A Computer Model to Predict Potential Wake Turbulence Encounters in the National Airspace System

Zheng Fan

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Civil Engineering

Antonio A. Trani, Chair

Antoine Hobeika

Abbas Montasir

Raghu Pasupathy

December 1, 2014

Blacksburg, Virginia

Keywords: Air Traffic Control, Wake Turbulence, Next-Gen Operation, Air Traffic Simulation, Air Traffic Safety, Air Traffic Capacity, Simulation, Modeling

A Computer Model to Predict Potential Wake Turbulence Encounters in the National Airspace

Zheng Fan

ABSTRACT

With an increasing population of super heavy aircraft operating in the National Airspace System and with the introduction of NextGen technologies, the wake vortex problem has become more important for airport capacity and the en-route air traffic operations. The vortices generated by heavy and super heavy aircraft can generate potential hazards to other aircraft on nearby flight paths. Moreover, the design of new airport procedures needs to consider the interactions between aircraft in closer paths. New methods and models are required to examine these effects before new operations are conducted in the National Airspace System (NAS).

Reducing wake vortex separations to safe levels between successive aircraft is essential for NextGen operations. One approach taken recently by ICAO and the FAA is to introduce a re-categorization (ReCat) of wake vortex separations to six groups from the existing five groups employed by the FAA in the United States. Reduced aircraft separations can increase capacity in the NAS with corresponding savings in delay times at busy airports. Future NextGen operations are likely to introduce smaller aircraft separations in the en-route and in the terminal area. Such operations would require better methods to identify potential wake hazards from reduced separation operations. This dissertation describes a model to identify potential wake encounters in the future NAS.

The goal of the dissertation is to describe the Enhanced Wake Encounter Model (EWEM), a model that employs a detailed NASA-developed wake model to generate wake zones for different aircraft categories under different flight conditions that can be used with aircraft flight

path data to identify potential wake encounters. The main contribution of this model is to gain an understanding of potential wake encounters under future NAS operations.

ACKNOWLEDGEMENT

First, I would like to thank my advisor, Dr. Trani, for guiding and supporting me through the process of completing this research. I would also like to thank my committee member, Dr. Hobeika, Dr. Abbas, and Dr. Pasupathy for serving on the committee and providing me advices all the time.

I would like to thank my parents for supporting me in my entire life. I am really sorry for not being around with them all these years. I am very grateful for all the understandings and guidance they gave me when I am down.

I would like to thank my boyfriend, Xusheng He, for the support he gave me during the time I complete my degree. Thank you for taking care of me and our dog. Thank you for supporting and encouraging me to chase my dream.

I want to thank all my colleagues, Howard, Zhihao Zou, Saloni Chirania, Thomas Spenser, Yang Zhang, David Pu, Tao Li and Osama Salous. It was a great pleasure working with all of you. Thank you for all the support and all the good times as well.

I want to thank all my friends in Blacksburg, as well as all of my friends overseas for the support and help. I want to give special thanks to Wenjing Xue, who was there for me all these years during bad and good time.

I would like to than Nicholas Hinze for helping me complete the research, learning MATLAB, and assist me with the GUI development.

Finally, I would like to tan the FAA for sponsoring this project, special thanks to Tom Proeschel, Ed Johnson, Dr. John Proschel and Tamas Kolos-Lakatos for their assistance and guidance.

Table of Contents

List of Tables

List of Figures

1 Introduction

The purpose of this dissertation is to develop an enhanced wake model to assess potential aircraft wake vortex encounters in the National Airspace System (NAS). This model is a substantial enhancement to an existing wake encounter model (WEM) developed by Swol (Swol 2009) and improved by Schroeder (Schroeder 2011).

This dissertation is structured in eight chapters. Chapter 1 introduces the background and the research motivation of this study. In Chapter 2, we review past efforts in wake vortex research, including policies, standards, data collection, and model development. In Chapter 3, we describe the approach and algorithms of the new model. We also discuss challenges and problems encountered during the model development. Chapter 4 presents sample results model with an application to model airspace capacity analysis. Chapter 5 presents information on model validation and verification. Chapter 6 introduces some model improvements regarding the most important factor in the model. Chapter 7 is the user's manual for the model. Finally, Chapter 8 provides conclusions and recommendations for the future research.

1.1 Background

Ever since Wilbur Wright took an employee along for a ride in 1908, commercial aviation has shown positive long-term growth for more than 100 years. In the year 2013, more than 9 million flights took off from every corner of the U.S. In 2013, the number of revenue passenger enplanements was more than 700 million (RITA 2014). The International Air Transport Association (IATA) reported that the global passenger traffic in September shows a continuous positive demand trend (IATA 2013). For example, in 2013, the total revenue passenger kilometers rose by 5.5% compared to the year 2012. Based on the FAA forecast report published

in 2014, the national enplanements will increase to 1,149.5 million by 2034 (FAA 2014). The annual growth rate of the airport operations at the core airports is expected to grow by 2.8% annually, and the growth of enplanements for the core airports is expected to be around 3%, as presented in Figure 1 (FAA 2012b). In contrast, the percent of on-time arrivals was 76.9% in 2013, dropping by 5% from the previous year (RITA 2014).

The Federal Aviation Administration (FAA) is leading efforts to increase the capacity of the National Airspace System (NAS). Different technology implementations are being considered to improve the current system, including new air traffic control procedures, and new aircraft surveillance technologies. These include: satellite-based navigation, surveillance, and communication systems. Beyond all the improvements cited (FAA 2013a), wake turbulence with continue to be a limiting factor to the improvements of the capacity of the National Airspace System.

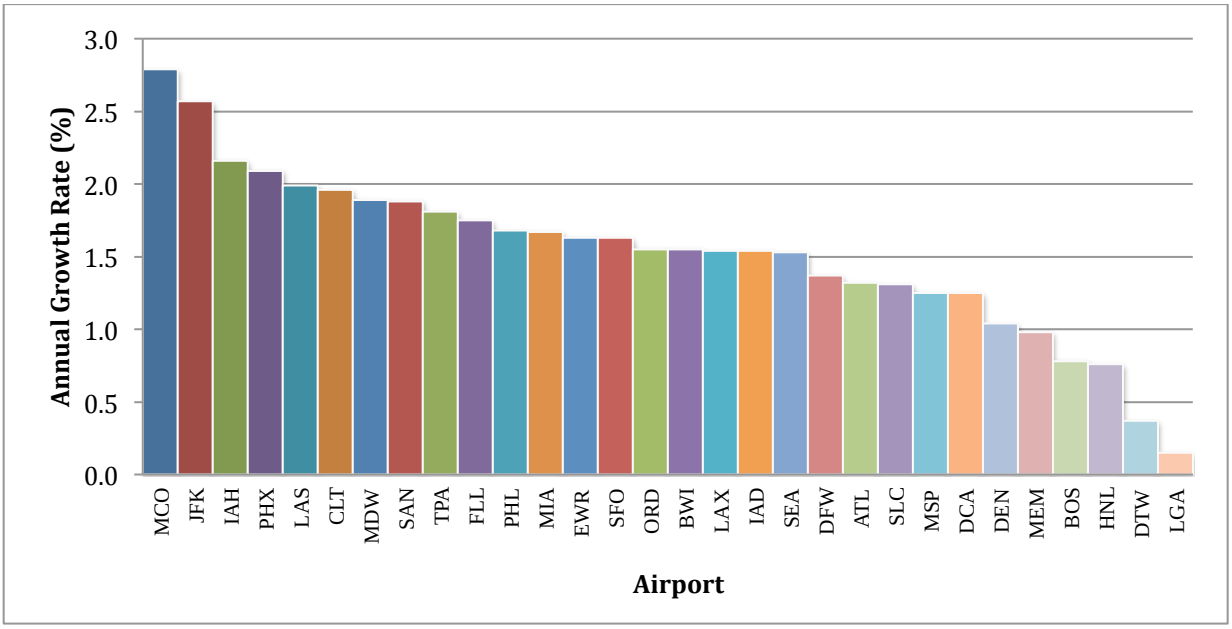


Figure 1 Airport Operations Growth Rates for the Core Airports – 2011 to 2040 (FAA 2012b).

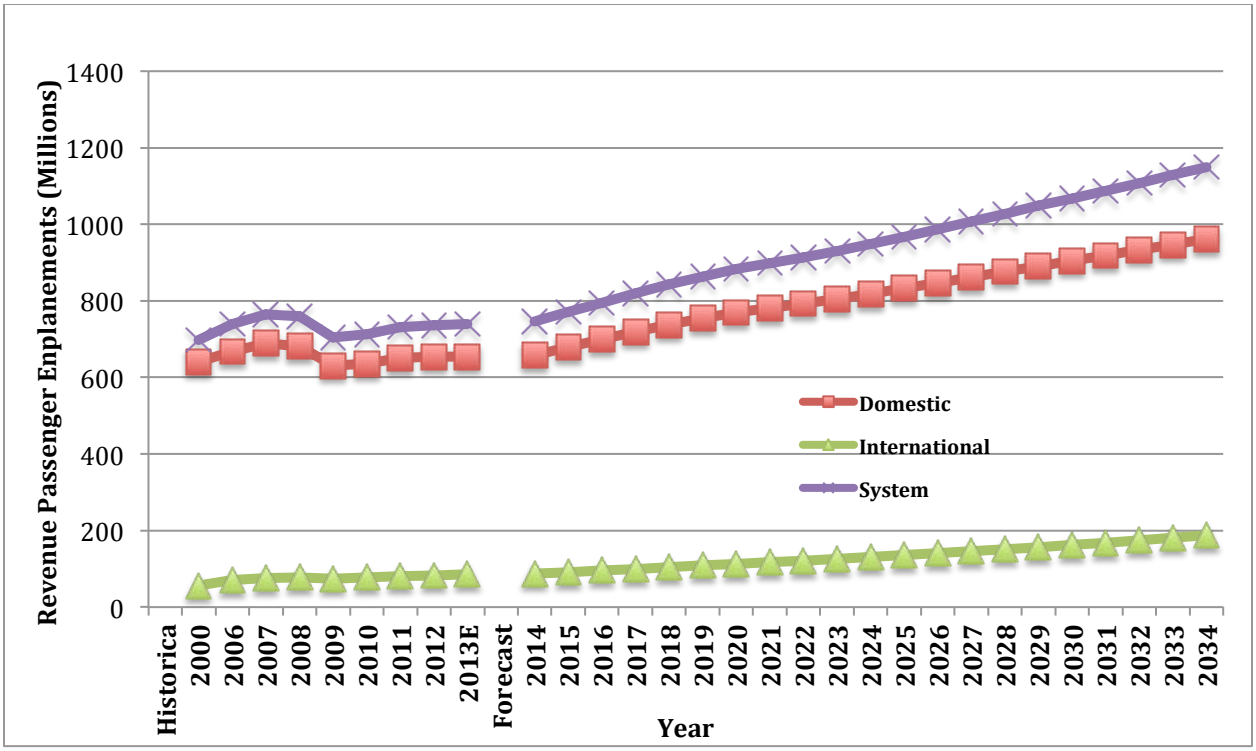


Figure 2 Revenue Passenger Enplanements (Historical data and Forecast Data) (FAA 2012b).

Wake turbulence, is a physical phenomena associated with the lift generated by the aircraft wings. Wakes limit the separation distances between successive flights arriving or departing a runway. Moreover, wake turbulence generated by large, heavy and super-heavy aircraft types is very strong and can be a potential hazard to trailing flights. In severe wake encounters, the result has been a temporary loss of control of the trailing aircraft. The FAA has published strict separation standards to prevent these hazards (FAA 2012a). As more aircraft types are introduced into the NAS, the separation standards need to be revised to ensure safety.

Without any improvement in the separation distance used today, other NextGen technologies will have a less significant role at improving the operational capacity of the system. Since wake turbulence cannot be eliminated, more research is needed to estimate potential wake interactions between aircraft in the future NextGen system. One of the stated goals of NextGen is to reduce

in-trail separations in the future. This objective can only be achieved through a better understanding of wake vortex interaction with aircraft.

1.2 Research Motivation

Eurocontrol performed a capacity study called ATC-Wake scheme in 2005 to evaluate the capacity benefits at various European airports and to assess the safety and capacity improvements with tactical reduced separation operation (WakeNet2-Europe 2006). The concept enables air traffic controllers to apply reduced separation from standard International Civil Aviation Organization (ICAO) separations based on the wake vortex prediction. The results showed a modest increase in capacity when schedule the landing aircraft with a 2.5 nm separation, see Table 1 (WakeNet2-Europe 2006). Although the NextGen technologies can significantly improve air traffic, the wake turbulence separation seems to have the most impact on airport capacity.

Table 1 Predicted Capacity Benefits for the ATC-Type of Operation (WakeNet2-Europe 2006).

Operation	Runway Throughput [ac/hr]			Delay [min]		
	ICAO	ATC-Wake	Change	ICAO	ATC-Wake	Change
Single Runway Departure	37.8	38.6	2.10%	N/A	N/A	N/A
Single Runway Arrival	35.2	37	5.00%	3	2.15	-29%
CSPR-A (non-segregated)	35.2	35.7	1.50%	3	2.74	-8.50%
CSPR-A (segregated)	35.2	37.6	6.90%	3	1.84	-38.70%
CSPR-A (semi-segregated)	35.2	35.8	1.60%	3	2.73	-9.90%

To better understand the wake turbulence behaviors under various conditions, there have been other significant efforts in vortex modeling, simulation and visualization. NASA's Aircraft Vortex Spacing System project started in the late 1990s, with the goal to predict and detect the wake behavior using light detection and ranging (LIDAR) technology. Since then, multiple models have been developed to simulate aircraft wakes. These models attempt to predict and

visualize wake behavior for a given set of flight parameters. These simulators, such as WakeScence (Holzapfel et al. 2009) or TASS (Proctor, Hamilton, and Switzer 2006), require information about aircraft characteristics and atmospheric conditions. Important aircraft characteristics include aircraft wingspan, flight speed, weight, wind speed, wind direction, atmospheric turbulence and air temperature. Some of these, including vehicle ground speed, wind speed and direction, can be detected with ground radar. Other parameters, such as aircraft mass and atmospheric turbulence have to be estimated indirectly. Moreover, most of the flight parameters exhibit substantial variation along the flight path. Such variability restricts the use of many wake models to understand wake interactions between many flights observed at an airport.

Modeling wake turbulence behaviors can also be very useful in many other concepts aimed at reducing aircraft separation standards. For example, models can be used to compare and evaluate the vortex strength of different aircraft types at given distance. This evaluation can help re-categorize aircraft groups that fit into a common hazard boundary. This is important for newly designed aircraft types. An important motivation in the development of wake prediction models is to predict the wake position of aircraft in real time and their integrating in the current air traffic control system to help avoid potential wake encounters. This is important if dynamic separation standards are to be adopted in the future. Both ICAO and FAA are promoting dynamic separations in the near future.

To improve the prediction capability of wake models, some include random effects associate with atmospheric turbulence and aircraft performance uncertainties. A basic wake simulation model presents the nominal wake characteristic associated with a single instance of a probabilistic model. An enhanced model with combine multiple instances of random wake

effects and represent such events as a polygon around the predicted vortex core positions representing the range of possible positions within a level of uncertainty (Holzapfel 2003b).

Inspired by all the existing wake vortex models, we evaluated some of the most commonly used network models and developed a user friendly, easy-to-use wake vortex model. : 1) to produce three-dimensional potential wake hazard zone based on aircraft characteristics for a variety of atmospheric parameters, 2) to integrate flight track data and perform wake potential encounter analysis, and 3) to study the potential wake encounters for nation-wide flight track data to estimate impacts of wakes.

2 Literature Review

The study of wake turbulence as an operational concern dates back to the early 1970s with the introduction of the Boeing 747-100 to the airspace system. The FAA has developed sensors to collect operational data and measure wake vortex behavior over time. However, it was not until recently, that the FAA started the wake turbulence program and started working on mitigating the adverse effects of wake turbulence (J. A. Tittsworth 2012).

2.1 NextGen and Re-Categorization

The Next Generation Air Transportation System, commonly known as NextGen, "is the transformation of how airplanes traverse the sky", according to the FAA (Government Accountability Office 2006). NextGen attempts to modernize the current air transportation system that relies on radar technology for surveillance and replace it with satellite-based technology surveillance system. In 2003, the Joint Planning and Development Office (JPDO) was created to manage work associated with the transformation of the air transportation system (Government Accountability Office 2006). By implementing new technologies, policies and air traffic control procedures, NextGen is expected to deliver a better flight experience by reducing delays, saving fuel and lowering aircraft emissions (FAA 2013a). NextGen technologies and procedures have been integrated to the current system. As of Feb 2014, 658 Automatic Dependent Surveillance-Broadcast (ADS-B) stations have been installed over the nation, providing traffic and weather information to all aircraft and air traffic control centers with equipped receivers. Wake vortex studies, are an important part of NextGen to help maximize the capacity potential of new technologies. The FAA has identified four research areas on wake vortex research that involves both technology and procedures.

1) Separation standards for closely spaced parallel runway approaches (FAA 2012a), 2) Wake Turbulence Mitigation for Departures (WTMD), 3) Wake Turbulence Mitigation for rrivals (WTMA), and 4) Wake Turbulence Re-Categorization. The 7110.308 Order enables the use of the dependent 1.5NM diagonal separation on Closely Spaced Parallel (CSPR) runways during instrument approaches with Large and Small aircraft as leaders in the pairing sequence. WTMD and WTMA allow closely parallel runways to process operations without reliance on wake vortex separations under certain crosswind conditions. Figure 3 shows the benefit of WTMD. In the figure, the wind blows the wake away from the parallel runway operation. This allows the trailing aircraft to takeoff without the standard wake vortex separations enforced for closed parallel runways.

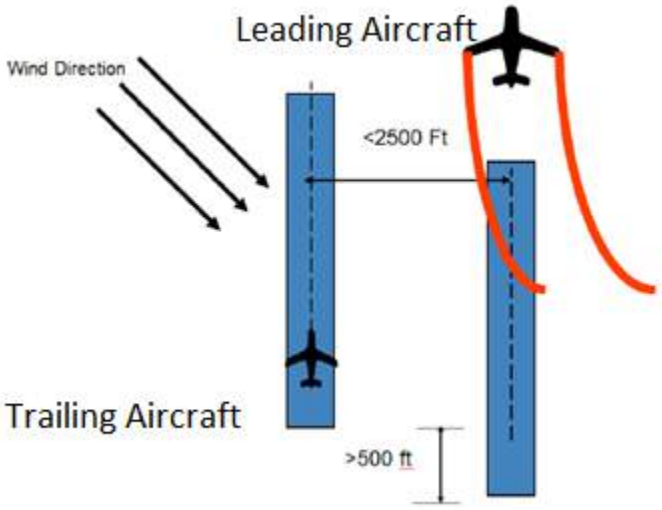


Figure 3 WTMD Concept Illustration (J. A. Tittsworth 2012).

This concept allows the upwind runway to perform takeoff at a reduced wake separation distance, or in some cases, without any wake separation constraints. The WTMD concept requires both real-time wind data collection and predictive knowledge of the wake turbulence behavior. This concept allows dynamic separation based on real weather conditions, allowing the airport to achieve higher capacity under such conditions. Few airports have experimented

with the WTMD concept before the national implementation. These include: the General Edward Lawrence Logan International Airport (BOS), Detroit Metropolitan Wayne County Airport (DTW), Newark Liberty International Airport (EWR), George Bush Intercontinental Airport (IAH), Memphis International Airport (MEM), Miami International Airport (MIA), Philadelphia International Airport (PHL), Seattle-Tacoma International Airport (SEA), San Francisco International Airport (SFO), and Lambert-St. Louise International Airport (STL).

Wake Turbulence Re-Categorization, or commonly refer as ReCat, is an ongoing three-phase effort to revise the current aircraft in-trail separation standards. The current in-trail separation standards are based primarily on weight with five wake categories published by FAA, as shown in Table 2 (FAA 2013b). ReCat Phase I, is the result from a joint effort of the FAA and EUROCONTROL. Phase I ReCat standards were published in 2011, replacing the old five wake groups with a new six group system, as shown in Table 3.

Table 2 Current FAA Wake Separation Standards (at the Threshold) (FAA 2013b).

		Follower (Nautical Miles)				
		Super	Heavy	B757	Large	Small
Leader	SuperHeavy	2.5*	6	7	7	8
	Heavy	2.5*	4	5	5	6
	B757	2.5*	4	4	4	5
	Large	2.5*	2.5*	2.5*	2.5*	4
	Small	2.5*	2.5*	2.5*	2.5*	2.5*

* MRS: Minimum Radar Separation (3 NM, or 2.5 NM when existing requirements are met)

Table 3 Separation Matrix (in nautical miles) for ReCat I (J. A. Tittsworth 2012).

		Follower (Nautical Mile)					
		A	B	C	D	E	F
Leader	A	MRS	5.0	6.0	7.0	7.0	8.0
	B	MRS	3.0	4.0	5.0	5.0	7.0
	C	MRS	MRS	MRS	3.5	3.5	6.0
	D	MRS	MRS	MRS	MRS	MRS	5.0
	E	MRS	MRS	MRS	MRS	MRS	4.0
	F	MRS	MRS	MRS	MRS	MRS	MRS

* MRS: Minimum Radar Separation (3 NM, or 2.5 NM when existing requirements are met)

The goal for ReCat three-phase analysis is eventually to allow dynamic separation standards between pairs of aircraft. The static pairwise standard will provide a separation matrix with each combination of leader and follower optimized for wake turbulence. Based on a recent study conducted by Volpe the top 95 airports of the world handle 99% of today's global air traffic (Hallock 2013).

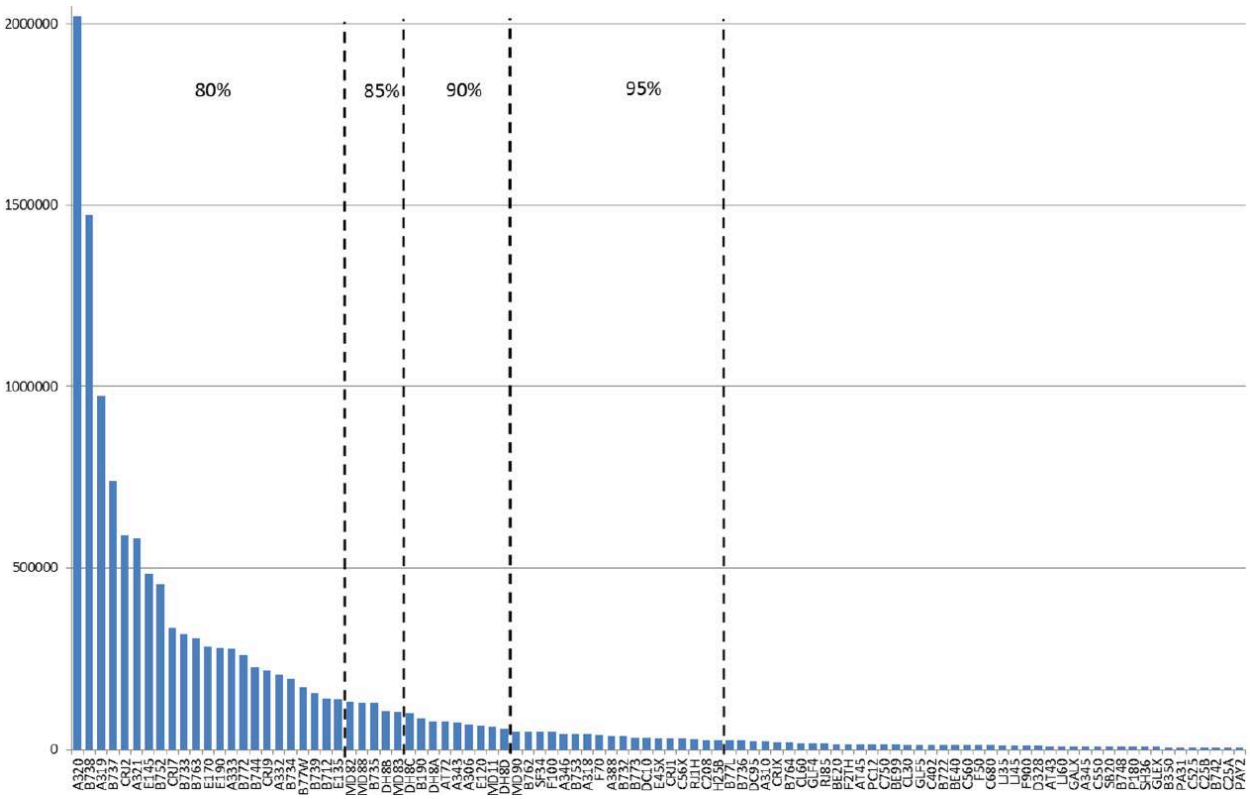


Figure 4 Top 99% of 2012 Global Traffic, 99 ICAO Types.

Moreover, 99% of the air traffic can be covered with 99 aircraft types, ranging from the Airbus 380 to smaller aircraft, like the Piper Cheyenne 2. With the static separation concept introduced with Re-Categorization, the proposed new separation matrix will be as large as 99 by 99. Weather conditions will also be considered and imbedded in the table during ReCat Phase III. In order to complete the table, significant more work is needed to evaluate the individual wake behavior for 99 aircraft. The practical implementation of such large matrix requires further study. Air traffic controllers will need a decision support.

2.2 Aviation Safety Report System (ASRS)

The Aviation Safety Report System is a NASA supported database that collects aircraft incident information supplied by pilots, air traffic controllers, cabin crew, and others. These reports are submitted voluntarily classify incidents according to severity. The ASRS system was established in 1976 and has provided human performance related information in the NAS for near four

decades. Initially, wake turbulence encounters were not included as reportable events. However, wake vortex encounters were added as an event type in 1998. Figure 5 shows the number of reports recorded as wake vortex encounter in the past 15 years. Since 2006, the system started a data collection at three major airports: Lambert-St. Louise International Airport (STL), John F. Kennedy International Airport (JFK) and San Francisco International Airport (SFO), and published their first Supplemental Question Set (SQS). The latest SQS was updated in June 2013, and contains more focused questions regarding wake encounter events. Questions are now divided into four sections, with detailed information for further analysis: wake-encounter aircraft information, flight phase information, wake producing aircraft information and consequences. Since the beginning of the ASRS SQS, the system has received 225 complete wake vortex reports. Based on these reports, NASA concluded that over 86% of the vortices encountered were generated by aircraft types in the Heavy category, which are Category D in the ReCat Phase I classification. This suggests that research should focus on wakes produced by larger aircraft types.

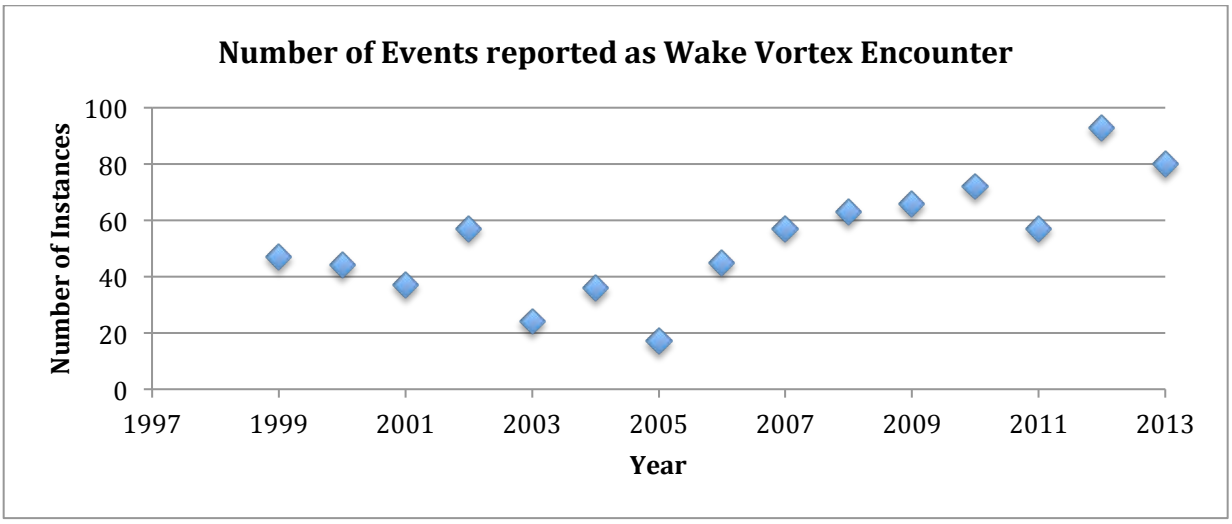


Figure 5 Wake Encounters Reported in the ASRS System from 1998 to 2013.

2.3 Wake Turbulence Models

An aircraft wing generates lift while flying. The generation of lift also generates two wakes, a pair of counter rotating vortices, one behind each wing. The initial spacing between the two vortices is about 78% of the wingspan (Shortle 2007). These wakes can be hazardous to a trailing aircraft, especially if the trailing aircraft is lighter than the leading one and if the distance between the aircraft is small. In general, the larger the aircraft, the more powerful the wake they create, since the heavier weight of the large aircraft and the longer wingspan, contribute to the wake vortex generation.

Figure 6 Wake Vortex (EUROCONTROL 2014b).

Current FAA rules mandate the aircraft must maintain a minimum distance from other aircraft ahead of them to avoid wake turbulence. The FAA has been working on wake research since 1970. Multiple wake vortex research efforts have been conducted in the past decades. To enhance the understanding of wake vortex behaviors, it is essential to have good wake vortex models. Simulation and visualization models can also provide sufficient information needed to quantify the benefits of dynamic separations to be introduced in future concepts of operation. In the following paragraphs we introduce four most popular wake vortex models developed in the last decade.

2.3.1 Sarpkaya Model

A vortex decay model predicts the descent behavior of the wake vortices considering real environmental conditions. The model was developed using LIDAR field data collected at Memphis and Dallas-Fort Worth airports. The model was built based on a wake vortex behavior model by originally developed by Greene in 1986. The model included atmospheric turbulence, crosswind, and in-ground-effect variables (Greene 1986). Figure 7 shows representative results using this model. The figure compares the difference between the predicted wake behavior in lines with the actual LIDAR data in circles and crosses. As shown in Figure 7, the model predicts the circulation decay and the vertical descent of the wake well, but not sufficient in the lateral expansion of the wake position.

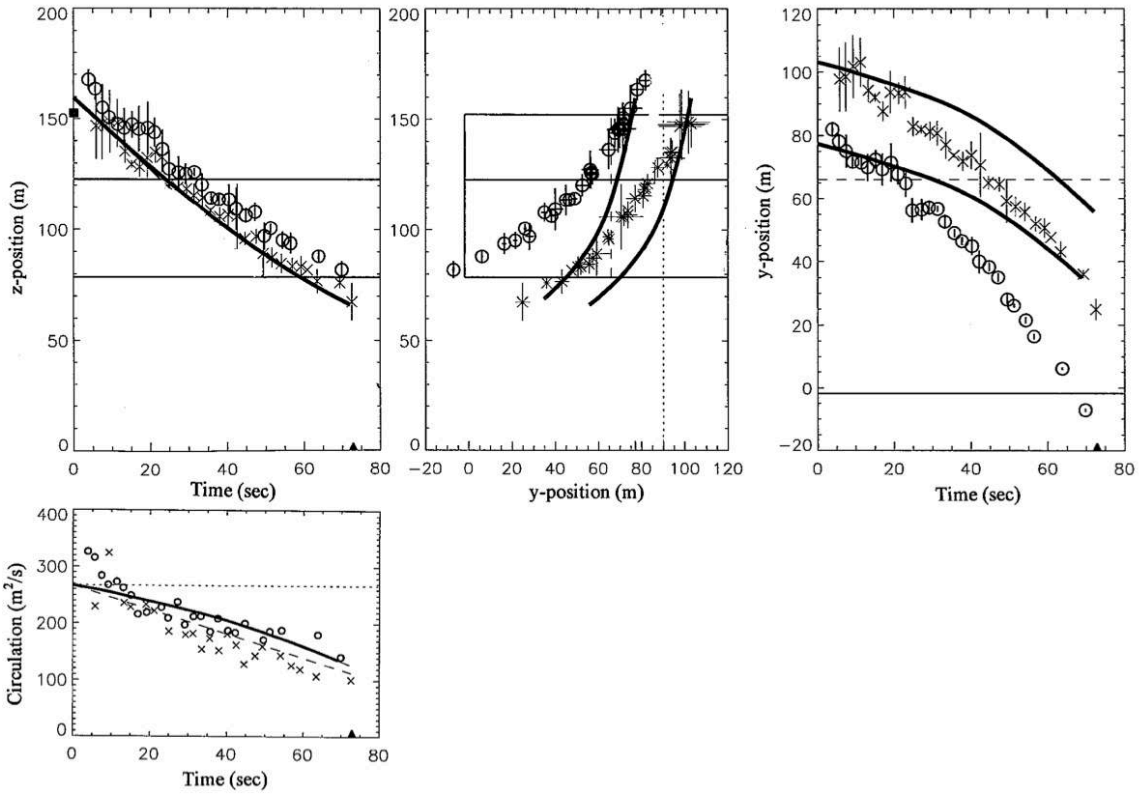


Figure 7 Comparison of the Predictions of the New Model with the LIDAR Data for a Boeing 727-100: O port vortices, × starboard (Greene 1986).

2.3.2 P2P Model

Frank Holzapfel published a probabilistic two-phase wake vortex decay model in 2003, known as the P2P model (Holzapfel 2003b). Before the development of this model, all the wake vortex prediction models were deterministic. The P2P model considered aircraft parameters including aircraft configuration, wind effects, atmospheric turbulence, atmospheric stratification and proximity to the ground. To illustrate the uncertainties in the environmental conditions, Holzapfel introduced probabilistic components formulated related factors into a normalized form, where all the characteristics are scaled to the initial vortex status. In addition, the model simulated the vortex decay process in a non-linear function with two phases, a slow decay followed by a rapid decay.

The output of the model contains both upper and lower boundaries of the potential vortex position, as shown in Figure 8. The results showed that the boundaries produced by the model include most of the field data successfully.

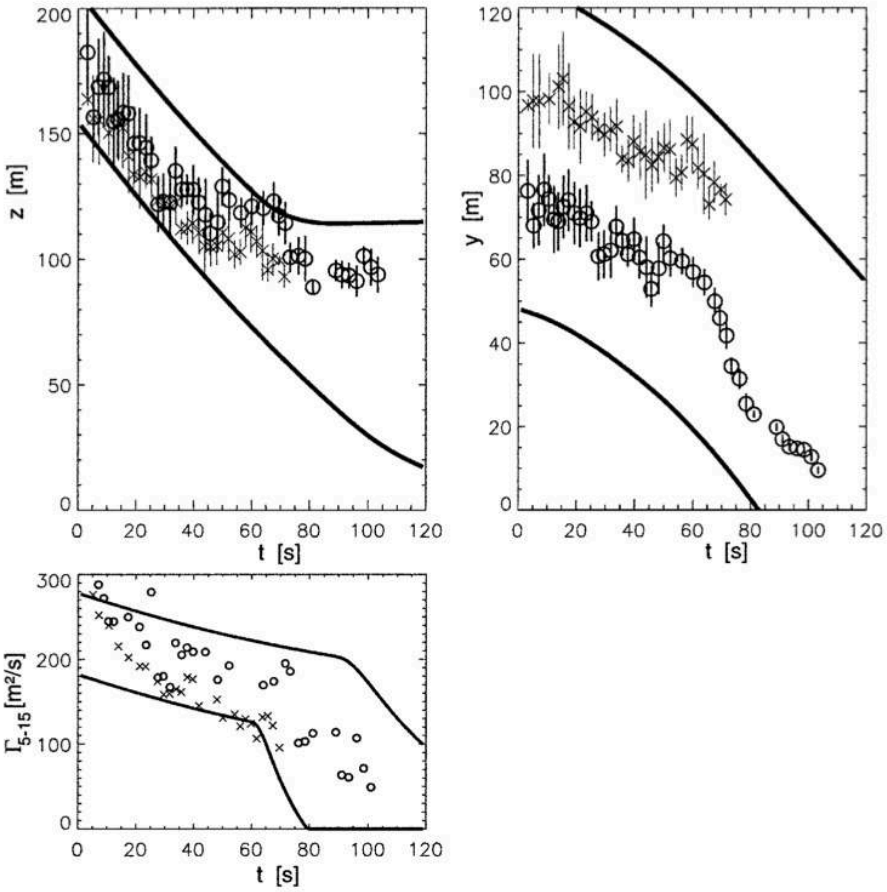


Figure 8 Comparison of P2P Predictions with LIDAR Observations with Error Bars for Memphis International Airport Field Data (Holzapfel 2003b).

2.3.3 TDAWP Model

The TASS (Terminal Area Simulation System) Driven Algorithms for Wake Prediction Model, known as TDAWP model, is a wake vortex model to predict wake decay and transport behaviors. The model was developed by NASA Langley Research Center. Compared to other wake vortex models, this one predicts wake descent behavior and circulation decay with separation algorithms. Earlier studies by F. Proctor showed that wake vortices decent rates have a non-uniform relationship to vortex radius and thus different decay rates (Proctor, Hamilton, and Switzer 2006). The circulation algorithms for both decay and vertical descent behavior of the wake increased the model accuracy. Furthermore, the model divided the wake decay procedure

into two phases based on the vortex core evolution process observed in Large Eddy Simulation (LES) models (Proctor, Hamilton, and Switzer 2006).

The model has been validated and calibrated with Memphis International Airport field data obtained during the Aircraft Vortex Spacing System (AVOSS) in 1995. The VOSS program collected data for different aircraft types under various atmospheric conditions. A comparison of the TDAWP results is shown in Figures 9 and 10. The field data supported the two-phase model showing the vortex decay rate is increasing with time. The results also showed a relative close match of the vertical drop of the vortex cores. However, the model failed to predict the vortex vertical behavior after a certain time period, as shown in Figure 10, due to the strong crosswind shear (Proctor, Hamilton, and Switzer 2006).

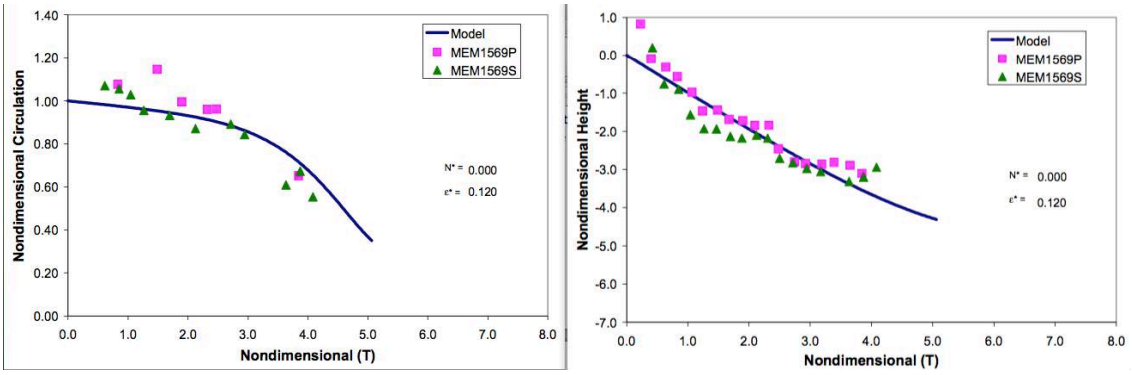


Figure 9 Comparison between TDAWP Predictions and Memphis Field Data (McDonnell Douglas DC – 9) (Proctor, Hamilton, and Switzer 2006).

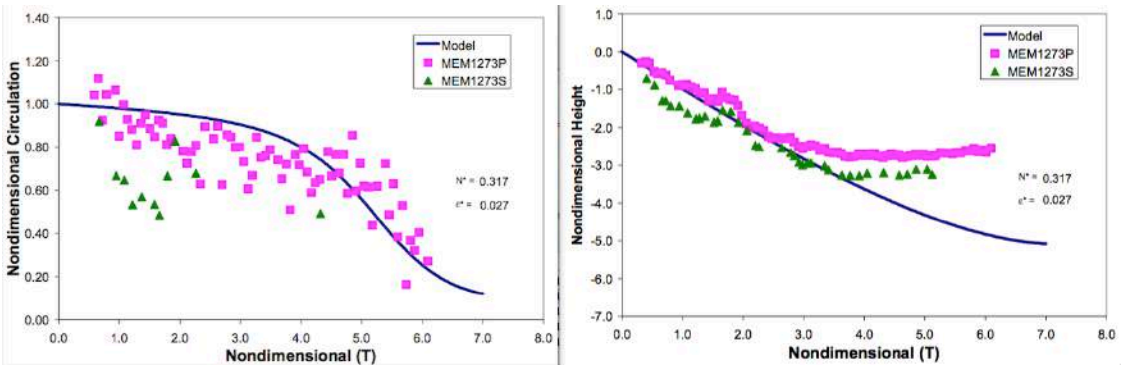


Figure 10 Comparison between TDAWP Predictions and Memphis Field Data (Aircraft Type DC-10-30) (Proctor, Hamilton, and Switzer 2006).

2.3.4 APA Suite of Wake Models

NASA Langley Center developed the APA Suite of Models integrating the most suitable vortex models into a hybrid model set. When the wake is out-of-ground (OGE) effect, the model allows the user to choose from three candidate models, Sarpkaya, TDAWP and Deterministic 2-Phase model (D2P), as shown in Figure 11. For in-ground-effect (IGE), the model uses a common IGE module derived from a TASS LES study (Proctor 2009).

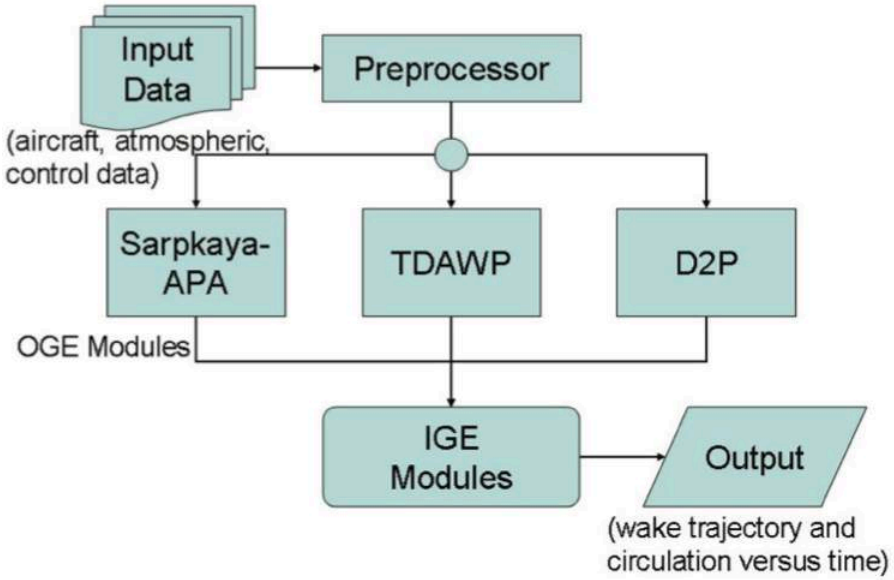


Figure 11 Architecture of APA 4.00 Suite (Proctor 2009).

The APA Suite provides flexibility to select from different OGE models and different environment variables. Figures 12 and 13 show some of differences and similarities between the results from the three OGE models in the APA Suite.

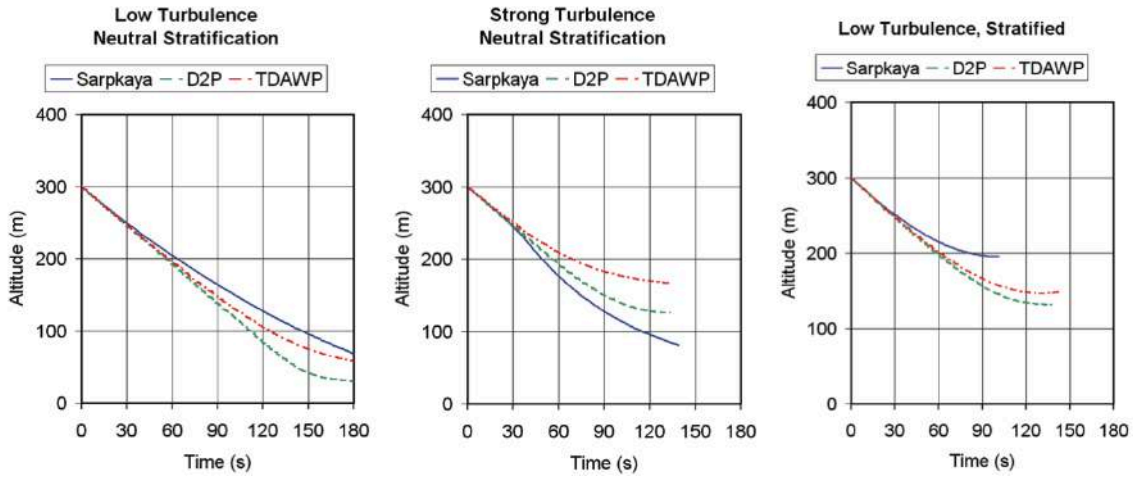


Figure 12 Altitude vs. Time Comparison between OGE Models (Proctor 2009).

All the models have similar altitude descent profiles during the first half of the wake lifetime. The greatest difference in the three models is evident under strong turbulence. The wake descends at a faster speed with the Sarpkaya model prediction under strong turbulence. The same model shows modest descent rates under low turbulence. The Sarpkaya model shows a different circulation prediction behavior compared to the other two models. Both two-phase decay models show a slower decay phase followed by a rapid decline in circulation levels. For circulation behavior, the Sarpkaya model shows a more convex shape for the wake decaying process, seen Figures 12 and 13.

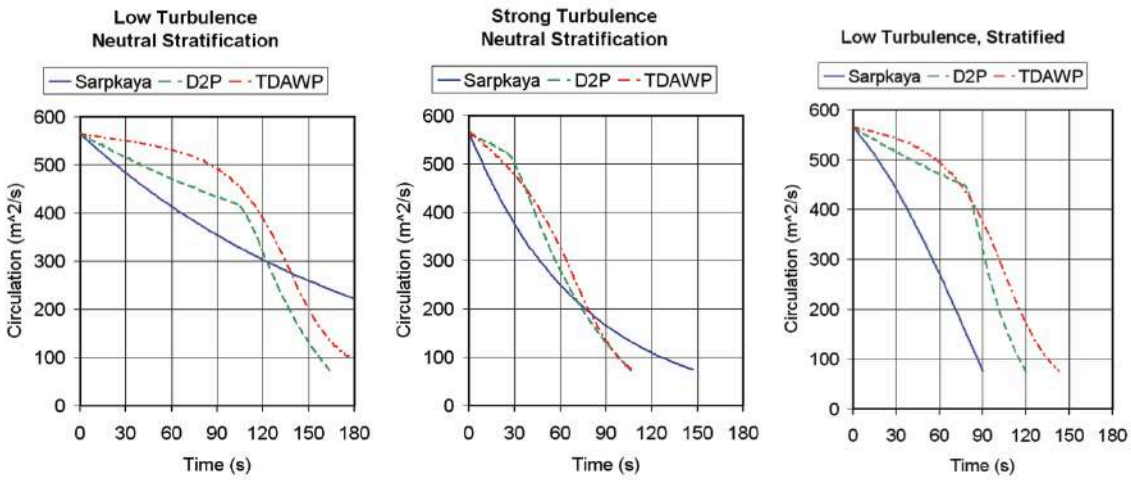


Figure 13 Circulation vs. Time Comparison between OGE Models (Proctor 2009).

To further evaluate the differences between models in the APA suite, a more explicit statistical analysis was conducted using Root Mean Square (RMS) values. Using a 24-hour period arrivals LIDAR data with aircraft at Denver International Airport (DIA), the model was evaluated. The summarized results for all 173 aircraft arrival cases observed on the day of the experiment are shown in Table 4. The lateral position prediction is close across models based on the RMS value. The APA-TDAWP model has the least vertical error in the altitude prediction metric.

Table 4 Statistics for Differences between Model and LIDAR Observations for a 24-hour Period at Denver International Airport (173 cases) (Proctor 2009).

Model	Average	RMS Lateral (m)	RMS Vertical (m)
Starboard Vortex			
APA - Sarpkaya	Medium	21	14
	90th percentage	49	29
APA – TDAWP	Medium	21	10
	90th percentage	48	24
APA – D2P	Medium	21	12
	90th percentage	50	24
Port Vortex			
APA – Sarpkaya	Medium	19	14
	90th percentage	40	29
APA – TDAWP	Medium	19	12
	90th percentage	42	29
APA – D2P	Medium	19	12
	90th percentage	38	26

3 Enhanced Wake Vortex Model

3.1 Introduction

Wake prediction fast-time simulators require information such as aircraft and environmental parameters. Aircraft data can be derived using radar data. Aircraft mass and atmospheric turbulence have uncertainty, because the aircraft landing or takeoff mass is only known to pilot. Such variability restricts the use of models like TDAWP Model, Sarpkaya Model, or the APA Suite of Models. The developed model in this dissertation, called EWEM, attempts to minimize the influences of these factors and present a practical model can be used to estimate potential wake encounters considering probabilistic estimates of the aircraft wake zone.

3.2 Objective

The EWEM model employs the TDAWP algorithm in APA to build a three-dimensional wake envelope for given set of flight conditions. These three dimensional wake envelopes are considered to be the protected wake zone that should be avoided by nearby aircraft. The wake zone depends on both the generator and the potential intruder. For example, a heavy aircraft would be able to penetrate a wake zone of a small aircraft with a mild flight upset effect. With a three-dimensional wake zone trailing behind an aircraft at each given flight position, the model can quantify potential wake encounters for a given set of flight paths. In other words, this model can help analyze and verify that new procedures such as close parallel runway operations will be “free” of wake conflicts. Up to know, flight procedures do not include such quality checks. This is perhaps the main contribute of this model.

3.3 Model Methodology

Using the deterministic results from the NASA APA Model, we developed an enhanced version of the Wake Encounter Model (EWEM). The EWEM model adopted the TDAWP algorithm using OGE analysis in the APA as the baseline case. In this new model, we add variability to the wake zone using the normalization concept introduced by Holzapfel in the TDAWP model. The stochastic parameters considered are: a) distribution of the aircraft mass, b) aircraft velocity, and c) atmospheric turbulence strength. The model performs a series of Monte Carlo simulations and constructs a three-dimensional zone with a simple, yet realistic, geometric structure trailing the aircraft as shown in Figure 14.

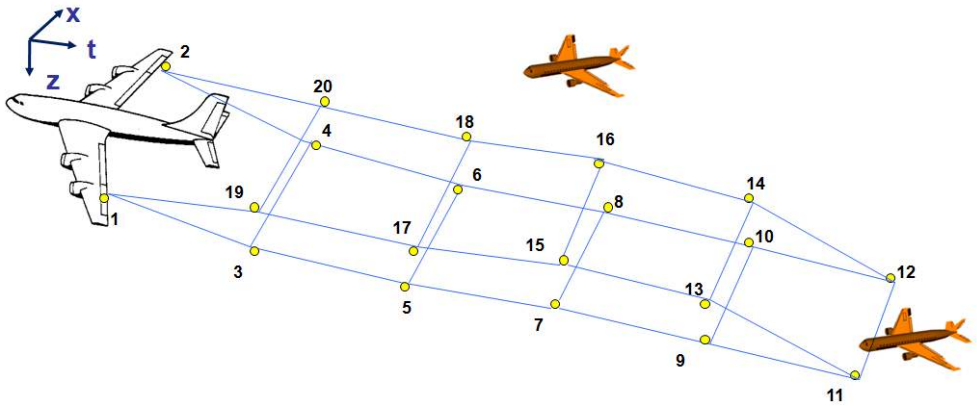


Figure 14 Generalized Wake Zone Framework.

The wake-zone boundary used in EWEM is defined by a series of control points that represent the potential wake vortex core positions behind an aircraft generator at various time frames after the wake has been generated. The control points bound the wake zone using both upper and lower bounds. The geometry of these control points is generated from a series of Monte Carlo simulations of the TDAWP model provided in the APA Suite developed by NASA.

The APA model relies upon the existence of four input files containing data as follows: 1) the cross-wind velocity profile (UDATA), 2) the environmental turbulence (QDATA), 3) the

environmental temperature profile (TDATA), and 4) the aircraft specification file (ACDATA). These variables are: 1) lateral position, 2) vertical height, 3) initial vortex descent rate (w_0) and 4) the initial vortex separation (b_0). Three environmental inputs use consist format. Each file starts with a single line header indicates the total number of records. The main body is a two columns matrix, which the number of rows defined by the header. The first column contains a list of altitude value, and the second column indicates the corresponding value for each altitude. Data between listed altitudes will be linear interpolated from the nearby values. For data that above the defined altitude range, the last available value will be applied. Table 6 shows some sample profile data contents (Switzer 2012).

Table 5 Sample Profile Data Contents in Table Format (Switzer 2012).

	Sample UDATA file (Cross-wind Velocity)		Sample QDATA file (Environmental Turbulence)		Sample TDATA file (Temperature)	
	Altitude (m)	Values (m/s)	Altitude (m)	Values (m^2/s^3)	Altitude (m)	Values (K)
Number of Records	**14**		**15**		**2**	
	0	0	0	0.00195095	0	300.62
	6.5	2.41	20	0.00186179	100	300.62
	7.5	2.47	40	0.0016832		
	11.5	2.65	60	0.00150461		
	12.5	2.7	80	0.00132629		
	13.5	2.75	100	0.00096745		
	14.5	2.79	120	0.00065471		
	15.5	2.85	140	0.00038781		
	16.5	2.9	160	0.00024345		
	17.5	2.95	180	0.00016081		
	18.5	3.01	200	0.00010563		
	19.5	3.07	300	0.00006991		
	100.5	3.12	400	0.00004618		
	200.5	3.12	130	0.00003045		
			140	0.00001973		

The initial vortex descent rate w_0 (m/s) required in the ACDATA file is defined in Equation (1).

$$w_0 = \frac{\Gamma_0}{2\pi b_0} \tag{1}$$

Where b_0 (m) is the wake initial separation, defined in terms of the aircraft wingspan and assumed to be $\pi/4$ times of the aircraft wingspan; Γ_0 is the initial circulation strength (m^2/s) determined by aircraft mass M (kg), air density ρ (kg/m^3), wingspan B (m) and aircraft speed V (m/s). The initial circulation (m^2/s) is calculated using Equation (2).

$$\Gamma_0 = \frac{Mg}{\rho sBV} \tag{2}$$

With sufficient input data provided, the APA generates the wake vortex core positions and circulation versus time in text format. Table 6 provides an example of the APA outputs for a given set of parameters for a Boeing 757-300 aircraft.

Table 6 Example of the APA Output in Table Format.

Time	Starboard Vortices			Port Vortices		
	Circulation Strength (m^2/s)	Lateral Position (m)	Altitude (m)	Circulation Strength (m^2/s)	Lateral Position (m)	Altitude (m)
0	-312.516	1.409	115.031	312.516	31.254	115.031
1	-311.666	-0.579	113.395	311.666	29.273	113.395
2	-310.811	-2.571	111.764	310.811	27.289	111.764
3	-309.949	-4.567	110.138	309.949	25.301	110.138
4	-309.081	-6.568	108.518	309.081	23.308	108.518
5	-307.342	-10.581	105.295	307.342	19.313	105.294

With the outputs generated from the model, we can form a single line trailing from the wake generator representing the wake behavior for both circulation decay and descent rate, as shown in Figure 15.

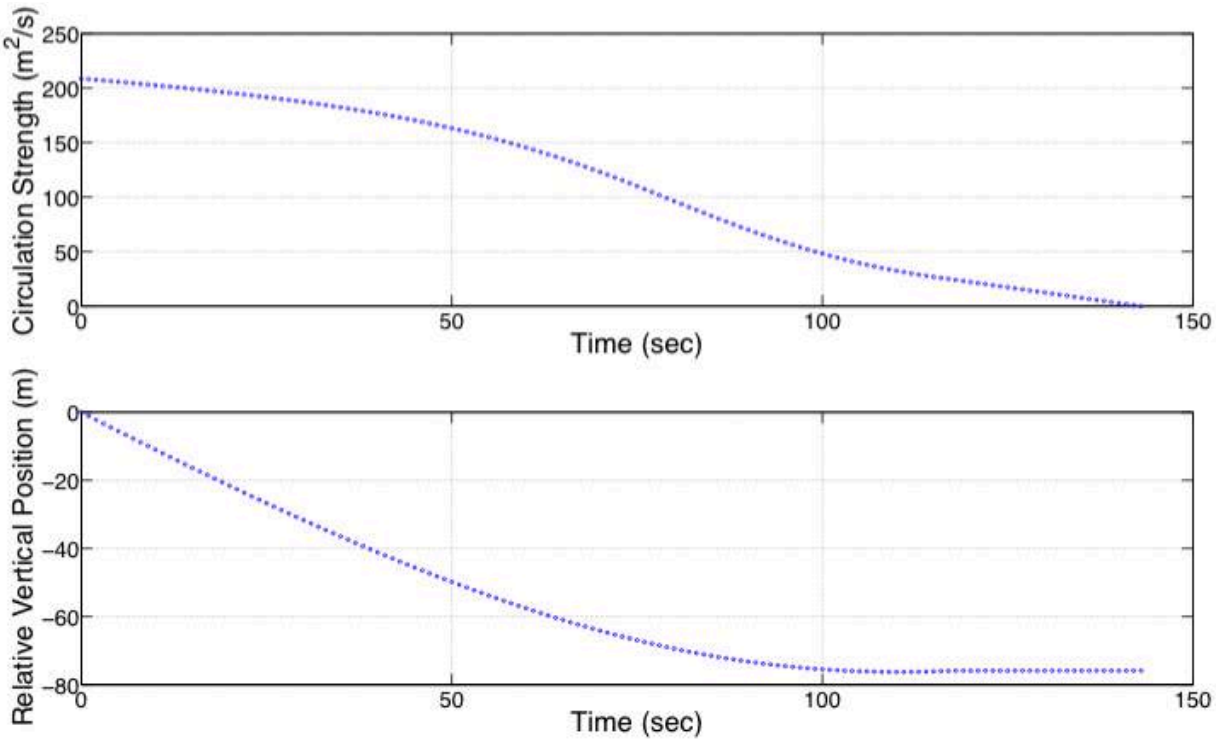


Figure 15 Single APA Run Output Visualization.

With fixed inputs for environmental and aircraft specifications, APA model provides a deterministic prediction of the wake core behavior. To build a three-dimensional hazard zone for

wake encounter analysis, we used Monte Carlo simulation to expand the single APA result and form a wake hazard envelope. Given as initial vortex descent rate defined in Equation (1), the parameters that affect the wake behavior are atmospheric environmental variables, the aircraft mass and the flight speed. To define a three-dimension wake zone, we used Monte Carlo simulation to estimate a set of possible combinations of aircraft mass and flight speeds consistent with an aircraft types.

We use the Base of Aircraft Data (BADA) (EUROCONTROL 2014a) database from EUROCONTROL to define feasible flight speed profiles. The database contains 116 aircraft types with typical flight profiles for climb, cruise and descent conditions as shown in Table 7.

Table 7 BADA Performance File for Aircraft Type Boeing 757-300.

BADA PERFORMANCE FILE Apr 01 2010

AC/Type: B753__

Source OPF File: Dec 19 2008
Source APF file: Mar 05 2009

Speeds:	CAS(LO/HI)	Mach	Mass Levels [kg]		Temperature: ISA
climb	- 250/300	0.78	low	- 77508	
cruise	- 250/300	0.79	nominal	- 101600	Max Alt. [ft]: 43000
descent	- 250/300	0.80	high	- 122500	

FL	CRUISE				CLIMB					DESCENT		
	TAS [kts]	fuel [kg/min]			TAS [kts]	ROCD [fpm]			fuel [kg/min]	TAS [kts]	ROCD [fpm]	fuel [kg/min]
		lo	nom	hi		lo	nom	hi	nom		nom	nom
0					173	2526	2070	1756	165.9	147	742	72.7
5					174	2512	2055	1741	164.6	148	758	72.1
10					175	2499	2041	1725	163.3	154	816	71.8
15					182	2588	2108	1780	162.6	165	946	71.7
20					183	2574	2093	1764	161.2	197	992	15.9
30	230	41.1	52.2	64.3	206	2917	2355	1981	160.9	230	1092	14.7
40	233	41.1	52.3	64.4	241	3332	2661	2227	161.5	233	1110	14.7
60	272	46.2	55.1	64.7	272	3659	2748	2181	158.7	272	1317	14.9
80	280	46.3	55.3	65.0	280	3542	2646	2085	153.4	280	1357	15.0

Aircraft weight varies according to the stage length flown by an arriving flight or the stage length to be flown by a departing aircraft. The BADA profile contains only low, nominal and high mass

boundaries. In order to find a distribution of aircraft weights, we generate a random aircraft mass distribution based on Equation (3) – (6), and explicit distribution factors for each group are listed in Table 8.

*Mean Departure Weight = 1.3*OEW + Dep_Mean *(MTOW-1.3*OEW);* *(3)*

Aircraft Mass = (Mean Departure Weight + R Mean Departure Weight * Dep_Std;* *(4)*

*Mean Landing Weight = OEW + Arr_Mean *(MALW - OEW);* *(5)*

Aircraft Mass = (Mean Landing Weight + R Mean Landing Weight *Arr_Std;* *(6)*

Where,

OEW: Operating Empty Weight (kg), gathered from BADA flight profile.

MTOW: Maximum Takeoff Weight (kg), gathered from BADA flight profile。

MALW: Maximum Allowable Landing Weight (kg), gathered from BADA flight profile。

Arr_Mean: Defined separately for each aircraft group, listed in Table 8.

Arr_Std: Defined separately for each aircraft group, listed in Table 8.

Dep_Mean: Defined separately for each aircraft group, listed in Table 8.

Dep_Std: Defined separately for each aircraft group, listed in Table 8.

R: Random Number between 0 and 1.

Table 8 Aircraft Mass Distribution Factors (unitless).

ReCat Group	Departures		Arrivals	
	Dep_Mean	Dep_Std	Arr_Mean	Arr_Std
F	0.8	0.05	0.8	0.1
E	0.8	0.05	0.8	0.1
D	0.8	0.03	0.6	0.05

C	0.85	0.03	0.5	0.05
B	0.85	0.03	0.4	0.05
A	0.85	0.03	0.4	0.05

With the factors set in Table 9, the aircraft will have distinct mass distribution during the departure and arrival phases. For example, the mass specifications for Boeing 747-400 class vehicles are listed in Table 10. With the distribution formula from (3) to (6), the mass distributions for Boeing 747-400 class vehicles are plotted in Figure 16 and 17.

Table 9 Weight Specifications for Boeing 747-400 Class Vehicles.

Operating Empty Weight (kg)	181726
Maximum Landing Weight (kg)	260366
Maximum Takeoff Weight (kg)	276696

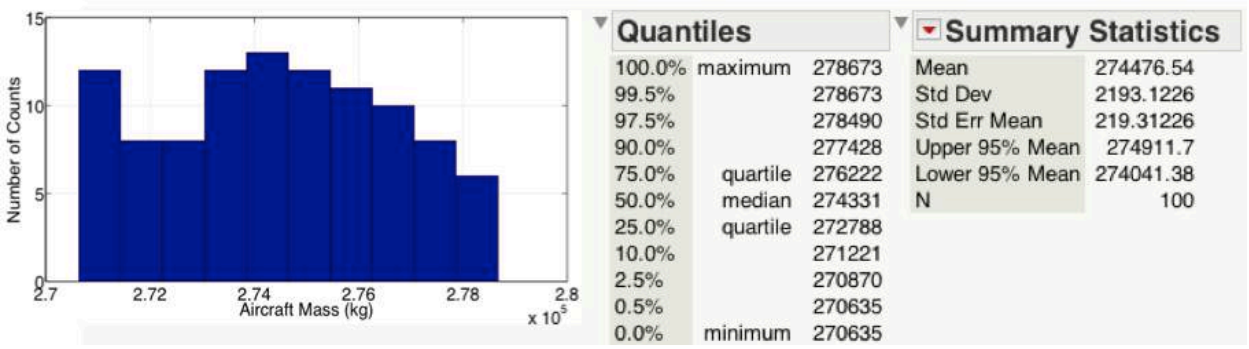


Figure 16 Boeing 747-400 Class Vehicles Mass Distribution for Arrival.

Figure 17 Boeing 747-400 Class Vehicles Mass Distribution for Departure.

Running the APA model for one hundred times at each profile state with different combinations in flight speed and aircraft mass, we can get a group of APA outputs for the selected aircraft type under all flight condition, with fixed environment. Figure 18 shows all the running results for Boeing 757-300 class vehicles with aircraft performance ranging from terminal area to cruise phase.

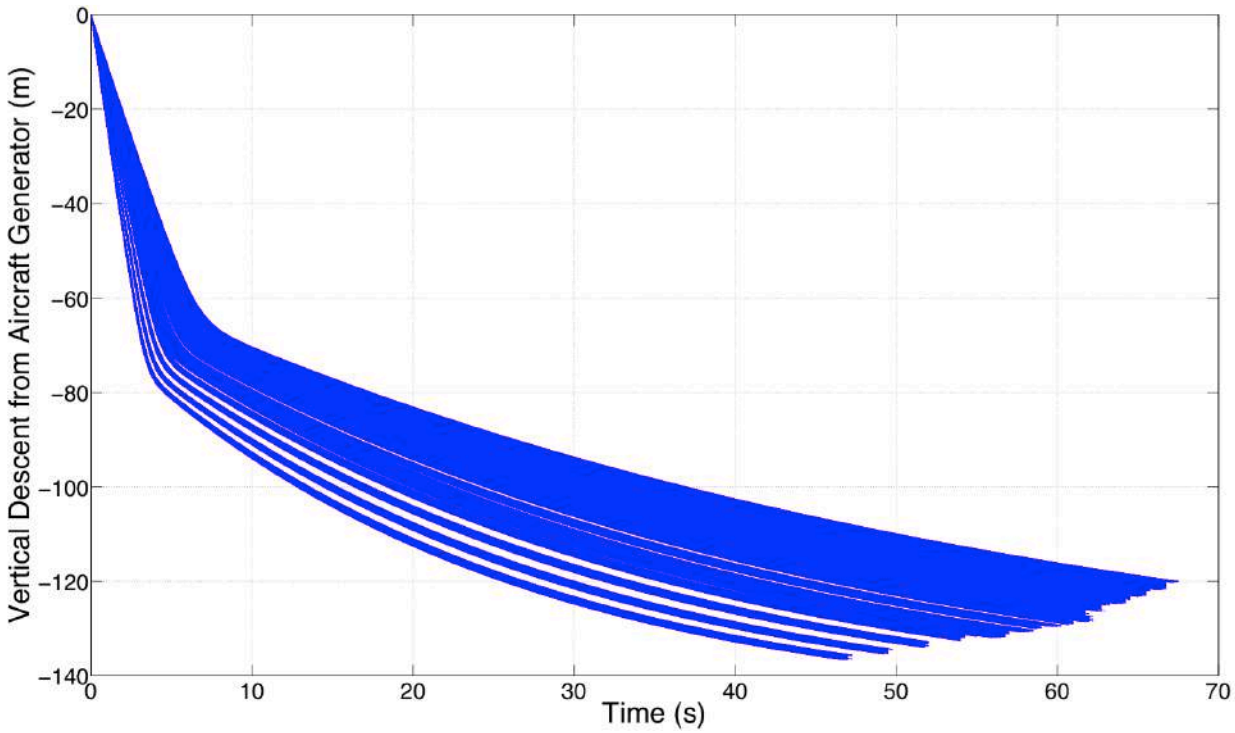


Figure 18 Sample Monte Carlo Simulation Results from APA Suite of Models for a Boeing 757-300 class vehicle with No Ground Effect under Intense Turbulence (Eddy Dissipation Rate = 0.015 m^2/s^3).

3.3.1 TDAWP vs. APA Model

The APA Suite of Models Version 4.51 developed by NASA contains few fast-time wake vortex algorithms, including: APA 3.2.5, APA 3.3, APA 3.4, TASS Driven Algorithm for Wake Prediction versions 1.0 and 2.1, and a hybrid of APA – TDAWP. For users outside of NASA, only the APA 3.2.5 and TDAWP 2.1 algorithms are available. We performed comparisons between the two algorithms under different atmospheric and flight conditions to select the most suitable model to build EWEM. The results are presented in Figures 19 and 20 for an aircraft similar to the Boeing 757-300 at an altitude of 610 m. Travelling at 175 knots under no wind conditions. Figure 19 shows the differences in wake circulation decay for two algorithms. The atmospheric turbulence values used in the analysis are shown in Table 11.

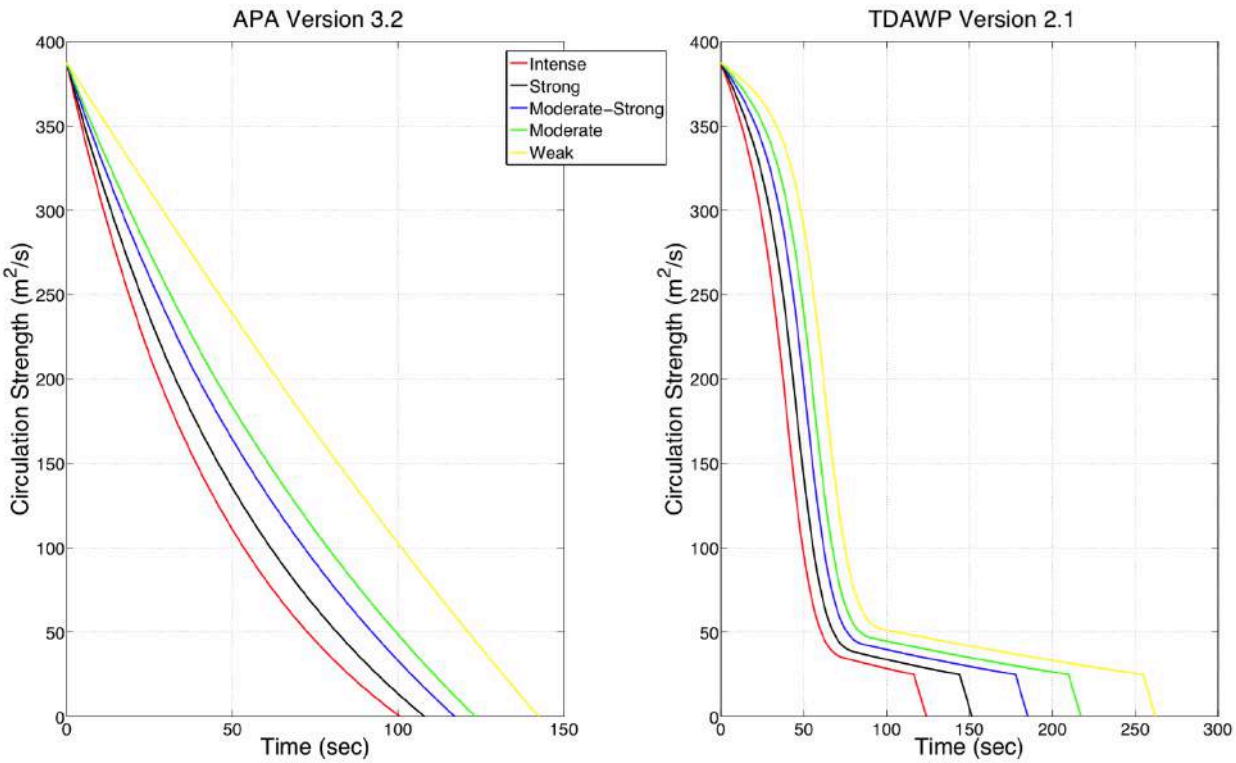


Figure 19 Circulation Decay Behavior Comparisons Between APA Version 3.2 and TDAWP Version 2.1 under Different Atmospheric Turbulence Strength.

As the atmospheric turbulence gets stronger, the wake vortices decay rate increases in both models. Comparing the consistent of decay rates in the APA model, TDAWP algorithm uses multi-phase prediction for the circulation decay. In TDAWP, wake decay is faster in the first phase and this follows with a steady phase transitioning to a linear phase after the circulation reaches a certain threshold level.

Figure 20 shows the differences in the wake vertical decay behavior for both algorithms. The TDAWP algorithm shows a consistent pattern: the stronger the environmental turbulence; the faster the wake drops. The APA model does not show a clear trend on the descent rate with increase turbulence strength. Similar behaviors have been observed for turbulence strengths defined from strong to moderate in the APA model.

The differences between the two models under In-Ground-Effect will be presented in Chapter 3.3.5. Based on the performance of two models under both In-Ground-Effect and Out-of-Ground-Effect, we use the TDAWP algorithm version 2.1 as the wake vortex generation engine to support the Enhanced Wake Encounter Model.

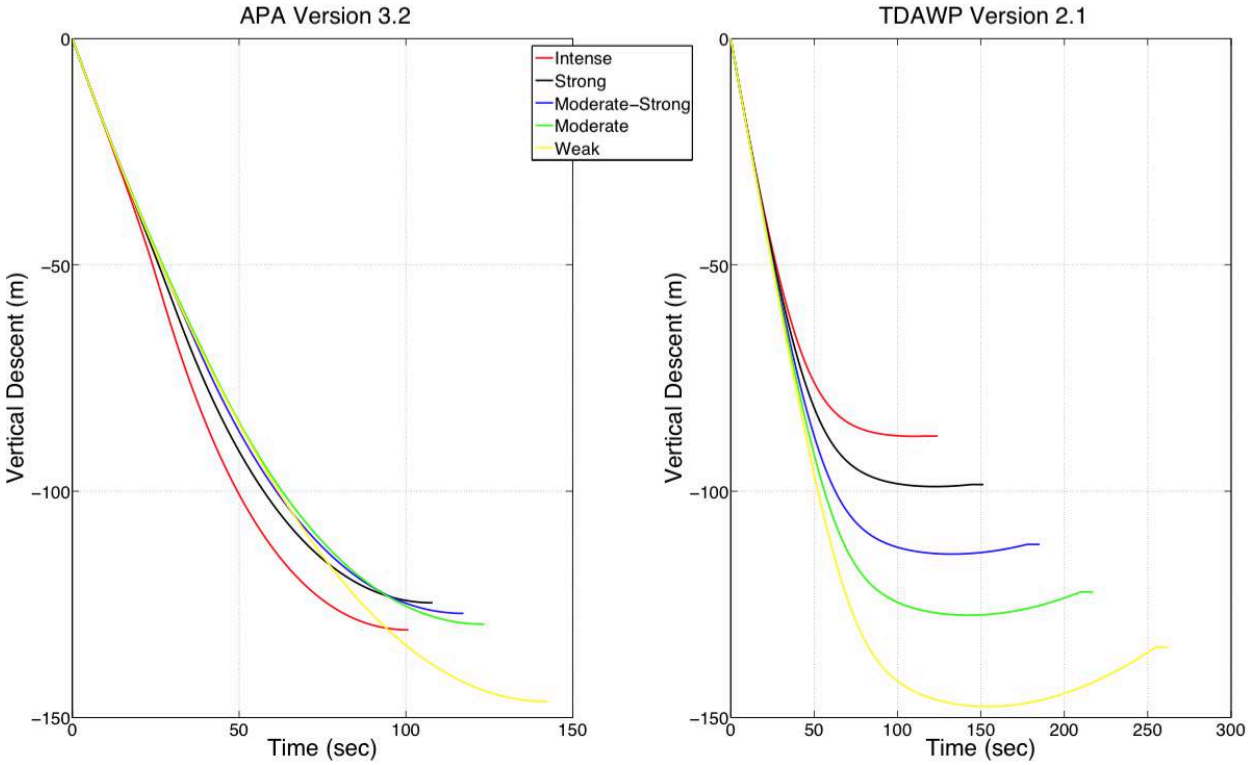


Figure 20 Wake Vertical Decay Behavior Comparisons between APA Version 3.2 and TDAWP Version 2.1 under Different Atmospheric Turbulence Strengths.

3.3.2 Re-Categorization

To create an accurate wake vortex model for all aircraft types, the ideal solution is to build individual envelopes for each aircraft type. Multiple aircraft types have a wide range of input parameters such as mass, speed, wingspan, etc. Moreover, a single aircraft flying in the NAS exhibits substantial differences in speed and mass conditions that result in variations of the wake zone. The Monte Carlo simulation explained in Chapter 3.2 is a time consuming process due to the wide range of aircraft performance values required to produce a three-dimensional wake

volume. Complete Monte Carlo simulation for all aircraft type will be not efficient. For this reason, we construct families of wake volumes for particular aircraft groups and perform Monte Carlo simulation for the most critical aircraft type in each group. This process generates a conservative wake volume for each aircraft group. These volumes are used in the analysis of potential wake encounters.

As mentioned in the Chapter 2.1, the FAA and the EUROCONTROL developed a new the ICAO in-trail wake separation standard to increase capacity while maintaining safety. This revision is called Wake Turbulence Re-Categorization, or ReCat. ReCat Phase I standards were published in 2011, introducing a six-category wake classes system to replace the current five wake class system employed by the FAA. The FAA published a simplified mapping scheme to assign 9000+ ICAO registered aircraft types to the ReCat groups based on both aircraft mass and wingspan, as shown in Figure 21 (J. A. Tittsworth 2012).

The Enhanced Wake Encounter Model employs the EUROCONTROL BADA model. We supplement the BADA model with aircraft information for each aircraft type, including the maximum takeoff weight, operating empty weight, maximum landing weight and wingspan. For 106 aircraft types in the BADA 3.9 model, we assigned each aircraft type to corresponding ReCat group using the scheme provided by the FAA. All information is stored along with assigned ReCat group in a table. This process was necessary because the FAA published the new ReCat groups for 61 aircraft types, as shown in Figure 22. Table 10 shows an example of the table containing the BADA aircraft mapping to the ReCat groups.

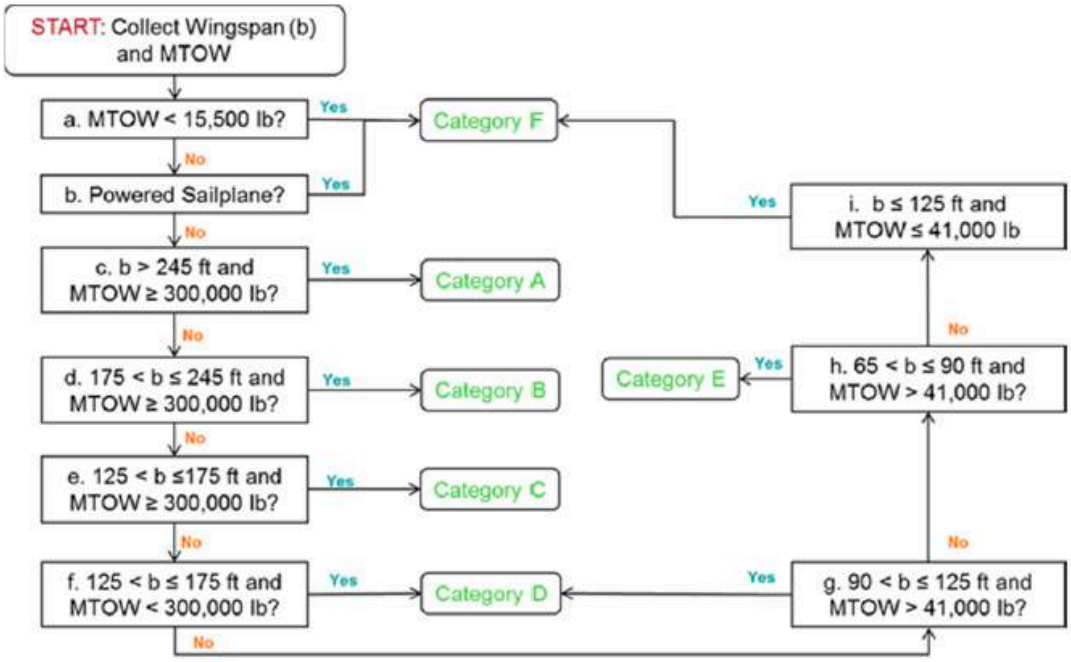


Figure 21 Generalized ReCat I Categorization Scheme (J. A. Tittsworth 2012).

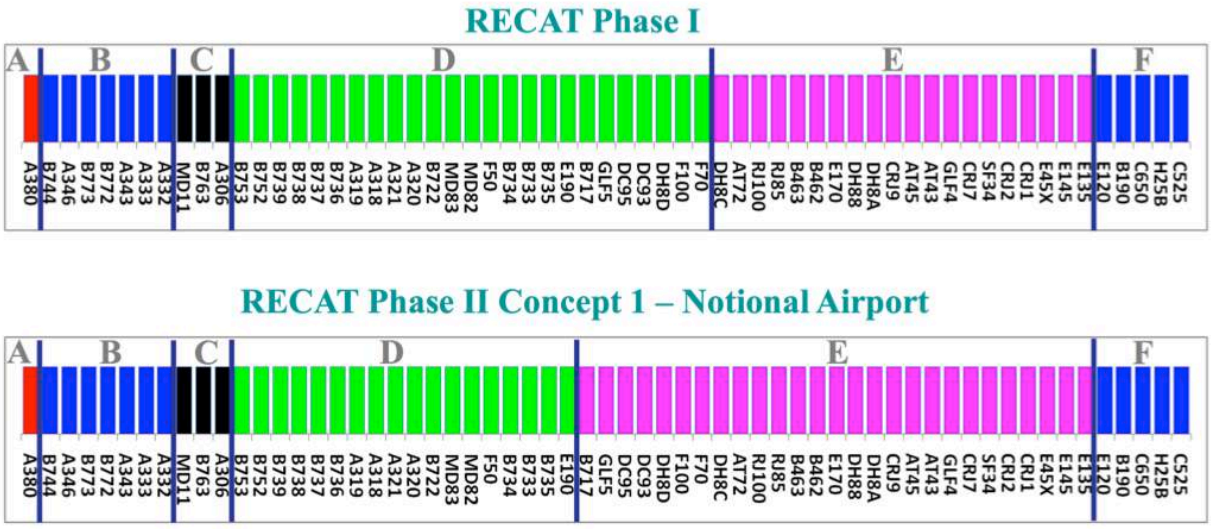


Figure 22 61 Aircraft Types with ReCat Group Presented by FAA on WakeNet Meeting (FAA 2012c).

Table 10 Partial Aircraft Information Table.

Key Field	Name	Manufacturer	Short Name	BADA Name	Wake Class	Operating Empty Weight (kg)	Maximum Landing Weight (kg)	Maximum Takeoff Weight (kg)	Wing span (m)	Wing Area (m^2)	ReCat Group
A388	A380-800	AIRBUS INDUSTRIE	A380-800	A388	Super-heavy	277014	386006	573009	80	845	A
A346	A340-600	AIRBUS INDUSTRIE	A340-600	A346	Heavy	177004	256004	380006	63	439	B
A310	A310-300	AIRBUS INDUSTRIE	A310-300	A310	Heavy	79207	124014	164022	44	219	C
A318	A318	AIRBUS INDUSTRIE	A318	A318	Large	38375	57501	68001	34	123	D

To improve the usability of the EWEM model, it is important to extend the BADA model list further. To do this we mapped 294 aircraft types that are commonly found in the RADAR data sets provided by the FAA and that constitute ingot files to our model.

3.3.3 Eddy Dissipation Rate

Eddy dissipation rate (EDR), is a commonly used parameter to quantify the strength of atmospheric turbulence. Ambient background turbulence is a key variable to predict aircraft wake behavior. Higher atmospheric turbulence results in faster wake dissipation. Different values of EDR represent different levels of the ambient turbulence intensity, as shown in Table 11(Proctor, Hamilton, and Switzer 2006) .

Table 11 Ambient Turbulence Intensity Levels.

Ambient Turbulence Intensity	Eddy Dissipation Rate (m^2/s^3)
Intense	1.50 * 10-2
Strong	7.17 * 10-3
Moderate – Strong	3.02 * 10-3
Moderate	1.35 * 10-3
Weak	4.00 * 10-5
Very Weak	1.00 * 10-7

We improve the generation of the three-dimensional wake volume by considering stochastic variations of atmospheric parameters. Figure 23 shows a group of Monte Carlo simulation results obtained from the APA model using a variety of input parameters including aircraft flight parameters and atmospheric turbulence parameters, for a Boeing 757-300 class vehicle. The figure shows the vertical wake core positions over time. As seen in the figure, the wake behavior is different for different atmospheric conditions. To ease the computational tasks in EWEM, it is possible to group wake core positions with similar behavior.

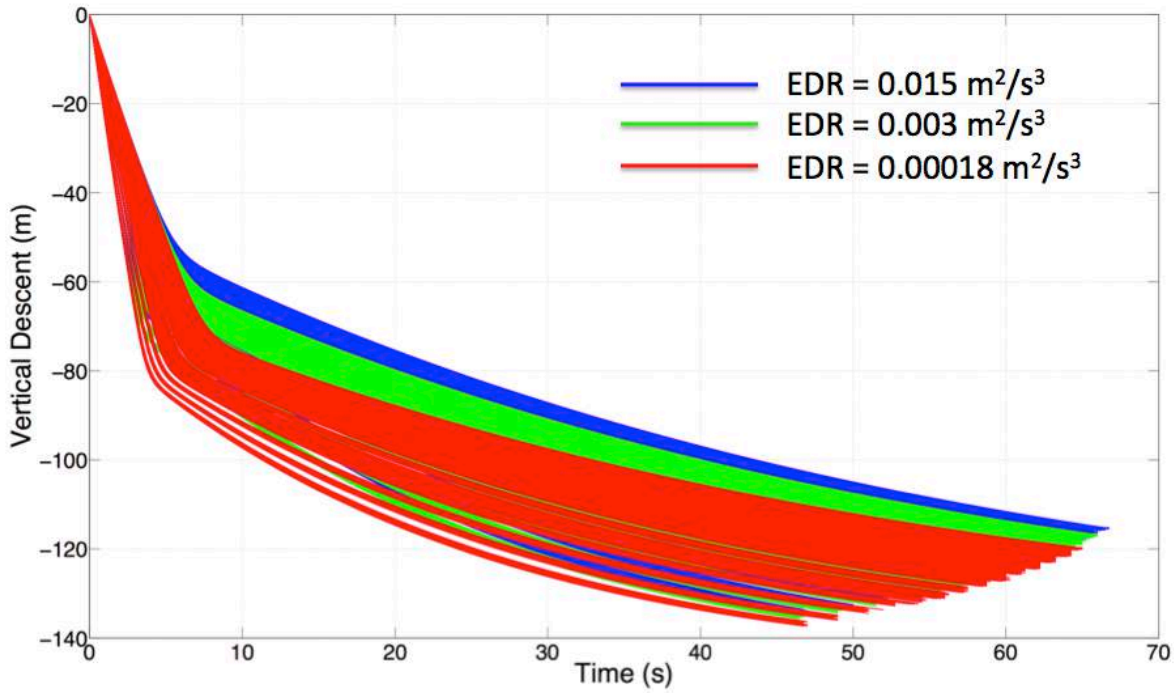


Figure 23 Sample Monte Carlo Results from APA Suite of Models for a Boeing 757-300 Class Vehicle with No Ground Effect under Different Turbulence Intensity Levels.

In Figure 23, wake vortices can be grouped based on aircraft performance factors. We use "normalized" units of input and output parameters introduced in the TDAWP model to help understand similar wake vortex patterns. With the normalization method adopted from the TDAWP model (Proctor 2004), all parameters are normalized to the reference values in Equations (7) – (9). The normalization formulas used are shown in Equations (10) - (12).

$$\Gamma_0 = \frac{Mg}{\rho s B V} \tag{7}$$

$$w_0 = \frac{\Gamma_0}{2\pi b_0} \tag{8}$$

$$t_0(\text{sec}) = \frac{b_0}{w_0} \tag{9}$$

$$\varepsilon^* = \frac{2\pi\varepsilon^{1/3} b_0^{4/3}}{\Gamma_0} \tag{10}$$

$$t^* = \frac{t}{t_0} \tag{11}$$

$$z^* = \frac{z}{b_0} \tag{12}$$

Where,

b_0 (m): Wake initial separation

Γ_0 (m^2/s): Initial circulation

M (kg): Aircraft mass

ρ (kg/m^3): Air density

B (m): wingspan

t*: normalized time

V (m/s): aircraft speed

W_0 (m/s): Initial descent speed

ε (m^2/s^3): Eddy Dissipation Rate

z (m): wake core vertical position

z*: normalized vertical position

ε*: normalized EDR

t_0 (sec): Approximate time to descent one initial vortex spacing

All the normalized parameters are unitless and distinguished from the reference value by adding a star sign. The main value of normalization is that ε* could represent more than one flight condition through a combination of initial flight parameters. Table 12 contains partial results of normalized EDR values for different flight conditions for a Boeing 757-300 Class Vehicle. Each row represents one flight profile, from altitude of 500 feet to 42000 feet. As seen in Table 12 and

Figure 23, different flight profiles can have similar normalized values of EDR, and close normalized EDR values share the same wake vortex behaviors.

Table 12 Sample Normalized EDR Value for a Boeing 757-300 Class Vehicle with EDR = 0.015 m^2/s^3.

Flight Profile		Monte Carlo Runs									
Altitude (FL)	TAS (kts)										
5	158	0.030	0.029	0.030	0.029	0.029	0.030	0.030	0.029	0.030	0.029
10	159	0.031	0.030	0.030	0.030	0.030	0.031	0.031	0.031	0.030	0.030
15	166	0.033	0.032	0.032	0.032	0.032	0.032	0.032	0.031	0.032	0.033
20	167	0.037	0.037	0.037	0.038	0.038	0.038	0.038	0.037	0.037	0.038
30	190	0.043	0.042	0.042	0.042	0.042	0.043	0.043	0.044	0.042	0.043
40	225	0.042	0.041	0.043	0.042	0.041	0.042	0.042	0.043	0.041	0.042
60	272	0.045	0.046	0.045	0.046	0.046	0.046	0.047	0.046	0.047	0.047
80	280	0.044	0.044	0.045	0.045	0.045	0.045	0.044	0.045	0.044	0.045
100	357	0.044	0.043	0.043	0.044	0.044	0.044	0.042	0.044	0.042	0.043
120	367	0.042	0.042	0.042	0.041	0.042	0.042	0.042	0.041	0.042	0.042

We conducted an analysis to understand the normalized EDR values. Figure 24 and 25 show the distribution of normalized EDR range for selected aircraft types in the six ReCat groups under all possible combinations of the flight conditions.

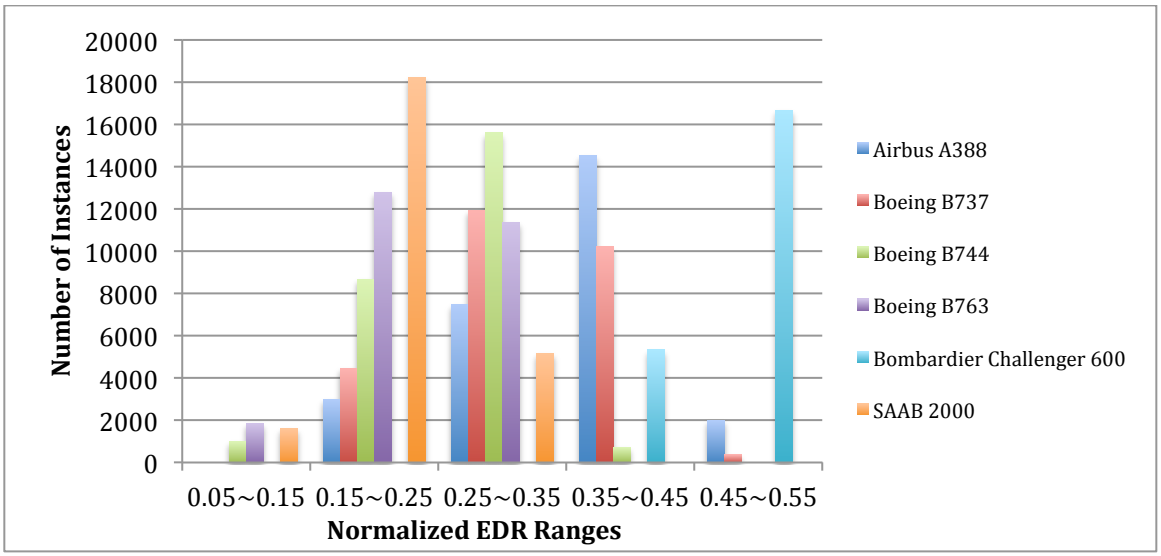


Figure 24 Distribution of Normalized EDR Range for Selected Aircraft Types under Arrival.

As shown in Figure 24, more than 84% of the flight conditions have the normalized EDR value falls between the range 0.15 to 0.45. Figure 24 shows that more than 80% of the departure flights have normalized EDR values ranging from of 0.05 to 0.25. Departure flights have higher weights comparing to arrival flights, thus results in a lower normalized EDR according to Equation (10). Flights under all flight conditions can be covered with normalized EDR from 0.05 to 0.55.

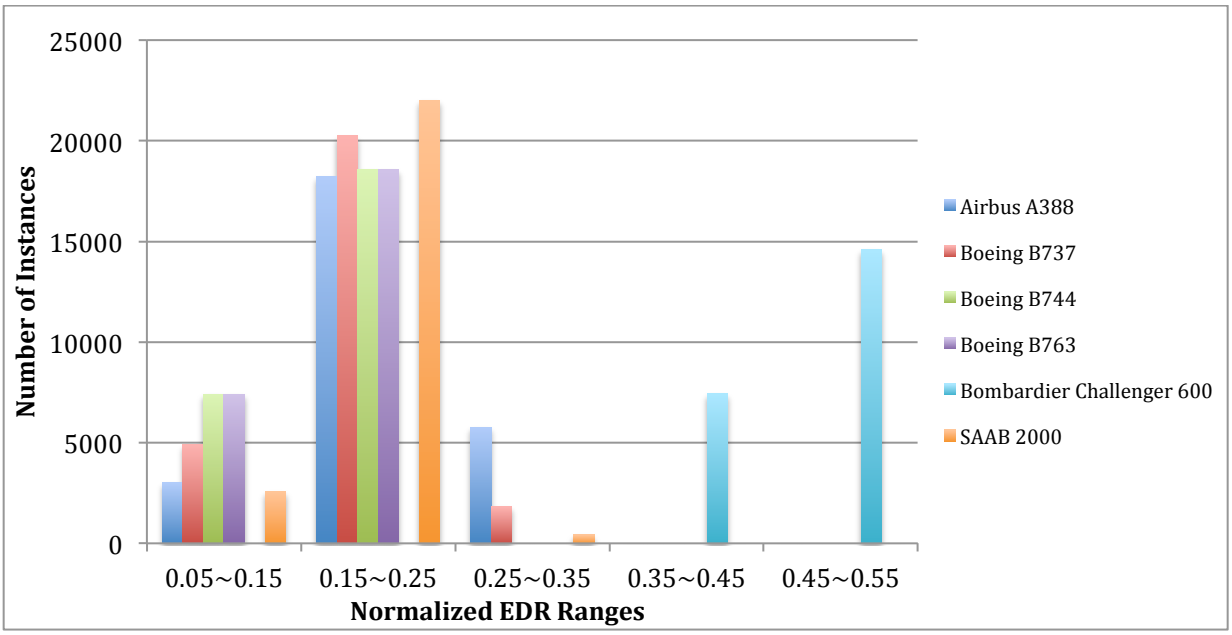


Figure 25 Distribution of Normalized EDR Range for Selected Aircraft Types under Departure.

Common values of ε* share similar wake behavior. Similarly, different flight parameters could result in the same normalized value of EDR. Thus, instead of having three distinct variables in the inputs (aircraft mass, aircraft speed and atmospheric turbulence value), one normalized EDR would cover a range of aircraft parameters, resulting in similar wake behaviors. Figure 26 illustrates this point for a Boeing 757-300 class vehicle for various values of ε*.

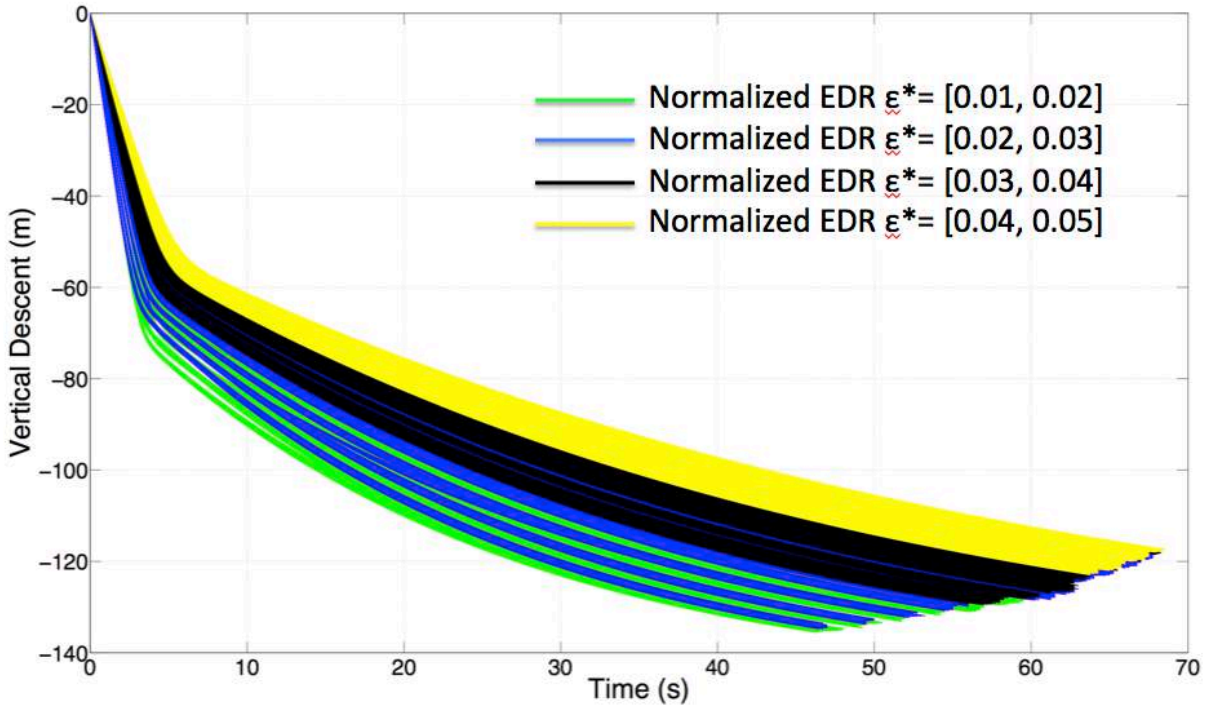


Figure 26 Sample Monte Carlo Results from APA Suite of Models for a Boeing 757-300 Class Vehicle with No Ground Effect under Different Turbulence Intensity Level Colored by Normalized EDR Value.

The normalization procedure helps aggregate different flight condition for the same aircraft type. A second benefit is that it can help group similar wake behaviors for different aircraft types. Figure 27 shows the wake behaviors for three different aircraft types with the same value of normalized EDR. Three different aircraft type, have different wingspan and aircraft mass, belong to the same ReCat group D. Wakes show very similar behavior with only differences in length.

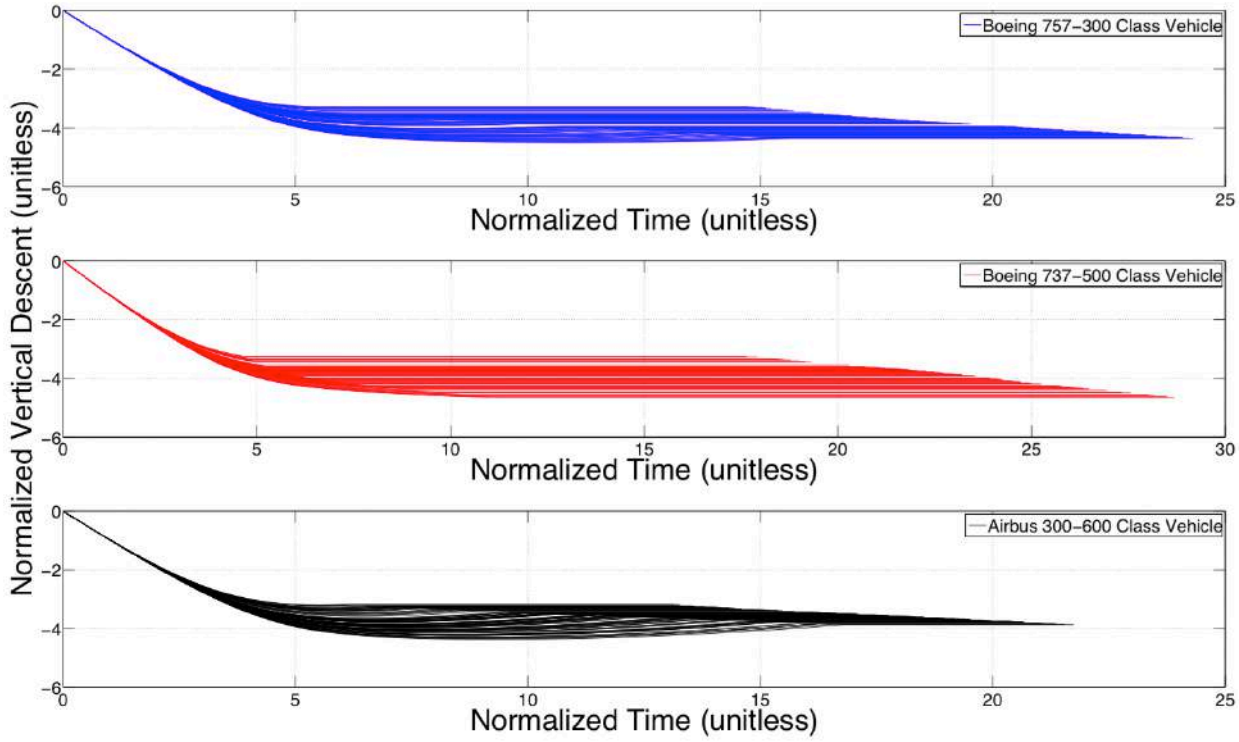


Figure 27 Wake Behavior Comparisons between Three Different Aircraft Types in ReCat Group D.

The normalization of parameters is used in the construction of a wake zone volume. To satisfy the coverage of all possible flight conditions, a Monte Carlo simulation approach is applied to the APA model to derive hundreds of combinations of the normalized wake zone parameters. To verify the coverage of the Monte Carlo simulation, we also performed some sensitivity analysis for different sample sizes in the simulation. We tested the coverage range for different size of Monte Carlo simulation for the same normalized EDR range. Table 13 shows the differences of five sensitivity tests. Figure 28 presents the trend of the wake envelope volume defined as the number of Monte Carlo simulation runs increased. The wake volume protected by the Monte Carlo runs has a steady state after reaching the sample size of 100. The improvement from increasing the sample size is very small comparing to the additional run time. The envelope extensions with more than 100 runs are subtle, but the run time increases significantly. We performed the Monte Carlo simulation with the sample size of 100 for each flight condition

available in the BADA input profile files. Thus, for each aircraft type, more than 10,000 runs are performed to cover all normalized EDR values.

Table 13 Comparisons of Different Monte Carlo Sample Sizes.

Number of Simulation Runs	Protected Wake Envelope Volume	Boundary Coordinate	
	(unitless)	Vertical (unitless)	Lateral (unitless)
1000	574.7374	-15.0262	126.0363
300	574.6372	-15.025	126.0147
100	574.2091	-15.0252	125.9425
50	574.1199	-15.0235	125.967
20	571.5446	-15.0167	125.5432

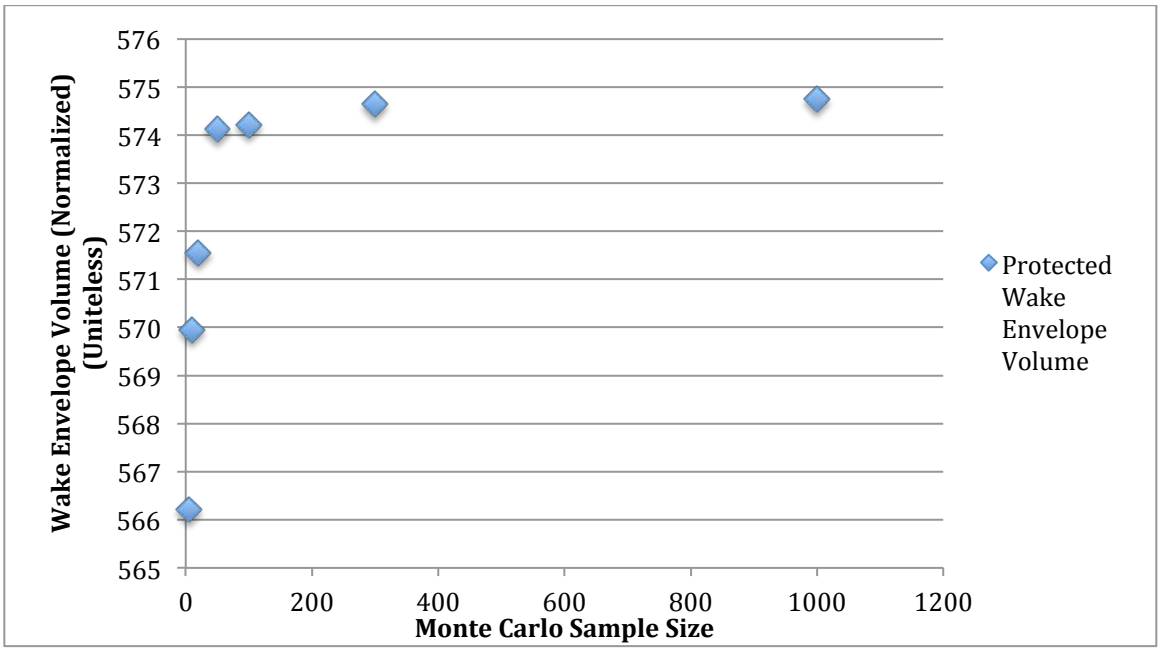


Figure 28 Wake Envelope Volume Defined with Different Monte Carlo Simulation Sample Size.

To expand the deterministic result obtained using the APA Model while keeping the level of accuracy, we choose the step of 0.1 of normalized EDR values. Figure 29 shows a series of outputs from the APA model with normalized EDR values ranging from 0.25 to 0.35. As discussed earlier, the wake envelope with the same normalized EDR range show similar behavior with small variation.

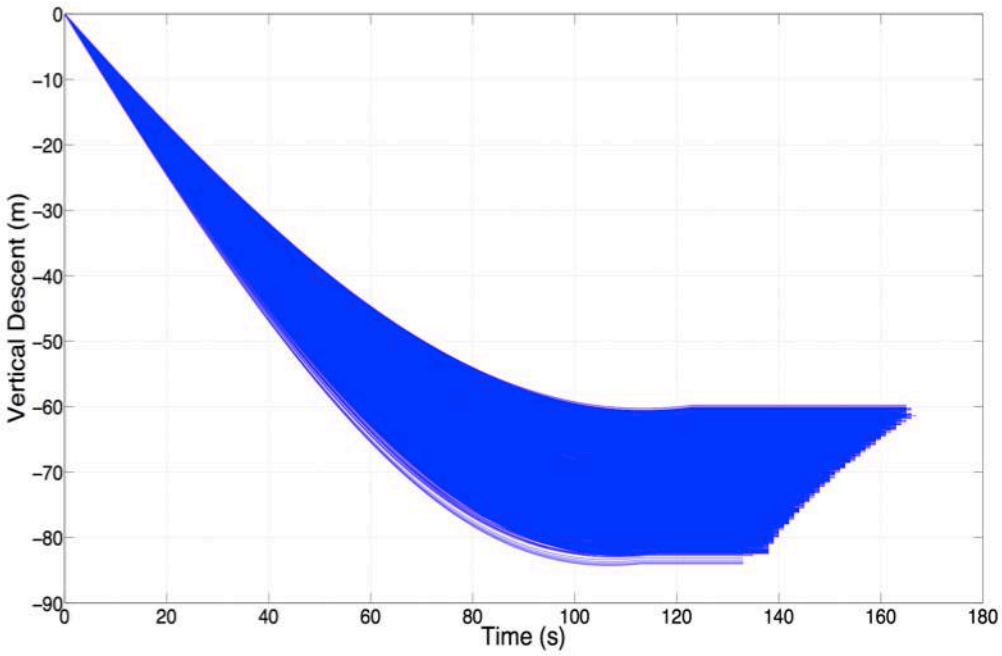


Figure 29 Sample Monte Carlo Results from the APA Suite of Model for Aircraft Type Boeing 757-300 with Normalized EDR Values between 0.25 and 0.35.

3.3.4 Brunt-Väisälä Frequency

Brunt-Väisälä Frequency (BVF), or buoyancy frequency is another atmospheric parameter that influences the behavior of wake vortices. BVF can be estimated using the vertical gradient of potential atmospheric temperature. BVF is defined in Equation (13).

$$N = \sqrt{\frac{g}{T}\frac{\partial T(\zeta)}{\partial \zeta}} \tag{13}$$

where,

N (radians per second): Brunt-Väisälä Frequency

g (m/s^2): Acceleration of gravity, 9.81

T(ζ): Vertical potential temperature profile

In the APA model, BVF is derived from the vertical temperature profile (TDATA) supplied to this model and using Equation (14) (R.E.Robins 2002).

$$N = \frac{g}{(T+T_0)}\left(\frac{dT}{d\zeta} + \gamma\right) \quad (14)$$

Where,

T_0 (°C): Convert Kelvin to degree Celsius, 273.15

γ (°C/m): Dry adiabatic atmosphere lapse rate, 0.00976

In this analysis we employ the International Standard Atmosphere (ISA) model to represent typical temperature variations with altitude. The ISA model assumes a linear temperature distribution with altitude. At sea level, the base temperature is 15 °C, and an initial lapse rate of -6.5 °C/km. The lapse rate applies up to 11 km, where the temperature drops to -56.5 °C. The temperature is then remain constant between 11 km and 20 km (United States Committee on Extension to the Standard 1976), and increase with the lapse rate of 1 °C/km when reaches the altitude of 32 km, as seen in Figure 30.

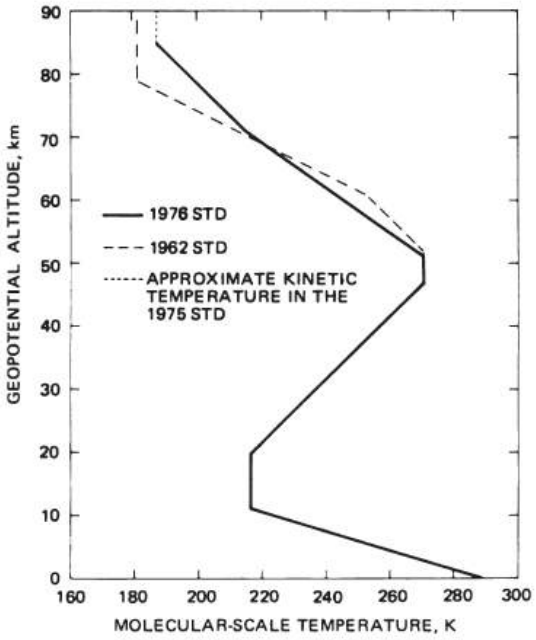


Figure 30 Molecular-scale Temperature as a Function of Geopotential Altitude (United States Committee on Extension to the Standard 1976).

Table 14 shows the numerical values of a temperature profile (TDATA) used in the APA simulations. The APA model interpolates linearly between adjacent height values. The highest

altitude in the profile is set to 32 km, consistent with ICAO standard (United States Committee on Extension to the Standard 1976).

Table 14 Sample TDATA File.

	Sample TDATA file	
Number of Records	**32**	
	Altitude (m)	Temperature (K)
	0	288
	100	287.35
	200	286.7
	300	286.05
	400	285.4
	500	284.75
	600	284.1
	800	282.8
	1000	281.5
	2000	275
	3000	268.5
	4000	262
	5000	255.5
	6000	249
	7000	242.5
	11000	216.5
	20000	216.5
	22000	218.5
	25000	221.5
	29000	225.5
	32000	228.5

3.3.5 Circulation Threshold

The APA model simulates the dynamic evolution of wake vortex cores to a point where the circulation strength reaches zero. In practical terms, circulations under certain threshold level can be considered relatively harmless for particular aircraft type. As stated before, larger transport aircraft types can tolerate stronger wake turbulence than small aircraft. To address this concern, we introduce a circulation threshold level, as the point where the envelope is no longer being tracked because it represents a non-hazardous condition for a following aircraft. Different circulation threshold values have been tested and suggested as values default for six categories modeled. These are presented in Table 15.

Table 15 Circulation Threshold Suggested in EWEM.

ReCat Group	Most Critical Aircraft	Circulation Threshold (m^2/s)
A	Airbus 380-800	250
B	Boeing 747-400	250
C	McDonnell Douglas 11	200
D	Boeing 757-300	125
E	British Aerospace Avro RJ85	100
F	Bombardier Challenger 600	100

The wake vortices build from the APA model, as shown in Figure 28, will be truncated based on the circulation threshold set in Table 15. With truncated the wake vortices at the appropriate threshold level, an example of a wake zone polygon is shown in Figure 31. The figure shows a wake zone, which is shorter in both time and vertical dimensions compared to the original unconstrained wake zone shown earlier in Figure 28.

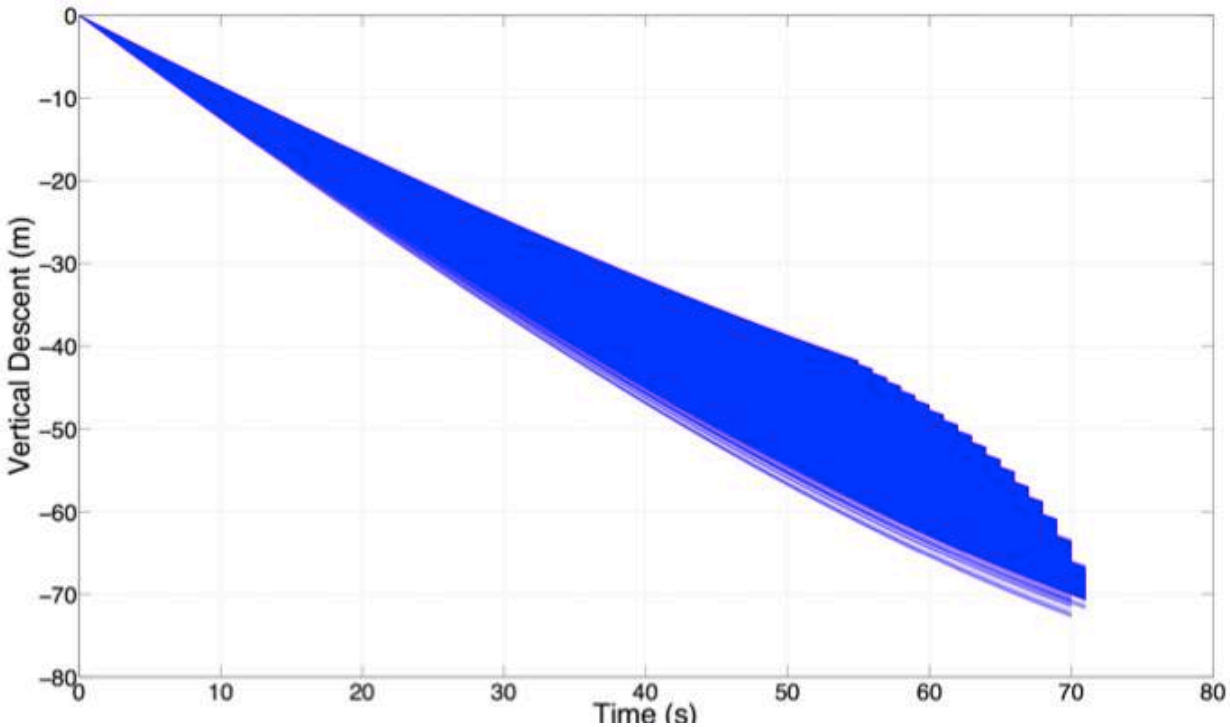


Figure 31 Sample Monte Carlo Simulation with Truncated Results at Circulation Threshold of 125 m^2/s from APA Suite of Model with Normalized EDR Values between 0.25 ~ 0.35.

The wake zone generated is considered as the potential wake hazard area to be avoided by other aircraft. Similar analyses were performed for selected six aircraft types under all conditions. The

protected zone will be defined with series of control points on the boundary. Control points defining the convex hull of the simulated wake cores, are computed for each aircraft category and stored in a convenient data structure. To reduce the number of wake zones stored within each ReCat group, the normalization method is applied. The wake zone will be normalized before storage, as shown in Figure 32. Table 17 shows an illustration of the wake construction look-up table, formed by a series of control points under different normalized flight conditions. These "stored" wake zones can be used in fast-time simulation models or in dedicated models to predict potential wake encounters (i.e., EWEM). Since the values contained in the look up table are normalized, application of such table in the EWEM model requires a transformation of parameters using the formulas presented in Equations (7) – (11).

The ground effect on the wake zone is modeled separately in the following section.

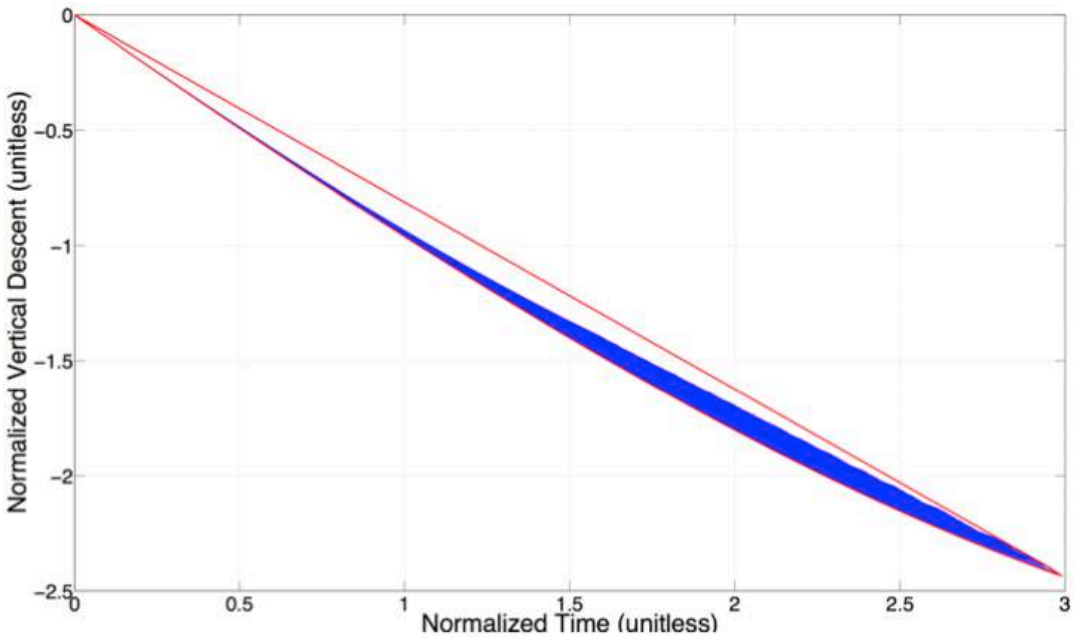


Figure 32 Sample envelope polygon used in the EWEM from the results of APA Suite of Model of Boeing 757-300 under Out-of-Ground-Effect with Normalized EDR values between 0.25 ~ 0.35.

Figure 32 shows a group of modeled wake vortices behavior in blue lines bounded by the convex hull drawn in red as the protected zone. EWEM use the points on the red line as control points stored in the look-up table. We think that protecting the bowl-shaped blue area does not ensure

the safety of the following aircraft. In another word, the differences between the blue and the red area shown in the Figure 32 is not safe for nearby aircraft to penetrate without any wake turbulence effect.

The look-up table, as shown in Table 16, is used to determine the longitudinal and vertical positions of the wake envelope control points behind the aircraft generator. The normalized value of ε* in Table 12 is computed based on the input parameters of an aircraft for a given set of flight conditions, based on Equation (10). The value of ε* calculated is compared against the ranges contained in the look-up table and values of normalized coordinates for time and vertical wake position are found. These normalized values of time and vertical position are converted to real values of time and vertical position (seconds and meters, respectively). Empirical studies (Holzapfel 2003b) have found that wakes under no wind condition, travel lateral at the same rate as they descent vertically. In this model, we assume that the lateral motion of the wake is the same as the vertical motion of the wake under Out-of-Ground-Effect conditions.

Table 16 Look up Table Illustration.

Look-up Table Illustration											
Re-Cat Group	Arrival / Departure	EDR range	Circulation Threshold Level	IGE / OGE	Altitude (in feet)	Lateral_Coordinates	x(i)	y(i)	Vertical_Coordinates	x(i)	z(i)
1.000	Arr	1.000	100.000	IGE	100.000	90 points	0.008	-0.006	152 points	0.037	-0.037
1.000	Dep	2.000	125.000	IGE	100.000	88 points	0.008	-0.008	159 points	0.037	-0.037
1.000	Arr	3.000	200.000	IGE	200.000	60 points	0.008	-0.008	100 points	0.037	-0.037
2.000	Arr	4.000	200.000	OGE	N/A	N/A	N/A	N/A	255 points	0.037	-0.037
3.000	Arr	1.000	150.000	OGE	N/A	N/A	N/A	N/A	240 points	0.037	-0.037
4.000	Dep	2.000	175.000	IGE	200.000	70 points	0.008	-0.006	160 points	0.037	-0.037
5.000	Arr	1.000	200.000	OGE	N/A	N/A	N/A	N/A	232 points	0.037	-0.037

To represent different aircraft mass profiles within a single ReCat aircraft group, we divide the normalized tables into departure and approach profiles. The ground effect evolution of the wake vortices on final approach is also considered separately. Considering all related factors, the wake zone boundary coordinates are constructed and stored in a MATLAB data structure to represent all of the aircraft operating conditions in the NAS.

3.3.6 In-Grounds-Effect Model

The behavior of wake vortices in ground effect is different to Out-of-Ground Effect behavior. (Sarpkaya, Robins, and D.P.Delisi 2001). The ground effect phenomena takes place when the flight below a certain altitude. The altitude was set based on a series of sensitivity analyze using the normalized value of EDR as the controlling factor. Thousands of APA Suite of Model runs were conducted at all flight conditions for altitude ranging from 0 to 2000 feet, as plotted in Figure 33. The step size of normalized values of EDR was 0.1 in the runs.

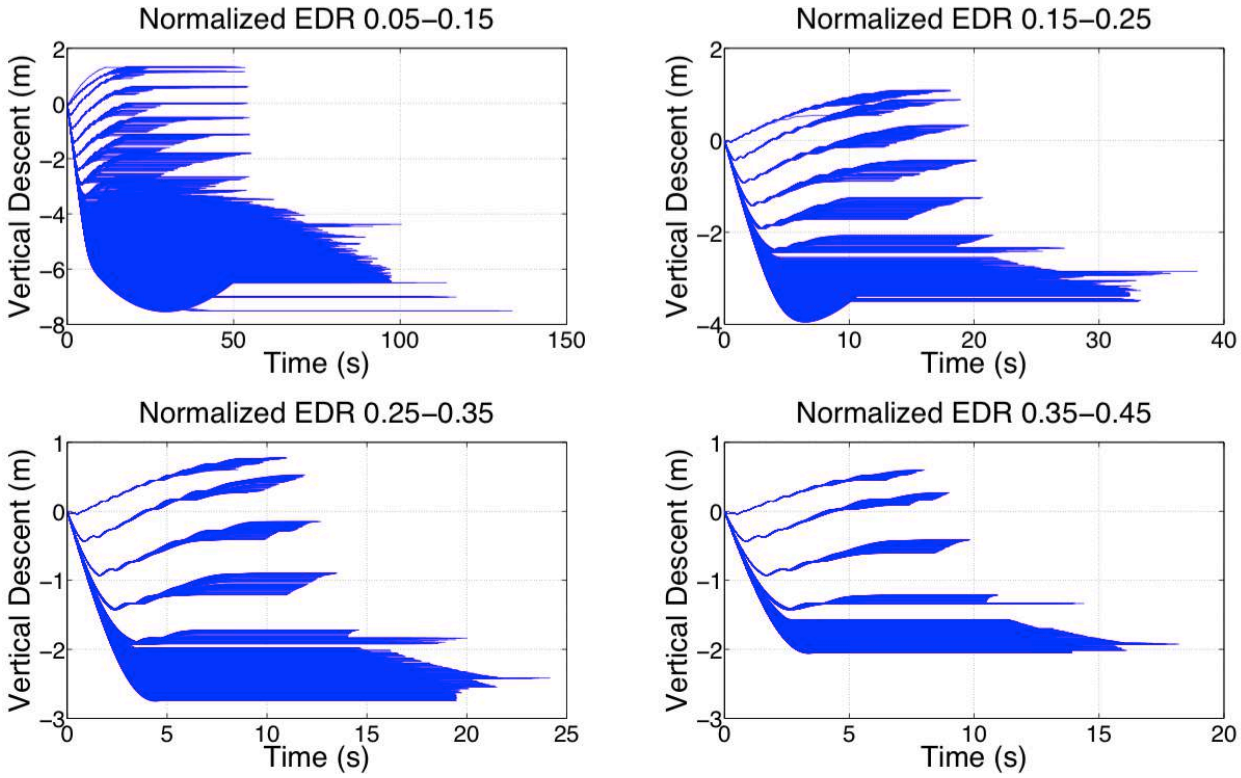


Figure 33 Wake Vortices Behavior for Aircraft Type Boeing 757-300 under In-Ground-Effect for Different EDR Range.

Each line represents one wake behavior under one unique altitude value. For each normalized EDR range, we recorded the altitude boundary that produces distinct IGE and OGE wake behavior, as shown in Table 17.

Table 17 Different Altitude Threshold Found for ReCat Group A under Different Normalized EDR range.

ReCat Group	Normalized EDR Range	Altitude Threshold for IGE	
A	0.05~0.15	290m	4*wingspan
	0.15~0.25	198m	2.5*wingspan
	0.25~0.35	140m	2*wingspan
	0.35~0.45	106m	1.4*wingspan
	0.45~0.55	60m	0.78*wingspan

We calculated the aircraft flight speed and aircraft weight combination for each selected aircraft type, and locate the feasible normalized EDR range affecting each aircraft type. As listed in Table 18, for an Airbus A380-800, at given altitude, with different flight profiles (aircraft mass and flight speeds) and environmental parameters, the initial descent rate of the wake can be very different. This is the result in the different values of the normalized EDR. For the Airbus A380-800 under IGE conditions, normalized EDR range from 0.15 to 0.25. These values correspond to an altitude of two and half wingspans set as IGE threshold boundary. The same analysis is repeated for other ReCat groups. After the analysis is completed for all six ReCat groups, we found that wake in ground effect develop two and a half wingspan from the ground.

Table 18 In-Ground-Effect Analysis at Different Normalized EDR Range for Airbus 380-800 Class Vehicle.

Normalized EDR	Input Data for the APA Model				Remarks
	Longitude	Altitude (m)	$V_{Initial}$ (m/s)	b_0 (m)	
0.05~0.15	0	15.24	2.89~8.31	62.72	Unfeasible flight speed and weight combination
0.15~0.25	0	15.24	1.70~2.82	62.72	Operating at MLW with speed of 140 knots
0.25~0.35	0	15.24	1.22~1.688	62.72	Unfeasible flight speed and weight combination
0.35~0.45	0	15.24	0.94~1.2	62.72	Unfeasible flight speed and weight combination
0.35~0.45	0	15.24	0.94~1.2	62.72	Unfeasible flight speed and weight combination

The ground effect can produce a “bouncing” wake behavior instead of continuously decaying in one, as shown in Figure 34.

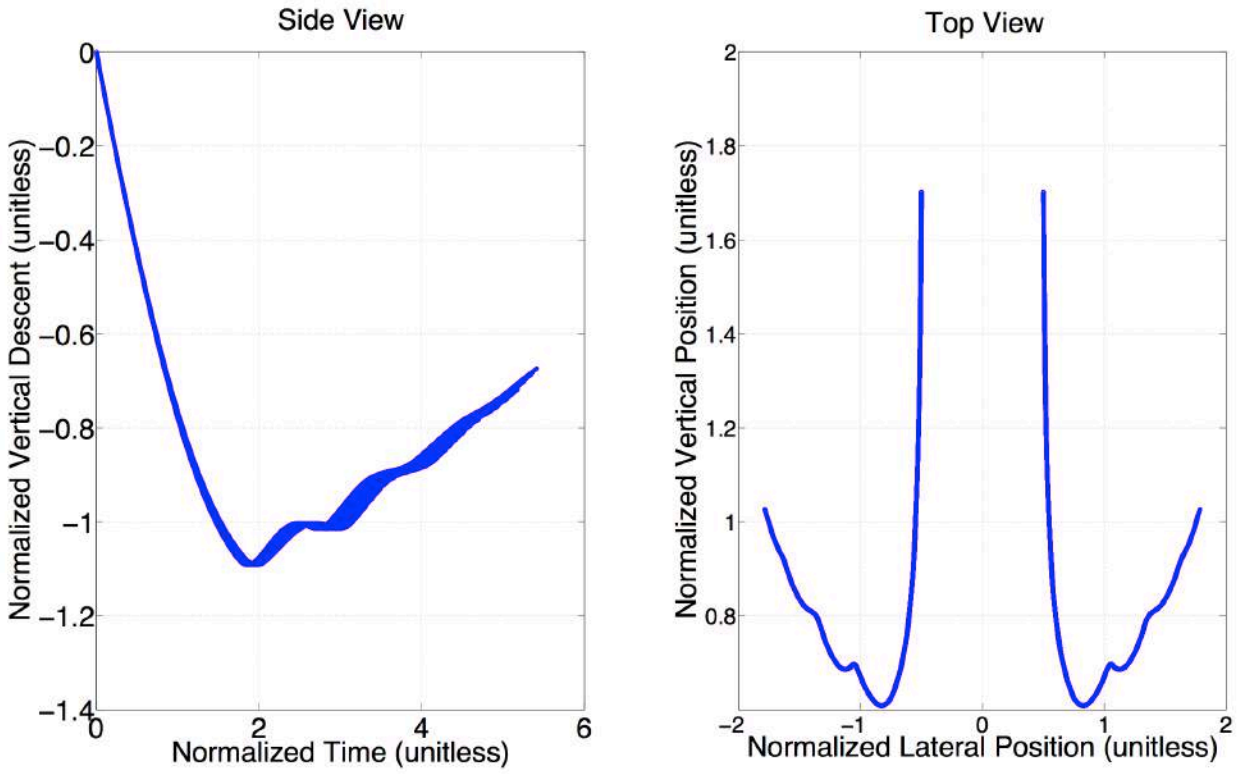


Figure 34 Monte Carlo Simulation Results from APA Suite of Models for Airbus A380-800 Class Vehicle under In-Ground-Effect at Normalized EDR 0.15~0.25.

In addition to the vertical prediction of the wake core positions under OGE, the APA model also produces the lateral behavior of the wake vortex under IGE, as shown in the Figure 33. When the wake vortices enter the IGE range, secondary vortices and their images are introduced in the algorithms of the APA model. The result is both a “bouncing behavior” and the lateral expansion of the wake zone. Figure 35 gives an illustration of wake vortices under IGE conditions. (R.E.Robins 2002).

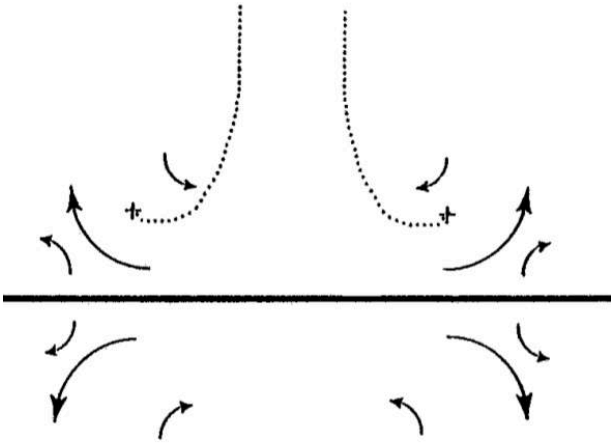


Figure 35 Wake Vortices Bouncing Behavior under IGE

+: Wake Core Positions (R.E.Robins 2002).

In addition, it is observed that the "bouncing behavior" varies with the initial vortex height. Figure 36 presents the In-Ground-Effect changes of the wake of the Airbus A380-800 vehicle class as the flight altitude increases. Figure 36 shows that at higher flight altitudes, the less "bouncing behavior" was observed. The differences in the "bouncing behavior" take place in a small altitude range. For this reason, we represent the In-Ground-Effect wake zone behavior at every 50 feet of altitude intervals. The look-up table contains control points' coordinates for each 50 feet under IGE. This provides enough fidelity to construct an envelope for aircraft under In-Ground-Effect.

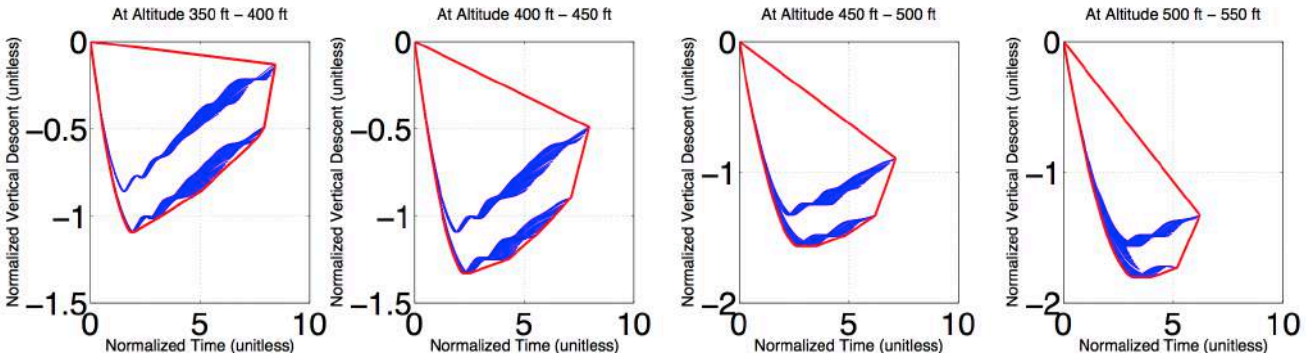


Figure 36 Monte Carlo Simulation Results from APA Suite of Model for In-Ground-Effect under Different Altitude for Airbus A380-800 at Normalized EDR 0.25~0.35.

The APA Suite of Models also provides an explicit lateral position of the wake under ground effect. We use a similar methodology to construct the normalized envelope along the lateral dimension.

With all the IGE wake envelopes constructed including lateral and vertical positions of the wake zone control points under all flight parameters and atmospheric conditions, we developed a data structure (i.e., look-up table) that contains the complete the definition in three dimensions of the wake zone behind the wake generator. This information is used in fast-time simulations and in the EWEM model.

3.4 Turning Flight Path and Wind Analysis

The wake envelopes developed so far assumed a straight-line right path. In real life, aircraft perform turns while flying from an origin to a destination. In maneuvering flight, the wake envelope "bends" with the flight track. Figure 37 shows an example of a bending envelope following with the flight track.

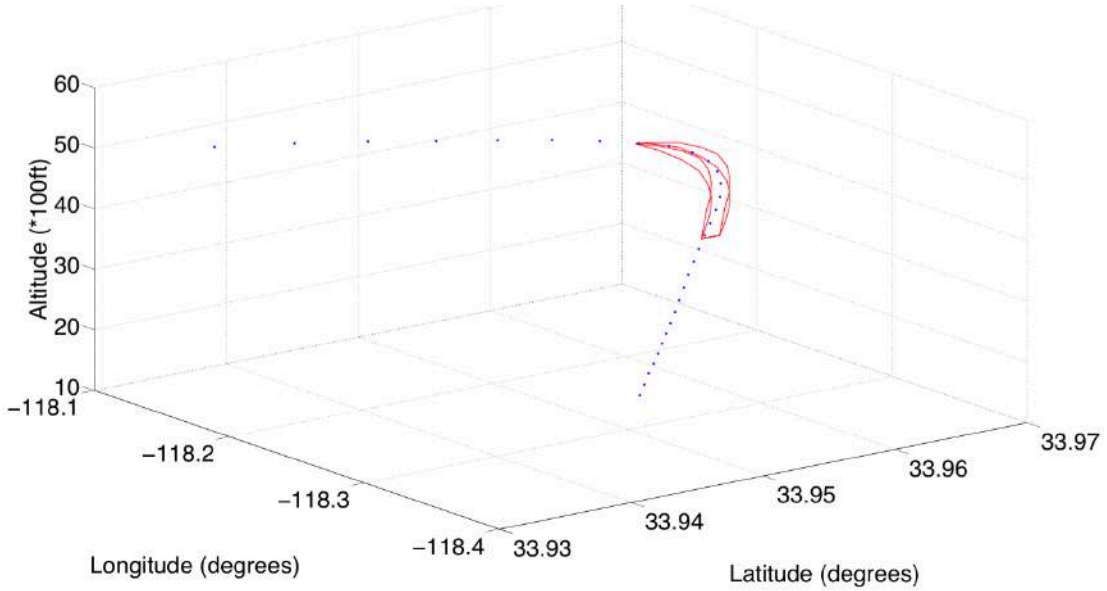


Figure 37 A Bending Envelope Illustration for A Boeing 747-400 Class Vehicle.

The model has a wind analysis module that can apply different types of wind data to improve the realism of the wake zone protection level. This will apply to cases, there is accurate wind data. EWEM allows preset crosswind information or direct data provided by the Automated Surface Observing System (ASOS) as additional input to the default scenario. With the wind module, the envelope will divert sideways based on the given wind information, as shown in Figure 38. The wind module would be useful for NextGen procedures such as Wake Turbulence Mitigation for Departures (MTWD) or Wake Turbulence Mitigation for Arrivals (WTMA) mentioned in Chapter 2.1. If wake zone can be predicted precisely under a given set of wind conditions, parallel runway operations can be processed at closer time intervals “free” of wake conflicts.

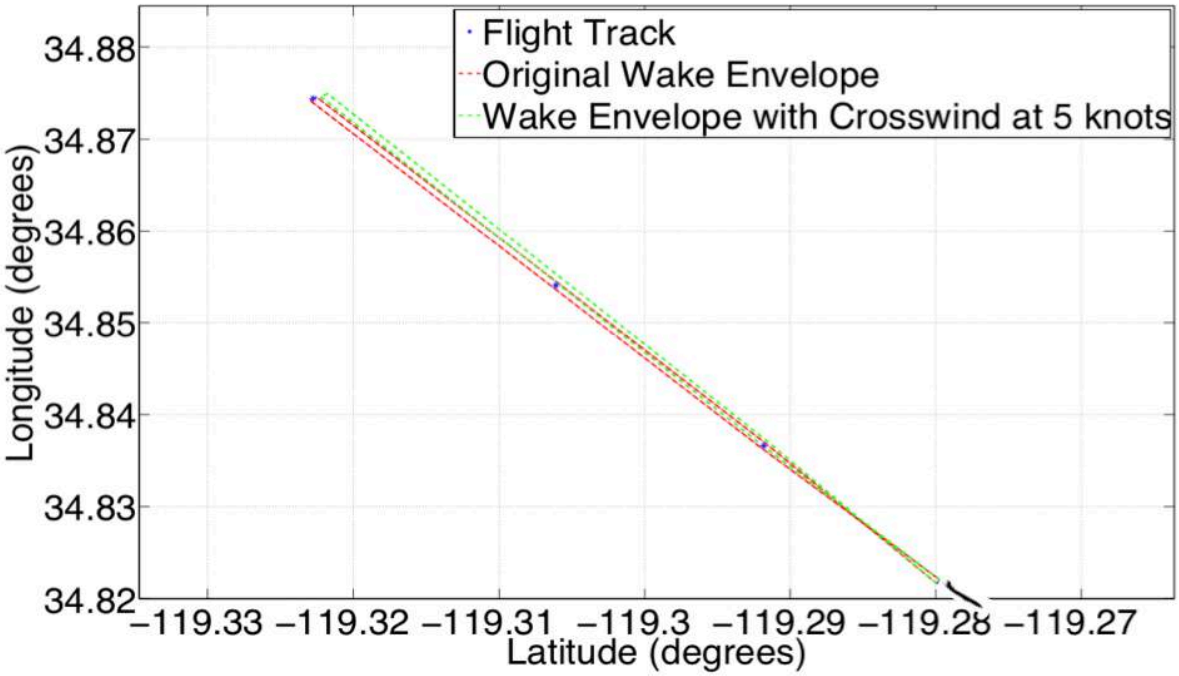


Figure 38 Wake Envelope without Wind Analysis vs. Wake Envelope with 5 knot Crosswind.

In other cases, when no wind data is available, or if the atmospheric condition is uncertain, the wind module adds an additional wake zone expansion by adding a wind distribution to the envelope. In this mode of operation, the wind module requires the mean and the standard deviation of the crosswind strength as inputs. This allows the model to expand the wake zone

laterally on both sides with a distribution by 2 sigma. Figure 39 represents the wake zone expansion behavior with wind distribution input.

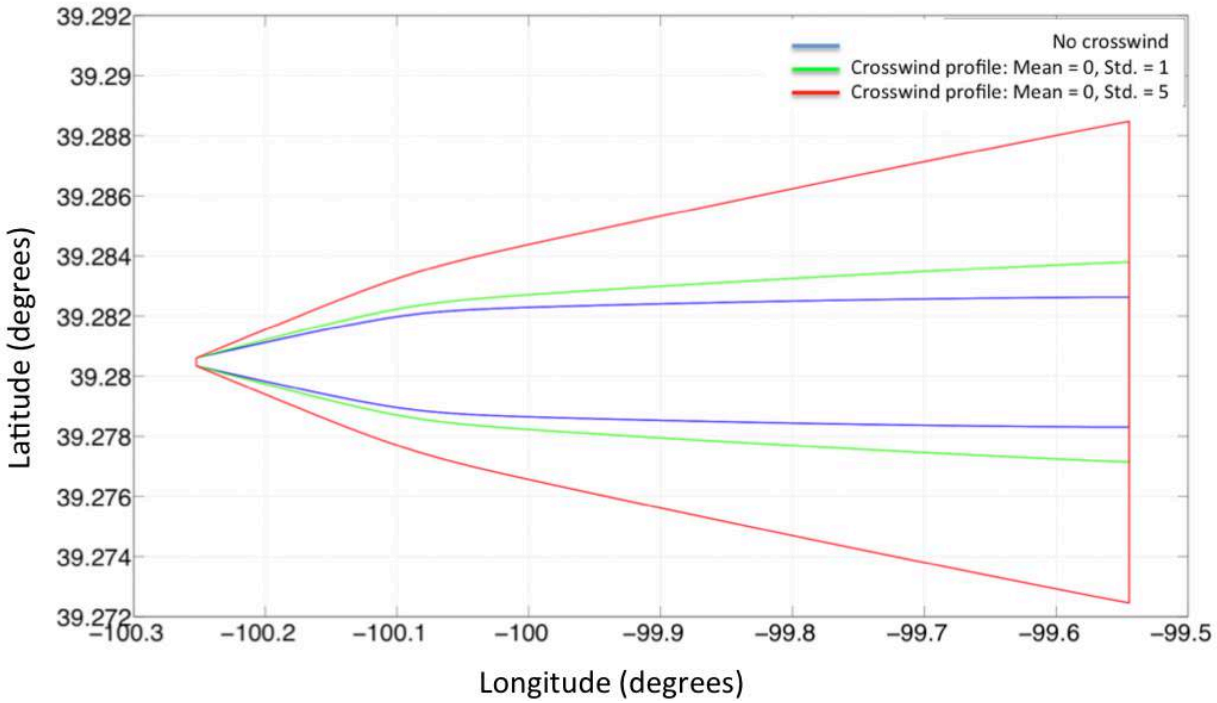


Figure 39 Top View of Wake Zone for B757-200 Class Vehicle with Different Crosswind Settings.

3.5 Model Output

With a complete wake look-up table, EWEM constructs wake zone trailing from the wake generator. The wake zone boundary is defined as a function of aircraft parameters, atmospheric conditions, and the “trailing” aircraft category. Figure 40 presents the envelope construction flowchart given the aircraft flight information. EWEM loads the flight information first, assigns the proper ReCat group, calculates the normalized EDR based on the environmental turbulence and aircraft characteristics, reads the wake boundary from the wake look-up table based on altitude, flight condition, and truncates the wake zone considering the following aircraft ReCat group.

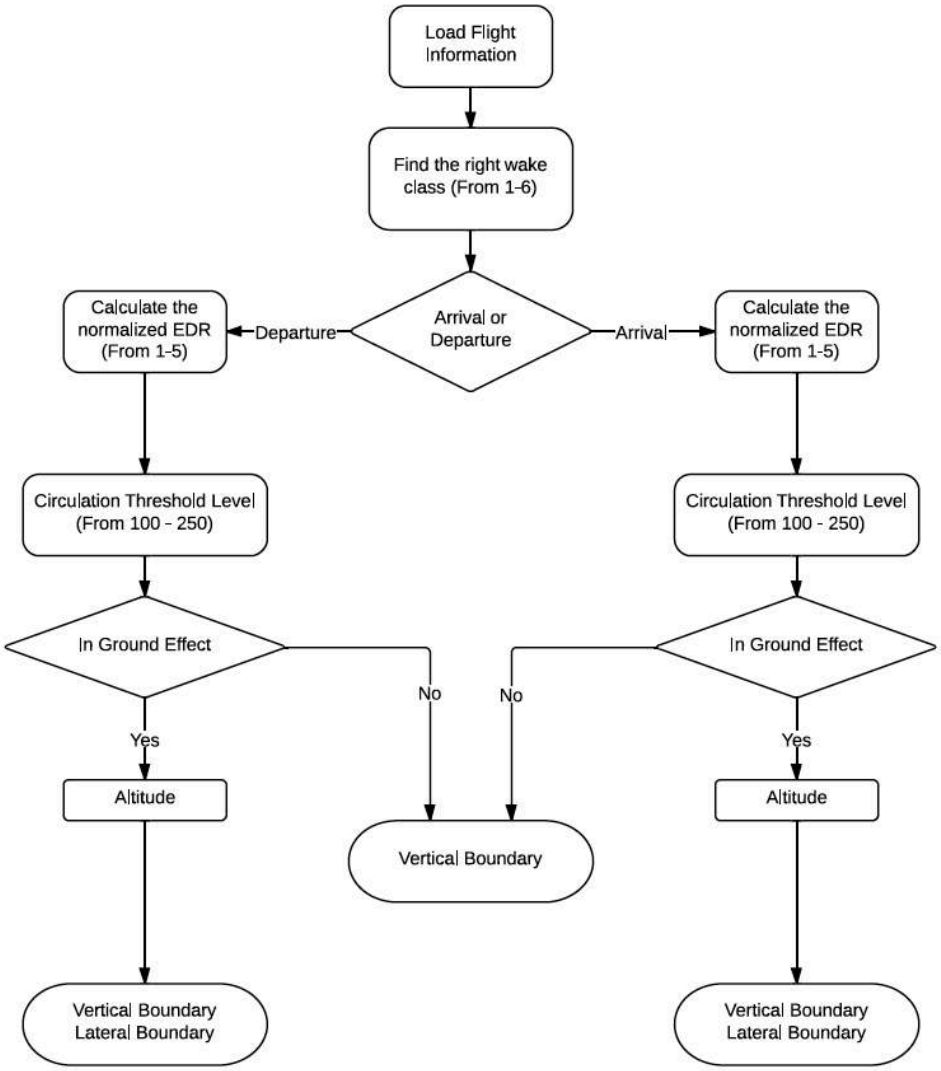


Figure 40 Data Structure of the EWEM Model Look-up Table Containing Wake Zones of Aircraft.

The main purpose of EWEM model is to provide an estimate of potential wake encounters for flight trajectories with limited data. The model has a set of default settings. These include environmental conditions, aircraft list, ReCat group assignment, and circulation thresholds. The default parameters in EWEM can be modified by the user as needed.

The only required input is the flight track data. Flight data is provided in three common formats used in aviation studies are introduced in Chapter 4. The model evaluates the number of potential wake encounters based on default settings. Flights near the wake generator will be considered as potential followers. The criteria we used to filter potential followers is based on time and space

criteria. If two aircraft separated less than 20 nm in Euclidean distance or within 4000 feet in altitude, they will be considered as a potential pair. The wake zone length varies depending on the follower aircraft ReCat group. After preprocessing all potential pairs, they are processed in the main wake analysis, see Figure 41. In the main wake encounter analysis, we generate the wake envelope of the lead aircraft at every position in the flight path of the vehicle position. Wake zone is truncated based on the characteristics of the follower ReCat group. The envelope is then bent to follow the flight track, and dispersed to account for wind conditions. Once the envelope is fully constructed, the protected wake zone is checked against all potential followers for any intrusion. If any intrusion is detected, it will be reported as one potential wake encounter in the output of the model. The mechanics of this process are illustrated in Figure 41.

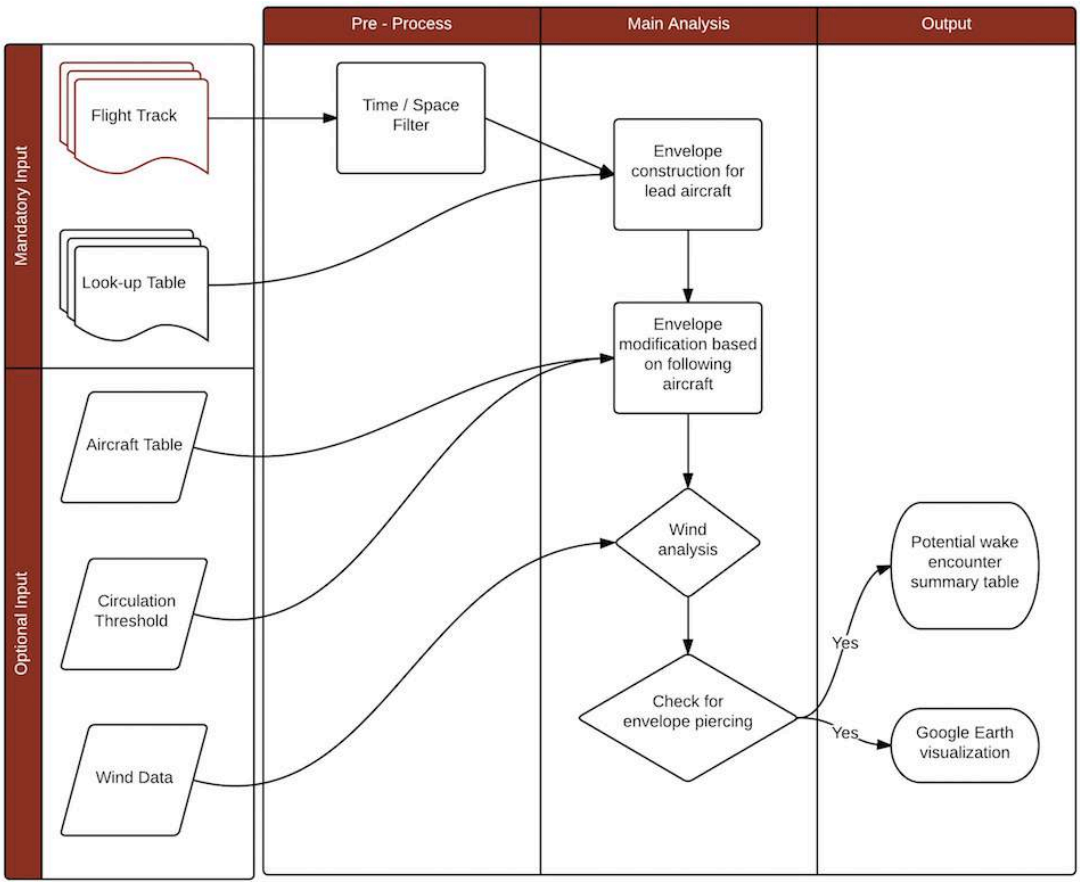


Figure 41 EWEM Model Flowchart.

A typical three-dimensional wake envelope generated from the model is shown in Figure 42. The envelope considers both vertical descent and the lateral transport behavior of the wake. Red lines define the protected wake zone generated from the normalized boundary coordinates using the look-up tables explained in Chapter 3.3. The parallel blue rectangles present cuts generate from the lateral transport behavior of the wake in ground effect conditions, as shown in Figure 43.

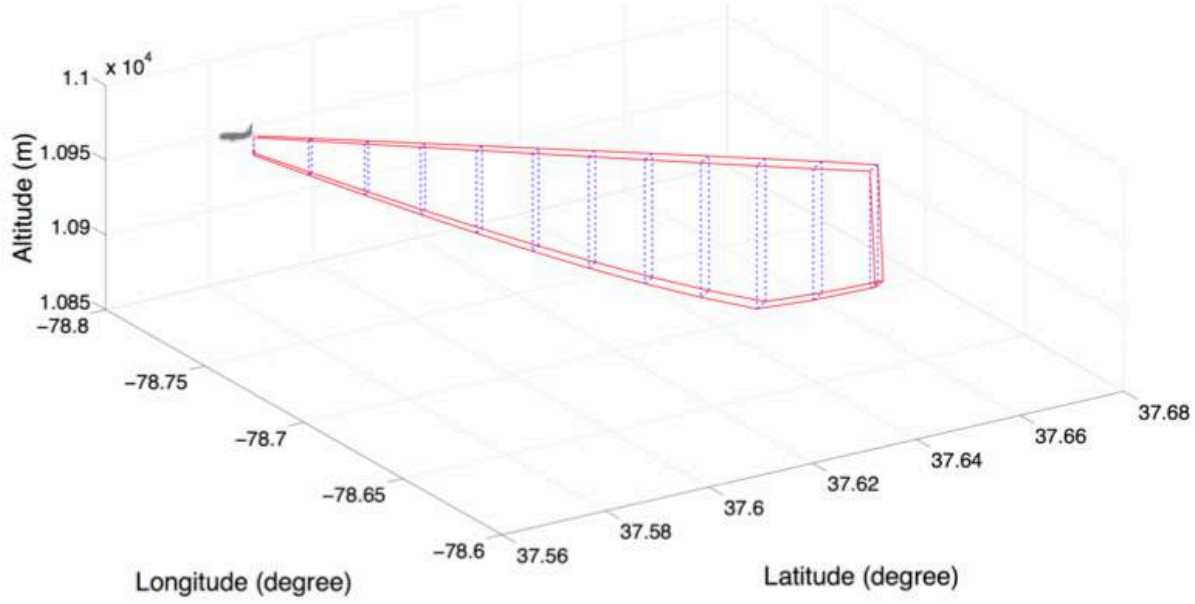


Figure 42 Sample Three-Dimensional Wake Envelope.

Figure 43 illustrates a front view of the wake envelope for IGE conditions. The wake's bouncing behavior reviewed in the earlier Chapter 3.3.6 results in the lateral expansion of the envelope and a somewhat irregular envelope shape. It is important to remember that the blue lines shown in Figure 43 are the positions of the cores of two vortices generated by the aircraft.

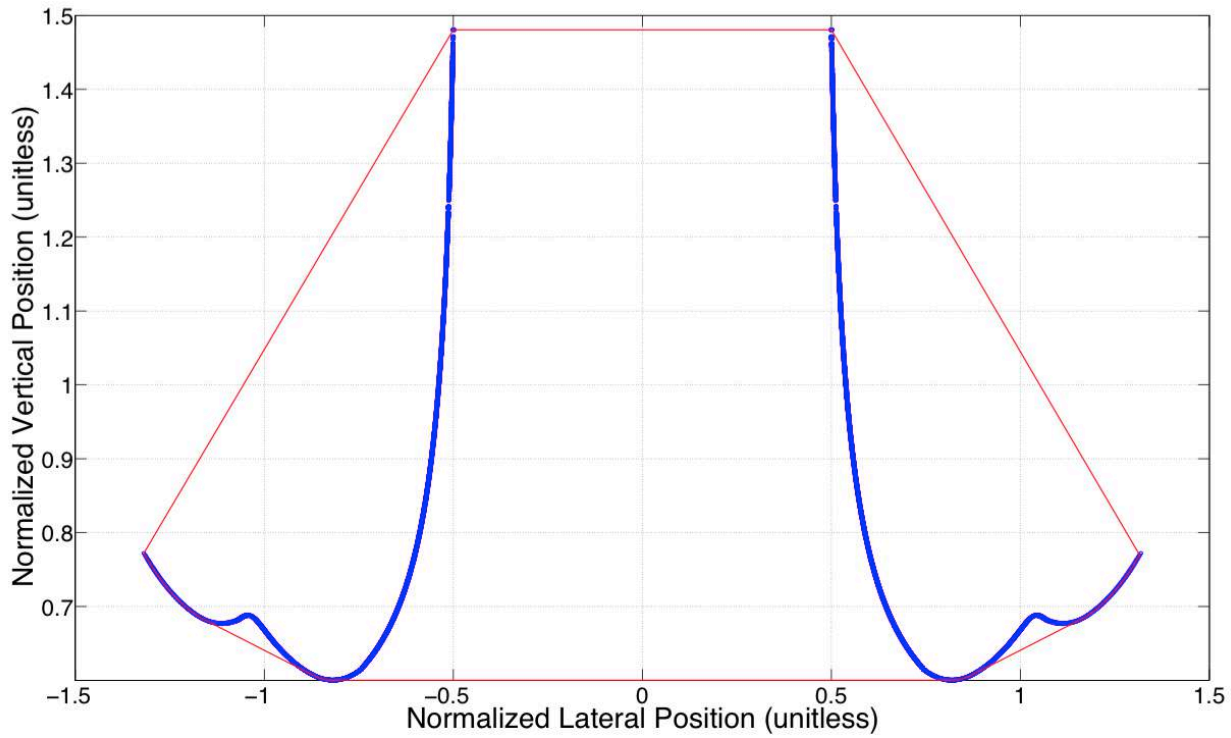


Figure 43 Front View of A Wake Zone in Ground Effect (IGE).

The model collects all potential wake encounters and writes them into a Matlab structure format, with relevant flight information for the two aircraft involved. The output includes flight identification, aircraft type, aircraft position when the encounter was detected, and aircraft flight path information suitable for Google Earth visualization. Figure 44 shows a screenshot of the Google Earth visualization of a potential wake encounter for two aircraft departing from San Diego International Airport. Flight tracks of the wake generator and the follower are plotted in different colors. A marker shows the position of the two aircraft when the encounter was detected. A red line defines the wake envelope zone trailing from the wake generator.

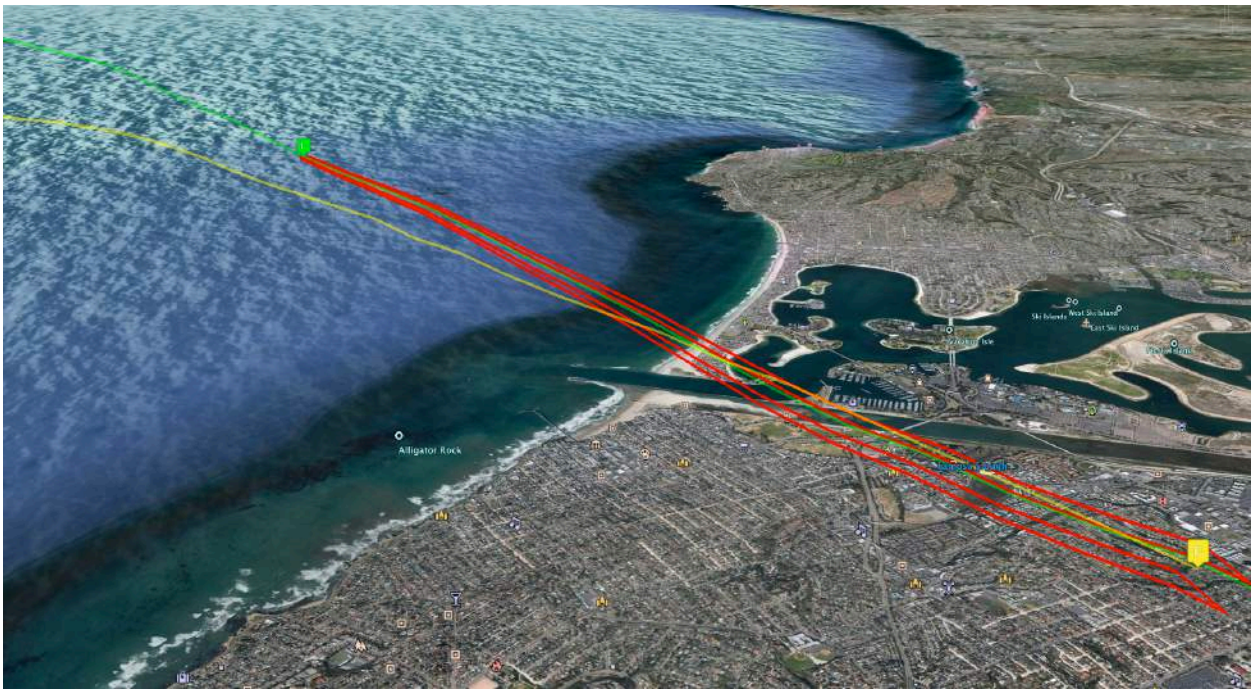


Figure 44 Google Earth Visualization for One Detected Potential Encounter.

The model has been integrated into a stand-alone computer program with a simpler wire-frame wake analysis model created by the George Mason University. Nicholas Hinze, a senior research associates in the Air Transportation Systems Laboratory (ATSL) at Virginia Tech, integrated both models and developed a graphic user interface for the stand-alone model. My contribution to the GUI was to develop the layout of the viewports, produce Google Earth ready wake zone information, and to produce the wake detailed outputs. Furthermore, I developed a users' manual for the model shown in Chapter 7.

4 EWEM Model Results and Analysis

The EWEM model is designed to work with different types of flight track data, including the Performance Data Analysis Reporting System (PDARS) radar data (Browder 2010), FlightAware output data, and synthetic flight track simulation data generated by programs such as The Terminal Area Route Generation, Evaluation and Traffic Simulation (TARGETS) (MITRE 2010) and Airspace Concept Evaluation System (ACES) (George and Wieland 2011). We have tested the model with four different types of data. This chapter will introduce all data types used for EWEM test and the running results of the model using each data type.

4.1 Performance Data Analysis Reporting System (PDARS) data

The FAA and NASA launched the PDARS project in 1997. The project has continuously collected flight radar track data from both Air Route Traffic Control Centers (ARTCCs) and Terminal Radar Approach Control (TRACON) facilities. The PDARS data contains flight track at the airport surface level and also flight track data of approaching and departing aircraft up to 50 miles from the airport (Browder 2010).

We tested the EWEM with PDARS data supplied by the FAA for an 18-day time period. The data included three different TRACON facilities, including Atlanta, New York and Los Angeles. The original data contains the flight identification, aircraft type, flight track waypoints flown and the time associated with each waypoint. Radar data have issues, such as missing waypoints, aircraft type information and unrealistic altitude or the flight track information shown as “spikes”. To manage these issues, the data was first smoothed to fill missing points or to replace unrealistic points. Then, the data was parsed into a Matlab structure file with estimated flight speeds and headings calculated from the radar information. For flight records without aircraft

type specified, EWEM assigns the smallest aircraft type in the BADA list (A general aviation aircraft) to the complete the flight information.

We ran 18 days of PDARS data from three TRACON facilities in EWEM with default settings. Very few potential encounters were observed. Figure 45 illustrates the only potential wake encounter observed at Los Angeles International Airport (LAX) on September 27, 2008. The "potential wake encounter" involves two departure flights using Runway 25R at LAX. The lead aircraft was a Boeing 737 and followed by an Airbus A318. The "potential wake encounter" occurred 97 seconds after the Boeing 737 departed the airport. At the position of the encounter, the Boeing 737 has flown 4.95 nm, and the distance between the two flights was 4.51 nm. The wake zone generated by the Boeing 737 is 5.01 nm with the circulation threshold set at 125 m^2/s for an Airbus 318. The estimated circulation strength the Airbus 318 encountered was about 129 m^2/s.

For 13419 flights on September 27, 2008, the probability of having a potential wake encounter is estimated to be 0.0745‰. The NASA ASRS reporting system estimate a potential wake encounters rate of 0.0089‰. It is known that wake encounters are under-reported in the ASRS database. Several factors can contribute to the differences observed between EWEM model and the ASRS system. The model estimates flight speed and employs stochastic weight factors to simulate the aircraft state. The result could be biased and need further validation of such weight factors need to be investigated. Such data is difficult to obtain because landing and departure aircraft weights are proprietary to the airline. In its stage of development, the model used real data flight track. The broad sanity checks can help us evaluate the credibility of the circulation thresholds used as default settings in the model. Unfortunately, numerous requests to obtain flight track data of actual wake encounters incidents was declined by the FAA sponsor.

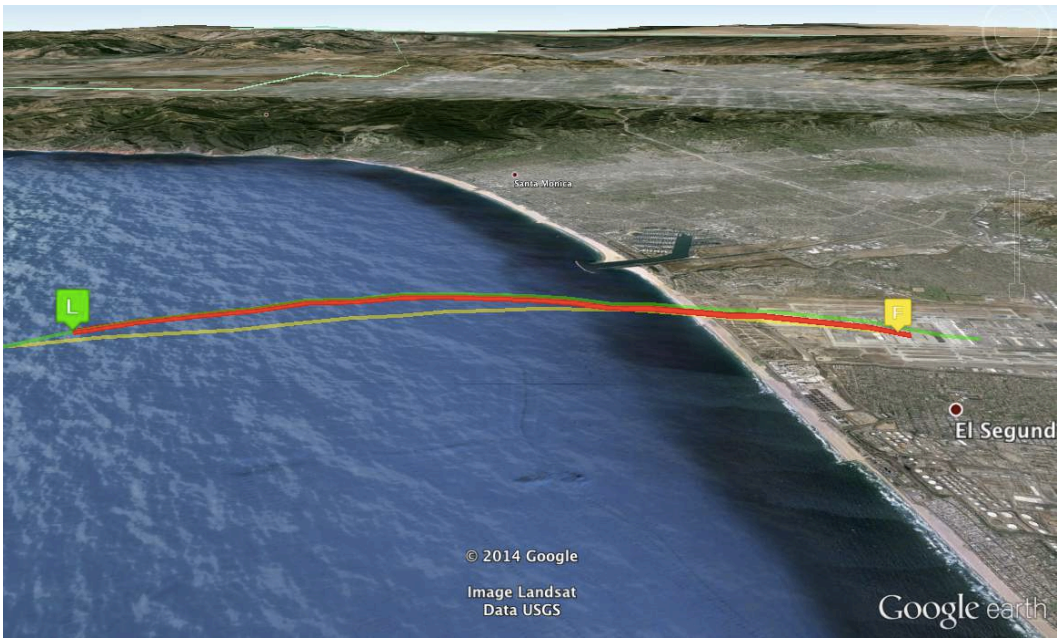


Figure 45 Potential Wake Encounter Detected near Los Angeles International Airport on September 27, 2008.

4.2 FlightAware Data

FlightAware is an online flight tracking service. It provides both private and commercial air traffic information. The website allows users to track live flights or download historic flight data from their servers. Waypoints reported by FlightAware come from three sources: 1) air traffic control centers, 2) ADS-B equipped aircraft, and 3) estimated points based on nearby reporting points. Similar to PDARS data, the FlightAware data is considered to represent actual flights and is subject to position report errors. In theory, FlightAware data should be conflict free but not necessarily free of potential wake encounters. Figure 46 provides a sample of FlightAware data.

Time	Position		Orientation		Groundspeed		Altitude		Reporting Facility
EDT	Latitude	Longitude	Course	Direction	KTS	MPH	feet	Rate	Location/Type
02:36PM	42.8145	-85.4242	173°	South	275	316	6,000		Cleveland Center
02:37PM	42.8000	-85.3108	100°	East	297	342	8,600	2,580	Cleveland Center
02:38PM	42.4449	-85.4427	166°	South					FlightAware Approximate
02:38PM	42.7808	-85.1711	101°	East	311	358	11,400		Cleveland Center
02:39PM	42.3614	-85.4167	180°	South					FlightAware Approximate
02:39PM	42.7650	-85.0494	100°	East	326	375	13,000		Cleveland Center
02:40PM	42.2759	-85.4094	173°	South					FlightAware Approximate
02:40PM	42.7486	-84.9192	100°	East	345	397	14,800		Cleveland Center
02:41PM	42.1908	-85.3944	173°	South					FlightAware Approximate
02:41PM	42.7303	-84.7897	101°	East	347	399	16,600		Cleveland Center
02:42PM	42.6861	-84.6822	119°	Southeast	335	386	18,400	1,140	Cleveland Center
02:43PM	42.5939	-84.6336	159°	South	324	373	19,000	240	Cleveland Center

Figure 46 FlightAware Track Log Sample Data.

To test the FlightAware data with EWEM model, we used a set of three-day containing instrument flight rule flights in the East Coast of United States around the Cape Canaveral. The dataset contains 12,824 flights and one “potential wake encounter” was observed. Figure 47 presents the potential wake encounter found. From the figure, it is clear that the flight track has abnormal zigzag behavior resulting from the estimated points in the FlightAware algorithm. The encounter was due to the inaccuracy of the data and cannot be considered legitimate. As mentioned earlier, a combination of radar and estimation data, the flight track can yield unrealistic flight track data.

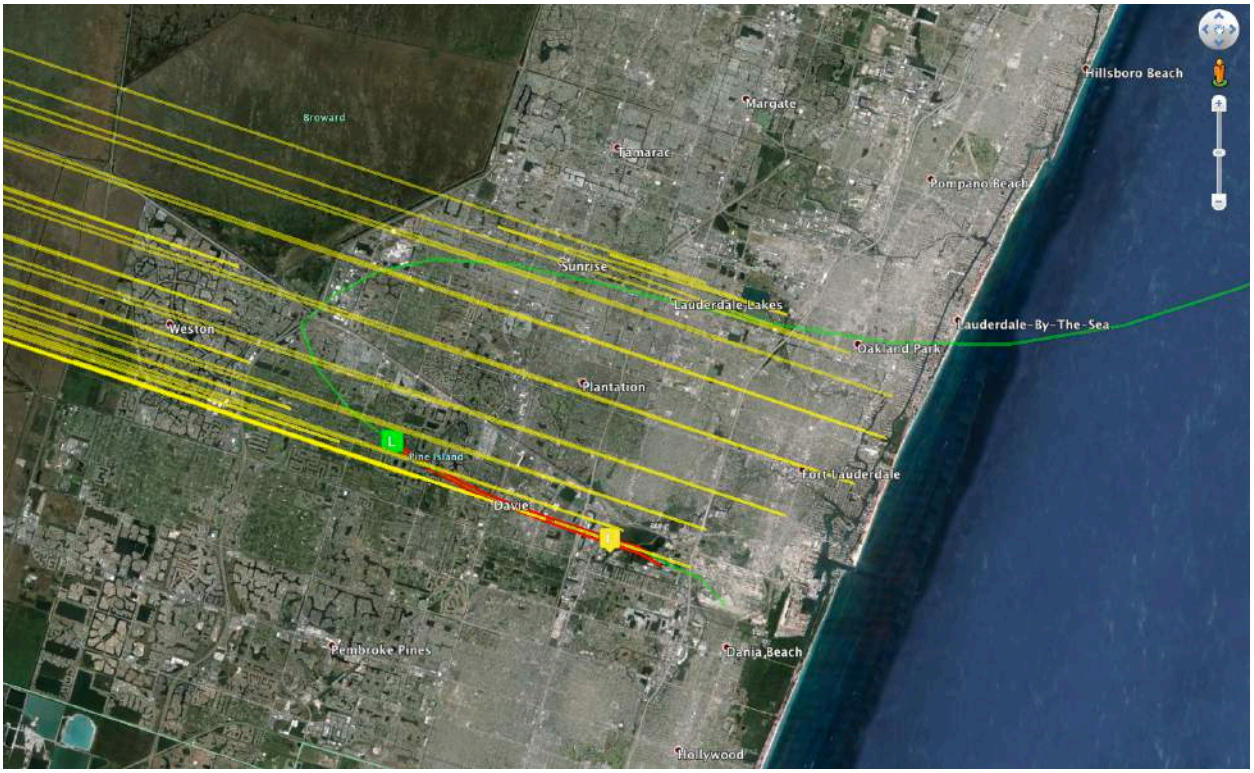


Figure 47 Potential Wake Encounter Detected by EWEM with three-days of FlightAware Data.

4.3 The Terminal Area Route Generation, Evaluation and Traffic Simulation (TARGETS) data

TARGETS is a simulation tool developed by MITRE under the sponsorship of the FAA. TARGETS is being used by the FAA to design new procedures. With the visualization and simulation capability of the model, air traffic controllers can be design new procedures and verify their “flyability”. However, TARGETS does not considered wake turbulence effects in the

design of procedures. The sample data we used in EWEM for validation are procedures designed by TARGETS (MITRE 2010).

TARGETS data was provided in the form of two separate files: 1) flight information and 2) detailed track data. TARGETS flight information produces waypoints at 5-second intervals. The data includes aircraft type and flight speed. For test purposes, the sample data received was generated by CSSI, Inc. The set contains 148 departure flights from runways 36L, 36C and 36R at Memphis International Airport. With no crosswind atmospheric conditions, no potential wake encounter was observed. Flight tracks are shown in the Figure 48.

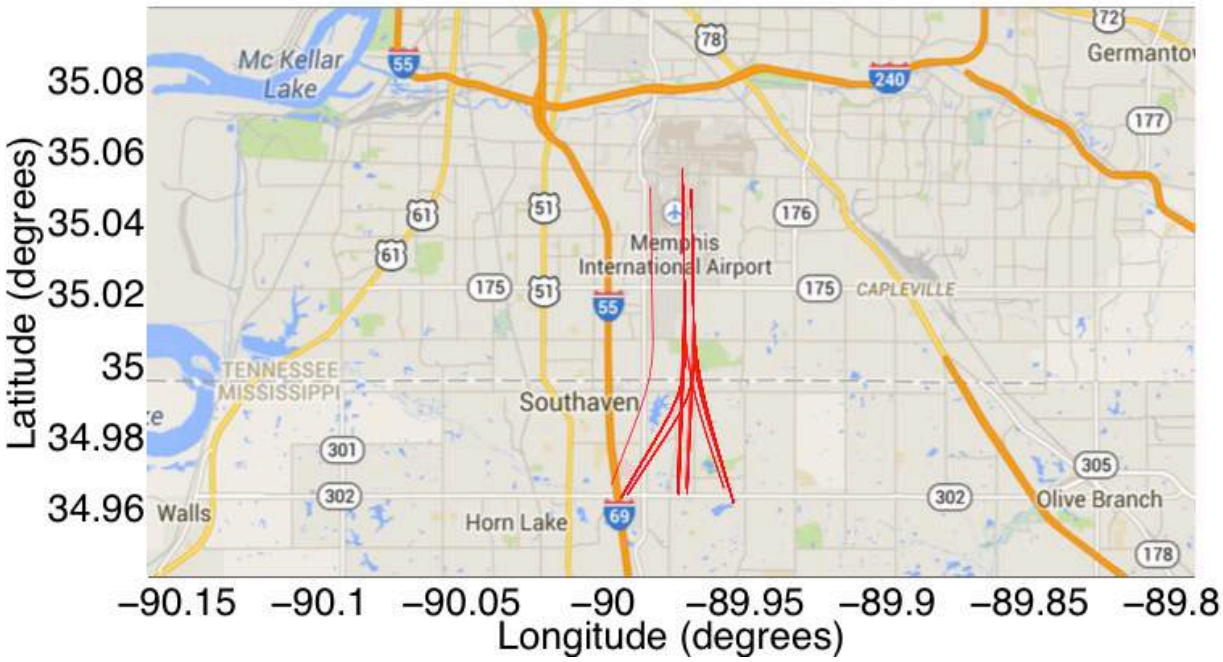


Figure 48 TARGETS Generated Flights at Memphis International Airport.

Twenty-two flight pairs were observed to have “close operations”. This is known because departures are released with one-minute headways from either the same runway or parallel runways. Figures 49 and 50 show two types of operations and the distance between two successive flights during these operations.

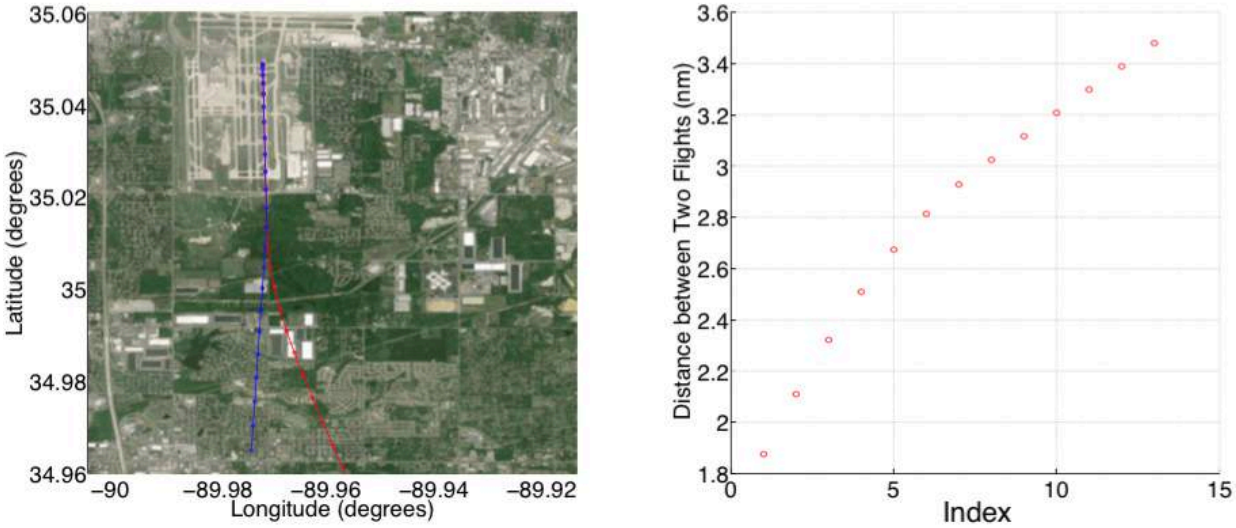


Figure 49 Successive Flights Depart from Runway 36R at Memphis International Airport.

Figure 49 shows two flights departing from runway 36R in succession. The aircraft were separated initially by 1.5 nm, which is 60 seconds of headway. The envelope constructed by the EWEM for a Boeing 727-200 is shown in Figure 50.

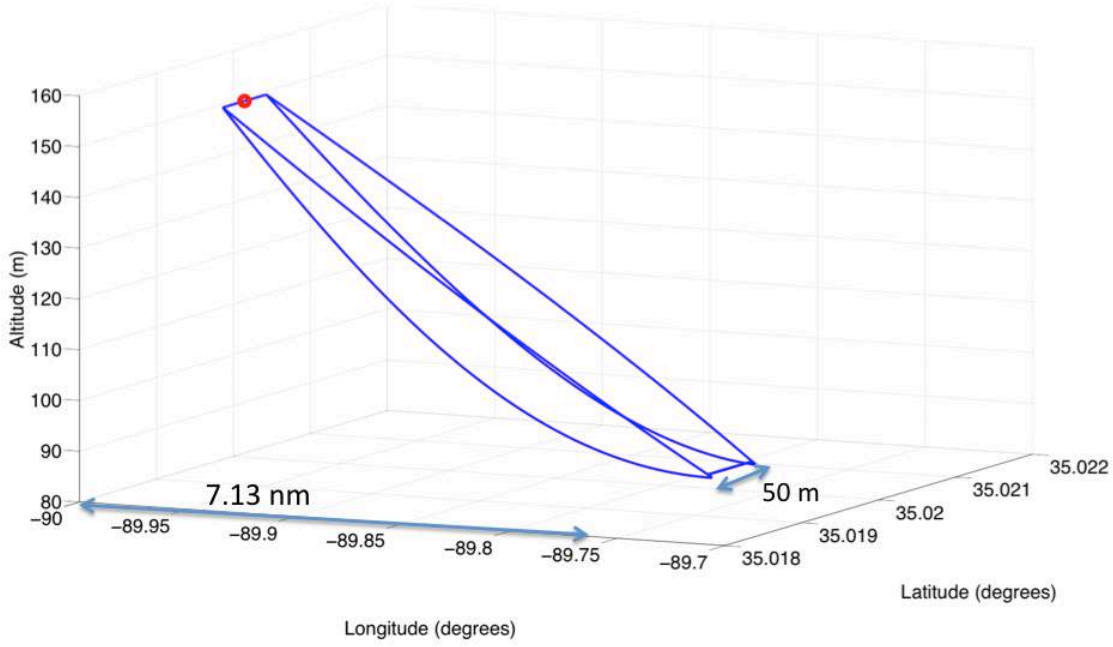


Figure 50 Three-Dimensional Envelope Projected for a Departing Boeing 727-200 Class Vehicle.

The envelope is about 7.13 nm long, 50 m wide and involve a 60 m vertical descend. The minimum separation rule for successive flights on the same runway is 60 s. Based on the BADA

model profile, a Boeing 727-200 in the initial climb a speed of 151 knots and thus can travel 1.3 nm in 60 seconds while climbing at 420 meters per minute. According to this analysis, the following aircraft will not penetrate the protected wake zone of the lead aircraft.

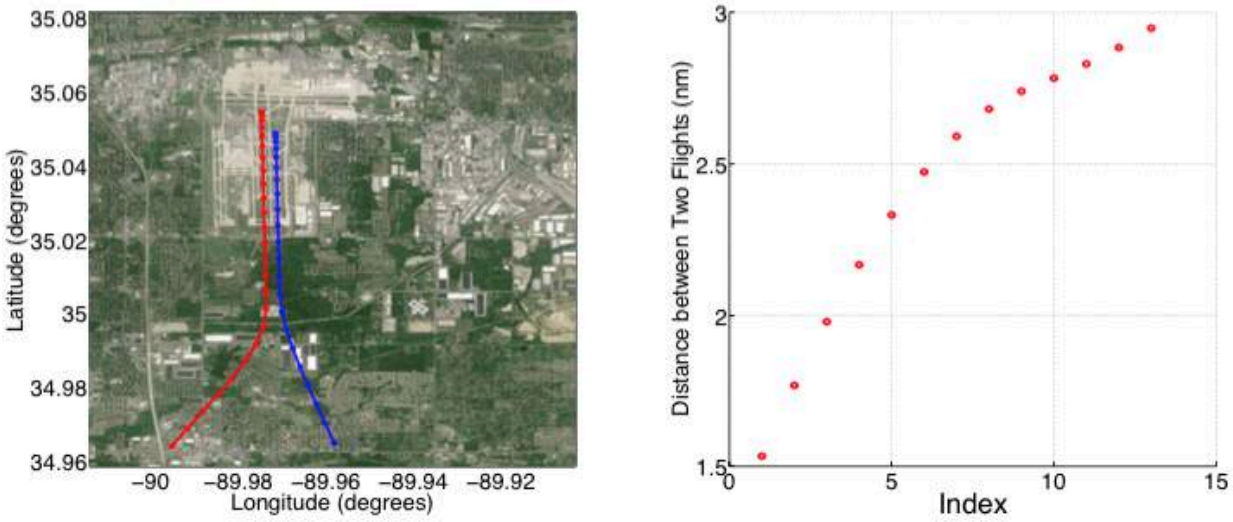


Figure 51 Parallel Operations on Runways 36R and 36C at Memphis International Airport.

For parallel operations, the runway separation between runways 36 R and 36 C at Memphis International Airport is 290 m. Under zero wind conditions, the wake generated from the lead aircraft will not reach the close parallel runway. Thus, no potential wake encounter is likely to be observed.

4.4 Airspace Concept Evaluation System (ACES) data

ACES is a modeling and simulation system developed by NASA Ames Research Center (George and Wieland 2011), to help assess new aviation operational concepts and technologies. The model provides detailed traffic information, including dynamic evolution of traffic and delays as outputs (George and Wieland 2011). To help understand the impact of wake separations in the en-route, EWEM was used to study potential wake encounters is en-route separations are reduced. This study provides a first-order analysis of the potential risk from reducing en-route separation minima.

For the study we used a 70,000 flight ACES flight tracks representing en-route operations over a 48-hour period. Flight tracks studies contain 5-second position information with typical file size at 6 gigabytes. To parse these data into EWEM recognizable format, we divided the ACES output file into dozens of subsets. To keep the data consistent, we use a four-minute overlap between subsets. Thus, close operations will not be eliminated or ignored from the data partitioning. Each subset will be analyzed and reported by EWEM separately.

The flights in the ACES output were generated without regard to conflict detection in the flight generation. Thus, unrealistic and close operations were included in the data set, as shown in Figures 52 and 53.

Figure 52 shows some random pairs from the first 10-20 minutes of the dataset that have very close separations. Each line represents the separation between a pair of departure flights from the same airport. The minimum in-trail separation for departures is 60 seconds, which equates to 1 to 1.5 nm. For parallel operations, separations between close aircraft could be low, but the separations are expected to increase as soon as they takeoff from the runway. Most lines shown in Figure 52 have less than 1 nm for the entire 60-second interval.

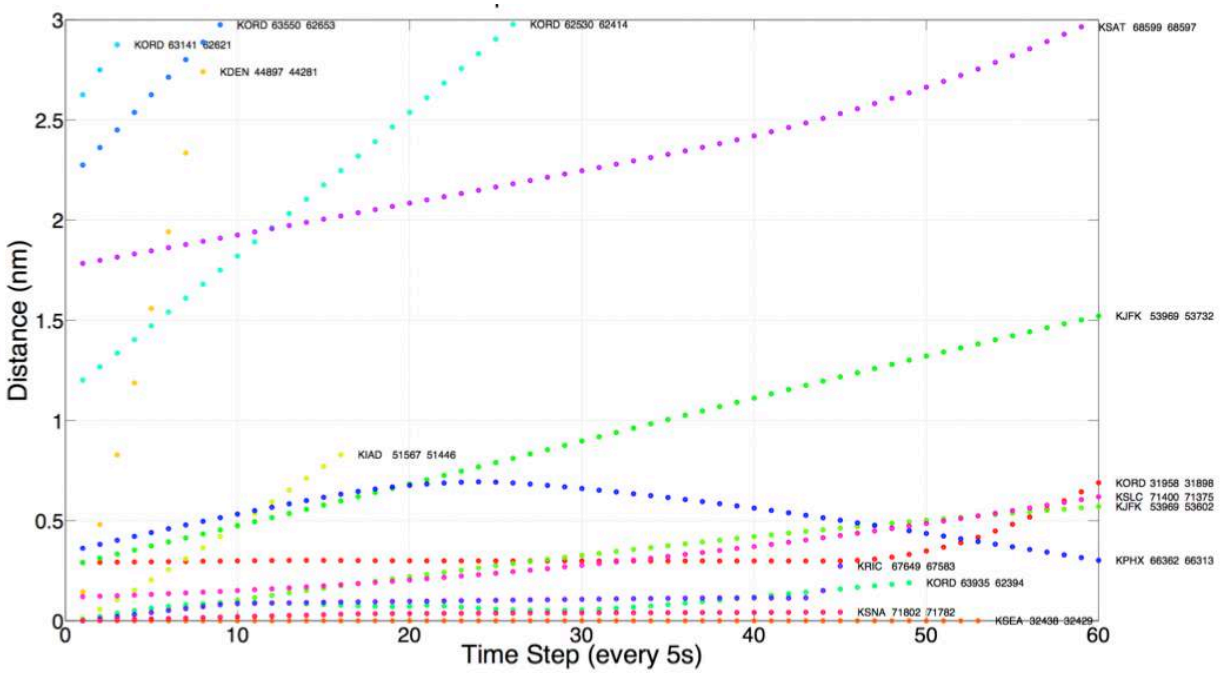


Figure 52 Flight in ACES Data with Close Separations.

One of the selected pairs is plotted in Figure 53. As shown in the figure, the two flights are following the same flight path before reaching the cruise level with very limited separation.

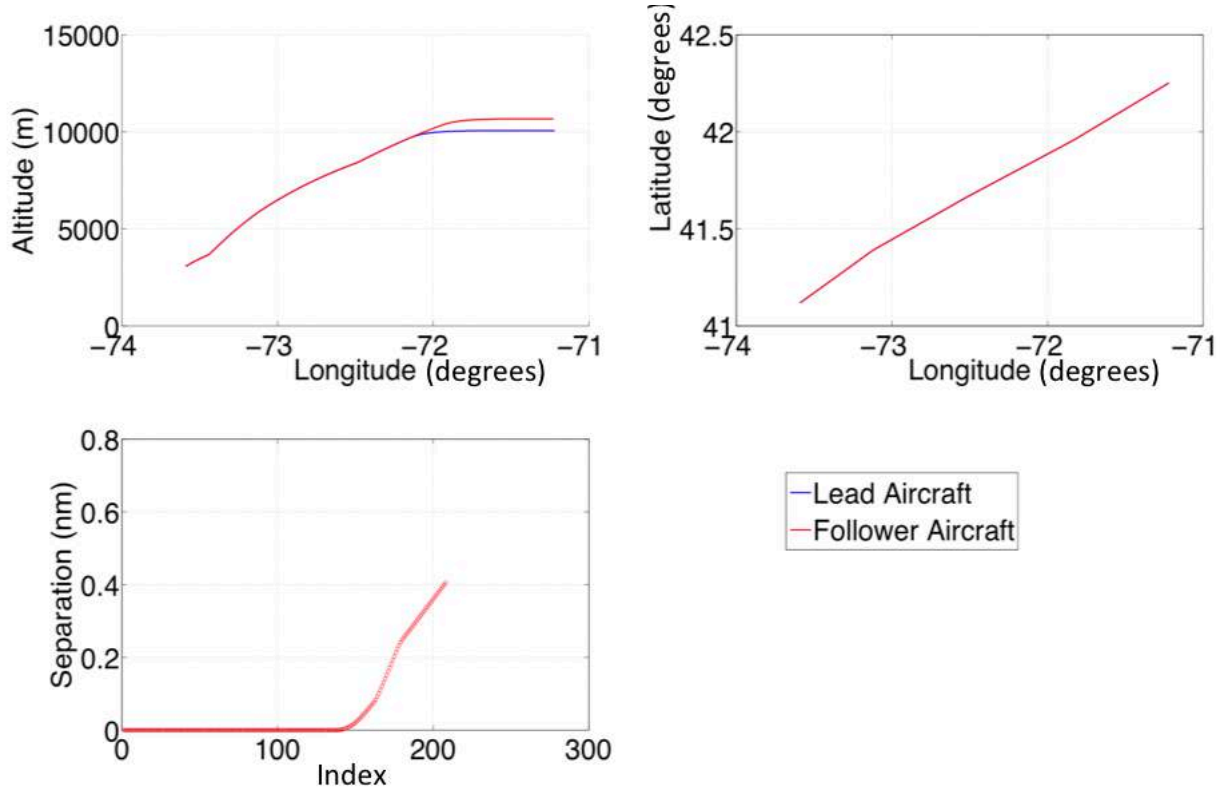


Figure 53 Two Aircraft Follow the Same Track with Very Limited Separation in ACES Dataset.

Without conflict detection, some flights, as shown in Figures 52 and 53, depart from the same airport near the same time. Moreover, these flights have the same destination airport with the same flight plan. These tracks are sharing the same record or only separated by a very small interval.

4.4.1 Conservative Wake Zone

The wake envelope generated in the EWEM trails the generator and descends. If two flights follow the same flight path at the same altitude separated by a few seconds, the model will not detect the situation as a potential wake encounter. To detect and filter these flights, we developed a conservative wake zone for more general analysis purposes. The conservative wake zone has a flat envelope top compared to the slanted descent top of a realistic wake zone, as shown in Figure 54. The figure shows the differences between the two types of envelopes. The realistic wake zone, bounded by the red solid line, presents the real wake behavior extracted from the APA Model. Such wake zone can only detect Flight B in the illustration. The conservative wake zone is constructed with flat envelope top, drawn with a blue dotted line. This expanded zone protects larger space and is able to detect close in-trail flights such as Flight A in the figure. The conservative approach can help detect close flights in the ACES analysis.

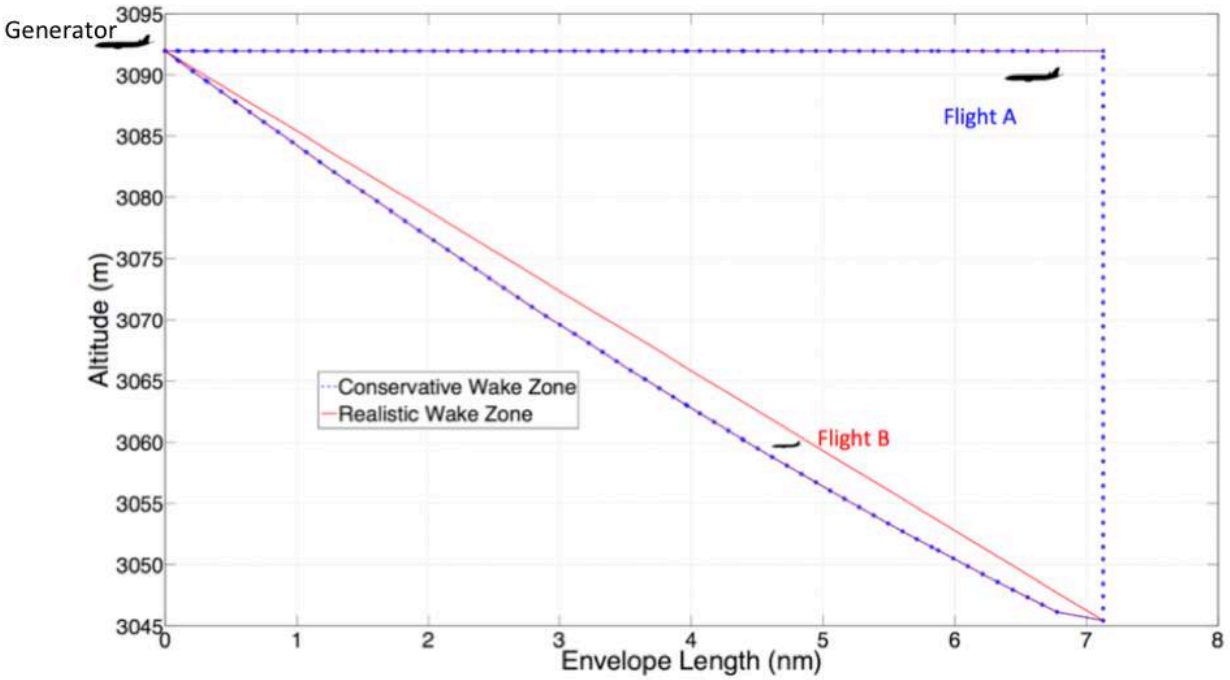


Figure 54 Differences between Two Types of Wake Zones. Realistic Wake Zone vs. Conservative Wake Zone for an Airbus 300-600 Class Vehicle.

The EWEM model have successfully detected 933 potential encounters out of 70,000 flights using the conservative wake zone mode, and 99 potential encounters with realistic wake zone mode. The ACES data used in the analysis contains flight tracks above 5000 ft. Thus, no encounters were detected near the ground. Most of the encounters were detected between 10000 feet and 15000 feet under realistic wake zone mode. Under the conservative wake zone mode, most encounters were detected above 25000 feet. The results are summarized in Figure 55.

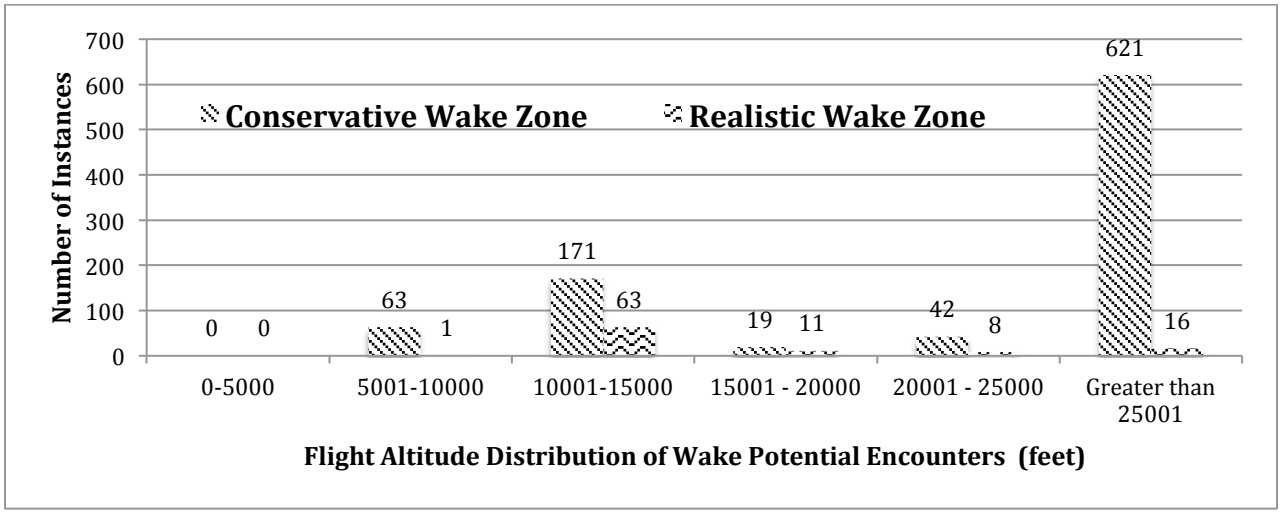


Figure 55 Flight Altitude Distribution of Wake Potential Encounters.

Figure 56 shows a clear distribution of the distance between the wake producing aircraft and the follower when the wake encounter occurs. Most of the encounters detected under the realistic wake zone happened at the distances below 5 nautical miles, the current minimum en-route separation rule. The average distance between two flights when encounters were detected is 4.8 nm.

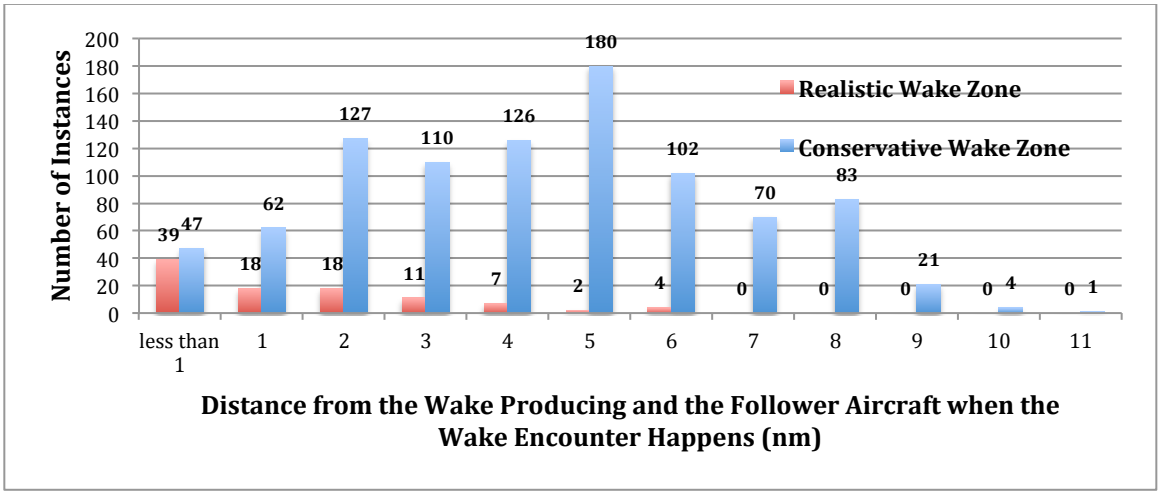


Figure 56 Distance from the Wake Producing Aircraft when Encounter Happens.

4.4.2 Potential Wake Encounter Analysis by Category

Table 19 shows that 68% of the potential wake encounters involve a ReCat D aircraft following. 93.6% of the aircraft is penetrating a wake belong to ReCat D and E. A comparison of results obtained with EWEM and the results reported by the ASRS system, is presented in Table 20. The distribution is close. The percent of potential wake encounters behind ReCat groups A and B is small compared to the ASRS reporting system. The ASRS reporting system is a voluntarily system based on pilots' judgment. As such, it is more likely to contain reports of strong turbulence encounters generated by larger aircraft types. The categorization system used in the ASRS report system is not the same as the ReCat system used for the ACES data. Among all the flight tracks provided in the ACES data, no aircraft from ReCat A was generated, only 6% of the

aircraft belong ReCat groups B and C. More than 69% of the flights in the ACES data belong ReCat groups D and E.

Table 19 En-Route Wake Encounters Summarized by Category from EWEM with Conservative Wake Zone.

		Wake Encountering Aircraft Follower						
		A	B	C	D	E	F	Total
Wake Producing Aircraft Leader	**A**	0	0	0	0	0	0	**0**
	B	0	3	0	3	0	0	**6**
	C	0	0	18	16	11	0	**45**
	D	0	6	28	528	75	0	**637**
	E	0	1	4	123	116	0	**244**
	F	0	0	0	0	1	0	**1**
		0	**10**	**50**	**670**	**203**	**0**	

Table 20 En-Route Wake Encounter Summarized by Wake Groups from ASRS System Database.

		Wake Encountering Aircraft Follower				
		Heavy	B757	Large	Small	Total
Wake Producing Aircraft Leader	Heavy	5	1	19	4	29
	B757			3	2	5
	Large	1	2	8	4	15
	Small				1	1
	Unknown	4	1	8	1	14
		10	4	38	12	

4.4.3 Wake Encounter Analysis in Transition Flight Phase

Based on report by the EUROCONTROL, most of the potential wake encounters occur during the transition phase of flight (Hoogstraten 2013). As shown in Table 21, all the potential wake encounters detected using realistic wake zone mode in EWEM occur during the transition phase of flight. For the conservative wake zone mode, 49% of the encounters were reported during the en-route cruising phase due to the “very close operations”.

When comparing the results with the flights distance in Figure 56, we conclude that those potential wake encounters detected with less than 1 nm separation are involve close operations with small vertical separation. The current vertical separation for en-route flights is 1000 feet. However, potential wake encounters could occur when flights climb and pass other cruising flights. More wake vortices studies need to be conducted for flights under en-route scenario.

Table 21 Encounters Analysis in Transition Phase.

		Wake Encountering Aircraft Follower					
		Descent		Cruise		Climb	
		Conservative	Realistic	Conservative	Realistic	Conservative	Realistic
Wake Producing Aircraft Leader	Descent	162	44	80	10	36	4
	Cruise	28	0	456	0	16	1
	Climb	15	0	24	0	116	40

4.4.4 Capacity Analysis with ACES

The main goal to evaluate potential wake separation benefits is to reduce the over all en-route separation from the current standard of 5 nm to a proposed separation of 3 nm. The aircraft conflict pair filter in EWEM was originally set to 5 nm. This means that flights within the 5 nm boundary from the wake generator will be considered as a candidate for a potential wake encounter and needs to be further analysis to check for penetration of the wake zone. After setting this filter to 3 nm, flights that are separated from the wake generator by 4 nm will no longer be considered as a potential pair for wake encounters analysis in EWEM. This is illustrated in Figure 57 (the red shadow zone). If no potential encounters are detected after changing the screening criteria, the proposed 3 nm rule could be stated. Otherwise, if a large portion of potential wake encounters is detected within the area, further analysis of the aircraft pair is necessary.

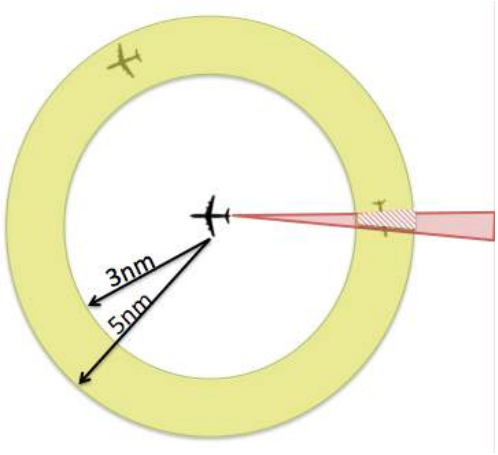


Figure 57 Concept of Reducing En-Route Separation Criteria from 5 nm to 3 nm (Top View).

For all the EWEM model runs using the ACES data, there were 18 additional potential wake encounters detected while reducing the en-route separation from 5nm to 3nm. Similarly there were 236 “incidents” were detected in the red shadow zone shown in Figure 57. All of these 18 involved flights that were descending or climbing.

As a general observation, we know that potential wake encounters will increase by 20% when the en-route separation criteria is reduced from 5nm to 3nm. However, the analysis involved flights that were never de-conflicted by the ACES simulation model. For this reason, the flights may not fully represent the flights in the national airspace system. The results using de-conflicted flight data are expected to produce significantly fewer number of potential wake encounters.

Therefore, the wake zone constructed in EWEM might be large comparing to hazardous space for following aircraft. The current circulation thresholds set in the model might be relative low, especially for larger aircraft types. Further investigations can be preformed to adjust the circulation threshold levels and with updated ACES data, a more realistic detailed research can be thus conducted.

5 Model Validation

To improve the credibility of the EWEM model, the best approach is to validate the model with historic data. Examine the model's result by recreate historic wake encounter events with accurate flight track and wake encounter data. However, no data was provided by the FAA. This chapter will describe two historic wake turbulence encounter events and examine the potential wake encounter result by using EWEM model.

5.1 Wake Turbulence Encounter Analysis at Sydney Airport

On November 3, 2008, a SAAB 340B encountered the wake turbulence generated from an Airbus 380 executing a final approach to runway 34R at Sydney airport (Australian Transport Safety Bureau 2009). One passenger sustained minor injuries, and the SAAB 340B had a momentary loss of control. A Bombardier Inc DHC-8-400 was 6.5 nm ahead of the SAAB 340B on final approach for runway 34R. The Airbus A380-800 was about 3.7 nm on final approach to the runway 34L ahead of, and to the left of the SAAB 340B at the time of the upset. The SAAB 340B was about 259 m (850 feet) to the right of the 34R centerline. The reported surface winds at the time were 270 ˚ at 8 knots, and increased gradually to 246˚ and 35 knots at 2400 feet. The situation is visualized in Figures 58 and 59.

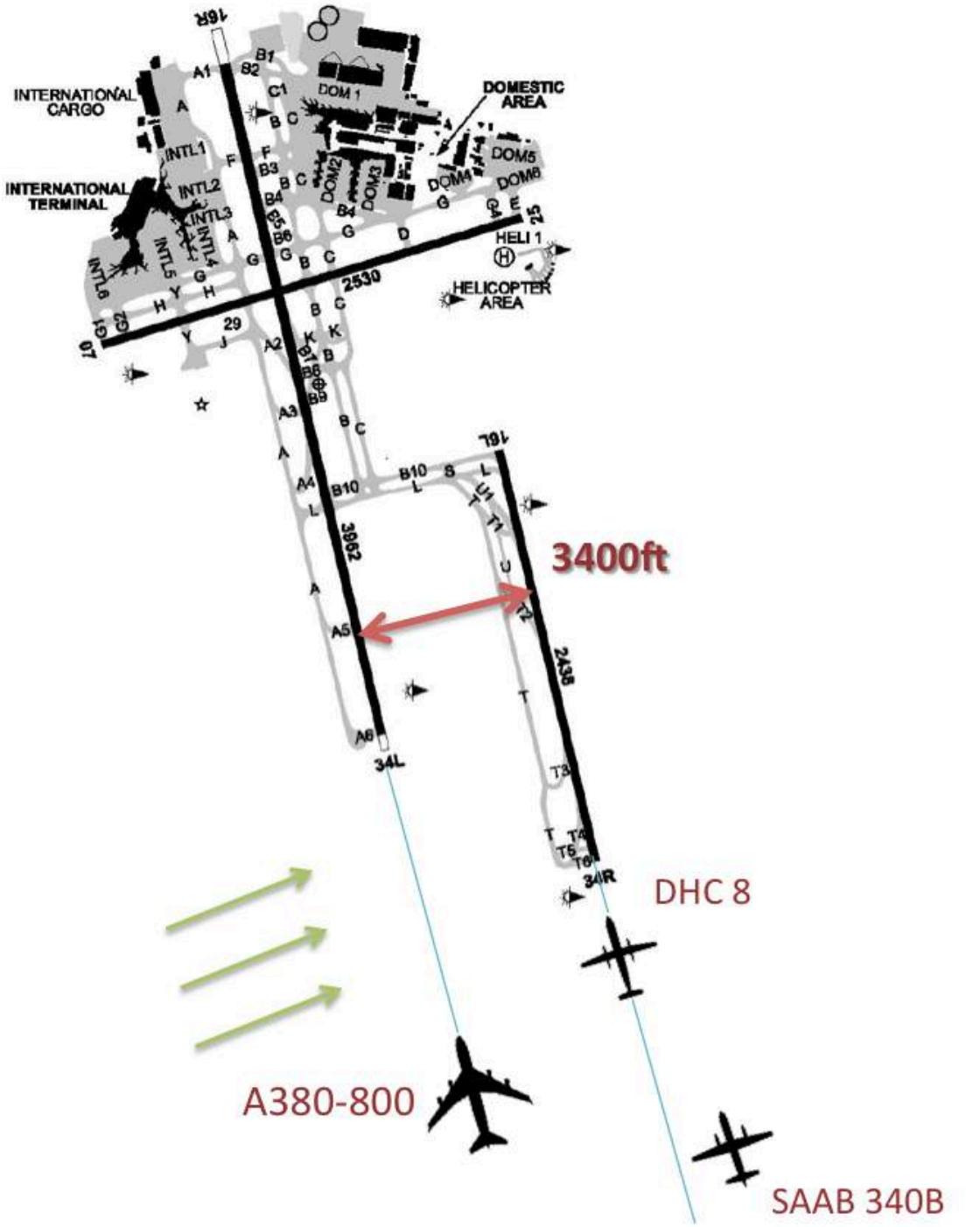


Figure 58 Visualization of the Wake Turbulence Event at Sydney Airport on November 3, 2008.

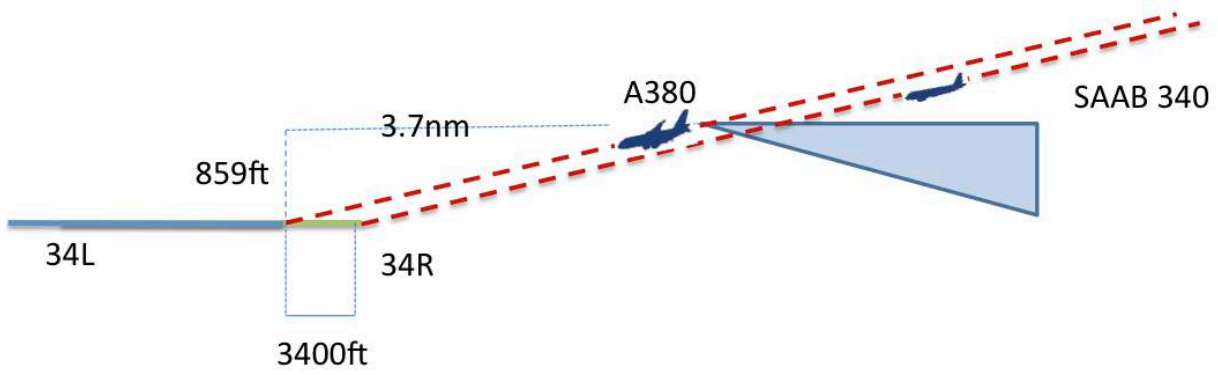


Figure 59 Visualization of the Wake Envelope at the Sydney Airport on November 3, 2008.

Using the atmospheric conditions reported, we used EWEM to simulate the scenario. Since the detailed flight information was not available, we generated a stand-alone Airbus A380-800 wake zone for known given flight conditions. The wake envelope generated for this Airbus A380-800 class vehicle is presented in Figure 60.

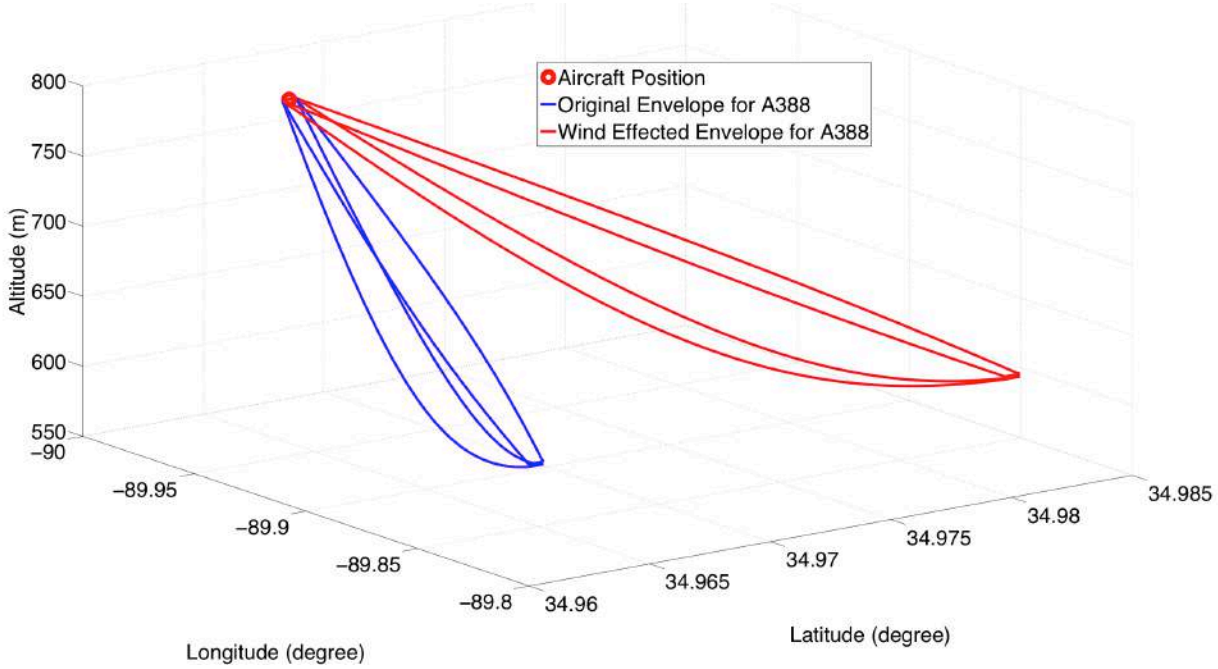


Figure 60 Three-Dimensional Wake Envelope Build for Airbus A380-800 Class Vehicle.

Based on the report, the estimated lateral distance between the Airbus A380-800 and the SAAB 340B was 4250 feet (1259.4 meters), the runway separation of 3400 feet (1036.32 meters) with an addition 850 feet (259.08 meters) offset between the SAAB 340 and the runway 34R centerline. The longitudinal separation between the two aircraft was estimated to be 3.8 nm.

Figure 61 shows the top and side views of the wake envelope generated for the Airbus A380-800. The envelope has a vertical drop of 200 m and a lateral transport of 1812 m due to the high crosswind condition. The estimated position of the SAAB 340 is also projected on the envelope's top view based on our distance calculation. The EWEM analysis shows that the SAAB 340 was piercing the Airbus A380-800 wake envelope.

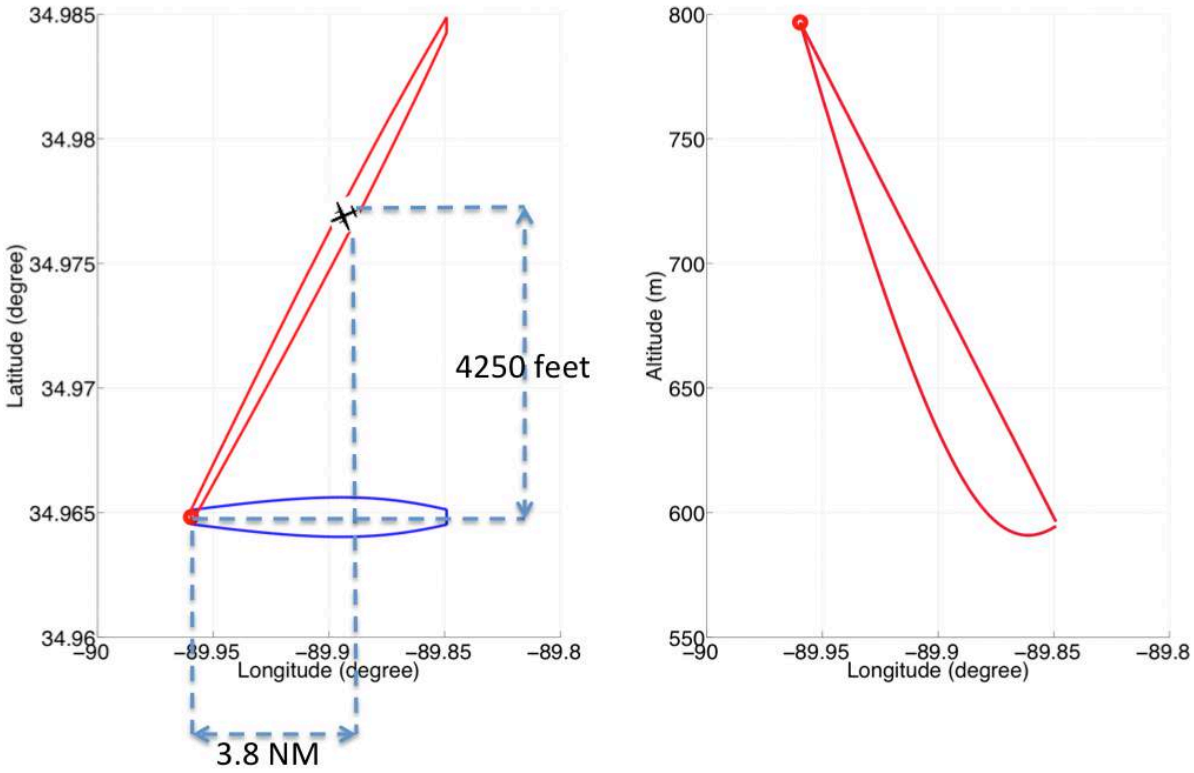


Figure 61 Top and Side Wake Envelope Views of An Airbus A380-800 Class Vehicle. (SAAB 340B's Position also Shown)

The potential wake encounter circulation level experienced by the SAAB 340B can be estimated using (nm) Equation (15):

$$\text{Circulation\_Strength} = (D/L) * (IC - CT) + IC \qquad (15)$$

Where, D (nm) is the distance between two flights when the encounter occurs; L (nm) is the total envelope length; IC (m^2/s) is the initial circulation strength calculated from Equation (2). CT (m^2/s) is the circulation threshold defined in Table 15. The estimated circulation strength of this

encountered was 348 m^2/s, which is strong based on our default circulation boundary for a SAAB 340B, which belongs to ReCat group E.

5.2 Mexico City Wake Encounter Accident

The Mexican Interior Ministry aircraft crashed in Mexico City around 18:45 local time on November 4, 2008. The Interior Secretariat owned Learjet 45 was 12 km short of landing at Mexico City International Airport when it crashed. According to the accident report, the Learjet was following too close a Boeing 767-300 heavy jet operated by Mexicana Airlines, and encountered a strong wake turbulence causing the corporate jet to lose control (Transportes 2009).

A video was published on Youtube for the accident including the radar view and the controller and pilot voice communications. Figure 61 shows the screen capture from the video. Since no historic flight track data was published, a simulated similar trajectories to Mexico City Airport Runway 5R were extracted using FlightAware data. This is shown in Figure 63.

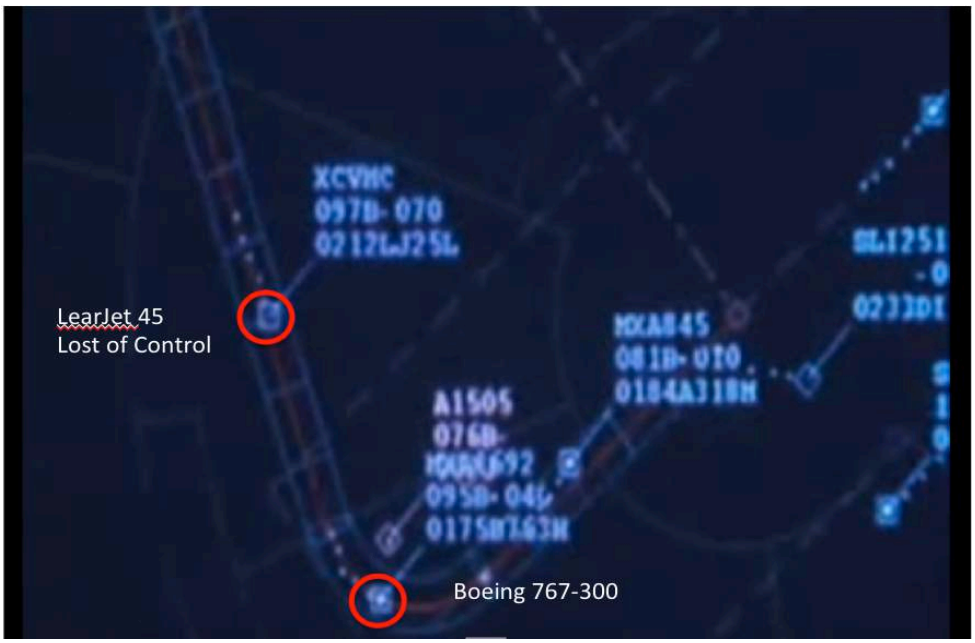


Figure 62 The Screen Capture of the Youtube Video when the Encounter Happened.

Figure 63 presents a satellite image of the area and trajectory extracted using FlightAware flight tracks and visualized using Google Earth. The picture shows the similarity of the simulation tracks with the original track recorded in the video shown in Figure 62. Aircraft flying approaches to Runway 5R at Mexico City Airport follow a very narrow corridor (about 0.5 miles wide) due to terrain.

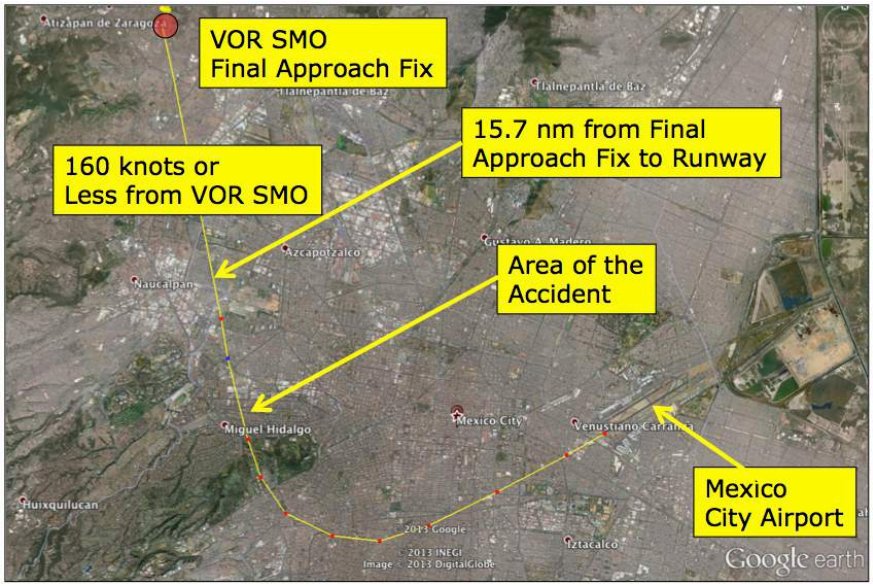


Figure 63 FlightAware Flight Track Approach to Mexico City Runway 5R Visualization in Google Earth.

The model examined two flight tracks gathered from FlightAware using modified flight speed and altitude based on the video information. The simulated tracks are separated with one minute and 20 seconds in trail as reported in the video sequence and in the accident report. Figure 64 presents the trajectories for two flights with the projected envelope of the leading aircraft. The lead aircraft Boeing 767-300 flight track is plotted in green dotted line. The Learjet 45 flight track is shown in red dots. The figure clearly shows that the Learjet track pierced the wake envelope generated by the Boeing 767-300.

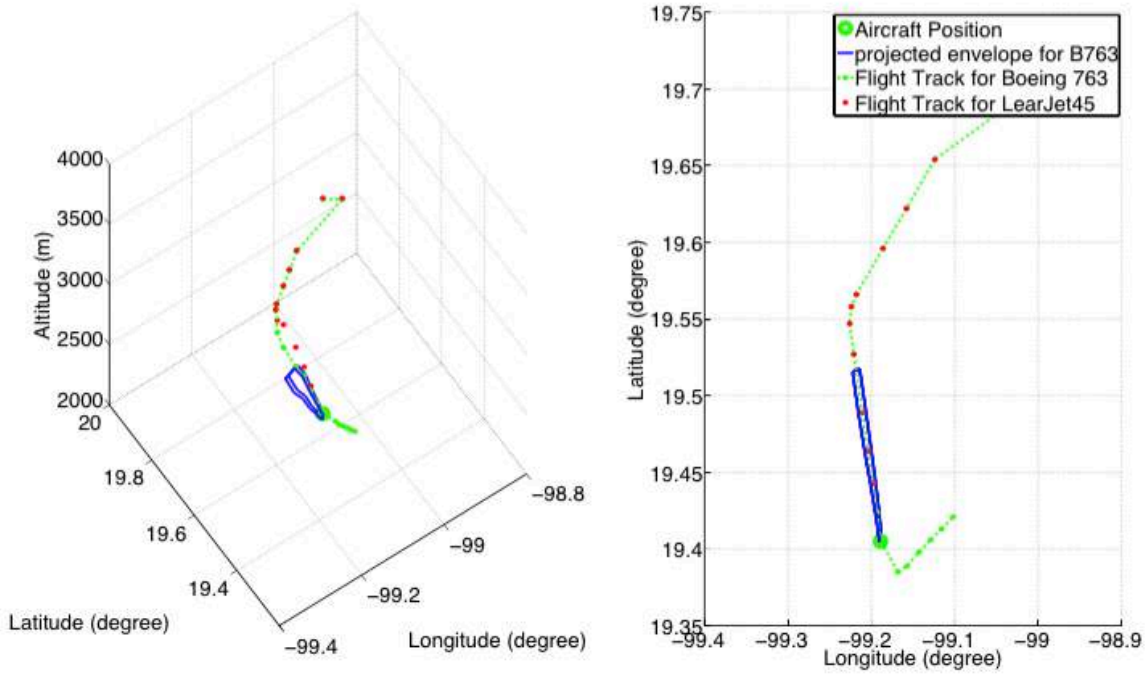


Figure 64 Wake Envelope of a Boeing 767-300 Class Vehicle (3D View and Top View).

The projected envelope for the Boeing 767-300 at the point in the flight profile where the accident occurs is about 8.44 nm long with a circulation threshold of 100 m^2/s for a LearJet 45 (ReCat Group F). The distance between two aircraft is estimated to be 3.7 nm when the loss of control sequence started. According to the video using EWEM, we estimated the wake turbulence strength to be 466 m^2/s calculated from Equation (15), which is considered to be a severe wake to an aircraft in ReCat F.

Similar to the Sydney case analyzed in Chapter 5.1, no confirmed historic data was obtained. The accident reconstruction was performed to validate the results obtained with the EWEM model. Flighttracks and weather conditions were estimated and the analysis was based on assumptions obtained from the accident reports (Australian Transport Safety Bureau 2009) (Transportes 2009).

6 Model Improvement

The main purpose of EWEM model is to assist the FAA procedures designers and air traffic controller to consider potential wake encounters in the design of future procedures. The model developed can be used as an additional assessment tool of available simulators to help develop new operational concepts. The EWEM model employs the NASA APA model algorithms, which have been validated with LIDAR data. However, further model validations are needed for this model to understand the levels of wake encounters.

As stated for case analysis in Chapter 5.1, the circulation threshold set for the aircraft type SAAB 340B is 100 m^2/s. The model provided the estimation of the encounter with the turbulence circulation strength of 350 m^2/s. Based on the safety report, one crew member reported that "a passenger sustained minor injuries following a possible wake turbulence event that resulted in a momentary loss of control of the aircraft" (Australian Transport Safety Bureau 2009). The ASRS has defined different levels of wake encounter severities based on the effects to the trailing aircraft. The magnitude of the effects is shown in Table 22. Compared to the ASRS published guidelines, the incident could be classified as server. If the SAAB 340B incident is classified as severe level, it is very likely that severe encounters will be reported by EWEM if the circulation threshold increased to 300 m^2/s instead of the 100 m^2/s for aircraft ReCat group F. The current circulation threshold levels in EWEM are designed to identify potential wake encounters. A low circulation threshold level permits identifying more potential wake encounters. To eliminate unnecessary false alarms, more studies on acceptable circulation thresholds need to be conducted.

This chapter introduces two approaches to assess proposed circulation thresholds to develop a guideline to be included in future EWEM modeling efforts.

Table 22 The ASR Assessment Guidelines for Wake Encounters.

Magnitude	Aircraft Reaction	Reaction Inside Aircraft
Light	Wake turbulence that momentarily causes slight erratic changes in altitude or attitude (pitch, yaw, or most usually, roll). May include rhythmic attitude changes.	Occupants may feel slight strain against seat belts or shoulder straps. Unsecured objects may be displaced slightly.
Moderate	Wake turbulence that is similar to Light Wake Turbulence, but of greater intensity. Larger changes in altitude and/or attitude occur, but the aircraft remains in positive control at all times.	Occupants feel definite strains against seat belts or shoulder straps. Unsecured objects are dislodged.
Severe	Wake turbulence that causes large, abrupt changes in altitude and/or attitude (most typically large uncommanded roll rates and resulting bank angles, often followed by a reverse direction roll). Aircraft may be momentarily out of control.	Occupants are forced violently against seat belts or shoulder straps. Unsecured objects are tossed about.
Extreme	Wake turbulence in which the aircraft experiences large altitude or attitude changes caused by vortices of larger strength than the aircraft's control authority and is practically impossible to control. It may cause structural damage.	

6.1 Circulation Threshold based on Current In-Trail Separation Rules

Today, no guidance has been developed by the FAA can be used to identify the wake's severity. The only criteria used in the EWEM model so far is based on circulation threshold. The circulation threshold level set in the model is based on general knowledge of previous case studies, where aircraft exposed to certain circulation levels could lose control (Treve 2012). However, we can compare the circulation thresholds set in the model with the separation standards published and accepted by the FAA. The ideal case is that the envelope truncated at the desired circulation threshold should be equal or less to the current in-trail separation standard.

The drawback of this approach is that we also need to consider the circulation strength of the following aircraft. If current separation standards are considered to be conservative, then there is

room for reductions, which implies that the circulation threshold we used on this approach might be over protecting the area from the trailing aircraft.

6.1.1 Terminal Separation (Arrival)

The FAA has published the new separation standard for ReCat in 2012, as mentioned in Chapter 2.1, and as shown in Table 4. For each aircraft type, we use the similar Monte Carlo simulation introduced in the EWEM methodology to create the relationship between circulation decay and the wake envelope's distance profiles to find the corresponding circulation strength for each separation distance set by the FAA. Envelopes were generated for each selected aircraft type based on BADA profiles at an altitude of 2000 feet, considered a typical altitude where aircraft follow each other in-trail. This is presented in Figure 65.

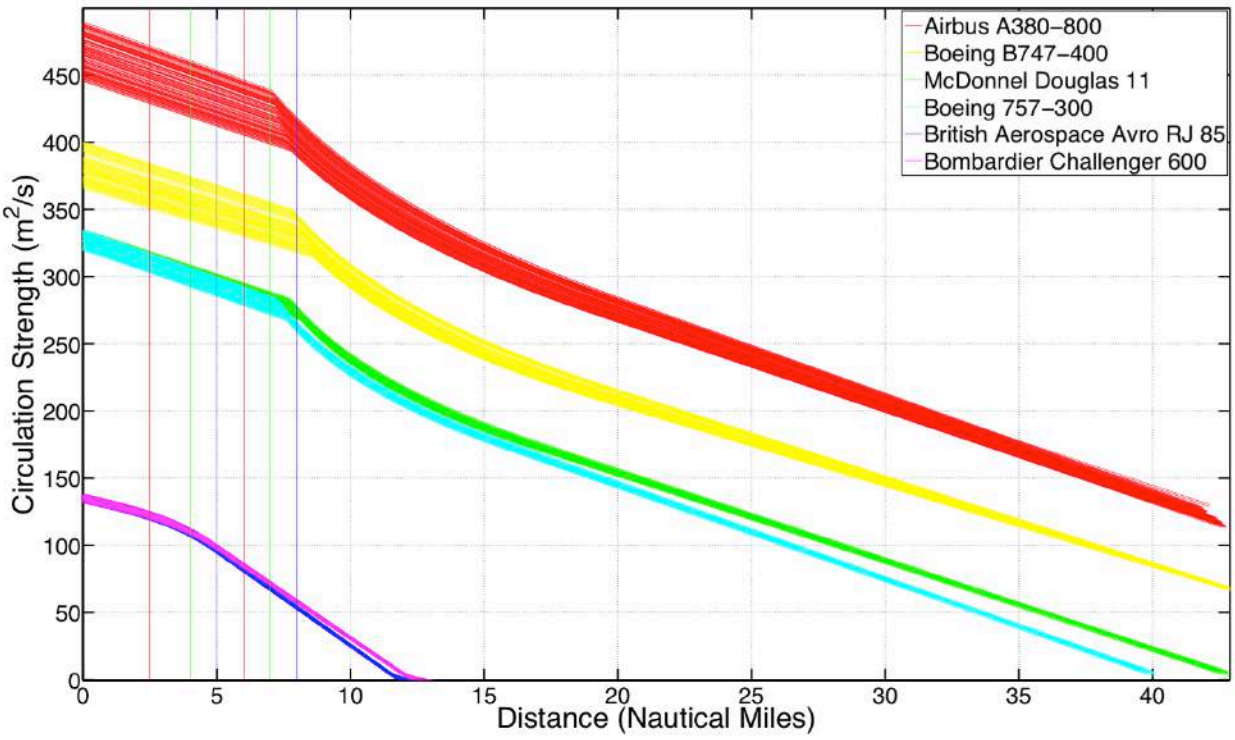


Figure 65 Wake Vortices Behavior for Six Aircraft Types Presenting Six ReCat Groups at the Altitude of 2000 feet.

Current wake separation standards in the terminal area are also plotted in Figure 64 as vertical lines. We can clearly see the corresponding circulation strength for each envelope length. For the

Minimum Radar Separation (MRS) at 2.5 nm, the wake strength for two aircraft types under ReCat groups E and F is 90, smaller than the threshold of 100 m^2/s we set for the EWEM. A complete separation matrix that compares the two values is shown in the Table 23.

Table 23 Circulation Strength for the FAA Separation Standards.

	A	B	C	D	E	F
	Circulation Strength (m^2/s)					
A	450	350	330	300	300	290
B	350*	340	310	275	275	240
C	300*	300*	300*	230	230	200
D	300*	300*	300*	300*	300*	200
E	90*	90*	90*	90*	90*	40
F	90*	90*	90*	90*	90*	90*
EWEM CT	**250**	**250**	**200**	**125**	**100**	**100**

*With the MRS Separation of 2.5 nm.

Circulation strength gathered from the plot is higher than circulation thresholds set in the EWEM, except for radar separation limits. Figure 66 shows the wake envelope length in the EWEM model when the default circulation thresholds are applied.

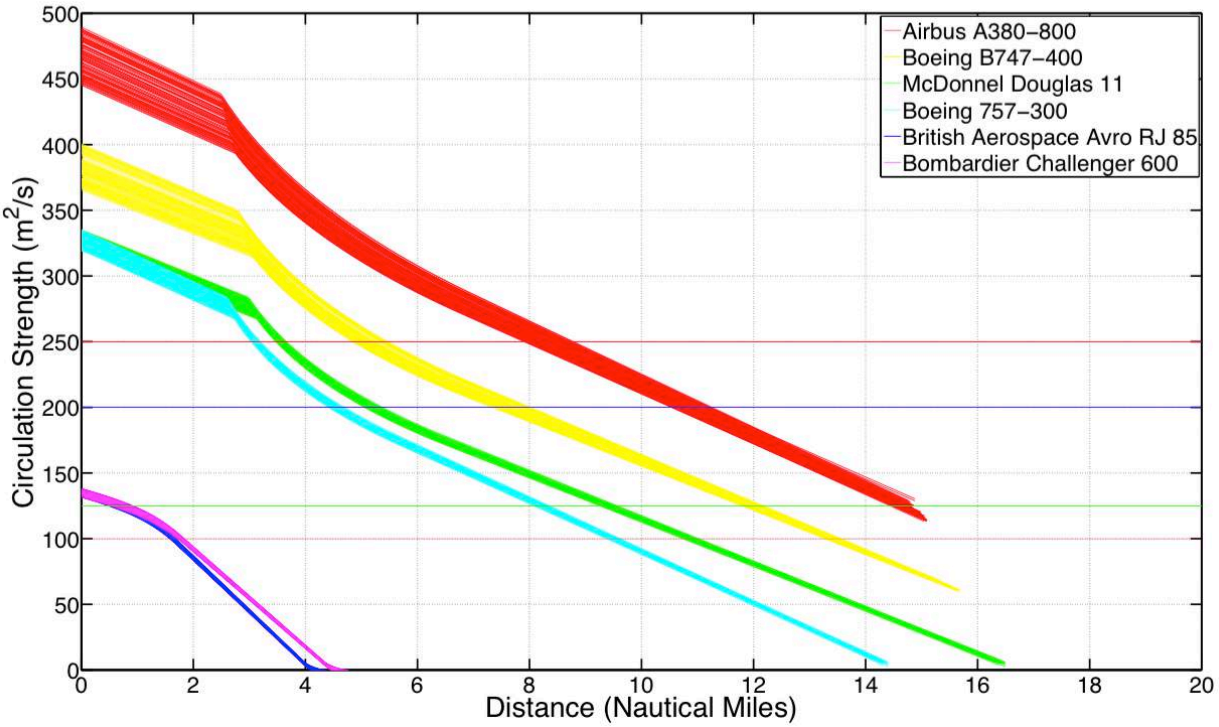


Figure 66 Wake Vortex Behavior for Six Aircraft Types at An Altitude of 2000 feet (Under Approach Conditions).

In general, the wake envelope circulation threshold levels in the EWEM model are longer than those derived using the FAA separation rules. The full results are summarized in Table 24.

Table 24 Wake Envelope Length using Default EWEM Circulation Threshold Levels.

	A	B	C	D	E	F
	Envelope Length (Separation)					
A	10*	10	13	17	18	18
B	6*	6	9	14	16	16
C	4*	4*	6*	11	13	13
D	3.8*	3.8*	5.8*	10*	11.5*	11.5
E	**	**	**	**	**	2
F	**	**	**	**	**	2*

*: MRS Separation 2.5 applied in the FAA standard
**: Not available. Circulation strength did not exceed the hazard boundary.

From Tables 23 and 24, comparing the FAA minimum separation matrix applied in the terminal area between each ReCat group pairs, the EWEM might be over protecting the following aircraft from minor “potential encounters”.

6.1.2 En-Route Cruise Level Separation Analysis

The current in-trail separation between flights in the en-route is 5 nm when aircraft are far from control centers at a distance greater than 40 miles of a radar facility. Air traffic controllers apply conservative buffers while separating traffic en-route. 7.5 nm to 10 nm are practical separation distances applied for in-trail separation at cruising altitude.

Referring to Equation (2), cruising aircraft are expected to generate stronger turbulence than aircraft in the landing phase due to the high mass and low air density.

The wake envelope lengths for each leader-follower pair are summarized in Table 25. Most of the wake envelope lengths calculated are longer than 10 nm using default EWEM circulation threshold levels. For Airbus A380-800, the wake envelope can be as long as 20 nm, albeit for a mild circulation level.

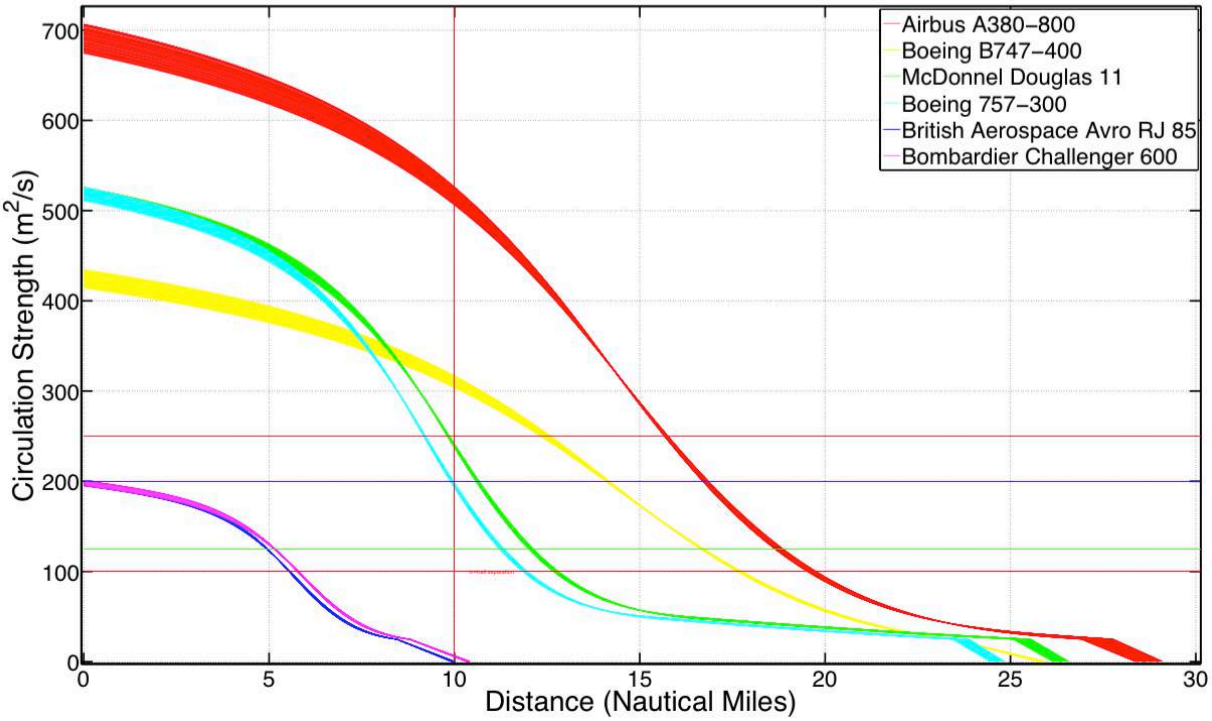


Figure 67 Wake Vortex Behavior for Six Aircraft Types at 29000 feet of Altitude.

Table 25 Wake Envelope Length using EWEM Circulation Threshold Levels at Cruise Altitude (29000 feet).

	A	B	C	D	E	F
	Envelope Length (Separation)					
A	16	16	17	18	20	20
B	12	12	14	16	17.5	17.5
C	10	10	11	12.5	13	13
D	9	9	10	11.5	12	12
E	**	**	**	5	6	6
F	**	**	**	5	6	6

6.2 Rolling Moment Coefficient

Another approach to understand the effect of circulation threshold levels using the rolling moment coefficient (RMC) standard published by the EUROCONTROL (Hoogstraten 2013). EUROCONTROL is using RMC to quantify and assess the severity of a potential wake encounter. The rolling moment coefficient is defined based on both the lead aircraft performance and the trailing aircraft type or category, as shown in Equation (16) (Treve 2012).

$$\textit{Rolling Moment Coefficient } Cl \propto \frac{\Gamma_i}{V_{fj} b_{fj}} \tag{16}$$

Where, Γ_i : Circulation Strength

V_{fj} : Aircraft Speed (follower)

b_{fj}: Aircraft Wingspan (follower)

Table 26 lists values of RMC thresholds proposed by the ReCat-EU that could be used to assess the severity of the wake turbulence encounters. The hazardous threshold is set to be 0.03 from historical data. Wake turbulence encounters with a value of RMC above 0.03 are more likely to be reported.

Table 26 Wake Encounter Severity and RMC Values (Hoogstraten 2013).

RMC Range	Wake Encounter Severity
RMC > 0.07	Severe
0.03 < RMC <0.07	Hazardous
RMC < 0.03	Harmless

RMC values were calculated for all EWEM reported encounters based on the aircraft performance at the time of the encounter. Circulation threshold levels can be estimated based on RMC standards. Terminal area analysis conducted in the EWEM model using BADA landing profile speeds at 2000 feet, are presented in Table 27. The values of RMC are also shown in the table. Table 28 summaries the circulation strength corresponding to different RMC boundaries.

Table 27 RMC Values of Circulation Threshold Levels Used in EWEM Model.

	Wake Encountering Aircraft ReCat Group (Follower)					
	A	**B**	**C**	**D**	**E**	**F**
Circulation Threshold (m^2/s)	250.00	250.00	200.00	125.00	100.00	100.00
Speed (m/s)	104.85	107.43	113.59	101.26	90.46	97.15
Wingspan (m)	80.00	64.00	52.00	38.00	26.00	20.00
RMC Value	**0.03**	**0.04**	**0.03**	**0.03**	**0.04**	**0.05**

The RMC level for EWEM circulation threshold levels are close to the hazardous boundary defined by EUROCONTROL. For ReCat group F, the RMC value is slightly higher than 0.03, which might suggests a low report rate of “potential wake encounters” by the EWEM model in th ReCat group F.

Table 28 Circulation Threshold Calculated with Two RMC Boundaries using BADA Landing Profiles at 2000 feet.

	Wake Encountering Aircraft (Follower)					
	A		**B**		**C**	
RMC Value	0.03	0.07	0.03	0.07	0.03	0.07
Speed (m/s)	104.85	104.85	107.43	107.43	113.59	113.59
Wingspan (m)	80.00	80.00	64.00	64.00	52.00	52.00
Circulation Threshold (m^2/s)	**251.64**	**587.16**	**206.27**	**481.29**	**177.20**	**413.47**
	D		**E**		**F**	
RMC Value	0.03	0.07	0.03	0.07	0.03	0.07
Speed (m/s)	101.26	101.26	90.46	90.46	97.15	97.15
Wingspan (m)	38.00	38.00	26.00	26.00	20.00	20.00
Circulation Threshold (m^2/s)	**115.44**	**269.35**	**70.56**	**164.64**	**58.29**	**136.01**

It can be observed in the Table 28 that circulation threshold levels used in the EWEM model are close to the circulation strength calculated with an RMC value of 0.03. Circulation thresholds with severe RMC values are much higher than those values used as EWEM defaults.

For flights in the cruise phase, the high values of airspeed tend to reduce turbulence and thus decrease the value of the rolling moment coefficient (RMC). The circulation strength calculated for cruise profiles is much higher compared to the values estimated in the terminal area, as summarized in Table 29.

Table 29 Circulation Threshold Calculated with Two RMC Boundaries with BADA Descent Profile at 29000 feet.

	Wake Encountering Aircraft (Follower)					
	A		**B**		**C**	
RMC Value	0.03	0.07	0.03	0.07	0.03	0.07
Speed (m/s)	235.93	235.93	243.12	243.12	243.12	243.12
Wingspan (m)	80.00	80.00	64.00	64.00	52.00	52.00
Circulation Threshold (m^2/s)	**566.22**	**1321.19**	**466.79**	**1089.19**	**379.27**	**884.96**
	D		**E**		**F**	
RMC Value	0.03	0.07	0.03	0.07	0.03	0.07
Speed (m/s)	235.93	235.93	212.80	212.80	235.93	235.93
Wingspan (m)	38.00	38.00	26.00	26.00	20.00	20.00
Circulation Threshold (m^2/s)	**268.96**	**627.56**	**165.98**	**387.29**	**141.56**	**330.30**

7 Model Users' Manual

The EWEM model has been integrated with the WakeWise Model developed by the George Mason University into a stand-alone application. Both models use the APA Suite program developed by NASA. The goal of both models is to produce an estimation of potential wake encounters for a given input flight data set. This manual will introduce the essential files and information needed to run EWEM.

7.1 Inputs

The model relies on several files with specific names stored in the installation directory. This section will explain the essential files needed to run the model, including installation files, and those that can be edited.

7.1.1 Flight Tracks

EWEM can process four different types of inputs: 1) Performance Data Analysis Reporting System (PDARS) data, 2) Airspace Concept Evaluation System (ACES) data, 3) Terminal Area Route Generation Evaluation and Traffic Simulation (TARGETS) data, and 4) FlightAware data. The installation package provides a small sample data for each one of the four supported input types.

If the user would like to use an additional flight track data type not supported in the model, a change in the data format is needed. Users can also refer to the sample files provided in the installation package to understand all four formats.

7.1.1.1 Performance Data Analysis Reporting System (PDARS) data

PDARS data can be imported into the model without any preprocessing. The raw data should be stored in the .csv format as shown in Figure 68.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK
1	1	BFF	2.6																																		
2	5	ATAC Corporati	BirdWatc	4.5.4																																	
3	0	==																																			
4	2	1219626306	7200	2013	628	0/ATL		DAL981	1	B752	ATL	BRA	D	ATL	?																						
5	4	1219626306	7200	2013	628	0/ATL	1353	DAL981	1	B752	ATL	BRA	N	-99	-99	370		?	-99	I	J	D	?	-99	?	5											
6	4	1219639343	7200	2013	628	0/ATL	1353	DAL981	1	B752	ATL	BRA	N	-99	-99	370		?	-99	I	J	D	?	-99	?	5		T19									
7	3	1219639212	7200	2013	628	0/ATL	1350	DAL981	1	33.64675	-84.4085	14.28	1	0.004	0.004	-99	163	82	2561	?	?	?	-99		-99						0	79	0	?	2	A	ATL
8	3	1219639217	7200	2013	628	0/ATL	1350	DAL981	1	33.64679	-84.4043	16.28	5	0.004	0.004	-99	163	82	2561	?	?	?	-99		-99						0	79	0	?	2	A	ATL
9	3	1219639226	7200	2013	628	0/ATL	1350	DAL981	1	33.64728	-84.3955	20.28	1	0.006	0.006	-99	171	79	2574	?	?	?	-99		-99						79	81	1	N	130	D	ATL
10	3	1219639231	7200	2013	628	0/ATL	1350	DAL981	1	33.64780	-84.3908	22.28	6	0.006	0.006	-99	174	75	2582	?	?	?	-99		-99						79	81	1	N	130	D	ATL

Figure 68 Sample raw PDARS data from Atlanta TRACON.

7.1.1.2 Airspace Concept Evaluation System (ACES) data

Direct output from an ACES simulation can be imported into the model with sorted times. The raw data should be stored in the .csv format as shown in Figure 69.

	A	B	C	D	E	F	G	H	I	J	K
1	simulationTime	flightId	latitude	longitude	altitude	airspeed	heading	groundspeed	groundTrackAngle	facilityName	aircraftType
2	1.35701E+12	50656	30.31516199	-95.2811	10068.443	214.18027	327.5433	248.1342	-32.45669	ZHU	B737
3	1.35701E+12	50656	30.32008996	-95.2848	10180.893	221.5942	327.5451	257.05356	-32.4549	ZHU	B737
4	1.35701E+12	50656	30.32518938	-95.2886	10297.429	228.81815	327.5475	265.791	-32.45252	ZHU	B737
5	1.35701E+12	50656	30.33045818	-95.2925	10414.854	235.95009	327.55014	274.4444	-32.44987	ZHU	B737
6	1.35701E+12	50656	30.33589458	-95.2965	10532.885	242.97762	327.55295	282.99768	-32.44704	ZHU	B737

Figure 69 Sample Raw ACES Data.

Note: Conservative wake zone mode analysis, as explained in Chapter 4.4.4, is suggested for the ACES data.

7.1.1.3 The Terminal Area Route Generation Evaluation and Traffic Simulation (TARGETS) data

Approach and departure procedures designed by TARGETS can be imported into the EWEM model with minor changes. TARGETS data should contain two separate files: a) one containing detailed track data and b) another one containing flight information. The track file name should ends with "_Tracks", and the flight information file name should ends with the string "_Flights". Both files need to be stored in .csv format. For example, the output file for parallel runway procedures could be named as ParallelOperations_Flights.csv and ParallelOperations_Track.csv.

	A	B	C	D	E
1	10001	GENE_181001	DC10	?	?
2	10002	BBKING_181001	B727	?	?
3	10003	OES_181001	B727	?	?
4	10004	SEP_181001	A306	?	?

Figure 70 Sample TARGETS Flight Information File.

	A	B	C	D	E	F	G
1	10001	0:00:00.0	35.04882	-89.973	2.77	0	179
2	10001	0:00:04.7	35.04856	-89.973	2.77	23	179
3	10001	0:00:09.4	35.04779	-89.9729	2.79	47	179
4	10001	0:00:14.1	35.0465	-89.9729	2.82	70	179
5	10001	0:00:18.8	35.0447	-89.9729	2.86	94	179

Figure 71 Sample TARGETS Track File.

7.1.1.4 *FlightAware data*

Track logs downloaded from FlightAware.com can be imported to the model without modification. Similar to the TARGETS data, the FlightAware data should also contain two separate files. One with detailed track data and the second one with flight information. The files should also be named with string ends "_Flights" and "_Tracks" at the end for differentiation, and stored in the.csv format.

	A	B	C	D	E
1	10001	GENE_181001	DC10	?	?
2	10002	BBKING_181001	B727	?	?
3	10003	OES_181001	B727	?	?
4	10004	SEP_181001	A306	?	?
5	10005	AZONE_181001	A306	?	?
6	10006	CRSON_181001	DC10	?	?
7	10007	CASN_18C1001	DC10	?	?
8	10008	JTEE_181001	DC10	?	?

Figure 72 Sample FlightAware Flight Information File.

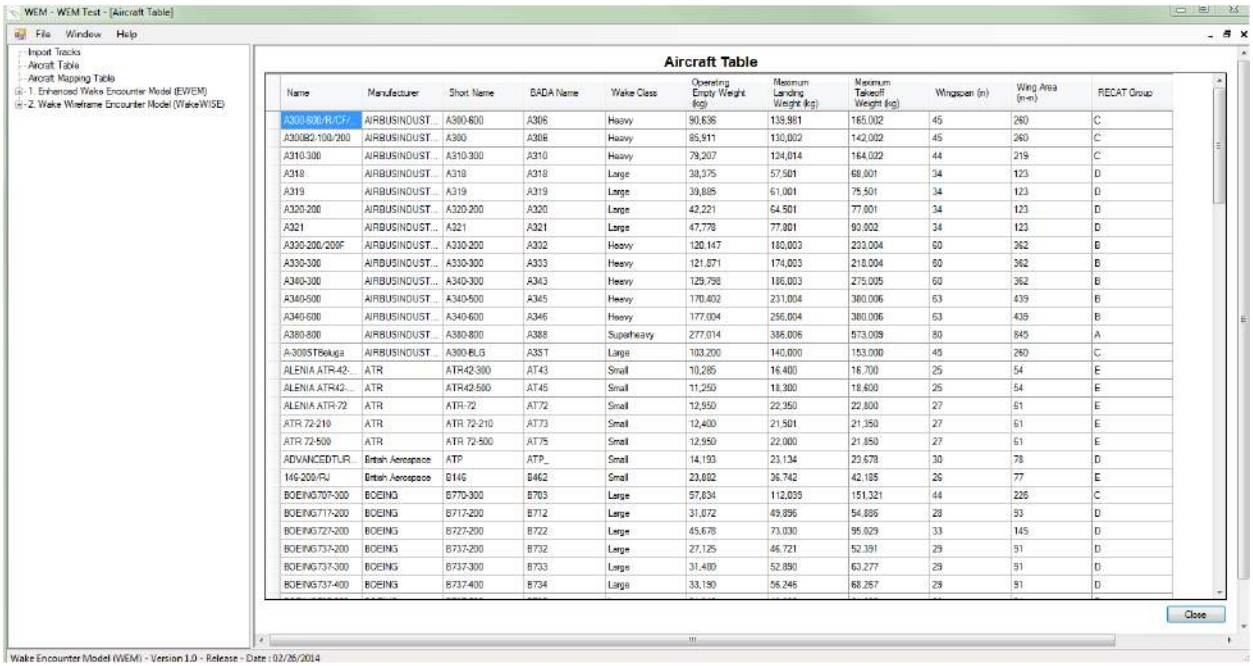
	A	B	C	D	E	F	G	H	I	J
1	Flight ID	Time (UTC)	Longitude	Latitude	Groundspeed (knots)	Altitude (feet)	Type	Facility Name	Facility Type	
2	1	5/14/2013 4:02	-118.4179993	33.93600082	161	800	A	FlightAware		
3	1	5/14/2013 4:02	-118.4430008	33.93299866	176	1600	A	FlightAware		
4	1	5/14/2013 4:03	-118.461998	33.92399979	204	1900	A	FlightAware		
5	1	5/14/2013 4:03	-118.487999	33.90800095	241	2300	A	FlightAware		
6	1	5/14/2013 4:04	-118.5189972	33.88600159	253	3400	A	FlightAware		
7	1	5/14/2013 4:04	-118.5439987	33.86800003	261	4200	A	FlightAware		
8	1	5/14/2013 4:05	-118.5559998	33.83200073	277	5400	A	FlightAware		

Figure 73 Sample FlightAware Track File.

7.1.2 Aircraft Table

EWEM can recognize 295 aircraft types, 116 out of which are from the Base of Aircraft Data (BADA) version 3.9 published in 2012. All 116 aircraft types have information including operating empty weight, maximum landing weight, maximum takeoff weight, wingspan, wing

area, wake class, and the ReCat group. The table can be accessed from the main panel in the EWEM model, as shown in Figure 74.



Name	Manufacturer	Short Name	BADA Name	Wake Class	Operating Empty Weight (kg)	Maximum Landing Weight (kg)	Maximum Takeoff Weight (kg)	Wingspan (m)	Wing Area (m×m)	RECAT Group
A300-600/R/CF/...	AIRBUSINDUST...	A300-600	A306	Heavy	90,636	139,981	165,002	45	260	C
A30082-100/200	AIRBUSINDUST...	A300	A306	Heavy	85,911	130,002	142,002	45	260	C
A310-300	AIRBUSINDUST...	A310-300	A310	Heavy	79,207	124,014	164,022	44	219	C
A318	AIRBUSINDUST...	A318	A318	Large	38,375	57,501	68,001	34	123	D
A319	AIRBUSINDUST...	A319	A319	Large	39,885	61,001	75,501	34	123	D
A320-200	AIRBUSINDUST...	A320-200	A320	Large	42,221	64,501	77,001	34	123	D
A321	AIRBUSINDUST...	A321	A321	Large	47,778	77,801	93,002	34	123	D
A330-200/200F	AIRBUSINDUST...	A330-200	A332	Heavy	120,147	180,003	233,004	60	362	B
A330-300	AIRBUSINDUST...	A330-300	A333	Heavy	121,871	174,003	218,004	60	362	B
A340-300	AIRBUSINDUST...	A340-300	A343	Heavy	129,798	186,003	275,005	60	362	B
A340-500	AIRBUSINDUST...	A340-500	A345	Heavy	170,402	231,004	380,006	63	439	B
A340-600	AIRBUSINDUST...	A340-600	A346	Heavy	177,004	256,004	380,006	63	439	B
A380-800	AIRBUSINDUST...	A380-800	A388	Superheavy	277,014	386,006	573,009	80	845	A
A-300ST Beluga	AIRBUSINDUST...	A300-BLG	A3ST	Large	103,200	140,000	153,000	45	260	C
ALENIA ATR-42-...	ATR	ATR42-300	AT43	Small	10,285	16,400	16,700	25	54	E
ALENIA ATR42-...	ATR	ATR42-500	AT45	Small	11,250	18,300	18,600	25	54	E
ALENIA ATR-72	ATR	ATR-72	AT72	Small	12,950	22,350	22,800	27	61	E
ATR 72-210	ATR	ATR 72-210	AT73	Small	12,400	21,501	21,350	27	61	E
ATR 72-500	ATR	ATR 72-500	AT75	Small	12,950	22,000	21,850	27	61	E
ADVANCEDTUR...	British Aerospace	ATP	ATP_	Small	14,193	23,134	23,678	30	78	D
146-200/RJ	British Aerospace	B146	B462	Small	23,802	36,742	42,185	26	77	E
BOEING707-300	BOEING	B770-300	B703	Large	57,834	112,039	151,321	44	226	C
BOEING717-200	BOEING	B717-200	B712	Large	31,072	49,896	54,886	28	93	D
BOEING727-200	BOEING	B727-200	B722	Large	45,678	73,030	95,029	33	145	D
BOEING737-200	BOEING	B737-200	B732	Large	27,125	46,721	52,391	29	91	D
BOEING737-300	BOEING	B737-300	B733	Large	31,480	52,890	63,277	29	91	D
BOEING737-400	BOEING	B737-400	B734	Large	33,190	56,246	68,267	29	91	D

Figure 74 The Aircraft Table.

To reduce the possibility of error, this table is fixed and not editable in the model. An aircraftmapping table is designed for users to add aircraft types that are not listed in the default dataset. The aircraft mapping table can be accessed from the main panel.

For editing, click on the cell that needs to be modified and change the desired value. To add an aircraft, scroll down to the end of the table and type in the new aircraft information. Make sure to fill in the “Aircraft Code” cell with the new aircraft type, and the “BADA File” cell with one of the existing BADA aircraft types (Please verify the “Aircraft Table” for correctness).

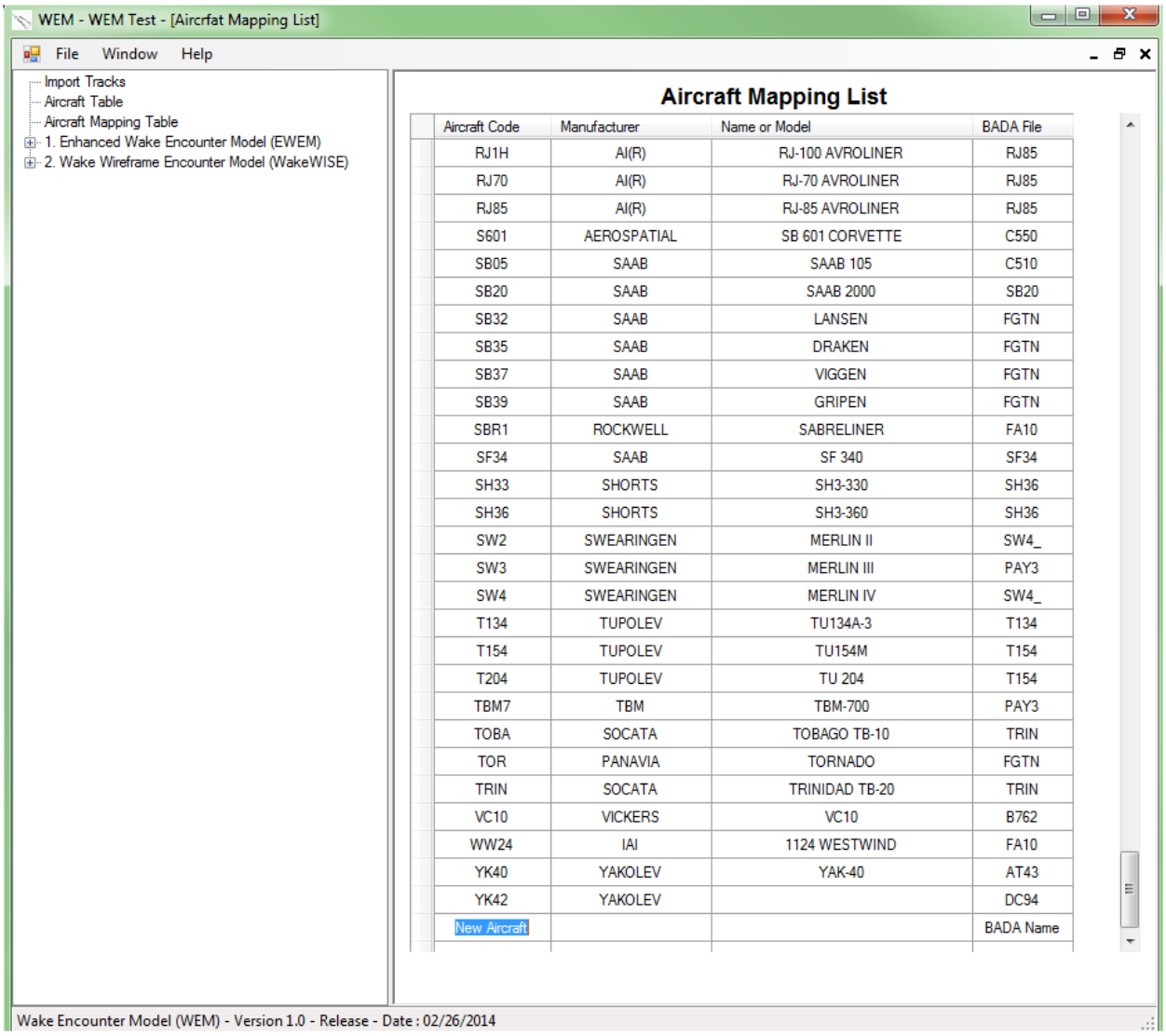


Figure 75 Aircraft Mapping Table.

Note: If the user is would like to make changes of the basic aircraft table, the raw file is called "Aircraft_Info_Common_metric_units.csv" under \WEM_GUI\DATA\EWEM. The modification will be automatically stored for future runs.

7.1.3 Wind Source

EWEM provides three different types of wind analyze 1) fixed wind input, 2) wind distribution and 3) ASOS data.

7.1.3.1 Fixed Input

This requires only wind strength and wind direction as the inputs. This is most suitable for historical data with known wind condition. The wake envelope will be transported laterally based on the given wind profile.

7.1.3.2 Wind Distribution

This requires mean wind strength and the wind strength standard deviation. This analysis is suggested for simulation data or unknown weather conditions. The envelope will be expanded laterally on both sides for the given distribution by 2 sigma. In other words, this analysis will provide a more conservative wake zone for a certain range of wind variation. See Chapter 3.4 for a more detailed explanation.

7.1.3.3 Automated Surface Observing System data

Automated Surface Observing System (ASOS) is a surface weather observing system that contains nationwide historic wind data. This analysis requires corresponding ASOS data for selected dates of wind condition at the TRACON. This is suggested for terminal area analysis. The model takes ASOS 1 minute data for analysis.

The ASOS data can be retrieved from the National Oceanic and Atmospheric Administration website via FTP (http://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/automated-surface-observing-system-asos). Please refer to the NOAA website for downloading instructions.

```
94789KJFK JFK20140101025207 52   0.060 N     0.085 N     0.191 N     280   10   273   12    04R60+
94789KJFK JFK20140101025307 53   0.062 N     0.084 N     0.191 N     279   11   260   16    04R60+
94789KJFK JFK20140101025407 54   0.060 N     0.084 N     0.191 N     281   13   283   16    04R60+
94789KJFK JFK20140101025507 55   0.062 N     0.086 N     0.189 N     284   11   290   12    04R60+
94789KJFK JFK20140101025607 56   0.062 N     0.085 N     0.188 N     287   10   285   12    04R60+
94789KJFK JFK20140101025707 57   0.061 N     0.086 N     0.188 N     286   11   293   14    04R60+
94789KJFK JFK20140101025807 58   0.061 N     0.085 N     0.188 N     288   13   289   17    04R60+
94789KJFK JFK20140101025907 59   0.061 N     0.086 N     0.190 N     290   15   290   17    04R60+
94789KJFK JFK20140101030008 00   0.062 N     0.087 N     0.190 N     286   13   274   13    04R60+
```

Figure 76 Sample ASOS Data.

Using wrong wind data for the analysis might crash the model or result in unexpected behavior.

Note: For simulation of tracks, the model will use the closest wind data for analysis if the wind file is selected.

7.1.4 Circulation Threshold

Circulation threshold is defined as value of circulation considered harmful for a given aircraft group. See Chapter 3.3.5 for a more detailed explanation.

The suggested values of circulation threshold are given as defaults in the model. However, these values can be modified by the user. Before running the model, the user has a chance to modify the circulation threshold in the “Run New Case...” window. Click on the number that is willing to change, and enter the desired value from 100 m^2/s to 250 m^2/s, with the step of 25 m^2/s.

Numbers beyond the range will result in errors.

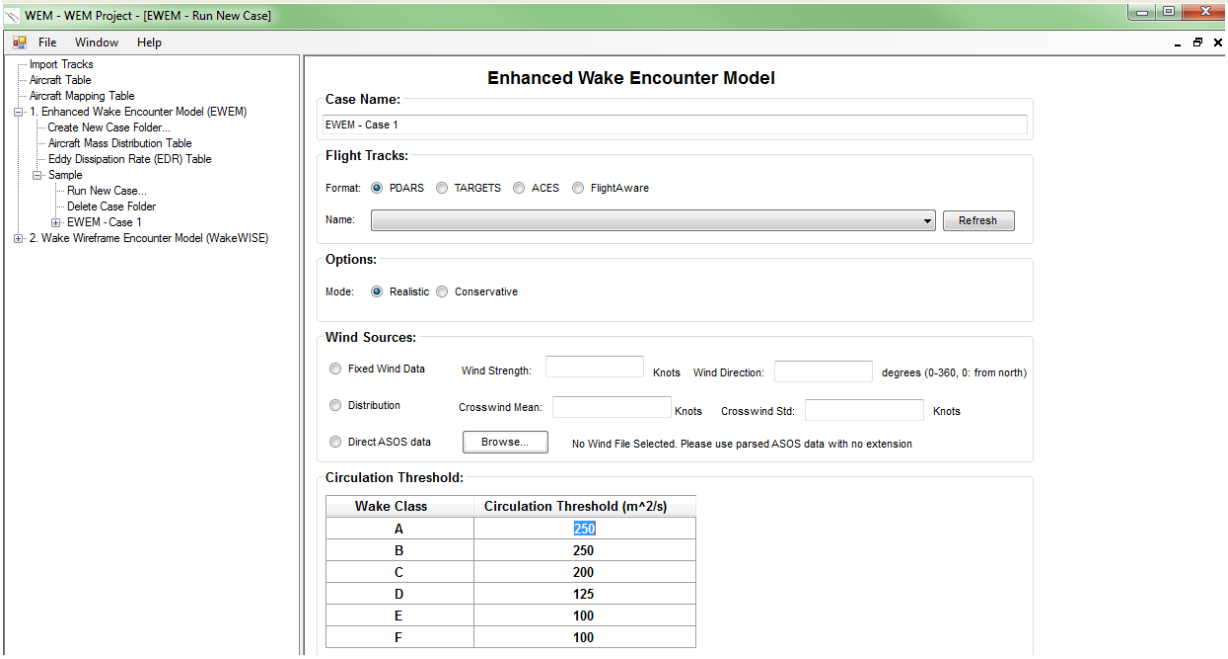


Figure 77 Run New Case Window – Circulation Threshold Setting.

7.1.5 Aircraft Mass Distribution

EWEM uses stochastic mass distribution to estimate the wake zone. The aircraft mass will be computed based on the mean and standard deviation provided in the Aircraft Mass Distribution Table, which can be accessed from the main panel anytime.

The model includes weight distribution defaults for the aircraft mass under different flight conditions. Users can modify the table by changing the cell values.

Changing this table may result in very unrealistic wake behavior

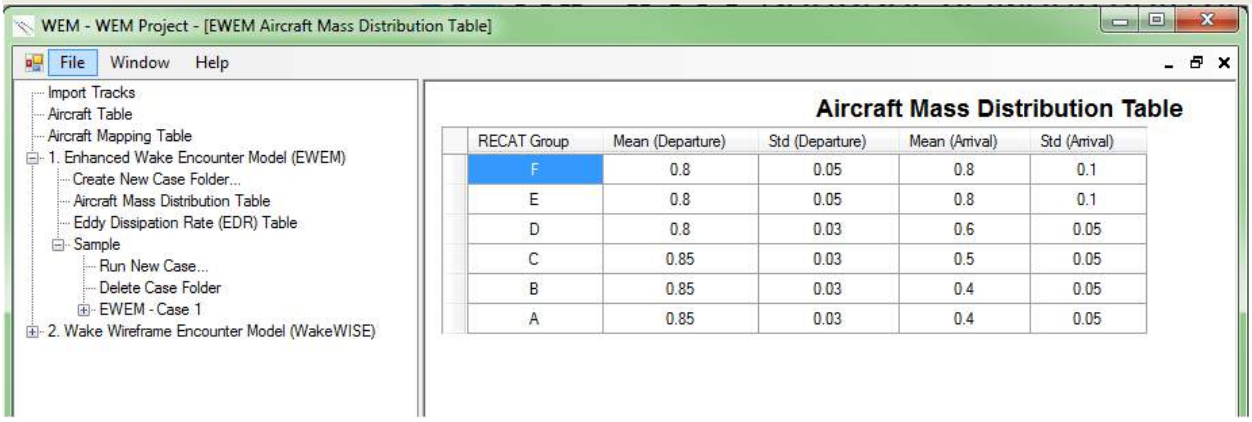


Figure 78 Aircraft Mass Distribution Table.

7.1.6 Eddy Dissipation Rate

EDR value (m^2/s^3) is used for the atmospheric turbulence measurement. A suggested list of EDR values is given in the model and can be modified before running a new case.

Users can access the table from the main panel and modify the values by directly changing the values in each cell. For altitudes above the maximum entry, the EDR value will be automatically set to the last value available in the table.

7.2 Quick Start Guide

This part will help you to create and run a simple case with either model selected. For more detailed information on the model, please refer to the Chapter 7.1.

This model is only available in the Windows 64 bit OS. It is not supported in Unix, Linux, or Mac OS.

7.2.1 Before Running the Model

Make sure the computer has the following programs installed.

- *The latest version of Google Earth*
- *MATLAB R2012b 64bit or MATLAB Runtimes 7.18*

EWEM model was developed in the MATLAB environment, requires a MATLAB runtime to execute its files. The MATLAB Runtime can be downloaded and installed via the MathWorks website at: http://www.mathworks.com/products/compiler/mcr/index.html?s_cid=BB.

7.2.2 Running the Model

First, copy the entire WEM_GUI folder to your desired installation path. For example, E:\

Second, double-click the executable file (WEM_GUI.exe) located in

E:\WEM_GUI\WEM_GUI\bin\

7.2.3 Start a new project

If you open the model for the first time, you have to start by creating a new project.

1) Go to menu File->New Project. For example, WEM Project
2) Enter the New Project Name and Browse to the desired directory to store the project related files, including inputs and outputs. The directory is set to C:\EWEM_Project\ in the example.

3) Click “Create Project”

Note: The project main run file will be created in this assigned directory, such as C:\EWEM_Project. The default file type is .prj, you will need this later to open an existing project.

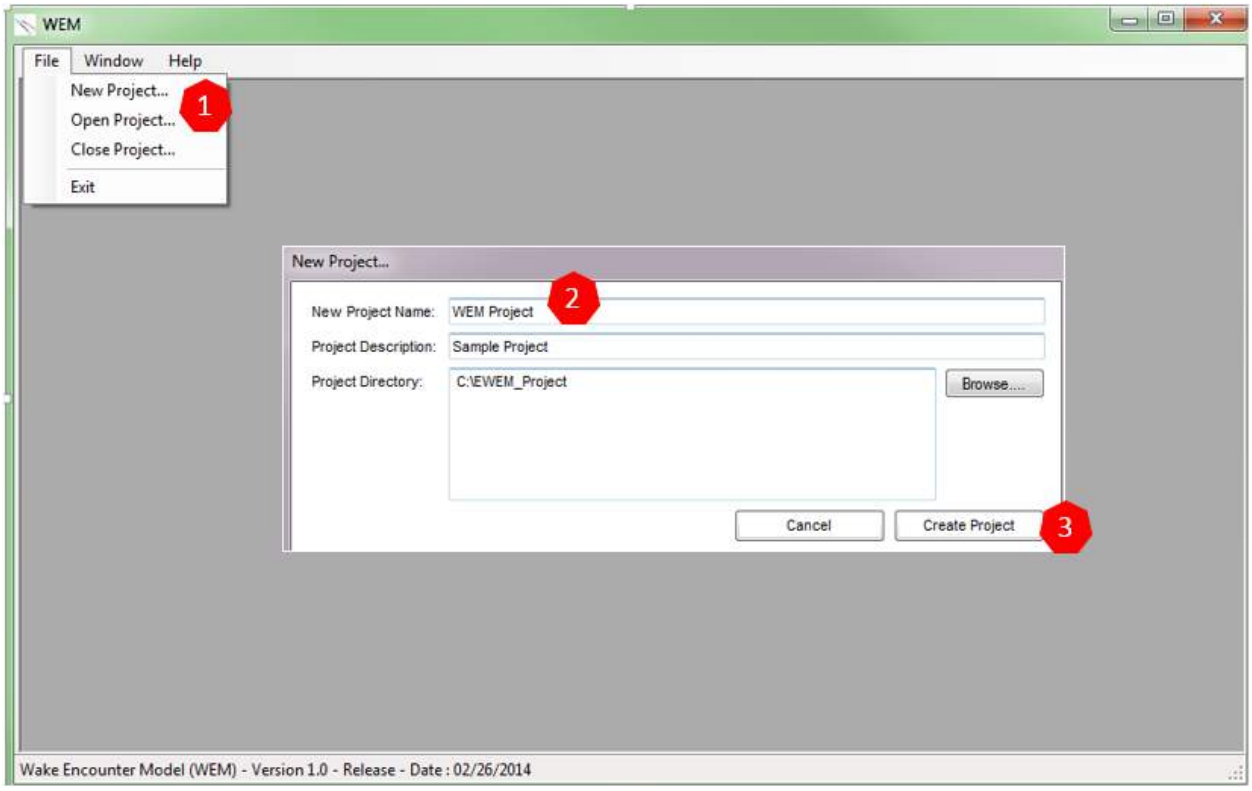


Figure 79 Create New Project Window.

7.2.4 Load an existing Project

1) Go to menu File->Open Project.

2) Choose the .prj file named after your project name from the “Browse Window”.

7.2.5 Main Screen

After creating a new project (or loaded an existing project), the software will show a welcome screen as shown in the Figure 80. The left side is the main panel includes all the links and functions to run a new case, delete an existing case, and modify related case settings.

The right side display window shows all functions available in the main panel.

The project name is always printed in the header of the main window.

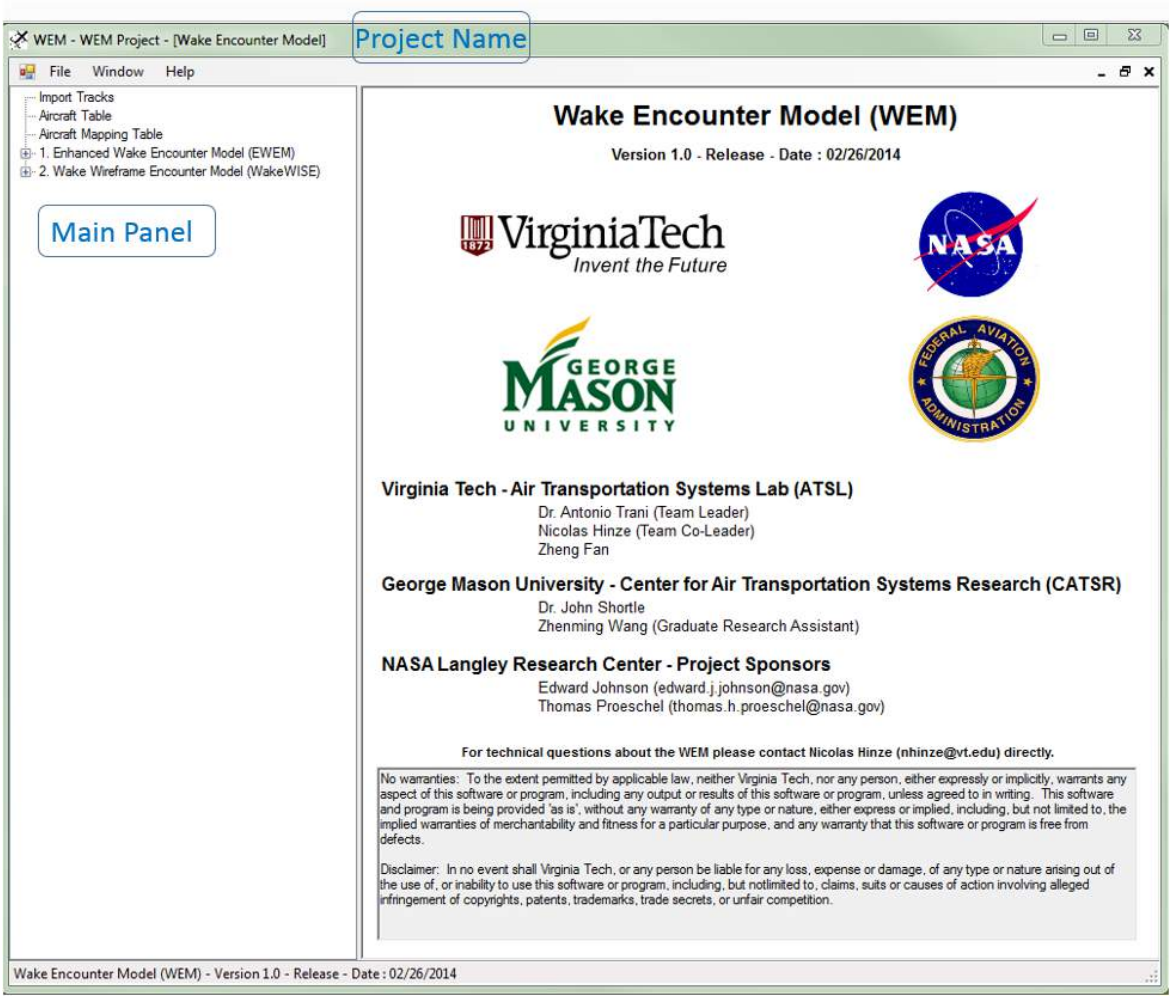


Figure 80 Model Main Screen Window.

7.2.6 Importing Tracks

Before you can run a new case, the most critical step is to import your track files.

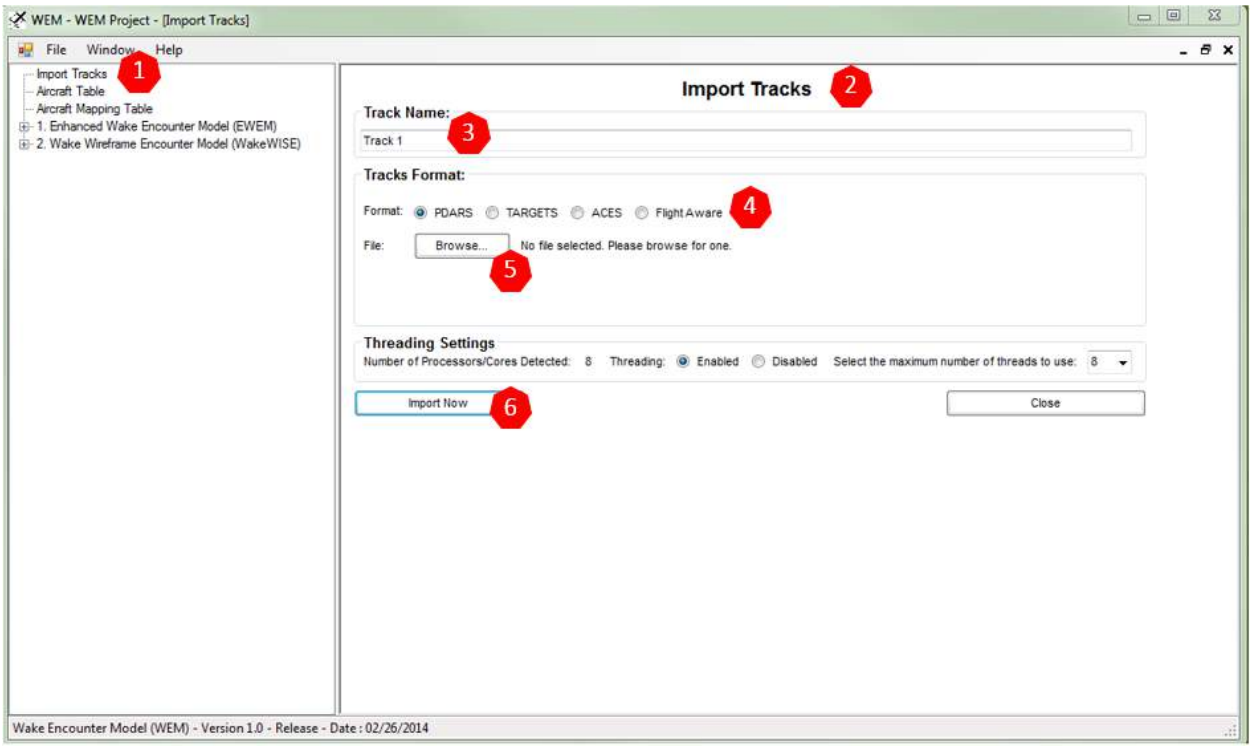


Figure 81 Import Tracks Window.

1) Click on “Import Tracks” in the main panel
2) “Import Track” screen will show on the right side
3) Enter the track name, the default is Track 1. This can be set to any wanted names
4) Select the correct format of your file

   The model currently can analyze four types of flight tracks:

   a) PDARS raw data is stored in .csv format
   b) TARGETS outputs are stored in .csv format, including both flight information and flight track data.

   The filename should be xxx_xxxx_Flights.csv and xxx_xxxx_Tracks.csv

   c) ACES outputs are stored in .csv format
   d) FlightAware data is stored in .csv format, including both flight information and flight track data.

The filename should be xxx_xxxx_Flights.csv and xxx_xxxx_Tracks.csv

5) Browse to your track file, and click “Open”

> *Make sure browse to xxx_xxxx_Flights.csv for TARGETS and FlightAware data*

6) Click “Import Now”

7) Please Wait

The progress window will pop up and update of the importing process.

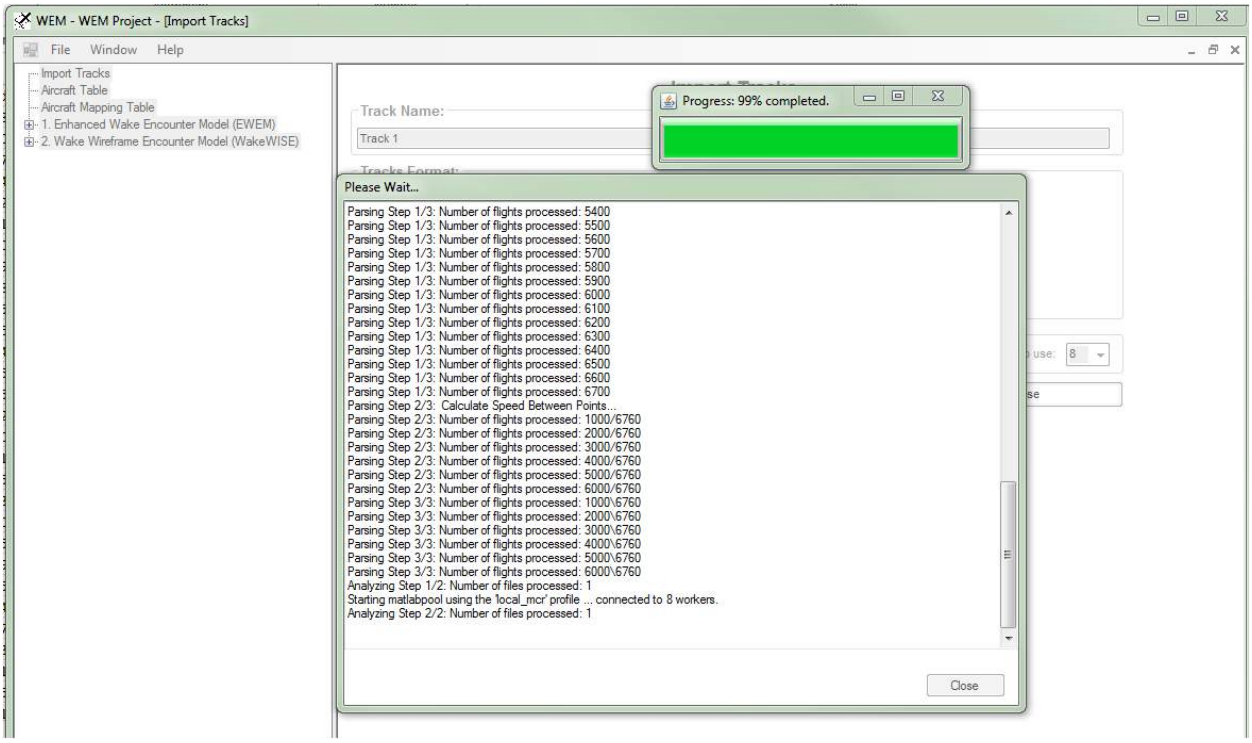


Figure 82 Progress Window.

Tip: All the imported tracks can be found under

InstallationDirectory\EWEM\Input\Tracks\Track Name\, including five files:

- Filtered_Pair_ Track Name,
- Parsed_ Track Name,
- Track Name,

- Track Name _FlightList.txt
- Track Name _Errors.txt

(For example the tracks imported in the sample will be stored in C:\EWEM_Project\EWEM\Input\Tracks\Track 1\)

Tip: Please refer to the Chapter 7.1.1 Flight Tracks for more details on all the supported flight track data formats. The installation kit contains a set of samples for each type of input data. Make sure that your file format looks exactly like those sample files.

7.2.7 Create New Case

Once you have the track imported, you can run potential wake encounter models.

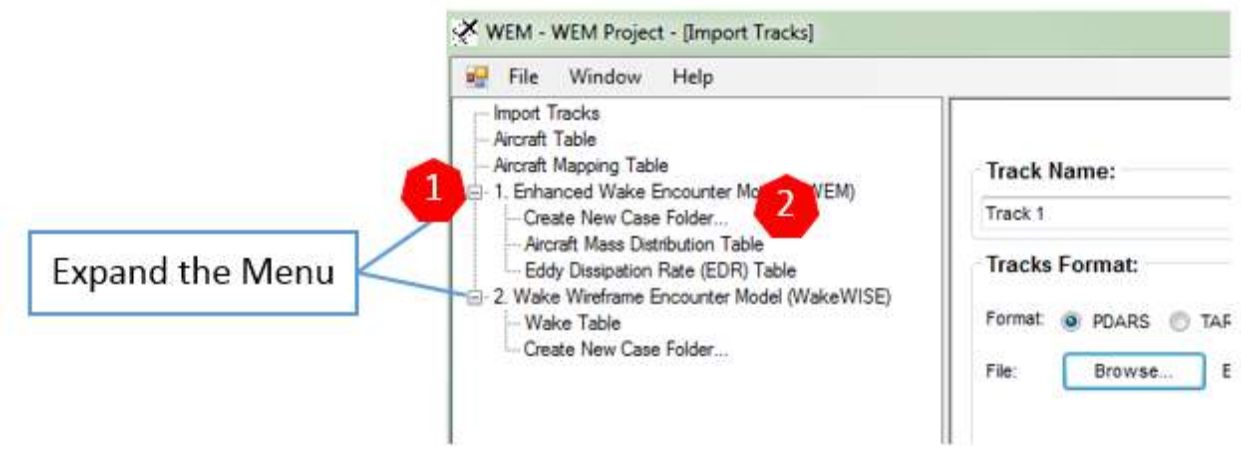


Figure 83 Create New Case Window.

1) Click on the "+" button by the selected model from the main panel to expand the function list
2) Click "Create New Case Folder" to create a new case

   Tip: Tracks are imported for each project generated. Tracks imported are sharable between cases, not projects. It is suggested to separate different flight tracks into different projects. In each project, create separate cases to model different scenarios (case settings).

3) Enter your case name, and click “Create Folder”

   This will create a separate case folder under your project’s output folder. For example, the sample case folder will be created in C:\EWEM_Project\EWEM\Output\Sample.

Figure 84 Create a New Case Folder Window.

7.2.8 Run New Case - EWEM

1) Click on the “+” button by the new created case name from the main panel
2) Click on “Run New Case…”
3) Enter the Case Name (Default is Case 1)
4) Select the correct data source of your parsed data, consistent with the track in step 5)
5) Click on the drop down bar and select the available imported track file.

   The drop down menu only shows tracks that have been imported in this project.

   Tip: If you want to use other tracks that have been previously imported to other projects, you can go to the track directory of that project, and copy the entire folder to your new project track folder.

   For example, If you have PDARS data parsed and named as Track_PDARS in Project 1, and I would like to use it again in Project 2. Go to C:\Project 1\EWEM\Input\Tracks\, copy the entire Track_PDARS folder, and paste it to C:\Project 2\EWEM\Input\Tracks\.
6) Select the desired model mode (Default is the realistic wake envelope analysis)

   Note: A simple illustration will be given based on the mode selected
7) Select wind source and input the right value, or use default (no selection) for zero wind condition

8) Change the circulation threshold or use default

9) Click “Run Now”

10) Make sure everything is correct in the pop-up Confirmation Window

11) Click “Yes” to run the model

Click “No” to reset the case settings

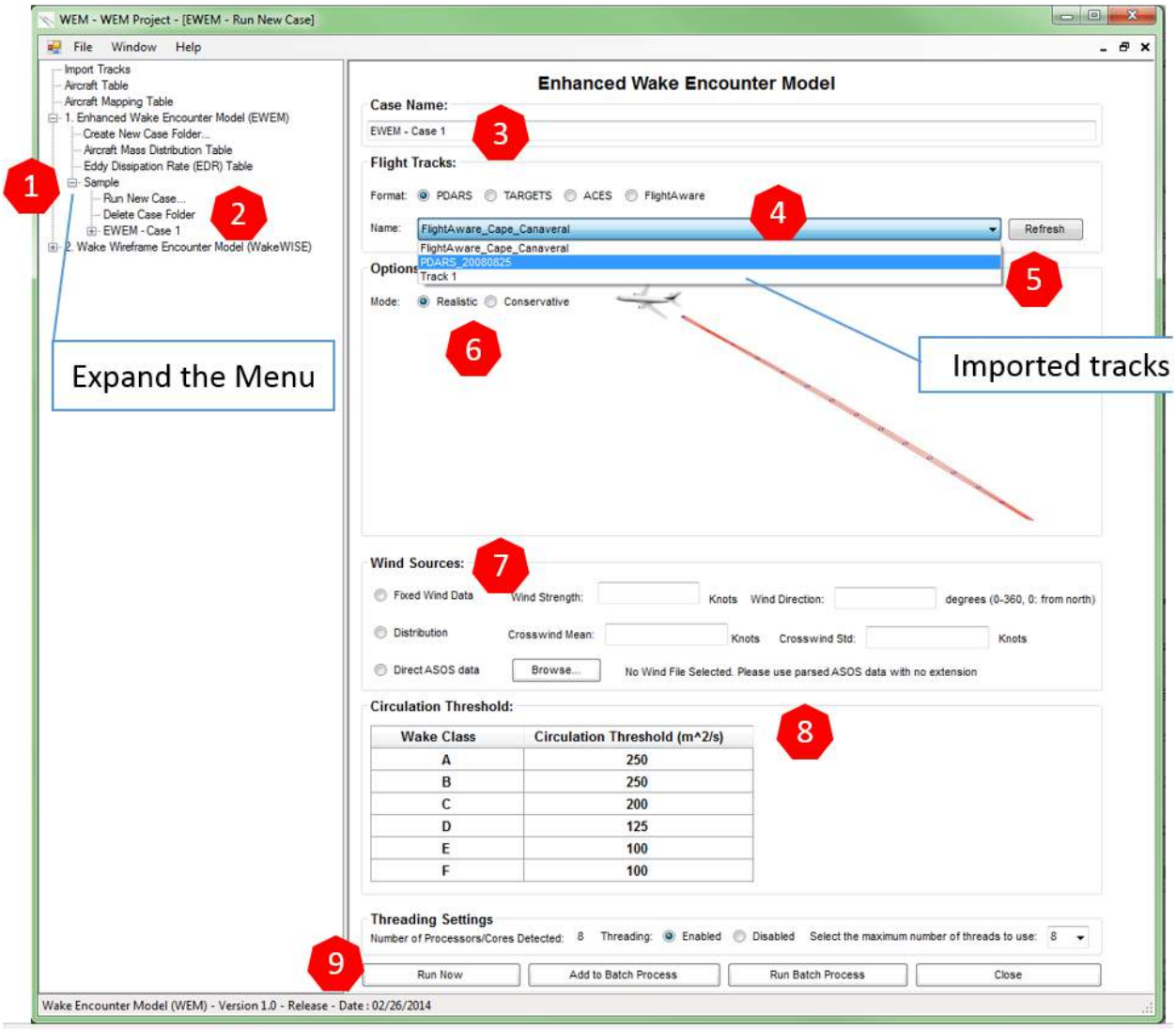


Figure 85 EWEM Run New Case Window.

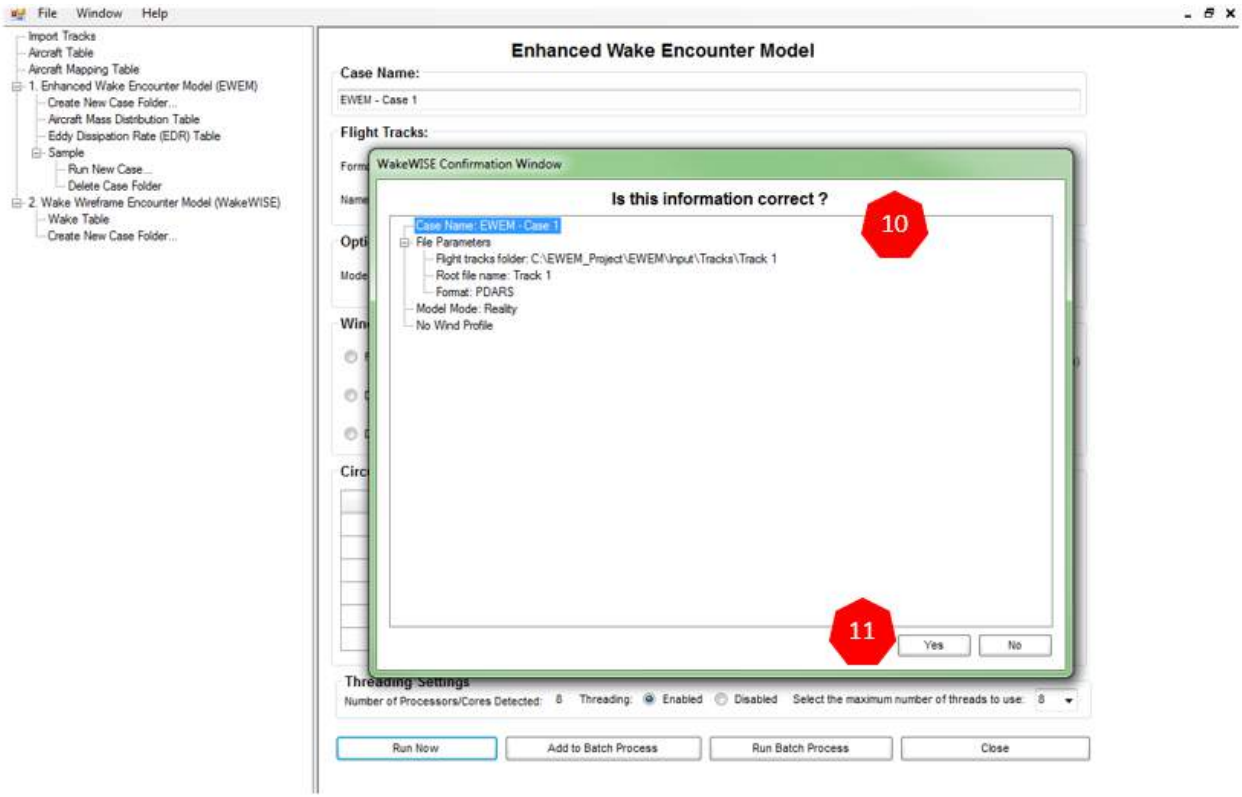


Figure 86 EWEM Run New Case – Confirmation Window.

7.3 Case Results

Once you complete running a case, or opening a previous case, if potential wake encounters have been found, they can be retrieved and studied using the main menu.

1) Click on the "+" button by the case name
2) Click on "Case Settings" to open case overview window
3) Click "Potential Encounters" to open Google Earth visualization window with simple encounter information

   Note: Double-Click on one of the potential encounters will zoom the Google Earth viewport of the potential wake encounter.
4) Click "Encounter Details" to open all the potential wake encounters summary table

Click "Delete Case" to delete the entire case

Note: Doing so will delete all the running results, but not the imported tracks

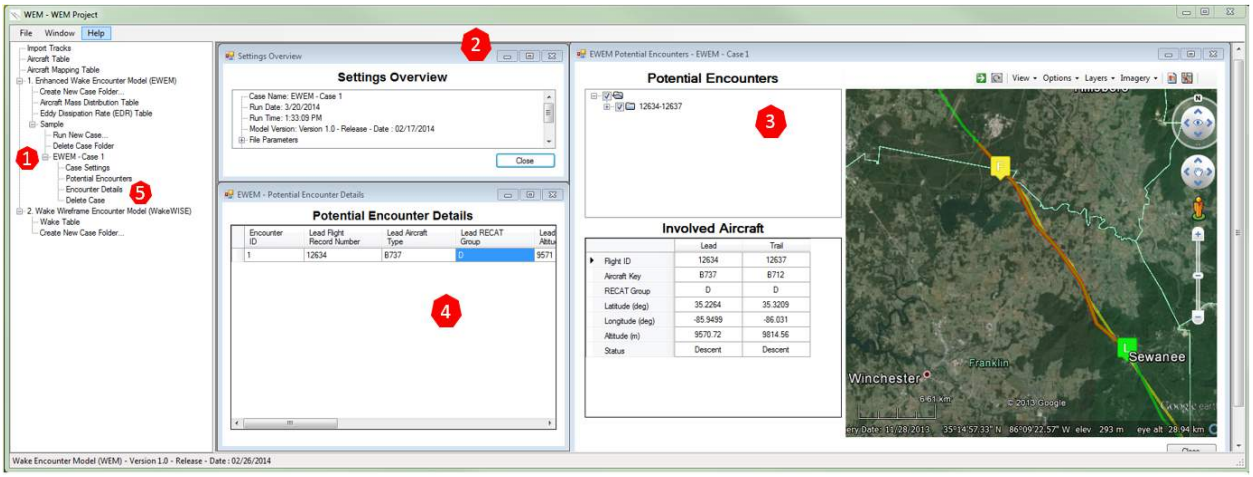


Figure 87 Case Results Sample.

7.3.1 Outputs

Both models provide details and Google Earth Visualization for each potential wake encounter found.

Note: All the outputs are stored in the Project Directory\Model Name\Output\Case Folder\Case Name\. For example, C:\EWEM_Project\EWEM\Output\Sample\EWEM - Case 1

7.3.2 Potential Wake Encounters

This provides a list of detected “potential wake encounters” with all related information and a Google Earth visualization display on the right panel.

The Google Earth add-on has all the basic functions of the software and allows users to move and rotate the view for better exploration of the potential wake encounters.

The lead aircraft (wake generator) flight track is marked in green with the green letter L representing the lead aircraft position when the potential wake encounter is detected. The trail aircraft (wake encounter) flight track is marked in yellow with a yellow letter F representing the follower aircraft position at the potential encounter. The wake zone generated by the lead aircraft is shown in red.

The .kml file can be retrieved from the output folder and opened separately in Google Earth.

The simplified table shows the basic flight status of the two aircraft involved when the potential wake encounter occurs.

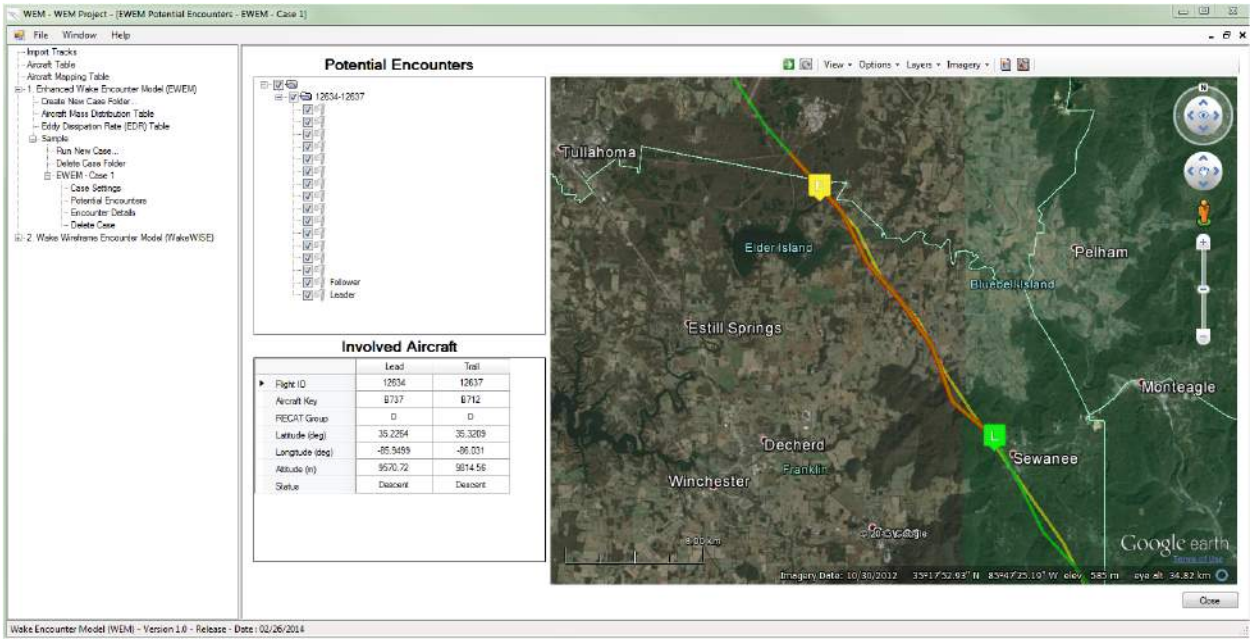


Figure 88 Potential Encounters Google Earth Visualization.

7.3.3 Potential Wake Encounter Summary

The summary table provides more information on the detected potential wake encounters, including the distance between two involved aircraft and the estimated circulation strength the trailing aircraft encountered, see Figure 89.

The data are also stored in the output folder, named after “Encounter_Detail_[Track Name].txt”. It can be read and retrieved by other software for further analysis.

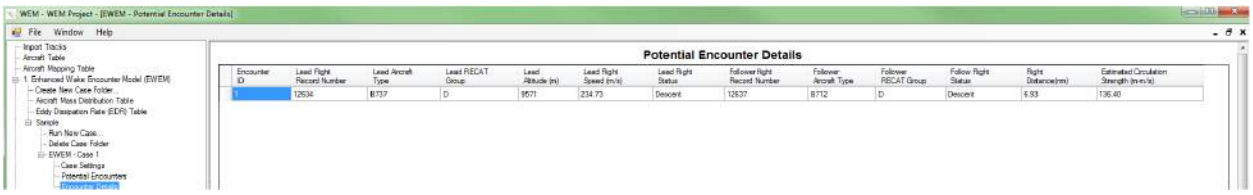


Figure 89 Encounter Summary Table.

8 Conclusion and Recommendations

8.1 Summary

A wake encounter analysis model has been developed in this research. Existing wake models mainly focus on the physical wake vortex behavior generated by aircraft. Models require explicit inputs to generate detail and accurate wake estimation over time and space. These models are intended to help analyze the wake behavior for research purposes.

In order to help planners and controllers to understand the consequences of wake vortices, the EWEM model is a first-order analysis tool to visualize protected wake zones and study potential wake encounters for a given input set of flight tracks. To generate a three-dimensional protected wake zone, we use the APA Suite of Models developed by the NASA Langley Center as the baseline to physically model wake behavior. The APA Suite of Models requires a set of explicit inputs including aircraft characteristic and atmospheric parameters. To minimize the requirement of inputs in EWEM without loosing the level of confidence, normalization of parameters and Monte Carlo simulation are used to generate wake zones that cover a multitude of environmental and flight conditions. The normalized EDR was used to decrease the complexity and allows the model to generate the wake behavior from a limited number of inputs. Monte Carlo simulation was added to ensure the confidence level of the envelope zone. 106 aircraft types were divided into six ReCat groups based on the updated FAA guideline for Re-Categorization. The most critical aircraft types were selected for each ReCat group as the representative aircraft, and a wake look-up table was build based on the critical aircraft. Unique wake envelopes are regenerated from the look-up table based on specific aircraft flight conditions.

The three-dimensional wake zone and their visualization improve the understanding of wake zones to be used in FAA terminal procedure design. In addition, the model has the ability to

address more complex problems with additional detailed inputs, such as wind condition, addition aircraft types, and different wake categorization standards. The model developed allows users to modify default variables, such as environment turbulence, temperature, or aircraft flight characteristics.

To evaluate the model performance, four types of flight data have been examined with the proposed model, including PDARS data, FlightAware data, TARGETS outputs and ACES outputs. Results show a low number of potential encounters as expected were detected in different cases. Using radar data, 0.0745‰ percent of potential encounters were reported under default circulation strength levels. More potential encounters were observed for simulation data (ACES) without conflict detection.

Two case studies were conducted with the model. With insufficient data, the model can reproduce the wake encounter with given aircraft type. Without more detailed historic data, no further evaluation can be done to address the accuracy level of the model.

This research discussed the model algorithm, and the combinatorial simulation approach to the wake partitioning analysis. The model can also be used with simulation data to do capacity analysis for selected airports or airspace.

8.2 Recommendations and Future Research

Like most models, the EWEM model can be further modified and improved as follows:

8.2.1 Model Selection

The current EWEM model was build upon the results of the TDAWP model of the APA Suite of Models. The credibility of the model depends highly on the APA model. The APA model has been verified and validated by NASA. As better wake behavior model algorithms are added to

the APA model, the EWEM should also be improved to keep the consistency and the reliability of the model.

During the research, the APA Suite of Model was the only model available to this work. No detailed comparison or evaluation was conducted between the results generated from the APA model and other models in the field. Models could have different level of confidence under different circumstances. Imbedding other models into the EWEM would allow users to choose a preferred model for a variety of conditions.

8.2.2 Data Collection

The model has only been verified using two historical wake encounter incidents cases with insufficient data. The “potential wake encounter” detected was generated based on assumptions. If complete historic data of incidents or accident were provided, the model could be modified based on the real accident results. The model credibility could also be improved.

8.2.3 Risk Assessment on Potential Wake Encounters

No general risk assessment guideline was provided or suggested in the model. The model assumes all potential encounters reported should receive the same level of awareness. The model currently has a single threshold to evaluate the hazardous of the encounters detected, the users need to evaluate encounters based on the estimated circulation strength the model predicted. It is preferable to have the level of circulation threshold to identify the severity level in potential wake encounters.

Providing simple assessments on the output can also be added to the model. Since no clear guidance was provided for the level of circulation each aircraft type can suffer, the assessment can only based on RMC value discussed in Chapter 6.2. RMC is not an accepted standard by the FAA and hence is not presented in the stand-along version of the model.

8.2.4 Dynamic Separation

The model currently categorized 99 aircraft types into the six ReCat group, and use only one aircraft type to represent each group. Under a proposed dynamic separation concept, ReCat Phase III, ICAO is proposing 100 aircraft groups to be introduced in the future. The EWEM model should be updated to match the new rules when they are published in 2016. The look-up table will need to be expanded to cover every aircraft type. As more detailed Look-up table being introduced, the wake envelopes generated are expected to be more accurate. The envelopes for many aircraft types could be smaller, hence offering capacity benefits. Less overprotection of the wake zone would yield fewer potential wake encounters.

References

"NextGEN." http://www.faa.gov/nextgen/.

Allen, Alexander, and Christian Breitsamter. 2008. "Landing Gear Influence on the Wake Vortex of a Large Transport Aircraft." *Journal of Aircraft* 45 (4):1367-1372. doi: 10.2514/1.34995.

Australian Transport Safety Bureau, Australian Government. 2009. "Wake Turbulence Event Sydney Airport, NSW 3 November 2008."

Brashears, MR, and James N Hallock. 1974. "Aircraft wake vortex transport model." *Journal of Aircraft* 11 (5):265-272.

Breitsamter, C. 2011. "Wake vortex characteristics of transport aircraft." *Progress in Aerospace Sciences* 47 (2):89-134. doi: 10.1016/j.paerosci.2010.09.002.

Browder, Jeff, Rich Gutterud and Joh Schade. 2010. "Performance Data Analysis Reporting System (PDARS) - A Valuable Addition to FAA Manager's Toolsets." *Managing the Skies* 8 (6).

Burnham, D. C.;Hallock, J.N; Greene, G.C. 2002. "Wake turbulence limits on paired approaches to Parallel Runways." *Journal of Aircraft* 39 (4).

Butler, Viggo. 2008. "Increasing airport capacity without increasing airport size."

Chein, R, and JN Chung. 1987. "Effects of vortex pairing on particle dispersion in turbulent shear flows." *International journal of multiphase flow* 13 (6):785-802.

EUROCONTROL. 2014a. USER MANUAL FOR THE BASE OF AIRCRAFT DATA (BADA) REVISION 3.12.

EUROCONTROL. 2014b. "Wake Vortex." http://www.eurocontrol.int/articles/wake-vortex.

EUROCONTROL, FAA and. 2013. "Wake RECAT Phase II Status Report." WakeNet USA, NASA Ames Conference Center, Moffett Field, CA.

FAA. 2010. FAA order 7110.308. edited by Federal Aviation Administration.

FAA. 2012a. "Task Order 7110.65."

FAA. 2012b. Terminal Area Forecast Summary Fiscal Years 2012-2040. Federal Aviation Administration.

FAA. 2013a. "NextGen Implementation Plan." In: Federal Aviation Administration.

FAA. 2013b. "SAFO 12007."

FAA. 2014. "FAA Aerospace Forecasts Fiscal Years 2014- 2034."

FAA, Wake Turbulence Research Program. 2012c. "Wake Recategorization Phase II." WakeNet USA, Boeing Field, WA, Oct 18, 2012.

Frank, Holzäpfel, Kladetzke Jan, Amelsberg Swantje, Lenz Helge, Schwarz Carsten, and Visscher Ivan De. 2008. "Aircraft Wake Vortex Scenarios Simulation For Take-Off and Departure." In *The 26th Congress of ICAS and 8th AIAA ATIO*. American Institute of Aeronautics and Astronautics.

Frech, Michael, and Frank Holz pfel. 2008. "Skill of an Aircraft Wake-Vortex Model Using Weather Prediction and Observation." *Journal of Aircraft* 45 (2):461-470. doi: 10.2514/1.28983.

Fred H. Proctor, David W. Hamilton, and David K. Rutishauser. 2004. Meteorology and Wake Vortex Influence on American Airlines FL-587 Accident.

Geisinger, Kenneth E. 1985. "Airspace Conflict Equations." *Transportation Science* 19 (2):139.

George, Sapna, and Frederick Wieland. 2011. "Build 8 of the airspace concept evaluation system (ACES)." Integrated Communications, Navigation and Surveilance Conference (ICNS), 2011.

Government Accountability Office. 2006. *Next Generation Air Transportation System: progress and challenges associated with the transformation of the National Airspace System : report to congressional requesters*. Washington, D.C.: U.S. Government Accountability Office.

Greene, G. C. 1986. "An approximate model of vortex decay in the atmosphere." *Journal of aircraft* 23 (7):566-573. doi: 10.2514/3.45345.

Hallock, James N. 2013. "Aircraft Design:Performance Data for Wake Re-Cat II." WakeNet USA, Milpitas, CA.

Hammer, Jonathan. 2000. "Case Study of Paired Approach Procedure to Closely Spaced Parallel Runways." *Air Traffic control Quarterly* 8 (3):223-252.

Hinton, S.E. Riddick; D. A. 2000. "An Initial Study of the Sensitivity of Aircraft Vortex Spacing System Spacing Sensitivity to Weather and Configuration."

Holzapfel, Frank. 2003a. "Analysis of wake vortex decay mechanisms in the atmosphere*1." *Aerospace Science and Technology* 7 (4):263-275. doi: 10.1016/s1270-9638(03)00026-9.

Holzapfel, Frank, M. Frech, T. Gerz, A. Tafferner, K. Hahn, C. Schwarz, H. Joos, B. Korn, H. Lenz, and R. Luckner. 2009. "Aircraft wake vortex scenarios simulation package – WakeScene." *Aerospace Science and Technology* 13 (1):1-11. doi: 10.1016/j.ast.2007.09.008.

Holzapfel, Frank. 2003b. "Probabilistic two-phase aircraft wake vortex decay and transport model." *Journal of Aircraft* 40 (2):9.

Holzapfel, Frank. 2003c. "Wake Vortex Transport and Decay in Ground Effect - Vortex Linking with Ground."

Holzapfel, Frank, and Meiko Steen. 2007. "Aircraft Wake–Vortex Evolution in Ground Proximity: Analysis and Parameterization." *AIAA Journal* 45 (1):218-227. doi: 10.2514/1.23917.

Holzapfel, Frank, and Jan Kladetzke. 2011. "Assessment of Wake-Vortex Encounter Probabilities for Crosswind Departure Scenarios." *Journal of Aircraft* 48 (3):812-822. doi: 10.2514/1.c000236.

Holzapfel, Frank, Jan Kladetzke, Swantje Amelsberg, Helge Lenz, Carsten Schwarz, and Ivan De Visscher. 2009. "Aircraft Wake Vortex Scenarios Simulation Package for Takeoff and Departure." *Journal of Aircraft* 46 (2):713-717. doi: 10.2514/1.39346.

Hoogstraten, Mike. 2013. "Improved Understanding of En-Route Wake Risk Factors." WakeNet USA, NASA Ames Centre, Moffett Field, CA.

IATA. 2013. Air Travel Demand Continues to Improve. edited by International Air Transport Association. http://www.iata.org/pressroom/pr/Pages/2013-10-31-01.aspx.

J. A. Tittsworth, S. R. Lang, E. J. Johnson and S. Barnes. 2012. "Federal Aviation Administration Wake Turbulence Program - Recent Highlights." The 57th Air Traffic control Association Annual Conference & Exposition, Gaylord National Resort and Convention Center, Maryland, Oct 1-3.

J.O'Connor, David K.Rutishauser and Cornelius. 2001. The NASA Aircraft Vortex Spacing System (AVOSS) Concept Demonstration Results.

Jeff Browder, Rich Gutterud and Joh Schade. 2010. "Performance Data Analysis Reporting System (PDARS) – A Valuable Addition to FAA Managers' Toolsets." *Managing the Skies* 8 (6):6-11.

Journade, Jerome. 2003. "Numerical Wake Vortex characterization." 21st Applied Aerodynamics Conference, Orlando, Florida.

Matthew Pruis, Donald Delisi. 2011. "Assessment of Fast-Time Wake Vortex Prediction Models using Pulsed and Continuous Wave Lidar Observations at Several Different Airports." 3rd AIAA Atmospheric and Space Environments Conference, 2011.

Mike Hoogstraten, Hendrikus Visser Dennis Hart, Vincent Treve, and Frederic Rooseleer. "An Improved Understanding of En-route Wake Vortex Encounters."

MITRE. 2010. "Simulation Tool Enables Quick Assessment of Air Traffic Procedures." http://www.mitre.org/sites/default/files/pdf/TARGETS.pdf.

Nanjo, A., N. Kanjo, K. Ueda, and S. Nozaki. 2005. "A structure database and its application: Maintenance Information Management System and bridge management system of Hanshin Expressway." *Structure and Infrastructure Engineering* 1 (4):285-294. doi: 10.1080/15732470500030612.

Patrick R. Veillette, Ph.D. 2002. "Data Show That U.S. Wake Turbulence Accidents are Most Frequent at Low Altitude and During Approach and Landing." *Flight Safety Digest* 21 (3-4):60.

Procter, Fred and Hinton, D. A. 1997. "Two Dimensional Wake Vortex Simulations in the Atmosphere: Preliminary Sensitivity Studies."

Proctor, David W. Hamilton and Fred H. 2000. "Wake Vortex Transport in Proximity to The Ground." 19th Digital Avionics Systems Conference, Philadelphia, Pennsylvania.

Proctor, F. H. 1996. "Numerical Simulation of Wake Vortices Measured During the Idaho Falls and Memphis Field Programs."

Proctor, F. H. and Hamilton, D. W. 2009. "Evaluation of Fast-Time Wake Vortex Prediction Models (TDAP D2P)."

Proctor, Fred. 2004. "TASS Driven Algorithm for Wake Prediction (TDAWP)." Wakenet USA, Boca Raton, FL.

Proctor, Fred H., David W. Hamilton, and George F. Switzer. 2006. "TASS Driven Algorithms for Wake Prediction." 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, USA.

R.E.Robins, D.P.Delisi. 2002. NWRA AVOSS Wake Vortex Prediction Algorithm Version 3.1.1. edited by National Aeronautics and Space Administration. Langley Research Center, Hampton, Virginia.

RITA. 2014. Airline On-Time Statistics and Delay Causes. edited by Research and Innovative Technology Administration.

Robins, Robert E, Donald P Delisi, and George C Greene. 2001. "Algorithm for prediction of trailing vortex evolution." *Journal of aircraft* 38 (5):911-917.

Sarpkaya, T. , R.E. Robins, and D.P.Delisi. 2001. "Wake-Vortex Eddy-Dissipation Model Predictions Compared with Observations." *Journal of Aircraft* 38 (4):6.

Schroeder, Nataliya. 2011. Analysis of Potential Wake Turbulence Encounters in Current and NextGen Flight Operations.

Shortle, J, and Michael Allocco. 2005. "Applying qualitative hazard analysis to support quantitative safety analysis for proposed reduced wake separation conops, 6th USA." Europe ATM R&D Seminar, Baltimore, MD, Paper.

Shortle, John and Jeddi, Babak. 2007. "Using Multilateration Data in Probabilistic Analysis of Wake Vortex Hazards for Landing Aircraft." *Transportation Research Record* 2007 (1):90-96. doi: 10.3141/2007-11.

Steven R. Lang, Jeffrey A. Tittsworth, Catalin Lepadatu,Donald P. Delisi,David Y. Lai, George C. Greene. 2009. "Progress on Joint FAA/Eurocontrol Effort to Develop an ICAO Wake Turbulence Re-Categorization." Eighth USA/Europe Seminar on Air Traffic Management Research & Development, Napa, California USA.

Switzer, George F. 2012.

Swol, Christopher Douglas. 2009. "Simulation-Based Analysis of Wake Turbulence Encounters in Current Flight Operations." Master of Science, Civil Engineering, Virginia Polytechnic Institute and State University.

Transportes, Secretaria de Comunicaciones Y. 2009. "Informe Accidente LearJet 45 XC-VMC, 4 Noviembre 2008."

Treve, Vincent. 2012. "European Proposed RECAT Amendement for Capacity Optimisation." Wake Turbulence in Current Operations and Beyond.

United States Committee on Extension to the Standard, Atmosphere. 1976. *U.S. standard atmosphere, 1976*. Washington: National Oceanic and Amospheric [sic] Administration : For sale by the Supt. of Docs., U.S. Govt. Print. Off.

WakeNet2-Europe. 2006. "Wake Vortex Research Needs for "Improved Wake Vortex Separation Ruling" and "Reduced Wake Signatures"."

Winckelmans, G., R. Cocle, L. Dufresne, and R. Capart. 2005. "Vortex methods and their application to trailing wake vortex simulations." *Comptes Rendus Physique* 6 (4-5):467-486. doi: 10.1016/j.crhy.2005.05.001.

Appendix

MATLAB Scripts

ACES_Parse_State_Data_Rolling_Window.m

```
function ACES_Parse_State_Data_Rolling_Window(Project_Dir, Track_FileName, Track_Name)
% aircraft state parser example for EWEM model
% clear all; close all; clc

% % -----------------
% % Set the Parallel Configuration file:
% if(isdeployed)
%     % -----------------
%     localConfig = [Install_Dir, '\local.settings']; % can also use uigetfile to let the USER select file
%     setmcruserdata('ParallelConfigurationFile',localConfig);
% end
%
% % matlabpool close force;
% if matlabpool('size') == 0
%     eval(['matlabpool open ', num2str(NumOfThreads)]);
% end

% mkdir('Output');

% Calculate maximum number of records for 2GB TimeWindow.
TwoGB_in_bytes = 2^31;

%% Definitions
% inputfile   = 'aircraftstate_70Kflights_VT_timesorted.csv';
% window_mins = 30;               % window size in minutes
window_overlap_mins    = 4;     % amount of overlap from last iteration (if needed)
% sampling_frequency_sec = 5;     % one sample every this many seconds for each aircraft
data_format = '%f %f %f %f %f %f %f %f %f %s %s';
% data_format = '%f %f %f %f %f %*[^\n]';

%% sequentially collect data from the aircraft state data file

% open the file for reading and grab the headerline
fid    = fopen(Track_FileName, 'r');              % open the file for reading
header = fgetl(fid);                    % get the header line
fields = textscan(header,'%s','delimiter',','); fields = fields{1};  % get a listing of the fields
Number_Of_Fields = length(fields);

% define the current time window of interest and initialize the data arrays
temp = fgetl(fid);                      % get the first data line
raw  = textscan(temp,data_format,'delimiter',',');  % parse the data
% to = raw{1};                          % first recorded simulation time
% twindow = [to to+window_mins*60*1000];  % current time window of interest in milliseconds

% for example purposes we won't cycle through the entire file (let's just cycle through N times)
N = Inf;              % limit number of time windows for this example
window_count = 0;   % count the number of times we repeat the while loop
records_count = 0;  % keep track of the number of records read (data lines)
keep_going = true;  % flag that can be used to stop the while loop

% initialize a data array and array pointers
N_window = 1e5;     % use a large array to store the data we need
Time_Window_Data_Empty = cell(N_window,Number_Of_Fields);   % initialize the array
Time_Window_Data = {};
ind_start = 1;      % pointer to start of data of interest in each window
ind_end   = 0;      % pointer to end of data of interest in each window


% for debugging only: use a figure to see how many records are in each time window
```

```
% figure(1); clf;
% subplot(2,1,1); hold on; grid on
% xlabel('Window Number')
% ylabel('Records')
% title('Number of Records For Each Time Window (Includes Overlap Records)')
% subplot(2,1,2); hold on; grid on
% xlabel('Window Number')
% ylabel('Time To Collect (s)')
% title('Time Taken to Collect Each Time Window')

% while we want to/can continue, run the while loop
% start = tic;
while keep_going

   window_count = window_count + 1;          % increment the loop counter

%     window_timer = tic;

   % scan a block of data until the end of the time window
   % disp(['Parsing Step 1/2: Reading Time Window ', num2str(window_count), '...']);
   while ~isnumeric(temp) && keep_going

      % parse the line determine if we've reached the end of the time window, save the data
      records_count = records_count + 1;

      % Resize Time Window for speed
      if mod(ind_end, N_window) == 0
        Time_Window_Data = [Time_Window_Data; Time_Window_Data_Empty];
      end

      raw  = textscan(temp,data_format,'delimiter',',');  % parse the data
      tnow = raw{1};                          % the current data sample time
      ind_end = ind_end + 1;                  % increment the location of the end of the data
      Time_Window_Data(ind_end,:) = raw;

      temp = fgetl(fid);          % get the next data line in preparation for a loop continuation

      % if tnow > twindow(2)
      if mod(ind_end-1, N_window/2) == 0
         twhos = whos('Time_Window_Data');
         tsize_bytes = twhos.bytes;

         % PercentFull = round(tsize_bytes / TwoGB_in_bytes * 1000) / 10;
         % disp(['   Processed Records: ', num2str(ind_end),  ' - ', num2str(PercentFull), '%']);

         if tsize_bytes > TwoGB_in_bytes * 0.95 % If TimeWindow array size is greater than 95% of maximum array size
            tstart_new = tnow-window_overlap_mins*60*1000;    % start of next window
            % twindow = [tstart_new tstart_new+window_mins*60*1000];  % next time window
            break;      % reached the end of this time window, stop
         end
      end
   end

   % shown me the time window that I have and the number of records in it
%     t_read = toc(window_timer);
   t1s = datestr(unixtime2mat(Time_Window_Data{ind_start,1}/1000),0);
   t2s = datestr(unixtime2mat(Time_Window_Data{ind_end-1,1}/1000),0);
   nrecs = ind_end-ind_start;
   disp(['Parsing Step 1/2: Time window ',num2str(window_count),' (',t1s,' - ',t2s,') has ',num2str(nrecs),' records']);
%     figure(1);
%     subplot(2,1,1); plot(window_count,nrecs,'b.');
%     subplot(2,1,2); plot(window_count,t_read,'r.');

   ACES = Time_Window_Data(ind_start:ind_end,:);

   filename = [Project_Dir, '\', Track_Name, '_Window_', num2str(window_count), '.mat'];

   % disp(['Saving Time Window ', num2str(window_count), '...']);
   save(filename, 'ACES', '-v7');
```

```
    clear ACES;

    % because we use a time window that may overlap, we need to make sure we keep any of the overlap
    % data in the time window for the next data read
    ind2overlap = find([Time_Window_Data{:,1}]>=tstart_new);
    if ~isempty(ind2overlap)
        Time_Window_Data_Overlapp = Time_Window_Data(ind2overlap,:);
        Time_Window_Data = Time_Window_Data_Empty;
        Time_Window_Data(1:length(ind2overlap),:) = Time_Window_Data_Overlapp;
        ind_end = length(ind2overlap);
        clear Time_Window_Data_Overlapp;
    else
        % reset the pointer before next window read
        ind_end = 0;
        Time_Window_Data = {};
    end

    % if we have reached the end of the file, stop
    if isnumeric(temp)
        keep_going = false;
    end

    % if we've reached our example limit number of times, stop the loop
    if window_count >= N
        keep_going = false;
    end


end
% toc(start)

fclose(fid);                    % close the file when done

save ([Project_Dir, '\window_count.mat'], 'window_count');
return;

ACES_Parser.m
% Project_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/ACES_NASA/ACES_Raw';
% Track_Name = 'ACES_70Kflights_VT';

function ACES_Parser(Project_Dir, Track_Name)
% This m-file parses the 5-min ACES data pre selected from
% example_parse_state_data_rolling_window.m
% Date: 5/24/2013

% clc;
% clear all;
%
% warning off;

%% Get all the flighttracks that with in the time window

%Open input file

% start = tic;

% figure(1); clf;
% hold on; grid on
% xlabel('Window Number')
% ylabel('Time To Parse (s)')
% title('Time Taken to Parse Each Time Window')

load ([Project_Dir, '\window_count.mat']);

for i = 1:window_count

    disp(['Parsing Step 2/2: Number of files processed: ', num2str(i)]);

    filename = [Project_Dir, '\', Track_Name, '_Window_', num2str(i), '.mat'];
```

```
%     window_timer = tic;

    load (filename);

    FlightTime = cell2mat(ACES(:,1));
    FlightID = cell2mat(ACES(:,2));
    FlightLatitude = cell2mat(ACES(:,3));
    FlightLongitude = cell2mat(ACES(:,4));
    FlightAltitude = cell2mat(ACES(:,5));
    FlightAirSpeed= cell2mat(ACES(:,6));
    FlightHeading = cell2mat(ACES(:,7));
    FlightGroundSpeed = cell2mat(ACES(:,8));
    FlightControlCenter = ACES(:,10);
    FlightType = ACES(:,11);

    %% Parse all tracks that filtered out from the time window into a structure file
    FlightNumber_List = unique(FlightID);
    %
    % idx = find(ACES5(2));

    FlightTrack = [];
    for j = 1:length(FlightNumber_List);

        Flight_ID = FlightNumber_List(j); %58609

        idx = find(Flight_ID == FlightID);

        %Find the last track record, this idx+1 is the first track record of
        %the next flight
        %     record_list(i+1) = find(FlightID==FlightNumber_List(i),1,'last');

        %Find the Aircraft Type from the MEM_SIDs_Baseline_flightkey.csv file
        %based on the flight record number retrieved in the MEM_SIDs_Baseline_tracks
        %     flightlist_index = find(FlightNumber_List(i) ==flightrecord,1,'first');
        %Save all the related flight track data separatedly in to the structure
        %file
        FlightTrack(j).actype = FlightType(idx(1));
        FlightTrack(j).actype = char(FlightTrack(j).actype{1});
        FlightTrack(j).Recordnum = FlightID(idx(1));
        FlightTrack(j).Time = FlightTime(idx)./1000;
        FlightTrack(j).Latitude = FlightLatitude(idx);
        FlightTrack(j).Longitude = FlightLongitude(idx);
        FlightTrack(j).Altitude = FlightAltitude(idx)*0.3048;  %meter
        FlightTrack(j).AirSpeed = FlightAirSpeed(idx);
        FlightTrack(j).Heading = FlightHeading(idx);
        FlightTrack(j).GroundSpeed = FlightGroundSpeed(idx)*0.514; %knots to m/s
        FlightTrack(j).ControlCenter = FlightControlCenter(idx);
        FlightTrack(j).Number_of_Trackpoints  = length(idx);
    end

    fname = [Project_Dir, '\', Track_Name, '_str_', num2str(i), '.mat'];

    save (fname, 'FlightTrack')

    fclose('all');
    % clear window_timer t_read

end

return;
```

ASOS Parser.m

```
function [wind_time_all,wind_direction_all, wind_speed_all] = ASOS_Parser(Wind_File)

%% Import data from text file.
% Script for importing data from the following text file:
%
%Wind_File = '64050KJFK201401_1.txt';
```

```
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

recordcount = 0;

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
%formatSpec = '%*9s%20s%8s%2s%10s%2s%10s%2s%8s%6s%6s%5s%[^\\n\r]';

%% Open the text file.
fileID = fopen(Wind_File,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = fgetl(fileID);

while  ischar(dataArray)
   recordcount = recordcount +1;
   %% Create output variable
   Wind_Year(recordcount,1) = str2num(dataArray(14:17));
   Wind_Month(recordcount,1)= str2num(dataArray(18:19));
   Wind_Date(recordcount,1) = str2num(dataArray(20:21));
   Wind_HH(recordcount,1) = str2num(dataArray(26:27));
   Wind_MM(recordcount,1) = str2num(dataArray(28:29));
   wind_time_all(recordcount,1) =
datenum(Wind_Year(recordcount,:),Wind_Month(recordcount,:),Wind_Date(recordcount,:),Wind_HH(recordcount,:),
Wind_MM(recordcount,:),0);
   wind_direction_all(recordcount,1) = str2num(dataArray(81:83));
   wind_speed_all(recordcount,1) = str2num(dataArray(88:89)); %knots

      dataArray = fgetl(fileID);
end

%% Clear temporary variables
%return
```

DensityLookup.m

```
function [density] = densitylookup(ISADensitytable,wakealtitude)

density = interp1(ISADensitytable(: , 1), ISADensitytable(: , 2), double(wakealtitude));

return;
```

Encounter_Summary.m

```
function Encounter_Summary(Install_Dir, Tracks_Dir, Project_Dir, FlightTracks_Name, Track_Type, Circulation_Threshold)

%% This script will plot conflicts detected by EWEM



% Install_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL (trunk)/';

% Project_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/ACES_NASA/CI_Mode_1/';

% Tracks_Dir  = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/ACES_NASA/Nick_Mode_1/';

% Track_FileName= sprintf('aircraftstate_70Kflights_VT_timesorted.csv');

% FlightTracks_Name ='ACES'; %'ACES';%'ACES_70Kflights_VT';

%

% Track_Type     = 'ACES'; % PDARS, MEM, ACES
```

```
if strcmp(Track_Type, 'ACES') == 1
    load ([Tracks_Dir, 'window_count.mat']);
    Number_Of_Files = window_count;
else
    Number_Of_Files = 1;
end
%% Global Variables
% g = 9.81;%Gravitational constant (m/s^2)
% s = pi/4;%Trailing vortices convert parameter
% m_to_km = 0.001;
% knots_to_ms = 0.514; %convert knots to m/s
% ft_to_meters = 0.3048; %convert ft to meters
% fl_to_meters = 100*ft_to_meters; %convert fight level to meters
number_of_total_encounters =0;
% boundary_all  = load([Install_Dir, 'boundary_FINAL.mat']);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%Import Aircraft information table (mass, wingspan, etc)
fid1 = fopen([Install_Dir, 'Aircraft_Info_Common_metric_units.csv'],'r');
aircraftinfolookup = textscan(fid1, '%s%s%s%s%s%s%f%f%f%f%f%s%f', 'HeaderLines',1,'delimiter', ',');
%Field 1: Aircraft ID (BADA Name)
%Field 2: Name
%Field 3: Manufacture Name
%Field 4: Short Name
%Field 5: BADA Name
%Field 6: Wake Class (Old)
%Field 7: Operating Empty Weight OEW (kg)
%Field 8: Maximum Landing Weight (kg)
%Field 9: Maximum Takeoff Weight (kg)
%Field 10: Wingspan (m)
%Field 11: ReCategorization wakeclass from A-F
%Field 12: ReCategorization wakeclass presented in number from 1-6
acid      = aircraftinfolookup{1};
% wingspan     = aircraftinfolookup{10}; %m
wakeclass = aircraftinfolookup{12};
```

```
wakeclass_num = aircraftinfolookup{13};
% oew_all = aircraftinfolookup{7}; %kg
% mlw_all = aircraftinfolookup{8}; %kg
% mtow_all = aircraftinfolookup{9}; %kg

fid2 = fopen([Install_Dir, '/BADA_Mapping.csv'],'r');
aircraftmappingtable = textscan(fid2, '%s%s%s%s%s%s%f%f%f%f%f%s%f', 'HeaderLines',1,'delimiter', ',');
%Field 1: Aircraft ID (BADA Name)
%Field 2: Manufacture Name
%Field 3: Name or Model
%Field 4: File (Which profile is going to use in the Aircraft_Info_Common_metric_units.csv table)
%Field 5: Old file name
aircraft_list        = aircraftmappingtable{1};
aircraft_mapping_list = aircraftmappingtable{4}; %m

% Open Summary File
% Header: {Conflict ID, leadrecordnum,{actype_lead},leader_aircraft_CAT,Flight_Altitude,Flight_GroundSpeed, Flight_angle_leader,
followerrecordnum,{actype_follower},follower_aircraft_CAT,Flight_distance(number_of_total_encounters),Circulation_Strength,Flight_angle};
%InitialCirculation,};
fid_summary = fopen([Project_Dir, 'Encounter_Detail_', FlightTracks_Name, '.txt'], 'w');
fprintf(fid_summary, '%s\n', 'Conflict ID,Lead Flight Recordnum,Lead Aircraft Type, Lead RECAT Group,Lead Altitude (m),Lead Flight Speed
(m/s),Lead Flight Statues,Follower Flight Recordnum,Follower Aircraft Type, Follower RECAT Group,Follow Flight Statues, Flight Distance
(nm),Estimated Circulation Strength (m-m/s)');

for z = 1: Number_Of_Files
   disp(['Summary: ', num2str(z), '/', num2str(Number_Of_Files)]);

   %% Import all flight track data

   if strcmp(Track_Type, 'ACES') == 1
      load([Tracks_Dir, FlightTracks_Name, '_str_', num2str(z), '.mat']); %load the flight track with constructed envelope
   else
      load([Tracks_Dir, 'Parsed_', FlightTracks_Name, '.mat']); %load the flight track with constructed envelope
   end
```

```
if strcmp(Track_Type, 'ACES') == 1
    WP_FileName = [Project_Dir, 'wakepiercing_', FlightTracks_Name, '_str_', num2str(z), '.mat'];
    FL_FileName = [Tracks_Dir, FlightTracks_Name, '_str_', num2str(z), '_FlightList.txt'];
else
    WP_FileName = [Project_Dir, 'wakepiercing_', FlightTracks_Name, '.mat'];
    FL_FileName = [Tracks_Dir, FlightTracks_Name, '_FlightList.txt'];
end

Detected_Encounters = load(WP_FileName);

%List of aicraft contained in the MEM structure file
fid4 = fopen(FL_FileName, 'r');
flightlist = textscan(fid4, '%f', 'delimiter', ',');
flightlist = flightlist{1};

for x = 1:length(Detected_Encounters.wakepiercing_sum)

    % Skip to next record if no encounter recorded
    if isempty(Detected_Encounters.wakepiercing_sum(x).filteredleadrecord) == 1
        continue;
    end

    number_of_total_encounters = number_of_total_encounters+1;

    leadrecordnum = Detected_Encounters.wakepiercing_sum(x).filteredleadrecord(1);
    i = find(leadrecordnum==flightlist);
    j = Detected_Encounters.wakepiercing_sum(x).filteredleadrecord(2);

    followerrecordnum = Detected_Encounters.wakepiercing_sum(x).filteredfollowerrecord(1);
    m = find(followerrecordnum==flightlist);
    n = Detected_Encounters.wakepiercing_sum(x).filteredfollowerrecord(2);

    actype_lead = FlightTrack(i).actype; %Get the flight type "B722"
    actype_follower = FlightTrack(m).actype;
```

```
% Mapping the aircraft type into a BADA aircraft type
leaderidx_mapping   = find(strcmp(aircraft_list,actype_lead),1);
followeridx_mapping = find(strcmp(aircraft_list,actype_follower),1);
leadertype_mapping   = aircraft_mapping_list(leaderidx_mapping);
followertype_mapping = aircraft_mapping_list(followeridx_mapping);

actype_lidx = find(strcmp(acid,leadertype_mapping),1); %Find the corresponding line in the Aircraft_info table for aircraft parameters
actype_fidx = find(strcmp(acid,followertype_mapping),1);
leader_aircraft_CAT_num = wakeclass_num(actype_lidx);
leader_aircraft_CAT= wakeclass{actype_lidx};
follower_aircraft_CAT = wakeclass{actype_fidx};



Near_Altitude = FlightTrack(m).Altitude(max(n-3,1):min(n+3,FlightTrack(m).Number_of_Trackpoints));

if (Near_Altitude(end) - Near_Altitude(1))>0
   Flight_angle ='Climb'; %climb
elseif (Near_Altitude(end) - Near_Altitude(1))<0
   Flight_angle = 'Descent'; %descent
else
   Flight_angle ='Cruise'; %cruise
end

Near_Altitude_leader = FlightTrack(i).Altitude(max(j-3,1):min(j+3,FlightTrack(i).Number_of_Trackpoints));

if (Near_Altitude_leader(end) - Near_Altitude_leader(1))>0
   Flight_angle_leader ='Climb'; %climb or 1
elseif (Near_Altitude_leader(end) - Near_Altitude_leader(1))<0
   Flight_angle_leader = 'Descent'; %descent or -1
else
   Flight_angle_leader ='Cruise'; %cruise
end

Flight_distance(number_of_total_encounters) = deg2nm(distance(FlightTrack(m).Latitude(n),
FlightTrack(m).Longitude(n),FlightTrack(i).Latitude(j),FlightTrack(i).Longitude(j)));
```

```
        Envelope_Distance = deg2nm(Detected_Encounters.wakepiercing_sum(x).envelopelength(end));

        InitialCirculation = Detected_Encounters.wakepiercing_sum(x).InitialCirculation;

        Flight_GroundSpeed = FlightTrack(i). GroundSpeed(j);
        Flight_Altitude = FlightTrack(i). Altitude(j);

        Circulation_Drop = Circulation_Threshold(leader_aircraft_CAT_num) - InitialCirculation;

        Circulation_Strength = (Flight_distance(number_of_total_encounters) /Envelope_Distance) * Circulation_Drop + InitialCirculation;

        % Header:'Conflict ID,Lead Flight Recordnum,Lead Aircraft Type, Lead RECAT Group,Lead Altitude (m),Lead Flight Speed (m/s),Lead
Flight Statues,Follower Flight Recordnum,Follower Aircraft Type, Follow RECAT Group,Follow Flight Statues, Flight Distance (nm),Estimated
Circulation Strength (m-m/s)');
        fprintf(fid_summary, '%.0f,%.0f,%s,%.0f,%s,%.2f,%s,%.0f,%s,%s,%s,%.2f,%.2f\n', z,leadrecordnum, actype_lead, Flight_Altitude,
leader_aircraft_CAT,
Flight_GroundSpeed,Flight_angle_leader,followerrecordnum,actype_follower,follower_aircraft_CAT,Flight_angle,Flight_distance(number_of_t
otal_encounters),Circulation_Strength);
        %Encounter_pair(number_of_total_encounters,:) = {leadrecordnum,
followerrecordnum,{actype_lead},{actype_follower},leader_aircraft_CAT,follower_aircraft_CAT,Flight_distance(number_of_total_encounters),
Circulation_Strength, Flight_angle_leader,Flight_angle,Flight_GroundSpeed,Flight_Altitude};%InitialCirculation,};
        %save ([Project_Dir, '/Wake_Summary_Table_', FlightTracks_Name, '.mat'], 'Encounter_pair');
    end
end
fclose all;
return
```

EWEM.m

```
function EWEM(Install_Dir, Tracks_Dir, Project_Dir, FlightTracks_Name, NumOfThreads, Track_Type, Model_Mode,
Circulation_Threshold,crosswind_all)

% % For Testing
% clc;
% clear all;
% close all;
%
% Install_Dir = 'D:/WEM_SVN/WEM_GUI/WEM_GUI/DATA/EWEM'; % pwd;
% Tracks_Dir  = 'D:/WEM Project/EWEM/Input/Tracks/PDARS Track 1'; % [pwd,'/Output'];
% Project_Dir = 'D:/WEM Project/EWEM/Output/Test/PDARS - Case 1'; % [pwd,'/Output'];
% FlightTracks_Name = 'PDARS Track 1';
% NumOfThreads = 8;
```

```
% Track_Type   = 'PDARS'; % PDARS, MEM, ACES
% Model_Mode   = 0; %  Model_Mode: Conservative or reality analysis, default value is 0 (reality)

%Wake Encounter Model
%Model to determine if an aircraft pierces a wake envelope
%parsed PDARS list containing flight paths of each aircraft

%Nataliya Schroeder, May 10, 2010
%%%Added comments with field names;changed imported files to floating;removed xlsread from function file;
%created a function generateenvelope that generates wake envelope
%corrected latitude and longitude, they were switched in the previous code
%heading values from original PDARS data inconsistent with lat/ long, values corrected and loaded a separate variable
%lat&long&altitude from PDARS smoothed
%added code to bend the envelope long the PDARS trajectory
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
% axes('FontSize',20);
% clear all
% clc

%% Load input files including FlightTrack.FlightTrack, Wake Envelope, filtered pairs
if strcmp(Track_Type, 'ACES') == 1
   FlightTrack = load([Tracks_Dir, FlightTracks_Name, '.mat'],'FlightTrack'); %load the flight track with constructed envelope
else
   FlightTrack = load([Tracks_Dir, 'Parsed_', FlightTracks_Name, '.mat'],'FlightTrack'); %load the flight track with constructed envelope
end
FlightTrack = FlightTrack.FlightTrack;
Filtered_Pairs = load([Tracks_Dir, 'Filtered_Pairs_', FlightTracks_Name, '.mat']);
boundary_all  = load([Install_Dir, 'Look_up_Table.mat']);

%% Import all required files to run the WEM

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%Import ISA Density table with values for density based on altitude
ISADensitytable = xlsread([Install_Dir, 'ISADensityTable.xls']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%Import Aircraft information table (mass, wingspan, etc)
fid1 = fopen([Install_Dir, 'Aircraft_Info_Common_metric_units.csv'],'r');
aircraftinfolookup = textscan(fid1, '%s%s%s%s%s%s%f%f%f%f%f%s%f', 'HeaderLines',1,'delimiter', ',');
%Field 1: Aircraft ID (BADA Name)
%Field 2: Name
%Field 3: Manufacture Name
%Field 4: Short Name
%Field 5: BADA Name
%Field 6: Wake Class (Old)
%Field 7: Operating Empty Weight OEW (kg)
%Field 8: Maximum Landing Weight (kg)
%Field 9: Maximum Takeoff Weight (kg)
%Field 10: wakeclass presented in number from 1-6
%Field 11: ReCategorization wakeclass from A-F
%Field 12: ReCategorization wakeclass presented in number from 1-6
acid      = aircraftinfolookup{1};
wingspan     = aircraftinfolookup{10}; %m
% wakeclass = aircraftinfolookup{12};
wakeclass_num = aircraftinfolookup{13};
oew_all = aircraftinfolookup{7}; %kg
mlw_all = aircraftinfolookup{8}; %kg
mtow_all = aircraftinfolookup{9}; %kg

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%Import the table containing departures and arrivals mean weights per aicraft class
acftdistro = csvread([Install_Dir, 'NewAircraftMassDistribution.csv'],0,1);
%Field 1: dept mean
%Field 2: dept sd
%Field 3: arr mean
%Field 4: arr sd
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%Import the edr list used in the APA Suite of Model to calculate the
%normalized edr value
fid2 = fopen([Install_Dir, 'QDATA'],'r');
edr_list = textscan(fid2,'%f%f','delimiter',',');
edr_value = edr_list{2};
edr_alt = edr_list{1};
edr_value(1) = [];
edr_alt(1) = [];
fclose all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%List of aicraft pairs produced by the Time Filter Script
filteredleadrecord     = double(Filtered_Pairs.filtered_pairs(:,1));
filteredfollowerrecord = double(Filtered_Pairs.filtered_pairs(:,2));
%List of aicraft contained in the MEM structure file
fid3 = fopen([Tracks_Dir, FlightTracks_Name, '_FlightList.txt'], 'r');
flightlist = textscan(fid3, '%f', 'delimiter', ',');
flightlist = flightlist{1};
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%Aircraft Mapping table for BADA
fid4 = fopen([Install_Dir, 'BADA_Mapping.csv'],'r');
aircraftmappingtable = textscan(fid4, '%s%s%s%s%s%s%f%f%f%f%f%s%f', 'HeaderLines',1,'delimiter', ',');
%Field 1: Aircraft ID (BADA Name)
%Field 2: Manufacture Name
%Field 3: Name or Model
%Field 4: File (Which profile is going to use in the Aircraft_Info_Common_metric_units.csv table)
%Field 5: Old file name
aircraft_list        = aircraftmappingtable{1};
aircraft_mapping_list = aircraftmappingtable{4}; %m
fclose all;

%%
%CONSTANTS and PREDETERMINED VALUES---CAN BE CHANGED BY USER
g = 9.81;%Gravitational constant (m/s^2)
s = pi/4;%Trailing vortices convert parameter

m_to_km = 0.001;

time_separation= 200; %sec
assigned_num_of_points_back = 20;
Base_Date_Number = datenum('01/01/1970'); % In number of days since 01/01/0000
EDR_range_Table = [1,2,3,4,5;1,2,3,4,5;1,2,3,4,5;1,2,3,4,4;1,2,3,4,4;1,2,2,2,2]; %Different normalized edr boundary for different aircraft type

OGE_boundary_factors = [4;2.5;2;1.5;1]; %Corresponding to different normalized edr range

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

number_of_pairs = length(filteredleadrecord);%length of available records from the time-filtered file. Multiple records per individual aircraft.
Will be use

%%

rng(0); % Reset random generator to make sure that 2 different runs produce the same results if input fiel was unchanged.

% Preallocation for speed
if number_of_pairs == 0 % Exit if no pairs found
    return;
else
    wakepiercing_sum(number_of_pairs).filteredleadrecord     = [];
    wakepiercing_sum(number_of_pairs).filteredfollowerrecord = [];
end

% -----------------
% Set the Parallel Configuration file:
if(isdeployed)
```

```
    % -----------------
    localConfig = [Install_Dir, 'local_mcr.settings']; % can also use uigetfile to let the USER select file
    setmcruserdata('ParallelConfigurationFile',localConfig);
end

% matlabpool close force;
if matlabpool('size') == 0
    eval(['matlabpool open ', num2str(NumOfThreads)]);
end

% Progress bar
if(isdeployed)
    %     JAVA_DIR = [Install_Dir, '/java'];
    %     eval(['pctRunOnAll javaaddpath(''', JAVA_DIR, ''')']);
else
    pctRunOnAll javaaddpath java;
end
progressStepSize = floor(number_of_pairs/100);
ppm = ParforProgMon('Progress: ', number_of_pairs, progressStepSize, 300, 80);

% tic;
% parfor i = 1:number_of_pairs
for i = 1:number_of_pairs

    if mod(i,progressStepSize)==0
        ppm.increment();
    end

    %Find the corresponding flight record in the filtered list
    leadidx     = find(flightlist==filteredleadrecord(i),1); %find the correspoinding structure file for the leading aircraft
    followeridx = find(flightlist==filteredfollowerrecord(i),1); %find the followers for the leading aircraft

    actype_lead   = FlightTrack(leadidx).actype; %Get the flight type
    actype_follow = FlightTrack(followeridx).actype;

    % Mapping the aircraft type into a BADA aircraft type
    leaderidx_mapping   = find(strcmp(aircraft_list,actype_lead),1);
    followeridx_mapping = find(strcmp(aircraft_list,actype_follow),1);

    if ~isempty(leaderidx_mapping) && ~isempty(followeridx_mapping)   %if the aircraft type is contained in the expanded BADA list

        leadertype_mapping   = aircraft_mapping_list(leaderidx_mapping);
        followertype_mapping = aircraft_mapping_list(followeridx_mapping);

        actype_lidx = find(strcmp(acid,leadertype_mapping),1); %Find the corresponding line in the Aircraft_info table for aircraft parameters
        actype_fidx = find(strcmp(acid,followertype_mapping),1);

        wspan = wingspan(actype_lidx); %m, Get the wingspan for this aircraft type

        OGE_boundary = OGE_boundary_factors*wspan; %we consider no IGE will be observed over 4*wingspan high

        SpanInitial         = s * wspan; %b0;
        SpanInitial_km      = SpanInitial * m_to_km;
        SpanInitialHalf_deg = km2deg(SpanInitial_km/2);
        aircraft_CAT = wakeclass_num(actype_lidx); %Get the Re-CAT group of this aircraft type in number
        aircraft_CAT_follower = wakeclass_num(actype_fidx); %Get the Re-CAT group of this aircraft type in number
        oew  = oew_all(actype_lidx); %kg, Get the empty operation weight of this aircraft type in number
        mtow = mtow_all(actype_lidx);%kg, Get the maximum takeoff weight of this aircraft type in number
        malw  = mlw_all(actype_lidx); %kg, Get the maximum landing weiht of this aircraft type in number

        number_of_lpoints = length(FlightTrack(leadidx).Latitude);
        % number_of_fpoints = length(FlightTrack(followeridx).Latitude);

        followtime      = FlightTrack(followeridx).Time;%[seconds since 1970]
        % followlatitude  = FlightTrack(followeridx).Latitude;%[degrees]
        % followlongitude = FlightTrack(followeridx).Longitude;%[degrees]
        % followaltitude  = FlightTrack(followeridx).Altitude;%[x100ft]

        for j = 2:number_of_lpoints  %for all flight level
```

```
leadtime     = FlightTrack(leadidx).Time(j);%[seconds]
leadlatitude = FlightTrack(leadidx).Latitude(j);%[degrees]
leadlongitude= FlightTrack(leadidx).Longitude(j);%[degrees]
leadaltitude = FlightTrack(leadidx).Altitude;%[m]
groundspeed  = FlightTrack(leadidx).GroundSpeed(j); %[m/s]
heading      = FlightTrack(leadidx).Heading(j); %[degree 0~360]

time_interaction = (abs(leadtime - followtime) <= time_separation);

if any(time_interaction) == 0 % Skip because no time interaction found
   continue;
end

index_follower = find(time_interaction==1);
number_of_close_points = length(index_follower);

for k = 1:number_of_close_points

   % Define some variables to prevent the parfor warnings
   boundary_coordinates = [];
   z_upper = [];
   z_lower = [];
   y_modified_left = [];
   y_modified_right = [];
   x_modified = [];

   followtime      = FlightTrack(followeridx).Time(index_follower(k));%[seconds since 1970]
   followlatitude  = FlightTrack(followeridx).Latitude(index_follower(k));%[degrees]
   followlongitude = FlightTrack(followeridx).Longitude(index_follower(k));%[degrees]
   followaltitude  = FlightTrack(followeridx).Altitude(index_follower(k));%[m]

   % Density lookup is slow. Only lookup after filtering
   rho = densitylookup(ISADensitytable,leadaltitude(j));
   edr_index = find (edr_alt > leadaltitude(j),1);

   if isempty(edr_index) %if the altitude is beyond the edr list altitude range, use the last edr value available on the list
      edr = edr_value(end);
   else
      edr_down = edr_value(edr_index); %interpolate the edr value
      edr_up = edr_value(edr_index-1);
      % edr = edr_down + vpa((leadaltitude(j)-edr_alt(edr_index-1))*(edr_up-edr_down)/(edr_alt(edr_index)-edr_alt(edr_index-1)));
      edr = edr_down + (leadaltitude(j)-edr_alt(edr_index-1))*(edr_up-edr_down)/(edr_alt(edr_index)-edr_alt(edr_index-1));
   end %if isempty(edr_index)

   if leadaltitude(end) >= leadaltitude (1) %Climb, use departure mass profile
      %% Generate a reasonable random aircraft mass by using the weight distribution
      meandepartureweight = 1.3*oew + min(1, acftdistro(aircraft_CAT, 1))*(mtow-1.3*oew);
      acmass = (meandepartureweight + rand(1)*meandepartureweight*acftdistro(aircraft_CAT, 2)); % kg
   else %Descent, use arrival mass profile
      meanlandingweight = oew + min(1, acftdistro(aircraft_CAT, 3))*(malw-oew);
      acmass = (meanlandingweight + rand(1)*meanlandingweight*acftdistro(aircraft_CAT, 4)); %kg
   end
   %% Calculate the Normalized EDR value, and determin which range it falls in the structure file used for the envelope construction

   CirculationInitial = (4*acmass*g)/(rho*pi*wspan*groundspeed);  %M^2/S
   VdropInitial = CirculationInitial/(2*pi*SpanInitial); %Initial Vortex Descent Rate m/s
   TimeInitial = SpanInitial/VdropInitial; %s
   EDRNorm =(2*pi*edr^(1/3)*(SpanInitial).^(4/3))/CirculationInitial;

   if EDRNorm >= 0 && EDRNorm < 0.15
      EDR_range = EDR_range_Table(aircraft_CAT,1); % [IGE boundary, OGE look-up table position]
   elseif EDRNorm > 0.15 && EDRNorm < 0.25
      EDR_range = EDR_range_Table(aircraft_CAT,2);
   elseif EDRNorm > 0.25 && EDRNorm < 0.35
      EDR_range = EDR_range_Table(aircraft_CAT,3);
   elseif EDRNorm > 0.35 && EDRNorm < 0.45
      EDR_range = EDR_range_Table(aircraft_CAT,4);
   else
      EDR_range = EDR_range_Table(aircraft_CAT,5);
   end % for if EDRNorm >= 0 & EDRNorm < 0.15
```

```
        if CirculationInitial > Circulation_Threshold(aircraft_CAT_follower) %Only if the leader can produce a significant wake strengh for
the follower

            Circulation_Threshold_Level = Circulation_Threshold(aircraft_CAT_follower)/25-3; %1 to 7 corresponds to 100 to 250

            %% Out of Ground Effect Envelope Formation
            % Simple formation of the envelope
            % Only the envelope length information is needed, and the
            % lateral expansion is computed based on the vertical drop

            if leadaltitude(j) >= OGE_boundary(EDR_range)

               if leadaltitude(end) >= leadaltitude (1)
                  boundary_coordinates =
boundary_all.boundary_all_ISA(aircraft_CAT).dep.edr(EDR_range).Circulation_Threshold(Circulation_Threshold_Level).ogecoordinates;
               else
                  boundary_coordinates =
boundary_all.boundary_all_ISA(aircraft_CAT).arr.edr(EDR_range).Circulation_Threshold(Circulation_Threshold_Level).ogecoordinates;
               end %if

               %If the follower aircraft can't form a envelope
               %with the leader's circulation strength level

               if isempty(boundary_coordinates)
                  continue;
               end

               number_of_points = length(boundary_coordinates);

               %% Generate_Envelope, copied out for parfor loop
               % *************************************************************************
               % Wake Heading Transformation Calculation
               % Heading is defined clockwise, the transformation formula is anti-clockwise
               % A is the transformation matrix to convert the straight envelope follows along the
               % flight track, but it's still straight, the bending will be done in the
               % EWEM main script
               % *************************************************************************

               % wakeheading = deg2rad(270-heading);

               % A = [cos(wakeheading),sin(wakeheading);-sin(wakeheading), cos(wakeheading)];

               % *************************************************************************
               % Wake Starting Point Calculation
               % The wake starts a the two wingtips that distance from the aircraft
               % centerline of s*Wingspan/2
               % *************************************************************************

               x_initial = leadlongitude;

               y_initial_left = leadlatitude - SpanInitialHalf_deg; % km2deg((SpanInitial_km/2));
               y_initial_right = leadlatitude + SpanInitialHalf_deg; % km2deg((SpanInitial_km/2));

               % *************************************************************************
               % Wake General Information Calculation
               % point_B is the last point generated by the APA suite of model, it
               % connects with the starting point and forms the upper boundary of the
               % envelope. This calculates the upper boundary formula and will be used
               % later to detect the descent drop for lateral expansion
               % *************************************************************************

               point_B = boundary_coordinates(:,number_of_points-1);

               slope_upper = point_B(2)/point_B(1);

               % *************************************************************************
               % Wake Altitude Calculation
               % Simply De-normalize the normalization coordination based on the TDAWP
               % model formula,  Normalized_Descent = Vertical_Descent/b0, b0 = Wingspan*pi/4;
```

```
% ************************************************************************

z_upper_nor = boundary_coordinates(1,:)*slope_upper;
z_lower_nor = boundary_coordinates(2,:);

%z_upper is the stright line from the side view, which connects the first
%point of the envelope and the last point of the envelope.

if (Model_Mode == 0)
   z_upper = leadaltitude(j).*ones(1,number_of_points) + z_upper_nor.*SpanInitial; %m
elseif (Model_Mode == 1) %Conservative Envelope Shape, with flat top
   z_upper = leadaltitude(j).*ones(1,number_of_points); %m
end
z_lower = leadaltitude(j).*ones(1,number_of_points) + z_lower_nor.*SpanInitial; %m
% ************************************************************************
% Wake Longitudinal Length Calculation
% De-normalize the normalization coordination based on the TDAWP
% model formula, Normalized_t = t/t0; t0 = b0/w0
% The length can be calculated by distance = time*speed.
% ************************************************************************

x_add = km2deg(boundary_coordinates(1,:)*TimeInitial*groundspeed*m_to_km);%deg

% % ************************************************************************
% % Wake Lateral Width Calculation
% % De-normalize the normalization coordination based on the TDAWP
% % model formula, Normalized_t = t/t0; t0 = b0/w0
% % The length can be calculated by distance = time*speed.
% ************************************************************************

deep_length = abs(km2deg((z_upper - z_lower)*m_to_km)); %deg
Lateral_Expansion = deep_length(1:number_of_points-2);
Lateral_Expansion(number_of_points-1:number_of_points) = deep_length(number_of_points-2);

y_modified_left = y_initial_left - Lateral_Expansion;
y_modified_right = y_initial_right + Lateral_Expansion;

y_modified_right = y_modified_right(1:end-1);
y_modified_left = y_modified_left(1:end-1);
x_modified = x_initial+x_add(1:end-1);
z_upper = z_upper(1:end-1);
z_lower = z_lower(1:end-1);

% y_modified_right_inv = y_modified_right(end:-1:1);
% y_modified_left_inv = y_modified_left(end:-1:1);
% x_modified_inv = x_modified(end:-1:1);
% z_upper_inv = z_upper(end:-1:1);
% z_lower_inv = z_lower(end:-1:1);

% envelope_coordinates =
[y_modified_left,y_modified_right_inv,y_modified_left,y_modified_right_inv;x_modified,x_modified_inv,x_modified,x_modified_inv;z_upper,
z_upper_inv,z_lower,z_lower_inv];
% envelope_coordinates = envelope_coordinates';

%% Ground Effect Envelope Formation
% Different from OGE envelope, this requires both the core
% coordinates for both lateral and longitudinal movement
elseif leadaltitude(j) < OGE_boundary(EDR_range)

p = floor(leadaltitude(j)/50);

if (p==1)
   p = p+1;
elseif(p==0)
   p=p+2;
end %if (p==1)

if leadaltitude(j) >= leadaltitude(1)
```

```
                    boundary_coordinates_side =
boundary_all.boundary_all_ISA(aircraft_CAT).dep.edr(EDR_range).Circulation_Threshold(Circulation_Threshold_Level).IGE(p).side_coordina
tes;
                    boundary_coordinates_lateral =
boundary_all.boundary_all_ISA(aircraft_CAT).dep.edr(EDR_range).Circulation_Threshold(Circulation_Threshold_Level).IGE(p).lateral_coordi
nates;
                else
                    boundary_coordinates_side =
boundary_all.boundary_all_ISA(aircraft_CAT).arr.edr(EDR_range).Circulation_Threshold(Circulation_Threshold_Level).IGE(p).side_coordinat
es;
                    boundary_coordinates_lateral =
boundary_all.boundary_all_ISA(aircraft_CAT).arr.edr(EDR_range).Circulation_Threshold(Circulation_Threshold_Level).IGE(p).lateral_coordin
ates;
                end %if leadaltitude(j) >= leadaltitude(1)

                if isempty(boundary_coordinates)
                    continue
                end
                % ************************************************************************
                % Wake Starting Point Calculation
                % The wake starts a the two wingtips that distance from the aircraft
                % centerline of s*Wingspan/2
                % ************************************************************************

                x_initial = leadlongitude;

                y_initial_left = leadlatitude -  SpanInitialHalf_deg; % km2deg((SpanInitial_km/2));
                y_initial_right = leadlatitude + SpanInitialHalf_deg; % km2deg((SpanInitial_km/2));
                % wakeheading = deg2rad(270-heading);
                % number_of_points = length(boundary_coordinates_side);
                % number_of_points_lateral = length(boundary_coordinates_lateral);

                % ************************************************************************
                % Find the edge points along the wake envelop to form a
                % simple rectangle/triangle for later computation
                % ************************************************************************

                % Index_End = find(boundary_coordinates_side(1,:)==max(boundary_coordinates_side(1,:)));
                Index_Top = find(boundary_coordinates_side(2,:)==max(boundary_coordinates_side(2,:)));
                Index_Bottom = find(boundary_coordinates_side(2,:)==min(boundary_coordinates_side(2,:)));

                % Point_End = [boundary_coordinates_side(1,Index_End),boundary_coordinates_side(2,Index_End)];
                Point_Top = [boundary_coordinates_side(1,Index_Top),boundary_coordinates_side(2,Index_Top)];
                Point_Bottom = [boundary_coordinates_side(1,Index_Bottom),boundary_coordinates_side(2,Index_Bottom)];

                Slope_Top = Point_Top(2)/Point_Top(1);
                % Slope_Bottom = (Point_End(2)-Point_Bottom(2))/(Point_End(1)-Point_Bottom(1));

                % ************************************************************************
                % Find the edge points along the wake envelop lateral
                % cut to form the envelope. Since lateral movement of the vortices are the
                % same from left to right, so we only need half of it for further computation
                % ************************************************************************
                Index_Lateral_Point_Wide = find(boundary_coordinates_lateral(2,:)==max(boundary_coordinates_lateral(2,:)));
                Index_Lateral_Point_Start = find(boundary_coordinates_lateral(2,:)==min(boundary_coordinates_lateral(2,:)),2);
                Usable_Range = Index_Lateral_Point_Start:Index_Lateral_Point_Wide;
                boundary_coordinates_lateral_modified = boundary_coordinates_lateral(2,Usable_Range) -
min(boundary_coordinates_lateral(2,:)); %For upper boundary
                boundary_coordinates_lateral_modified_bottom = boundary_coordinates_lateral_modified + Point_Bottom(2); %For lower
boundary
                boundary_coordinates_side_modified_top = boundary_coordinates_lateral_modified/Slope_Top;
                % boundary_coordinates_side_modified_bottom = (boundary_coordinates_lateral_modified_bottom -
Point_Bottom(2))/Slope_Bottom + Point_Bottom(1);
                % ************************************************************************
                % Wake Envelope Calculation
                % Last_Point_Coordinate is the last point generated by the APA suite of model, it
                % connects with the starting point and forms the upper boundary of the
                % envelope. This calculates the upper boundary and read off the coordinate
                % for lower boundary
                % ************************************************************************
```

```
                % Lower_boundary_altitude = leadaltitude(j) + boundary_coordinates_side_modified_top * SpanInitial;
                % Upper_boundary_altitude = leadaltitude(j) + boundary_coordinates_side_modified_bottom* SpanInitial;
                % ************************************************************************
                % Longitude Boundary Calculation
                % De-normalize the normalization coordination based on the TDAWP
                % % model formula, Normalized_t = t/t0; t0 = b0/w0
                % % The length can be calculated by distance = time*speed.
                % ************************************************************************
                Envelope_length = km2deg(boundary_coordinates_side_modified_top*TimeInitial*groundspeed*0.001); %deg
                % % ************************************************************************
                % % Wake Lateral Width Calculation
                % % Latitude Calculation
                % % De-normalize the normalization coordination based on the TDAWP
                % % model formula, Normalized_t = t/t0; t0 = b0/w0
                % % The length can be calculated by distance = time*speed.
                % ************************************************************************
                Lateral_Expansion = abs(km2deg(boundary_coordinates_lateral_modified*SpanInitial*0.001)); %getting wider, start from the
middle
                % % ************************************************************************
                % % Wake Lateral Width Calculation
                % % Latitude Calculation
                % % De-normalize the normalization coordination based on the TDAWP
                % % model formula, Normalized_t = t/t0; t0 = b0/w0
                % % The length can be calculated by distance = time*speed.
                % ************************************************************************
                y_modified_left = y_initial_left - Lateral_Expansion;
                y_modified_right = y_initial_right + Lateral_Expansion;

                y_modified_right = y_modified_right(1:end-1);
                y_modified_left = y_modified_left(1:end-1);
                x_modified = x_initial + Envelope_length(1:end-1);
                z_upper = boundary_coordinates_lateral_modified(1:end-1);
                z_lower = boundary_coordinates_lateral_modified_bottom(1:end-1);

                % y_modified_right_inv = y_modified_right(end:-1:1);
                % y_modified_left_inv = y_modified_left(end:-1:1);
                % x_modified_inv = x_modified(end:-1:1);
                % z_upper_inv = z_upper(end:-1:1);
                % z_lower_inv = z_lower(end:-1:1);
                % envelope_coordinates =
[y_modified_left,y_modified_right_inv,y_modified_left,y_modified_right_inv;x_modified,x_modified_inv,x_modified,x_modified_inv;z_upper,
z_upper_inv,z_lower,z_lower_inv];
                % envelope_coordinates = envelope_coordinates';
            end  %if leadaltitude(j) >= OGE_boundary(EDR_range)

            %% ************************************************************************
            %  Calculate the envelope dimensions for bending purpose
            %% ************************************************************************
            number_of_points_envelope = length(z_upper);
            distLB = abs(distance(y_modified_left(1:number_of_points_envelope-1),x_modified(1:number_of_points_envelope-
1),y_modified_left(2:number_of_points_envelope),x_modified(2:number_of_points_envelope)));
            envelope_cum_distLB = cumsum(distLB);

            halfdist = abs(distance(y_modified_left(1:number_of_points_envelope-1),x_modified(1:number_of_points_envelope-
1),y_modified_right(1:number_of_points_envelope-1),x_modified(1:number_of_points_envelope-1)));

            %% ************************************************************************
            %  Start to bend the envelope
            %% ************************************************************************
            index_start = j;  %find the corresponding point along the track

            index_end   = max(index_start-assigned_num_of_points_back,1); %Last point cannot be less than the first track point

            num_of_points_back = index_start-index_end;

            if num_of_points_back>5 %Need minimum of 5 track points to construct theenvelope

               use_this_lat_to_compute_track_length  = FlightTrack(leadidx).Latitude(index_end:index_start); %[degrees](1-14)
               use_this_long_to_compute_track_length = FlightTrack(leadidx).Longitude(index_end:index_start);%[degrees]
               use_this_alt_to_compute_track_length  = FlightTrack(leadidx).Altitude(index_end:index_start);
```

```
use_this_lat_to_plot_envelope  =flipud(use_this_lat_to_compute_track_length); %[degrees] (from 14 to 1)
use_this_long_to_plot_envelope =flipud(use_this_long_to_compute_track_length);%[degrees]
use_this_alt_to_plot_envelope  =flipud(use_this_alt_to_compute_track_length);%

% Compute the distance between each sets of points along the PDARS
% trajectory, then compute the cummulative distance and compare to the
% actual envelope lenght, then select the actual number of points

compute_dist_between_each_point = distance(use_this_lat_to_compute_track_length(1:num_of_points_back),
use_this_long_to_compute_track_length(1:num_of_points_back),
use_this_lat_to_compute_track_length(2:num_of_points_back+1),use_this_long_to_compute_track_length(2:num_of_points_back+1));

altitude_drop_index = find(compute_dist_between_each_point == 0);

if ~isempty(altitude_drop_index)
   compute_dist_between_each_point(altitude_drop_index) =
km2deg(m_to_km*(use_this_alt_to_compute_track_length(altitude_drop_index) -
use_this_alt_to_compute_track_length(altitude_drop_index+1)));
end

compute_dist_between_each_point = abs(compute_dist_between_each_point);
% dist_between_each_point = compute_dist_between_each_point;
%Flip the matrix so the end of the trajectory is the beggining of the
%envelope and compute the cummulative distances
dist_between_each_point_flipped= flipud(compute_dist_between_each_point); %Flip it so it goes in the direction of the envelope-
backwords
%from 13--index start to 1--index_end

%Now compute the cummulative distance starting at pt.0 (going backwards on track 14-1)
cum_dist = cumsum(dist_between_each_point_flipped);

%First find the last point of the flight track that will be used
%When the envelope length is shorter than the
%cum_distance, the point will be the final point for
%envelope bending
final_index = find (envelope_cum_distLB(end) < cum_dist,1,'first');

if (final_index>1) %The envelope is at least longer than one of the distance of those previous track points ( 14 to 13)

   %Interpolate along the envelope centerline to find the corresponding width
   %at the available track locations

   %Only need the length until it is longer than the envelope
   cum_dist_for_envelope= cum_dist(1:final_index);
   %Calculate the difference between the track length
   %and the envelope length
   % distance_to_be_added = envelope_cum_distLB(end) - cum_dist_for_envelope(end-1);

   usable_lat = use_this_lat_to_plot_envelope (1:final_index);
   usable_long= use_this_long_to_plot_envelope(1:final_index);
   usable_alt= use_this_alt_to_plot_envelope(1:final_index);

   lat_add = interp1(cum_dist_for_envelope,usable_lat,envelope_cum_distLB(end));
   long_add = interp1(cum_dist_for_envelope,usable_long,envelope_cum_distLB(end));
   alt_add = interp1(cum_dist_for_envelope,usable_alt,envelope_cum_distLB(end));

   usable_lat = [usable_lat(1:end-1);lat_add;usable_lat(end)];
   usable_long = [usable_long(1:end-1); long_add;usable_long(end)];
   usable_alt = [usable_alt(1:end-1); alt_add;usable_alt(end)];

   cum_dist_for_envelope = [cum_dist_for_envelope(1:end-1);envelope_cum_distLB(end);cum_dist_for_envelope(end)];
   %Compute the azimuths between all needed points (will be less than 80)
   azimuth_between_each_point = azimuth(usable_lat(2:final_index+1), usable_long(2:final_index+1),
usable_lat(1:final_index),usable_long(1:final_index));

   projected_point_coord_LB = zeros(final_index,3);
   projected_point_coord_RB = zeros(final_index,3);
```

```
Index = zeros(final_index,1);

%Interpolate along the envelope width to find the corresponding value
for m = 1:final_index
   %LOWER BOUND (LB)
   %The indeces will be from 1 to 5
   Index(m) = find(cum_dist_for_envelope(m) <= envelope_cum_distLB,1,'first'); %index - 29, until the 29th point in the
envelope, it starts to divert from the path. (The first point need to be correct)
   %                    if Index ==0
   %                    end %if Index==0

   %                    Distance_to_add (m)= cum_dist_for_envelope(m)*(halfdist(Index)-
halfdist(1))/envelope_cum_distLB(Index);
   %                    Distance_to_project(m) =Distance_to_add (m) +  halfdist(1);
   %
   %                    %reckon==Point at specified azimuth, range on sphere or ellipsoid
   %                    [latoutL,lonoutL]= reckon(usable_lat(m), usable_long(m),Distance_to_project(m),
(azimuth_between_each_point(m)-90));
   %                    projected_point_coord_LB(m,1)= (latoutL);
   %                    projected_point_coord_LB(m,2)= (lonoutL);
   %                    [latoutR,lonoutR]= reckon(usable_lat(m), usable_long(m),Distance_to_project(m),
(azimuth_between_each_point(m)+90));
   %                    projected_point_coord_RB(m,1)= (latoutR);
   %                    projected_point_coord_RB(m,2)= (lonoutR);
end  %for m=1:(final_index-1)

Distance_to_add     = cum_dist_for_envelope(1:final_index)'.*(halfdist(Index)-halfdist(1))./envelope_cum_distLB(Index);
Distance_to_project = Distance_to_add + halfdist(1);
%reckon==Point at specified azimuth, range on sphere or ellipsoid
projected_point_coord_LB(:,1:2) = reckon(usable_lat(1:final_index), usable_long(1:final_index),
Distance_to_project(1:final_index)', (azimuth_between_each_point(1:final_index)-90));
projected_point_coord_RB(:,1:2) = reckon(usable_lat(1:final_index), usable_long(1:final_index),
Distance_to_project(1:final_index)', (azimuth_between_each_point(1:final_index)+90));


%% ************************************************************************
% Calculate the envelope expansion under crosswind
%% ************************************************************************

envelope_cumdist = deg2km(cum_dist_for_envelope(1:final_index))*1000;%m
time_elapsed_since_generate = [envelope_cumdist/groundspeed]; %s, in one column

% Calculate the crosswind strength from the input
if length(crosswind_all) ==2 %Distribution

   crosswind_kts = crosswind_all(1) +2*crosswind_all(2)/3;% knots,mean+2*std/3, expand the envelope with 2sigma
   crosswind = crosswind_kts *0.514*0.001; %in km/s
   distance_crosswind = km2deg(crosswind.*time_elapsed_since_generate);
   projected_point_coord_RB(:,1:2)=  reckon(projected_point_coord_RB(:,1),projected_point_coord_RB(:,2),
distance_crosswind, (azimuth_between_each_point-90));
   projected_point_coord_LB(:,1:2) =  reckon(projected_point_coord_LB(:,1),projected_point_coord_LB(:,2),
distance_crosswind, (azimuth_between_each_point+90));

elseif length(crosswind_all) ==3 %Fixed Input

   wind_speed = crosswind_all(1); %knots
   wind_direction = crosswind_all(3); %0-360
   crosswind_kts = wind_speed*sin((azimuth_between_each_point-wind_direction)/360*2*pi);
   crosswind = crosswind_kts *0.514*0.001; %km/s
   distance_crosswind = km2deg(crosswind.*time_elapsed_since_generate);
   projected_point_coord_LB(:,1:2) =
reckon(projected_point_coord_LB(:,1),projected_point_coord_LB(:,2),distance_crosswind, (azimuth_between_each_point-90));
   projected_point_coord_RB(:,1:2) =
reckon(projected_point_coord_RB(:,1),projected_point_coord_RB(:,2),distance_crosswind, (azimuth_between_each_point-90));

elseif length(crosswind_all) == 4 %ASOS imported data

   wind_time_all = crosswind_all(:,1);
   wind_speed_all = crosswind_all(:,2);
   wind_direction_all = crosswind_all(:,4);
```

```
                    if strcmp(Track_Type, 'PDARS') == 1 %PDARS data time tag need to be converted
                       Number_Of_Days = leadtime / (24 * 3600); % Number of days since 01/01/1970 for track point
                       RecordTime = Base_Date_Number + Number_Of_Days; %in sec
                      %Actual_Time_Formatted = datestr(Actual_Date_In_Days, 'HH:MM'); % Time formatted HH:MM:SS
                      % RecordTime = str2num([Actual_Hours_Formatted(1:2),Actual_Hours_Formatted(4:5)]);%[HH,MM]
                    else
                       RecordTime = leadtime ;
                    end

                    tmp = abs(wind_time_all-RecordTime);
                    [index_wind,index_wind] = min(tmp); %index of closest value
                    wind_speed = wind_speed_all(index_wind); %wind speed at the closest time point
                    wind_direction = wind_direction_all(index_wind); %%wind direction at the closest time point

                    crosswind_kts = wind_speed*sin((azimuth_between_each_point-wind_direction)/360*2*pi);
                    crosswind = crosswind_kts *0.514*0.001; %km/s
                    distance_crosswind = km2deg(crosswind.*time_elapsed_since_generate);
                    projected_point_coord_LB(:,1:2) =
reckon(projected_point_coord_LB(:,1),projected_point_coord_LB(:,2),distance_crosswind, (azimuth_between_each_point-90));
                    projected_point_coord_RB(:,1:2) =
reckon(projected_point_coord_RB(:,1),projected_point_coord_RB(:,2),distance_crosswind, (azimuth_between_each_point-90));
                 end
                 % save a vector to plot it for LB
                 projected_coord_RLB = projected_point_coord_RB;
                 projected_coord_LLB = projected_point_coord_LB;
                 % save a vector to plot it for UB
                 projected_coord_RUB = projected_point_coord_RB;
                 projected_coord_LUB = projected_point_coord_LB;



                 %Interpolate the altitudes. We need the cummulative distance starting from
                 %0
                 Track_altitude= usable_alt(1:final_index);
                 envelope_cum_distLB_rev = [0,envelope_cum_distLB];


                 record_length= length(cum_dist_for_envelope)-1; %we want 1 less record cause it will be NAN

                 compute_altitudeLB =  interp1q(envelope_cum_distLB_rev' ,z_lower',cum_dist_for_envelope);
                 %                 altitude_to_subtract_LB= compute_altitudeLB(1,1);
                 altitude_to_subtract_LB = leadaltitude(j);
                 altitudeLB= compute_altitudeLB(1:record_length, :);
                 altitudeLB_final= Track_altitude + altitudeLB -altitude_to_subtract_LB;

                 compute_altitudeUB =  interp1q(envelope_cum_distLB_rev' ,z_upper',cum_dist_for_envelope);
                 %                 altitude_to_subtract_UB = compute_altitudeUB(1,1);
                 altitude_to_subtract_UB = leadaltitude(j);
                 altitudeUB= compute_altitudeUB(1:record_length,:);
                 altitudeUB_final= altitudeUB+ Track_altitude-altitude_to_subtract_UB;
                 altitudeUB_final(1,1)=Track_altitude(1,1);

                 projected_coord_LLB (:,3) = altitudeLB_final;
                 projected_coord_RLB (:,3) = altitudeLB_final;
                 projected_coord_LUB(:,3) = altitudeUB_final;
                 projected_coord_RUB(:,3) = altitudeUB_final;

                 projected_coord_RUB(record_length,3)= projected_coord_RLB(record_length,3);
                 projected_coord_LUB(record_length,3)= projected_coord_LLB(record_length,3);

                 projected_coord_RLB_flipped= flipud(projected_coord_RLB);
                 Lower_surface= [projected_coord_LLB
                    projected_coord_RLB_flipped];
                 projected_coord_RB_flipped= flipud(projected_coord_RUB);
                 Upper_surface= [projected_coord_LUB
                    projected_coord_RB_flipped];
                 bentenvelope_coordinates= [Lower_surface
                    Upper_surface];
```

```
                    aircraftposition_corrected = double([followlatitude,followlongitude,followaltitude]);
                    [B,b] = vert2lcon(bentenvelope_coordinates);

                    wakepiercing = all(B*(aircraftposition_corrected')<=b);

                    if wakepiercing
                       wakepiercing_sum(i).filteredleadrecord = [filteredleadrecord(i),j];
                       wakepiercing_sum(i).filteredfollowerrecord = [filteredfollowerrecord(i),index_follower(k)];
                       wakepiercing_sum(i).envelope = bentenvelope_coordinates';
                       wakepiercing_sum(i).envelopelength = envelope_cum_distLB;
                       wakepiercing_sum(i).InitialCirculation = CirculationInitial;
                    end %wakepiercing
                 end %if (final_index>1)
              end%if num_of_points_back>5
           end %if CirculationInitial > Circulation_Threshold(aircraft_CAT_follower)
        end  %for k = 1:number_of_close_points
     end % for j = 2: number_of_lpoints-1
  end %for ~isempty(actype_lidx) && ~isempty(actype_fidx)
end %for i = 1:number_of_pairs
% toc;

save ([Project_Dir, 'wakepiercing_', FlightTracks_Name, '.mat'], 'wakepiercing_sum');
ppm.delete()

% matlabpool close force;

return;
```

FlightList_Generator.m

```
%% This will generate the flight list for your flight track
% Flight list is one of the essential input for EWEM
function Flightlist_Generator(Project_Dir, FlightTracks_Name,Track_Type)

% clc;
% clear all;
% close all;

% Install_Dir = pwd;
% Tracks_Dir  = [pwd '\Output\'];
% Project_Dir = [pwd '\Output\'];
% FlightTracks_Name = 'Parsed_PDARS_SCT_20080618';

if strcmp(Track_Type, 'ACES') == 1
   load ([Project_Dir, '\window_count.mat']);
   Number_Of_Files = window_count;
else
   Number_Of_Files = 1;
end

for f = 1:Number_Of_Files

   disp(['Analyzing Step 1/2: Number of files processed: ', num2str(f), '/', num2str(Number_Of_Files)]);

   if strcmp(Track_Type, 'ACES') == 1
      load([Project_Dir,'\', FlightTracks_Name, '_str_', num2str(f), '.mat']); %load the flight track with constructed envelope
   else
      load([Project_Dir, '/Parsed_', FlightTracks_Name, '.mat']); %load the flight track with constructed envelope
   end

   number_of_tracks = length(FlightTrack);

   recordnum_list = zeros(number_of_tracks,1);
   for i = 1:number_of_tracks
      recordnum_list(i) = FlightTrack(i).Recordnum;
   end

   if strcmp(Track_Type, 'ACES') == 1
      fid = fopen([Project_Dir, '\', FlightTracks_Name, '_str_', num2str(f), '_FlightList.txt'], 'w');
   else
```

```
        fid = fopen([Project_Dir, '/', FlightTracks_Name, '_FlightList.txt'], 'w');
    end

    fprintf(fid, '%d, \n', recordnum_list);
    fclose(fid);

    clear FlightTrack;

end % for f = 1:Number_Of_Files

fclose('all');

return;
```

FlightTrack_Import.m

```
% This function imports the flight tracks for EWEM
function FlightTrack_Import(Install_Dir, Project_Dir, Track_Name, Track_FileName, Track_Type, NumOfThreads)

% ------------
% Testing
% ------------
% clc;
% clear all;
% close all;
% %
% % Install_Dir    = 'E:\WEM_SVN\WEM_GUI\trunk\WEM_GUI\DATA\EWEM';
% % Project_Dir    = 'E:\WEM_SVN\FlightAware_East_Coast\WEM Project\EWEM\Input\Tracks';
% % Track_Name     = 'FlightAware_Test'; % 'FlightAware_Test'; % 'ACES_70Kflights_VT_DynamicTimeWindow'; %
'PDARS_SCT_20080618'; % 'ACES_70Kflights_VT';
% % Track_FileName = 'D:\WEM_SVN\WEM_MODEL\iff_ssSCT_20080618_080025_86374.csv'; % PDARS File
% % Track_FileName = 'D:\WEM Project\WakeWISE\Input\Tracks\WTMD_SingleRunway_ShortRoutes_Flights.csv'; % MEM File
% % Track_FileName = 'D:\WEM_SVN\WEM_MODEL\aircraftstate_70Kflights_VT_timesorted.csv'; % ACES File
% % Track_FileName = 'E:\WEM_SVN\FlightAware_East_Coast\FlightAware_Cape_Canaveral_Flights.csv'; % FlightAware File
% % Track_Type     = 'FlightAware'; % PDARS, TARGET, ACES, FlightAware
% % NumOfThreads   = 8;

% ------------
% Create Directory
% ------------
Project_Dir = [Project_Dir, '\', Track_Name];
mkdir(Project_Dir);

% ------------
% Individual Parsers
% ------------
if strcmp(Track_Type, 'PDARS') == 1
    PDARS_Parser_Swol(Project_Dir, Track_FileName, Track_Name);
    PDARS_Parser_Structure(Project_Dir, Track_Name);
elseif strcmp(Track_Type, 'TARGET') == 1
    Track_FileName = Track_FileName(1:end-12);
    MEM_Parser(Project_Dir, Track_FileName, Track_Name);
elseif strcmp(Track_Type, 'ACES') == 1
    ACES_Parse_State_Data_Rolling_Window(Project_Dir, Track_FileName, Track_Name);
    ACES_Parser(Project_Dir, Track_Name);
elseif strcmp(Track_Type, 'FlightAware') == 1
    Track_FileName = Track_FileName(1:end-12);
    Flightaware_Parser(Project_Dir, Track_FileName, Track_Name)
end

% ------------
% Posprocessing
% ------------
Flightlist_Generator(Project_Dir, Track_Name,Track_Type);
Pair_Filter(Install_Dir, Project_Dir, Track_Name, NumOfThreads,Track_Type);

return;
```

Google_Earth_Visualization.m

```
function Google_Earth_Visualization(Tracks_Dir, Project_Dir, FlightTracks_Name, Track_Type)

%Google Earth Visulization for Wake Encounter Model Result
%To plot the wake in a series of rectangle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

% %% This script will plot conflicts detected by EWEM
% Tracks_Dir  = 'C:\WEM Project\EWEM\Input\Tracks\PDARS_SCT_20080618'; % [pwd '/Output'];
% Project_Dir = 'C:\WEM Project\EWEM\Output\Test\EWEM - Case 1'; % [pwd '/Output'];
% FlightTracks_Name = 'PDARS_SCT_20080618';

% Tracks_Dir  = 'C:\WEM Project\EWEM\Input\Tracks\ACES_70Kflights_VT_1HR'; % [pwd,'/Output'];
% Project_Dir = 'C:\WEM Project\EWEM\Output\Test\ACES - Case 1'; % [pwd,'/Output'];
% FlightTracks_Name = 'ACES_70Kflights_VT_1HR';
% Track_Type     = 'ACES'; % PDARS, MEM, ACES

% --------------------------------
% Open Output Files
% --------------------------------
% Open Summary File
fid_summary = fopen([Project_Dir, 'ConflictSummary_', FlightTracks_Name, '.txt'], 'w');
fprintf(fid_summary, '%s\n', 'Conflict ID,Lead Flight ID,Lead Aircraft Type,Lead Latitude (deg),Lead Longitude (deg), Lead Altitude
(m),Follower Flight ID,Follower Aircraft Type,Follower Latitude (deg),Follower Longitude (deg), Follower Altitude (m)');

% Open KML file
kml_filename = [Project_Dir,'Google_Earth_Visualization_Encounters.kml'];
Output_File_ID = fopen(kml_filename,'w'); %filePointer

% --------------------------------
% Create Header of KML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '<?xml version="1.0" encoding="UTF-8"?>');
fprintf(Output_File_ID, '%s\n', '<kml xmlns="http://earth.google.com/kml/2.1">');
fprintf(Output_File_ID, '%s\n', '   <Document>');

fprintf(Output_File_ID, '%s\n', '          <Style id="Envelope">'); %RED
fprintf(Output_File_ID, '%s\n', '                 <LineStyle>');
fprintf(Output_File_ID, '%s\n', '                 <color>#ff0000ff</color>');
fprintf(Output_File_ID, '%s\n', '                 <width>4</width>');
fprintf(Output_File_ID, '%s\n', '                 </LineStyle>');
%     fprintf(Output_File_ID, '%s\n', '                    <PolyStyle>');
%     fprintf(Output_File_ID, '%s\n', '                    <color>7dff0000</color>');
%     fprintf(Output_File_ID, '%s\n', '                    </PolyStyle>');
fprintf(Output_File_ID, '%s\n', '          </Style>');
fprintf(Output_File_ID, '%s\n', '          <Style id="Leader">'); %in GREEN
fprintf(Output_File_ID, '%s\n', '                 <LineStyle>');
fprintf(Output_File_ID, '%s\n', '                 <color>#7f00ff00</color>');
fprintf(Output_File_ID, '%s\n', '                 <width>4</width>');
fprintf(Output_File_ID, '%s\n', '                 </LineStyle>');
fprintf(Output_File_ID, '%s\n', '          </Style>');

fprintf(Output_File_ID, '%s\n', '          <Style id="Follower">'); %in YELLOW
fprintf(Output_File_ID, '%s\n', '                 <LineStyle>');
fprintf(Output_File_ID, '%s\n', '                 <color>#7f00ffff</color>');
fprintf(Output_File_ID, '%s\n', '                 <width>4</width>');
fprintf(Output_File_ID, '%s\n', '                 </LineStyle>');
fprintf(Output_File_ID, '%s\n', '          </Style>');

fprintf(Output_File_ID, '%s\n', '          <Style id="LeaderIcon">'); %GREEN
fprintf(Output_File_ID, '%s\n', '                 <IconStyle>');
fprintf(Output_File_ID, '%s\n', '                 <Icon>');
fprintf(Output_File_ID, '%s\n', '                 <href>https://googledrive.com/host/0B-
KVUg7rd7N9TFNINmFNUUx3d0k/LETTER_L_GREEN.png</href>');
fprintf(Output_File_ID, '%s\n', '                        </Icon>');
fprintf(Output_File_ID, '%s\n', '                 </IconStyle>');
fprintf(Output_File_ID, '%s\n', '          </Style>');
```

```
fprintf(Output_File_ID, '%s\n', '                <Style id="FollowerIcon">'); %YELLO
fprintf(Output_File_ID, '%s\n', '                        <IconStyle>');
fprintf(Output_File_ID, '%s\n', '                        <Icon>');
fprintf(Output_File_ID, '%s\n', '                        <href>https://googledrive.com/host/0B-
KVUg7rd7N9TFNINmFNUUx3d0k/LETTER_F_YELLOW.png</href>');
fprintf(Output_File_ID, '%s\n', '                                </Icon>');
fprintf(Output_File_ID, '%s\n', '                        </IconStyle>');
fprintf(Output_File_ID, '%s\n', '                </Style>');

% --------------------------------
% Setup loop for ACES
% --------------------------------
if strcmp(Track_Type, 'ACES') == 1
   load ([Tracks_Dir, 'window_count.mat']);
   Number_Of_Files = window_count;
else
   Number_Of_Files = 1;
end

for f = 1:Number_Of_Files

   disp(['Visualization: ', num2str(f), '/', num2str(Number_Of_Files)]);

   % --------------------------------
   % Loading
   % --------------------------------
   if strcmp(Track_Type, 'ACES') == 1
      load([Tracks_Dir, FlightTracks_Name, '_str_', num2str(f), '.mat']); %load the flight track with constructed envelope
   else
      load([Tracks_Dir, 'Parsed_', FlightTracks_Name, '.mat']); %load the flight track with constructed envelope
   end

   if strcmp(Track_Type, 'ACES') == 1
      WP_FileName = [Project_Dir, 'wakepiercing_', FlightTracks_Name, '_str_', num2str(f), '.mat'];
      FL_FileName = [Tracks_Dir, FlightTracks_Name, '_str_', num2str(f), '_FlightList.txt'];
   else
      WP_FileName = [Project_Dir, 'wakepiercing_', FlightTracks_Name, '.mat'];
      FL_FileName = [Tracks_Dir, FlightTracks_Name, '_FlightList.txt'];
   end

   % Skip if no EWEM output
   if exist(WP_FileName, 'file') == 0
      continue;
   end
   Detected_Encounters = load(WP_FileName);
   fid = fopen(FL_FileName, 'r');

   flightlist = textscan(fid, '%f', 'delimiter', ',');
   flightlist = flightlist{1};
   fclose(fid);

   number_of_encounters = length(Detected_Encounters.wakepiercing_sum);

   % --------------------------------
   % Create Body of KML File
   % --------------------------------
   % Encounter_Counter = 0;
   for i = 1: number_of_encounters

      % Skip to next record if no encounter recorded
      if isempty(Detected_Encounters.wakepiercing_sum(i).filteredleadrecord) == 1
         continue;
      end
      %     Encounter_Counter = Encounter_Counter + 1;

      envelope_coordinates = Detected_Encounters.wakepiercing_sum(i).envelope;

      leadidx    =  find(flightlist==Detected_Encounters.wakepiercing_sum(i).filteredleadrecord(1)); %find the correspoinding structure file for
the leading aircraft
```

```
    followeridx = find(flightlist==Detected_Encounters.wakepiercing_sum(i).filteredfollowerrecord(1)); %find all the followers for the same
leading aircraft

    number_of_lpoints = length(FlightTrack(leadidx).Latitude);
    number_of_fpoints = length(FlightTrack(followeridx).Latitude);

    id_Leader          = FlightTrack(leadidx).Recordnum;
    description_Leader = ['Conflict ID: ', num2str(id_Leader), ' - ', FlightTrack(leadidx).actype];
    id_follower        = FlightTrack(followeridx).Recordnum;
    description_follower = ['Conflict ID: ', num2str(id_follower), ' - ', FlightTrack(followeridx).actype];

    TracksName = sprintf('%d-%d', id_Leader, id_follower);

    fprintf(Output_File_ID, '%s\n', ['          <Folder>']);
    %    if Encounter_Counter == 1
    %       fprintf(Output_File_ID, '%s\n', ['     <visibility>1</visibility>']);
    %    else
    %       fprintf(Output_File_ID, '%s\n', ['     <visibility>0</visibility>']);
    %    end
    fprintf(Output_File_ID, '%s\n', ['                    <name>', TracksName, '</name>']);
    %    fprintf(Output_File_ID, '%s\n', ['                    <description>', TracksDescriptions, '</description>']);

    % ---------------------------------
    % Plot Leader Aircraft Flight Track
    % ---------------------------------
    fprintf(Output_File_ID, '%s\n', '                    <Placemark>');
    fprintf(Output_File_ID, '%s\n', ['                          <name>', 'Leader', '</name>']);
    fprintf(Output_File_ID, '%s\n', ['                          <description>', description_Leader,'</description>']);
    fprintf(Output_File_ID, '%s\n', '                          <styleUrl>#Leader</styleUrl>');
    fprintf(Output_File_ID, '%s\n', '                          <LineString>');
    fprintf(Output_File_ID, '%s\n', '                          <altitudeMode>absolute</altitudeMode>');
    fprintf(Output_File_ID, '%s\n', '                       <coordinates>');
    % Write coordinate

    for j= 1:number_of_lpoints
       latitude  = num2str(FlightTrack(leadidx).Latitude(j)); % degrees
       longitude = num2str(FlightTrack(leadidx).Longitude(j)); % degrees
       altitude  = num2str(FlightTrack(leadidx).Altitude(j)); % in meters
       fprintf(Output_File_ID, '%s\n', ['                                                ', longitude, ',', latitude, ',', altitude]);
    end
    % Write footer of flight
    fprintf(Output_File_ID, '%s\n', '                                          </coordinates>');
    fprintf(Output_File_ID, '%s\n', '                          </LineString>');
    fprintf(Output_File_ID, '%s\n', '                    </Placemark>');

    %    ---------------------------------
    %    Plot Follower Aircraft Flight Track
    %    ---------------------------------


    %    opsType_follower = 'Follower';

    fprintf(Output_File_ID, '%s\n', '                    <Placemark>');
    fprintf(Output_File_ID, '%s\n', ['                          <name>', 'Follower', '</name>']);
    fprintf(Output_File_ID, '%s\n', ['                          <description>', description_follower,'</description>']);
    fprintf(Output_File_ID, '%s\n', '                          <styleUrl>#Follower</styleUrl>');
    fprintf(Output_File_ID, '%s\n', '                          <LineString>');
    fprintf(Output_File_ID, '%s\n', '                          <altitudeMode>absolute</altitudeMode>');
    fprintf(Output_File_ID, '%s\n', '                       <coordinates>');
    % Write coordinate

    for k= 1:number_of_fpoints
       latitude_follower  = num2str(FlightTrack(followeridx).Latitude(k)); % degrees
       longitude_follower = num2str(FlightTrack(followeridx).Longitude(k)); % degrees
       altitude_follower  = num2str(FlightTrack(followeridx).Altitude(k)); % in meters
       fprintf(Output_File_ID, '%s\n', ['                                                ', longitude_follower, ',', latitude_follower, ',',
altitude_follower]);
    end
    % Write footer of flight
    fprintf(Output_File_ID, '%s\n', '                                          </coordinates>');
```

```
    fprintf(Output_File_ID, '%s\n', '                                </LineString>');
    fprintf(Output_File_ID, '%s\n', '                        </Placemark>');


    % --------------------------------
    % Plot Wake Envelope where the encounter occurs
    % --------------------------------

    [~,b] = size(envelope_coordinates);
    number_of_envelope_points = b/4;
    Coordinate_a = zeros(number_of_envelope_points,3);
    Coordinate_b = zeros(number_of_envelope_points,3);
    Coordinate_c = zeros(number_of_envelope_points,3);
    Coordinate_d = zeros(number_of_envelope_points,3);

    for m = 1: number_of_envelope_points

        Coordinate_a(m,:) = [envelope_coordinates(1,m),envelope_coordinates(2,m),envelope_coordinates(3,m)];
        Coordinate_b(m,:) = [envelope_coordinates(1,2*number_of_envelope_points-
m+1),envelope_coordinates(2,2*number_of_envelope_points-m+1),envelope_coordinates(3,2*number_of_envelope_points-m+1)];
        Coordinate_c(m,:) =
[envelope_coordinates(1,2*number_of_envelope_points+m),envelope_coordinates(2,2*number_of_envelope_points+m),envelope_coordinates(3,
2*number_of_envelope_points+m)];
        Coordinate_d(m,:) = [envelope_coordinates(1,4*number_of_envelope_points-
m+1),envelope_coordinates(2,4*number_of_envelope_points-m+1),envelope_coordinates(3,4*number_of_envelope_points-m+1)];

        fprintf(Output_File_ID, '%s\n', '                        <Placemark>');
        fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#Envelope</styleUrl>');
        fprintf(Output_File_ID, '%s\n', '                                <Polygon>');
        fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
        fprintf(Output_File_ID, '%s\n', '                                <outerBoundaryIs>');
        fprintf(Output_File_ID, '%s\n', '                                <LinearRing>');
        fprintf(Output_File_ID, '%s\n', '                <coordinates>');
        fprintf(Output_File_ID, '%s\n', '                                                                ',
[num2str(Coordinate_a(2)),',',num2str(Coordinate_a(1)),',',num2str(Coordinate_a(3))]);
        fprintf(Output_File_ID, '%s\n', '                                                                ',
[num2str(Coordinate_b(2)),',',num2str(Coordinate_b(1)),',',num2str(Coordinate_b(3))]);
        fprintf(Output_File_ID, '%s\n', '                                                                ',
[num2str(Coordinate_d(2)),',',num2str(Coordinate_d(1)),',',num2str(Coordinate_d(3))]);
        fprintf(Output_File_ID, '%s\n', '                                                                ',
[num2str(Coordinate_c(2)),',',num2str(Coordinate_c(1)),',',num2str(Coordinate_c(3))]);
        fprintf(Output_File_ID, '%s\n', '                                                                ',
[num2str(Coordinate_a(2)),',',num2str(Coordinate_a(1)),',',num2str(Coordinate_a(3))]);
        fprintf(Output_File_ID, '%s\n', '                                    </coordinates>');
        fprintf(Output_File_ID, '%s\n', '                                </LinearRing>');
        fprintf(Output_File_ID, '%s\n', '                                </outerBoundaryIs>');
        fprintf(Output_File_ID, '%s\n', '                                </Polygon>')   ;
        fprintf(Output_File_ID, '%s\n', '                        </Placemark>');

    end %for m = 1: number_of_points

    fprintf(Output_File_ID, '%s\n', '                        <Placemark>');
    fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#Envelope</styleUrl>');
    fprintf(Output_File_ID, '%s\n', '                                <LineString>');
    fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
    fprintf(Output_File_ID, '%s\n', '                    <coordinates>');
    % Write coordinate

    for m = 1:number_of_envelope_points
        fprintf(Output_File_ID, '%s\n', '
            ',[num2str(Coordinate_a(m,2)),',',num2str(Coordinate_a(m,1)),',',num2str(Coordinate_a(m,3))]);
    end
    % Write footer of flight
    fprintf(Output_File_ID, '%s\n', '                                                    </coordinates>');
    fprintf(Output_File_ID, '%s\n', '                                </LineString>');
    fprintf(Output_File_ID, '%s\n', '                        </Placemark>');

    fprintf(Output_File_ID, '%s\n', '                        <Placemark>');
    fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#Envelope</styleUrl>');
    fprintf(Output_File_ID, '%s\n', '                                <LineString>');
```

```
fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
fprintf(Output_File_ID, '%s\n', '                            <coordinates>');
% Write coordinate

for m = 1:number_of_envelope_points
   fprintf(Output_File_ID, '%s\n', '
        ',[num2str(Coordinate_b(m,2)),',',num2str(Coordinate_b(m,1)),',',num2str(Coordinate_b(m,3))]);
end
% Write footer of flight
fprintf(Output_File_ID, '%s\n', '                                    </coordinates>');
fprintf(Output_File_ID, '%s\n', '                                </LineString>');
fprintf(Output_File_ID, '%s\n', '                            </Placemark>');

fprintf(Output_File_ID, '%s\n', '                            <Placemark>');
fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#Envelope</styleUrl>');
fprintf(Output_File_ID, '%s\n', '                                <LineString>');
fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
fprintf(Output_File_ID, '%s\n', '                              <coordinates>');
% Write coordinate

for m = 1:number_of_envelope_points
   fprintf(Output_File_ID, '%s\n', '
        ',[num2str(Coordinate_c(m,2)),',',num2str(Coordinate_c(m,1)),',',num2str(Coordinate_c(m,3))]);
end
% Write footer of flight
fprintf(Output_File_ID, '%s\n', '                                    </coordinates>');
fprintf(Output_File_ID, '%s\n', '                                </LineString>');
fprintf(Output_File_ID, '%s\n', '                            </Placemark>');

fprintf(Output_File_ID, '%s\n', '                            <Placemark>');
fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#Envelope</styleUrl>');
fprintf(Output_File_ID, '%s\n', '                                <LineString>');
fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
fprintf(Output_File_ID, '%s\n', '                              <coordinates>');
% Write coordinate

for m = 1:number_of_envelope_points
   fprintf(Output_File_ID, '%s\n', '
        ',[num2str(Coordinate_d(m,2)),',',num2str(Coordinate_d(m,1)),',',num2str(Coordinate_d(m,3))]);
end
% Write footer of flight
fprintf(Output_File_ID, '%s\n', '                                    </coordinates>');
fprintf(Output_File_ID, '%s\n', '                                </LineString>');
fprintf(Output_File_ID, '%s\n', '                            </Placemark>');

j = Detected_Encounters.wakepiercing_sum(i).filteredleadrecord(2);

n = Detected_Encounters.wakepiercing_sum(i).filteredfollowerrecord(2);

% --------------------------------
% Insert Icon for Leader and the Follower Position when the encounter
% happens
% --------------------------------

fprintf(Output_File_ID, '%s\n', '                            <Placemark>');
fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#LeaderIcon</styleUrl>');
fprintf(Output_File_ID, '%s\n', '                                <Point>');
fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
fprintf(Output_File_ID, '%s\n', '                              <coordinates>');
% Write coordinate

latitude_lead  = num2str(FlightTrack(leadidx).Latitude(j)); % degrees
longitude_lead = num2str(FlightTrack(leadidx).Longitude(j)); % degrees
altitude_lead  = num2str(FlightTrack(leadidx).Altitude(j)); % in meters
fprintf(Output_File_ID, '%s\n', ['                                                    ', longitude_lead, ',', latitude_lead, ',', altitude_lead]);

% Write footer of flight
fprintf(Output_File_ID, '%s\n', '                                    </coordinates>');
fprintf(Output_File_ID, '%s\n', '                                </Point>');
fprintf(Output_File_ID, '%s\n', '                            </Placemark>');
```

```
        fprintf(Output_File_ID, '%s\n', '                        <Placemark>');
        fprintf(Output_File_ID, '%s\n', '                                <styleUrl>#FollowerIcon</styleUrl>');
        fprintf(Output_File_ID, '%s\n', '                                <Point>');
        fprintf(Output_File_ID, '%s\n', '                                <altitudeMode>absolute</altitudeMode>');
        fprintf(Output_File_ID, '%s\n', '                           <coordinates>');
        % Write coordinate

        latitude_follower  = num2str(FlightTrack(followeridx).Latitude(n)); % degrees
        longitude_follower = num2str(FlightTrack(followeridx).Longitude(n)); % degrees
        altitude_follower  = num2str(FlightTrack(followeridx).Altitude(n)); % in meters
        fprintf(Output_File_ID, '%s\n', ['                                                        ', longitude_follower, ',', latitude_follower, ',',
altitude_follower]);

        % Write footer of flight
        fprintf(Output_File_ID, '%s\n', '                                        </coordinates>');
        fprintf(Output_File_ID, '%s\n', '                                </Point>');
        fprintf(Output_File_ID, '%s\n', '                        </Placemark>');

        % Close Folder
        fprintf(Output_File_ID, '%s\n', ['          </Folder>']);

        % Summary file
        fprintf(fid_summary, '%.0f,%.0f,%s,%s,%s,%s,%.0f,%s,%s,%s,%s\n', i, id_Leader, description_Leader, latitude_lead, longitude_lead,
altitude_lead, id_follower, description_follower, latitude_follower, longitude_follower, altitude_follower);

    end  %i = 1: number_of_encounters

end % for f = 1:Number_Of_Files

% --------------------------------
% Create Footer of KML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '   </Document>');
fprintf(Output_File_ID, '%s\n', '</kml>');

fclose(Output_File_ID);

fclose(fid_summary);
fclose('all');

return;
```

MEM_Parser.m

```
function MEM_Parser(Project_Dir, Track_FileName, FlightTracks_Name)
% ************************************************************************
% This M-file parses the MEM data provided by CSSI
% The original sample data from CSSI is as shown:
% 10001, 0:00:45.0,35.02388,-89.98689,3.21,0,359
% The data is been saved to Tab separated txt file and erased all the
% commas and became:
% 10001 0 00 45.0 35.02388 -89.98689 3.21 0,359
% Date: Feb 4, 2013
% ************************************************************************
% clear all;
% close all;
% clc;

% FlightTracks_Name = 'WTMD_SingleRunway_ShortRoutes';
% Tracks_Dir  = 'C:\WEM Project\WakeWISE\Input\Tracks';
% Project_Dir = 'C:\WEM Project\EWEM\Input\Tracks';

fid1 = fopen([Track_FileName, '_Flights.csv'], 'r');
flightlist = textscan(fid1, '%f%s%s%s%s', 'delimiter', ',');
%Field 1: Flight record number
%Field 2: Flight ID
%Field 3: Aircraft Type
```

```
flightrecord = flightlist{1};
actype_list = flightlist{3};

fid2 = fopen([Track_FileName, '_Tracks.csv'],'r');
flighttrack = textscan(fid2, '%d%s%f%f%f%d%d','delimiter',',');
%Field 1: Flight record number
%Field 2: Flight record time
%Field 3: Flight Latitude %degree
%Field 4: Flight Longitude %degree
%Field 5: Flight GroudSpeed (m/s)
%Field 6: Flight Altitude (fl)
%Field 7: Flight Heading

FlightNumber = flighttrack{1};
FlightTime = datevec(datenum(flighttrack{2})); %stored in vectors with 6 columns in yyyy,mm,dd,hh,mm,ss
FlightTime_HM = [FlightTime(:,4),FlightTime(:,5)];
%%Put hh and mm into one cell, consistent with future wind data
b =sprintf('%d%d*', FlightTime_HM(:, 1:2).');
FlightTime_Formatted = sscanf(b, '%d*');
FlightTime_sec = FlightTime(:,4)*3600+FlightTime(:,5)*60+FlightTime(:,6);
FlightLatitude = flighttrack{3};
FlightLongitude = flighttrack{4};
FlightSpeed = flighttrack{6}; %knots
FlightAltitude = flighttrack{5}*100*0.3048; %FL->ft->m
FlightHeading = flighttrack{7};

FlightNumber_List = unique(FlightNumber);

%Save different flight information separately in structure file
%FlightTrack(i).Flight.[actype,Time,Latitude,Longitude,Speed,Altitude,Heading]

record_list(1) = 0;

Number_Of_Flights = length(FlightNumber_List);

for i = 1:Number_Of_Flights;

   % FLight Counter
   if mod(i, 100) == 0
      disp(['Parsing Step 1/1: Number of flights processed: ', num2str(Number_Of_Flights)]);
   end

   %Find the last track record, this idx+1 is the first track record of
   %the next flight
   record_list(i+1) = find(FlightNumber==FlightNumber_List(i),1,'last');

   %Find the Aircraft Type from the MEM_SIDs_Baseline_flightkey.csv file
   %based on the flight record number retrieved in the MEM_SIDs_Baseline_tracks
   flightlist_index = find(FlightNumber_List(i) ==flightrecord,1,'first');
   %Save all the related flight track data separatedly in to the structure
   %file
   FlightTrack(i).Recordnum = FlightNumber_List(flightlist_index);
   FlightTrack(i).actype = actype_list(flightlist_index);
   FlightTrack(i).Time = double(FlightTime_sec(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).Time_Formatted = FlightTime_Formatted(record_list(i)+1:record_list(i+1));
   FlightTrack(i).Latitude = FlightLatitude(record_list(i)+1:record_list(i+1),:);
   FlightTrack(i).Longitude = FlightLongitude(record_list(i)+1:record_list(i+1),:);
   FlightTrack(i).Altitude = double(FlightAltitude(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).GroundSpeed = double(FlightSpeed(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).Heading =  double(FlightHeading(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).Number_of_Trackpoints  = record_list(i+1)- record_list(i)+1;
end

save ([Project_Dir, '/Parsed_', FlightTracks_Name, '.mat'], 'FlightTrack')

return;
```

Pair_Filter.m

```
% This m-file do the filter computation check for the selected FlightTrack
% Date: 5/28/2013
function Pair_Filter(Install_Dir, Project_Dir, FlightTracks_Name, NumOfThreads, Track_Type)

% clc;
% clear all;
% close all;
%
% %% Load Flight Record
%
% Install_Dir = pwd;
% Tracks_Dir  = [pwd '\Output\'];
% Project_Dir = [pwd '\Output\'];
% FlightTracks_Name = 'Parsed_PDARS_SCT_20080618';

%% Set Global Variables

if strcmp(Track_Type, 'ACES') == 1
   distance_separation_nm = 20;  % nm
else
   distance_separation_nm = 5;  % nm
end
distance_separation_deg = nm2deg(distance_separation_nm); %conversion factor to degrees
vertical_separation = 40; %in 100x ft
time_separation = 5*60; % Seconds

% --------------------------
% Parallel Toolbox
% --------------------------
% if matlabpool('size') == 0
%     matlabpool open local
% end

% -----------------
% Set the Parallel Configuration file:
if(isdeployed)
   % -----------------
   localConfig = [Install_Dir, '\local_mcr.settings']; % can also use uigetfile to let the USER select file
   setmcruserdata('ParallelConfigurationFile',localConfig);
end

% matlabpool close force;
if matlabpool('size') == 0
   eval(['matlabpool open ', num2str(NumOfThreads)]);
end

% progress bar
if(isdeployed)
%     JAVA_DIR = [Install_Dir, '\DATA\EWEM\java'];
%     eval(['pctRunOnAll javaaddpath(''', JAVA_DIR, ''')']);
else
   pctRunOnAll javaaddpath java;
end

if strcmp(Track_Type, 'ACES') == 1
   load ([Project_Dir, '\window_count.mat']);
   Number_Of_Files = window_count;
else
   Number_Of_Files = 1;
end

for f = 1:Number_Of_Files

   disp(['Analyzing Step 2/2: Number of files processed: ', num2str(f)]);

   if strcmp(Track_Type, 'ACES') == 1
      FlightTrack = load([Project_Dir, '\', FlightTracks_Name, '_str_', num2str(f), '.mat'],'FlightTrack'); %load the flight track with constructed
envelope
   else
      FlightTrack = load([Project_Dir, 'Parsed_', FlightTracks_Name, '.mat'],'FlightTrack'); %load the flight track with constructed envelope
```

```
    end
    FlightTrack = FlightTrack.FlightTrack;

    number_of_tracks = length(FlightTrack);

    % Progress bar
    progressStepSize = floor(number_of_tracks/100);
    ppm = ParforProgMon('Progress: ', number_of_tracks, progressStepSize, 300, 80);

    % Memory pre allocation
    flight_pairs_temp = [];
    for j = number_of_tracks:-1:1
        flight_pairs_temp(j).tracks(number_of_tracks).pairs = [];
    end

%     tic;
    parfor j = 1 : number_of_tracks

%         if mod(j,100) == 0
%             disp(['Number of tracks processed: ', num2str(j), '/', num2str(number_of_tracks)]);
%         end %if mod(j,100) == 0
        if mod(j,progressStepSize)==0
            ppm.increment();
        end

        for k = (j+1) : number_of_tracks

            % Find out the leader and follower from the time based
            if FlightTrack(j).Time(1) <= FlightTrack(k).Time(1)
                Lead_Track = FlightTrack(j);
                Follow_Track = FlightTrack(k);
            else
                Lead_Track = FlightTrack(j);
                Follow_Track = FlightTrack(k);
            end % if FlightTrack(j).Time(1) <= FlightTrack(k).Time(1)

            number_of_lpoints = length(Lead_Track.Latitude);
            number_of_fpoints = length(Follow_Track.Latitude);

            followtime      = Follow_Track.Time;%[seconds since 1970]
            followlatitude  = Follow_Track.Latitude;%[degrees]
            followlongitude = Follow_Track.Longitude;%[degrees]
            followaltitude  = Follow_Track.Altitude;%[x100ft]

            %Now we are comparing the distance between all the tracks within the same time window

            %This loop compares chosen flight against other flights and filters out
            %those flights we want to create wake envelopes for

            % Fast time interaction pre check to speed up calcs
            if (Follow_Track.Time(1) - Lead_Track.Time(number_of_lpoints)) > time_separation;
              continue;
            end

            % Vectorized check horizontal interaction only if a vertical interaction is found
            % First ignore if seprated by more than distance_separation_deg of latitude
            leadlatitude = Lead_Track.Latitude;%[degrees]
            latitude_interaction = abs(bsxfun(@minus,leadlatitude',followlatitude)) <= distance_separation_deg;
%             latitude_interaction_2 = abs(bsxfun(@minus,leadlatitude',followlatitude)) <= distance_separation_deg;
            if any(latitude_interaction(:)) == 0 % Skip if no latitude interaction
                continue;
            end
            % Second ignore if seprated by more than 1 degree of longitude
            leadlongitude = Lead_Track.Longitude;%[degrees]
            longitude_interaction = abs(bsxfun(@minus,leadlongitude',followlongitude)) <= 1 & latitude_interaction;
%             longitude_interaction_2 = abs(bsxfun(@minus,leadlongitude',followlongitude)) <= 1;
            if any(longitude_interaction(:)) == 0 % Skip if no latitude interaction
                continue;
            end
```

```
        % Vectorized check vertical interaction first, because it is fast and eliminates lots of records
        leadaltitude = Lead_Track.Altitude;%[x100ft]
        vertical_interaction = abs(bsxfun(@minus,leadaltitude',followaltitude)) <= vertical_separation & longitude_interaction;
%           vertical_interaction_2 = abs(bsxfun(@minus,leadaltitude',followaltitude)) <= vertical_separation;
        if any(vertical_interaction(:)) == 0 % Skip because no vertical interaction found
           continue;
        end

        % Vectorized check if time is conflict for big speed up
        leadtime = Lead_Track.Time;
        time_interaction = abs(bsxfun(@minus,leadtime',followtime)) <= time_separation & vertical_interaction;
%           time_interaction_2 = abs(bsxfun(@minus,leadtime',followtime)) <= time_separation;
        if any(time_interaction(:)) == 0 % Skip because no time interaction found
           continue;
        end

        % Vectorized check actual horizontal sepration.
        leadlatitude_rep = repmat(leadlatitude,1,number_of_fpoints)';
        leadlongitude_rep = repmat(leadlongitude,1,number_of_fpoints)';
        followlatitude_rep = repmat(followlatitude,1,number_of_lpoints);
        followlongitude_rep = repmat(followlongitude,1,number_of_lpoints);

        horizontal_interaction = distance('gc',leadlatitude_rep,leadlongitude_rep,followlatitude_rep,followlongitude_rep) <=
distance_separation_deg & time_interaction; % warning distance in degrees (0.1332 = 8 miles)
%           horizontal_interaction_2 = distance('gc',leadlatitude_rep,leadlongitude_rep,followlatitude_rep,followlongitude_rep) <=
distance_separation_deg;
        if any(horizontal_interaction(:)) == 0 % Skip because no horizontal interaction found
           continue;
        end

%           % Check total interactions
%           interaction_results = time_interaction_2 & vertical_interaction_2 & latitude_interaction_2 & longitude_interaction_2 &
horizontal_interaction_2;
%           if any(interaction_results(:)) == 0 % Skip if no interactions
%              continue;
%           end

        flight_pairs_temp(j).tracks(k).pairs = [Lead_Track.Recordnum, Follow_Track.Recordnum];

%           % In case time conflict, check rest
%           for m = 1: number_of_lpoints %for all flight tracks
%
%              % Check if tracks are close in time
%              leadtime = Lead_Track.Time(m);%[seconds since 1970]
%              time_interaction = (abs(leadtime - followtime) <= time_separation);
%              if any(time_interaction) == 0 % Skip because no time interaction found
%                 continue;
%              end
%              % Check vertical interaction first, because it is fast and eliminates lots of records
%              leadaltitude = Lead_Track.Altitude(m);%[x100ft]
%              vertical_interaction = (abs( leadaltitude - followaltitude) <= vertical_separation) == 1;
%              if any(vertical_interaction) == 0 % Skip because no vertical interaction found
%                 continue;
%              end
%              % Check horizontal interaction only if a vertical interaction is found
%              % First ignore if seprated by more than distance_separation_deg of latitude
%              leadlatitude = Lead_Track.Latitude(m);%[degrees]
%              latitude_interaction = (abs( leadlatitude - followlatitude) <= distance_separation_deg) == 1;
%              if any(latitude_interaction) == 0 % Skip if no latitude interaction
%                 continue;
%              end
%              % Second ignore if seprated by more than 1 degree of longitude
%              leadlongitude = Lead_Track.Longitude(m);%[degrees]
%              longitude_interaction = (abs( leadlongitude - followlongitude) <= 1) == 1;
%              if any(longitude_interaction) == 0 % Skip if no latitude interaction
%                 continue;
%              end
%              % Check actual horizontal sepration.
%              % leadlongitude = leadlongitude * ones(1,number_of_fpoints);
%              % leadlatitude  = leadlatitude  * ones(1,number_of_fpoints);
```

```
%               leadlongitude = leadlongitude * ones(number_of_fpoints,1);
%               leadlatitude  = leadlatitude  * ones(number_of_fpoints,1);
%               horizontal_interaction = (distance('gc',leadlatitude,leadlongitude,followlatitude,followlongitude) <= distance_separation_deg) == 1;
% warning distance in degrees (0.1332 = 8 miles)
%               if any(horizontal_interaction) == 0 % Skip because no horizontal interaction found
%                   continue;
%               end
%
%               %               % Combine interactions
%               %               interaction_results = vertical_interaction == 1 & horizontal_interaction == 1; % & time_interaction == 1
%               %               if any(interaction_results) == 0 % Skip because no interaction found
%               %                   continue;
%               %               else
%               %                   flight_pairs_temp(j).tracks(k).pairs(end+1,:) = [Lead_Track.Recordnum, Follow_Track.Recordnum];
%               %                   break; % No need to check the remaining points
%               %               end % if any(interaction_results) == 0
%
%               % Save pair that interact
%               flight_pairs_temp(j).tracks(k).pairs = [Lead_Track.Recordnum, Follow_Track.Recordnum];
%               break; % No need to check the remaining points
%
%           end %for m = 1: number_of_lpoints

        end %for k = j +  1 : number_of_tracks
    end % parfor  j = 1 : number_of_tracks
%     toc;

    % Reformat filtered_pairs
    flight_pairs_counter = 0;
    flight_pairs = zeros(number_of_tracks^2,2);
    for j = 1 : number_of_tracks
        for k = (j+1) : number_of_tracks

            pairs = flight_pairs_temp(j).tracks(k).pairs;
            if isempty(pairs) == 0
                flight_pairs_counter = flight_pairs_counter + 1;
                flight_pairs(flight_pairs_counter,:) = pairs;
            end

        end % for k = (j+1) : number_of_tracks
    end % for j = 1 : number_of_tracks

    flight_pairs = flight_pairs(1:flight_pairs_counter,:);
    filtered_pairs = unique(flight_pairs,'rows'); % Delete duplicate records
    filtered_pairs(filtered_pairs==0) = []; % Delete zero records

    if strcmp(Track_Type, 'ACES') == 1
        save ([Project_Dir, '\Filtered_Pairs_', FlightTracks_Name, '_str_', num2str(f), '.mat'], 'filtered_pairs');
    else
        save ([Project_Dir, '\Filtered_Pairs_', FlightTracks_Name, '.mat'], 'filtered_pairs');
    end
    ppm.delete();

    clear FlightTrack;
    clear filtered_pairs;
    clear flight_pairs_temp;

end % for f = 1:Number_Of_Files

return;
```

PDARS_Parser_Structure.m

```
function PDARS_Parser_Structure(Project_Dir, FlightTracks_Name)
% ************************************************************************
% This M-file parses the PDARS structure data
% If you have the raw PDARS data that is stored in the txt file
```

```
% Please run PDARS_parser_raw first
% This file might run slow due to the size of the PDARS file
% Date: Oct 2, 2013
% ********************************************************************************
% clc;
% clear all;
% close all;

% Unit Transformation Constants

% m_to_km = 0.001;
knots_to_ms = 0.514; %convert knots to m\s
ft_to_meters = 0.3048; %convert ft to meters
fl_to_meters = 100*ft_to_meters; %convert fight level to meters

% Tracks_Dir  = [pwd '/Output'];
% Project_Dir = [pwd '/Output'];
% FlightTracks_Name = 'PDARS_SCT_20080618';/
load([Project_Dir, '\', FlightTracks_Name, '.mat'], 'PDARS'); %load the flight track with constructed envelope

number_of_tracks = length(PDARS);

% Preallocate for speed
FlightTrack(number_of_tracks).actype = [];
FlightTrack(number_of_tracks).Recordnum = [];
FlightTrack(number_of_tracks).Time = [];
FlightTrack(number_of_tracks).Latitude = [];
FlightTrack(number_of_tracks).Longitude = [];
FlightTrack(number_of_tracks).Altitude = [];
FlightTrack(number_of_tracks).Heading = [];
FlightTrack(number_of_tracks).GroundSpeed = [];
FlightTrack(number_of_tracks).Number_of_Trackpoints = [];

for i = 1:number_of_tracks

   % FLight Counter
   if mod(i, 1000) == 0
      disp(['Parsing Step 3/3: Number of flights processed: ', num2str(i), '/', num2str(number_of_tracks)]);
   end

   smoothed_latitude  = smooth(PDARS(i).Track.latitude);
   smoothed_longitude = smooth(PDARS(i).Track.longitude);
   smoothed_altitude  = smooth(PDARS(i).Track.alt);

   % corrected_heading = zeros(1,length(smoothed_longitude)); %prealocate the variable

   aircraft_position = [smoothed_latitude,smoothed_longitude,smoothed_altitude];

   [aircraft_position_unique,idx] = unique(single(aircraft_position),'rows'); %reduce some level of precision to prevent 29.00000001 from
29.0000000

   idx = sort(idx);

   %     for k = 2:(size(smoothed_latitude)-1);
   %        corrected_heading (k)= azimuth(smoothed_latitude(k),smoothed_longitude(k),smoothed_latitude(k+1),smoothed_longitude(k+1));
   %     end  %for k=2:(size(smoothed_latitude)-1);
   %     corrected_heading (1)= corrected_heading (2);
   %     corrected_heading (k+1)=corrected_heading (k);
   %     heading = round(corrected_heading');

   number_of_pooints = size(smoothed_latitude,1);
   if number_of_pooints>1
      corrected_heading = zeros(number_of_pooints,1);
      corrected_heading(2:number_of_pooints-1) = azimuth(smoothed_latitude(2:number_of_pooints-
1),smoothed_longitude(2:number_of_pooints-1),smoothed_latitude(3:number_of_pooints),smoothed_longitude(3:number_of_pooints));
      corrected_heading(1)   = corrected_heading(2);
      corrected_heading(end) = corrected_heading(number_of_pooints-1);
   else
      corrected_heading = zeros(number_of_pooints,1);
   end
```

```
    %save everything in a struct array
    FlightTrack(i).actype = PDARS(i).acType;
    FlightTrack(i).Recordnum = PDARS(i).Header.fltKey;
    FlightTrack(i).Time = PDARS(i).Track.recTime(idx)';
    FlightTrack(i).Latitude = PDARS(i).Track.latitude(idx); %degree
    FlightTrack(i).Longitude = PDARS(i).Track.longitude(idx); %degree
    FlightTrack(i).Altitude = PDARS(i).Track.alt(idx)'*fl_to_meters;  % fl to meter
    FlightTrack(i).Heading = round(corrected_heading);
    FlightTrack(i).GroundSpeed =  PDARS(i).Track.Speed_knots(idx)'*knots_to_ms ; %knots to m\s
    FlightTrack(i).Number_of_Trackpoints  = length(PDARS(i).Track.recTime(idx));

end

save ([Project_Dir, '\Parsed_', FlightTracks_Name, '.mat'], 'FlightTrack');

return;
```

Target_Parser.m

```
function TARGET_Parser(Project_Dir, Track_FileName, FlightTracks_Name)
% ************************************************************************
% This M-file parses the MEM data provided by CSSI
% The original sample data from CSSI is as shown:
% 10001, 0:00:45.0,35.02388,-89.98689,3.21,0,359
% The data is been saved to Tab separated txt file and erased all the
% commas and became:
% 10001 0 00 45.0 35.02388 -89.98689 3.21 0,359
% Date: Feb 4, 2013
% ************************************************************************
% clear all;
% close all;
% clc;

% FlightTracks_Name = 'WTMD_SingleRunway_ShortRoutes';
% Tracks_Dir  = 'C:\WEM Project\WakeWISE\Input\Tracks';
% Project_Dir = 'C:\WEM Project\EWEM\Input\Tracks';

fid1 = fopen([Project_Dir, '/',Track_FileName, '_Flights.csv'], 'r');
flightlist = textscan(fid1, '%f%s%s%s%s', 'delimiter', ',');
%Field 1: Flight record number
%Field 2: Flight ID
%Field 3: Aircraft Type

flightrecord = flightlist{1};
actype_list = flightlist{3};

fid2 = fopen([Project_Dir, '/',Track_FileName, '_Tracks.csv'],'r');
flighttrack = textscan(fid2, '%d%s%f%f%f%d%d','delimiter',',');
%Field 1: Flight record number
%Field 2: Flight record time
%Field 3: Flight Latitude %degree
%Field 4: Flight Longitude %degree
%Field 5: Flight GroudSpeed (m/s)
%Field 6: Flight Altitude (fl)
%Field 7: Flight Heading

FlightNumber = flighttrack{1};
FlightTime = datevec(datenum(flighttrack{2})); %stored in vectors with 6 columns in yyyy,mm,dd,hh,mm,ss
FlightTime_HM = [FlightTime(:,4),FlightTime(:,5)];
%%Put hh and mm into one cell, consistent with future wind data
b =sprintf('%d%d*', FlightTime_HM(:, 1:2).');
FlightTime_Formatted = sscanf(b, '%d*');
FlightTime_sec = FlightTime(:,4)*3600+FlightTime(:,5)*60+FlightTime(:,6);
FlightLatitude = flighttrack{3};
FlightLongitude = flighttrack{4};
FlightSpeed = flighttrack{6}; %knots
FlightAltitude = flighttrack{5}*100*0.3048; %FL->ft->m
FlightHeading = flighttrack{7};
```

```
FlightNumber_List = unique(FlightNumber);

%Save different flight information separately in structure file
%FlightTrack(i).Flight.[actype,Time,Latitude,Longitude,Speed,Altitude,Heading]

record_list(1) = 0;

Number_Of_Flights = length(FlightNumber_List);

for i = 1:Number_Of_Flights;

   % FLight Counter
   if mod(i, 100) == 0
      disp(['Parsing Step 1/1: Number of flights processed: ', num2str(Number_Of_Flights)]);
   end

   %Find the last track record, this idx+1 is the first track record of
   %the next flight
   record_list(i+1) = find(FlightNumber==FlightNumber_List(i),1,'last');

   %Find the Aircraft Type from the MEM_SIDs_Baseline_flightkey.csv file
   %based on the flight record number retrieved in the MEM_SIDs_Baseline_tracks
   flightlist_index = find(FlightNumber_List(i) ==flightrecord,1,'first');
   %Save all the related flight track data separatedly in to the structure
   %file
   FlightTrack(i).Recordnum = FlightNumber_List(flightlist_index);
   FlightTrack(i).actype = actype_list(flightlist_index);
   FlightTrack(i).Time = double(FlightTime_sec(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).Time_Formatted = FlightTime_Formatted(record_list(i)+1:record_list(i+1));
   FlightTrack(i).Latitude = FlightLatitude(record_list(i)+1:record_list(i+1),:);
   FlightTrack(i).Longitude = FlightLongitude(record_list(i)+1:record_list(i+1),:);
   FlightTrack(i).Altitude = double(FlightAltitude(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).GroundSpeed = double(FlightSpeed(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).Heading =  double(FlightHeading(record_list(i)+1:record_list(i+1),:));
   FlightTrack(i).Number_of_Trackpoints  = record_list(i+1)- record_list(i)+1;
end

save ([Project_Dir, '/Parsed_', FlightTracks_Name, '.mat'], 'FlightTrack')

return;
```

Encounter_Plot.m

```
%clear all;
%clc;
%% This script will plot conflicts detected by EWEM

Install_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL (trunk)/';
Project_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/ACES_NASA/Nick_Mode_1/';
Tracks_Dir  = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/ACES_NASA/Nick_Mode_1/';
% Track_FileName= sprintf('aircraftstate_70Kflights_VT_timesorted.csv');
FlightTracks_Name ='ACES'; %'ACES';%'ACES_70Kflights_VT';

Track_Type     = 'ACES'; % PDARS, MEM, ACES
load ([Tracks_Dir,'window_count.mat']);
Number_Of_Files = window_count;

%% Global Variables

g = 9.81;%Gravitational constant (m/s^2)
s = pi/4;%Trailing vortices convert parameter

m_to_km = 0.001;
knots_to_ms = 0.514; %convert knots to m/s
ft_to_meters = 0.3048; %convert ft to meters
fl_to_meters = 100*ft_to_meters; %convert fight level to meters

number_of_total_encounters =0;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%Import Aircraft information table (mass, wingspan, etc)
fid1 = fopen([Install_Dir, 'Aircraft_Info_Common_metric_units.csv'],'r');
aircraftinfolookup = textscan(fid1, '%s%s%s%s%s%s%f%f%f%f%f%s%f', 'HeaderLines',1,'delimiter', ',');
%Field 1: Aircraft ID (BADA Name)
%Field 2: Name
%Field 3: Manufacture Name
%Field 4: Short Name
%Field 5: BADA Name
%Field 6: Wake Class (Old)
%Field 7: Operating Empty Weight OEW (kg)
%Field 8: Maximum Landing Weight (kg)
%Field 9: Maximum Takeoff Weight (kg)
%Field 10: Wingspan (m)
%Field 11: ReCategorization wakeclass from A-F
%Field 12: ReCategorization wakeclass presented in number from 1-6
acid      = aircraftinfolookup{1};
wingspan     = aircraftinfolookup{10}; %m
wakeclass = aircraftinfolookup{12};
wakeclass_num = aircraftinfolookup{13};
oew_all = aircraftinfolookup{7}; %kg
mlw_all = aircraftinfolookup{8}; %kg
mtow_all = aircraftinfolookup{9}; %kg

fid2 = fopen([Install_Dir, '/BADA_Mapping.csv'],'r');
aircraftmappingtable = textscan(fid2, '%s%s%s%s%s%s%f%f%f%f%f%s%f', 'HeaderLines',1,'delimiter', ',');
%Field 1: Aircraft ID (BADA Name)
%Field 2: Manufacture Name
%Field 3: Name or Model
%Field 4: File (Which profile is going to use in the Aircraft_Info_Common_metric_units.csv table)
%Field 5: Old file name
aircraft_list         = aircraftmappingtable{1};
aircraft_mapping_list = aircraftmappingtable{4}; %m

for i = 1: Number_Of_Files

   %% Import all flight track data

   if strcmp(Track_Type, 'ACES') == 1

      FlightTracks_Name_num = [FlightTracks_Name, '_str_', num2str(i)];

      FlightTrack = load([Tracks_Dir,  FlightTracks_Name_num, '.mat'],'FlightTrack'); %load the flight track with constructed envelope
   else
      FlightTrack = load([Tracks_Dir, 'Parsed_', FlightTracks_Name, '.mat'],'FlightTrack'); %load the flight track with constructed envelope
   end

   FlightTrack = FlightTrack.FlightTrack;
   %     Filtered_Pairs = load([Project_Dir, 'Filtered_Pairs_', FlightTracks_Name_num, '.mat']);

   Detected_Encounters = load([Project_Dir, 'wakepiercing_', FlightTracks_Name_num, '.mat'],'wakepiercing_sum');

   %List of aircraft contained in the MEM structure file
   fid4 = fopen([Tracks_Dir, '/', FlightTracks_Name_num, '_flightlist.txt'], 'r');
   flightlist = textscan(fid4, '%f', 'delimiter', ',');
   flightlist = flightlist{1};

   Circulation_Boundary = [250,250,200,125,100,100];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
   fclose all;

   %%
   %CONSTANTS and PREDETERMINED VALUES---CAN BE CHANGED BY USER

   for x = 1:length(Detected_Encounters.wakepiercing_sum)
```

```
% Skip to next record if no encounter recorded
if isempty(Detected_Encounters.wakepiercing_sum(x).filteredleadrecord) == 1
   continue;
end

number_of_total_encounters = number_of_total_encounters+1;

leadrecordnum = Detected_Encounters.wakepiercing_sum(x).filteredleadrecord(1);
i = find(leadrecordnum==flightlist);
j = Detected_Encounters.wakepiercing_sum(x).filteredleadrecord(2);

followerrecordnum = Detected_Encounters.wakepiercing_sum(x).filteredfollowerrecord(1);
m = find(followerrecordnum==flightlist);
n = Detected_Encounters.wakepiercing_sum(x).filteredfollowerrecord(2);

actype_lead = FlightTrack(i).actype; %Get the flight type "B722"
actype_follower = FlightTrack(m).actype;

% Mapping the aircraft type into a BADA aircraft type
leaderidx_mapping   = find(strcmp(aircraft_list,actype_lead),1);
followeridx_mapping = find(strcmp(aircraft_list,actype_follower),1);
leadertype_mapping   = aircraft_mapping_list(leaderidx_mapping);
followertype_mapping = aircraft_mapping_list(followeridx_mapping);

actype_lidx = find(strcmp(acid,leadertype_mapping),1); %Find the corresponding line in the Aircraft_info table for aircraft parameters
actype_fidx = find(strcmp(acid,followertype_mapping),1);
leader_aircraft_CAT = wakeclass_num(actype_lidx);
follower_aircraft_CAT = wakeclass_num(actype_fidx);

%       Near_Altitude = FlightTrack(m).Altitude(max(n-3,1):min(n+3,FlightTrack(m).Number_of_Trackpoints));
%
%       if (Near_Altitude(end) - Near_Altitude(1))>0
%          Flight_angle =1; %climb
%       elseif (Near_Altitude(end) - Near_Altitude(1))<0
%          Flight_angle = -1; %descent
%       else
%          Flight_angle =0; %cruise
%       end
%
%       Near_Altitude_leader = FlightTrack(i).Altitude(max(j-3,1):min(j+3,FlightTrack(i).Number_of_Trackpoints));
%
%       if (Near_Altitude_leader(end) - Near_Altitude_leader(1))>0
%          Flight_angle_leader =1; %climb
%       elseif (Near_Altitude_leader(end) - Near_Altitude_leader(1))<0
%          Flight_angle_leader = -1; %descent
%       else
%          Flight_angle_leader =0; %cruise
%       end

%       Flight_distance(number_of_total_encounters) = deg2nm(distance(FlightTrack(m).Latitude(n),
FlightTrack(m).Longitude(n),FlightTrack(i).Latitude(j),FlightTrack(i).Longitude(j)));

%       Envelope_Distance = deg2nm(Detected_Encounters.wakepiercing_sum(x).envelopelength(end));

        InitialCirculation = Detected_Encounters.wakepiercing_sum(x).InitialCirculation;
%
        Flight_GroundSpeed = FlightTrack(i). GroundSpeed(j);
        Flight_Altitude = FlightTrack(i). Altitude(j);
%
        Circulation_Drop = Circulation_Boundary(leader_aircraft_CAT) - InitialCirculation;
%
        Circulation_Strength = (Flight_distance(number_of_total_encounters) /Envelope_Distance) * Circulation_Drop + InitialCirculation;

        Encounter_pair(number_of_total_encounters,:) = {leadrecordnum,
followerrecordnum,{actype_lead},{actype_follower},leader_aircraft_CAT,follower_aircraft_CAT,Flight_distance(number_of_total_encounters),
InitialCirculation,Circulation_Strength, Flight_angle_leader,Flight_angle,Flight_GroundSpeed,Flight_Altitude}; %Circulation_Strength,
        groundspeed =  FlightTrack(i).GroundSpeed(j);
%
envelope_coordinates = Detected_Encounters.wakepiercing_sum(x).envelope;
[a,b] = size(envelope_coordinates);
```

```
number_of_envelope_points = b/4;

%
%       %%%%%%%%%%%%%%%%%%%
%       Plot the 3D View
%       %%%%%%%%%%%%%%%%%%%
figure
for m = 1: number_of_envelope_points

   Coordinate_a = [envelope_coordinates(1,m),envelope_coordinates(2,m),envelope_coordinates(3,m)];
   Coordinate_b = [envelope_coordinates(1,2*number_of_envelope_points-m+1),envelope_coordinates(2,2*number_of_envelope_points-
m+1),envelope_coordinates(3,2*number_of_envelope_points-m+1)];
   Coordinate_c =
[envelope_coordinates(1,2*number_of_envelope_points+m),envelope_coordinates(2,2*number_of_envelope_points+m),envelope_coordinates(3,
2*number_of_envelope_points+m)];
   Coordinate_d = [envelope_coordinates(1,4*number_of_envelope_points-m+1),envelope_coordinates(2,4*number_of_envelope_points-
m+1),envelope_coordinates(3,4*number_of_envelope_points-m+1)];

   Coordinate_Rectangle = [Coordinate_a;Coordinate_b;Coordinate_d;Coordinate_c;Coordinate_a];

   plot3( Coordinate_Rectangle(:,2), Coordinate_Rectangle(:,1), Coordinate_Rectangle(:,3),'--b','LineWidth',2);
         hold on
end %for m = 1: number_of_points

   plot3(envelope_coordinates(2,:),envelope_coordinates(1,:),envelope_coordinates(3,:),'-r','LineWidth',2);
   hold on
%   for p = 1: length(FlightTrack(i).Longitude) %Plot the leader FlightTrack.FlightTrack
%      h2 = plot3(FlightTrack(i).Longitude(p),FlightTrack(i).Latitude(p),FlightTrack(i).Altitude(p),'-*r','LineWidth',1);
%      % text(FlightTrack.FlightTrack(m).longitude(x),FlightTrack.FlightTrack(m).Altitude(x)*100*0.3048, num2str(x));
%   end %for x = 1: number_of_trackpoints
%


%   for q = 1:length(FlightTrack(m).Longitude) %Plot the follower FlightTrack.FlightTrack
%      h3= plot3(FlightTrack(m).Longitude(q), FlightTrack(m).Latitude(q),FlightTrack(m).Altitude(q),'-*g','LineWidth',1); %Plot the
follower FlightTrack.FlightTrack
%   end %for y = 1:length(FlightTrack.FlightTrack(m).longitude)
%
% grid on
xlabel('Longitude (degree)','Fontsize',50);
ylabel('Latitude (degree)','Fontsize',50);
zlabel('Altitude (m)','Fontsize',50);
set(gca, 'FontSize', 40);

Title = ['Wake Zone for ',[num2str(leadrecordnum),'~',actype_lead]];%'followed by',[num2str(followerrecordnum),' ',actype_follower]];
title(Title,'Fontsize',50)

%   legend([h2,h3],['Lead Aircraft ',actype],['Follow Aircraft ',actype_follower]);
%
%       %%%%%%%%%%%%%%%%%%%
%       Plot the Side View
%       %%%%%%%%%%%%%%%%%%%
%           if ismember(encounter_pair,encounter_diff_5nm_3nm,'rows')  %Plot the difference only

%               number_of_total_diff = number_of_total_diff + 1;


%               Encounter_diff(number_of_total_diff,:) = encounter_pair;
%
figure

%         subplot(2,1,1)

plot(envelope_coordinates(2,:),envelope_coordinates(3,:),'-b','LineWidth',2);
hold on
%               for p = 1: length(FlightTrack(i).Longitude) %Plot the leader FlightTrack.FlightTrack
%                  h4 = plot(FlightTrack(i).Longitude(p),FlightTrack(i).Altitude(p),'-*r','LineWidth',1);
%                  % text(FlightTrack.FlightTrack(m).longitude(x),FlightTrack.FlightTrack(m).Altitude(x)*100*0.3048, num2str(x));
%               end %for x = 1: number_of_trackpoints
%
```

```
    %
    %           plot(FlightTrack(i).Longitude(j),FlightTrack(i).Altitude(j),'-or','LineWidth',3);
    %
    %           for q = 1:length(FlightTrack(m).Longitude) %Plot the follower FlightTrack.FlightTrack
    %              h5= plot(FlightTrack(m).Longitude(q), FlightTrack(m).Altitude(q),'-*g','LineWidth',1); %Plot the follower
FlightTrack.FlightTrack
    %           end %for y = 1:length(FlightTrack.FlightTrack(m).longitude)
    %
    %           plot(FlightTrack(m).Longitude(n), FlightTrack(m).Altitude(n),'-og','LineWidth',3); %Plot the follower FlightTrack.FlightTrac
    %
    grid on
    xlabel('Longitude (degree)','Fontsize',50);
    ylabel('Altitude (m)','Fontsize',50);
    set(gca, 'FontSize', 40);

    Title = ['Wake Zone for ',num2str(leadrecordnum),'-',num2str(followerrecordnum)];
    title(Title,'Fontsize',50)

    %           legend([h4,h5],['Lead Aircraft ',actype],['Follow Aircraft ',actype_follower]);

    subplot(2,1,2)

    hold on
    for p = 1: length(FlightTrack(i).Longitude) %Plot the leader FlightTrack.FlightTrack
       h6 = plot(FlightTrack(i).Longitude(p),FlightTrack(i).Latitude(p),'-*r','LineWidth',1);
       % text(FlightTrack.FlightTrack(m).longitude(x),FlightTrack.FlightTrack(m).Altitude(x)*100*0.3048, num2str(x));
    end %for x = 1: number_of_trackpoints


    plot(FlightTrack(i).Longitude(j),FlightTrack(i).Latitude(j),'-or','LineWidth',3);

    for q = 1:length(FlightTrack(m).Longitude) %Plot the follower FlightTrack.FlightTrack
       h7= plot(FlightTrack(m).Longitude(q), FlightTrack(m).Latitude(q),'-*g','LineWidth',1); %Plot the follower FlightTrack.FlightTrack
    end %for y = 1:length(FlightTrack.FlightTrack(m).longitude)

    plot(FlightTrack(m).Longitude(n), FlightTrack(m).Latitude(n),'-og','LineWidth',3); %Plot the follower FlightTrack.FlightTrack

    grid on
    xlabel('Longitude (degree)','Fontsize',50);
    ylabel('Latitude (degree)','Fontsize',50);
    set(gca, 'FontSize', 40);

    Title = ['Wake Zone for ',num2str(leadrecordnum),'-',num2str(followerrecordnum)];
    title(Title,'Fontsize',50)

    legend([h6,h7],['Lead Aircraft ',actype],['Follow Aircraft ',actype_follower]);

    %           end
  end %for x = 1:length(Detected_Encounters.wakepiercing_sum)

end%
```

FlightAware_Parser.m

```
%function Flightaware_Parser(Project_Dir, Track_FileName, FlightTracks_Name)
%*************************************************************************
% This M-file parses the flighttrack from FlightAware into structure file
% The original sample data from Flightaware is as shown:
% Flight ID          Time (UTC)          Longitude          Latitude          Groundspeed (knots) Altitude (feet)          Type
          Facility Name          Facility Type
%   1          5/14/13 4:02          -118.4179993          33.93600082          161                    800          A          FlightAware
% Date: March 3, 2014
% *************************************************************************

Project_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/Flightaware_East_Coast/';
Track_FileName = 'FlightAware_Cape_Canaveral';
FlightTracks_Name = ['Parsed_',Track_FileName];
```

```
%Global Variables
knots_to_ms = 0.514; %convert knots to m/s
ft_to_meters = 0.3048; %convert ft to meters

fid1 = fopen([Project_Dir,Track_FileName, '_Flights.csv'], 'r');
flightlist = textscan(fid1, '%f%s%s%s%s%s%f%f%s', 'delimiter', ',','headerLines', 1);
%Field 1: Flight ID
%Field 2: Airline Name
%Field 3: Ident
%Field 4: Aircraft Type
%Field 5: Origin
%Field 6: Destination
%Field 7: Filed Altitude (feet)
%Field 8: Filed Speed (knots)
%Field 9: Route

flightrecord = flightlist{1};
Airline_Name = flightlist{2};
FlightID = flightlist{3};
actype_list = flightlist{4};
Origin_list = flightlist{5};
Destination_list = flightlist{6};
Filed_Altitude = flightlist{7};
Filed_Speed = flightlist{8};
Route = flightlist{9};

fid2 = fopen([Project_Dir,Track_FileName, '_Tracks.csv'],'r');
flighttrack = textscan(fid2, '%f%s%f%f%f%f%s%s%s', 'delimiter', ',','headerLines', 1);
%Field 1: Flight ID
%Field 2: Time(UTC)
%Field 3: Longitude
%Field 4: Latitude
%Field 5: Groundspeed (kts)
%Field 6: Altitude (feet)
%Field 7: Type
%Field 8: Facility Name
FlightID = flighttrack{1};
FlightTime_sec = datenum(flighttrack{2});
FlightTime_UTC = datevec(FlightTime_sec);
%%Put hh and mm into one cell, consistent with future wind data
%FlightTime_HM = [FlightTime(:,4),FlightTime(:,5)];
%b =sprintf('%d%d*', FlightTime_HM(:, 1:2).');
%FlightTime_Formatted = sscanf(b, '%d*');
FlightLongitude = flighttrack{3}; %degree
FlightLatitude = flighttrack{4};
FlightSpeed = knots_to_ms*flighttrack{5}; %m/s
FlightAltitude = ft_to_meters*flighttrack{6}; %meters

numberofrecords = length(FlightAltitude) ;

% Fill in the missing altitude of the FlightAware data
for i = 1:numberofrecords
   if isnan(FlightAltitude(i))
      FlightAltitude(i) = FlightAltitude(i-1);
   end
end

FlightNumber_List = unique(FlightID);

Number_Of_Flights = length(FlightNumber_List);

record_list = zeros(Number_Of_Flights,1);

count =0;

%Save different flight information separately in structure file
%FlightTrack(i).Flight.[actype,Time,Latitude,Longitude,Speed,Altitude,Heading]

for i = 1:Number_Of_Flights
```

```
    % FLight Counter
    if mod(i, 1000) == 0
        disp(['Parsing Step 1/1: Number of flights processed: ',num2str(i), '/', num2str(Number_Of_Flights)]);
    end

    %Find the last track record, this idx+1 is the first track record of
    %the next flight
    record_list(i+1) = find(FlightID==FlightNumber_List(i),1,'last');

    %Find the Aircraft Type from the _flightlist.csv file
    %based on the flight record number retrieved in the _tracks
    flightlist_index = find(FlightNumber_List(i) ==flightrecord,1,'first');
    %Save all the related flight track data separatedly in to the structure
    %file

    if (record_list(i+1)- record_list(i))== 1 %if only one FlightTrack point on record, skip the track
        continue;
    end

    count = count +1;

    FlightTrack(count).actype = actype_list{flightlist_index};
    FlightTrack(count).AirlineName = Airline_Name{flightlist_index};
    FlightTrack(count).FlightID = FlightID(flightlist_index);
    FlightTrack(count).Origin = Origin_list{flightlist_index};
    FlightTrack(count).Destination = Destination_list{flightlist_index};
    FlightTrack(count).Filed_Altitude = Filed_Altitude(flightlist_index);
    FlightTrack(count).Filed_Speed = Filed_Speed(flightlist_index);
    FlightTrack(count).Route = Route{flightlist_index};
    FlightTrack(count).Latitude = FlightLatitude(record_list(i)+1:record_list(i+1),:);
    FlightTrack(count).Longitude = FlightLongitude(record_list(i)+1:record_list(i+1),:);
    FlightTrack(count).Altitude = double(FlightAltitude(record_list(i)+1:record_list(i+1),:));
    FlightTrack(count).Recordnum = FlightNumber_List(flightlist_index);
    FlightTrack(count).TimeSec = double(FlightTime_sec(record_list(i)+1:record_list(i+1),:));
    FlightTrack(count).Time_UTC= FlightTime_UTC(record_list(i)+1:record_list(i+1),:);
    %FlightTrack(count).Time_HHMM = FlightTime_Formatted(record_list(i)+1:record_list(i+1));
    FlightTrack(count).GroundSpeed = double(FlightSpeed(record_list(i)+1:record_list(i+1),:));
    FlightTrack(count).Number_of_Trackpoints  = length(FlightTrack(count).Latitude);
    FlightTrack(count).Heading  = azimuth(FlightLatitude(record_list(i)+1:record_list(i+1)-1,:),FlightLongitude(record_list(i)+1:record_list(i+1)-
1,:),FlightLatitude(record_list(i)+2:record_list(i+1),:),FlightLongitude(record_list(i)+2:record_list(i+1),:));
    FlightTrack(count).Heading(FlightTrack(count).Number_of_Trackpoints) =
FlightTrack(count).Heading(FlightTrack(count).Number_of_Trackpoints-1);


    FlightTrack(count).Latitude  = smooth(FlightTrack(count).Latitude);
    FlightTrack(count).Longitude = smooth(FlightTrack(count).Longitude);
    FlightTrack(count).Altitude  = smooth(FlightTrack(count).Altitude);
    %Eliminate duplicate points, and update the strucuture file
    aircraft_position = [FlightTrack(count).Latitude,FlightTrack(count).Longitude];

    [aircraft_position_unique,idx] = unique(single(aircraft_position),'rows'); %reduce some level of precision to prevent 29.00000001 from
29.0000000

    idx = sort(idx);
    FlightTrack(count).Latitude = aircraft_position_unique(:,1);
    FlightTrack(count).Longitude = aircraft_position_unique(:,2);
    FlightTrack(count).Altitude = FlightTrack(count).Altitude(idx);
    FlightTrack(count).Timesec = FlightTrack(count).TimeSec(idx);
    FlightTrack(count).Time_UTC= FlightTrack(count).Time_UTC(idx,:);
    % FlightTrack(count).Time_HHMM = FlightTrack(count).Time_HHMM(idx);
    FlightTrack(count).GroundSpeed =  FlightTrack(count).GroundSpeed(idx);
    FlightTrack(count).Heading  = FlightTrack(count).Heading(idx);
    FlightTrack(count).Number_of_Trackpoints  = length(FlightTrack(count).Latitude);

end

save ([Project_Dir, '/Parsed_', FlightTracks_Name, '_FULL.mat'], 'FlightTrack')

%return
```

RUN_EWEM.m

```
Install_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL (trunk)/';
Project_Dir = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/PDARS/N90_20080319/';
Tracks_Dir  = '/Users/zhengfan/Documents/New_EWEM_MODEL/WEM_MODEL_Self/PDARS/N90_20080319/';
FlightTracks_Name = 'N90_20080319';

NumOfThreads = 4;
% Define
Track_Type   = 'PDARS'; % PDARS, TARGET, ACES, Flightaware
% Define Circulation threshold table for each wake groups
Circulation_Threshold = [250,250,200,125,100,100]; %User Definable, Please enter suitable number for each group between 100 to 250, with the
step of 25 (e.g: 125, 175)
% Choose from conservative and reality wake structure analysis, default value is 0 (reality)
Model_Mode = 1;
% Define wind data, 1)Directly from ASOS output, fix input (with both
% direction and strength), distribution (expanding the analysis based
% on given wind mean and std)
Wind_Source   = 'Fix Input';

%% Run EWEM
if strcmp(Track_Type, 'ACES') == 1
   load ([Tracks_Dir, 'window_count.mat']);
   Number_Of_Files = window_count;
else
   Number_Of_Files = 1;
end

%% Load the right wind profile
if strcmp(Wind_Source, 'Distribution') == 1

   %% User Input
   crosswind_mean = Wind_Profile(3);
   crosswind_std = Wind_Profile(4);

   % User input is the crosswind strength, no need to input wind direction

   crosswind_all = [crosswind_mean,crosswind_std];

elseif strcmp(Wind_Source, 'Fix Input') == 1

   %% User Input
   %wind_speed = Wind_Profile(1); %Can be fixed as a single number or a list varies by the altitude
   %wind_direction = Wind_Profile(2); %in degree, north as 0, east as 90

   % User input is the fixed wind data, no variation needed.
   %crosswind_std = Wind_Profile(3);
   wind_speed = 0;
   crosswind_std = 0;
   wind_direction = 0;
   crosswind_all = [wind_speed,crosswind_std,wind_direction];
   % crosswind_kts = wind_speed*sin((wind_direction-heading)/360*2*pi);
   % crosswind = crosswind_kts*0.514*0.001;
   %%
elseif strcmp(Wind_Source, 'ASOS') == 1
   [wind_time_all,wind_direction_all, wind_speed_all] = ASOS_Parser(Wind_File);
   crosswind_std =zeros(length(wind_time_all),1);
   crosswind_all = [wind_time_all,wind_speed_all,crosswind_std,wind_direction_all];
else
   crosswind_all = [];
end

tic;
for i = 1:Number_Of_Files

   % Calclate time left
   Elapsed_Time_hrs = toc / 3600;
   Average_Time_By_File_hrs = Elapsed_Time_hrs / i;
   Remaining_Time_hrs = (Number_Of_Files - i) * Average_Time_By_File_hrs;
```

```
    % Status
    Elapsed_Time_hrs_round   = num2str(round(Elapsed_Time_hrs * 100) / 100);
    Remaining_Time_hrs_round = num2str(round(Remaining_Time_hrs * 100) / 100);
    disp([num2str(i),'/', num2str(Number_Of_Files), ' - Elapsed Time: ', Elapsed_Time_hrs_round, ' hrs - Remaing Time: ',
Remaining_Time_hrs_round, ' hrs.']);

    % EWEM
    if strcmp(Track_Type, 'ACES') == 1
       FlightTracks_Name_num = [FlightTracks_Name, '_str_', num2str(i)];
    else
       FlightTracks_Name_num = FlightTracks_Name;
    end %if strcmp(Track_Type, 'ACES') == 1

    EWEM(Install_Dir, Tracks_Dir, Project_Dir, FlightTracks_Name_num, NumOfThreads, Track_Type, Model_Mode,
Circulation_Threshold,crosswind_all)

end % for i = 1:Number_Of_Files

Google_Earth_Visualization(Tracks_Dir, Project_Dir, FlightTracks_Name, Track_Type);
Encounter_Summary(Install_Dir, Tracks_Dir, Project_Dir, FlightTracks_Name, Track_Type, Circulation_Threshold);

return;
```