIND/20070112 - processed.csv");
      wind20070113 = load_wind("../Data/DATA _ WIND/20070113 - processed.csv");
      wind20070114 = load_wind("../Data/DATA _ WIND/20070114 - processed.csv");
      wind20070115 = load_wind("../Data/DATA _ WIND/20070115 - processed.csv");
      wind20070116 = load_wind("../Data/DATA _ WIND/20070116 - processed.csv");
      wind20070117 = load_wind("../Data/DATA _ WIND/20070117 - processed.csv");
      wind20070323 = load_wind("../Data/DATA _ WIND/20070323 - processed.csv");
      wind20070324 = load_wind("../Data/DATA _ WIND/20070324 - processed.csv");
      wind20070325 = load_wind("../Data/DATA _ WIND/20070325 - processed.csv");
      wind20070326 = load_wind("../Data/DATA _ WIND/20070326 - processed.csv");
```

```
	wind20070327 = load_wind("../Data/DATA _ WIND/20070327 - processed.csv");
	wind20070328 = load_wind("../Data/DATA _ WIND/20070328 - processed.csv");
	wind20070329 = load_wind("../Data/DATA _ WIND/20070329 - processed.csv");
	wind20070721 = load_wind("../Data/DATA _ WIND/20070721 - processed.csv");
	wind20070722 = load_wind("../Data/DATA _ WIND/20070722 - processed.csv");
	wind20070723 = load_wind("../Data/DATA _ WIND/20070723 - processed.csv");
	wind20070724 = load_wind("../Data/DATA _ WIND/20070724 - processed.csv");
	wind20070725 = load_wind("../Data/DATA _ WIND/20070725 - processed.csv");
	wind20070726 = load_wind("../Data/DATA _ WIND/20070726 - processed.csv");
	wind20070727 = load_wind("../Data/DATA _ WIND/20070727 - processed.csv");
	wind20071001 = load_wind("../Data/DATA _ WIND/20071001 - processed.csv");
	wind20071002 = load_wind("../Data/DATA _ WIND/20071002 - processed.csv");
	wind20071003 = load_wind("../Data/DATA _ WIND/20071003 - processed.csv");
	wind20071004 = load_wind("../Data/DATA _ WIND/20071004 - processed.csv");
	wind20071005 = load_wind("../Data/DATA _ WIND/20071005 - processed.csv");
	wind20071006 = load_wind("../Data/DATA _ WIND/20071006 - processed.csv");
	wind20071007 = load_wind("../Data/DATA _ WIND/20071007 - processed.csv");
//load aircraft data
	vector<AIRCRAFT> Aircrafts;
	load_aircraft_data(Aircrafts, AIRCRAFT_DATA_FILE);
	AIRCRAFT *p_aircraft = &Aircrafts[0];
//load flight/track data file names
	string trackfiles[N_DAYS], flightfiles[N_DAYS];
	for (int i = 1; i <= N_DAYS; ++i) {
		flightfiles[i-1] = "../Data/DATA _ Preprocessed/" + CURRENTAIRPORT
+ "_" + CURRENTDATA +"/[input] flights " + CURRENTAIRPORT + DATES[i-1] + "
arrival.csv";
		trackfiles[i-1] = "../Data/DATA _ Preprocessed/" + CURRENTAIRPORT
+ "_" + CURRENTDATA + "/[input] tracks " + CURRENTAIRPORT + DATES[i-1] + "
arrival.csv";
	}
//load airport & runway data, build wireframe
	loadAirportRunwaydata();
//plot wireframes
	/*vector<RUNWAY> EWRrunway22L;
	EWRrunway22L.push_back(EWR22L);
	cout<<EWR22L.location_12nm.lat<<","<<EWR22L.location_12nm.lng;
	plot_finalapproachsegment(FINALAPPROACHWIREFRAMEFILE_EWR, EWRrunway22L);
	*/
	//plot_finalapproachsegment(FINALAPPROACHWIREFRAMEFILE_JFK, JFKrunways);
	//plot_finalapproachsegment(FINALAPPROACHWIREFRAMEFILE_LGA, LGArunways);
	//plot_finalapproachsegment(FINALAPPROACHWIREFRAMEFILE_EWR, EWRrunways);
	//plot_finalapproachsegment(FINALAPPROACHWIREFRAMEFILE_MDW, MDWrunways);
//load flight/track data
	//preprocessPDARStracksforall(); // generate PDARS track files
	vector<FLIGHT> Flights;
	int i_flights_pre = 0;
	for (int i = 1; i <= N_DAYS; ++i) {
		cout<<"loading flights and tracks in " << DATES[i-1] << "\n";
		load_flights(&Flights, flightfiles[i-1], p_aircraft,
Aircrafts.size());
		load_tracks(&Flights, trackfiles[i-1], i_flights_pre);
		i_flights_pre = Flights.size();
	}
//preprocess tracks
	//plot_tracks(Flights, CURRENTTRACKSKMLOUTPUT, "ffffffff", 3,
AIRCRAFT_ICON); //raw
	vector<TRACKPOINT> track_linear_interpolated, track_smoothed;
	for (int i = 1; i <= Flights.size(); ++i) {
		cout << "Preprocessing track " << Flights[i-1].trackindex << "\n";
```

```
            track_linear_interpolated = linear_interpolate(&(Flights[i-
1].Track));
            track_smoothed = smooth_track(&track_linear_interpolated);
            derived_track_data(&track_smoothed, Flights[i-1].date);
            Flights[i-1].Track = track_smoothed;
            track_linear_interpolated.clear();
            track_smoothed.clear();
      }
//identify landing runway for each flight
      identify_runway(Flights, CURRENTAIRPORT);
      if (CURRENTAIRPORT == "EWR") {
            process("22L", &Flights);
            //process("04R", &Flights);
            //process("22R", &Flights);
            //process("04L", &Flights);
            //process("11", &Flights);
            //process("29", &Flights);
      }
      if (CURRENTAIRPORT == "JFK") {
            process("13R", &Flights);
            process("31L", &Flights);
            process("04L", &Flights);
            process("22R", &Flights);
            process("13L", &Flights);
            process("31R", &Flights);
            process("04R", &Flights);
            process("22L", &Flights);
      }
      if (CURRENTAIRPORT == "LGA") {
            process("13", &Flights);
            process("31", &Flights);
            process("22", &Flights);
            process("04", &Flights);
      }
}
```

APPENDIX B: MATLAB CODE FOR SUPERVISED LEARNING

Main scripts

```
clear; close all; clc

%% training
data = load('[input] All basic features and events.csv');
m_samples = size(data,1); %total number of rows (8158)
m_train = 5000; %e.g. 5000 of 8158
m_test = m_samples - m_train;
n_locations = 10; %10, 9, 8, 7, 6, 5.5, 5, 4.5, 4, 3.5 nm
n_events = 21; %for each stabilization check point: 7 events (1 overall event
UA, 6 subevents: USD,USA,ERD,SAA,SAL,SAR)
n_basicfeatures = 22;
n_allfeatures = 2*n_basicfeatures + nchoosek(n_basicfeatures,2);
% n basic terms + n squared terms + n-choose-2 interaction terms

%Input data: <n_locations*n_allfeatures + n_events> columns by <n_samples>
rows.
for iii = 1:10 %replicates of experements
   rand_indices{iii} =  randperm(m_samples);  %randomly permutate sample
indices
current_randindices = rand_indices{iii};

%load training/test set into cells X_train/test{}
col = 1; %a column index
for i_location = 1:n_locations %location index ordering 1~11: 10, 9, 8, 7, 6,
5.5, 5, 4.5, 4, 3.5, 3nm
    X_train{i_location} = data(current_randindices(1:m_train),
col:col+(n_basicfeatures-1));
    X_test{i_location} = data(current_randindices(m_train+1:end),
col:col+(n_basicfeatures-1));
    %e.g. for i_location = 1, columns 1:n_basicfeatures are for all 10nm data
    col = col + n_basicfeatures; %jump to columns for next location
    fprintf('loading and processing training/test set at location ID = %d, \n',
i_location);

    squared_term = X_train{i_location}.^2;
    interaction_term = zeros(m_train, nchoosek(n_basicfeatures,2));
    squared_term_test = X_test{i_location}.^2;
    interaction_term_test = zeros(m_test, nchoosek(n_basicfeatures,2));
    column = 1;
    for j =  1:n_basicfeatures-1
        for k = j+1:n_basicfeatures
            interaction_term(:, column) =
X_train{i_location}(:,j).*X_train{i_location}(:,k);
            interaction_term_test(:, column) =
X_test{i_location}(:,j).*X_test{i_location}(:,k);
            column = column + 1;
        end
```

```
    end
    X_train{i_location} = [X_train{i_location}, squared_term,
interaction_term];
    X_test{i_location} = [X_test{i_location}, squared_term_test,
interaction_term_test];

    %normalization by (x-miu)/sigma
    means = zeros(m_train, size(X_train{i_location}, 2));
    stdevs = zeros(m_train, size(X_train{i_location}, 2));
    means_test = zeros(m_test, size(X_test{i_location}, 2));
    stdevs_test = zeros(m_test, size(X_test{i_location}, 2));
    for ii = 1:m_train
        means(ii, :) = mean(X_train{i_location});
        stdevs(ii, :) = std(X_train{i_location});
    end
    for ii = 1:m_test
        means_test(ii, :) = mean(X_train{i_location}); %still use train set
        stdevs_test(ii, :) = std(X_train{i_location});
    end
    X_train{i_location} = (X_train{i_location} - means)./stdevs;
    X_test{i_location} = (X_test{i_location} - means_test)./stdevs_test;

    %add additional default feature (bias unit x0) for all training sets
    X_train{i_location} = [ones(m_train, 1), X_train{i_location}];
    X_test{i_location} = [ones(m_test, 1), X_test{i_location}];
end
col = n_basicfeatures*n_locations + 1; %jump to columns of events
%unstable events index ordering:
%1~7: unstable1000,USD1000,USA1000,ERD1000,SAA1000,SAL1000,SAR1000
%8~14: unstable750,USD750,USA750,ERD750,SAA750,SAL750,SAR750
%15~21: unstable500,USD500,USA500,ERD500,SAA500,SAL500,SAR500
for j_event = 1:n_events
   y_train{j_event} =  data(current_randindices(1:m_train), col); %m_train by 1
for each cell content
   y_test{j_event} =  data(current_randindices(m_train+1:end), col);
   col = col + 1;
end

row = 1;
Theta = zeros(n_locations*n_events, n_allfeatures+1); % (n_locations*n_events)
by (n_allfeatures+1)
% coefficients for each case stored in each row of Theta matrix
for i_location = 1:n_locations
    for j_event = 1:n_events
        fprintf('Training model... location ID %d, event ID %d', i_location,
j_event);
        initial_theta = zeros(n_allfeatures+1, 1); %column vector
(n_allfeatures+1 by 1)
        options = optimset('GradObj', 'on', 'MaxIter', 1000);
        [theta, cost] = fminunc(@(t)(costFunction(t, X_train{i_location},
y_train{j_event})), initial_theta, options);
        Theta(row,:) = theta'; %convert to row for output
        row =  row + 1;
    end
end
csvwrite('[output] Trained_theta.csv', Theta);
%% testing
%define location lables, note the cell index ordering
Location{1} = '10nm'; Location{2} = '9nm';
Location{3} = '8nm'; Location{4} = '7nm';
```

```
Location{5} = '6nm'; Location{6} = '5.5nm';
Location{7} = '5nm'; Location{8} = '4.5nm';
Location{9} = '4nm'; Location{10} = '3.5nm';
Location{11} = '3nm';

%predict at each location
for i_location = 1:n_locations
    %prediction
    y_test_predicted = zeros(m_test, n_events);
    y_train_predicted = zeros(m_train, n_events); % this is for error
calculation on train set only
    row = 1 + (i_location-1)*n_events; %starting row of trained Theta, 10nm ->
row1, 9nm -> row22,...

    for i_event = 1:n_events
        %predict(theta <nfeatures by 1>, X <n_testsamples by n_allfeatures>
        %output: assign single value 0 or 1
        y_train_predicted(:,i_event) =  predict(Theta(row,:)',
X_train{i_location}); % for error calculation on trainset only
        y_test_predicted(:,i_event) = predict(Theta(row,:)',
X_test{i_location}); % for each event: n_test by 1
        row = row + 1;
    end

    %performance measures
    %n_...: for all events, each event correspond to a single number of count
    n_correct = zeros(1, n_events); %if predicted value is same with true value
(both stable and unstable)
    n_stableevents = zeros(1, n_events); %number of stable events in test set
    n_unstableevents = zeros(1, n_events); %number of unstable events in test
set
    n_stablecorrect = zeros(1, n_events); %number of correctly predicted stable
events
    n_unstablecorrect = zeros(1, n_events); %number of correctly predicted
unstable events
    %"1" for unstable, "0" for stable
    for i = 1:m_test
        for j = 1:n_events
            if y_test{j}(i) == 1 %count actual unstable events
                n_unstableevents(j) = n_unstableevents(j) + 1;
            end
            if y_test{j}(i) == 0 %count actual stable events
                n_stableevents(j) = n_stableevents(j) + 1;
            end

            if y_test{j}(i) == y_test_predicted(i,j)
                n_correct(j) = n_correct(j) + 1;
            end
            if y_test{j}(i) == 0 && y_test_predicted(i,j) == 0
                n_stablecorrect(j) = n_stablecorrect(j) + 1;
            end
            if y_test{j}(i) == 1 && y_test_predicted(i,j) == 1
                n_unstablecorrect(j) = n_unstablecorrect(j) + 1;
            end
        end
    end

    %accuracy, precision, recall, F1 Score
    Accuracy =
(n_unstablecorrect+n_stablecorrect)./(n_unstableevents+n_stableevents);
```

```
    Precision = n_unstablecorrect./(n_unstablecorrect+n_stableevents-
n_stablecorrect);
    Recall = n_unstablecorrect./n_unstableevents;
    F1_Score = (2*Precision.*Recall)./(Precision + Recall);
    AllMeasures = [Accuracy; Precision; Recall; F1_Score];
    %columns:
        %1~7: unstable1000,USD1000,USA1000,ERD1000,SAA1000,SAL1000,SAR1000
        %8~14: unstable750,USD750,USA750,ERD750,SAA750,SAL750,SAR750
        %15~21: unstable500,USD500,USA500,ERD500,SAA500,SAL500,SAR500
    csvwrite(strcat('[output] Prediction performance ',
Location{i_location},num2str(iii),'.csv'), AllMeasures);

%errors for train and test set (for Unstable1000 event)
    fprintf(Location{i_location});
    error_train = 0;
    for i = 1:m_train
       error_train =  error_train + (y_train_predicted(i,1)-y_train{1}(i))^2;
    end
    error_train = error_train/m_train;
    fprintf(', error for train = %d, ', error_train);

    error_test = 0;
    for i = 1:m_test
        error_test = error_test + (y_test_predicted(i,1)-y_test{1}(i))^2;
    end
    error_test = error_test/m_test;
    fprintf('error for test = %d\n', error_test);
end
end
```

Prediction function

```
function p = predict(theta, X)
m = size(X, 1); % Number of rows (test examples)
p = zeros(m, 1); % m by 1 vector, labels for all test examples

for i=1:m
    if X(i,:)*theta >= 0
        p(i) = 1;
    else
        p(i) = 0;
    end
end
```

Sigmoid function

```
function g = sigmoid(z)
g = zeros(size(z));
g = 1./(1+exp(-z));
end
```

REFERENCES

Andrews, J., and J. Robinson. (2001). *Radar-Based Analysis of the Efficiency of Runway Use*. Guidance AIAA-2001-4359. Presented at Navigation and Control Conference, American Institute of Aeronautics and Astronautics, Montreal, Quebec, Canada, 2001.

Chandola, Varun, A. Banerjee, and V. Kumar. (2009). *Outliner detection: A survey*. ACM Computing Surveys: 1-72.

Darby. R. (2010). *Safety in Numbers*. Aerosafety world; 2010 p. 48. Retrieved from http://flightsafety.org/aerosafety-world-magazine/july-2010/safety-in-numbers

DGAC. (2006). Direction Générale de l'Aviation Civil. *Unstabilised Approaches* - the symposia.

EUROCONTROL. (2012). Base of Aircraft Data (BADA) v3.8 | EUROCONTROL. Retrieved November 25, 2012. http://www.eurocontrol.int/services/bada

FAA. (2003). *Advisory Circular. 120-71a Standard Operating Procedures for Flight Deck Crewmembers*, Washington, DC.

FAA. (2004). *Advisory Circular. 120-82 Flight Operational Quality Assurance*, Washington, DC.

FAA. (2006). *Stabilized Approach Concept*. Retrieved from https://www.faasafety.gov/files/gslac/courses/content/35/376/Stabilized%20Approach%20Concept.pdf.

FSF. (1999). *Killers in Aviation: FSF Task Force Presents Facts About Approach-and-landing and Controlled-flight-into-terrain Accidents.* Flight Safety Digest Volume 17 (November-December 1998) and Volume 18 (January-February 1999): 1-121.

FSF. (2010). *FSF ALAR Briefing Note 7.1 Stabilized approach*. Retrieved from http://www.skybrary.aero/bookshelf/content/bookDetails.php?bookId=864.

Hall, T., M. Soares. (2008). *Analysis of Localizer and Glide Slope Flight Technical Error*. 27th Digital Avionics Systems Conference, St. Paul, MN.

Haynie, C. (2002). *An Investigation of Capacity and Safety in Near-terminal Airspace Guiding Information Technology Adoption*. PhD dissertation. George Mason University, Fairfax, VA.

ICAO. (2014). Safety Report 2014. Retrieved from http://www.icao.int/safety/Documents/ICAO_2014%20Safety%20Report_final_02042014_web.pdf

Jeddi, B., J. F. Shortle, and L. Sherry. (2006). *Statistics of the Approach Process at Detroit Metropolitan Wayne County Airport*. Proc., International Conference on Research in Air Transportation, Belgrade, Serbia, and Montenegro, pp. 85–92.

Jeddi, B., J. Shortle. (2007). Throughput, risk, and economic optimality of runway landing operations. 7th USA/Europe ATM R&D Seminar, Barcelona, Spain, Paper 162.

Li, L, M. Gariel, R. Hansman, and R. Palacios. (2011). *Anomaly detection in onboard-recorded flight data using cluster analysis*. in Proc. 30th IEEE/AIAA Digital Avionics Systems Conference (DASC), Oct. 2011, pp. 4A4–1 – 4A4–11.

Matthews, B., S. Das, K. Bhaduri., K. Das., R. Martin., N. Oza. (2013) *Discovering Anomalous Aviation Safety Events using Scalable Data Mining Algorithms*. AIAA Journal of Aerospace Computing, Information, and Communication. 2013.

Matthews, B., Nielsen, D. (2014). *Automated Discovery of Flight Track Anomalies*. in Proc. 33th IEEE/AIAA Digital Avionics Systems Conference (DASC), Oct. 2014.

McGovern, S. M., S. B. Cohen; M. Truong; G. Farley. *Kinematics-Based model for Stochastic Simulation of Aircraft Operating in the National Airspace System*. US DOT National Transportation Systems Center, EG&G Technical Services 39173.

Merritt A, J. R. Klinect. (2006). *Defensive Flying for Pilots: An Introduction to Threat and Error Management*. The University of Texas human factors research project.

Miquel, T., M. Felix, L. Jean-Marc. (2006). *Path Searching and Tracking for Time-Based Aircraft Spacing at Meter Fix*. Direction des Services de la Navigation Aerienne, Ecole National de l' Aviation Civile.

Moriarty, D., S. Jarvis. (2014). *A systems perspective on the unstable approach in commercial aviation*. Reliability Engineering & System Safety, 131, 197-202.

Musialek, B., C. F. Munafo, H. Ryan, M. Paglione. (2010). *Literature Survey of Trajectory Predictor Technology*. Technical Report DOT/FAA/TCTN11/1, Federal Aviation Administration, William J. Hughes Technical Center

Rakas, J., and H. Yin. (2005). *Statistical Modeling and Analysis of Landing Time Intervals: Case Study of Los Angeles International Airport, California*. In Transportation Research Record: Journal of the Transportation Research Board, No. 1915, Transportation Research Board of the National Academies, Washington, D.C., 2005, pp. 69–78.

Richards, R. (2002). *Application of multiple artificial intelligence techniques for an aircraft carrier landing decision support tool*. In Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on (Vol. 1, pp. 7-11). IEEE.

Rodriguez, J. M. C., et al. *A model to 4D descent trajectory guidance*. Digital Avionics Systems Conference, 2007. DASC'07. IEEE/AIAA 26th. IEEE, 2007.

Sherry, L., Z. Wang, H. Kourdali, J. Shortle. (2013). *Big data analysis of irregular operations: aborted approaches and their underlying factors*. Integrated Communication, Navigation, and Surveillance Conference, Herndon, VA.

Shortle, J., B. Jeddi. (2007). *Probabilistic Analysis of Wake Vortex Hazards for Landing Aircraft Using Multilateration Data*. Transportation Research Record: Journal of the Transportation Research Board. No.2007, 90-96.

Shortle, J., Y. Zhang, J. Wang. (2010). *Statistical Characteristics of Aircraft Arrival Tracks*. Transportation Research Record: Journal of the Transportation Research Board. No. 2177, 98-104.

Shortle, J., L. Sherry. (2013). *A Model for Investigating the Interaction between Go-Arounds and Runway Throughput*. 2013 Aviation Technology, Integration, and Operations Conference.

Thipphavong, D.P., C.A. Schultz, A.G. Lee, and S. Chan. (2013). *Adaptive Algorithm to Improve Trajectory Prediction Accuracy of Climbing Aircraft.* Journal of Guidance, Control, and Dynamics, 36(1), p. 15-24.

Treder, B., B. Crane. (2004) *Application of Insightful Corporations Data Mining Algorithms to FOQA Data at JetBlue Airways*. Flight Safety Foundation Report. Retrieved from http://flightsafety.org/files/FOQA_data_mining_report.pdf.

Turner, T. (2011). *Flying Lessons – 111208*. Retrieved from http://www.faasafety.gov/files/gslac/library/documents/2011/Dec/59270/111208%20FLYING%20LESSONS.pdf.

Wang, Z., J. Wang, J. F. Shortle. (2012). *Sensitivity Analysis of Potential Wake Encounters to Stochastic Flight-Track Parameters*. In Fifth International Conference on Research in Air Transportation, University of California, Berkeley, May 22 – 25.

Wang, Z, L. Sherry, and J. Shortle. (2015) *Airspace Risk Management Using Surveillance Track Data: Stabilized Approaches*. 8th Integrated Communications Navigation and Surveillance (ICNS) Conference, Dulles, VA.

Xie, Y. (2005). *Quantitative Analysis of Airport Arrival Capacity and Arrival Safety Using Stochastic Models*. PhD dissertation, George Mason University, Fairfax, VA.

Zhang, Y., J. F. Shortle. (2010). *Comparison of Arrival Tracks at Different Airports*. In Proceedings of 4th International Conference on Research in Air Transportation, Budapest Hungary, June 01 – 04.

BIOGRAPHY

Zhenming Wang received B.S. in Optical Information Science and Technology from Harbin Engineering University, China, in 2009. He received M.S. in Operations Research from George Mason University, Fairfax, Virginia in 2012.