A Model to Convert Airport Geographic and Geometric Information into a Node-Link Network

by

Yang Zhang

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

Master of Science
In
Civil and Environmental Engineering

Antonio A. Trani

Montasir M. Abbas

Linbing Wang

November 20, 2014
Blacksburg, Virginia

Keywords: Airport Node-Link Model, ADSIM+, AIXM, Topology, GIS

A Model to Convert Airport Geographic and Geometric Information into a Node-Link Network

by

Yang Zhang

ABSTRACT

An airport node-link network model is an important input for most airport simulation models. Developing, maintaining and updating detailed airport surface node-link models require significant work. A model to convert airport geographic and geometric information into a node-link network is thus needed.

In this thesis, an efficient model to automate the procedure of converting airport geographic and geometric information into a node-link network is proposed. The geographic and geometric information are obtained from the Aeronautical Information Exchange Model (AIXM). The node-link network is generated by converting the geographic and geometric information contained into the AIXM files using coincident geometry management and polygon representation development.

Finally, using the airport node-link network generation model, a standalone computer application, called *Taxiway Toolkit*, is developed to improve the Airfield Delay Simulation Model (ADSIM+).

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Dr. Antonio A. Trani, for his invaluable guidance, patience and inspiration during my studies at Virginia Tech. Without his support, this thesis would not have been completed. I am also indebted to him for providing financial support from a Federal Aviation Administration (FAA) contract.

Besides my advisor, I have also been honored to have Dr. Montasir Abbas and Dr. Linbing Wang as members of my committee. I would like to thank them for their encouragement and insightful comments.

I would also like to thank Dr. Martin Durbin and Joseph Post of the Federal Aviation Administration (FAA) Systems Analysis and Modeling Office for their technical and financial support and valuable feedback of this work.

I would like to thank my colleagues in Air Transportation System Laboratory (ATSL) at Virginia Tech: Mr. Howard Swingle, Mr. Nicolas K. Hinze, Zheng Fan, Tao Li, Thomas Spencer, Osama Alsalous and Davide Pu for their support and warmhearted friendship.

I would like to thank my friends including, Lijie Tang, Wenjing Xue, Wei Zhang, Liyang Ma, Jianli Chen, Zejing Wu and Yucheng Huang for all the fun we have in the last two years.

I would like to express my deepest gratitude to my parents, Ling Wang and Yunjian Zhang for giving birth to me at the first place and supporting me spiritually throughout my life.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The Federal Aviation Administration (FAA) is making significant improvements to the National Airspace System (NAS) by implementing a series of technology and operational changes as part of the Next Generation Air Transportation System (NextGen). In NextGen, a satellite-based surveillance system and digital communication technologies are proposed to replace the traditional ground-based surveliance system to mitigate air transportation system congestion, reduce fuel usage, ensure safety, and improve air traffic management.

An important part of NextGen, terminal operations involve streamlined procedures to get aircraft into and out of the airports safely and efficiently. However, air transportation ground delays are becoming increasingly acute at many hub airports. Improving airport surface operations is recognized as an important element of the terminal improvement productivity programs at the FAA. Multiple technologies and airport simulation models have been developed and used to improve airport surface operations.

In most airport simulation models, such as Simmod PRO! and the Airfield Delay Simulation Model (ADSIM+), an airport node-link network is used as the airport surface model to model aircraft ground movements on taxiways and runways (FAA, 2004). Developing a precise airport ground network is therefore critical for simulating airport surface operations and predicting ground delays. Creating detailed airport surface models is a time consuming endeavor requiring manual extraction of coordinates of nodes from various data sources, such as aerial photography and CAD files.

Developing, maintaining and updating detailed airport surface node-link models require significant work. Methods to automate node-link models are thus necessary and needed. In this thesis, we developed a geographic and geometric based model to create airport surface node-link networks using geographic information data from the Aeronautical Information Exchange Model (AIXM). In the case study, we developed a standalone computer application, called *Taxiway Toolkit* in ADSIM+, to automate the process.

1.2 Problem Description

As part of the NextGen effort, it is necessary to model the effects of proposed airport surface operational changes and their effect on airport capacity. The final goal of NextGen is to optimize surface operations at airports. One of the FAA in-house models to estimate airport capacity and delays is the second generation of the Airfield Delay Simulation Model (ADSIM+). ADSIM+ utilizes discrete event simulation along with user-provided airfield layouts, operations schedules and separation criteria to estimate levels of delay and airport runway capacity (Lucic, He, Adhami, & Post, 2010). In order to expand its capabilities, the FAA has undertaken an effort to improve the ADSIM+ model. Air Transportation System Laboratory (ATSL) at Virginia Tech was contracted to improve the ADSIM+ model by developing a taxiway and runway analysis toolkit.

ADSIM+ is a microscopic simulation model that simulates the aircraft movement in airport taxiways, runways and nearby airspace areas. The model requires detailed information to represent the airport runway, taxiway and gate systems. In ADSIM+, airport runways, taxiways and gates are stored as a link-node network model. Also, taxiway paths between gates and runways are built on the network model.

In the current version of the ADSIM+ model (ver_1191_1), users can configure the airport node-link network on the base of an aerial photography of the airport. Large airports have complex taxiway and runway geometry configurations (CSSI, 2009). It can take a trained individual dozens of hours to configure a single hub airport like Atlanta Hartsfield - Jackson International Airport (ATL) without considering the raw data preparation period. Furthermore, there is always uncertainty in the accuracy of a node-link airport model created manually.

To create taxiway paths in the current version of ADSIM+, users have to define gates nodes and runway exits/entries locations manually-a time consuming activity. The taxiway paths should be defined by connecting the gates nodes and runway exits/entries along the node-link network.

With the use of remote sensing, geographic information systems, and image processing technology, high-resolution aerial photography is able to assist in the automation of airport node-link model creation. The combination of all these technologies working in the airport node-link model automation is shown in Figure 1-1. The Aeronautical Information Exchange Model (AIXM) is an ICAO-based (International Civil Aviation Organization) and internationally accepted model, which contains accurate geographical features of airports (B. K. Brunk & Porosnicu, 2005). The AIXM data provided by the FAA National Flight Data Center (NFDC) represents airport taxiway, runway and apron information in polygon format, which is an ideal data source for automating airport node-link model creation in ADSIM+.

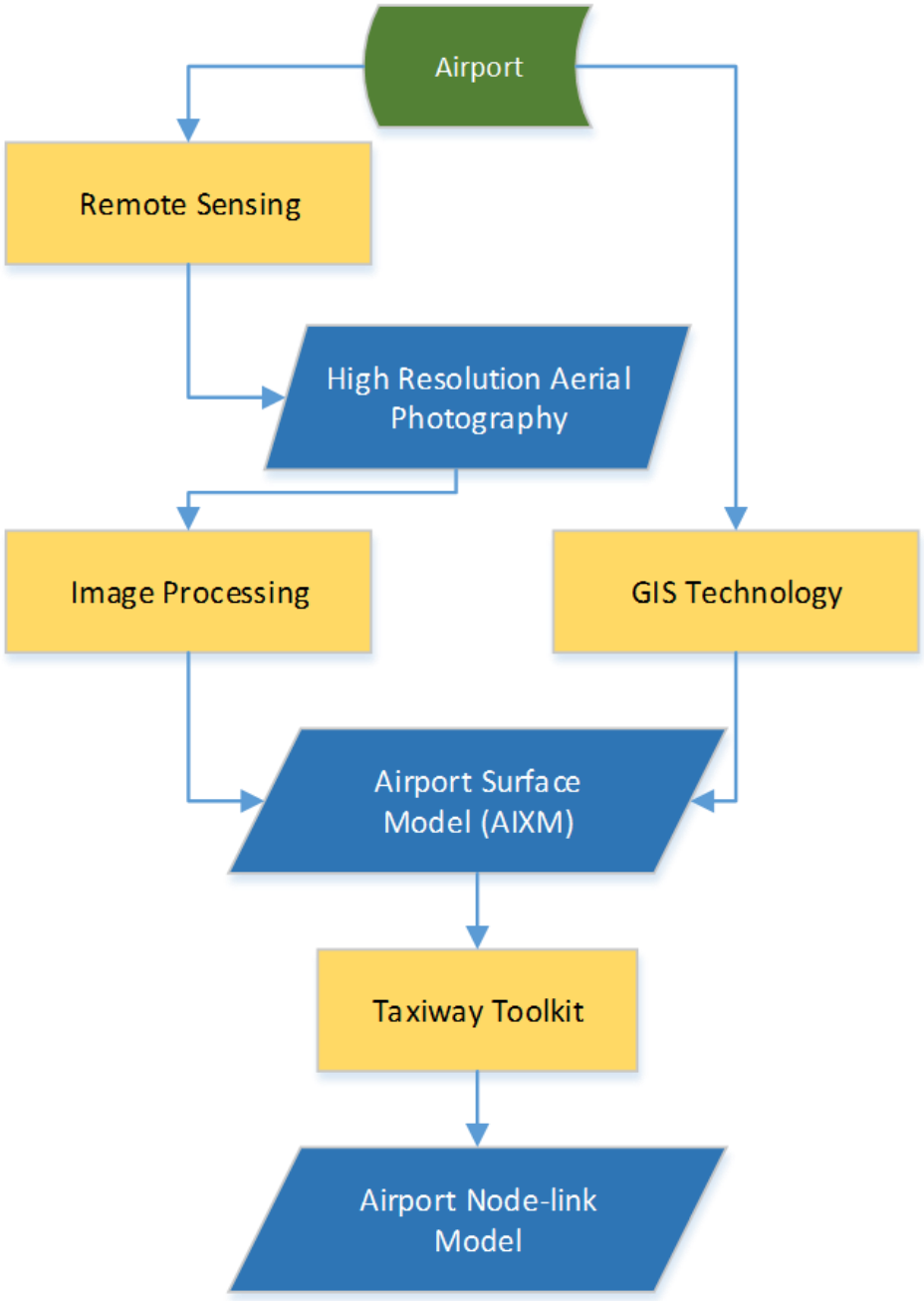

Figure 1-1 Combination of Technologies Working in the Airport Node-link Model Automation

This thesis will introduce the development of a *Taxiway Toolkit* with algorithm to:

1. Import the AIXM data of a selected airport,
2. Process the AIXM data and convert it to an airport node-link model,
3. Export the node-link model to an application software (ADSIM+) usable format file,

4. Find the shortest network path between airport gates and runway exits/entries as the taxiway paths,

The steps taken in the *Taxiway Toolkit* Development are shown in Figure 1-2.

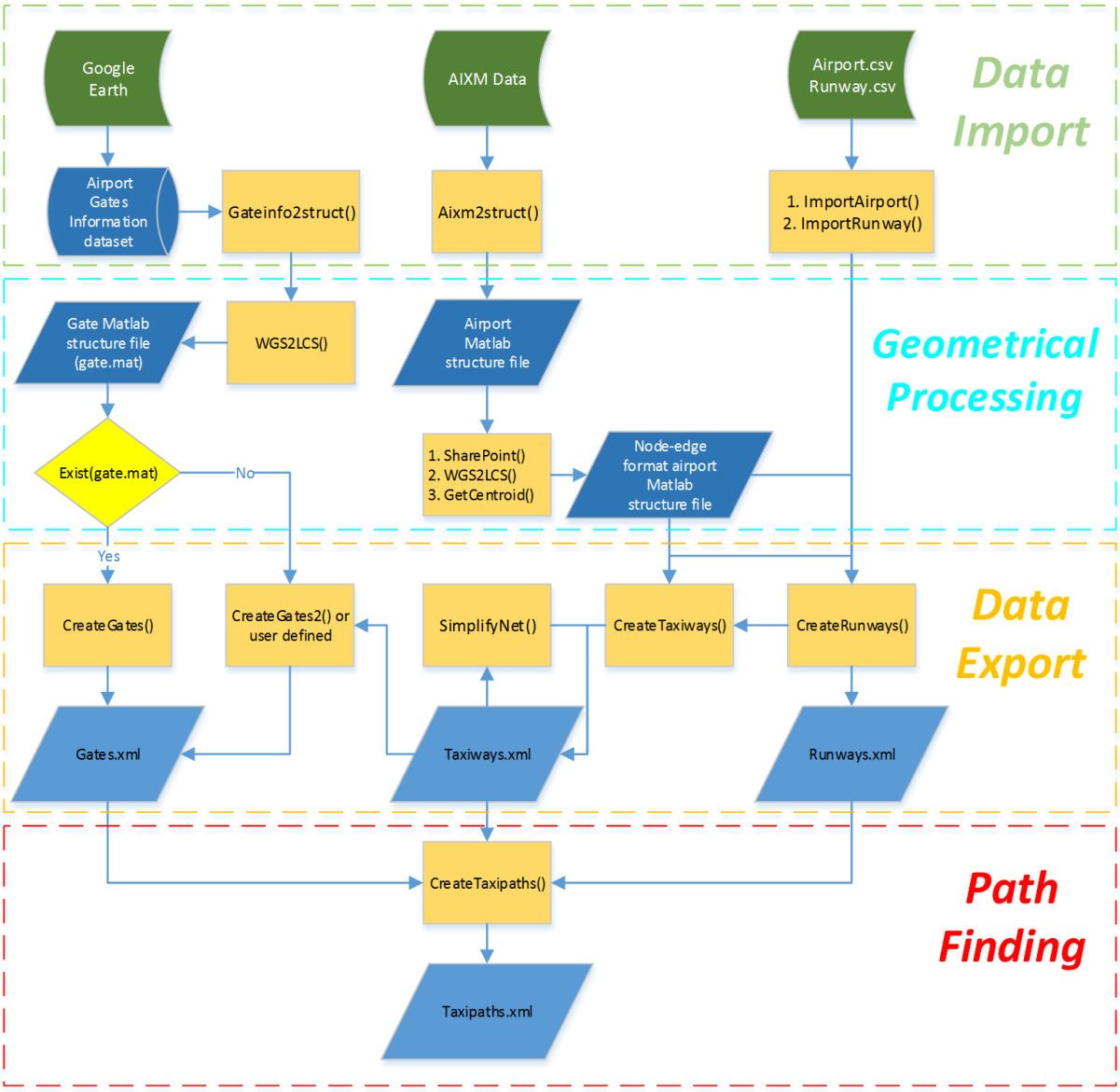

Figure 1-2 Flowchart of the Taxiway Toolkit Development

2 Literature Review

2.1 Previous Studies on Network Automation

Node-link network model automation from image data problem can be applied to many fields in engineering. Urban road network, river network and even handwriting-like image are some areas of application of node-link network models. The automation has used diverse methods depending upon the available input data. The input data includes high-resolution aerial image raster data and vector map data. The automation methods involve research areas such as topology, image processing, and Geographic Information System (GIS). Most of the previous studies on network automation used one or two technologies mentioned above which are introduced in the following section.

In early node-link network automation research, the centerlines of the original network are first recognized to represent the network. For this reason, significant node-link automation research focused on the centerline creation. Since remote sensing technology was at an early stage, high-resolution aerial image data was unavailable at that time. Network contour maps processed from low resolution image data or field survey data were used as the input. Due to the low resolution of the source image data, it is inaccurate to divide the network contour into pieces. Early on, the topology method was the preferred method applied to such input. The straight skeleton algorithm was employed to obtain polygon centerlines by moving the parallel edges angularly at same speed into the polygon (*Felkel & Obdrzalek, 1998*). Convex polygons and non-convex polygons had different algorithms. While these methods could find the network centerline with high efficiency, they brought topology errors that need to be corrected manually. The Voronoi-diagram based medial axis generalization method was introduced to extract centerlines from river network (*McAllister & Snoeyink, 2000*). This method interpolates the contour map and

constructs a Voronoi diagram to find the medial axis of a contour map. The medial axis is generalized to the centerline. Compared with other methods, the Voronoi-diagram based medial axis generalization method is more accurate than the Voronoi approximation method, centroid and midpoint methods.

Recently, the GIS and more advanced remote sensing technology are widely applied, the digital vector maps of airport surface network with accurate GIS information, such as Safe Flight 21 (SF21) (*Nolan, 2010*) data and AIXM data, are available from well pre-processed aerial image data or high accuracy GIS survey data. The network contour map can be divided into polygons with all the vertices on the edges contain accurate geographic information. The node-link network can be automated by finding and connecting the centerlines of all the polygons in the network. For regular shape polygons, the centerlines can be found by connecting the centroid of polygons. But for irregular shape polygons, specialized algorithms need to be developed. A shortest hypotenuse-based centerline generation algorithm is proposed, which constructs right triangles with the key points of polygons and connects the midpoints of hypotenuses to form the centerlines (*Pan, Chen, & Sheng, 2009*). Intersection polygons need to be processed with different algorithms. An intersection identification method was used to detect intersections of a road network in the contour map. Based on the shortest hypotenuse-based centerline generation algorithm, a principal axis-based algorithm is formulated to identify and divide complex polygons (*Lee & Romer, 2011*). In this method, the centerlines are further improved by adjusting node positions, using the known centerlines from rectangular polygons and removing redundant nodes and links.

Besides, using the vector map as input, there are some efforts to automate the node-link model using image raster data as input. Since the image data is the only input, the image processing

based method is selected to extract the node-link network. Painted road centerline markings were considered to represent the road network. The moving window method is proposed to recognize and track the road centerline markings in the high-resolution urban road aerial photography (*Kim, Park, Kim, Jeong, & Kim, 2004*). Least squares correlation matching is used in the moving window method to ensure efficiency. The searching strategy of moving window method is improved by applying a sequential similarity detection algorithm (SSDA) to speed up the matching process and reduce the noise interference from the two sides of road centerlines (*Yang & Zhu, 2010*). The existence of painted centerline markings is the prerequisite of the moving window method. For networks do not have painted centerline markings, such as river networks and airport taxiway networks, the image processing-based method is switch from the pattern recognition concerned type to the pixel thinning concerned type. In the widely used GIS software ArcMap, the procedure of extracting node-link network from image data contains the following steps: first, the image data is converted to a binary raster image that the network is filled in different color with the background. Second, the thinning tool in the ArcGIS toolbox can thin the network to a skeleton network based on different pixel brightness values. Finally, the thinned raster image is converted back to a node-link network (*Law & Collins, 2013*). This method is applied in the river centerline monitor (*Dey & Bhattacharya, 2014*). The Matlab image-processing toolbox was used to study temporal variations of width and centerline of the river Brahmaputra in its 300 km reach in the state of Assam, India. The thinning method is also used in the handwriting-like network. An improved one pass thinning algorithm (OPTA) and Hilditch model is proposed to generate a thinned binary image with better topology structure and connection of the network (*Zhu, Yang, & Zhu, 2014*).

2.2 Comparison of Previous Methods

The format of airport surface map (AIXM) is a type of vector map. The topology and GIS based method is the ideal choice for processing such data. The method proposed in this thesis is a geometric and geographic based method. This method can be evaluated using the following criteria:

1. Efficiency. The node-link extracting algorithm should complete the process quickly. Although the computed calculation ability is much stronger these days, the data itself contains much more detailed geographic information as well.
2. Accuracy. The extracted node-link network should represent the original network accurately. The accuracy has two aspects. 1) The output should be both accurate in topology structure, which means the adjacency between nodes is correct. 2) The geographic reference system should be consistent.
3. Transmissibility. Some polygon-based vector map may contain useful information to indicate the characteristics of polygons acting in the network. Some node-link network extraction methods lose such information while reconstructing the data structure. A good extraction method is expected to transmit most of the useful information from the input to the output.

In Table 2-1 the previous methods of network automation are reviewed and evaluated using the above criteria.

Table 2-1 Comparison and Evaluation of Previous Methods of Network Automation

Method	Pros	Cons
Straight skeleton algorithm (Felkel & Obdrzalek, 1998)	Efficient for network with complicated shape	Topology errors

Voronoi-diagram based medial axis generalization method (McAllister & Snoeyink, 2000)	Efficient and accurate for centerline automation for large scale area network with large variation in width	Not efficient for small scale area network with constant width
Shortest hypotenuse-based centerline generation algorithm (Pan, Chen, & Sheng, 2009)	Accurate in finding centerline; good transmissibility	Not efficient for well processed network with a great amount of information
Principal axis-based algorithm (Lee & Romer, 2011)	Accurate and efficient in automating node-link network; good transmissibility	Low efficient in dealing irregular shape polygons
Moving window method (Kim, Park, Kim, Jeong, & Kim, 2004), (Yang & Zhu, 2010)	Effective for high-resolution image road network with visible centerline; accurate in ensuring topology structure	Not applicable for image data without visible centerline
Binary image data thinning algorithm (Law & Collins, 2013), (Dey & Bhattacharya, 2014), (Zhu, Yang, & Zhu, 2014)	Efficient and accurate for regular shape network; the low resolution image can also be used for obtaining centerline	Bad transmissibility when reconstructing the data

3 Input and Output Model Description

The Aeronautical Information Exchange Model (AIXM) contains the airport surface information in vector map format. In AIXM data, airport taxiways, runways, and apron areas are stored as pieces of polygons. Each polygon contains information to indicate the location, pavement condition, and polygon type in the network, which makes the AIXM data desirable for automating an airport surface node-link model.

The node-link automation method is applied in ADSIM+ as part of the *Taxiway Toolkit*. The output of the airport surface node-link automation method is input files of the ADSIM+ model airport ground configuration. The following sections will describe the input and output relationship of the node-link model automation method.

3.1 Input: AIXM data

The AIXM is designed to enable the management and distribution of Aeronautical Information Services (AIS) data in digital format (AIXM, 2014). The AIXM version 5.1 data is written using the Geography Markup Language (GML), which is an alternative, standards-based approach to representing geometries in Extensible Markup Language (XML) developed by the Open Geospatial Consortium (B. K. Brunk & Porosnicu, 2005). The AIXM data has the following characteristics that make it an ideal input of the airport node-link network automation method.

1. The AIXM data contains the precise geographic information of the airport surface elements. Airport runways, taxiways and apron areas are stored as polygon elements. The latitude and longitude of all the vertices on the polygon boundary are contained in the AIXM data.

2. The AIXM is increasingly being adopted by military and civil government organizations and aeronautical and GIS vendors. The United States Federal Aviation Administration (FAA) has collected and converted AXIM data for 42 major hub airport in the United States, The *Taxiway Toolkit* can produce node-link models for 42 airports in the US. In the near future, with the undergoing of NextGen project, more AIXM data will be available for more airports, which makes the automation method more usable.
3. The Geography Mark-up Language (GML), used by the AIXM data, is an internationally accepted standard for exchanging geographical features using XML. All the AIXM data is written in XML, which is both human-readable and machine-readable. Several software can read and display the AIXM data, such as the Esri Aeronautical Solution for Enterprise Aeronautical Data Management, Safe Software's FME (see Figure 3-1), Snowflake Software and FAA AIXM 5 Viewer. The data is also readable for common users. The AIXM data is easy to read and maintain.

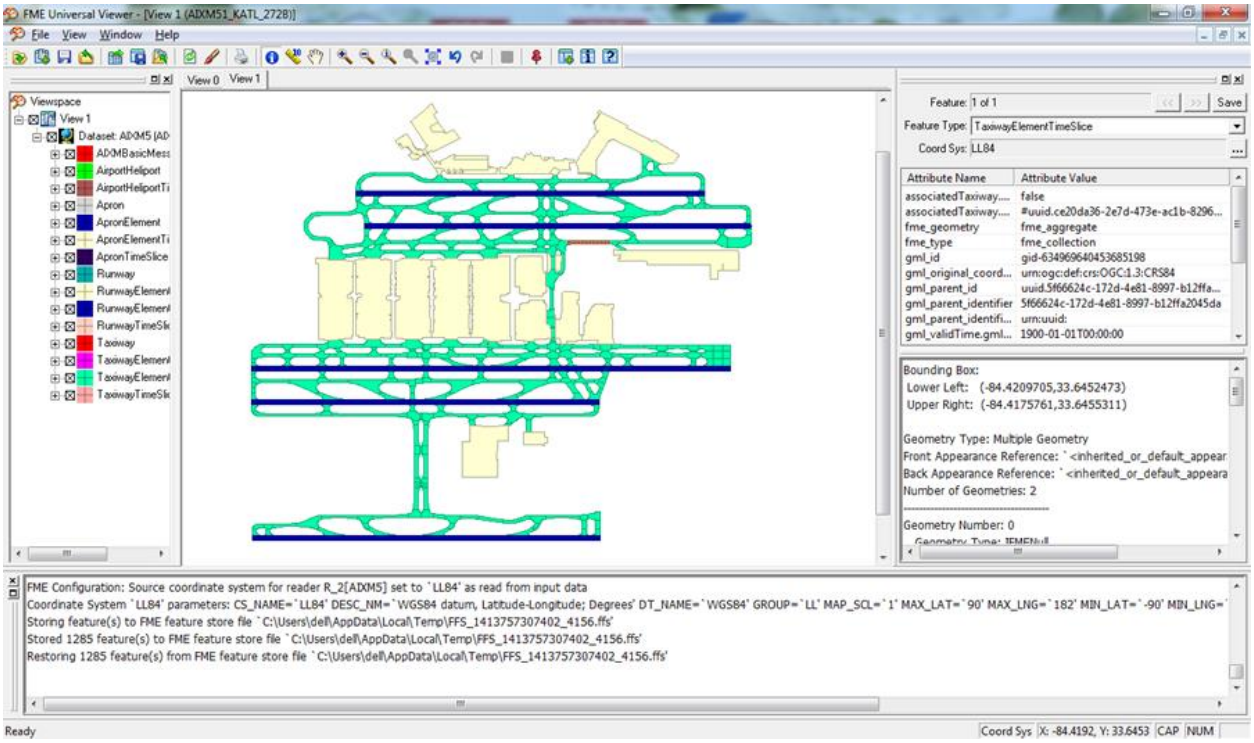

Figure 3-1 the AIXM Data of Hartsfield-Jackson Atlanta International Airport Displayed in FME

3.2 Output: ADSIM+ Input data

In the current version of ADSIM+, there are four basic input files to define the airport surface configuration, *taxiways.xml*, *runways.xml*, *gates.xml* and *taxipaths.xml*. The content of each input file is explained in the following paragraphs.

1. *taxiways.xml*: All the links, including taxiways, runways and taxilanes connecting gates and taxiways, are defined by two nodes of the link. All the nodes are labeled identically. Nodes are defined by the latitude and longitude.
2. *runways.xml*: Runways are defined by a starting node and ending node of the runway centerlines. All the exit and entrance points are also defined in the file. The node name is used to refer the node in the *taxiways.xml* file.

3. *gates.xml*: The gates in the airport are represented by nodes in the file. The node name is used to refer the node in the *taxiways.xml* file.
4. *taxipaths.xml*: The taxiway paths connecting runway exits/entrances and gates are stored in this file. The paths are defined by all the nodes along the route. The node position information is retrieved from the *taxiways.xml* by the node name.

All these files are written in Extensible Markup Language (XML). As mentioned before, these files are readable and writeable for a common user. In the current version of ADSIM+, these files are automated once the users finish manually processing the airport surface information. The *Taxiway Toolkit*, the application of node-link network automation method, will assist users to automate such files without manually entering the airport surface configuration.

4 Airport Node-Link Network Automation

4.1 Assumptions and Constraints

The geographic information in the AIXM data is represented in a Geographic Coordinate System (GCS), which means latitude and longitude are used to indicate position. The geographic information in the ADSIM+ built-in database is also represented in GCS format.

The World Geodetic System of 1984 (WGS-84) and the North American Datum of 1983 (NAD-83) are two common geodetic reference datums adopted in the GCS in North America. These two different geodetic reference datums use distinct ellipsoid models of the Earth. However, the differences in distance between these two datums for North America are not discernible within the scale of an airport.

In the AIXM airport data provided, code `srsName="urn:ogc:def:crs:OGC:1.3:CRS84"`, indicates that WGS-84 is used as the GCS system (Whiteside, 2009). According to ICAO Annex 15, all "published aeronautical geographical coordinates (indicating latitude and longitude) shall be expressed in terms of the WGS-84 geodetic reference datum", all the geographic information in ADSIM+ built-in database is assumed to use the WGS-84.

In the ADSIM+ model, all the geographic information needs to be projected to a local coordinate system (LCS), which uses the airport geometric center location in the airport database as the origin of the LCS. Easting and northing, in unit of meters, are used in the LCS instead of latitude and longitude. Thus, the Universal Transverse Mercator (UTM) projected coordinate system (PCS) is used to make the projection from GCS to PCS. The UTM zone is decided by the airport center point location in the airport database, which may lead to small inaccuracies if the airport crosses two UTM zones.

The taxiway paths between gates and runways are calculated based on the shortest distance. Dijkstra's algorithm is used to calculate all the possible taxiway paths between gates and runways. A penalty factor with a default value of 100 is assigned to a crossing runway path to prevent taxiway paths crossing runways. However, the taxiway paths finding algorithm can be optimized if the aircraft traveling characteristics on taxiways are available. All the *Taxiway Toolkit* development is performed using Matlab version 8.2.0 (R2013b) environment.

4.2 Data Import

The data import step is presented in Figure 4-1. Three different sources of data are imported into Matlab readable format: 1) the AIXM data for taxiway network, 2) ADSIM+ built-in airports and runways database for locating airport and runways, and 3) gate geographic information collected from Google Earth. This last step is optional because there is no consolidated database of airport gates publicly available.

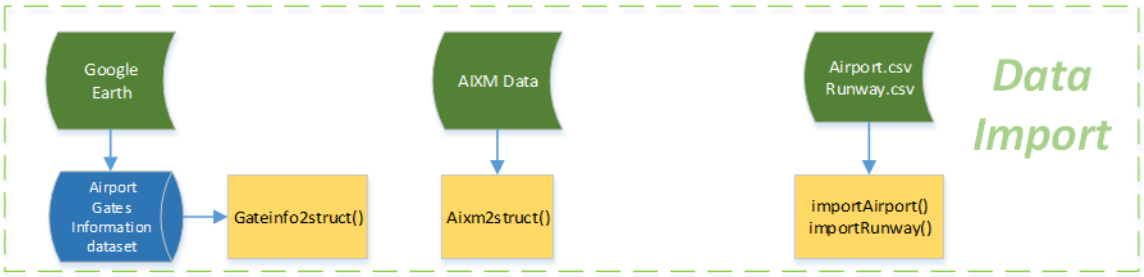

Figure 4-1 Data Import Step Flowchart

Since the AIXM data is stored in GML formart, it is possible to use Matlab built-in functions such as *xmlread ( )* to read the AIXM data. In the creation of the *Taxiway Toolkit*, a set of functions and scripts are developed to parse and reorganize the data into a Matlab struct file. In the output airport struct file, each polygon element in the airport is stored as a sub struct file. Each sub struct file contains eight fields. The fields name and their information are shown in

Table 4-1. All the functions and scripts to read the AIXM data into Matlab struct file are combined into the *aixm2struct ( )* function.

Table 4-1 Field description of output structure file of the aixm2struct ( ) function

Field Name	Type	Description
IATA	string	International Air Transport Association airport code of the airport the polygon element belongs to
ICAO	string	International Civil Aviation Organization airport code of the airport the polygon element belongs to
element_feature_type	string	Polygon element feature type, whose value can be aixm_RunwayElement, aixm_TaxiwayElement, and aixm_ApronElement
element_sub_type	string	Polygon sub type in the feature type, whose value can be NORMAL, INTERSECTION, and OTHER
element_ID	string	Polygon element geographical ID used in GML
surface_char	string	Polygon element surface pavement characteristics
label	string	Polygon element label used in actual operation
extent	double	An n*2 matrix stored the latitude and longitude of the n vertices on the polygon element edges

The ADSIM+ built-in airports and runways databases are developed in Structured Query Language (SQL). However, to speed up the *Taxiway Toolkit* development, the database files are exported into .csv files, which are easier to read. Two functions, *importAirport ( )* and *importRunway ( ),* are created to import .csv files into Matlab. The airport IATA code, ICAO code, latitude and longitude of airport geometric center are imported from *Airport.csv* via *importAirport ( )*. The runway end label, elevation, slope width and latitude and longitude of the runway end points are imported from the *Runway.csv* via function *importRunway ( ).*

Airport gate information is not available in the ADSIM+ database or the AIXM data. Unless a user has proprietary *Jeppesen* gate data, the gate data needs to be collected manually. Google Earth is an ideal source to collect gate information manually because it uses the same GCS reference system of WGS-84 as the AIXM data. To facilitate the gate data collection, a standard

format of spreadsheet is defined. The field description of the standard spreadsheet is shown in Table 4-2.

Table 4-2 Field Description of the Standard Airport Gate Information

Field Name	Type	Description
Airport	string	Airport ICAO code
Name	string	Gate name
Terminal	string	Name of the terminal the gate belonging to
Airline	string	Name of the airline using the gate
Gate Size	double	Width and length of the gate parking site
Elevation	double	Elevation of the gate
Position-lat	double	Latitude of the gate
Position-lon	double	Longitude of the gate
Sharing	string	Name of the gate sharing the same parking site
Taxilane	string	Name of the taxilane connecting the gate and the taxiway network
Intersection-lat	double	Latitude of the intersection between the gate and taxilane
Intersection-lon	double	Longitude of the intersection between the gate and taxilane

Once the gate information data has been collected using the standard spreadsheet format, the spreadsheet is processed using the function *gateinfo2struct ( )*. This function reads the .xls formatted spreadsheet into a Matlab structured file (called struct files in Matlab). The field description of the output struct file of *gateinfo2struct ( )* is the same as the standard airport gate information spreadsheet. The gate information data collection is time consuming. However, the *Taxiway Toolkit* will still work without the gate information dataset. If no gate information is available, dummy gates are constructed located at the intersections of taxiways and apron areas, are created and explained in the following steps.

4.3 Node-Link Automation

After all the raw data from either the AIXM or Google Earth has been imported into Matlab, the polygons representing the airport facilities need to be converted into a link-node network. All the geographic information of airport singular facilities uses WGS-84 as the GCS (geographic

coordinate system). ADSIM+ requires all the position data in LCS (local coordinate system). Therefore, a projection and conversion are needed.

The major steps to convert AIXM airport facilities polygon elements into a link-node network are as follows:

1. Manage coincident geometry. The AIXM data divides airport runways and taxiways facilities into polygons, which contain geographic information of the vertices of the polygon boundary. The AIXM data does not contain the adjacency relation between the polygons. The adjacency relation is critical to create a link-node airport network. Therefore, the geometric processing analysis should manage the coincident geometry to derive the adjacency relation between all polygons.

2. Develop polygon representations. When converting polygons into a link-node network, it is important to develop equivalent link-node representations, which can ensure the accuracy for simulation purposes. The centerline of polygons is an ideal representation for a network. However, it is cumbersome to find the centerline for irregular-shape polygons. Algorithms are developed to analyze such polygons.

The following sections will introduce the procedure used in the *Taxiway Toolkit* to process airport facilities data and convert polygons in GCS into an airport link-node network in LCS.

4.3.1 Coincident Geometry Management

In Euclidean geometry, *coincident* means geometries that occupy the same space. In the airport facilities AIXM data, coincident geometry means that polygon elements that share the same edge

are connected. If polygon elements are detected as coincident, there will be edges to connect them in the airport facilities link-node network.

When projecting position data from GCS into LCS, care must be taken to handle the geometric coincidence between polygons. Thus, the *Taxiway Toolkit* needs to manage coincident geometry as a first step.

In AIXM data, the polygon boundary is defined by a set of vertices on the polygon edges. The number of vertices for one polygon varies from four (a simple rectangle polygon) to hundreds (an irregular shape polygon). Since different polygons use different sets of vertices, even if connected polygons seldom shared the same vertices, the coincidence between polygons could not be detected via vertices. In the *Taxiway Toolkit*, a Matlab function called *SharePoint ( )* is developed to detect the coincidence between neighboring polygons. This is explained in the following paragraphs and shown visually in Figure 4-2.

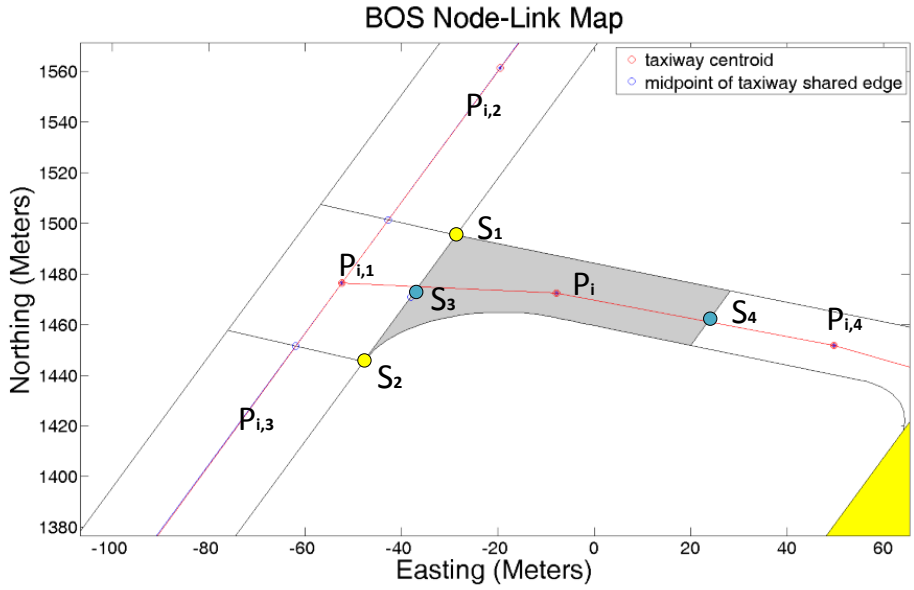

Figure 4-2 Illustration of Function SharePoint ( )

In *SharePoint ( )*, the coincidence between polygons is detected using the following steps:

1. Set a critical distance to judge if a vertex is coincident with a polygon, which means if the distance from the vertex to the polygon is less than the critical distance, the vertex is coincident with the polygon.

2. Find all the polygons ($P_{i,1}, P_{i,2}, P_{i,3}, P_{i,4}$ as shown in Figure 4-2), which have vertices coincident with polygon (P_i). Record the coincident vertices geographic information into the adjacency matrix of polygon (P_i).

3. If the number of coincident vertex between two polygons is one, which is the situation (S_1, S_2) shown in Figure 4-2, these two pairs of polygons ($P_{i,2}$ and P_i , $P_{i,3}$ and P_i) are not judged as coincident polygons.

4. If the number of coincident vertices is more than one, this set of vertices should be on the edge shared by two polygons. Find out the midpoint (S_3, S_4 in Figure 4-2) of the shared edge defined by the set of vertices. Store the midpoint geographic information to update the adjacency matrix of polygon (P_i).

5. Create a new field in the airport struct file named as *sharemid*. Store the adjacency matrix of each polygon into its *sharemid* field.

The midpoints of the shared edges can be used as nodes in the network. Also, the independent adjacency matrix of each polygon can be combined as the network adjacency matrix for path finding purposes.

4.3.2 Projection from GCS to LCS

All the geographic information is represented in the geographic coordinate system (GCS), which is not compatible with ADSIM+. ADSIM+ position data is represented in a local coordinate system (LCS), which is a converted projection coordinate system (PCS) in UTM using the airport center as the origin.

In this step, a Matlab function called *WGS2LCS ( )* is used to do the projection as follows:

1. Find the airport center geographic information in the ADSIM+ built-in database via matching the airport IATA code.

2. Detect the UTM zone the airport geographic center located at.

3. Project the geographic information of all the points in the airport struct file from WGS-84 to UTM using the UTM zone as input.

4. Use the airport center as the origin of the local coordinate system to replace the original origin of the UTM zone. Convert the geographic information in UTM to LCS and use Easting and Northing to indicate the position.

5. If the gate information file exists, the function *WGS2LCS ( )* can also do the projection of the gate information file.

4.3.3 Polygon Representation Development

When converting a network of polygon into a link-node network, the centerlines of polygons are the ideal representation of the link structure. If all the polygons are divided into regular shapes, it is easy to find the centerline by connecting the centroid of polygons. However, some polygons in

the AIXM data provided had odd shapes including C, U and S shapes. An algorithm is created to develop robust polygon representation.

In the popular GIS software, ArcGIS, the procedure to convert polygon network to link-node network can be done with the following steps:

1. Convert the polygon network into raster data, which fill the polygon with particular color.

2. Based on the different colors of polygon and background, extract polyline from polygon via thinning the width of polygon.

3. Convert the polyline raster back to polyline network.

This method is efficient and easy to use. However, when converting a polygon network into a raster file, all the information about the polygons, such as the polygon feature type and label, is lost. The output polyline network has a totally different data structure compared to the polygon network, which makes it difficult to re-match the polygon network information with the polyline network. A new algorithm is created to perform the conversion. Another feature of the algorithm is that it is applicable to irregular shape polygons.

A Matlab function called *GetCentroid ( )* is created to develop a simple polygon representation. The function finds the centroid for regular shape polygons. Furthermore, the function can find multiple centroids for irregular polygons.

The function *GetCentroid ( )* performs the following steps:

1. Adds a new field called centroid in the airport struct file to store the geographic information of the polygon centroid.

2. If the polygon is an intersection in the taxiway network, the algorithm finds the geometry centroid and uses the centroid to represent the polygon in the taxiway network. The centroid information is stored in the centroid field of the airport struct file.

3. If the polygon is not an intersection in the taxiway network and if the polygon has two adjacent polygons, the function finds the geometry centroid. The function runs a test to see if the centroid is inside the polygon. If the centroid is inside the polygon, then go to next step. If not, the function will find a projected point of centroid on the polygon centerline. The method of projecting a centroid to centerline is shown in Figures 4-3 and 4-4.

    a. Find the centroid P_0 and the symmetry axis of the U-shape polygon as shown in Figure 4-3.

    b. Draw a line perpendicular to the symmetry axis at point P_0 as shown in Figure 4-4.

    c. The perpendicular line intersects the polygon with two intersection points i_1 and i_2 . Find the midpoint P_1 of the edge i_1, i_2 , which is the projected point of centroid on the polygon centerline.

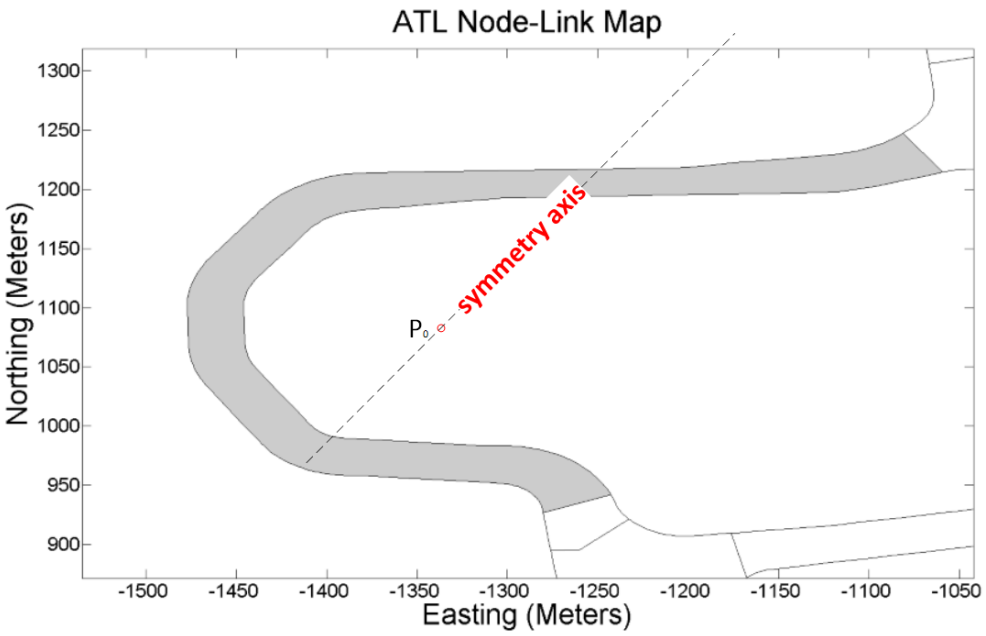

Figure 4-3 Illustration of Function GetCentroid ( ) - Step 1

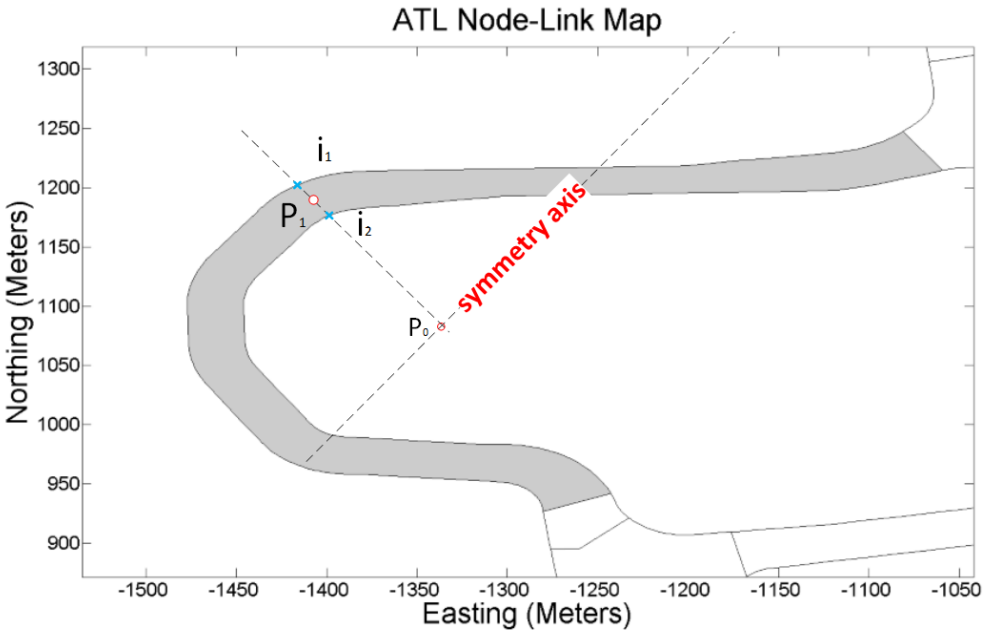

Figure 4-4 Illustration of Function GetCentroid ( ) -- Step 2

4. After projecting an outside-of-polygon centroid onto the polygon centerline, the function still needs to check if the new centroid can represent the polygon well in the network. This is to warrant that links can be created to represent a centerline. In Figure 4-5, the function connects P_1 with the shared midpoints of the neighboring polygons and creates

two edges P_1, S_1 and P_1, S_2 . Check if the edges intersect the polygon. If the edge has intersections with the polygon, edge P_1, S_1 , a sub-function called *findCent ( )* will be triggered. In *findCent ( )*, the edge formed by two intersection points i_1, i_2 between edge P_1, S_2 and the polygon will be found. Draw the perpendicular line to edge i_1, i_2 at the midpoint M_0 . The perpendicular line intersects the polygon with point i_3, i_4 . The midpoint P_2 of edge i_3, i_4 will be found and used as part of the multi centroids.

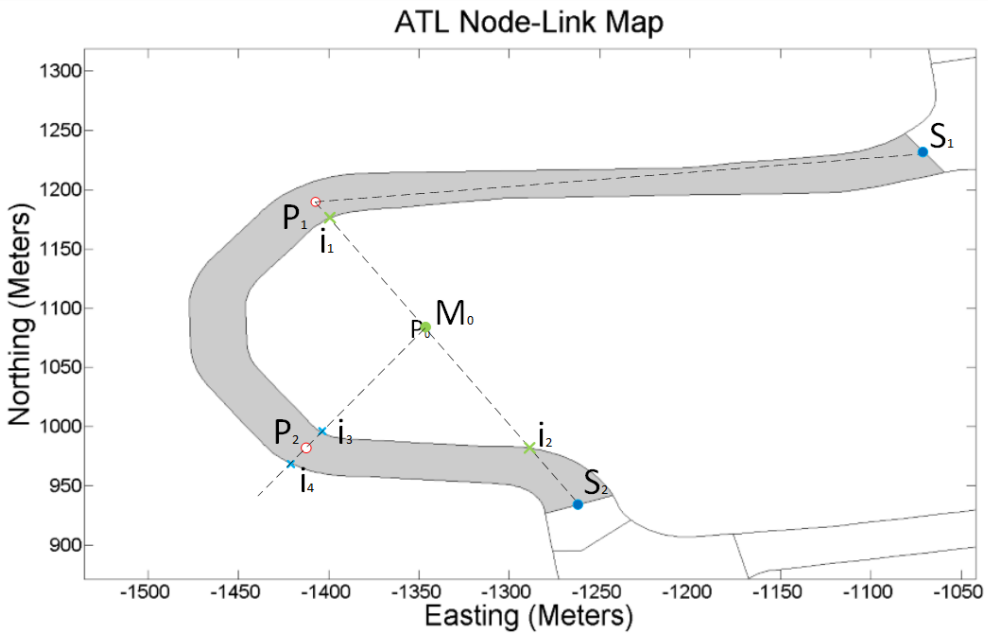

Figure 4-5 Illustration of Function GetCentroid ( ) -- Step 3

5. In *findCent ( )*, a loop runs to check all the multi centroids by connecting them with their closest right hand side and left hand side centroids or shared midpoints to see if they have intersections with the polygon. If they intersect each other, then the same procedure described in the previous step is adopted to add new centroids (P_3, P_4 as shown in Figure 4-6).

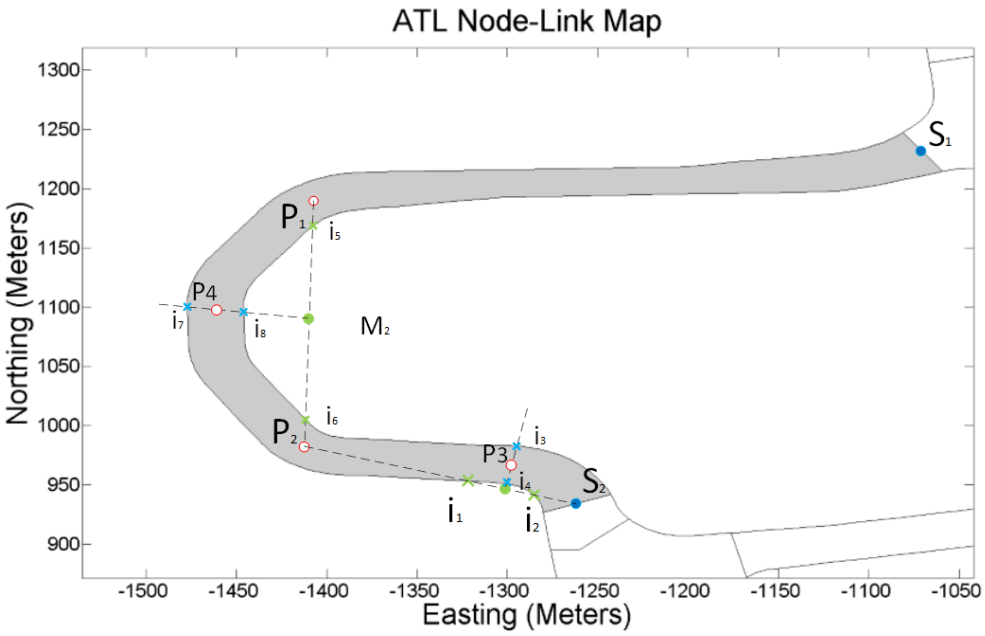

Figure 4-6: Illustration of Function GetCentroid ( ) -- Step 4

6. When all the connections between centroids and shared midpoints are checked and have no intersection with the polygon, as shown in Figure 4-7, the set of centroids will be used to represent the polygon in the link-node network.

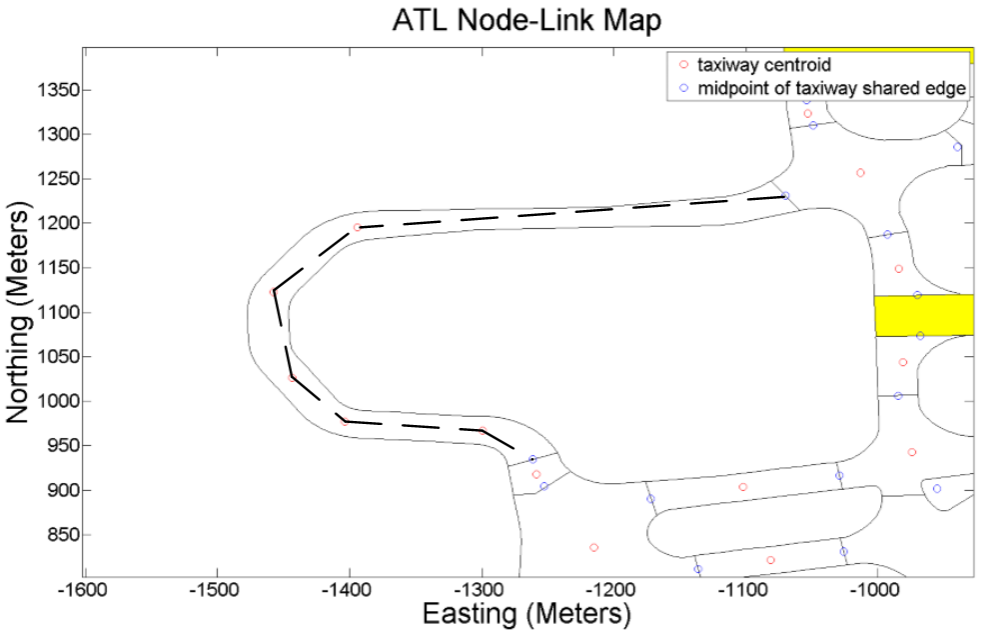

Figure 4-7: Illustration of Function GetCentroid ( ) -- Final Output

7. If the polygon type is a runway element, then it is not necessary to find the centroid of a runway. However, the function will project all the shared midpoints between the runway element polygon and the exit/entrance taxiway element polygon to the centerline of the runway polygon. All the projected shared midpoints are stored in the centroid field of the airport struct file.

4.4 Data Export

After the toolkit runs the geometric processing part, the AIXM polygon-based data is converted into a link-node model and stored in a Matlab struct file format. All these Matlab struct files need to be converted to .xml input files required by the ADSIM+ model. ADSIM+ requires four .xml input files: *taxiways.xml*, *runways.xml*, *gates.xml* and *taxipaths.xml*.

The *taxiways.xml* file contains the node and link information. The nodes can be, taxiways, runway ends and exits or gates. The file *taxiways.xml* acts as the fundamental network where the runways and gates networks are built. The taxiways.xml is created first.

4.4.1 Taxiways.xml

The taxiway input data format is provided below:

```
<TaxiWay IncludeInStudy=" " >
     <Airport></Airport>
     <Name> </Name>
     <AircraftCategories></AircraftCategories>
     <AircraftSpeedMph>
        <DistributionType></DistributionType>
        <DistributionParameters></DistributionParameters>
     </AircraftSpeedMph>
     <EndPoint Intersecting=" " Name=" " Position="" />
     <EndPoint Intersecting=" " Name=" " Position="" />
 </TaxiWay>
```

Table 4-3 ***Field Description of Taxiways.xml***

Field		Type	Value	Description	Source
IncludeInStudy		String	TRUE/FALSE	Enable or disable this taxiway in the simulation.	User define
Airport		String	Arbitrary string	The name of the "owning" airport.	Airport Matlab struct file: ICAO
Name		String	Arbitrary string	The name of this taxiway	Airport Matlab struct file: label
AircraftCategories		String		Comma separated aircraft categories	User define
AircraftSpeedMph		Distribution	mph	The transit speed on this taxiway	User define
BoundingPoint				Both normal endpoint and intersection point can have arbitrary name. The intersection points have to be globally named. That means only points with intersection type and identical name are considered to be intersected. The normal endpoint can have no name.	Airport Matlab struct file: sharemid
	Intersecting	Boolean	TRUE/FALSE	TRUE: intersection point FALSE: normal endpoint (default)	Airport Matlab struct file: element_sub_type
	Name	String			Function assign or user define
	Position		x y	Space separated coordinate of bounding point	Airport Matlab struct file: centroid

The field description and sources of each field are shown in Table 4-3. The value of fields `<Airport>, <Name>` can be found in the airport Matlab struct file. The values of fields `<AircraftCategories>, <AircraftSpeedMph>` are user defined. One taxiway link is defined by two `<EndPoint>`s. The value of `<Intersecting>` indicates the type of the `<EndPoint>`. The value of `<Name>` is the label of the `<EndPoint>` in ADSIM+. The values of `<Position>` are the x, y coordinates of `<EndPoint>` in the Local Coordinate System. The *taxiways.xml* not only contains the taxiway network, but also the runways network and gates. So the nodes of gates, runway ends and exits are also included.

The Matlab function *CreateTaxiways ( )* is developed to:

1. Read the ADSIM+ built-in runways database and find the runway end points geographic information. Find the runway exits node geographic information in the airport Matlab struct file. The function uses the geographic information and node labels to fill values of `<Position>` and `<Name>` for each `<EndPoint>` in the runway network.

2. Load the name and geographic information of nodes in the taxiway network. Create taxiway links based on the *sharemid* field in airport Matlab struct file. For multi-centroid taxiway elements, a sub-function *fDrawMulti ( )* is developed to sort the multiple centroids and make sure the inner links of such elements are compatible to the outer network.

3. Create nodes of gate via reading gate information Matlab struct file, which contains the gate name and geographic information.

To simplify the network and help speed up the simulation in ADSIM+, the Matlab function *simplifyNet ( )* can simplify the taxiway network via following procedure.

1. Gates and runway network cannot be simplified, the function only works on the taxiway network.

2. The function can measure the angle between every two connected taxiway links. If the angle is close to π ($\pm 0.05\pi$ is used as the default range), then the node connects two links will be eliminated from the network. The two links will be reconnected as one link.

3. In the network created by *CreateTaxiways ( )*, the taxiway, runway and gate networks are separated. Function *simplifyNet ( )* can create links to combine these three networks.

4. Tables containing the runway list, the simplified node link list and the adjacency matrix are created for future use. The details about the runway list variables are shown in Table 4-4.

Table 4-4 Description of Runway List Variables

Column	Type	Description
Runway label	String	Runway label actually used
Runway end	String	Runway end name
Position	1*2 double	Runway end x, y coordinates
Elevation	Double	Runway elevation
Slope	Double	Runway slope
Width	Double	Width of the runway
Index	Integer	Runway element index in airport Matlab struct file
Exit	N*1 integer	Runway exit element index in airport Matlab strut file
ICAO	String	Airport ICAO code

4.4.2 Runways.xml

After the toolkit builds the fundamental network, the runway network is constructed next. The runway input data format is provided below:

```
<RunWay IncludeInStudy=" " >
        <Airport></Airport>
        <RunWayEnd>
            <Name></Name>
            <Position></Position>
            <Elevation></Elevation>
            <GlideSlopeDegrees></GlideSlopeDegrees>
            <Exit Name="" Position="" />
                  …
            <Exit Name="" Position="" />
        </RunWayEnd>
        <RunWayEnd>
            <Name></Name>
            <Position></Position>
            <Elevation></Elevation>
            <GlideSlopeDegrees></GlideSlopeDegrees>
            <Exit Name="" Position="" />
```

```
            …
        <Exit Name="" Position="" />
    </RunWayEnd>
</RunWay>
```

Table 4-5 Field Description of Runways.xml

Field			Type	Value	Description	Source
IncludeInStudy			String	TRUE/FALSE	Enable or disable this runway in the simulation.	User define
Airport			String	Arbitrary string	The name of the "owning" airport.	Airport Matlab struct file: ICAO
RunWayEnd	Name		String	String	Valid runway end name string	Runway list: runway end
	Position		Double		X, Y Coordinate of Runway End	Runway list: position
	GlideSlopeDegrees		Double		Degrees	Runway list: slope
	Runway End Elevation		Double		Elevation	Runway list: elevation
	Exit	Name	String		Name of the exit	Node list: name
		Position	Double		X, Y Coordinate	Node list: position info

The details on the runway input data format and the source of the data are shown in Table 5. Since a runway list is created by *simplifyNet ( )*, most of the information required by *runways.xml* is already available. The Matlab function *CreateRunways ( )* reads the runway information in the runway list. Then it matches and writes the runway information into the *runways.xml* file. More details on individual fields are provided in Table 4-5.

4.4.3 Gates.xml

The gate input data format is provided below:

```
<Gate IncludeInStudy=" " >
    <Airport></Airport>
    <Name></Name>
    <Type></Type>
    <Airline></Airline>
        …
    <Airline></Airline>
    <AircraftCategory></AircraftCategory>
        …
    <AircraftCategory></AircraftCategory>
```

```
        <Capacity></Capacity>
        <Elevation></Elevation>
        <BoundingPoint Intersecting="" Name="" Position="" />
            …
        <BoundingPoint Intersecting="" Name="" Position="" />
    </Gate>
```

Table 4-6 Field Description of Gates.xml

Field		Type	Value	Description	Source
IncludeInStudy		String	TRUE/FALSE	Enable or disable a gate in simulation	User define
Airport		String	Arbitrary string	The name of the "owning" airport	Airport Matlab struct file: ICAO
Name		String	Arbitrary string	Name of this Gate	Airport Gateinfo: Name
Gate Type		Char		R- regular; G – general aviation basing area; H – holding area	User define
Airline		String	Arbitrary string	A space separated list of airlines allowed to use this gate.	User define
AircraftCategory		String	Arbitrary string	A comma separated list of aircraft categories allowed to use this gate	User define
Capacity		Integer		Maximum instantaneous number of aircraft	User define
Elevation		Decimal			Airport Gateinfo: Elevation
BoundingPoint				Both normal endpoint and intersection point can have arbitrary name. The intersection points have to be globally named. That means only points with intersection type and identical name are considered to be intersected. The normal endpoint can have no name.	

	Intersecting	Boolean	TRUE/FALSE	TRUE: intersection point FALSE: normal endpoint (default)	User define
	Name	String			Airport Gateinfo: Taxilane
	Position		x y	Space separated coordinate of bounding point	Airport Gateinfo: Position

The gate input data format requires the user to provide a set of (x, y) relative coordinates bounding the specific gate area. The airport *gateinfo* Matlab struct file can provide such information. The Matlab function *CreateGates ( )* can read the requested information from the *gateinfo* file and write it into the *gates.xml* file. However, if there is no *gateinfo* file provided as input, a function called *CreateGates2 ( )* can create a set of dummy gates. The dummy gates are the intersection of the apron area and the taxiway system. Functions *CreateGates ( )* and *CreateGates2 ( )* create the *gates.xml* file. Figure 4-8 shows the dummy gates created for Houston George Bush Intercontinental Airport (IAH).

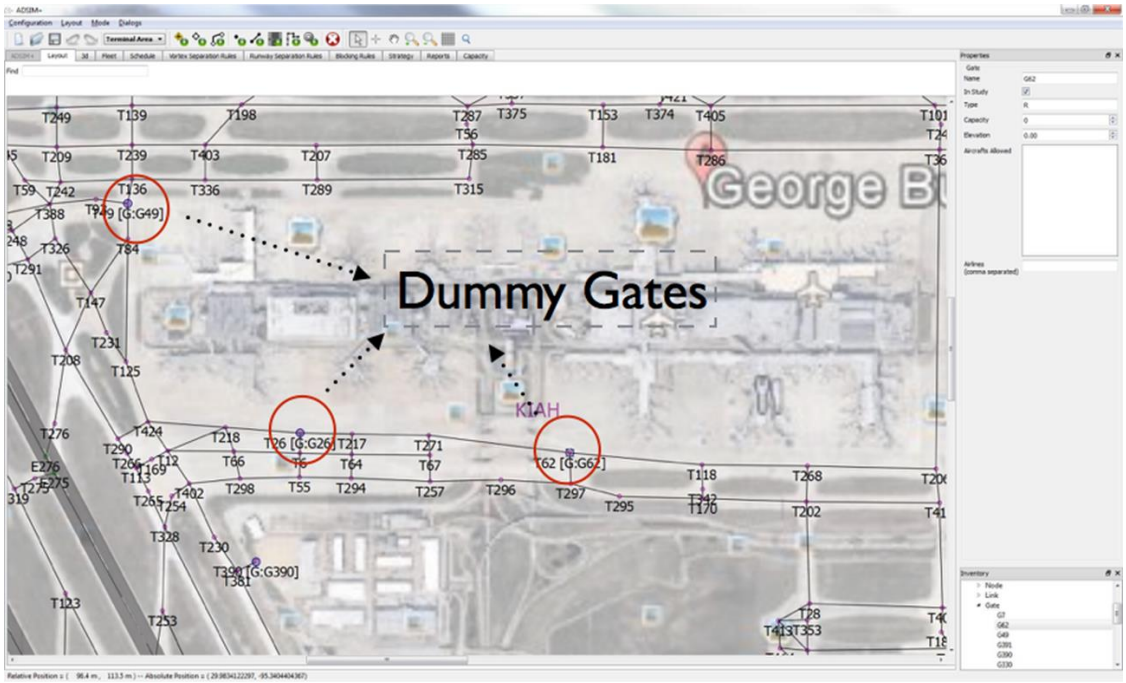

Figure 4-8 Dummy Gates at IAH Viewed in ADSIM+

The source codes of all the functions mentioned in Chapter 4 are attached in Appendces C and D.

An output of this set of functions is shown in Figures 4-9 and 4-10.

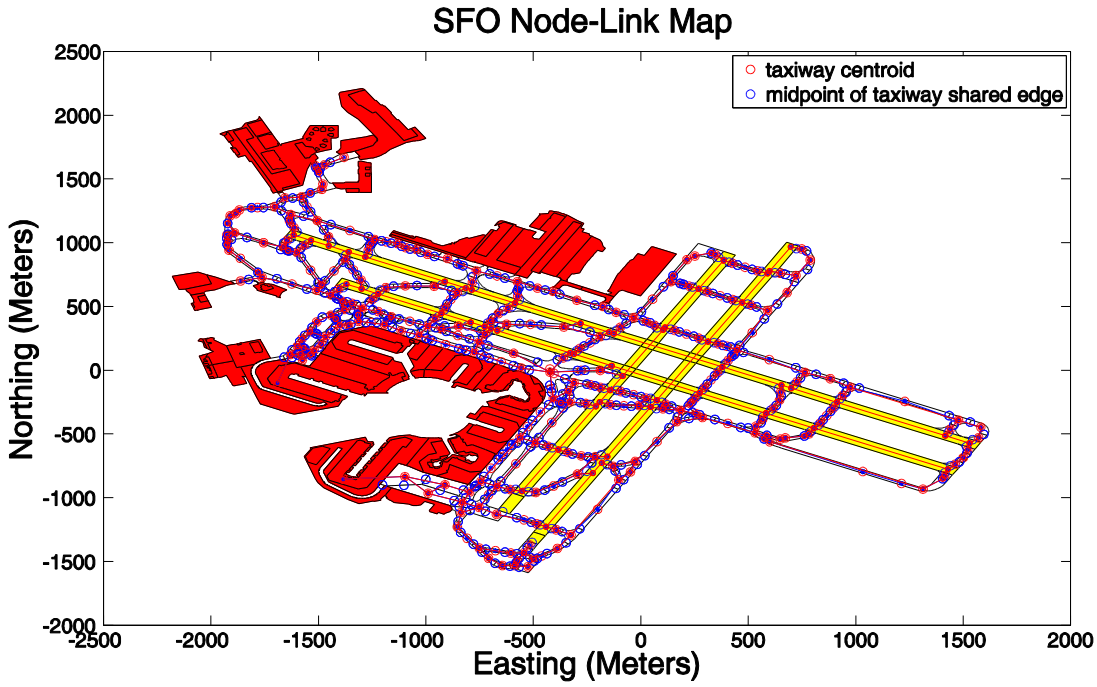

Figure 4-9 San Francisco International Airport Node-Link Network

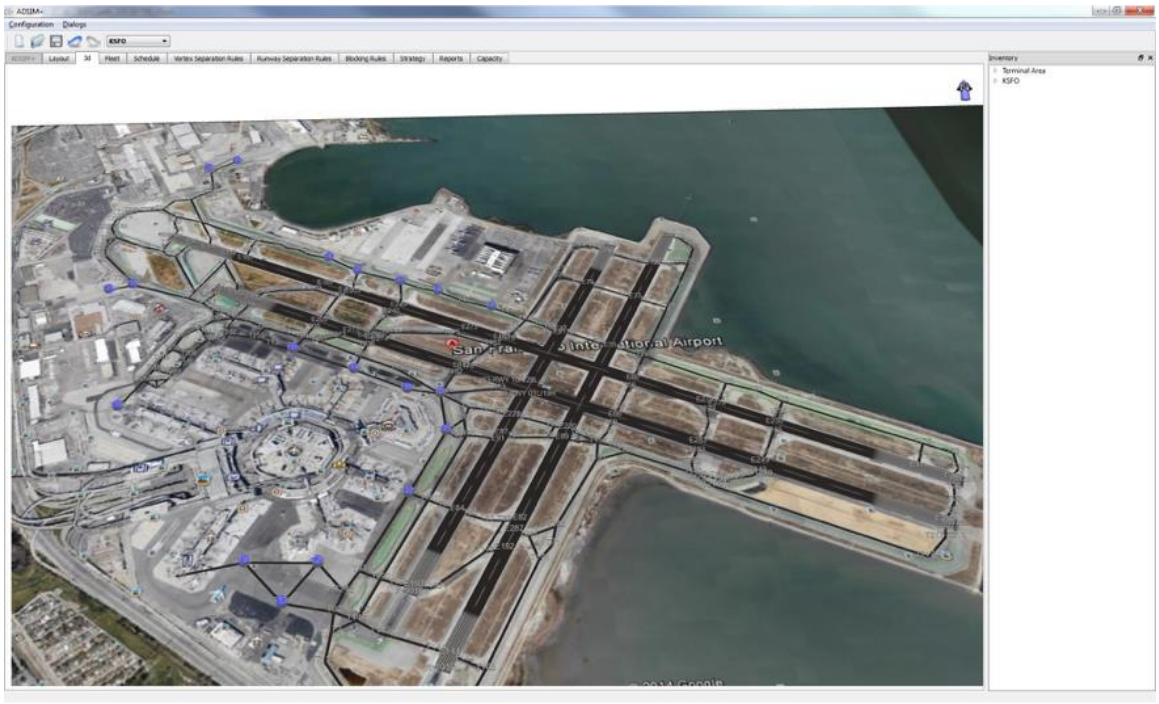

Figure 4-10 3-D View of San Francisco International Airport in ADSIM+

4.5 Path Finding on Node-Link Network

ADSIM+ requires the user to specify a path between runway entrances/exits and gates in the *taxipaths.xml* input file. A taxiway path is defined by identifying the starting and ending nodes. The name and geographic information of the nodes are included in the file *taxipaths.xml*.

The Matlab function *CreateTaxipaths ( )* is developed to find all the possible taxipaths between the runway entrances/exits and the gates. In the function, the airport runway list, the *gates.xml*, the node-link list and the adjacency matrix are used as input; the Matlab built-in function *graphshortestpath ( )* is used as the core algorithm to calculate the shortest path between all the gates and the runway entrances/exits. Since the Euclidean distance is the only available source for calculating shortest paths, all the shortest paths created by *CreateTaxipaths ( )* are shortest Euclidean distance paths.

The Matlab function *CreateTaxipaths ( )* works as follows:

1. Read the airport runway list and the *gates.xml* file to find all the possible OD (origin and destination) pairs between the gates and the runway entrances/exits.

2. Locate the origin and the destination node of each OD pair in the node-link list and the adjacency matrix. Use the information as input for the Matlab built-in function *graphshortestpath ( )* to calculate the shortest path between the origin and the destination nodes. When the function calculates the shortest path, a penalty factor with a default value of 100 is assigned to crossing a runway to prevent the shortest path crossing a runway. The output of *graphshortestpath ( )* is the set of nodes along the shortest path and the Euclidean length of the path.

3. Write the output of the Matlab function *graphshortestpath ( )* into the *taxipaths.xml* file. The sources of the *gates.xml* required information is described in Table 4-7.

Table 4-7 Field Description of Taxipaths.xml

Field	Type	Value	Description	Source
IncludeInStudy	String	TRUE/FALSE	Enable or disable this taxiway in simulation model. Default value is TRUE.	User define
Airport	String	Arbitrary string	The airport which this taxiway belongs to.	Airport Matlab struct file: ICAO
Name	String	Arbitrary string	The name of this taxipath	Function assign or user define
Direction	Character	F/B	Allowed operation direction: F --- forward (from first point to end point) B --- backward (from end point to first point)	Default
Unit	String	Arbitrary string	Unit name. The unit can be one of Runway, Gate and Taxiway.	The output of the *graphshortestpath ( )* and the airport link-node list
EndPoint	String	Arbitrary string	One of the endpoints/intersection points in the referred unit.	The output of the *graphshortestpath ( )* and the airport link-node list

4. Because the Euclidean distance of the paths works for both directions, from the gate to the runway entrances/exit and from the runway entrances/exit to the gate, the same shortest path is used for both directions. Figure 4-11 shows an example of a taxipath between the runway entrances/exit and the gate in ADSIM+ at George Bush Intercontinental Airport (IAH).

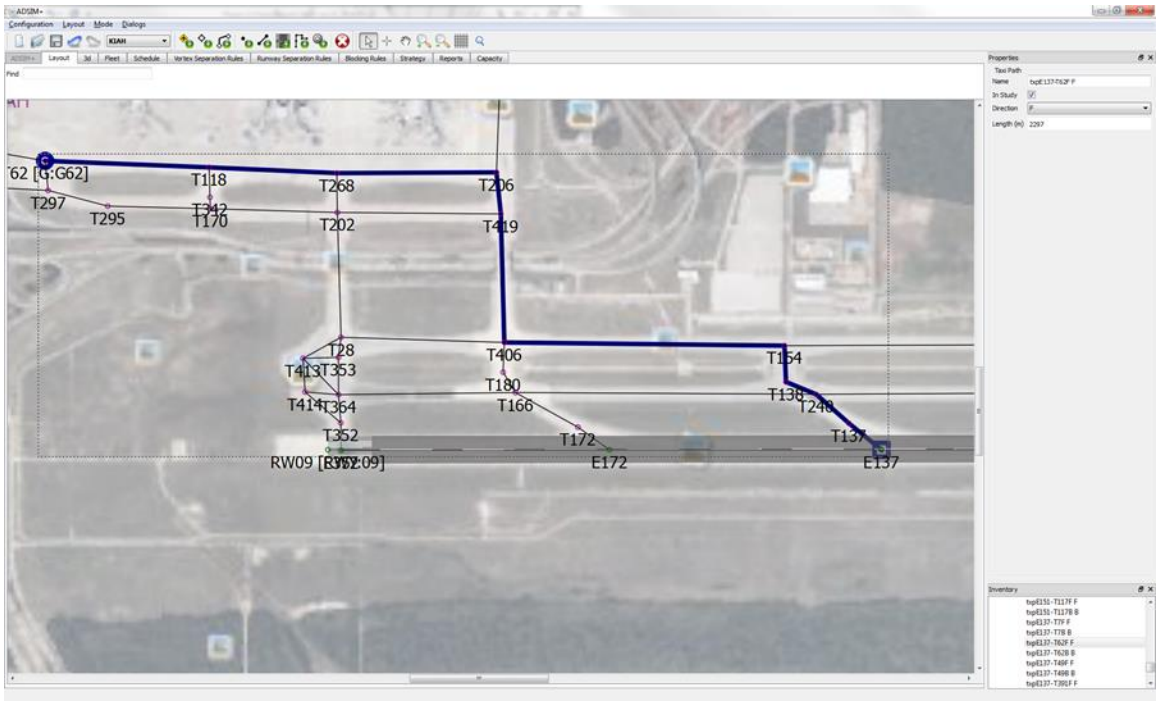

Figure 4-11 Taxipath from Runway Exit E137 to the Gate G62 at IAH Viewed in ADSIM+

5 Case Study

5.1 Taxiway Toolkit GUI

Once the node-link network automation functions and path finding functions are developed, a graphical user interface (GUI) of the ADSIM+ *Taxiway Toolkit* was created using Matlab (see Figure 5-1).

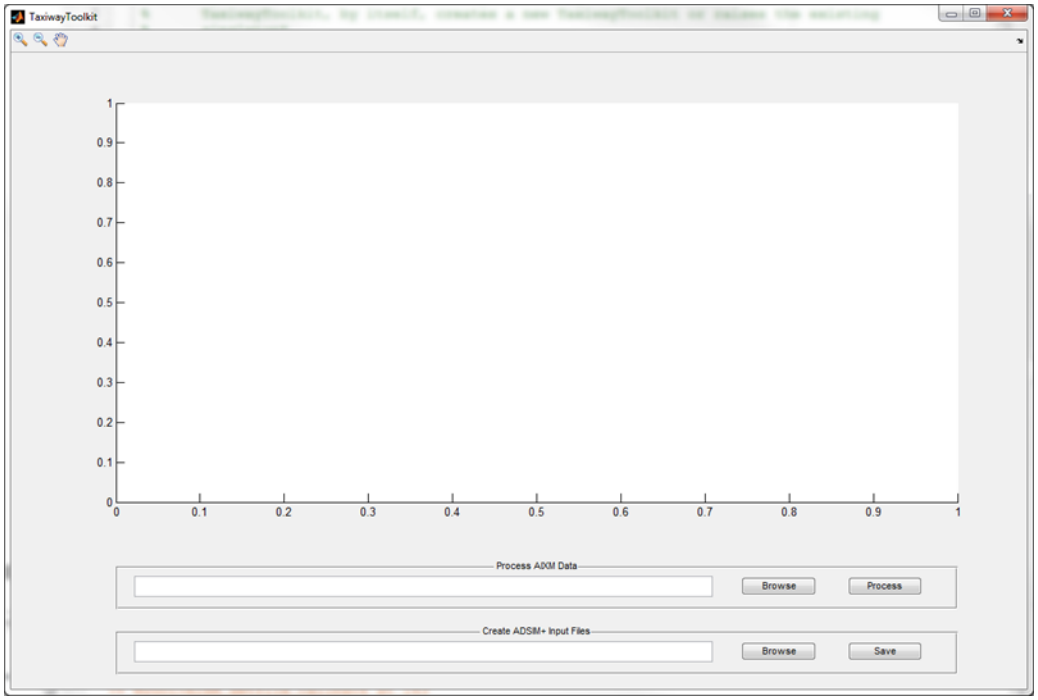

Figure 5-1 Taxiway Toolkit GUI

The GUI allows users to select an AIXM file as the starting point for analysis. The users can let the toolkit process the AIXM data and present the airport node-link map in the GUI. The users can also navigate to the ADSIM+ airport input folder to save the output of the toolkit.

The *Taxiway Toolkit* is developed and packaged on Matlab R2013b (8.2). To execute the *Taxiway Toolkit*, the computer requires the MATLAB Compiler Runtime (MCR) version 8. If the users have installed MCR version 8.2, the *Taxiway Toolkit* can be executed by launching the TaxiwayToolkit_64bit.exe (or TaxiwayToolkit_32bit.exe based on the Windows OS version)

file. If the MCR version 8.2 is not installed, the users can run the intstallation file, TaxiwayToolkit_64bit_Installer_web.exe (or TaxiwayToolkit_32bit_Installer_web.exe based on the Windows OS version) to install both the MCR version 8.2 and the *Taxiway Toolkit*. An active Internet connection is required for installation. Since ADSIM+ can only be run on the Windows operating system, the *Taxiway Toolkit* is only available for that operating system.

The data and files requirements to run the *Taxiway Toolkit* are:

1. The airport AIXM database: 42 airport AXIM data files are currently available on the FAA National Flight Data Center (NFDC) website (https://nfdc.faa.gov/amdb/).
2. ADSIM+: the users can request the ADSIM+ software from the FAA.
3. Taxiway Toolkit Installer: the installer file is available from the Air Transportation System Laboratory at Virginia Tech.

The Taxiway Toolkit GUI allows the users to:

1. Select the AIXM data of the airport to be studied;
2. Automate the airport surface node-link network model from available AIXM data;
3. Generate the taxiway paths between the airport gates and runways;
4. Export the airport node-link model and taxiway paths file to the ADSIM+ database, which allow the follow-up setting of the airport surface operation simulation.

A detailed Taxiway Toolkit GUI user manual can be found in Appendix A.

5.2 Taxiway Toolkit Output and Evaluation

The *Taxiway Toolkit* was tested using 42 AIXM files containing US airport data. These files are collected from the FAA National Flight Data Center. The airport node-link networks produced by the *Taxiway Toolkit* are appended in Appendix B. In the analysis of the airport node-link networks, we uncover some discrepancies between the AIXM data and the ADSIM+ built-in database.

1. The data of the newly opened Runway 10C/28C at Chicago O'Hare International Airport (ORD) is missing both in the AXIM data and the ADSIM+ built-in database.

2. At Fort Lauderdale-Hollywood International Airport (FLL), Runway 13/31 was permanently closed and decommissioned in 2013. The ADSIM+ built-in database already updated that. However, it still exists in the AXIM data. Runway 9R/27L has been re-designated Runway 10R/28L to correct for magnetic variation in 2014. The ADSIM+ uses Runway 10R/28L, while the AIXM data still uses Runway 9R/27L.

3. The runway displacement parts of Chicago O'Hare International Airport (ORD), General Mitchell International Airport (MKE), Minneapolis-St Paul International/Wold-Chamberlain Airport (MSP), Chicago Midway International Airport (MDW), Bradley International Airport (BDL) and Nashville International Airport (BNA) are missing in the ADSIM+ built-in database.

The processing speed of the *Taxiway Toolkit* at 42 airports was tested and recorded using an Intel Core i7-4770 Quad-Core CPU (@3.40 GHz) desktop. The run time of the toolkit at each airport

varies from 0.35 to 8.02 minutes. The top 10 airports in run time test are shown in Table 5-3. The number of nodes and links in the ADSIM+ node-link model of each airport are also shown.

Table 5-1 Top 10 Airports Run Time Statistics

Airport	Run Time (S)	Run Time (min)	Number of Nodes	Number of Links
KDFW	481.14	8.02	746	1230
KDEN	356.53	5.94	378	500
KABQ	280.65	4.68	219	273
KLAX	246.93	4.12	322	419
KIAH	244.33	4.07	341	486
KORD	227.20	3.79	556	747
KATL	193.44	3.22	357	469
KDTW	187.91	3.13	390	502
KMIA	174.97	2.92	311	455
KJFK	166.49	2.77	379	466

The run time of most airport is within an acceptable range, less than 5 minutes (Figure 5-2).

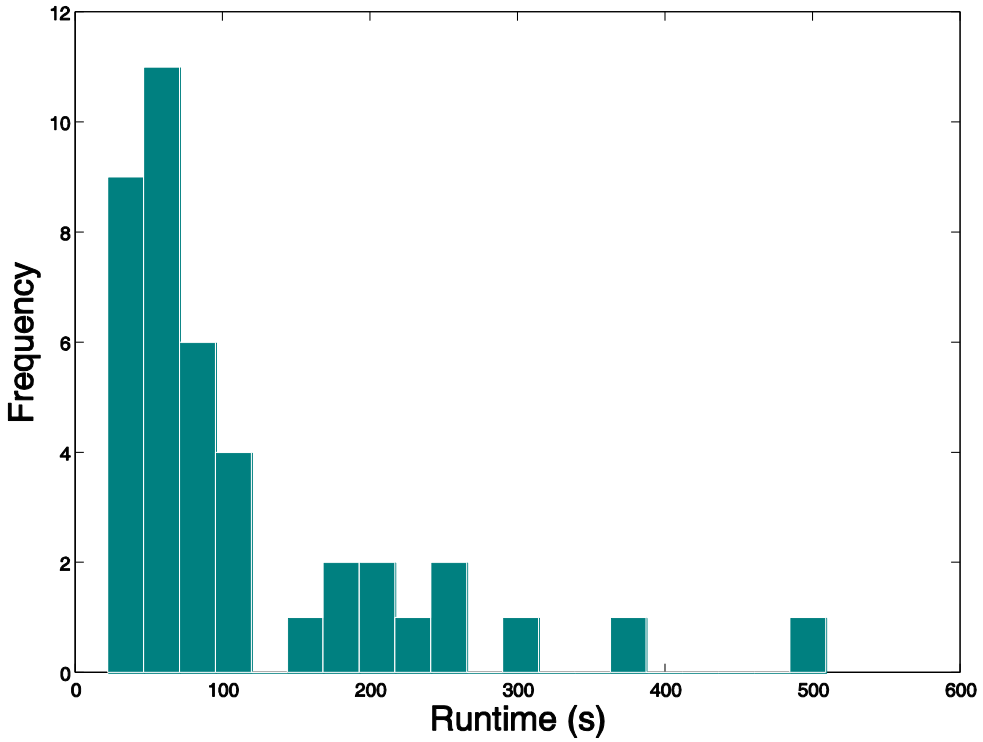

Figure 5-2 Taxiway Toolkit GUI Runtime Test Results for 42 US Airports

6 Conclusions and Future Research

This thesis developed methods to automate the creation of airport node-link network models. A number of Matlab functions are developed to extract the centerlines from the polygons of an airport layout contained in AIXM files. These functions apply geometric and geographic algorithms to manage the geometry coincidence relations, geographic reference system and polygon representation. An application based on the Matlab functions, called *Taxiway Toolkit*, was created to make the method friendly to users who have access to the AIXM airport data. In our testing, the method works well with data for 42 US airports. The run times to process the airport data were acceptable. The result indicates that the methods developed are accurate and efficient. Also, all the information, such as geographical ID used in GML, surface pavement characteristics and the geographic information, is successfully converted from the AIXM data to the node-link network model. Therefore, the geometric and geographic automation method proposed in the study meet the transmissibility criteria mentioned in Chapter 2 quite well.

The output of the *Taxiway Toolkit* can be used as the airport surface network for ADSIM+ to run airport simulations. Also, the taxiway paths are created using the *Taxiway Toolkit*, which can save days of work for users. The precise airport surface node-link network model can help expedite simulation runs. As the FAA generates more AIXM data for medium and small size airports, the *Taxiway Toolkit* will enable ADSIM+ to simulate and analyze more airports efficiently.

Furthermore, the geometric and geographic node-link network model automation method can be not only applied on the airport surface data, it also can be used on urban or highway traffic network data. Most transportation simulation software, such as Vissim, Synchro and Cube, require a node-link network for the road network. The predefined traffic paths are also desirable.

The AIXM data is a preprocessed high-resolution image data. If similar data is available for a road network, the core algorithms developed will work well to automate node-link network model for road networks as well.

From the development of the geometric and geographic automation method, a number of improvements became evident that might be investigated in future work. This will enable the airport simulation model, both airport surface or ground road network, to be more useful.

1. In the case study, the taxiway paths are generated based on the shortest Euclidean distance. However, in real airport operations, the shortest travel time paths are more commonly used than the shortest Euclidean distance paths. In future work, the traffic track data, such as PDARS/ASDE-X data for aircraft, may be used to evaluate the travel time of each link of the network. With such input, the shortest travel time paths can be calculated.
2. The bidirectional paths from certain origin to certain destination are the same in the current output. However, if the analysis of aircraft or vehicle traveling behavior is available, the bidirectional travel time of one link may be different. Therefore, the bidirectional paths from a certain origin to a certain destination may be different.
3. Different aircraft or vehicle types share the same taxiway path in the current output. Since the PDARS/ASDE-X data specifies the aircraft type, the traveling behavior of different aircraft type can be analyzed. Different types of aircraft may show distinct behaviors on the same links. Therefore, in future work, the path output would be based on aircraft or vehicle type.

Since the airport node-link model is available, aircraft travel characteristics studies based on node-link network can be conducted in the future. With the help of the aircraft track data,

PDARS/ASDE-X data, we could formulate aircraft traveling “laws” similar to the traffic flow theories used in ground transportation. For example, on aircraft following model could consider aircraft speed, traffic volume, density or even the taxiway length and width as factors. This research should be helpful at improving air traffic simulation models and understanding airport ground delays.

References

AIXM. (2014). Aeronautical Information Exchange Model: Home. . Retrieved July 1, 2014, from http://www.aixm.aero/public/subsite_homepage/homepage.html

Baik, H. (2000). *Development of Optimization and Simulation Models for the Analysis of Airfield Operations.* Virginia Tech. (Generic)

Brunk, B. (2008). *Introduction to AIXM.* Paper presented at the Singapore Workshop-Technical Focus-16 June.

Brunk, B., & Porosnicu, E. (2006). Aeronautical Information Exchange Model (AIXM).

Brunk, B. K., & Porosnicu, E. (2005). *Aeronautical Information Exchange Model (AIXM) GIS interoperability through GML.* Paper presented at the Proceedings of the Twenty-Fifth Annual ESRI User Conference.

Chris, B., Jimmy, K., Brian, C., & Stephen, A. (2002). Improved Taxi Prediction Algorithms for the Surface Management System *AIAA Guidance, Navigation, and Control Conference and Exhibit*: American Institute of Aeronautics and Astronautics.

CSSI. (2009). (Advanced) Airfield Delay Simulation Model (ADSIM+) User's Manual.

Dey, A., & Bhattacharya, R. (2014). Monitoring of River Center Line and Width—A Study on River Brahmaputra. *Journal of the Indian Society of Remote Sensing, 42*(2), 475-482.

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization, 10*(2), 112-122. doi: 10.3138/FM57-6770-U75U-7727

FAA. (2004). How Simmod Works (SIMMOD Manual). from http://www.tc.faa.gov/acb300/how_simmod_works.pdf

Felkel, P., & Obdrzalek, S. (1998). *Straight Skeleton Implementation.* Paper presented at the Proceedings of Spring Conference on Computer Graphics.

Hong, S., Heo, J., & Vonderohe, A. P. (2013). Simulation-based Approach for Uncertainty Assessment: Integrating GPS and GIS. *Transportation Research Part C: Emerging Technologies, 36*(0), 125-137. doi: http://dx.doi.org/10.1016/j.trc.2013.08.008

Kim, T., Park, S.-R., Kim, M.-G., Jeong, S., & Kim, K.-O. (2004). Tracking Road Centerlines from High Resolution Remote Sensing Images by Least Squares Correlation Matching. *Photogrammetric Engineering & Remote Sensing, 70*(12), 1417-1422. doi: 10.14358/PERS.70.12.1417

Lacoste, C., Descombes, X., & Zerubia, J. (2010). Unsupervised Line Network Extraction in Remote Sensing using a Polyline Process. *Pattern Recognition, 43*(4), 1631-1641. doi: http://dx.doi.org/10.1016/j.patcog.2009.11.003

Law, M., & Collins, A. (2013). *Getting to know ArcGIS for desktop*: Esri Press.

Lee, H.-T., & Romer, T. F. (2011). Automating the Process of Terminal Area Node-Link Model Generation. *Journal of Guidance, Control, and Dynamics, 34*(4), 1228-1237. doi: 10.2514/1.49184

Li, J., Qin, Q., Xie, C., & Zhao, Y. (2012). Integrated Use of Spatial and Semantic Relationships for Extracting Road Networks from Floating Car Data. *International Journal of Applied Earth Observation and Geoinformation, 19*(0), 238-247. doi: http://dx.doi.org/10.1016/j.jag.2012.05.013

Liu, Y., Hansen, M., Gupta, G., Malik, W., & Jung, Y. (2014). Predictability Impacts of Airport Surface Automation. *Transportation Research Part C: Emerging Technologies, 44*(0), 128-145. doi: http://dx.doi.org/10.1016/j.trc.2014.03.010

Lucic, P., He, D., Adhami, L., & Post, J. (2010). *ADSIM+: A New Development of the Airfield Delay Simulation Model*. Paper presented at the 27th International Congress of the Aeronautical Sciences, Nice, France.

Manley, B., & Sherry, L. (2010). Analysis of Performance and Equity in Ground Delay Programs. *Transportation Research Part C: Emerging Technologies, 18*(6), 910-920. doi: http://dx.doi.org/10.1016/j.trc.2010.03.009

Mansourian, A., Valadan Zoje, M. J., Mohammadzadeh, A., & Farnaghi, M. (2008). Design and Implementation of an On-Demand Feature Extraction Web Service to Facilitate Development of Spatial Data Infrastructures. *Computers, Environment and Urban Systems, 32*(5), 377-385. doi: http://dx.doi.org/10.1016/j.compenvurbsys.2008.07.002

McAllister, M., & Snoeyink, J. (2000). Medial Axis Generalization of River Networks. *Cartography and Geographic Information Science, 27*(2), 129-138. doi: 10.1559/152304000783547966

Mena, J. B. (2003). State of the Art on Automatic Road Extraction for GIS Update: a Novel Classification. *Pattern Recognition Letters, 24*(16), 3037-3058. doi: http://dx.doi.org/10.1016/S0167-8655(03)00164-8

Mokhtarzade, M., & Zoej, M. J. V. (2007). Road Detection from High-Resolution Satellite Images Using Artificial Neural Networks. *International Journal of Applied Earth Observation and Geoinformation, 9*(1), 32-40. doi: http://dx.doi.org/10.1016/j.jag.2006.05.001

Mosaic ATM, I. (2013). *SODAA User's Guide Version 3.0.0*. Leesburg, VA.

Niu, X. (2006). A Semi-Automatic Framework for Highway Extraction and Vehicle Detection Based on a Geometric Deformable Model. *ISPRS Journal of Photogrammetry and Remote Sensing, 61*(3–4), 170-186. doi: http://dx.doi.org/10.1016/j.isprsjprs.2006.08.004

Nolan, M. (2010). *Fundamentals of air traffic control*: Cengage Learning.

Pan, Y., Chen, W., & Sheng, Y. (2009). The Shortest Hypotenuse-Based Centerline Generation Algorithm. *IJCSNS International Journal of Computer Science and Network Security, 9*, 155-159.

Plans, U. D. o. T. F. A. A. F. P. a. (2013). Terminal Area Forecast Summary Fiscal Years 2013–2040.

Poullis, C., & You, S. (2010). Delineation and Geometric Modeling of Road Networks. *ISPRS Journal of Photogrammetry and Remote Sensing, 65*(2), 165-181. doi: http://dx.doi.org/10.1016/j.isprsjprs.2009.10.004

Ramer, U. (1972). An Iterative Procedure For the Polygonal Approximation of Plane Curves. *Computer Graphics and Image Processing, 1*(3), 244-256. doi: http://dx.doi.org/10.1016/S0146-664X(72)80017-0

Robert, W., & Larry, M. (2007). The Airspace Concepts Evaluation System Terminal Area Plant Model *AIAA Modeling and Simulation Technologies Conference and Exhibit*: American Institute of Aeronautics and Astronautics.

Robinson, D. D. (2009). *GIS for Airports–Electronic Terrain and Obstacle Data Model.* (Master), University of Redlands. Retrieved from http://inspire.redlands.edu/gis_gradproj/135

Sarantis, D., & Xydeas, C. S. (1996). A Model-Based Approach for the Detection of Airport Transportation Networks in Sequences of Aerial Images. In B. G. Mertzios & P. Liatsis (Eds.), *Proceedings IWISP '96* (pp. 657-660). Oxford: Elsevier Science Ltd.

Surface Management System (SMS) Adaptation Procedures Manual (2004).

Thomas, F. (1998). Generating Street Center-Lines from Inaccurate Vector City Maps. *Cartography and Geographic Information Systems, 25*(4), 221-230.

Valero, S., Chanussot, J., Benediktsson, J. A., Talbot, H., & Waske, B. (2010). Advanced Directional Mathematical Morphology for the Detection of the Road Network in very High Resolution Remote Sensing Images. *Pattern Recognition Letters, 31*(10), 1120-1127. doi: http://dx.doi.org/10.1016/j.patrec.2009.12.018

Whiteside, A. (2009). Definition identifier URNs in OGC namespace. *OpenGIS Best Practice document, OGC*.

Yang, Y., & Zhu, C. (2010). Extracting Road Centrelines from High-Resolution Satellite Images Using Active Window Line Segment Matching and Improved SSDA. *International Journal of Remote Sensing, 31*(9), 2457-2469. doi: 10.1080/01431160903019288

Zhu, S., Yang, J., & Zhu, X. F. (2014) A Thinning Model for Handwriting-Like Image Skeleton. *Vol. 277 LNEE. Lecture Notes in Electrical Engineering* (pp. 687-693).

Appendix

A. Taxiway Toolkit GUI User Manual

The users of the *Taxiway Toolkit* should have ADSIM+ installed first. If ADSIM+ is installed, users can use the *Taxiway Toolkit* to generate input files for ADSIM+. Users can use the toolkit as follows:

1. Open ADSIM+ to create an airport folder first (Figures A-1, A-2).

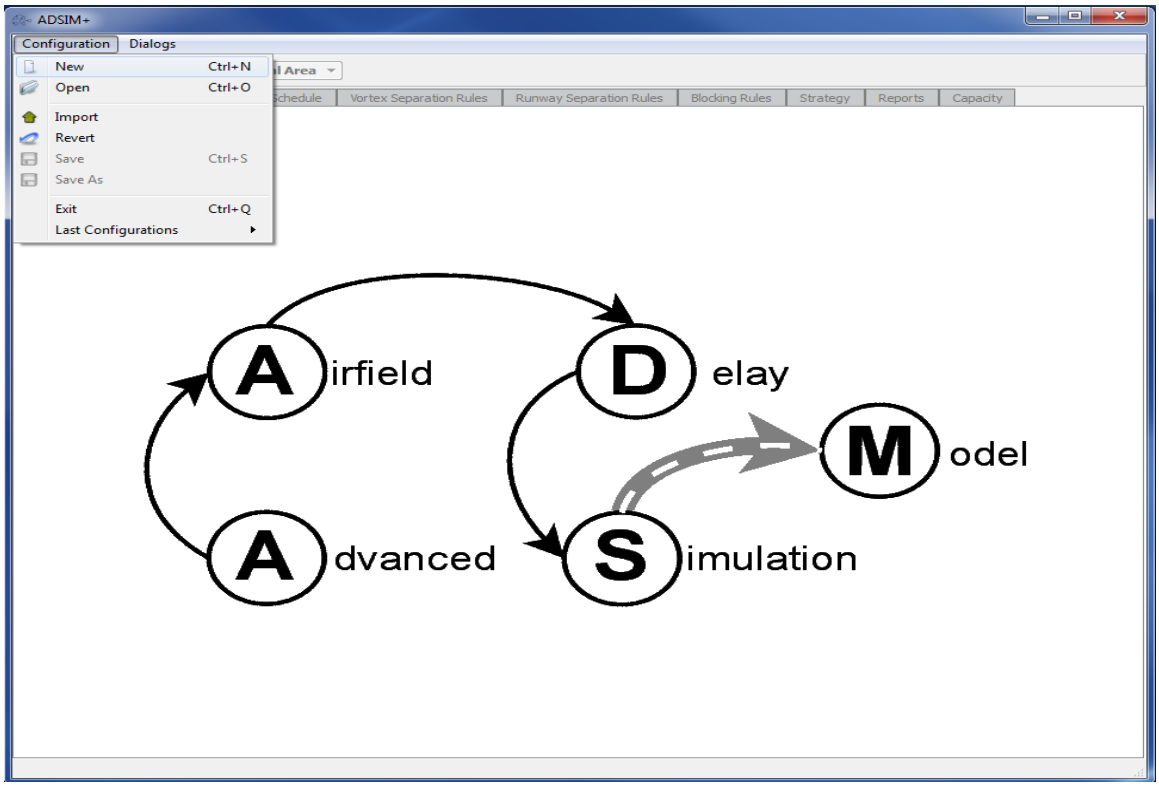

Figure A-1: Create New an Airport Folder in ADSIM+ (Source: FAA)

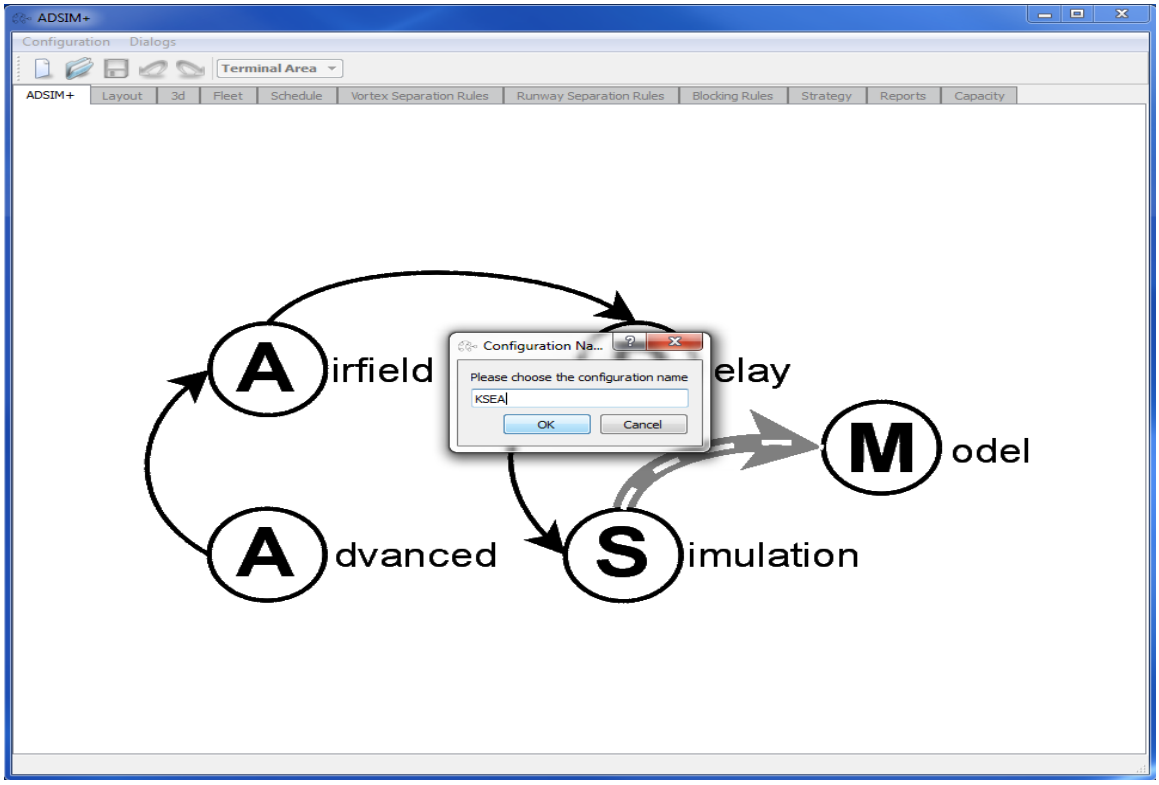

Figure A-2: Name the Airport Folder in ADSIM+ (Source: FAA)

2. Import and save the basic airport settings to the airport folder (Figures A-3, A-4).

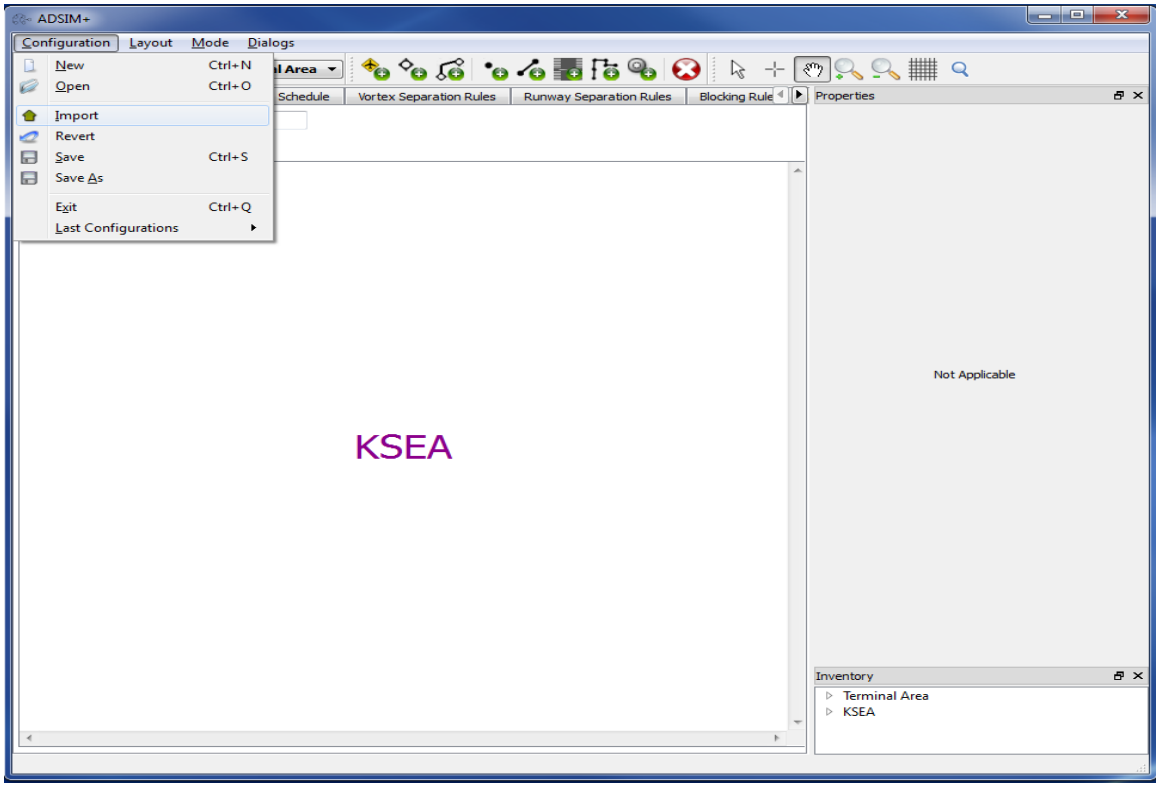

Figure A-3: Import Basic Settings for the Airport Created in ADSIM+ (Source: FAA)

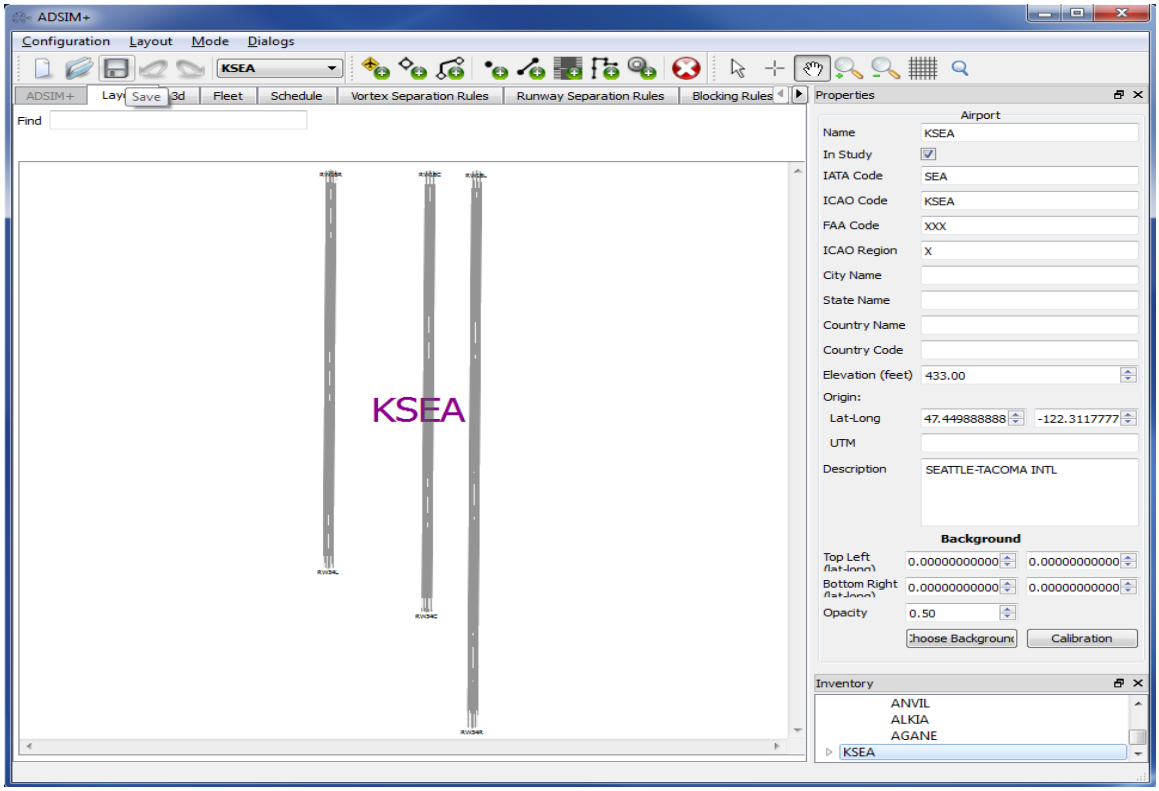

Figure A-4: Save the New Airport (Source: FAA)

3. Open the *Taxiway Toolkit* and click the 'Browse' button in the panel 'Process AIXM Data' and navigate to the AIXM file of the airport created in ADSIM+ (Figures A-5, A-6).

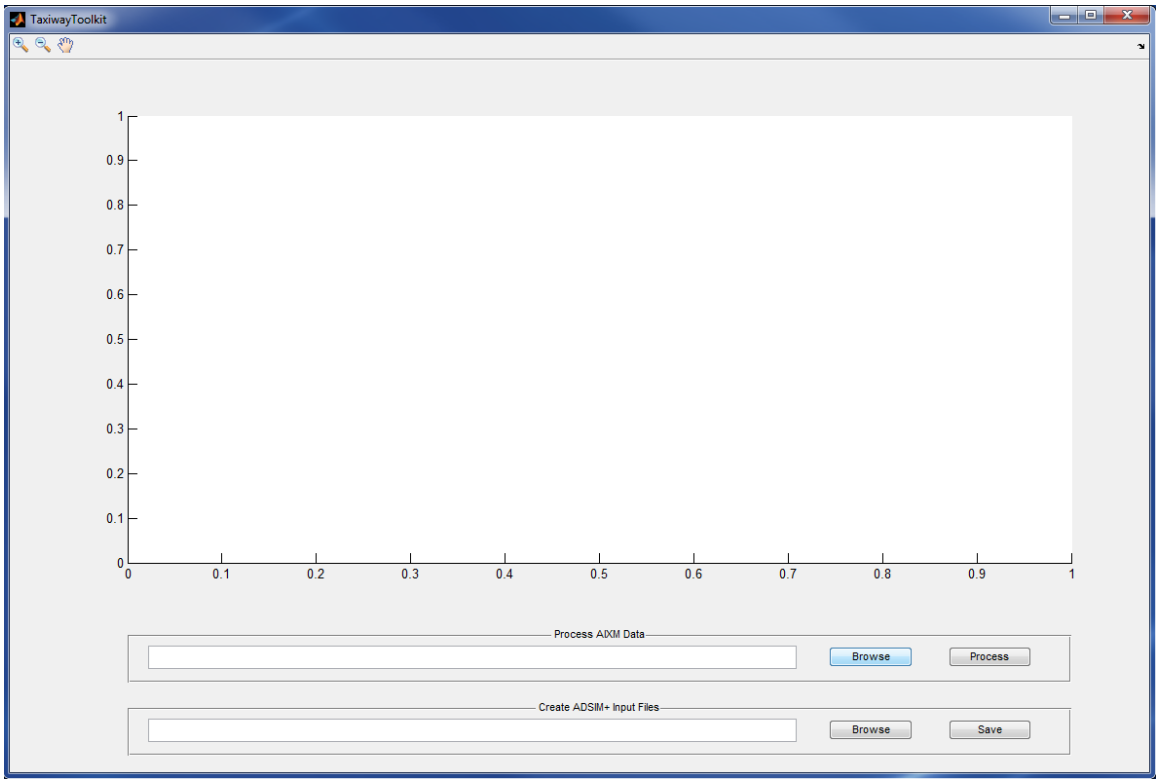

Figure A-5: Browse to Navigate the Airport AIXM Data

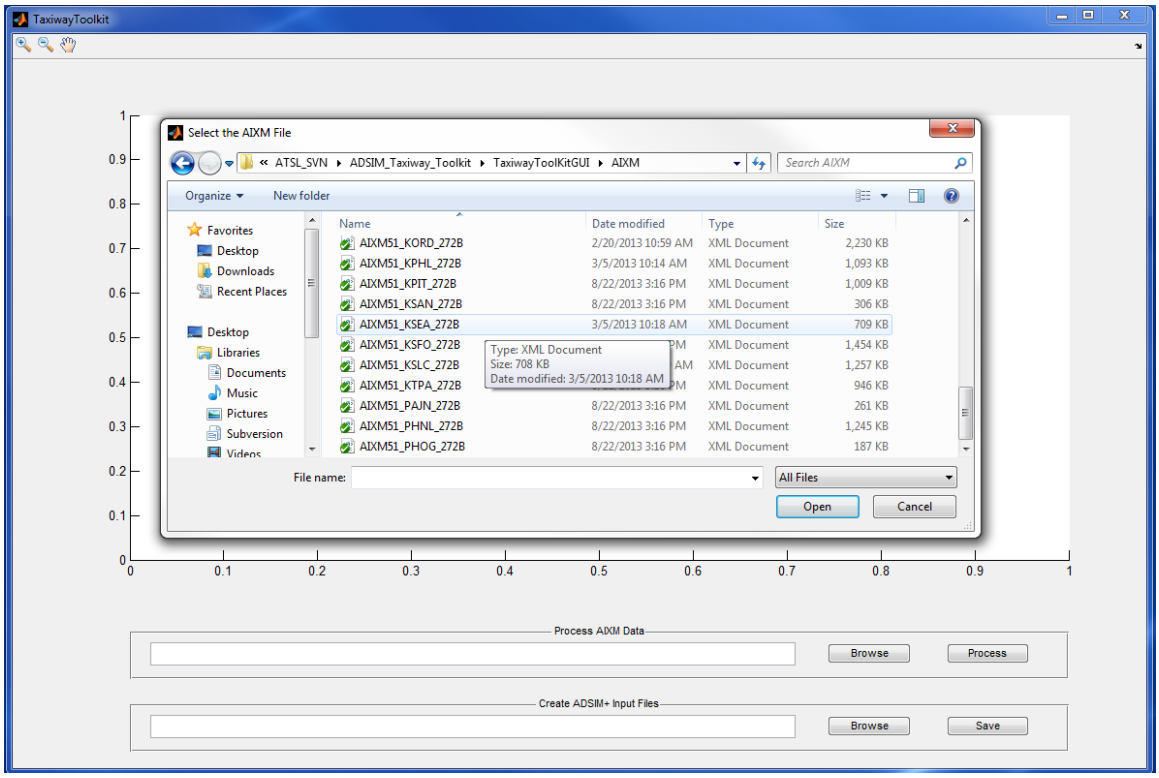

Figure A-6: Navigate the Airport AIXM Data

4. After selecting the airport AIXM data, click ‘Process’ to process the AIXM data. The airport node-link map will be presented in the GUI (Figures A-7, A-8).

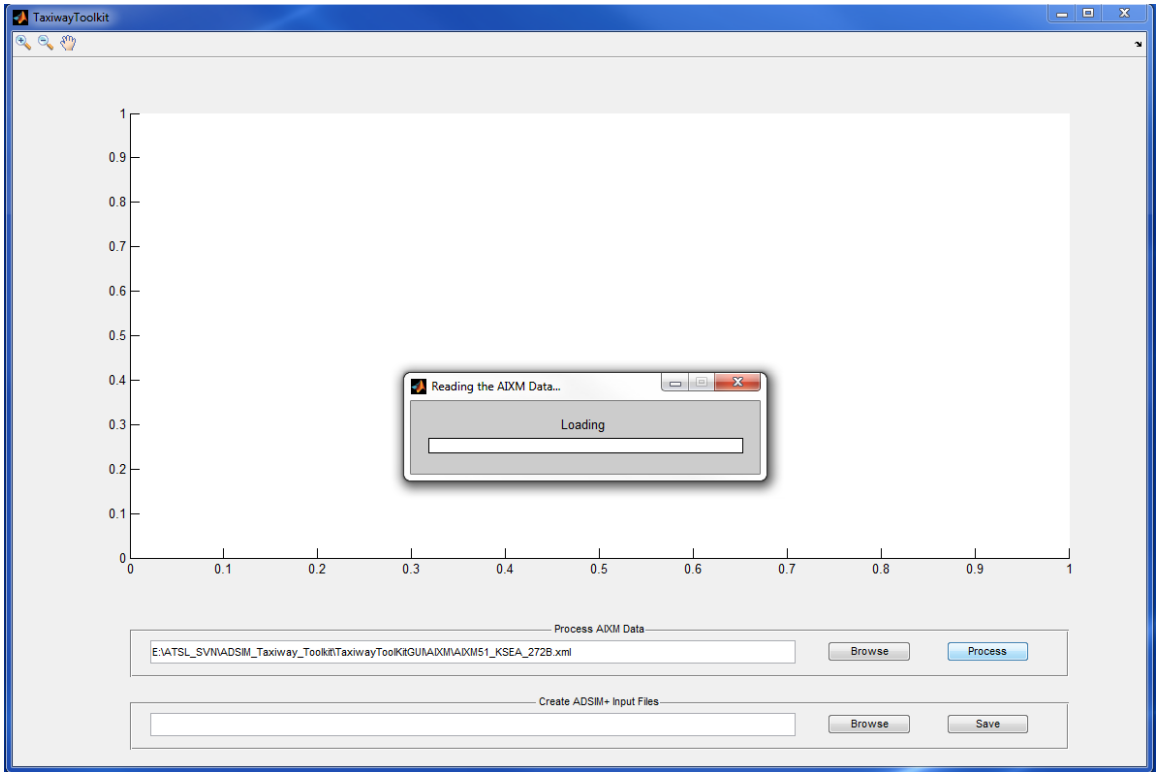

Figure A-7: Taxiway Toolkit is Processing the Airport AIXM Data

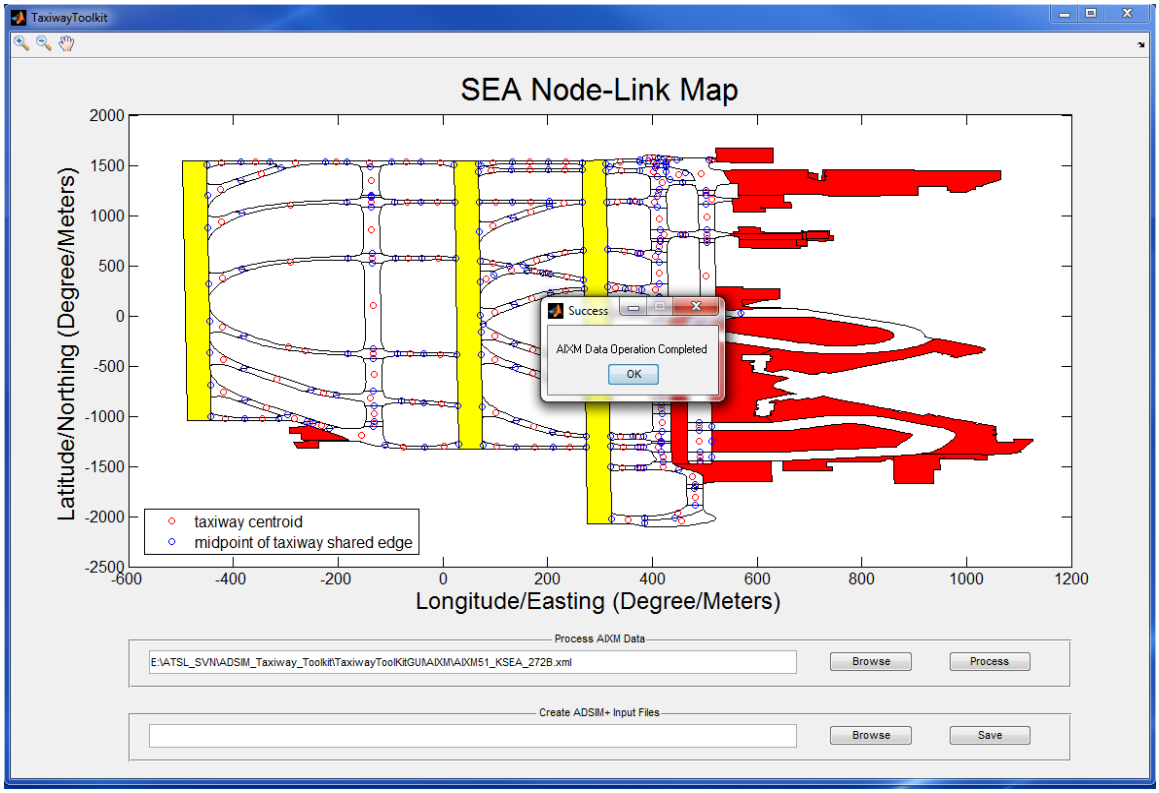

Figure A-8: Taxiway Toolkit Completes the AIXM Data Processing

5. Click ‘Browse’ on the ‘Create ADSIM+ Input Files’ panel to select the airport folder created in ADSIM+. Click ‘Save’ to generate the input files for ADSIM+ (Figures A-9, A-10, A-11).

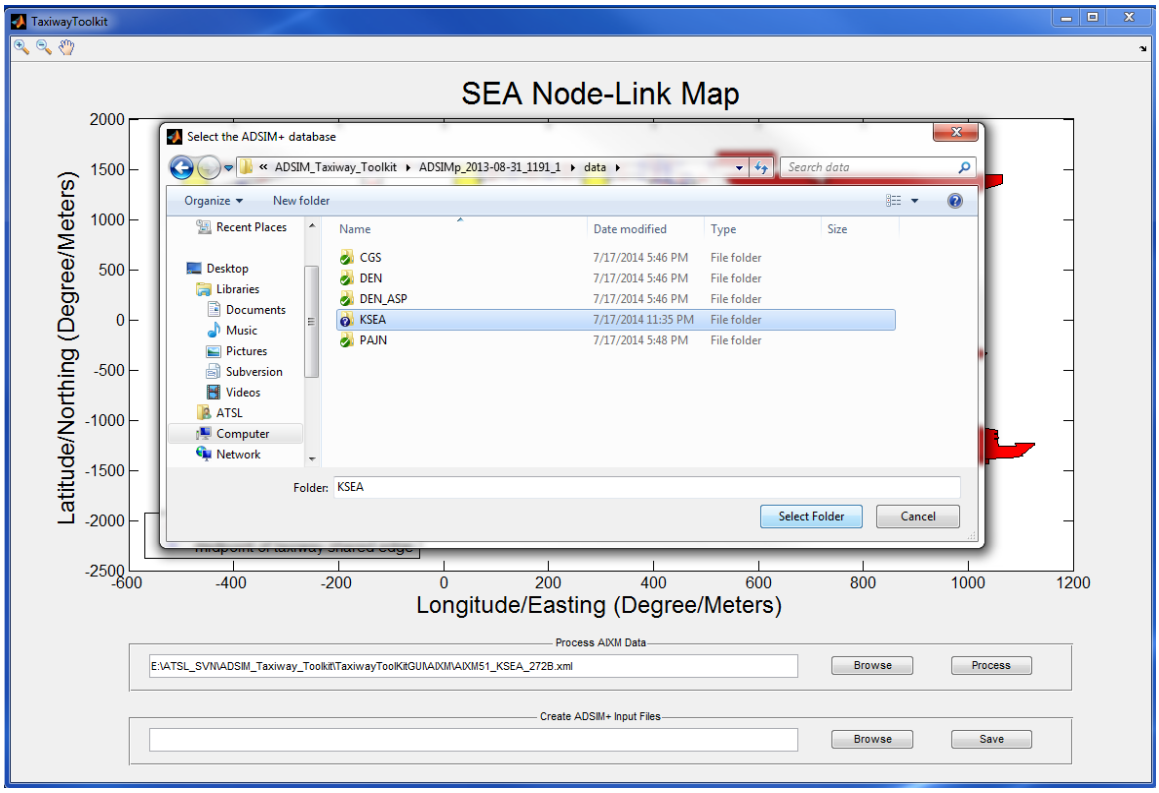

Figure A-9: Navigate to the ADSIM+ Created Airport Folder

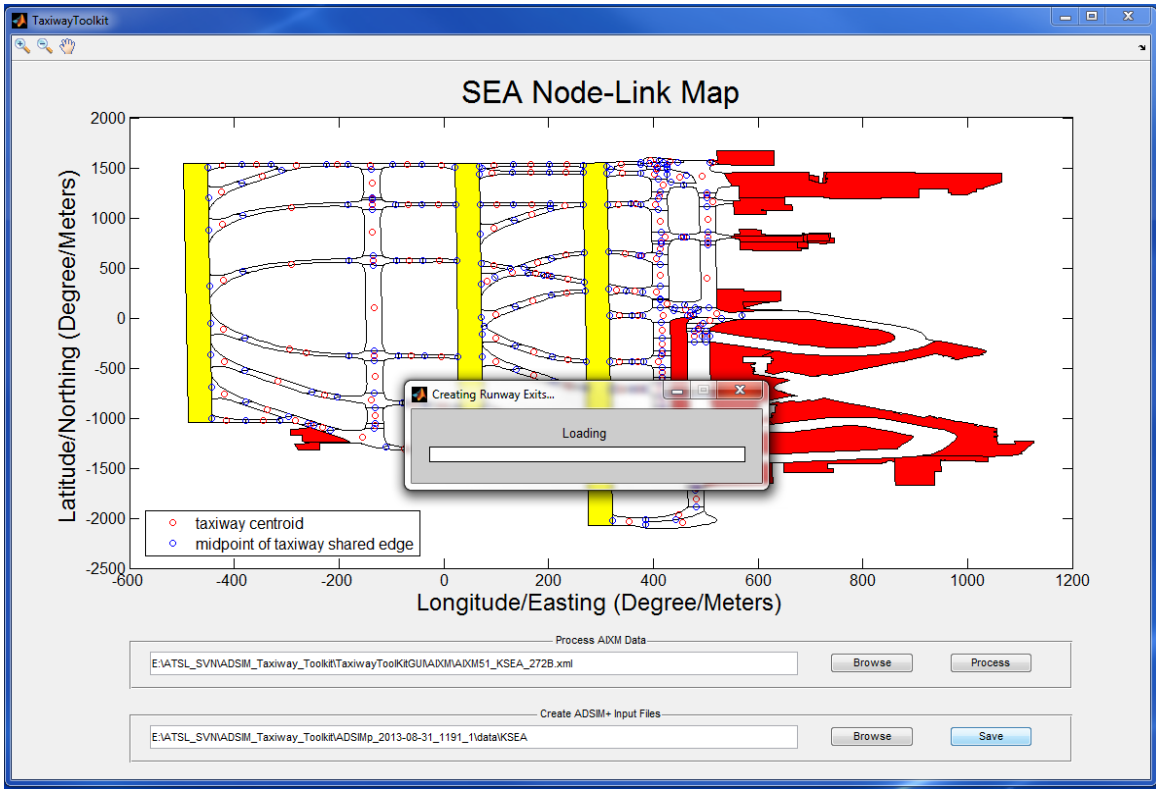

Figure A-10: Taxiway Toolkit is Creating the ADSIM+ Input Files

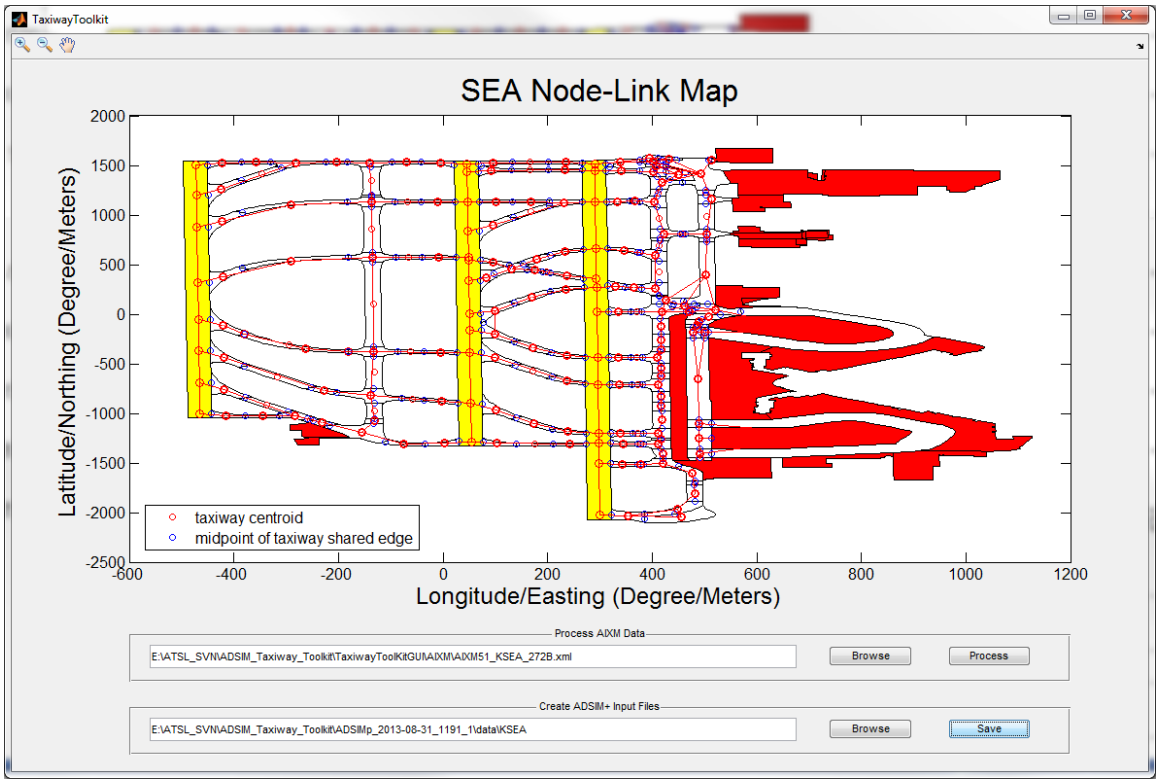

Figure A-11: Taxiway Toolkit Completes the Creation of ADSIM+ Input Files

6. Close *Taxiway Toolkit* and load the airport in ADSIM+. The airport runways, taxiways, gates, and taxipath system have been generated. (Figures A-12, A-13).

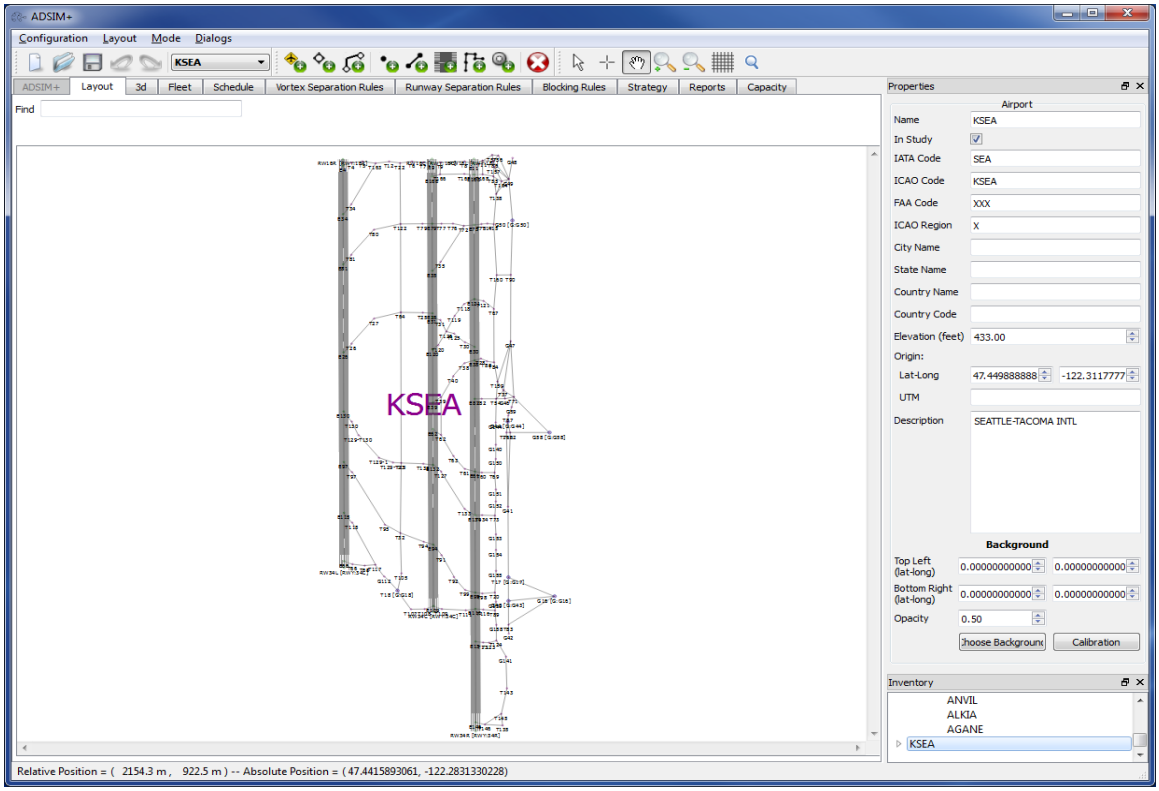

Figure A-12: Airport Taxiways, Runways and Gates Viewed in ADSIM+ (Source: FAA)

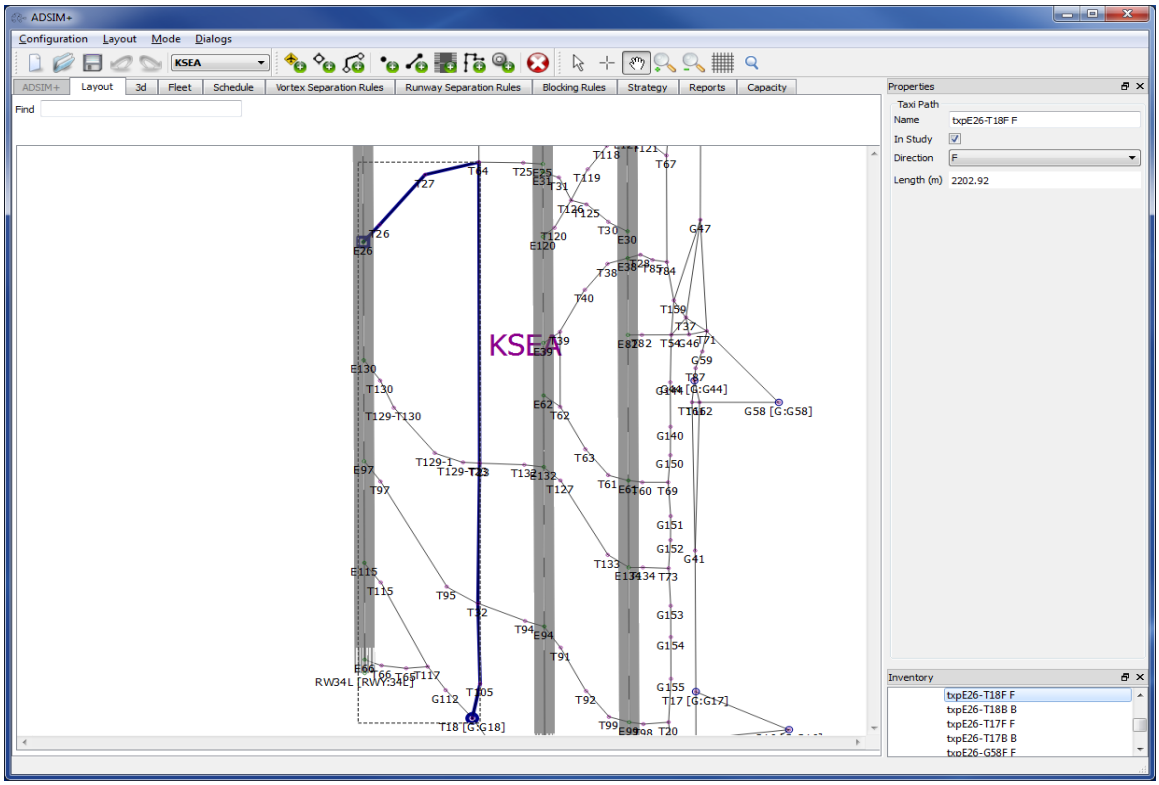

Figure A-13: Taxipath Network Viewed in ADSIM+ (Source: FAA)

B. Node-Link Networks Produced by Taxiway Toolkit

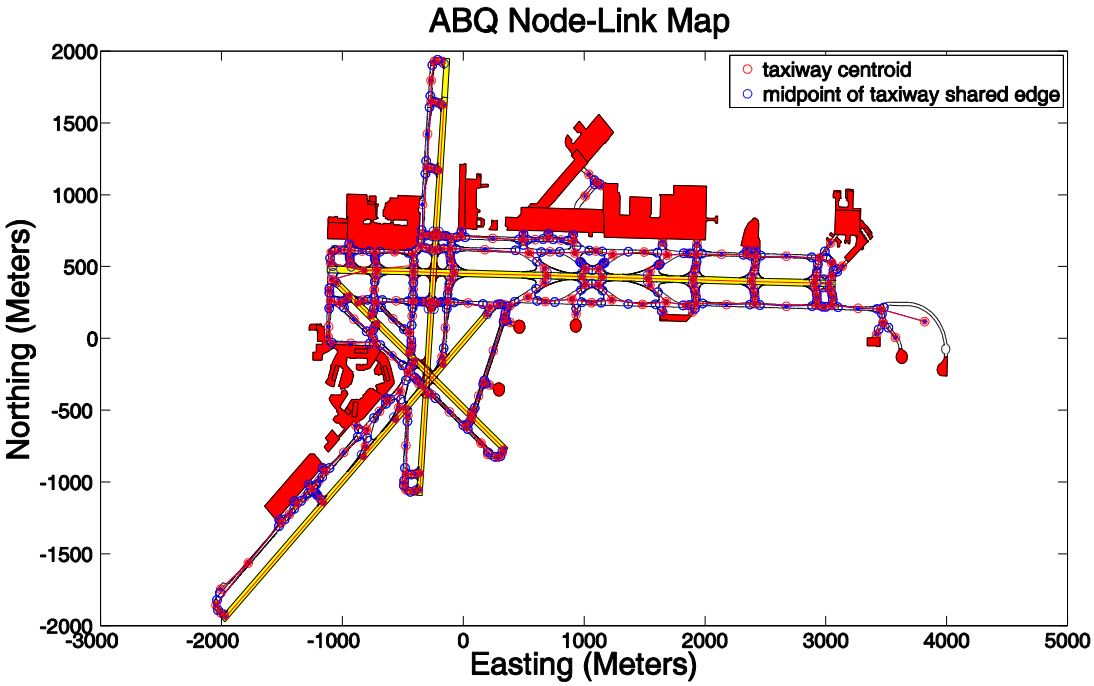

Figure B-1 Albuquerque International Sun-port Node-Link Network

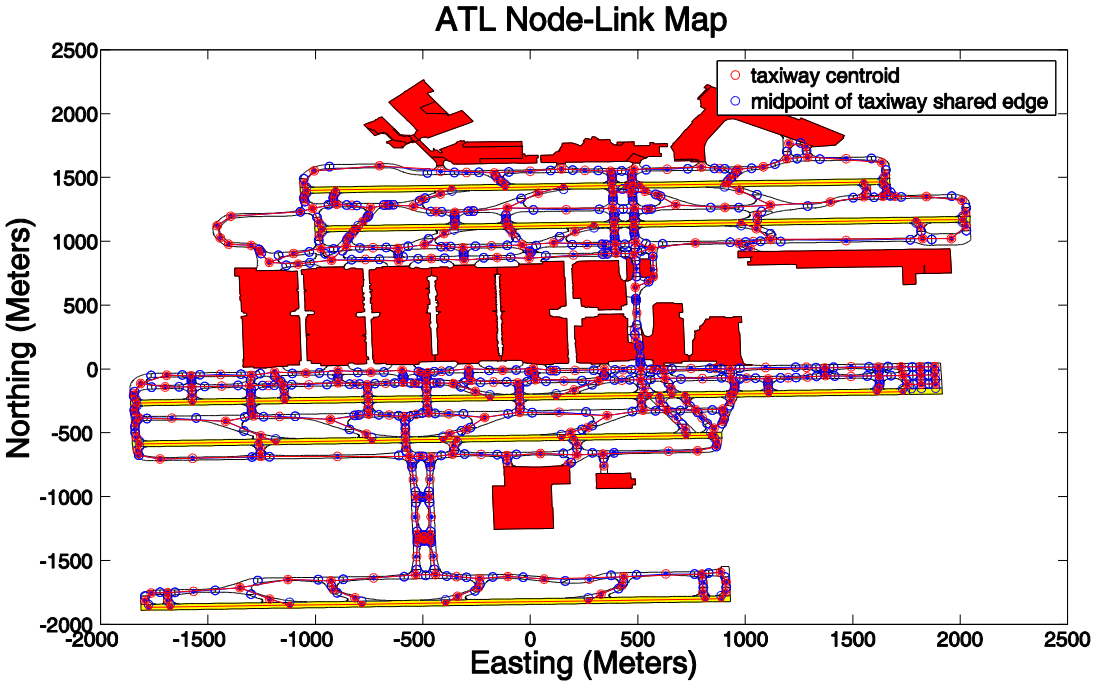

Figure B-2 Hartsfield-Jackson Atlanta International Airport Node-Link Network

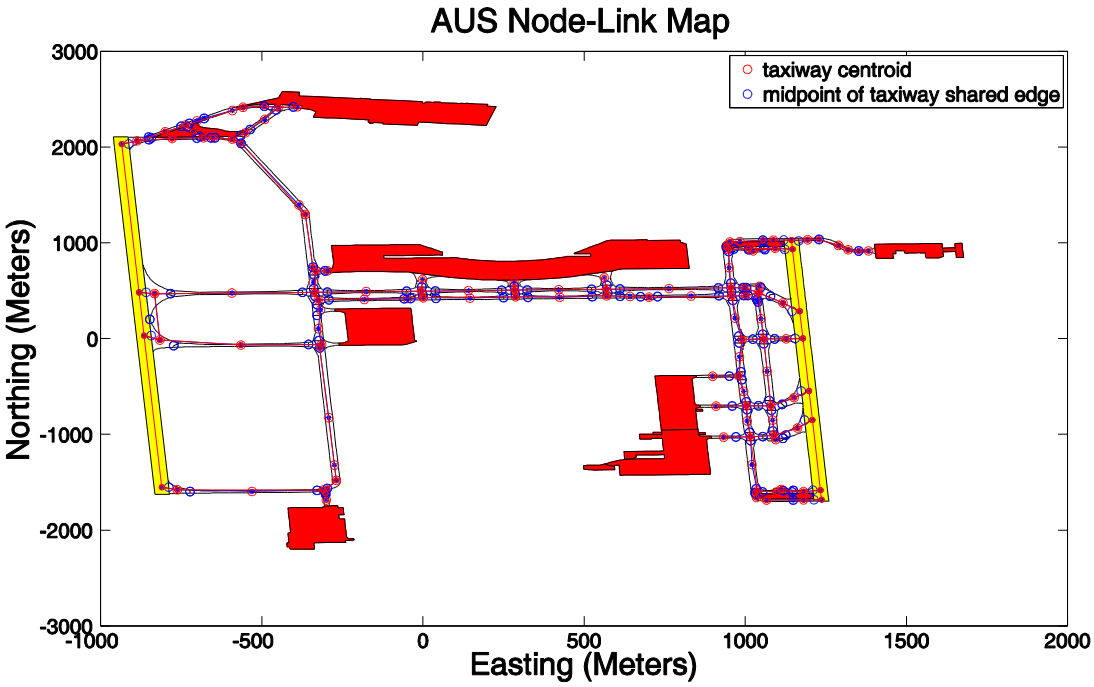

Figure B-3 Austin-Bergstrom International Airport Node-Link Network

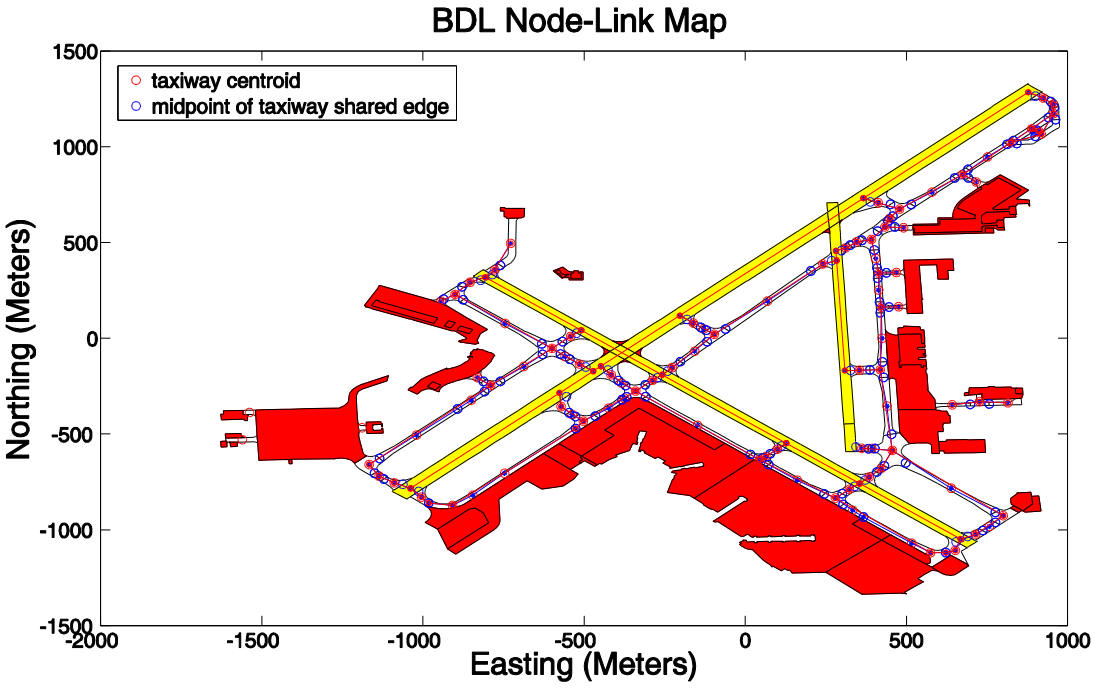

Figure B-4 Bradley International Airport Node-Link Network

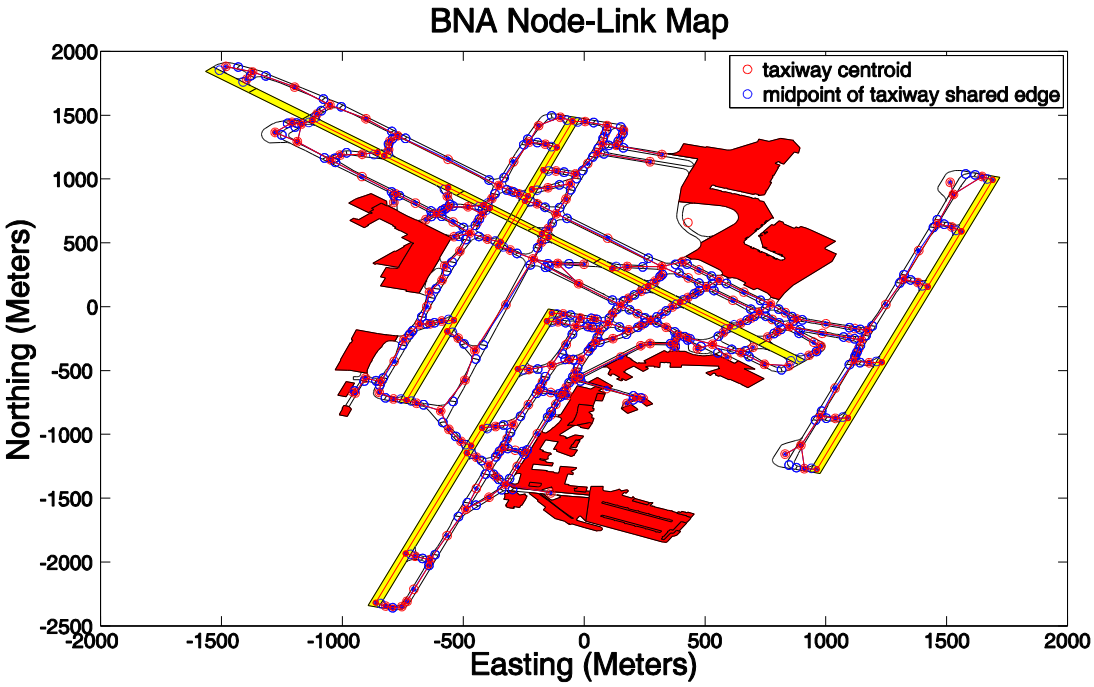

Figure B-5 Nashville International Airport Node-Link Network

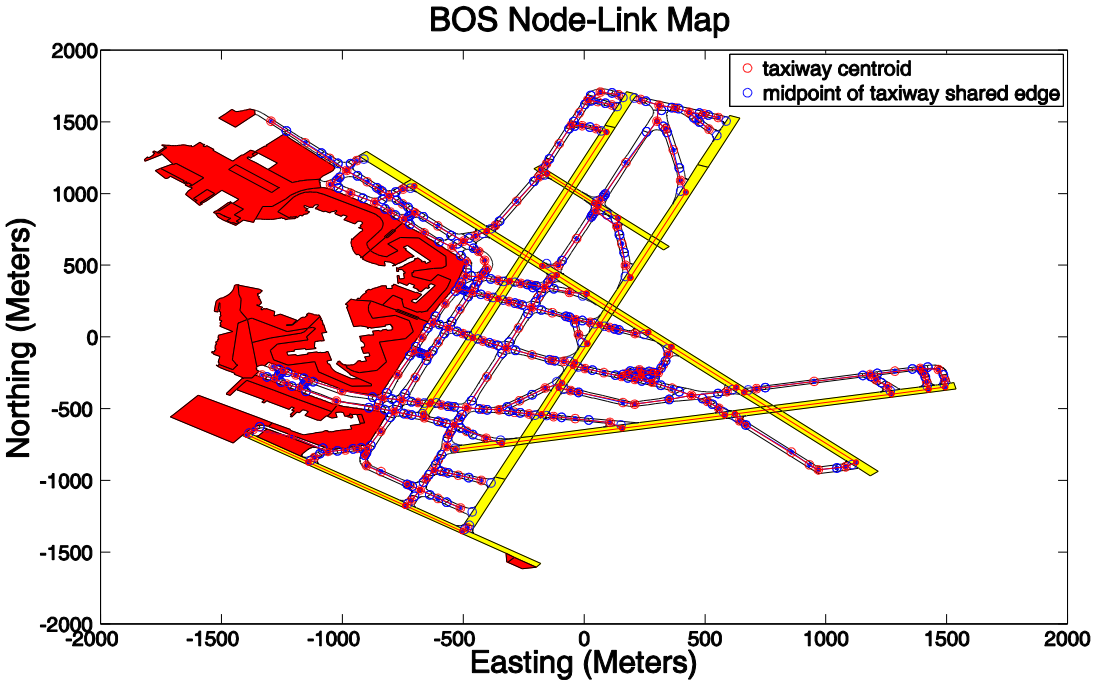

Figure B-6 Boston Logan International Airport Node-Link Network

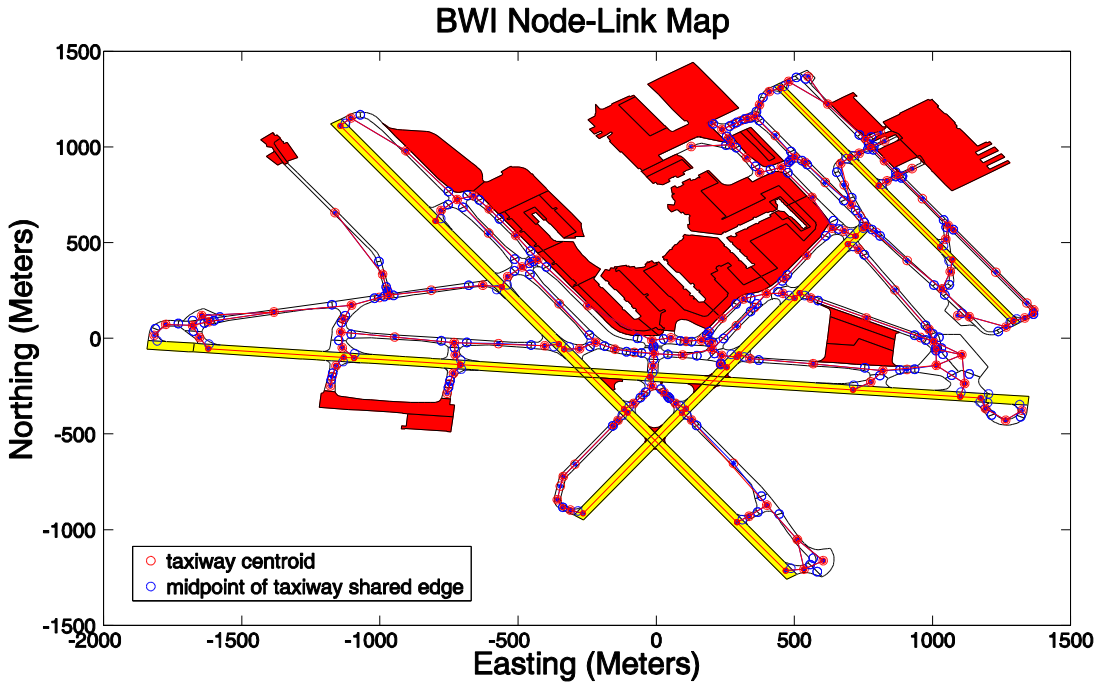

Figure B-7 Baltimore/Washington International Thurgood Marshall Airport Node-Link Network

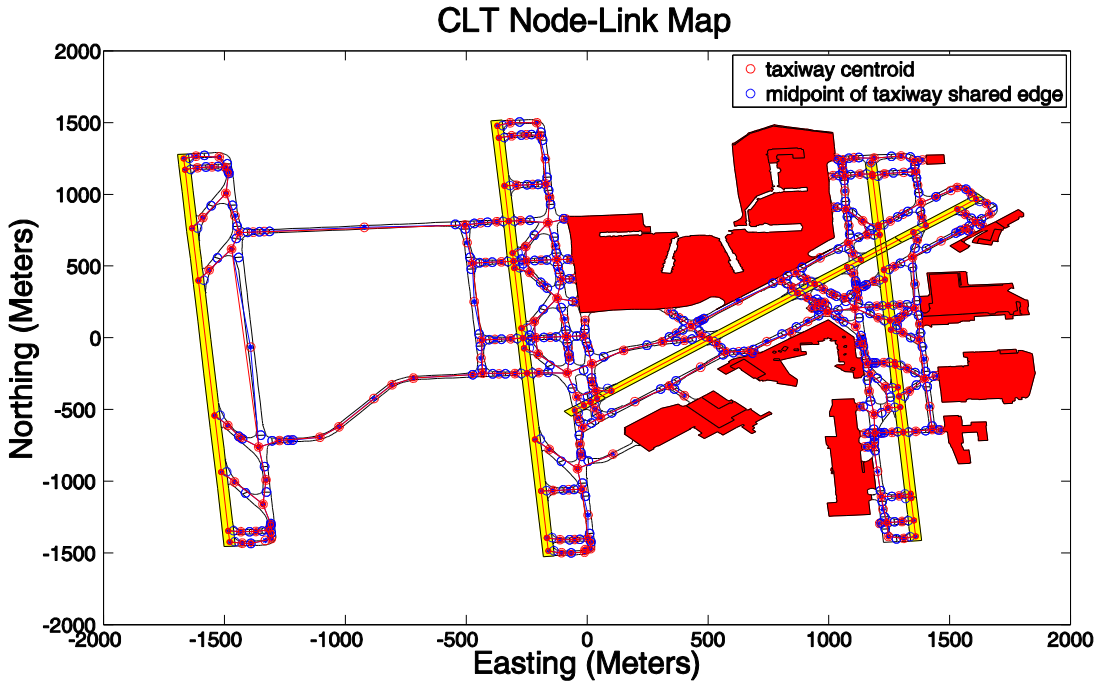

Figure B-8 Charlotte Douglas International Airport Node-Link Network

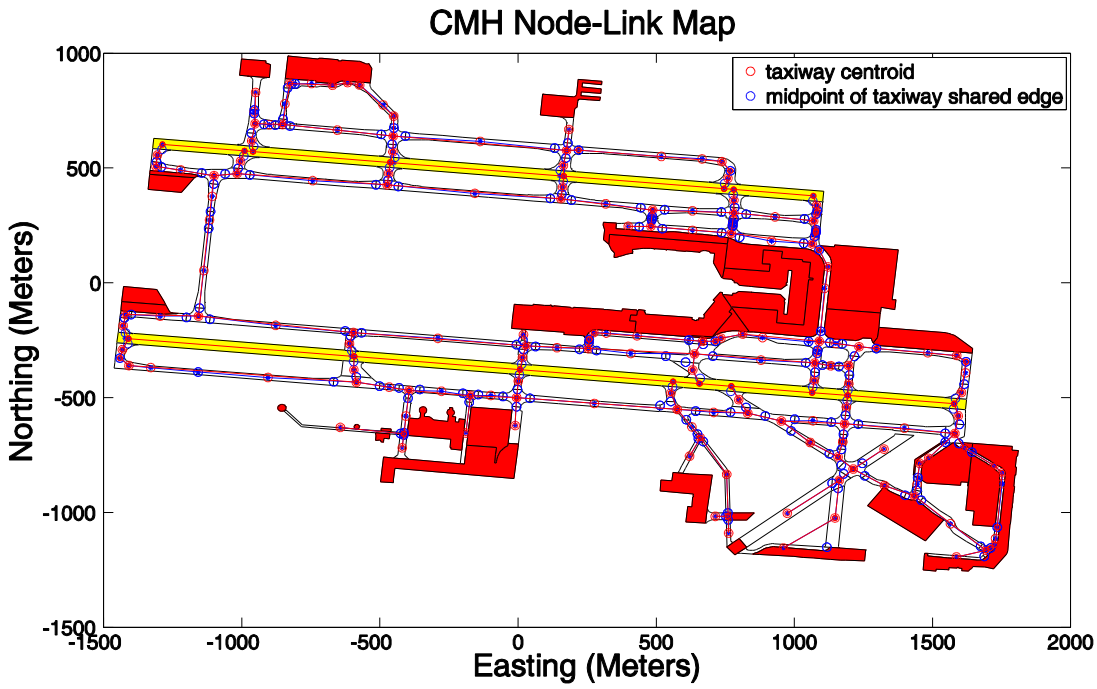

Figure B-9 Port Columbus International Airport Node-Link Network

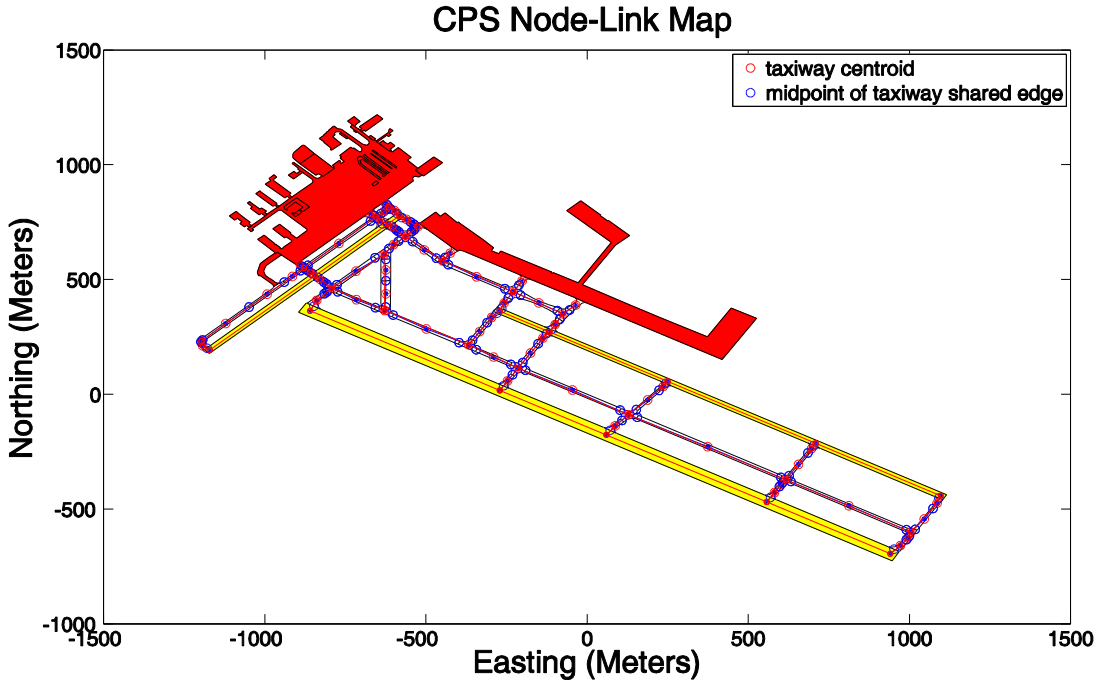

Figure B-10 St. Louis Downtown Airport Node-Link Network

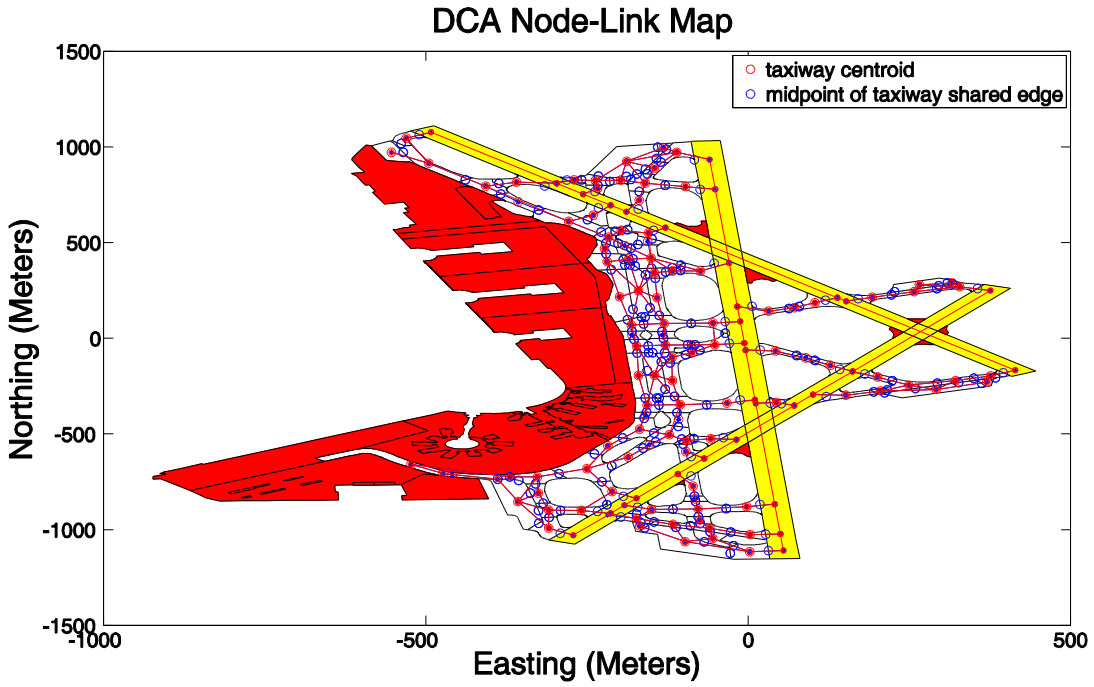

Figure B-11 Ronald Reagan Washington National Airport Node-Link Network

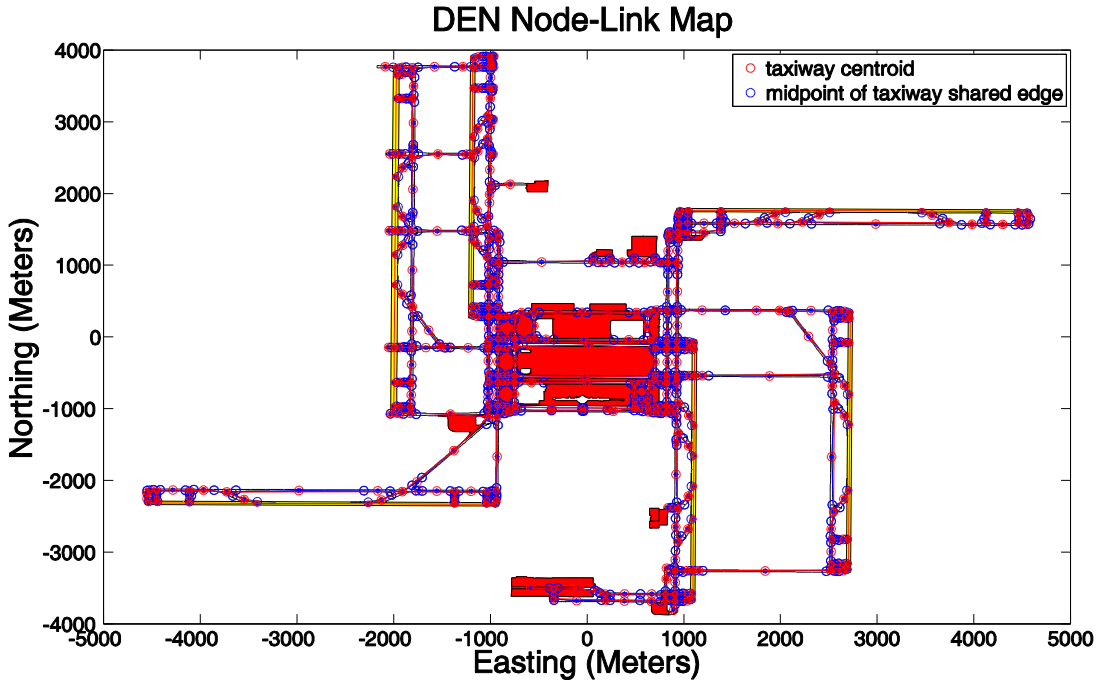

Figure B-12 Denver International Airport Node-Link Network

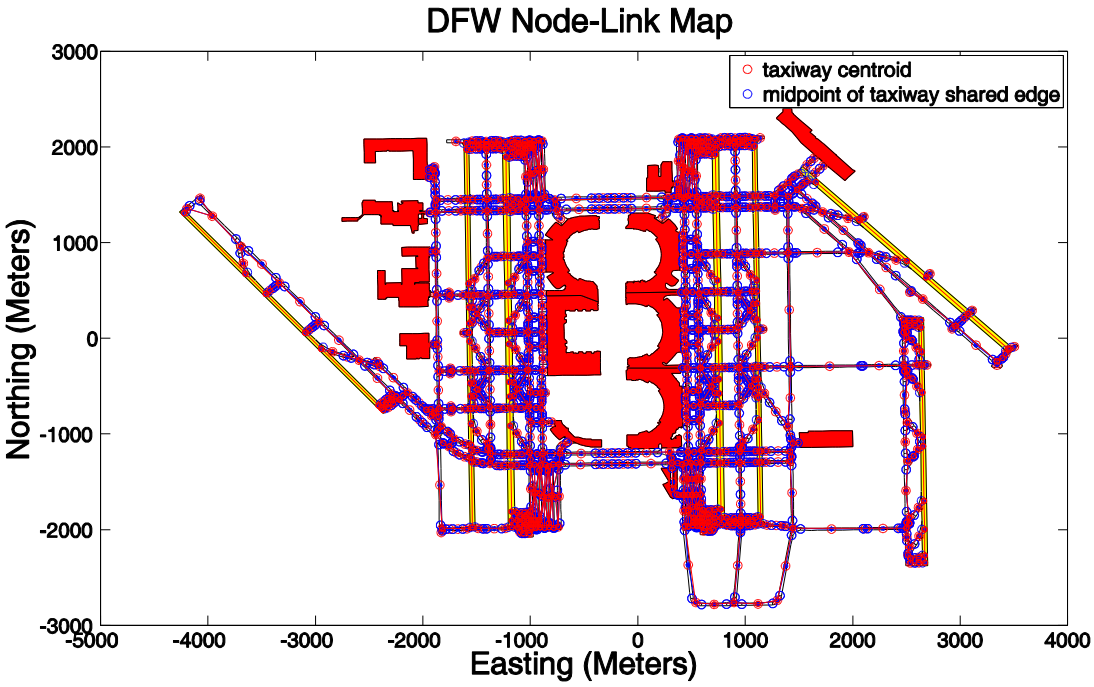

Figure B-13 Dallas/Fort Worth International Airport Node-Link Network

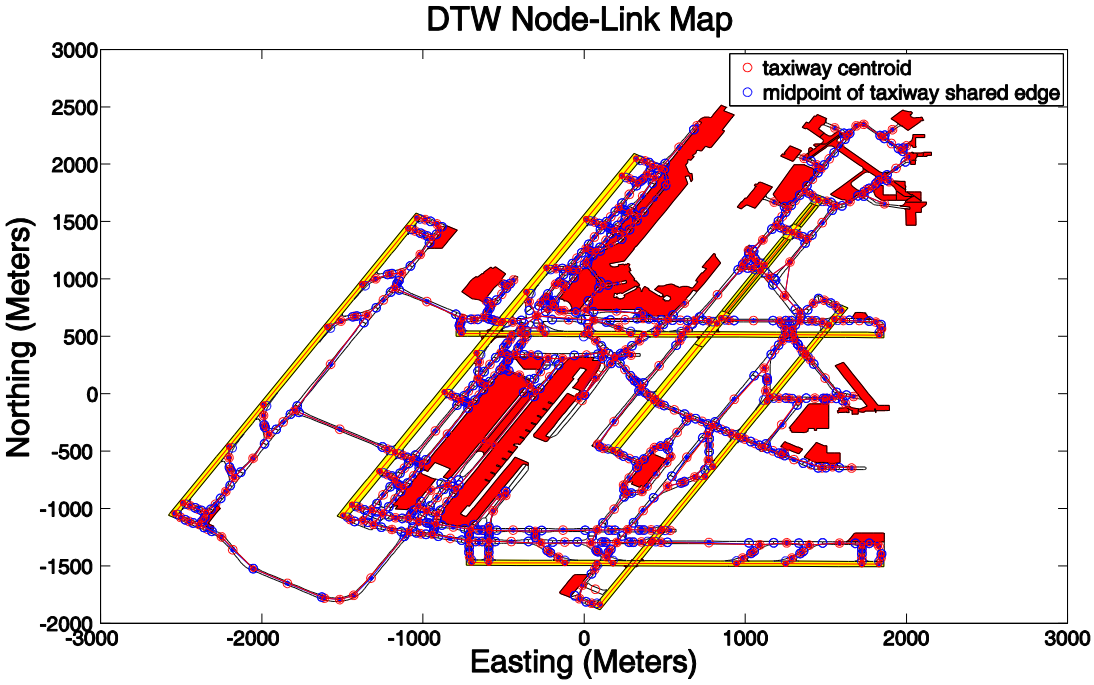

Figure B-14 Detroit Metropolitan Wayne County Airport Node-Link Network

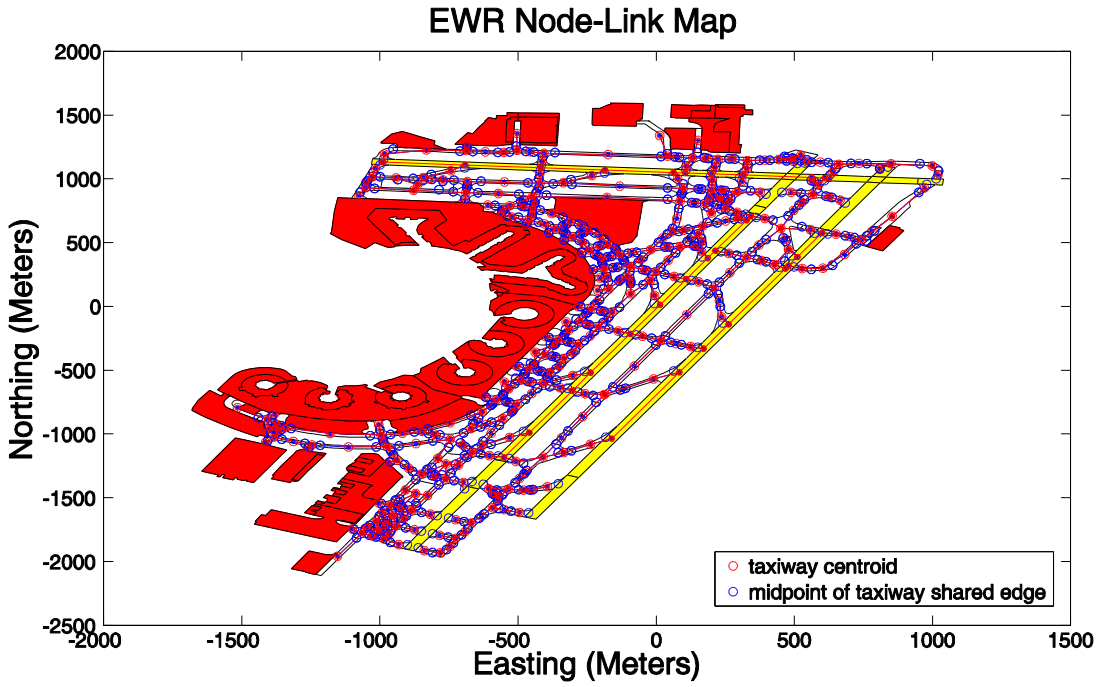

Figure B-15 Newark Liberty International Airport Node-Link Network

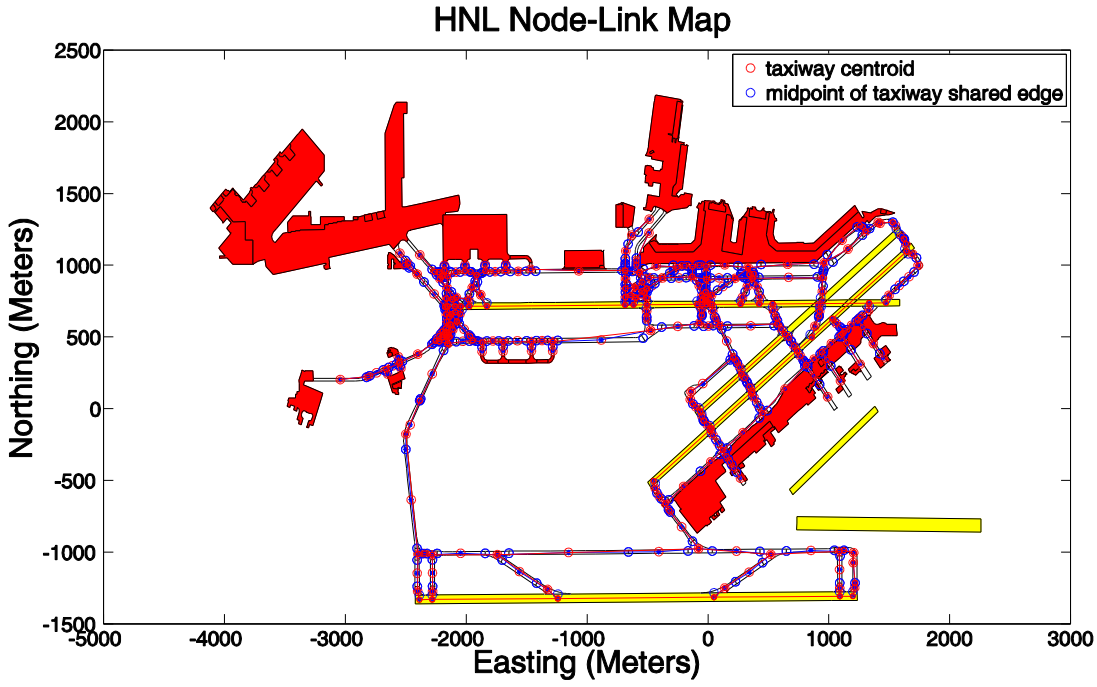

Figure B-16 Honolulu International Airport Node-Link Network

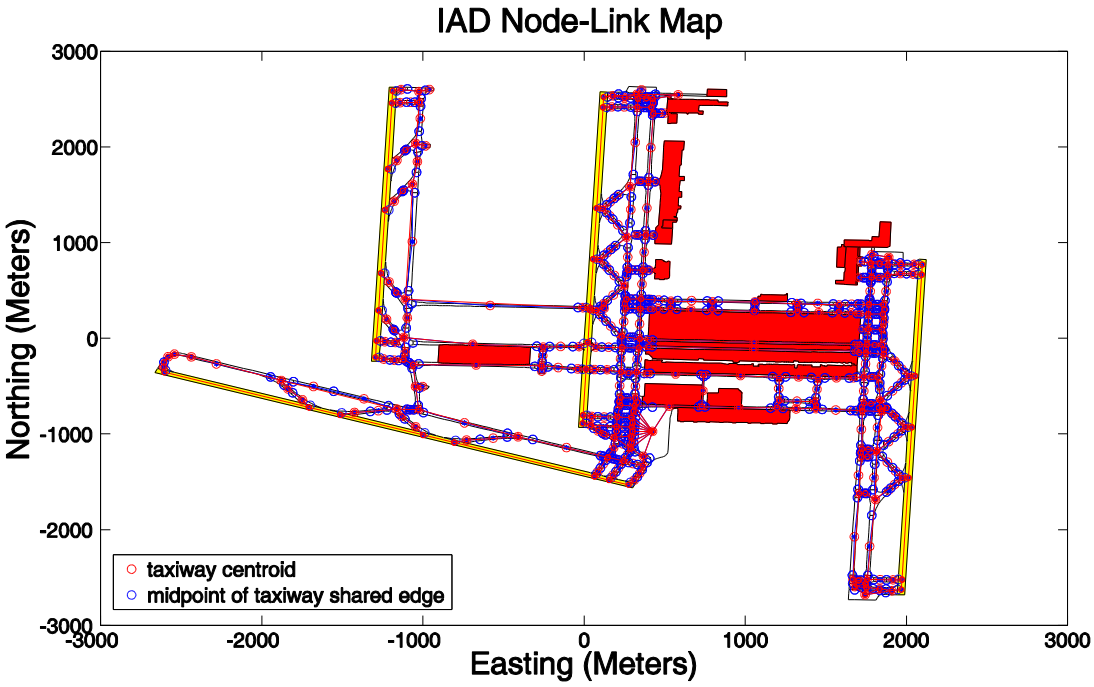

Figure B-17 Washington Dulles International Airport Node-Link Network

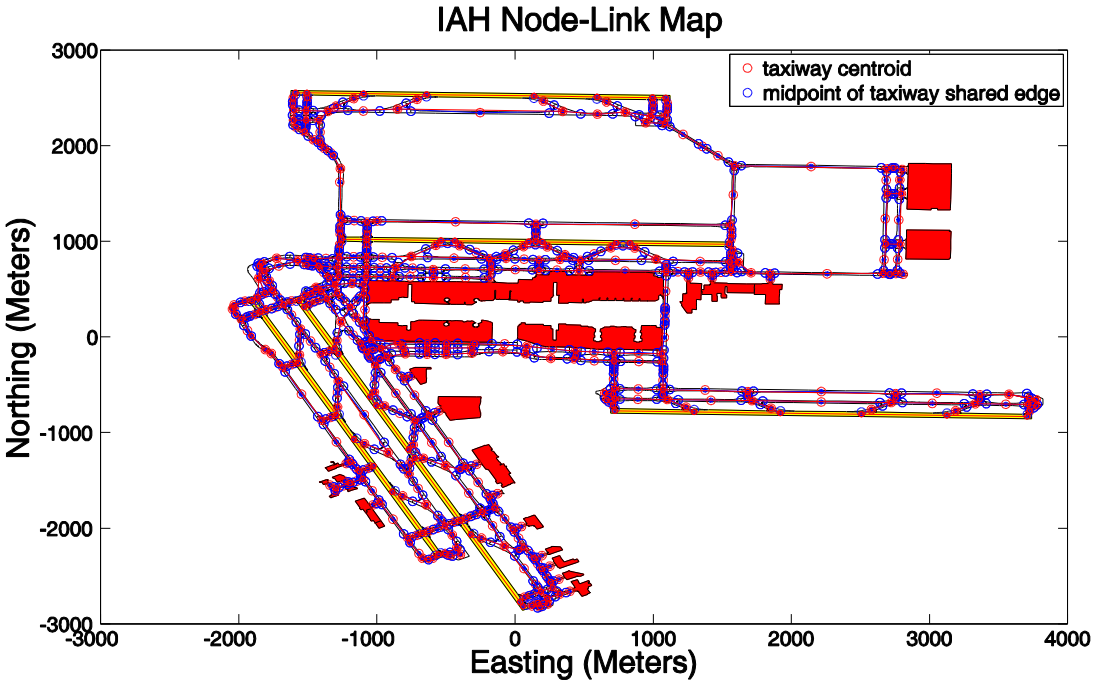

Figure B-18 George Bush Intercontinental Airport Node-Link Network

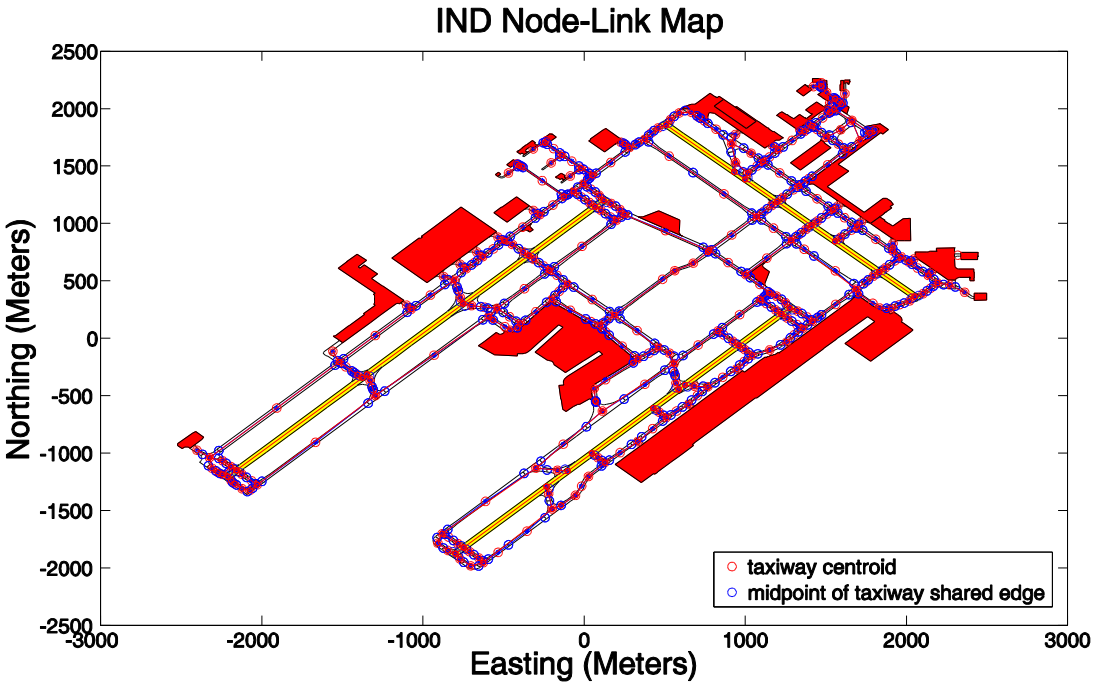

Figure B-19 Indianapolis International Airport Node-Link Network

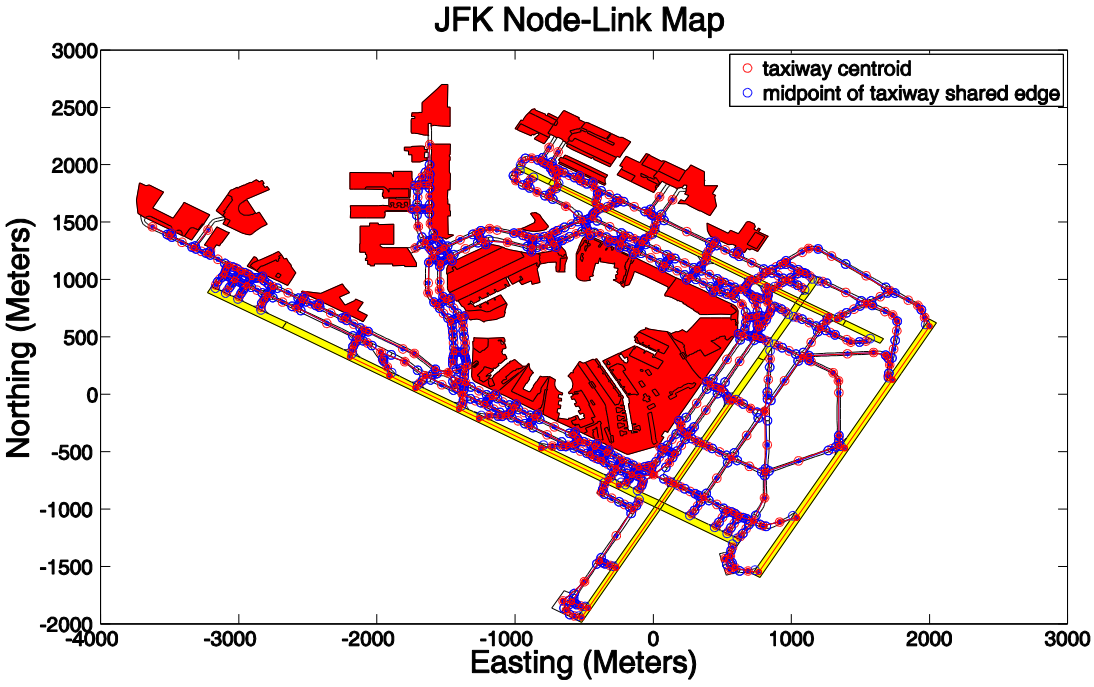

Figure B-20 John F. Kennedy International Airport Node-Link Network

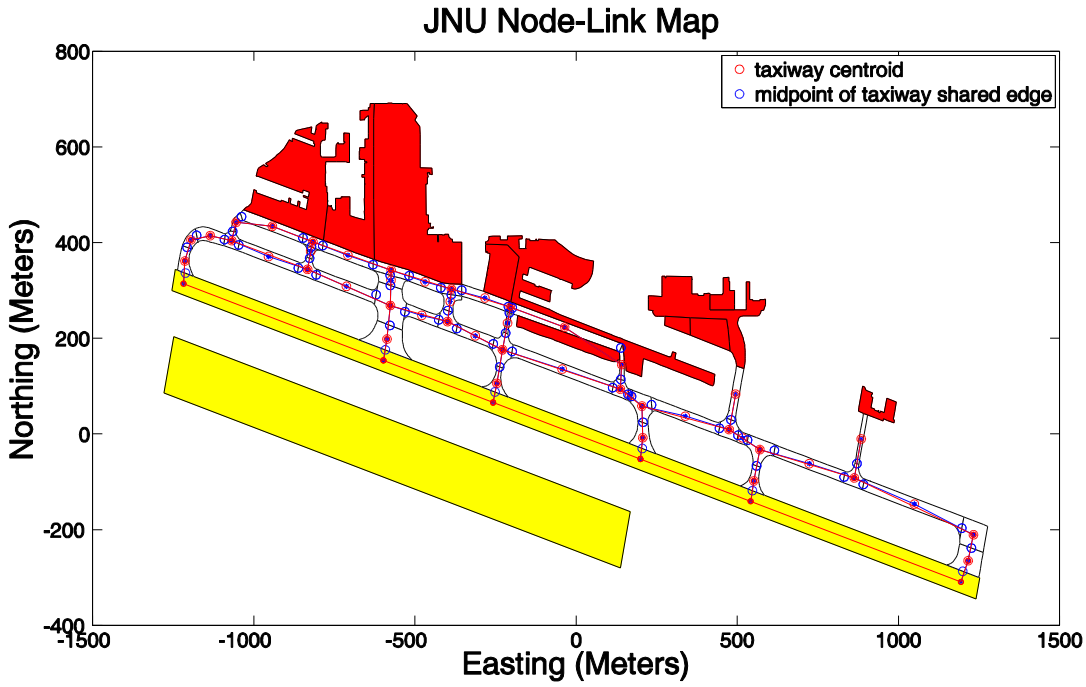

Figure B-21 Juneau International Airport Node-Link Network

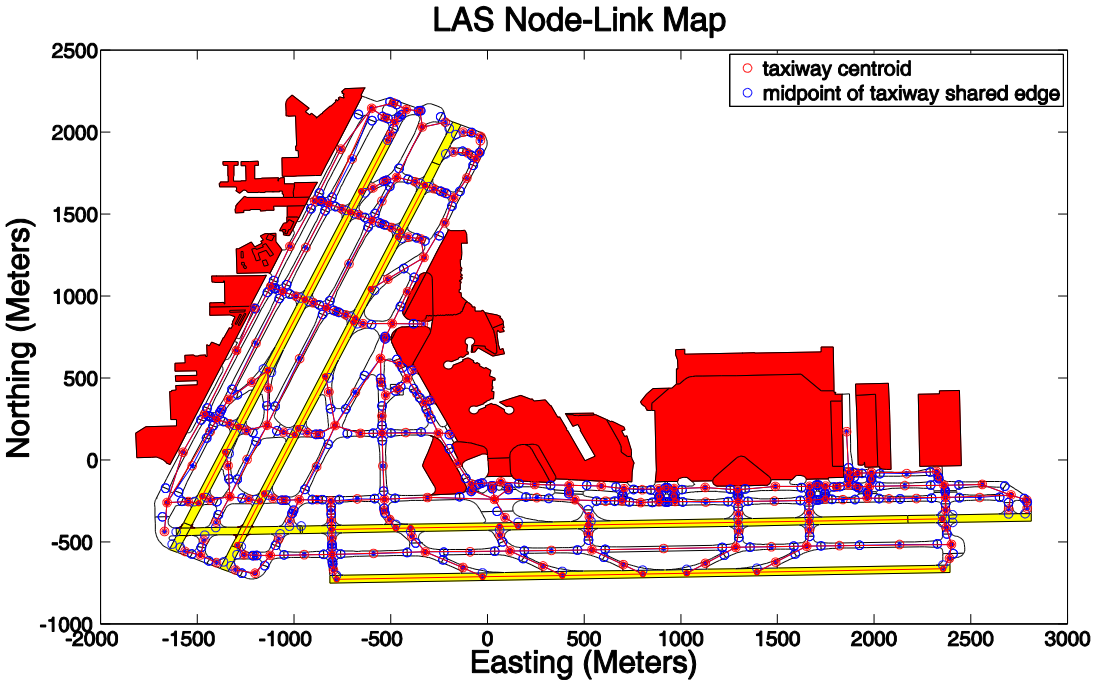

Figure B-22 McCarran International Airport Node-Link Network

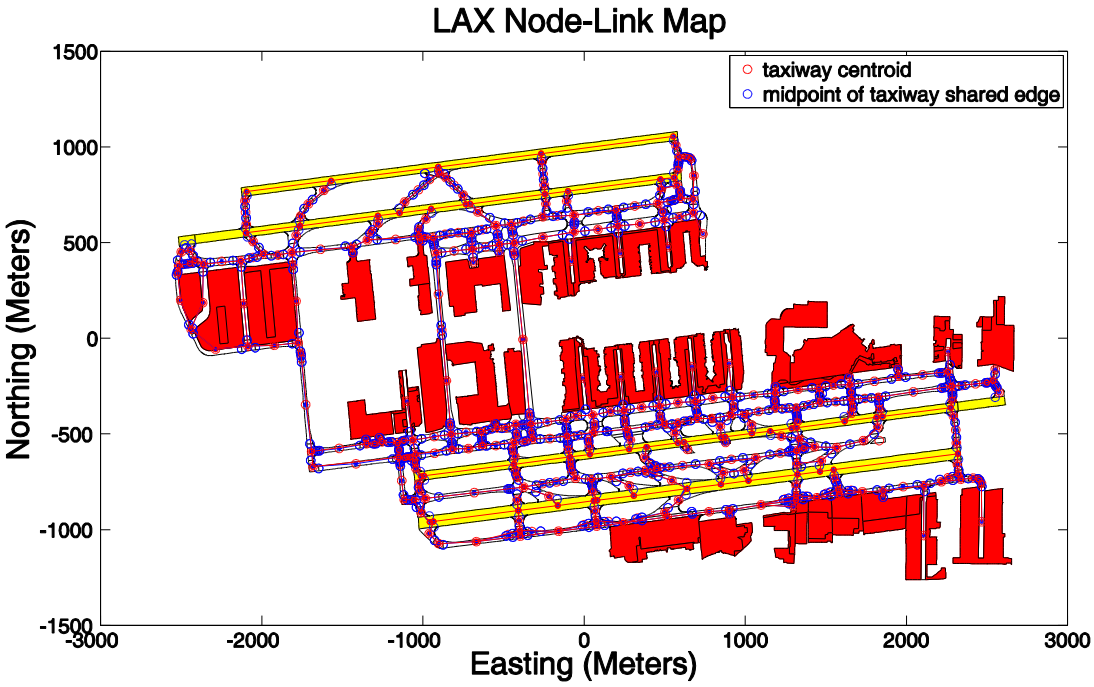

Figure B-23 Los Angeles International Airport Node-Link Network

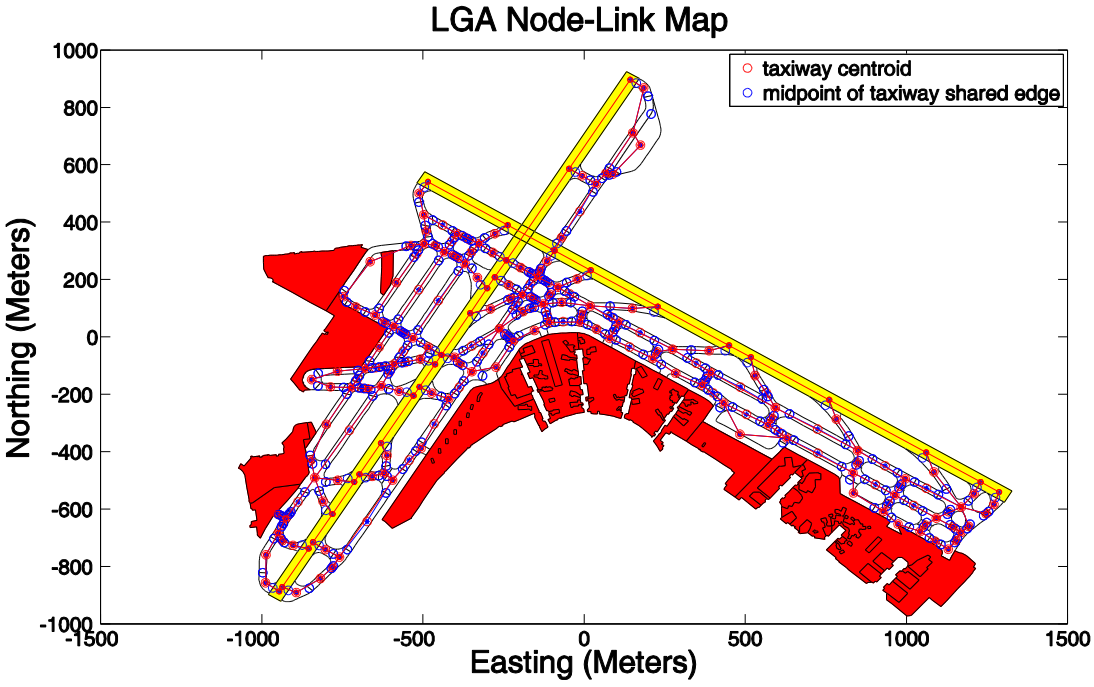

Figure B-24 LaGuardia Airport Node-Link Network

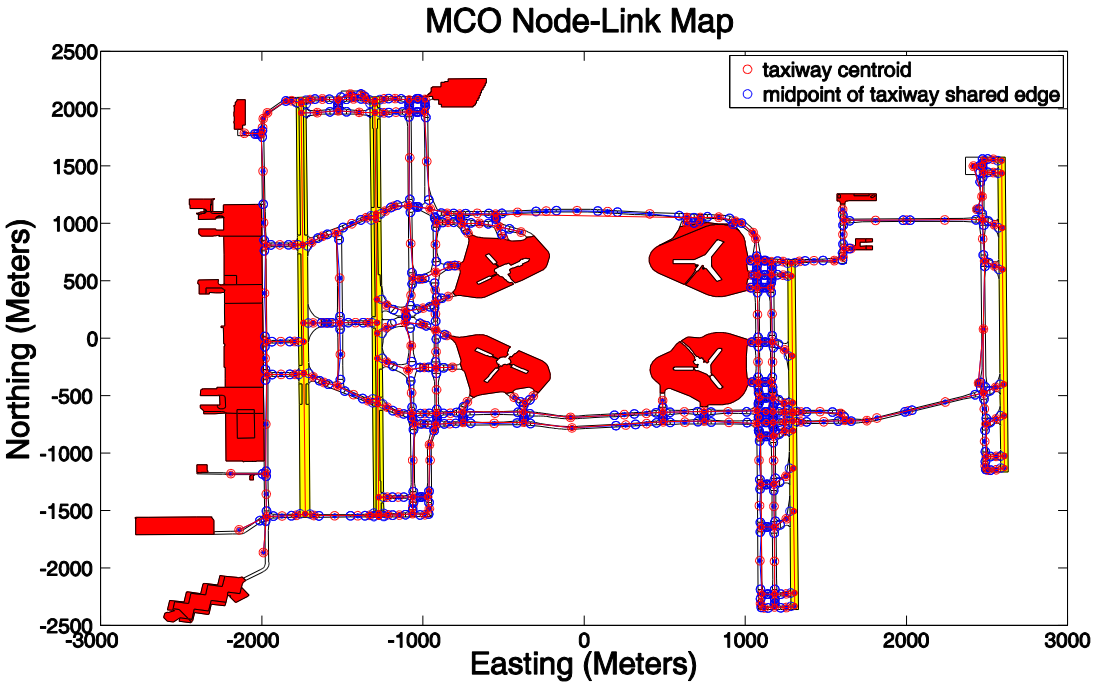

Figure B-25 Orlando International Airport Node-Link Network

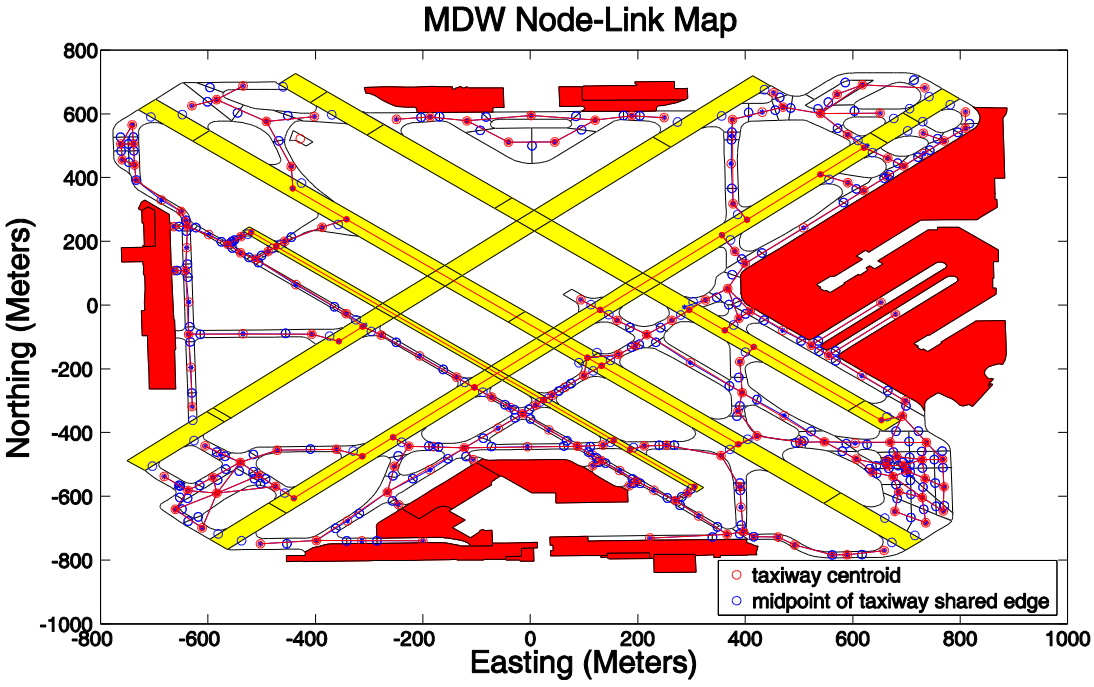

Figure B-26 Chicago Midway International Airport Node-Link Network

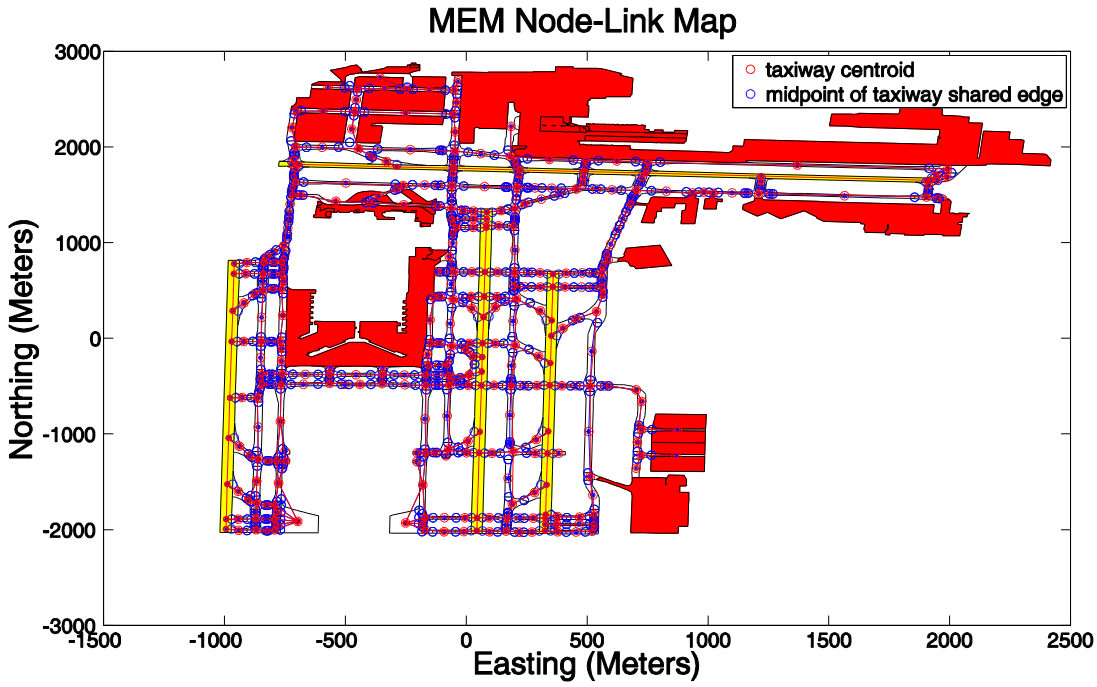

Figure B-27 Memphis International Airport Node-Link Network

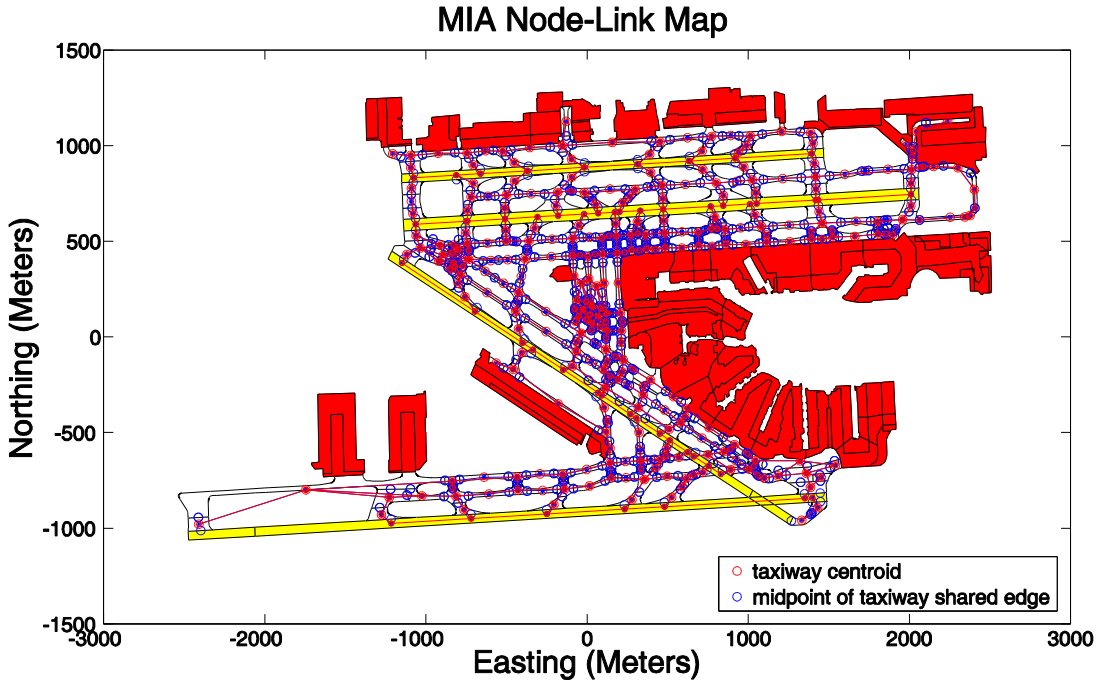

Figure B-28 Miami International Airport Node-Link Network

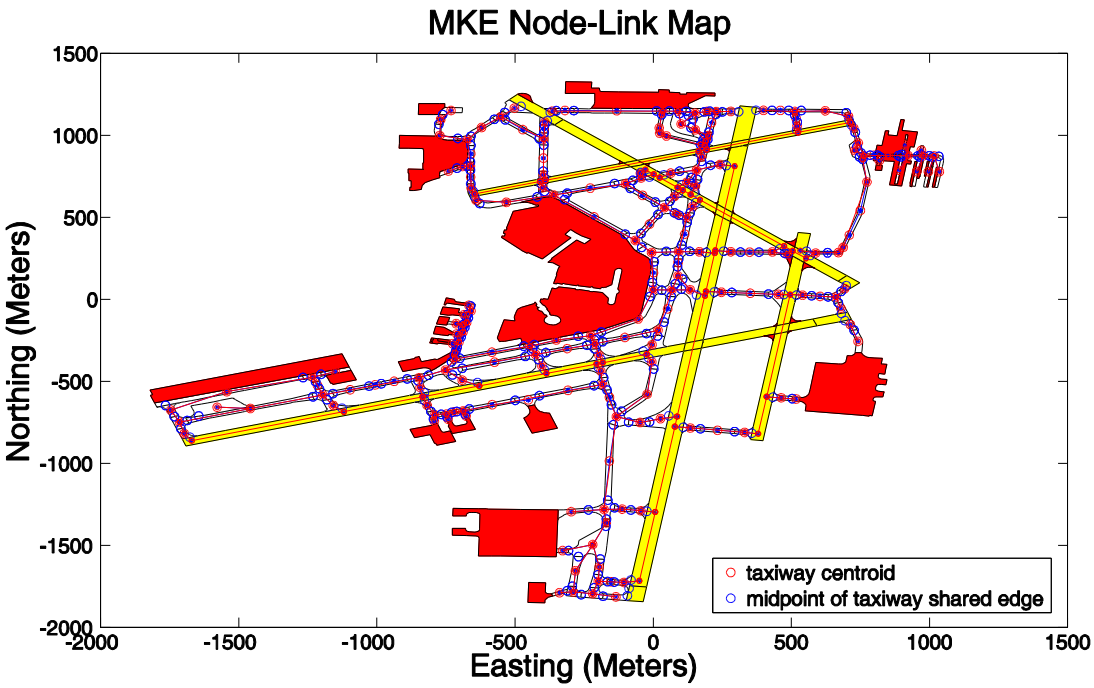

Figure B-29 General Mitchell International Airport Node-Link Network

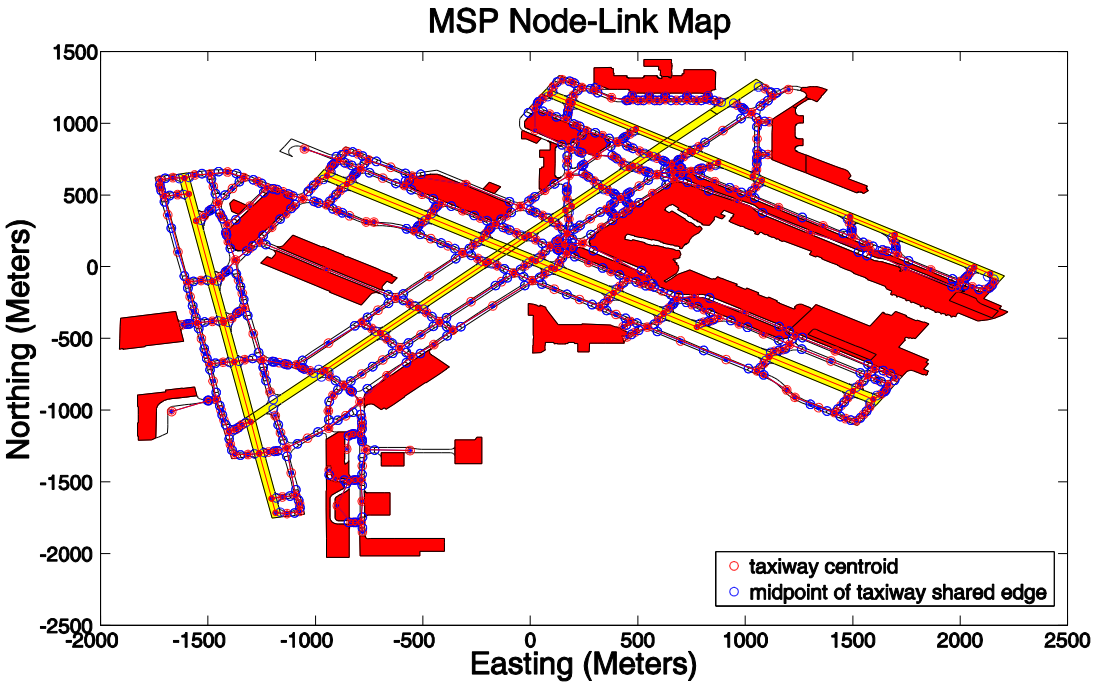

Figure B-30 Minneapolis-Saint Paul International Airport Node-Link Network

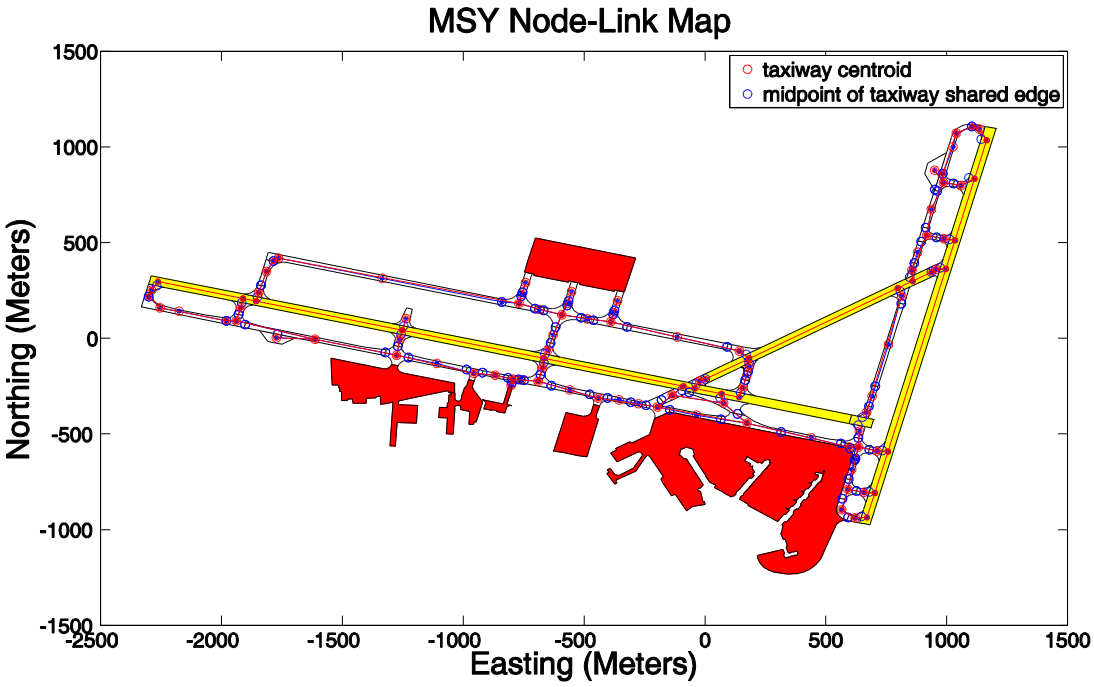

Figure B-31 Louis Armstrong International Airport Node-Link Network

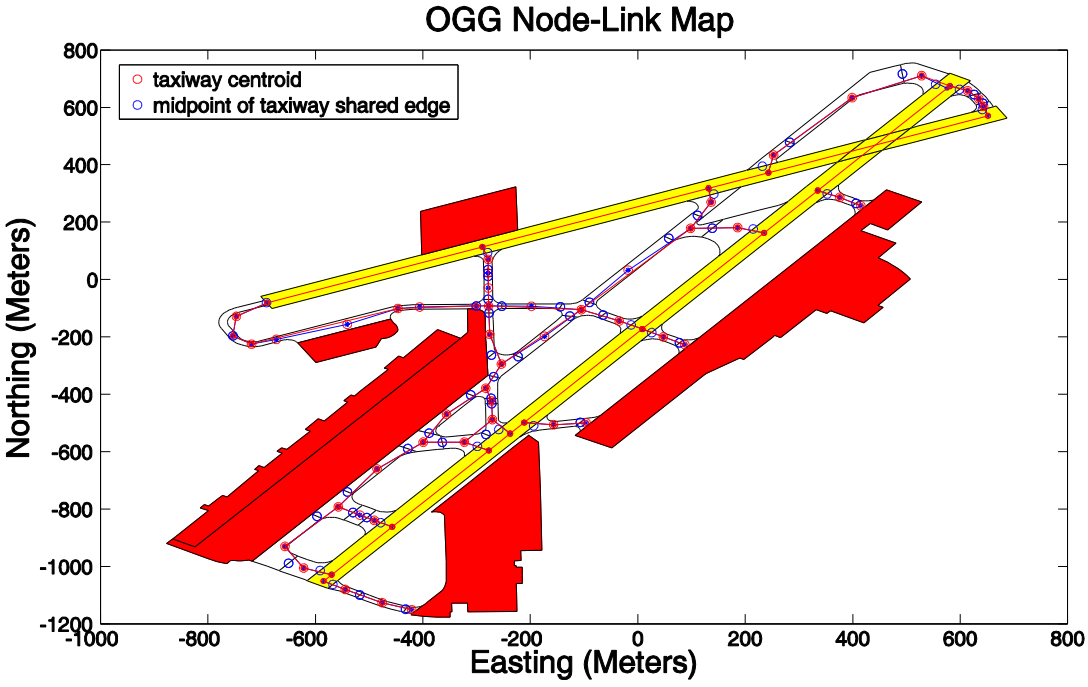

Figure B-32 Kahului Airport Node-Link Network

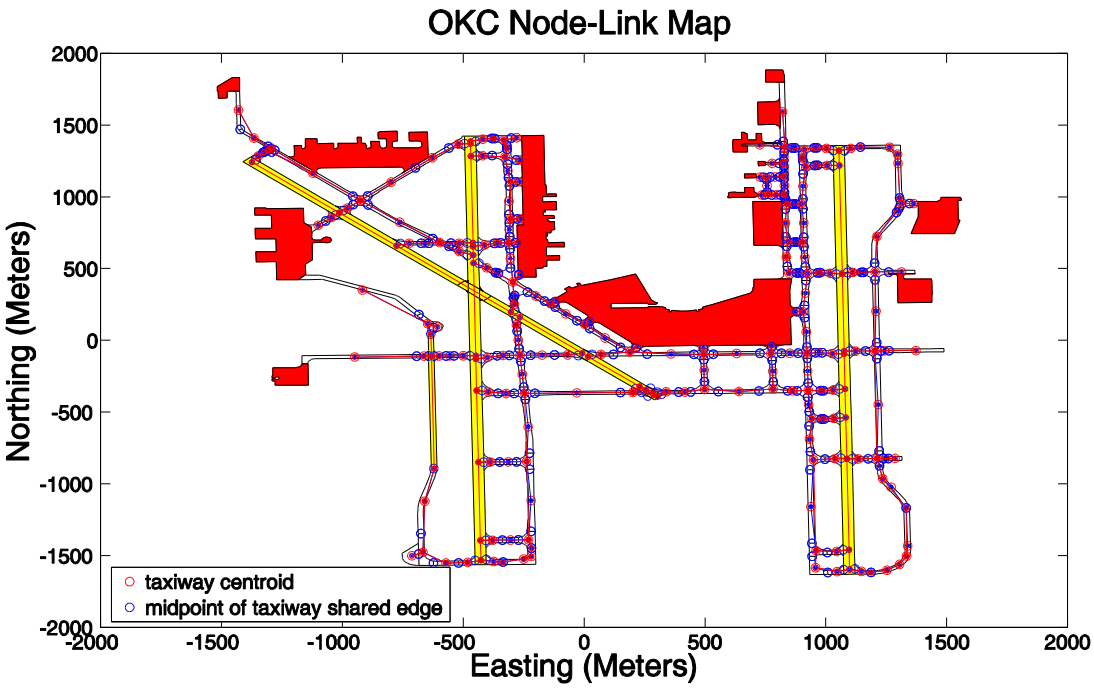

Figure B-33 Will Rogers World Airport Node-Link Network

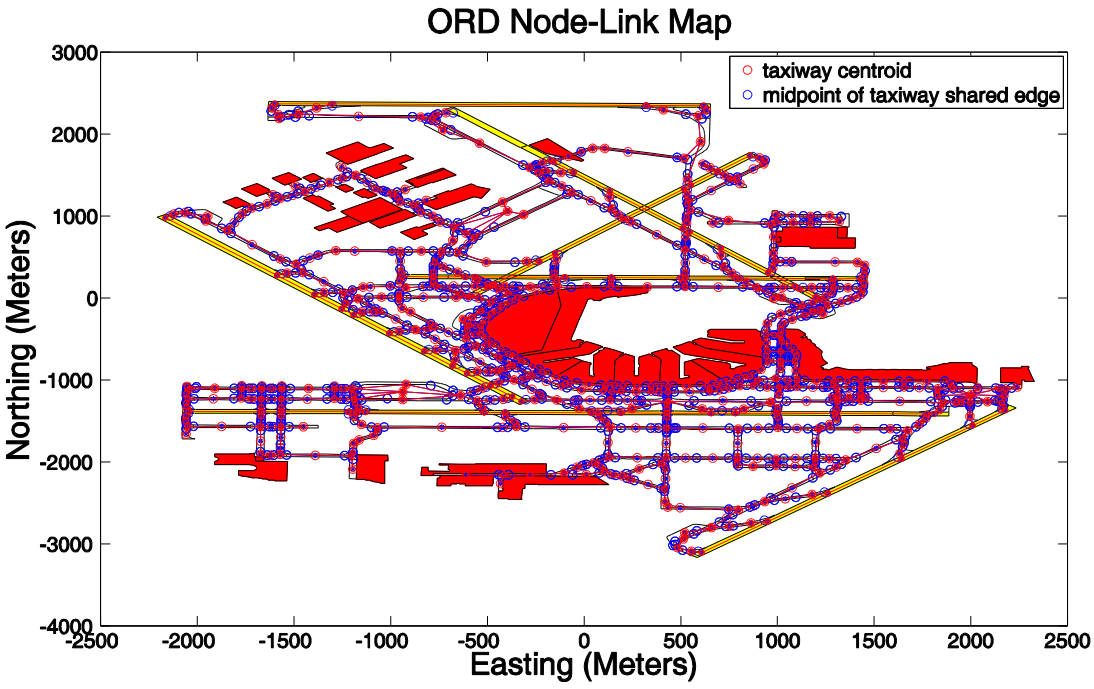

Figure B-34 Chicago O'Hare International Airport Node-Link Network

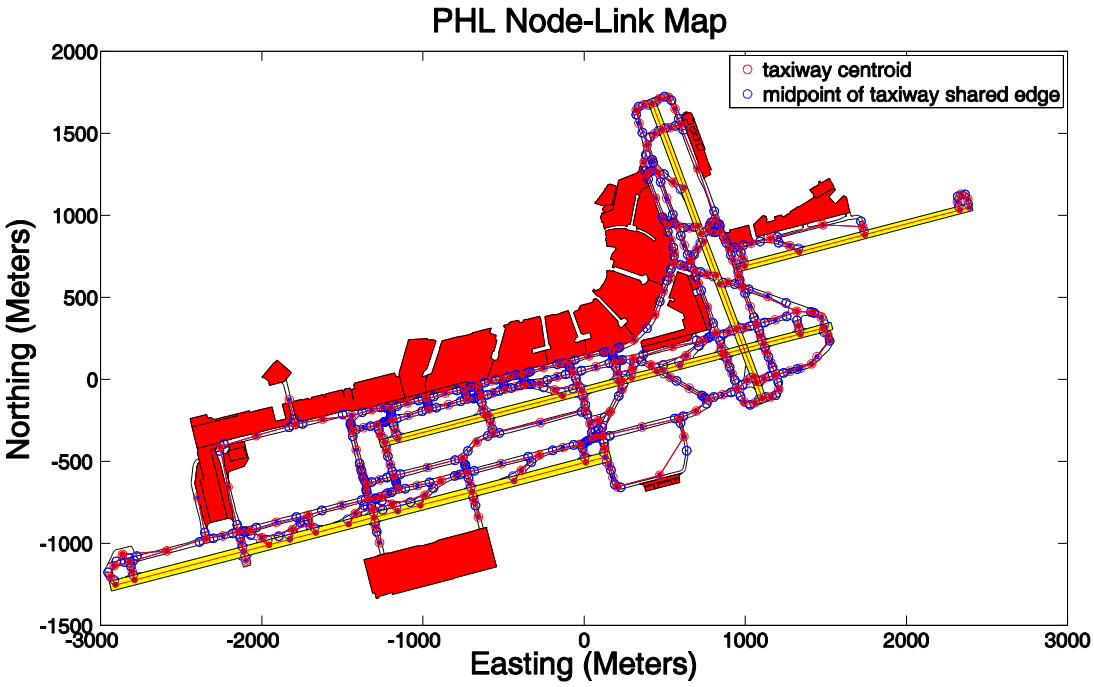

Figure B-35 Philadelphia International Airport Node-Link Network

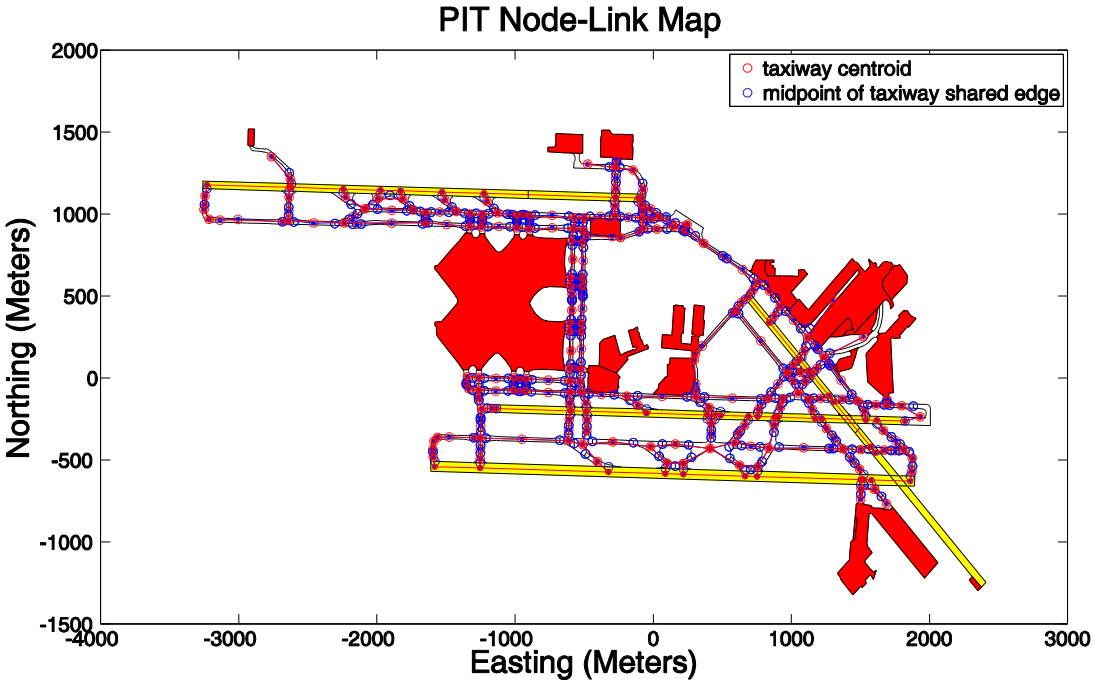

Figure B-36 Pittsburgh International Airport Node-Link Network

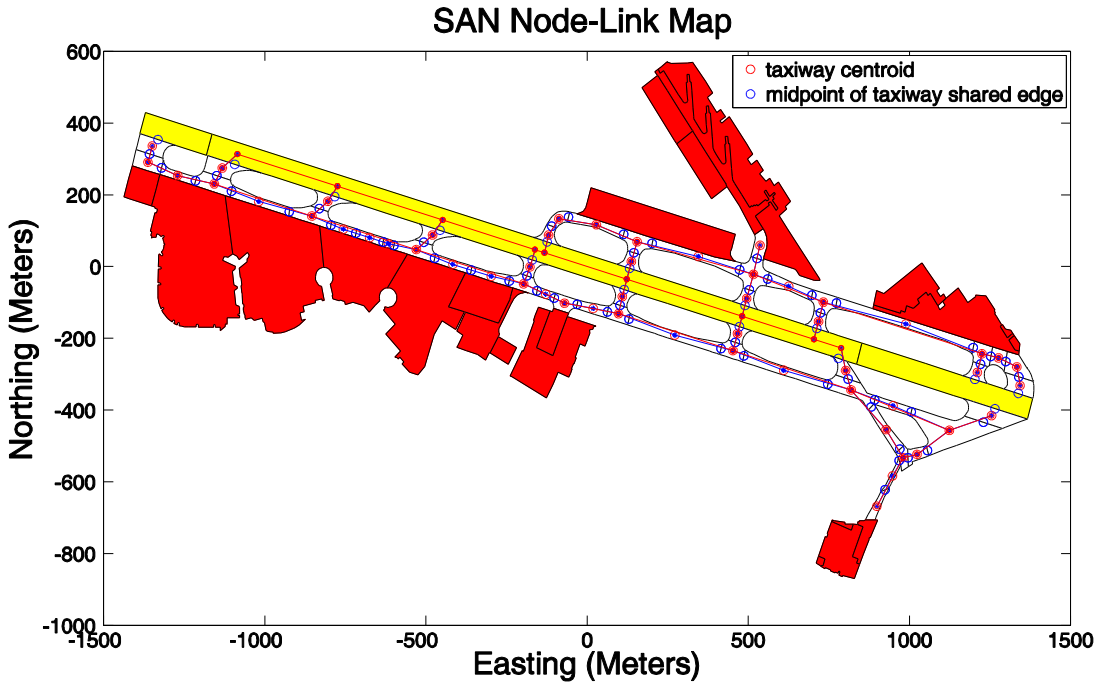

Figure B-37 San Diego International Airport Node-Link Network

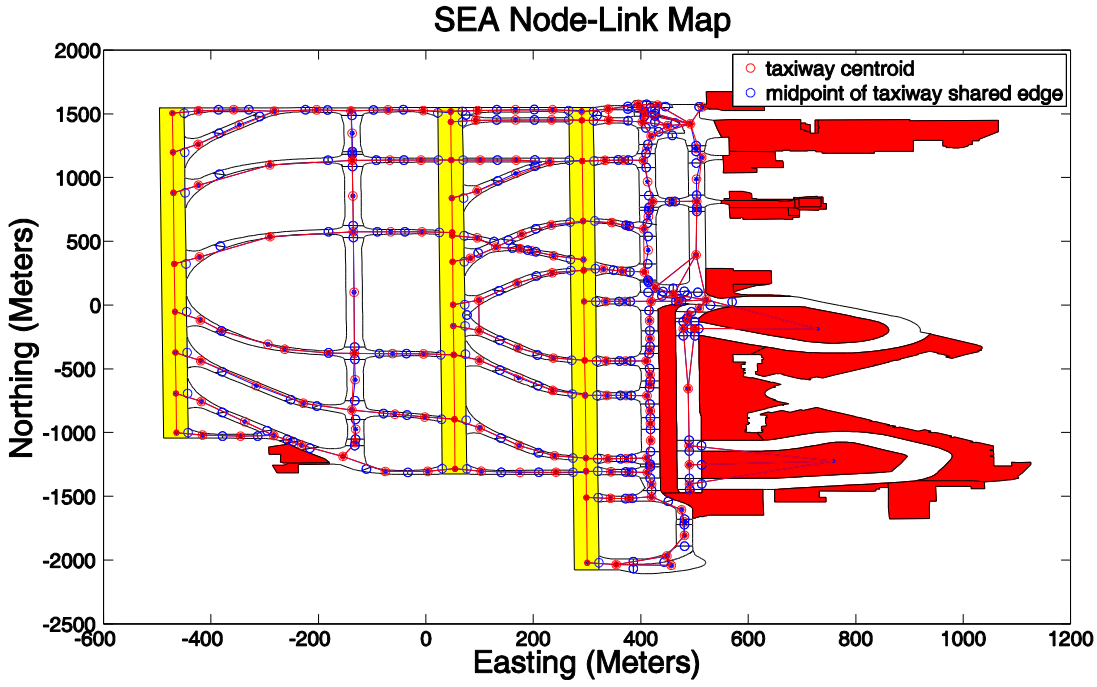

Figure B-38 Seattle-Tacoma International Airport Node-Link Network

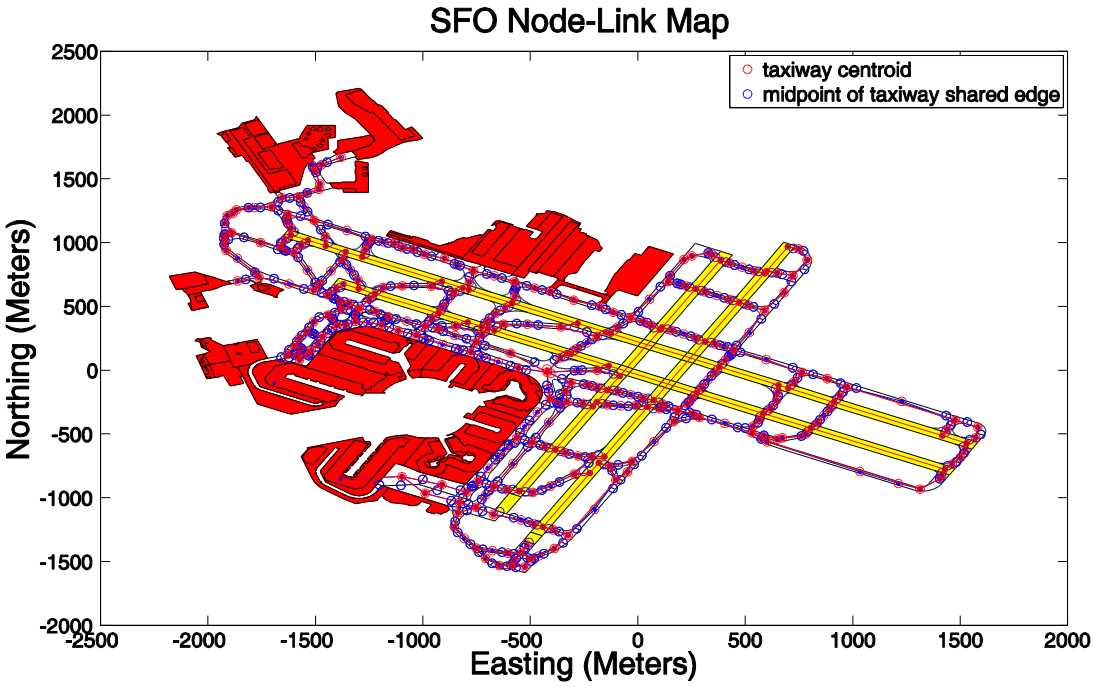

Figure B-39 San Francisco International Airport Node-Link Network

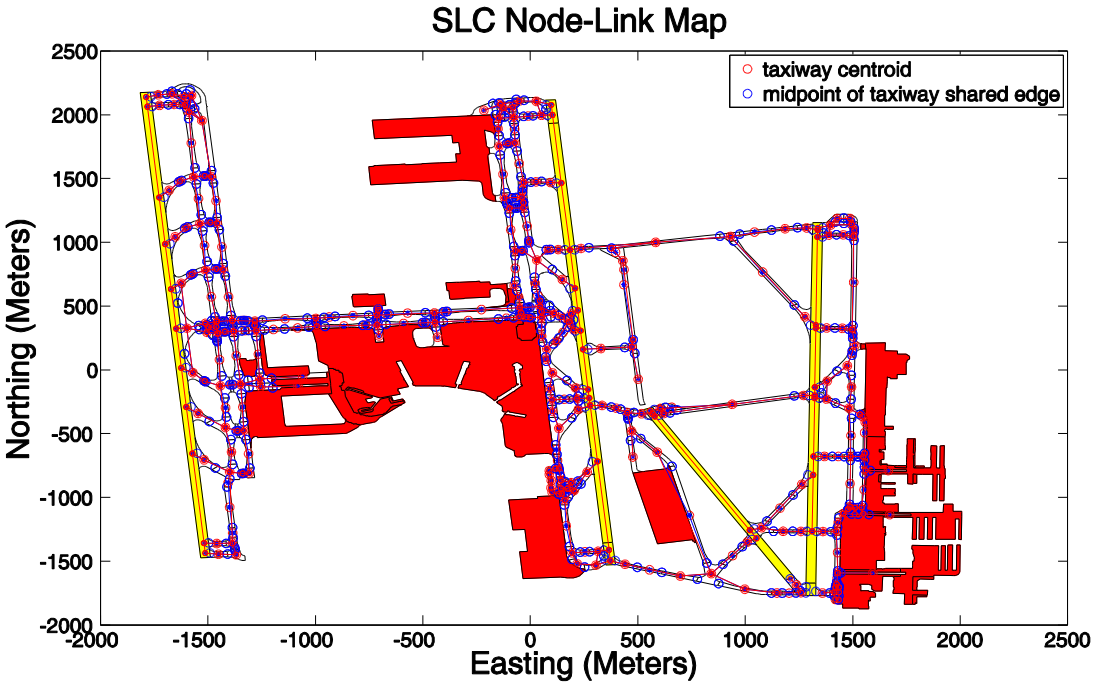

Figure B-40 Salt Lake City International Airport Node-Link Network

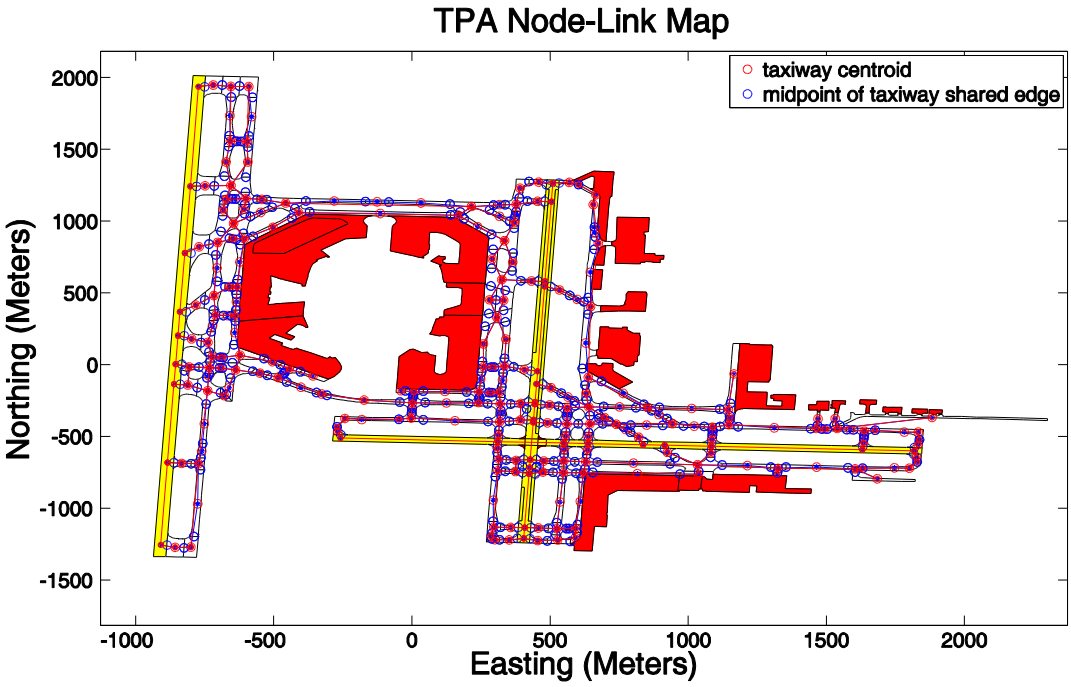

Figure B-41 Tampa International Airport Node-Link Network

C. Source Code Guideline

The Taxiway Toolkit GUI calls 43 Matlab functions. There are 10 functions called directly in the GUI main script. The core code of the GUI main script (TaxiwayToolkit.m) is shown below.

```
airport = aixm2struct(AIXM_dir*);       %load AIXM data to Matlab struct file

airport_LCS = WGS2LCS(airport);     %convert airport coordinate system

airport_LCS = SharePoint(airport_LCS);      %get mid points of shared edges

airport_LCS = GetCentroid(airport_LCS);     %get centroids of all polygons

plotNet(airport_LCS);       %plot airport

[taxiwayFile,runwayList] = CreateTaxiways(airport_LCS,ADSIM_dir);
%create ADSIM+ input file taxiways.xml, which contains taxiway network

[edgeListSpf,edgeSpf,nodeListSpf,nodeSpf,adjMatrixSpf,runwayList] =
simplifyNet(airport_LCS,runwayList,taxiwayFile);
%simplify the network by deleting unnecessary nodes

runwayFile = CreateRunways(nodeListSpf,nodeSpf,runwayList,ADSIM_dir**);
%create ADSIM+ input file runways.xml

gateFile = CreateGates2(airport_LCS,nodeListSpf,nodeSpf,ADSIM_dir);
%create the gates.xml file

CreateTaxipaths(runwayFile,gateFile,edgeListSpf,edgeSpf,nodeListSpf,nodeSpf,a
djMatrixSpf,ADSIM_dir);
%find all the possible shortest paths between runway exits and gates, and
store them to taxipaths.xml.
```

*, **AIXM_dir, ADSIM-dir are the user defined AIXM data and ADISM+ data directories.

Table C-1 shows a brief description of the 43 Matlab functions used in the *Taxiway Toolkit*.

Table C-1 Taxiway Toolkit Functions

Main Function	Sub-Function	Description
aixm2struct		Converts an aixm.xml file into a Matlab structure file
	parseChildNodes	Search and parse the node children in the tree structure of the xml file
	getNodeData	Create structure of node info
	parseAttributes	Create attributes structure
WGS2LCS		Convert airport and/or gate data under WGS coordinate system into a UTM based local coordinate system (LCS), which use the airport centroid as the origin. (unit is meter)

	wgs2utm	Convert WGS84 coordinates (Latitude, Longitude) into UTM coordinates (northing, easting) according to (optional) input UTM zone and hemisphere. Author: Alexandre Schimel MetOcean Solutions Ltd New Plymouth, New Zealand
	importAirport	Import numeric data from the ADSIM+ Airport database csv file as column vectors
SharePoint		Find midpoint of shared edge and add it to airport structure file as a field
	distancePointPolygon	Compute the shortest distance between a point and a polygon
	polylineCentroid	Compute the centroid of a curve defined by a series of points
	distancePointPolyline	Compute the shortest distance between a point and a polyline
	distancePointEdge	Compute the minimum distance between a point and an edge
GetCentroid		1. Get centroid of taxiway element of airport 2. Measure the length of taxiway element and width of runway 3. Add .centroid field to airport structure file 4. Add .length field to airport structure file
	polygeom	polygeom (X, Y) returns area, X centroid, Y centroid and perimeter for the planar polygon specified by vertices in vectors X and Y. Author: H.J. Sommer III, Ph.D., Professor of Mechanical Engineering, 337 Leonhard Bldg The Pennsylvania State University, University Park, PA 16802
	distancePointPolygon	Compute the shortest distance between a point and a polygon
	distancePointPolyline	Compute the shortest distance between a point and a polyline
	distancePointEdge	Compute minimum distance between a point and an edge
	intersectLinePolygon	Find the intersection points between a line and a polygon
	edgeToLine	Convert an edge to a straight line
	intersectLines	Return all intersection points of N lines in 2-D
	linePosition	Position of a point on a line
	polylineCentroid	Compute the centroid of a curve defined by a series of points
	findCent	Find multiple centroids for irregular shape polygon (i), where the polygon is a taxiway link.
	RemoveTarget	Remove the Target cell from the Origin cell array and keep the sort of Origin cell array
	intersectEdgePolygon	Find the intersection point of an edge with a polygon

	angle2Points	Compute the horizontal angle between two points
	distancePoints	Compute the distance between two points
plotNet		Plot the airport polygons and nodes in the node-link network
	PlotAirport	Plot airport polygons and fill them with different colors
CreateTaxiways		Write taxiway and gate position information into ADSIM+ input file taxiways.xml, where the taxiway position info from Airport structure file is required and gate position info from Gate structure file is optional
	fDrawMulti	Draw taxiway link with multiple nodes
	importRunway	Import numeric data from the runway database csv file as column vectors
	importAirport	Import numeric data from the Airport database csv file as column vectors
	pointProjectEdge	Project points on to an edge
simplifyNet		1. Read taxiways.xml into Matlab structure file for network analysis purpose. 2. Delete the node only connected to two edges, while the angle between the two edges is around 180 3. Connect two node if they are close enough, via subtitle either one. 4. Add edges between runway exits inside runway. 5. Rewrite the simplified network into taxiways.xml
	readEdgeList	Read taxiways.xml file into Matlab struct file
	parseChildNodes	Search and parse the node children in the tree structure of the xml file
	getNodeData	Create structure of node info
	parseAttributes	Create attributes structure
	pointProjectEdge	Project points on to an edge
CreateRunways		Write runway information into ADSIM+ input file runways.xml
gateinfo2struct*		Import gateinfo from the template spreadsheet, which contains gate info data into gateinfo struct file
CreateGates**		Write gate information into ADSIM+ input file gates.xml
CreateGates2		Create dummy gates and write them into ADSIM+ input file gates.xml
CreateTaxipaths		1. Read runways.xml and gates.xml as the start and end points of taxipaths 2. Calculate shortest paths (based on actual distance, which can be substituted as travel time if related analysis is available between all the gates and runway exits with Dijkstra's algorithm. 3. Write taxipath into taxipaths.xml as input file for ADSIM+
	parseChildNodes	Search and parse the node children in the tree structure of the xml file
	getNodeData	Create structure of node info
	parseAttributes	Create attributes structure

*, ** Since the user defined gateinfo functionality is disabled in the current version of the *Taxiway Toolkit*, these two functions are not used.

D. Source Code

```
function [s] = aixm2struct(file)
%%
% [ s ] = aixm2struct( aixm file )
% Description:
%   Convert aixm.xml file into MATLAB structure
% Input:
%   file: the directory of the .xml aixm file.
%   The aixm file should contain the structure:
%       <XMLname attrib1="Some value">
%           <Element>Some text</Element>
%           <DifferentElement attrib2="2">Some more text</Element>
%       <DifferentElement attrib3="2" attrib4="1">Even more text</DifferentElement>
%       </XMLname>
% Output:
%   s: a structure file, which contains the fields:
%       s.IATA = airport IATA code;
%       s.ICAO = airport ICAO code;
%       s.element_feature_type = airport element type, which can be
'aixm_RunwayElement','aixm_TaxiwayElement',aixm_ApronElement';
%       s.element_sub_type = element subtype of RunwayElement, TaxiwayElement or ApronElement;
%       s.element_ID = Geographic ID derived from aixm file;
%       s.surface_char = element surface characteristic;
%       s.extent = location information of vertices on element edges;
%       s.label = element actual label
% Subfunction:
%   parseChildNodes(), getNodeData(), parseAttributes()
% Note the characters : ' ', - and . are not supported in structure fieldnames and
% are replaced by _
% Yang Zhang - 07/01/2014 - tested under MATLAB v8.1
%%
    if (nargin < 1)
        clc;
        help aixm2struct
        return
    end

    if (isempty(strfind(file,'.')))
        file = [file '.xml'];
    end

    if (exist(file,'file') == 0)
        error(['The file ' file ' could not be found']);
    end

    h = waitbar(0,'Loading','Name','Reading the AIXM Data...');
    xDoc = xmlread(file);       %read .xml file
    a = parseChildNodes(xDoc);      %parse all the child nodes of the .xml file to a temp struct
file: a
    s = struct;     %define the struct file: s
    j =1;       %initiate the index of struct file
    %run a loop to write all the useful information from struct file a to s
    NumOfElement = length(a.message_AIXMBasicMessage.message_hasMember);
    for i=1:NumOfElement
        waitbar(i/NumOfElement,h,sprintf('Preparing Index %g out of %g',i,NumOfElement));
        %save the IATA code of the airport to the struct file field IATA, if it exists.
        if isfield(a.message_AIXMBasicMessage.message_hasMember{1,
1}.aixm_AirportHeliport.aixm_timeSlice.aixm_AirportHeliportTimeSlice, 'aixm_designatorIATA')
            s(j,1).IATA = {a.message_AIXMBasicMessage.message_hasMember{1,
1}.aixm_AirportHeliport.aixm_timeSlice.aixm_AirportHeliportTimeSlice.aixm_designatorIATA.Text};
        else
            s(j,1).IATA = {[]};
        end
        %save the ICAO code of the airport to the struct file field ICAO, if it exists.
        if isfield(a.message_AIXMBasicMessage.message_hasMember{1,
1}.aixm_AirportHeliport.aixm_timeSlice.aixm_AirportHeliportTimeSlice, 'aixm_designator')
            s(j,1).ICAO = {a.message_AIXMBasicMessage.message_hasMember{1,
1}.aixm_AirportHeliport.aixm_timeSlice.aixm_AirportHeliportTimeSlice.aixm_designator.Text};
```

```
        else
            s(j,1).ICAO = {[]};
        end
        %if the polygon is a runway element, save the useful info from a to s
        if isfield(a.message_AIXMBasicMessage.message_hasMember{1, i} ,'aixm_RunwayElement')
            s(j,1).element_feature_type =
fieldnames(a.message_AIXMBasicMessage.message_hasMember{1, i});
            s(j,1).element_sub_type = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_type.Text};
            s(j,1).element_ID = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_extent.aixm_ElevatedSurface
.Attributes.gml_id};
            if isfield(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice,'aixm_surfaceProperties')
                s(j,1).surface_char = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_surfaceProperties.aixm_Surf
aceCharacteristics};
            else
                s(j,1).surface_char = {[]};
            end
            m = length(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_associatedRunway);
            if m == 1
                s(j,1).label = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_associatedRunway.Attributes
.xlink_title};
            else
                for n =1:1:m;
                    s(j,1).label{1,n} = char(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_associatedRunway{1,n}.Attri
butes.xlink_title);
                end
            end
            b=str2num(regexprep(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_RunwayElement.aixm_timeSlice.aixm_RunwayElementTimeSlice.aixm_extent.aixm_ElevatedSurface
.gml_patches.gml_PolygonPatch.gml_exterior.gml_LinearRing.gml_posList.Text, ':','_COLON_', 'once',
'ignorecase'));
            s(j,1).extent = {(reshape(b,2,length(b)/2))'};  %reshape the vertices coordinate from
one row array to a n*2 matrix (n is the number of the vertices)
%             x = s(j, 1).extent{1, 1}(:,1);
%             y = s(j, 1).extent{1, 1}(:,2);
%             [ geom, iner, cpmo ] = polygeom( x, y );
%             x_cen = geom(2);
%             y_cen = geom(3);% get geometry
%             s(j, 1).centroid = {[x_cen, y_cen]};
            j=j+1;
        end
        %if the polygon is a taxiway element, save the useful info from a to s
        if isfield(a.message_AIXMBasicMessage.message_hasMember{1, i} ,'aixm_TaxiwayElement');
            s(j,1).element_feature_type=
fieldnames(a.message_AIXMBasicMessage.message_hasMember{1, i});
            if isfield(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice,'aixm_type')
                s(j,1) .element_sub_type= {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice.aixm_type.Text []};
            else
                s(j,1) .element_sub_type= {char('OTHER') []};
            end
            s(j,1) .element_ID= {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice.aixm_extent.aixm_ElevatedSurfa
ce.Attributes.gml_id};
            if isfield(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice,'aixm_surfaceProperties')
                s(j,1).surface_char = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice.aixm_surfaceProperties.aixm_Su
rfaceCharacteristics};
            else
                s(j,1).surface_char = {[]};
            end
```

```
            s(j,1).label = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice.aixm_associatedTaxiway.Attribu
tes.xlink_title};
            b=str2num(regexprep(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_TaxiwayElement.aixm_timeSlice.aixm_TaxiwayElementTimeSlice.aixm_extent.aixm_ElevatedSurfa
ce.gml_patches.gml_PolygonPatch.gml_exterior.gml_LinearRing.gml_posList.Text, ':','_COLON_',
'once', 'ignorecase'));
            s(j,1).extent = {(reshape(b,2,length(b)/2))'};
%             x = s(j, 1).extent{1, 1}(:,1);
%             y = s(j, 1).extent{1, 1}(:,2);
%             [ geom, iner, cpmo ] = polygeom( x, y );
%             x_cen = geom(2);
%             y_cen = geom(3);% get geometry
%             s(j, 1).centroid = {[x_cen, y_cen]};
            j=j+1;
        end
        %if the polygon is a apron element, save the useful info from a to s
        if isfield(a.message_AIXMBasicMessage.message_hasMember{1, i} ,'aixm_ApronElement');
            s(j,1).element_feature_type=
fieldnames(a.message_AIXMBasicMessage.message_hasMember{1, i});
            s(j,1).element_sub_type = {'n/a'};
            s(j,1).element_ID = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_ApronElement.aixm_timeSlice.aixm_ApronElementTimeSlice.aixm_extent.aixm_ElevatedSurface.A
ttributes.gml_id};
            if isfield(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_ApronElement.aixm_timeSlice.aixm_ApronElementTimeSlice,'aixm_surfaceProperties')
                s(j,1).surface_char = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_ApronElement.aixm_timeSlice.aixm_ApronElementTimeSlice.aixm_surfaceProperties.aixm_Surfac
eCharacteristics};
            else
                s(j,1).surface_char = {[]};
            end
            s(j,1).label = {a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_ApronElement.aixm_timeSlice.aixm_ApronElementTimeSlice.aixm_associatedApron.Attributes.xl
ink_title};
            b=str2num(regexprep(a.message_AIXMBasicMessage.message_hasMember{1,
i}.aixm_ApronElement.aixm_timeSlice.aixm_ApronElementTimeSlice.aixm_extent.aixm_ElevatedSurface.g
ml_patches.gml_PolygonPatch.gml_exterior.gml_LinearRing.gml_posList.Text, ':','_COLON_', 'once',
'ignorecase'));
            s(j,1).extent = {(reshape(b,2,length(b)/2))'};
            j=j+1;
        end
    end
    delete(h);
    for i = 1:size(s,1)
        if isempty(s(i).IATA{1})&&s(i).ICAO{1}(1) == 'K'
            s(i).IATA{1} = s(i).ICAO{1}(2:4);
        end
    end
end

% ----- Subfunction parseChildNodes -----
function [children,ptext] = parseChildNodes(theNode)
    % Recurse over node children.
    children = struct;
    ptext = [];
    if theNode.hasChildNodes
        childNodes = theNode.getChildNodes;
        numChildNodes = childNodes.getLength;

        for count = 1:numChildNodes
            theChild = childNodes.item(count-1);
            [text,name,attr,childs] = getNodeData(theChild);

            if (~strcmp(name,'#text') && ~strcmp(name,'#comment'))
                %XML allows the same elements to be defined multiple times,
                %put each in a different cell
                if (isfield(children,name))
                    if (~iscell(children.(name)))
                        %put existsing element into cell format
                        children.(name) = {children.(name)};
```

```
                    end
                    index = length(children.(name))+1;
                    %add new element
                    children.(name){index} = childs;
                    if(~isempty(text))
                        children.(name){index}.('Text') = text;
                    end
                    if(~isempty(attr))
                        children.(name){index}.('Attributes') = attr;
                    end
                else
                    %add previously unkown new element to the structure
                    children.(name) = childs;
                    if(~isempty(text))
                        children.(name).('Text') = text;
                    end
                    if(~isempty(attr))
                        children.(name).('Attributes') = attr;
                    end
                end
            elseif (strcmp(name,'#text'))
                %this is the text in an element (i.e. the parentNode)
                if (~isempty(regexprep(text,'[\s]*','')))
                    ptext = text;
                end
            end
        end
    end
end

% ----- Subfunction getNodeData -----
function [text,name,attr,childs] = getNodeData(theNode)
    % Create structure of node info.

    %make sure name is allowed as structure name
    name = regexprep(char(theNode.getNodeName),'[-:.]','_');

    attr = parseAttributes(theNode);
    if (isempty(fieldnames(attr)))
        attr = [];
    end

    %parse child nodes
    [childs,text] = parseChildNodes(theNode);

    if (isempty(fieldnames(childs)))
        %get the data of any childless nodes
        try
            %faster then if any(strcmp(methods(theNode), 'getData'))
            text = char(theNode.getData);
        catch
            %no data
        end
    end

end

% ----- Subfunction parseAttributes -----
function attributes = parseAttributes(theNode)
    % Create attributes structure.

    attributes = struct;
    if theNode.hasAttributes
       theAttributes = theNode.getAttributes;
       numAttributes = theAttributes.getLength;

       for count = 1:numAttributes
          attrib = theAttributes.item(count-1);
          attr_name = regexprep(char(attrib.getName),'[-:.]','_');
          attributes.(attr_name) = char(attrib.getValue);
       end
```

```
    end
end
```

```
function varargout = WGS2LCS (varargin)
%%
% [airportLCS,gateLCS] = WGS2LCS(airportWGS,gateWGS)
% airportLCS = WGS2LCS(airportWGS)
% gateLCS = WGS2LCS(gateWGS)
%
% Description:
%   Convert airport and/or gate data under WGS coordinate system into a UTM
%   based local coordinate system (LCS), which use the airport centroid as the
%   origin. (unit is meter)
% Input:
%   AirportWGS: airport structure file generated from aixm file and
%               processed by SharePoint() and GetCentroid(), whose all the
%               location related data under WGS.
%   GateWGS: airport gate infomation structure file generated from gateinfo
%            and processed by gateinfo2struct(), whose all the
%            location related data under WGS.
%   Airport.csv: Airport database from ADSIM+ built-in database.
% Output:
%   AirportLCS: airport structure file, whose all the location related data
%               under UTM based local coordinate system.
%   GateLCS: gate dtructure file, whose all the location related data under
%               UTM based local coordinate system.
% External functions required:
%   wgs2utm(), importAirport()
% Yang Zhang - 02/20/2014 - tested under MATLAB v8.2
%%
h = waitbar(0,'Loading','Name','Converting the Coordinate System');

narginchk(1, 2)
if nargout < 2
    %AirportLCS = WGS2LCS(AirportWGS) or GateLCS = WGS2LCS(GateWGS)
    if size(varargin{1}) == [1,1] %Gate structure file is organized as 1*1 structure
        GateWGS = varargin{1};
    else
        AirportWGS = varargin{1};
    end
else
    %[AirportLCS,GateLCS] = WGS2LCS(AirportWGS,GateWGS)
    AirportWGS = varargin{1};
    GateWGS = varargin{2};
end

if exist('AirportWGS','var')
    AirportLCS = AirportWGS;
    IATA = strcat('"',AirportWGS(1, 1).IATA{1, 1},'"');     %find airport from the ADSIM+
database, and use the airport location point in database as the origin of the local coodinate
system
    [IATA_CODE,ICAO_CODE,FAA_CODE,LATITUDE,LONGITUDE] = importAirport('Airport.csv');

    ind = find(ismember(IATA_CODE,IATA));       %index of the target airport in the ADSIM+ built-
in database

    if isempty(ind)
        msgbox('Cannot find this airport in database');
        return
    else
        UTMzone = utmzone(LATITUDE(ind),LONGITUDE(ind));        %find the UTMzone of the airport
        UTMzone = regexp(UTMzone,'\d+','match');
        UTMzone = str2num(UTMzone{1});%cut UTMzone into the format required by wgs2utm()
        [OriginUTM(1,1),OriginUTM(1,2)] =
wgs2utm(LATITUDE(ind),LONGITUDE(ind),UTMzone,'N'); %change 'N' for proper hemisphere if used
outside US continenet
    end

    index1 = length(AirportWGS);        %total number of elements at the airport
    %run a loop to do projection for all elements from WGS to LCS
    for i = 1:1:index1
        waitbar(i/index1,h,sprintf('Preparing Index %g out of %g',i,index1));
        ind1 = size(AirportWGS(i, 1).extent{1, 1});     %total number of the vertices on the
edges of the element
```

```
        %run a loop to project all the vertice on the edges of the element
        for j = 1:1:ind1(1,1)
            [x, y] =
wgs2utm(AirportWGS(i,1).extent{1,1}(j,2),AirportWGS(i,1).extent{1,1}(j,1),UTMzone,'N');        %p
roject to UTM
            AirportLCS(i,1).extent{1,1}(j,:) = ([x, y]-
[OriginUTM(1,1),OriginUTM(1,2)]);        %convert to LCS
        end
        %If the field sharemid exists, run a loop to project all the
        %midpoints on the shared edges from WGS to LCS
        if isfield(AirportWGS,'sharemid')
            ind2 = size(AirportWGS(i,1).sharemid);
            for j = 1:1:ind2(1,1)
                if ~isempty(AirportWGS(i,1).sharemid{j,1})
                    [x, y] =
wgs2utm(AirportWGS(i,1).sharemid{j,1}(1,2),AirportWGS(i,1).sharemid{j,1}(1,1),UTMzone,'N');
                  AirportLCS(i,1).sharemid{j,1}(1,:) = ([x, y] -
[OriginUTM(1,1),OriginUTM(1,2)]);
                end
            end
        end
        %If the field centroid exists, run a loop to project the centroid of the element from WGS
to LCS
        if isfield(AirportWGS,'centroid')
            ind3 = size(AirportWGS(i,1).centroid{1, 1});
            for j = 1:1:ind3(1,1)
                [x, y] =
wgs2utm(AirportWGS(i,1).centroid{1,1}(j,2),AirportWGS(i,1).centroid{1,1}(j,1),UTMzone,'N');
                AirportLCS(i,1).centroid{1,1}(j,1:2) =([x, y]-[OriginUTM(1,1),OriginUTM(1,2)]);
            end
        end
    end
    %If the GateWGS is part of the input variables, project all the nodes
    %of the variable from WGS to LCS
    if exist('GateWGS','var')
        GateLCS = GateWGS;
        index2 = length(GateWGS.Position);
        for i = 1:1:index2
            [x, y] = wgs2utm(GateWGS.Position(i,2),GateWGS.Position(i,1),UTMzone,'N');
            GateLCS.Position(i,1:2) = ([x, y] - [OriginUTM(1,1),OriginUTM(1,2)]);
            [x1, y1] = wgs2utm(GateWGS.Intersection(i,2),GateWGS.Intersection(i,1),UTMzone,'N');
            GateLCS.Intersection(i,1:2) = ([x1, y1] - [OriginUTM(1,1),OriginUTM(1,2)]);
        end
        varargout{2} = GateLCS;
    end
    varargout{1} = AirportLCS;
else    %GateLCS = WGS2LCS(GateWGS)
    if exist('GateWGS','var')
        GateLCS = GateWGS;
        ICAO = strcat('"',GateWGS.Airport,'"');     %Gateinfo use ICAO code instead of IATA used
by Airport structure file
        [IATA_CODE,ICAO_CODE,FAA_CODE,LATITUDE,LONGITUDE] = importAirport('Airport.csv');

        ind = find(ismember(ICAO_CODE,ICAO));
        if isempty(ind)
            msgbox('Cannot find this airport in database');
            return
        else
            UTMzone = utmzone(LATITUDE(ind),LONGITUDE(ind));
            UTMzone = str2num(UTMzone(1:2));
            [OriginUTM(1,1),OriginUTM(1,2)] =
wgs2utm(LATITUDE(ind),LONGITUDE(ind),UTMzone,'N');        %change 'N' for proper hemisphere if
used outside US continenet
        end
        index2 = length(GateWGS.Position);
        for i = 1:1:index2
            [x, y] = wgs2utm(GateWGS.Position(i,2),GateWGS.Position(i,1),UTMzone,'N');
            GateLCS.Position(i,1:2) = ([x, y] - [OriginUTM(1,1),OriginUTM(1,2)]);
            [x1, y1] = wgs2utm(GateWGS.Intersection(i,2),GateWGS.Intersection(i,1),UTMzone,'N');
            GateLCS.Intersection(i,1:2) = ([x1, y1] - [OriginUTM(1,1),OriginUTM(1,2)]);
        end
```

```
        varargout{1} = GateLCS;
    end
end
delete(h);
end

function [IATA_CODE,ICAO_CODE,FAA_CODE,LATITUDE,LONGITUDE] = importAirport(filename, startRow,
endRow)
%IMPORTAirport Import numeric data from a text file as column vectors.
%   [IATA_CODE,ICAO_CODE,FAA_CODE,ICAO_REGIONELEVATION,LATITUDE,LONGITUDE]
%   = importAirport(FILENAME) Reads data from text file FILENAME for the
%   default selection.
%
%   [IATA_CODE,ICAO_CODE,FAA_CODE,ICAO_REGIONELEVATION,LATITUDE,LONGITUDE]
%   = importAirport(FILENAME, STARTROW, ENDROW) Reads data from rows STARTROW
%   through ENDROW of text file FILENAME.
%
% Example:
%   [IATA_CODE,ICAO_CODE,FAA_CODE,ICAO_REGIONELEVATION,LATITUDE,LONGITUDE]
%   = importAirport('Airport.csv',1, 14277);
%
%    See also TEXTSCAN.

% Auto-generated by MATLAB on 2014/01/16 17:23:18

%% Initialize variables.
delimiter = ',';
if nargin<=2
    startRow = 1;
    endRow = inf;
end

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'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 = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter', delimiter,
'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1, 'Delimiter',
delimiter, 'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[11,12,13,14,15,16,17,22,23]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
```

```
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-
+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [11,12,13,14,15,16,17,22,23]);
rawCellColumns = raw(:, [1,2,3,4,5,6,7,8,9,10,18,19,20,21]);


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); % Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
IATA_CODE = rawCellColumns(:, 3);
ICAO_CODE = rawCellColumns(:, 4);
FAA_CODE = rawCellColumns(:, 5);
ELEVATION = cell2mat(rawNumericColumns(:, 1));
LATITUDE = cell2mat(rawNumericColumns(:, 2));
LONGITUDE = cell2mat(rawNumericColumns(:, 3));
end

function  [x,y,utmzone,utmhemi] = wgs2utm(Lat,Lon,utmzone,utmhemi)
% -------------------------------------------------------------------------
% [x,y,utmzone] = wgs2utm(Lat,Lon,Zone)
%
% Description:
%    Convert WGS84 coordinates (Latitude, Longitude) into UTM coordinates
%    (northing, easting) according to (optional) input UTM zone and
%    hemisphere.
%
% Input:
%    Lat: WGS84 Latitude scalar, vector or array in decimal degrees.
%    Lon: WGS84 Longitude scalar, vector or array in decimal degrees.
%    utmzone (optional): UTM longitudinal zone. Scalar or same size as Lat
%       and Lon.
%    utmhemi (optional): UTM hemisphere as a single character, 'N' or 'S',
%       or array of 'N' or 'S' characters of same size as Lat and Lon.
%
% Output:
%    x: UTM easting in meters.
%    y: UTM northing in meters.
%    utmzone: UTM longitudinal zone.
%    utmhemi: UTM hemisphere as array of 'N' or 'S' characters.
%
% Author notes:
%    I downloaded and tried deg2utm.m from Rafael Palacios but found
%    differences of up to 1m with my reference converters in southern
```

```
%    hemisphere so I wrote my own code based on "Map Projections - A
%    Working Manual" by J.P. Snyder (1987). Quick quality control performed
%    only by comparing with LINZ converter
%    (www.linz.govt.nz/apps/coordinateconversions/) and Chuck Taylor's
%    (http://home.hiwaay.net/~taylorc/toolbox/geography/geoutm.html) on a
%    few test points, so use results with caution. Equations not suitable
%    for a latitude of +/- 90deg.
%
%    UPDATE: Following requests, this new version allows forcing UTM zone
%    in input.
%
% Examples:
%
%    % set random latitude and longitude arrays
%    Lat= 90.*(2.*rand(3)-1)
%    Lon= 180.*(2.*rand(3)-1)
%
%    % let the function find appropriate UTM zone and hemisphere from data
%    [x1,y1,utmzone1,utmhemi1] = wgs2utm(Lat,Lon)
%
%    % forcing unique UTM zone and hemisphere for all data entries
%    % note: resulting easting and northing are way off the usual values
%    [x2,y2,utmzone2,utmhemi2] = wgs2utm(Lat,Lon,60,'S')
%
%    % forcing different UTM zone and hemisphere for each data entry
%    % note: resulting easting and northing are way off the usual values
%    utmzone = floor(59.*rand(3))+1
%    utmhemi = char(78 + 5.*round(rand(3)))
%    [x3,y3,utmzone3,utmhemi3] = wgs2utm(Lat,Lon,utmzone,utmhemi)
%
% Author:
%   Alexandre Schimel
%   MetOcean Solutions Ltd
%   New Plymouth, New Zealand
%
% Version 2:
%   February 2011
%-------------------------------------------------------------------------
%% Argument checking
if ~sum(double(nargin==[2,4]))
    error('Wrong number of input arguments');return
end
n1=size(Lat);
n2=size(Lon);
if (n1~=n2)
    error('Lat and Lon should have same size');return
end
if exist('utmzone','var') && exist('utmhemi','var')
    n3=size(utmzone);
    n4=size(utmhemi);
    if (sort(n3)~=sort(n4))
        error('utmzone and utmhemi should have same size');return
    end
    if max(n3)~=1 && max(n3)~=max(n1)
        error('utmzone should have either same size as Lat and Long, or size=1');return
    end
end

% expand utmzone and utmhemi if needed
if exist('utmzone','var') && exist('utmhemi','var')
    n3=size(utmzone);
    n4=size(utmhemi);
    if n3==[1 1]
        utmzone = utmzone.*ones(size(Lat));
        utmhemi = char(utmhemi.*ones(size(Lat)));
    end
end

%% coordinates in radians
lat = Lat.*pi./180;
lon = Lon.*pi./180;
```

```
%% WGS84 parameters
a = 6378137;            %semi-major axis
b = 6356752.314245;     %semi-minor axis
% b = 6356752.314140;  %GRS80 value, originally used for WGS84 before refinements
e = sqrt(1-(b./a).^2); % eccentricity

%% UTM parameters
% lat0 = 0;                % reference latitude, not used here
if exist('utmzone','var')
    Lon0 = 6.*utmzone-183; % reference longitude in degrees
else
    Lon0 = floor(Lon./6).*6+3; % reference longitude in degrees
end
lon0 = Lon0.*pi./180;       % in radians
k0 = 0.9996;                % scale on central meridian

FE = 500000;              % false easting
if exist('utmhemi','var')
    FN = double(utmhemi=='S').*10000000;
else
    FN = (Lat < 0).*10000000; % false northing
end

%% Equations parameters
eps = e.^2./(1-e.^2);  % e prime square
% N: radius of curvature of the earth perpendicular to meridian plane
% Also, distance from point to polar axis
N = a./sqrt(1-e.^2.*sin(lat).^2);
T = tan(lat).^2;
C = ((e.^2)./(1-e.^2)).*(cos(lat)).^2;
A = (lon-lon0).*cos(lat);
% M: true distance along the central meridian from the equator to lat
M = a.*(  ( 1 - e.^2./4 - 3.*e.^4./64 - 5.*e.^6./256 )  .* lat         ...
        -( 3.*e.^2./8 + 3.*e.^4./32 + 45.*e.^6./1024 ) .* sin(2.*lat) ...
        +( 15.*e.^4./256 + 45.*e.^6./1024 )            .* sin(4.*lat) ...
        -(35.*e.^6./3072 )                             .* sin(6.*lat) );

%% easting
x = FE + k0.*N.*(                                  A       ...
                + (1-T+C)                        .* A.^3./6 ...
                + (5-18.*T+T.^2+72.*C-58.*eps) .* A.^5./120 );

%% northing
% M(lat0) = 0 so not used in following formula
y = FN + k0.*M + k0.*N.*tan(lat).*(                                    A.^2./2  ...
                                  + (5-T+9.*C+4.*C.^2)               .* A.^4./24 ...
                                  + (61-58.*T+T.^2+600.*C-330.*eps) .* A.^6./720 );

%% UTM zone
if exist('utmzone','var') && exist('utmhemi','var')
    utmzone = utmzone;
    utmhemi = utmhemi;
else
   utmzone = floor(Lon0./6)+31;
   utmhemi = char( 83.* (Lat < 0) + 78.* (Lat >= 0) );
end
```

```
function [Airport] = SharePoint(Airport)
%%
% Description:
%   find mid point of shared edge and add it to airport structure file
% Input:
%   airport structure file generated by aixm2struct()
% Output:
%   airport structure file contain .sharemid field
% External functions required:
%   distancePointPolygon(), polylineCentroid()
% Yang Zhang - 06/25/2013 - tested under MATLAB v8.1
%%
index = length(Airport);        %total number of elements at the airport
%initiate cell for new field sharemid to store mid point of shared edge
for i = 1:1:index;
    Airport(i,1).sharemid = cell(index, 2);
    Airport(i,1).sharemid(:,2) = {0};
end
h = waitbar(0,'1','Name','Processing AIXM Data...');    %wait bar to indicate progress of the
function
%run a loop to find the mid points of all the shared edges by taxiway
%elements or runway elements
for i = 1:1:index;  %i is the index of the element A
    waitbar(i/index,h,sprintf('Preparing Index %g out of %g',i,index))
    if ~strcmp(Airport(i,1).element_feature_type,'aixm_ApronElement')       %not necessary to
find the mid point of the edges shared with apron element
        for j = 1:1:index;  %j is the index of the element B
            if i~=j&&~strcmp(Airport(j,1).element_feature_type,'aixm_ApronElement')     %element
A and B must not be identical and not necessary to find the mid point of the edges shared with
apron element
                dist = distancePointPolygon(Airport(i,1).extent{1,1},
Airport(j,1).extent{1,1});    %recognize vertices on the edge shared by element A and B
                p = dist < 0.5;     %0.5 meter is used as the default value to judge if the
vertices is shared by two elements
                Share = Airport(i,1).extent{1,1}(p,:);      %vertices on shared edge
                if ~isempty (Share)     %find the mid point if the shared edge exist
                    if size(Share,1)~=1
                        Airport(i,1).sharemid{j,2} = polylineLength(Share);       %shared edge
length
                        if Airport(i,1).sharemid{j,2}>5         %if the shared edge length is
larger than 5 meters, these two can be considered as connected, which to avoid the situation that
two polygon only share one point or a short edge.
                            if size(Share,1)>5    %if the amount of vertices on the edge is
enough (>5), the median of such vertices can represent a precise mid point
                                Airport(i,1).sharemid{j,1} = median(Share);
                            else       %if the amount of vertices on the edge is not enough (<=5),
polylineCentroid function is needed to find out the mid point.
                                Airport(i,1).sharemid{j,1} =
polylineCentroid(Share);       %polylineCentroid() is quite time consuming, which is the reason
use median() to find the mid point of edge with lots vertices on.
                            end
                        end
                    end
                end
            end
        end
    end
end
delete(h);
for i = 1:1:index;  %i is the index of element A
    if ~strcmp(Airport(i,1).element_feature_type,'aixm_ApronElement')
        for j = 1:1:index;  %j is the index of element B
            if i~=j&&~strcmp(Airport(j,1).element_feature_type,'aixm_ApronElement')
                if Airport(j,1).sharemid{i,2}>5     %if element a and B shared a edge longer than
5 meters
                    Airport(i,1).sharemid{j,1} = Airport(j,1).sharemid{i,1};    %then set the mid
point of edge shared by A and B same as the edge shared by B and A
                end
            end
        end
    end
```

```
end
end


function varargout = distancePointPolygon(point, poly)
%DISTANCEPOINTPOLYGON  Compute shortest distance between a point and a polygon
%   output = distancePointPolygon(POINT, POLYGON)
%
%   Example
%   distancePointPolygon
%
%   See also
%   polygons2d, points2d, distancePointPolyline
%   distancePointEdge, projPointOnPolyline
%
% ------
% Author: David Legland
% e-mail: david.legland@grignon.inra.fr
% Created: 2009-04-30,    using Matlab 7.7.0.471 (R2008b)
% Copyright 2009 INRA - Cepia Software Platform.

% eventually copy first point at the end to ensure closed polygon
if sum(poly(end, :)==poly(1,:))~=2
    poly = [poly; poly(1,:)];
end

% call to distancePointPolyline
minDist = distancePointPolyline(point, poly);

% process output arguments
if nargout<=1
    varargout{1} = minDist;
end

function center = polylineCentroid(varargin)
%POLYLINECENTROID Compute centroid of a curve defined by a series of points
%
%   PT = polylineCentroid(POINTS);
%   Computes center of mass of a polyline defined by POINTS. POINTS is a
%   [NxD] array of double, representing a set of N points in a
%   D-dimensional space.
%
%   PT = polylineCentroid(PTX, PTY);
%   PT = polylineCentroid(PTX, PTY, PTZ);
%   Specifies points as separate column vectors
%
%   PT = polylineCentroid(..., TYPE);
%   Specifies if the last point is connected to the first one. TYPE can be
%   either 'closed' or 'open'.
%
%   Example
%   poly = [0 0;10 0;10 10;20 10];
%   polylineCentroid(poly)
%   ans =
%       [10 5]
%
%   See also:
%   polygons2d, centroid, polygonCentroid, polylineLength
%
%   ---------
%   author : David Legland
%   INRA - TPV URPOI - BIA IMASTE
%   created the 22/05/2006.
%

%% process input arguments

% check whether the curve is closed
closed = false;
var = varargin{end};
if ischar(var)
```

```
    if strcmpi(var, 'closed')
        closed = true;
    end
    varargin = varargin(1:end-1);
end

% extract point coordinates
if length(varargin)==1
    points = varargin{1};
elseif length(varargin)==2
    points = [varargin{1} varargin{2}];
end

% compute centers and lengths composing the curve
if closed
    centers = (points + points([2:end 1],:))/2;
    lengths = sqrt(sum(diff(points([1:end 1],:)).^2, 2));
else
    centers = (points(1:end-1,:) + points(2:end,:))/2;
    lengths = sqrt(sum(diff(points).^2, 2));
end

% centroid of edge centers weighted by edge length
%weigths = repmat(lengths/sum(lengths), [1 size(points, 2)]);
center = sum(centers.*repmat(lengths, [1 size(points, 2)]), 1)/sum(lengths);

function varargout = distancePointPolyline(point, poly, varargin)
%DISTANCEPOINTPOLYLINE  Compute shortest distance between a point and a polyline
%   output = distancePointPolyline(POINT, POLYLINE)
%
%   Example:
%       pt1 = [30 20];
%       pt2 = [30 5];
%       poly = [10 10;50 10;50 50;10 50];
%       distancePointPolyline([pt1;pt2], poly)
%       ans =
%           10
%            5
%
%   See also
%   polygons2d, points2d
%   distancePointEdge, projPointOnPolyline
%
% ------
% Author: David Legland
% e-mail: david.legland@grignon.inra.fr
% Created: 2009-04-30,    using Matlab 7.7.0.471 (R2008b)
% Copyright 2009 INRA - Cepia Software Platform.

%   HISTORY
%   2009-06-23 compute all distances in one call

% number of points
Np = size(point, 1);

% allocate memory for result
minDist = inf * ones(Np, 1);

% process each point
for p = 1:Np
    % construct the set of edges
    edges = [poly(1:end-1, :) poly(2:end, :)];

    % compute distance between current each point and all edges
    dist = distancePointEdge(point(p, :), edges);

    % update distance if necessary
    minDist(p) = min(dist);
end

% process output arguments
```

```
if nargout<=1
    varargout{1} = minDist;
end

function varargout = distancePointEdge(point, edge)
%DISTANCEPOINTEDGE Minimum distance between a point and an edge
%
%   DIST = distancePointEdge(POINT, EDGE);
%   Return the euclidean distance between edge EDGE and point POINT.
%   EDGE has the form: [x1 y1 x2 y2], and POINT is [x y].
%
%   If EDGE is N-by-4 array, result is N-by-1 array computed for each edge.
%   If POINT is a N-by-2 array, the result is computed for each point.
%   If both POINT and EDGE are array, they must have the same number of
%   rows, and the result is computed for each couple point(i,:);edge(i,:).
%
%   [DIST POS] = distancePointEdge(POINT, EDGE);
%   Also returns the position of closest point on the edge. POS is
%   comprised between 0 (first point) and 1 (last point).
%
%   See also:
%   edges2d, points2d, distancePoints, distancePointLine
%
%
%   ---------
%   author : David Legland
%   INRA - CEPIA URPOI - MIA MathCell
%   created the 07/04/2004.
%

%   HISTORY
%   2005-06-24 rename, and change arguments sequence
%   2009-04-30 add possibility to return position of closest point
%   2011-04-14 add checkup for degenerate edges, improve speed, update doc

% direction vector of each edge
dx = edge(:, 3) - edge(:,1);
dy = edge(:, 4) - edge(:,2);

% compute position of points projected on the supporting line
% (Size of tp is the max number of edges or points)
delta = dx .* dx + dy .* dy;
tp = ((point(:, 1) - edge(:, 1)) .* dx + (point(:, 2) - edge(:, 2)) .* dy) ./ delta;

% ensure degenerated edges are correclty processed (consider the first
% vertex is the closest)
tp(delta < eps) = 0;

% change position to ensure projected point is located on the edge
tp(tp < 0) = 0;
tp(tp > 1) = 1;

% coordinates of projected point
p0 = [edge(:,1) + tp .* dx, edge(:,2) + tp .* dy];

% compute distance between point and its projection on the edge
dist = sqrt((point(:,1) - p0(:,1)) .^ 2 + (point(:,2) - p0(:,2)) .^ 2);

% process output arguments
varargout{1} = dist;
if nargout > 1
    varargout{2} = tp;
end
```

```
function [Airport] = GetCentroid(Airport)
%%
% Description:
%   Get centroid of taxiway element of airport
%   Measure the length of taxiway element and width of runway
%   Add .centroid field to airport structure file
%   Add .length field to airport structure file
% Input:
%   airport structure file processed by SharePoint()
% Output:
%   airport structure file with .centroid and .length field.
% External function required:
%   polygeom(), distancePointPolygon(), intersectLinePolygon(),
%   interLinexPoly(), polylineCentroid(), findCent()
% Yang Zhang - 06/20/2013 - tested under MATLAB v8.1
%%

% constants
%d2r = pi / 180;

index = length(Airport);        %total number of elements of the airport

%run a loop to find the centroid of all the elements
h = waitbar(0,'1','Name','Finalizing AIXM Data...');
for i = 1:1:index;
    waitbar(i/index,h,sprintf('Preparing Index %g out of %g',i,index))
    Airport( i , 1 ).centroid = cell( 1 , 1 );      %create a new field(cell) called centroid to
store the centroid coordinate
    Airport( i , 1 ).length = 0;        %create a new field called length to store the length of
the element
    Airport( i , 1 ).width = 0;         %create a new field called width to store the width of
the element
    poly = Airport(i, 1).extent{1, 1};      %store all the vertices on the edges of the element
to a polygon variable
    %if the element is a runway element, then store all the shared mids with
    %taxiway elements(runway exits) to the field centroid for future use.
    if strcmp(Airport(i,1).element_feature_type,'aixm_RunwayElement')
        interNumber = find(~cellfun(@isempty,Airport(i,1).sharemid(:,1)));      %find the total
number of shared mids with others
        interNumber1 = interNumber;
        a = length(interNumber);
        %run a loop to eliminate the shared mids with runway elements
        for j = 1:1:a;
            if ~strcmp(Airport(interNumber1(j),1).element_feature_type,'aixm_RunwayElement')
                Airport(interNumber1(j),1).element_sub_type{1,2} = char('RUNWAY_EXIT');
            end
            if strcmp(Airport(interNumber1(j),1).element_feature_type,'aixm_RunwayElement')
                interNumber = RemoveTarget(interNumber,interNumber1(j));
            end
        end
        %store the runway exits to centroid field
        Airport( i , 1 ).centroid =
{cat(2,reshape([Airport(i,1).sharemid{interNumber,1}],2,length([Airport(i,1).sharemid{interNumber
,1}])/2)',interNumber)};
        %find the centroid of the runway to measure the runway width
        [ geom, ~, ~ ] = polygeom( poly(:,1), poly(:,2) );
        cent = [ geom(2) geom(3)];
        Airport( i , 1 ).width = 2*distancePointPolygon(cent, Airport(i,1).extent{1,1});
    else
        %if the element is not a runway element find the centroid(s) of it
        x = poly(:,1);
        y = poly(:,2);
        [ geom, ~, cpmo ] = polygeom( x, y );
        angle2 = cpmo(4);       %angle of the symmetry axis
        x_cen = geom(2);
        y_cen = geom(3);        %geometry centroid
        Airport( i , 1 ).centroid = {[x_cen; y_cen]'};
        interNumber = find(~cellfun(@isempty,Airport(i,1).sharemid(:,1)));
        in = inpolygon(x_cen,y_cen,x,y);        %check if the geometry centroid is inside the
polygon
        while ~in    %if not, move out-of-polygon centroid into polygon temporarily
```

```
            interLinexPoly = intersectLinePolygon([x_cen y_cen cos(angle2)
sin(angle2)],poly);      %draw the symmetry axis crossing the geometry centroid and intersects
the polygon
            criticInter = interLinexPoly( 1:2,:);       %the intersection between symmtry axis
and polygon
            newCent = polylineCentroid(criticInter);        %mid point of the two intersections,
which is the new centroid
            x_cen = newCent(1,1);
            y_cen = newCent(1,2);
            in = inpolygon(x_cen,y_cen,x,y);        %check if it is inside the polygon
        end
        %if the taxiway element is normal type and has only two connected
        %elements, then run the sub function findCent() to find the
        %multiple centroids (if exists)
        if strcmp(Airport(i,1).element_feature_type,'aixm_TaxiwayElement')&&(strcmp(Airport(i,
1).element_sub_type{1,1},'NORMAL'))&&(length(interNumber)==2)
            %store the left and right hand side share mids as left and
            %right hand side points of this centroid
            x_p_l(1) = Airport(i,1).sharemid{interNumber(1,1),1}(1,1);      %x position of left-
hand side polygon
            y_p_l(1) = Airport(i,1).sharemid{interNumber(1,1),1}(1,2);      %y position of left-
hand side polygon
            x_p_r(1) = Airport(i,1).sharemid{interNumber(2,1),1}(1,1);      %x position of right-
hand side polygon
            y_p_r(1) = Airport(i,1).sharemid{interNumber(2,1),1}(1,2);      %y position of right-
hand side polygon


            [x_cen, y_cen, ele_length] = findCent(x_cen, y_cen, x_p_l, y_p_l, x_p_r, y_p_r, poly);

            Airport( i , 1 ).centroid = {[x_cen; y_cen]'};      %store the multi centroids
            Airport( i , 1 ).length = ele_length;
            Airport( i , 1 ).width = 2*distancePointPolygon(Airport( i , 1 ).centroid{1,1},
Airport(i,1).extent{1,1});
        end
    end
end
delete(h);
end



function [ C ] = RemoveTarget(Origin , Target)
%remove the Target from the Origin and keep the sort of Origin
C=[];
if ~isempty(Origin)&&~isempty(Target)
    indexa = size(Origin);
    indexb = size(Target);
    k =1;
    for i = 1:1:indexb(1,1);
        for j = 1:1:indexa(1,1);
            if Origin(j,:)~=Target(i,:)
                C(k,:) = Origin(j,:);
                k=k+1;
            end
        end
    end
else
    if isempty(Origin)
        C = [];
    else
        C = Origin;
    end
end
end

function  [x_cen, y_cen, ele_length] = findCent(x_cen, y_cen, x_p_l, y_p_l, x_p_r, y_p_r, poly)
%%find the multi centroids of a polygon and its length
%input:
%   x_cen, y_cen: the original inside polygon centroid coordinates
%   x_p_l, y_p_l: the coordinates of the sharemid on the left hand side of centroid
```

```
%   x_p_r, y_p_r: the coordinates of the sharemid on the right hand side of centroid
%output:
%   x_cen, y_cen: the list of the multi centroids coordinates
%   ele_length: the length of the element
%%
ele_length = 0;     %initiate the element length

done = 0;       %indicator of the multi centroid finding procedure

%indicator of the total number of multi centroids
i =1;
k =1;
while ~done
    %check if there is necessary to add multi centroid between centroid and
    %left hand side point
    INTER = RemoveTarget(intersectEdgePolygon([x_cen(i) y_cen(i) x_p_l(i) y_p_l(i)],
poly),[x_p_l(i) y_p_l(i)]);        %find the intersections between edge(centroid-lefthand side
point) and the polygon (remove the lefthand side point from the intersections)
    index = size(INTER);        %number of intersections
    %if the number is larger than 2, then move the mid point of the
    %intersections into the centerline of the polygon as part of the multi
    %centroids
    if index(1,1)>=2
        j = max(i,k)+1;     %add 1 to the total number of the multi centroid
        midcritic = polylineCentroid(INTER(1:2,:));     %find the mid point of the intersections
        x_mid = midcritic(1,1);
        y_mid = midcritic(1,2);

        %move the mid point onto the polygon centerline
        angle = angle2Points(midcritic, [x_p_l(i) y_p_l(i)])-0.5*pi;
        InterPolyxLine = intersectLinePolygon([x_mid y_mid cos(angle) sin(angle)],poly);
        critcInter  = InterPolyxLine( 1:2,: );
        centNew = polylineCentroid(critcInter);

        x_cen(j) = centNew(1,1);
        y_cen(j) = centNew(1,2);

        %store the new left and right hand side point of this centroid
        x_p_r(j) = x_cen(i);
        y_p_r(j) = y_cen(i);
        x_p_l(j) = x_p_l(i);
        y_p_l(j) = y_p_l(i);
        k = k+1;
    else
        ele_length = ele_length + distancePoints( [x_cen(i) y_cen(i)], [x_p_l(i) y_p_l(i)]);
    end

    %run a similar procedure to check the right hand side point
    INTER = RemoveTarget(intersectEdgePolygon([x_cen(i) y_cen(i) x_p_r(i) y_p_r(i)],
poly),[x_p_r(i) y_p_r(i)]);
    index = size(INTER);
    if index(1,1)>=2
        j = max(i,k)+1;
        midcritic = polylineCentroid(INTER(1:2,:));
        x_mid = midcritic(1,1);
        y_mid = midcritic(1,2);

        angle = angle2Points(midcritic, [x_p_r(i) y_p_r(i)])-0.5*pi;
        InterPolyxLine = intersectLinePolygon([x_mid y_mid cos(angle) sin(angle)],poly);
        critcInter  = InterPolyxLine( 1:2,: );
        centNew = polylineCentroid(critcInter);

        x_cen(j) = centNew(1,1);
        y_cen(j) = centNew(1,2);

        x_p_l(j) = x_cen(i);
        y_p_l(j) = y_cen(i);
        x_p_r(j) = x_p_r(i);
        y_p_r(j) = y_p_r(i);

        k = k+1;
```

```
    else
        ele_length = ele_length + distancePoints( [x_cen(i) y_cen(i)], [x_p_r(i) y_p_r(i)]);
    end

    if i == k
        done = 1;
    end
    i =i+1;
end
end

function [ geom, iner, cpmo ] = polygeom( x, y )
%POLYGEOM Geometry of a planar polygon
%
%   POLYGEOM( X, Y ) returns area, X centroid,
%   Y centroid and perimeter for the planar polygon
%   specified by vertices in vectors X and Y.
%
%   [ GEOM, INER, CPMO ] = POLYGEOM( X, Y ) returns
%   area, centroid, perimeter and area moments of
%   inertia for the polygon.
%   GEOM = [ area   X_cen  Y_cen  perimeter ]
%   INER = [ Ixx    Iyy    Ixy    Iuu    Ivv    Iuv ]
%     u,v are centroidal axes parallel to x,y axes.
%   CPMO = [ I1     ang1   I2     ang2   J ]
%     I1,I2 are centroidal principal moments about axes
%         at angles ang1,ang2.
%     ang1 and ang2 are in radians.
%     J is centroidal polar moment.  J = I1 + I2 = Iuu + Ivv

% H.J. Sommer III - 02.05.14 - tested under MATLAB v5.2
%
% sample data
% x = [ 2.000  0.500  4.830  6.330 ]';
% y = [ 4.000  6.598  9.098  6.500 ]';
% 3x5 test rectangle with long axis at 30 degrees
% area=15, x_cen=3.415, y_cen=6.549, perimeter=16
% Ixx=659.561, Iyy=201.173, Ixy=344.117
% Iuu=16.249, Ivv=26.247, Iuv=8.660
% I1=11.249, ang1=30deg, I2=31.247, ang2=120deg, J=42.496
%
% H.J. Sommer III, Ph.D., Professor of Mechanical Engineering, 337 Leonhard Bldg
% The Pennsylvania State University, University Park, PA  16802
% (814)863-8997  FAX (814)865-9693  hjs1@psu.edu  www.me.psu.edu/sommer/

% begin function POLYGEOM

% check if inputs are same size
if ~isequal( size(x), size(y) ),
  error( 'X and Y must be the same size');
end

% number of vertices
[ x, ns ] = shiftdim( x );
[ y, ns ] = shiftdim( y );
[ n, c ] = size( x );

% temporarily shift data to mean of vertices for improved accuracy
xm = mean(x);
ym = mean(y);
x = x - xm*ones(n,1);
y = y - ym*ones(n,1);

% delta x and delta y
dx = x( [ 2:n 1 ] ) - x;
dy = y( [ 2:n 1 ] ) - y;

% summations for CW boundary integrals
A = sum( y.*dx - x.*dy )/2;
Axc = sum( 6*x.*y.*dx -3*x.*x.*dy +3*y.*dx.*dx +dx.*dx.*dy )/12;
Ayc = sum( 3*y.*y.*dx -6*x.*y.*dy -3*x.*dy.*dy -dx.*dy.*dy )/12;
```

```
Ixx = sum( 2*y.*y.*y.*dx -6*x.*y.*y.*dy -6*x.*y.*dy.*dy ...
          -2*x.*dy.*dy.*dy -2*y.*dx.*dy.*dy -dx.*dy.*dy.*dy )/12;
Iyy = sum( 6*x.*x.*y.*dx -2*x.*x.*x.*dy +6*x.*y.*dx.*dx ...
          +2*y.*dx.*dx.*dx +2*x.*dx.*dx.*dy +dx.*dx.*dx.*dy )/12;
Ixy = sum( 6*x.*y.*y.*dx -6*x.*x.*y.*dy +3*y.*y.*dx.*dx ...
          -3*x.*x.*dy.*dy +2*y.*dx.*dx.*dy -2*x.*dx.*dy.*dy )/24;
P = sum( sqrt( dx.*dx +dy.*dy ) );

% check for CCW versus CW boundary
if A < 0,
  A = -A;
  Axc = -Axc;
  Ayc = -Ayc;
  Ixx = -Ixx;
  Iyy = -Iyy;
  Ixy = -Ixy;
end

% centroidal moments
xc = Axc / A;
yc = Ayc / A;
Iuu = Ixx - A*yc*yc;
Ivv = Iyy - A*xc*xc;
Iuv = Ixy - A*xc*yc;
J = Iuu + Ivv;

% replace mean of vertices
x_cen = xc + xm;
y_cen = yc + ym;
Ixx = Iuu + A*y_cen*y_cen;
Iyy = Ivv + A*x_cen*x_cen;
Ixy = Iuv + A*x_cen*y_cen;

% principal moments and orientation
I = [ Iuu  -Iuv ;
     -Iuv   Ivv ];
[ eig_vec, eig_val ] = eig(I);
I1 = eig_val(1,1);
I2 = eig_val(2,2);
ang1 = atan2( eig_vec(2,1), eig_vec(1,1) );
ang2 = atan2( eig_vec(2,2), eig_vec(1,2) );

% return values
geom = [ A  x_cen  y_cen  P ];
iner = [ Ixx  Iyy  Ixy  Iuu  Ivv  Iuv ];
cpmo = [ I1  ang1  I2  ang2  J ];

% end of function POLYGEOM

function [intersects edgeIndices] = intersectLinePolygon(line, poly, varargin)
%INTERSECTLINEPOLYGON Intersection points between a line and a polygon
%
%   P = intersectLinePolygon(LINE, POLY)
%   Returns the intersection points of the lines LINE with polygon POLY.
%   LINE is a 1-by-4 row vector containing parametric representation of the
%   line (in the format [x0 y0 dx dy], see the function 'createLine' for
%   details).
%   POLY is a NV-by-2 array containing coordinates of the polygon vertices
%   P is a K-by-2 array containing the coordinates of the K intersection
%   points.
%
%   P = intersectLinePolygon(LINE, POLY, TOL)
%   Specifies the tolerance for geometric tests. Default is 1e-14.
%
%   [P INDS] = intersectLinePolygon(...)
%   Also returns the indices of edges involved in intersections. INDS is a
%   K-by-1 column vector, such that P(i,:) corresponds to intersection of
%   the line with the i-th edge of the polygon. If the intersection occurs
%   at a polygon vertex, the index of only one of the two neighbor edges is
%   returned.
%   Note that due to numerical approximations, the use of function
```

```
%   'isPointOnEdge' may give results not consistent with this function.
%
%
%   Examples
%   % compute intersections between a square and an horizontal line
%     poly = [0 0;10 0;10 10;0 10];
%     line = [5 5 1 0];
%     intersectLinePolygon(line, poly)
%     ans =
%           10     5
%            0     5
%     % also return indices of edges
%     [inters inds] = intersectLinePolygon(line, poly)
%     inters =
%           10     5
%            0     5
%     inds =
%           4
%           2
%
%   % compute intersections between a square and a diagonal line
%     poly = [0 0;10 0;10 10;0 10];
%     line = [5 5 1 1];
%     intersectLinePolygon(line, poly)
%     ans =
%            0     0
%           10    10
%
%   See Also
%   lines2d, polygons2d, intersectLines, intersectRayPolygon
%
%   ---------
%   author : David Legland
%   INRA - TPV URPOI - BIA IMASTE
%   created the 31/10/2003.
%

%   HISTORY
%   2008-11-24 rename 'pi' as 'intersects', update doc
%   2009-07-23 removed forgotten occurence of 'pi' variable (thanks to Bala
%       Krishnamoorthy)
%   2010-01-26 rewrite using vectorisation
%   2011-05-20 returns unique results
%   2011-07-20 returns intersected edge indices

% get computation tolerance
tol = 1e-14;
if ~isempty(varargin)
    tol = varargin{1};
end

% create the array of edges
N = size(poly, 1);
edges = [poly(1:N, :) poly([2:N 1], :)];

% compute intersections with supporting lines of polygon edges
supportLines = edgeToLine(edges);
intersects = intersectLines(line, supportLines, tol);

% find edges that are not parallel to the input line
inds = find(isfinite(intersects(:, 1)));

% compute position of intersection points on corresponding lines
pos = linePosition(intersects(inds, :), supportLines(inds, :));

% and keep only intersection points located on edges
b = pos > -tol & pos < 1+tol;
inds = inds(b);
intersects = intersects(inds, :);

% remove multiple vertices (can occur for intersections located at polygon
```

```
% vertices)
[intersects I J] = unique(intersects, 'rows'); %#ok<NASGU>

if nargout > 1
    % return indices of edges involved in intersection
    % (in case of intersection located at a vertex, only one of the
    % neighbor edges is returned)
    edgeIndices = inds(I);
end

function line = edgeToLine(edge)
%EDGETOLINE Convert an edge to a straight line
%
%   LINE = edgeToLine(EDGE);
%   Returns the line containing the edge EDGE.
%
%   Example
%       edge = [2 3 4 5];
%       line = edgeToLine(edge);
%       figure(1); hold on; axis([0 10 0 10]);
%       drawLine(line, 'color', 'g')
%       drawEdge(edge, 'linewidth', 2)
%
%   See also
%   edges2d, lines2d
%
% ------
% Author: David Legland
% e-mail: david.legland@grignon.inra.fr
% Created: 2009-07-23,    using Matlab 7.7.0.471 (R2008b)
% Copyright 2009 INRA - Cepia Software Platform.

line = [edge(:, 1:2) edge(:, 3:4)-edge(:, 1:2)];

function point = intersectLines(line1, line2, varargin)
%INTERSECTLINES Return all intersection points of N lines in 2D
%
%   PT = intersectLines(L1, L2);
%   returns the intersection point of lines L1 and L2. L1 and L2 are 1-by-4
%   row arrays, containing parametric representation of each line (in the
%   form [x0 y0 dx dy], see 'createLine' for details).
%
%   In case of colinear lines, returns [Inf Inf].
%   In case of parallel but not colinear lines, returns [NaN NaN].
%
%   If each input is [N*4] array, the result is a [N*2] array containing
%   intersections of each couple of lines.
%   If one of the input has N rows and the other 1 row, the result is a
%   [N*2] array.
%
%   PT = intersectLines(L1, L2, EPS);
%   Specifies the tolerance for detecting parallel lines. Default is 1e-14.
%
%   Example
%   line1 = createLine([0 0], [10 10]);
%   line2 = createLine([0 10], [10 0]);
%   point = intersectLines(line1, line2)
%   point =
%       5   5
%
%   See also
%   lines2d, edges2d, intersectEdges, intersectLineEdge
%   intersectLineCircle
%
%   ---------
%   author : David Legland
%   INRA - TPV URPOI - BIA IMASTE
%   created the 31/10/2003.
%

%   HISTORY
```

```
%   2004-02-19 add support for multiple lines.
%   2007-03-08 update doc
%   2011-10-07 code cleanup


%% Process input arguments

% extract tolerance
tol = 1e-14;
if ~isempty(varargin)
    tol = varargin{1};
end

% check size of each input
N1 = size(line1, 1);
N2 = size(line2, 1);
N = max(N1, N2);
if N1 ~= N2 && N1*N2 ~= N
    error('matGeom:IntersectLines:IllegalArgument', ...
        'The two input arguments must have same number of lines');
end


%% Check parallel and colinear lines

% coordinate differences of origin points
dx = bsxfun(@minus, line2(:,1), line1(:,1));
dy = bsxfun(@minus, line2(:,2), line1(:,2));

% indices of parallel lines
denom = line1(:,3) .* line2(:,4) - line2(:,3) .* line1(:,4);
par = abs(denom) < tol;

% indices of colinear lines
col = abs(dx .* line1(:,4) - dy .* line1(:,3)) < tol & par ;

% initialize result array
x0 = zeros(N, 1);
y0 = zeros(N, 1);

% initialize result for parallel lines
x0(col) = Inf;
y0(col) = Inf;
x0(par & ~col) = NaN;
y0(par & ~col) = NaN;

% in case all line couples are parallel, return
if all(par)
    point = [x0 y0];
    return;
end


%% Extract coordinates of itnersecting lines

% indices of intersecting lines
inds = ~par;

% extract base coordinates of first lines
if N1 > 1
    line1 = line1(inds,:);
end
x1 =  line1(:,1);
y1 =  line1(:,2);
dx1 = line1(:,3);
dy1 = line1(:,4);

% extract base coordinates of second lines
if N2 > 1
    line2 = line2(inds,:);
end
```

```
x2 =  line2(:,1);
y2 =  line2(:,2);
dx2 = line2(:,3);
dy2 = line2(:,4);

% re-compute coordinate differences of origin points
dx = bsxfun(@minus, line2(:,1), line1(:,1));
dy = bsxfun(@minus, line2(:,2), line1(:,2));


%% Compute intersection points

denom = denom(inds);
x0(inds) = (x2 .* dy2 .* dx1 - dy .* dx1 .* dx2 - x1 .* dy1 .* dx2) ./ denom ;
y0(inds) = (dx .* dy1 .* dy2 + y1 .* dx1 .* dy2 - y2 .* dx2 .* dy1) ./ denom ;

% concatenate result
point = [x0 y0];
function d = linePosition(point, line)
%LINEPOSITION Position of a point on a line
%
%   POS = linePosition(POINT, LINE);
%   Computes position of point POINT on the line LINE, relative to origin
%   point and direction vector of the line.
%   LINE has the form [x0 y0 dx dy],
%   POINT has the form [x y], and is assumed to belong to line.
%
%   POS = linePosition(POINT, LINES);
%   If LINES is an array of NL lines, return NL positions, corresponding to
%   each line.
%
%   POS = linePosition(POINTS, LINE);
%   If POINTS is an array of NP points, return NP positions, corresponding
%   to each point.
%
%   POS = linePosition(POINTS, LINES);
%   If POINTS is an array of NP points and LINES is an array of NL lines,
%   return an array of [NP NL] position, corresponding to each couple
%   point-line.
%
%   Example
%   line = createLine([10 30], [30 90]);
%   linePosition([20 60], line)
%   ans =
%       .5
%
%   See also:
%   lines2d, createLine, projPointOnLine, isPointOnLine
%
%   ---------
%
%   author : David Legland
%   INRA - TPV URPOI - BIA IMASTE
%   created the 25/05/2004.
%

%   HISTORY
%   2005-07-07 manage multiple input
%   2011-06-15 avoid the use of repmat when possible

% number of inputs
Nl = size(line, 1);
Np = size(point, 1);

if Np == Nl
    % if both inputs have the same size, no problem
    dxl = line(:, 3);
    dyl = line(:, 4);
    dxp = point(:, 1) - line(:, 1);
    dyp = point(:, 2) - line(:, 2);
```

```
elseif Np == 1
    % one point, several lines
    dxl = line(:, 3);
    dyl = line(:, 4);
    dxp = point(ones(Nl, 1), 1) - line(:, 1);
    dyp = point(ones(Nl, 1), 2) - line(:, 2);

elseif Nl == 1
    % one line, several points
    dxl = line(ones(Np, 1), 3);
    dyl = line(ones(Np, 1), 4);
    dxp = point(:, 1) - line(1);
    dyp = point(:, 2) - line(2);

else
    % expand one of the array to have the same size
    dxl = repmat(line(:,3)', Np, 1);
    dyl = repmat(line(:,4)', Np, 1);
    dxp = repmat(point(:,1), 1, Nl) - repmat(line(:,1)', Np, 1);
    dyp = repmat(point(:,2), 1, Nl) - repmat(line(:,2)', Np, 1);
end

% compute position
d = (dxp.*dxl + dyp.*dyl) ./ (dxl.^2 + dyl.^2);

function  [x_cen,y_cen,length] = findCent(x_cen, y_cen, x_p_l, y_p_l, x_p_r, y_p_r, poly)
%%
% Descroption:
%   Find multiple centroids for irregular shape polygon(i), which the polygon is a taxiway link.
% Input:
%   x_cen: x location of temporarily inpolygon centroid
%   y_cen: y location of temporarily inpolygon centroid
%   x_p_l: x location of left-hand-side polygon sharemid point with
%   polygon(i)
%   y_p_l: y location of left-hand-side polygon sharemid point with
%   polygon(i)
%   x_p_r: x location of right-hand-side polygon sharemid point with
%   polygon(i)
%   y_p_r: y location of right-hand-side polygon sharemid point with
%   polygon(i)
%   poly: vertices on polygon(i)
% Output:
%   x_cen: x location of multiple centroids of polygon(i)
%   y_cen: y location of multiple centroids of polygon(i)
%   length: length of polygon(i)
% External function required:
%   intersectEdgePolygon(), polylineCentroid(), angle2Points(),
%   intersectLinePolygon(), distancePoints()
% Yang Zhang - 06/20/2013 - tested under MATLAB v8.1
length = 0;

done = 0;

i =1;
k =1;
while ~done
    INTER = RemoveTarget(intersectEdgePolygon([x_cen(i) y_cen(i) x_p_l(i) y_p_l(i)],
poly),[x_p_l(i) y_p_l(i)]);
    index = size(INTER);
    if index(1,1)>=2
        j = max(i,k)+1;
        midcritic = polylineCentroid(INTER(1:2,:));
        x_mid = midcritic(1,1);
        y_mid = midcritic(1,2);

        angle = angle2Points(midcritic, [x_p_l(i) y_p_l(i)])-0.5*pi;
        InterPolyxLine = intersectLinePolygon([x_mid y_mid cos(angle) sin(angle)],poly);
        critcInter  = InterPolyxLine( 1:2,: );
        centNew = polylineCentroid(critcInter);

        x_cen(j) = centNew(1,1);
```

```
        y_cen(j) = centNew(1,2);

        x_p_r(j) = x_cen(i);
        y_p_r(j) = y_cen(i);
        x_p_l(j) = x_p_l(i);
        y_p_l(j) = y_p_l(i);
        k = k+1;
    else
        length = length + distancePoints( [x_cen(i) y_cen(i)], [x_p_l(i) y_p_l(i)]);
    end

    INTER = RemoveTarget(intersectEdgePolygon([x_cen(i) y_cen(i) x_p_r(i) y_p_r(i)],
poly),[x_p_r(i) y_p_r(i)]);
    index = size(INTER);
    if index(1,1)>=2
        j = max(i,k)+1;
        midcritic = polylineCentroid(INTER(1:2,:));
        x_mid = midcritic(1,1);
        y_mid = midcritic(1,2);

        angle = angle2Points(midcritic, [x_p_r(i) y_p_r(i)])-0.5*pi;
        InterPolyxLine = intersectLinePolygon([x_mid y_mid cos(angle) sin(angle)],poly);
        critcInter  = InterPolyxLine( 1:2,: );
        centNew = polylineCentroid(critcInter);

        x_cen(j) = centNew(1,1);
        y_cen(j) = centNew(1,2);

        x_p_l(j) = x_cen(i);
        y_p_l(j) = y_cen(i);
        x_p_r(j) = x_p_r(i);
        y_p_r(j) = y_p_r(i);

        k = k+1;
    else
        length = length + distancePoints( [x_cen(i) y_cen(i)], [x_p_r(i) y_p_r(i)]);
    end

    if i == k
        done = 1;
    end
    i =i+1;
end
end

function theta = angle2Points(varargin)
%ANGLE2POINTS Compute horizontal angle between 2 points
%
%   ALPHA = angle2Points(P1, P2),
%   Pi are either [1*2] arrays, or [N*2] arrays, in this case ALPHA is a
%   [N*1] array. The angle computed is the horizontal angle of the line
%   (P1 P2)
%   Result is always given in radians, between 0 and 2*pi.
%
%   See Also:
%   points2d, angles2d, angle3points, normalizeAngle, vectorAngle
%
%
% ---------
% Author: David Legland
% e-mail: david.legland@grignon.inra.fr
% created the 02/03/2007.
% Copyright 2010 INRA - Cepia Software Platform.

%   HISTORY:
%   2011-01-11 use bsxfun

% process input arguments
if length(varargin)==2
    p1 = varargin{1};
    p2 = varargin{2};
```

```
elseif length(varargin)==1
    var = varargin{1};
    p1 = var(1,:);
    p2 = var(2,:);
end

% ensure data have correct size
n1 = size(p1, 1);
n2 = size(p2, 1);
if n1~=n2 && min(n1, n2)>1
    error('angle2Points: wrong size for inputs');
end

% angle of line (P2 P1), between 0 and 2*pi.
dp = bsxfun(@minus, p2, p1);
theta = mod(atan2(dp(:,2), dp(:,1)) + 2*pi, 2*pi);


function dist = distancePoints(p1, p2, varargin)
%DISTANCEPOINTS Compute distance between two points
%
%   D = distancePoints(P1, P2)
%   Return the Euclidean distance between points P1 and P2.
%
%   If P1 and P2 are two arrays of points, result is a N1*N2 array
%   containing distance between each point of P1 and each point of P2.
%
%   D = distancePoints(P1, P2, NORM)
%   Compute distance using the specified norm. NORM=2 corresponds to usual
%   euclidean distance, NORM=1 corresponds to Manhattan distance, NORM=inf
%   is assumed to correspond to maximum difference in coordinate. Other
%   values (>0) can be specified.
%
%   D = distancePoints(..., 'diag')
%   compute only distances between P1(i,:) and P2(i,:).
%
%   See also:
%   points2d, minDistancePoints, nndist
%
%
%   ---------
%
%   author : David Legland
%   INRA - TPV URPOI - BIA IMASTE
%   created the 24/02/2004.
%

%   HISTORY :
%   25/05/2004: manage 2 array of points
%   07/04/2004: add option for computing only diagonal.
%   30/10/2006: generalize to any dimension, and manage different norms
%   03/01/2007: bug for arbitrary norm, and update doc
%   28/08/2007: fix bug for norms 2 and infinite, in diagonal case


%% Setup options

% default values
diag = false;
norm = 2;

% check first argument: norm or diag
if ~isempty(varargin)
    var = varargin{1};
    if isnumeric(var)
        norm = var;
    elseif strncmp('diag', var, 4)
        diag = true;
    end
    varargin(1) = [];
end
```

```
% check last argument: diag
if ~isempty(varargin)
    var = varargin{1};
    if strncmp('diag', var, 4)
        diag = true;
    end
end


% number of points in each array and their dimension
n1  = size(p1, 1);
n2  = size(p2, 1);
d   = size(p1, 2);

if diag
    % compute distance only for apparied couples of pixels
    dist = zeros(n1, 1);
    if norm==2
        % Compute euclidian distance. this is the default case
        % Compute difference of coordinate for each pair of point
        % and for each dimension. -> dist is a [n1*n2] array.
        for i=1:d
            dist = dist + (p2(:,i)-p1(:,i)).^2;
        end
        dist = sqrt(dist);
    elseif norm==inf
        % infinite norm corresponds to maximal difference of coordinate
        for i=1:d
            dist = max(dist, abs(p2(:,i)-p1(:,i)));
        end
    else
        % compute distance using the specified norm.
        for i=1:d
            dist = dist + power((abs(p2(:,i)-p1(:,i))), norm);
        end
        dist = power(dist, 1/norm);
    end
else
    % compute distance for all couples of pixels
    dist = zeros(n1, n2);
    if norm==2
        % Compute euclidian distance. this is the default case
        % Compute difference of coordinate for each pair of point
        % and for each dimension. -> dist is a [n1*n2] array.
        for i=1:d
            % equivalent to:
            % dist = dist + ...
            %   (repmat(p1(:,i), [1 n2])-repmat(p2(:,i)', [n1 1])).^2;
            dist = dist + (p1(:, i*ones(1, n2))-p2(:, i*ones(n1, 1))').^2;
        end
        dist = sqrt(dist);
    elseif norm==inf
        % infinite norm corresponds to maximal difference of coordinate
        for i=1:d
            dist = max(dist, abs(p1(:, i*ones(1, n2))-p2(:, i*ones(n1, 1))'));
        end
    else
        % compute distance using the specified norm.
        for i=1:d
            % equivalent to:
            % dist = dist + power((abs(repmat(p1(:,i), [1 n2]) - ...
            %     repmat(p2(:,i)', [n1 1]))), norm);
            dist = dist + power((abs(p1(:, i*ones(1, n2))-p2(:, i*ones(n1, 1))')), norm);
        end
        dist = power(dist, 1/norm);
    end
end
```

```
function [] = plotNet(varargin)
if nargin == 2
    Airport = varargin{1};
    Gate = varargin{2};
elseif nargin == 1
    Airport = varargin{1};
    Gate =[];
end

PlotAirport(Airport);
index = size(Airport);
for i =1:1:index(1,1);
    if strcmp(Airport(i,1).element_feature_type,'aixm_TaxiwayElement')
        h1 = plot(Airport(i, 1).centroid{1, 1}(:,1),Airport(i, 1).centroid{1,
1}(:,2),'or','markers',5);
    end
end
hold on;
index = size(Airport);
for i =1:1:index(1,1);
    if strcmp(Airport(i,1).element_feature_type,'aixm_TaxiwayElement')
        for j =1:1:index(1,1);
            if ~isempty(Airport(i,1).sharemid{j,1})
                hold on
                h2 =
plot(Airport(i,1).sharemid{j,1}(1,1),Airport(i,1).sharemid{j,1}(1,2),'o','markers',5);
            end
        end
    end
end
xlabel('Longitude/Easting (Degree/Meters)','fontsize',18);
ylabel('Latitude/Northing (Degree/Meters)','fontsize',18);
set(gca,'FontSize',12)
title(strcat(Airport(1,1).IATA{1, 1},' Node-Link Map'),'fontsize',24);
hold on;
if ~isempty(Gate)
    h3 = plot(Gate.Position(:,1),Gate.Position(:,2),'*','markers',5);
    legend([h1,h2,h3],'taxiway centroid','midpoint of taxiway shared edge','gate position');
else
    legend([h1,h2],'taxiway centroid','midpoint of taxiway shared edge');
end

function [] = PlotAirport(Airport)
for i = 1:1:length(Airport);
    if strcmp(Airport(i,1).element_feature_type,'aixm_TaxiwayElement')
        fill(Airport(i, 1).extent{1,1}(:,1),Airport(i,1).extent{1,1}(:,2),'w');
    end;
    if strcmp(Airport(i,1).element_feature_type,'aixm_RunwayElement');
        fill(Airport(i, 1).extent{1,1}(:,1),Airport(i,1).extent{1,1}(:,2),'y');
    end;
    if strcmp(Airport(i,1).element_feature_type,'aixm_ApronElement');
        fill(Airport(i, 1).extent{1,1}(:,1),Airport(i,1).extent{1,1}(:,2),'r');
    end;
    hold on;
end
```

```
function [fileName,runwayList] = CreateTaxiways(varargin)
%%
% Description:
%   write taxiway and gate position information into ADSIM+ input file
%   taxiways.xml, where taxiway position info from Airport structure file
%   is required and gate position info from Gate structure file is optional
% Input:
%   Airport (required): airport structure file processed by WGS2LCS(),
%   SharePoint(),
%            GetCentroid()
%   Gate (optional): gate structure file processed by gateinfo2struct()
%   Airport.csv: Airport database from ADSIM+ built-in database.
%   Runway.csv: Runway database from ADSIM+ built-in database.
% Output:
%   taxiways.xml: ADSIM+ required input file.
%   fileName: taxiways.xml file directory (will be used as input for following function).
%   runwayList: cell file containing all the runway info, whose col 1: runway
%   label, col 2: runway end name, col 3: runway end location info, col 4:
%   runway elevation info, col 5: runway slope, col 6: runway width.
% Subfunctions:
%   fDrawMulti(), importRunway(), importAirport(), pointProjectEdge()
% Yang Zhang - 01/24/2014 - tested under MATLAB v8.1
%% check input
narginchk(1, 2)
if nargout < 3
    %[] = CreateTaxiways(Airport,ADSIM_dir)
    Airport = varargin{1};
    ADSIM_dir = varargin{2};
else
    %[] = WGS2LCS(Airport,Gate,ADSIM_dir)
    Airport = varargin{1};
    ADSIM_dir = varargin{3};
    Gate = varargin{2};
end

%%
fileName = [ADSIM_dir,'\taxiways.xml'];     %set filename of taxiways.xml input file
Output_File_ID = fopen(fileName, 'w');      %create the file or overwrite the taxiways.xml
Output_File_ID = fopen(fileName, 'a+');


%% --------------------------------
% Create Header of XML File --------------------------------
fprintf(Output_File_ID, '%s\n', '<!DOCTYPE ADSIMpInput>');
fprintf(Output_File_ID, '%s\n', '<ADSIMpInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ADSIMpInput.xsd">');
%%
% create taxiways network
n = 1;      % link number
s = length(Airport);        %total number of airport elements
h = waitbar(0,'Loading','Name','Creating Taxiway Network...');      %create wait bar for the
procedure
for i = 1:1:s
    %the taxiway link is defined by a f(from) node and a t(to) node
    waitbar(i/s,h,sprintf('Preparing Index %g out of %g',i,s));
    f = i;      %set element index = i element as the 'from' element
    if strcmp(Airport(f,1).element_feature_type,'aixm_TaxiwayElement')
        fNumOfCen = size(Airport(f, 1).centroid{1, 1});     %number of centroids of the 'from'
element
        if fNumOfCen(1,1) >1        %if more than one centroid, call fDrawMulti() function to
create inside element links between centroids
            n = fDrawMulti(Airport,f,n,fileName);
        end
        neigh = ~cellfun(@isempty,Airport(f, 1).sharemid(:,1));     %find the index of
neighbouring element to 'from' element
        ind = find(neigh~=0);       %ind store the index of neighbouring element
        %run a loop to draw links between 'from' element and 'to' elements
        for j = 1:1:length(ind)
            if ind(j)>f     %check if the 'to' element index is larger than the 'from' element to
avoid to create links repeatedly
                t = ind(j);     %t is the index of 'to' element
```

```
                if
strcmp(Airport(t,1).element_feature_type,'aixm_TaxiwayElement')&&strcmp(Airport(f,1).element_feat
ure_type,'aixm_TaxiwayElement')     %check and only create links between taxiway elements in this
block
                    tNumOfCen = size(Airport(t, 1).centroid{1, 1});     %number of centroids of
the 'to' element
                    if (fNumOfCen(1,1) == 1)&&(tNumOfCen(1,1) == 1)     %if both f and t element
only have one centroid, than use the two centroids to draw link
                        fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
                        fprintf(Output_File_ID, '%s\n', ['      <Airport>',Airport(1,1).ICAO{1,1},
'</Airport>']);
                        fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n),
'</Name>']);
                        fprintf(Output_File_ID, '%s\n', '
<AircraftCategory></AircraftCategory>');
                        fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
                        fprintf(Output_File_ID, '%s\n', '
<DistributionType>number</DistributionType>');
                        fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
                        fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
                        %check if either of it is possible dummy gate, name
                        %the node start with 'G', else name the node start
                        %with 'T'.
                        if strcmp(Airport(f,1).element_sub_type{1,1},'OTHER:APN_INTERSECTION')
                            fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="G', num2str(f), '" Position="',
num2str(Airport(f,1).centroid{1,1}(1,1)), ' ', num2str(Airport(f,1).centroid{1,1}(1,2)), '"/>']);
                        else
                            fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="T', num2str(f), '" Position="',
num2str(Airport(f,1).centroid{1,1}(1,1)), ' ', num2str(Airport(f,1).centroid{1,1}(1,2)), '"/>']);
                        end
                        if strcmp(Airport(t,1).element_sub_type{1,1},'OTHER:APN_INTERSECTION')
                            fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="G', num2str(t), '" Position="',
num2str(Airport(t,1).centroid{1,1}(1,1)), ' ', num2str(Airport(t,1).centroid{1,1}(1,2)), '"/>']);
                        else
                            fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="T', num2str(t), '" Position="',
num2str(Airport(t,1).centroid{1,1}(1,1)), ' ', num2str(Airport(t,1).centroid{1,1}(1,2)), '"/>']);
                        end
                        fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
                        n =n+1;
                    else
                        if (fNumOfCen(1,1) > 1)&&(tNumOfCen(1,1) == 1)  %if f element has more
than one centroid, than the link is defined by the sharemid of f/t and t centroid
                            fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
                            fprintf(Output_File_ID, '%s\n', ['
<Airport>',Airport(1,1).ICAO{1,1}, '</Airport>']);
                            fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n),
'</Name>']);
                            fprintf(Output_File_ID, '%s\n', '
<AircraftCategory></AircraftCategory>');
                            fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
                            fprintf(Output_File_ID, '%s\n', '
<DistributionType>number</DistributionType>');
                            fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
                            fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
                            fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="T', num2str(f),'-T',num2str(t), '" Position="',
num2str(Airport(f,1).sharemid{t,1}(1,1)), ' ', num2str(Airport(f,1).sharemid{t,1}(1,2)), '"/>']);
                            if  strcmp(Airport(t,1).element_sub_type{1,1},'OTHER:APN_INTERSECTION')
                                fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="G', num2str(t), '" Position="',
num2str(Airport(t,1).centroid{1,1}(1,1)), ' ', num2str(Airport(t,1).centroid{1,1}(1,2)), '"/>']);
                            else
                                fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="T', num2str(t), '" Position="',
num2str(Airport(t,1).centroid{1,1}(1,1)), ' ', num2str(Airport(t,1).centroid{1,1}(1,2)), '"/>']);
```

```
                            end
                            fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
                            n =n+1;
                        else
                            if(fNumOfCen(1,1) == 1)&&(tNumOfCen(1,1)> 1)        %if t element has
more than one centroid, than the link is defined by the sharemid of f/t and f centroid
                                fprintf(Output_File_ID, '%s\n', '   <TaxiWay
IncludeInStudy="true">');
                                fprintf(Output_File_ID, '%s\n', ['
<Airport>',Airport(1,1).ICAO{1,1}, '</Airport>']);
                                fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n),
'</Name>']);
                                fprintf(Output_File_ID, '%s\n', '
<AircraftCategory></AircraftCategory>');
                                fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
                                fprintf(Output_File_ID, '%s\n', '
<DistributionType>number</DistributionType>');
                                fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
                                fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
                                if
strcmp(Airport(f,1).element_sub_type{1,1},'OTHER:APN_INTERSECTION')
                                    fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="G', num2str(f), '" Position="',
num2str(Airport(f,1).centroid{1,1}(1,1)), ' ', num2str(Airport(f,1).centroid{1,1}(1,2)), '"/>']);
                                else
                                    fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="T', num2str(f), '" Position="',
num2str(Airport(f,1).centroid{1,1}(1,1)), ' ', num2str(Airport(f,1).centroid{1,1}(1,2)), '"/>']);
                                end
                                fprintf(Output_File_ID, '%s\n', ['       <EndPoint
Intersecting="true" Name="T', num2str(f),'-T',num2str(t), '" Position="',
num2str(Airport(f,1).sharemid{t,1}(1,1)), ' ', num2str(Airport(f,1).sharemid{t,1}(1,2)), '"/>']);
                                fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
                                n=n+1;
                                % if both f and t have more than one
                                % centroid, than the fDrawMulti() function
                                % can create all necessary links.
                            end
                        end
                    end
                end
            end
        end
    end
end
delete(h);
%%
%Draw link connected gate
if exist('Gate','var')
    index = length (Gate.Airport);

    for i = 1:1:index
        fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
        fprintf(Output_File_ID, '%s\n', ['      <Airport>',Airport(1,1).ICAO{1,1}, '</Airport>']);
        fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n), '</Name>']);
        fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
        fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
        fprintf(Output_File_ID, '%s\n', '           <DistributionType>number</DistributionType>');
        fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
        fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
        if isnumeric(Gate.Name{i,1})
            fprintf(Output_File_ID, '%s\n', ['      <EndPoint Intersecting="true" Name="G',
num2str(Gate.Name{i,1}), '" Position="', num2str(Gate.Position(i,1)),'
',num2str(Gate.Position(i,2)),'"/>']);
        else
            fprintf(Output_File_ID, '%s\n', ['      <EndPoint Intersecting="true" Name="G',
Gate.Name{i,1}, '" Position="', num2str(Gate.Position(i,1)),'
',num2str(Gate.Position(i,2)),'"/>']);
        end
```

```
        fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="I',
num2str(i), '" Position="', num2str(Gate.Intersection(i,1)), ' ', num2str(Gate.Intersection(i,2)),
'"/>']);
        fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
        n=n+1;
    end
end

%%
% Create Runway exit
h = waitbar(0,'Loading','Name','Creating Runway Exits...');
ICAO = Airport(1, 1).ICAO{1, 1};
[ICAO_All,Runway,Latitude,Longitude,Elevation,Slope,Width] = importRunway('Runway.csv');
ind = find(ismember(ICAO_All,ICAO));
numOfRWY = length(ind);
%% find origin of local projection system
IATA = strcat('"',Airport(1, 1).IATA{1, 1},'"');
[IATA_CODE,~,~,LATITUDE,LONGITUDE] = importAirport('Airport.csv');

ind2 = find(ismember(IATA_CODE,IATA));

if isempty(ind2)
    msgbox('Cannot find this airport in database');
    return
else
    UTMzone = utmzone(LATITUDE(ind2),LONGITUDE(ind2));        %find the UTMzone of the airport
    UTMzone = regexp(UTMzone,'\d+','match');
    UTMzone = str2num(UTMzone{1});%cut UTMzone into the format required by wgs2utm()
    [OriginUTM(1,1),OriginUTM(1,2)] =
wgs2utm(LATITUDE(ind2),LONGITUDE(ind2),UTMzone,'N'); %change 'N' for proper hemisphere if used
outside US continenet
end

%% convert WGS to UTM
[location(:,1),location(:,2)] = wgs2utm(Latitude(ind),Longitude(ind),UTMzone,'N');
%% load runway name and other runway info and transfer UTM to local projection system
name = cell(numOfRWY,1);
for i = 1:1:numOfRWY
    name{i,1} = Runway{ind(i),1};
    location(i,:) = location(i,:) - OriginUTM(1,:);
end

ele = Elevation(ind,1);
slope = Slope(ind,1);
width = Width(ind,1);

% create runway list
runwayList = cell(numOfRWY,9);
n = 1;
cellfind = @(string)(@(cell_contents)(strcmp(string,cell_contents)));
for i = 1:size(Airport,1)
    if
strcmp(Airport(i,1).element_feature_type,'aixm_RunwayElement')&&strcmp(Airport(i,1).element_sub_t
ype,'NORMAL')
        if ~cellfun(cellfind(Airport(i).label{1}),runwayList(:,1))
            runwayList{n,1} = Airport(i).label;
            runwayList{n+1,1} = Airport(i).label;
            n =n+2;
        end
    end
end

% fill runway list with the correct record
cellfind = @(string)(@(cell_contents)(strfind(cell_contents,string)));
for i = 1:numOfRWY
    ind = cellfun(cellfind(name{i}(3:end)),runwayList(:,1));
    ind = find(~cellfun(@isempty,ind)==1);
    if isempty(runwayList{ind(1),3})
        j = ind(1);
    else
        j = ind(2);
```

```
    end
    runwayList{j,2} = name(i);
    runwayList{j,3} = location(i,:);
    runwayList{j,4} = ele(i,1);
    runwayList{j,5} = slope(i,1);
    runwayList{j,6} = width(i,1);
end

% project exits on runway centerline and write its info into Taxiways.xml
for i=1:1:length(Airport)
    waitbar(i/(length(Airport)),h,sprintf('Preparing Index %g out of %g',i,length(Airport)));
    if
strcmp(Airport(i,1).element_feature_type,'aixm_RunwayElement')&&strcmp(Airport(i,1).element_sub_t
ype,'NORMAL')&&size(Airport(i,1).centroid{1,1},2)==3
        ind = cellfun(cellfind(Airport(i).label{1}),runwayList(:,1));
        ind = find(~cellfun(@isempty,ind)==1);
        RWY1 = ind(1);
        RWY2 = ind(2);
        exit = Airport(i,1).centroid{1,1};
        %find projected point of runway exit on runway centerline
        for j = 1:1:size(Airport(i,1).centroid{1,1},1)
            exit(j,1:2) =
pointProjectEdge(Airport(i,1).centroid{1,1}(j,1:2),[runwayList{RWY1,3}(1),
runwayList{RWY1,3}(2),runwayList{RWY2,3}(1),runwayList{RWY2,3}(2)]);
        end
        exit(:,3) = Airport(i,1).centroid{1,1}(:,3);
        numOfExit = size(exit,1);
        for k =1:1:numOfExit
            fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
            fprintf(Output_File_ID, '%s\n', ['      <Airport>',Airport(1,1).ICAO{1,1},
'</Airport>']);
            isMulti = find(cellfun(@isempty,Airport(exit(k,3),1).sharemid(:,1))~=1);
            if
length(isMulti)>1&&strcmp(Airport(isMulti(2),1).element_feature_type,'aixm_RunwayElement')
                if strcmp(Airport(isMulti(2),1).label,Airport(i,1).label)
                    fprintf(Output_File_ID, '%s\n', ['      <Name>Exit ',num2str(exit(k,3)),'-
',strcat(Airport(i,1).label{1,1},'-',num2str(i)),'</Name>']);
                else
                    fprintf(Output_File_ID, '%s\n', ['      <Name>Exit ',num2str(exit(k,3)),'-
',Airport(i,1).label{1,1},'</Name>']);
                end
            else
                fprintf(Output_File_ID, '%s\n', ['      <Name>Exit
',num2str(exit(k,3)),'</Name>']);
            end
            fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
            fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
            fprintf(Output_File_ID, '%s\n', '
<DistributionType>number</DistributionType>');
            fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
            fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
            if
length(isMulti)>1&&strcmp(Airport(isMulti(2),1).element_feature_type,'aixm_RunwayElement')
                if strcmp(Airport(isMulti(2),1).label,Airport(i,1).label)
                    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true"
Name="E', num2str(exit(k,3)),'-',strcat(Airport(i,1).label{1,1},'-',num2str(i)), '"
Position="',num2str(exit(k,1)),' ',num2str(exit(k,2)),'"/>']);
                else
                    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true"
Name="E', num2str(exit(k,3)),'-',Airport(i,1).label{1,1}, '" Position="',num2str(exit(k,1)),'
',num2str(exit(k,2)),'"/>']);
                end
            else
                fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="E',
num2str(exit(k,3)), '" Position="',num2str(exit(k,1)),' ',num2str(exit(k,2)),'"/>']);
            end
            fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T',
num2str(exit(k,3)), '" Position="', num2str(Airport(exit(k,3),1).centroid{1,1}(1,1)), ' ',
num2str(Airport(exit(k,3),1).centroid{1,1}(1,2)), '"/>']);
            fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
```

```
        end
    end
end
delete(h);
%%
% Create footer of XML File
fprintf(Output_File_ID, '%s\n', '</ADSIMpInput>');
fclose all;
end
%%
function [n] = fDrawMulti(Airport,f,n,fileName)
Output_File_ID = fopen(fileName, 'a+');
fNumOfCen = size(Airport(f, 1).centroid{1, 1},1);
neigh = ~cellfun(@isempty,Airport(f, 1).sharemid(:,1)); %find sharemids
ind = find(neigh~=0);  % length(ind) should equal to 2.

nodes =  ones(fNumOfCen+2,2);   %create nodes to store all the centroids and sharemids
nodes(1,:) = Airport(f,1).sharemid{ind(1),1};
nodes(fNumOfCen+2,:) = Airport(f,1).sharemid{ind(2),1};
for i = 1:1:fNumOfCen
    nodes(i+1,:) = Airport(f,1).centroid{1,1}(i,:);
end

%sort all the nodes start from ind(1) anti-clockwise
cx = mean(nodes(:,1));
cy = mean(nodes(:,2));
a = atan2(nodes(:,2)-cy, nodes(:,1)-cx);
[~,order] = sort(a);
nodes(:,1) = nodes(order,1);
nodes(:,2) = nodes(order,2);
nodes(:,1) = circshift(nodes(:,1),(-find(nodes(:,1) ==Airport(f,1).sharemid{ind(1),1}(1,1),1)
+1));
nodes(:,2) = circshift(nodes(:,2),(-find(nodes(:,2) ==Airport(f,1).sharemid{ind(1),1}(1,2),1)
+1));


if nodes(fNumOfCen+2,:)~=Airport(f,1).sharemid{ind(2),1}
    if nodes(2,:) == Airport(f,1).sharemid{ind(2),1}  % if anti-clockwise connect the two
sharemid directly then rotate to clockwise
        a = -atan2(nodes(:,2)-cy, nodes(:,1)-cx);
        [~,order] = sort(a);
        nodes(:,1) = nodes(order,1);
        nodes(:,2) = nodes(order,2);
        nodes(:,1) = circshift(nodes(:,1),(-find(nodes(:,1)
==Airport(f,1).sharemid{ind(1),1}(1,1),1) +1));
        nodes(:,2) = circshift(nodes(:,2),(-find(nodes(:,2)
==Airport(f,1).sharemid{ind(1),1}(1,2),1) +1));
    else  %if the clockwise or anti-clockwise can not sort the centroids properly. Move the
(cx,cy) to make sure the result of sorting to be correct. Although this method cannot fit all the
situations (extreme irregular shape), the 34 hub airports tested have no related issue.
        angle =
angle2Points(Airport(f,1).sharemid{ind(1),1}(1,:),Airport(f,1).sharemid{ind(2),1}(1,:));
        cx = cx -500*sin(angle);
        cy = cy +500*cos(angle);
        a = atan2(nodes(:,2)-cy, nodes(:,1)-cx);
        [~,order] = sort(a);
        nodes(:,1) = nodes(order,1);
        nodes(:,2) = nodes(order,2);
        nodes(:,1) = circshift(nodes(:,1),(-find(nodes(:,1) ==
Airport(f,1).sharemid{ind(1),1}(1,1),1) +1));
        nodes(:,2) = circshift(nodes(:,2),(-find(nodes(:,2) ==
Airport(f,1).sharemid{ind(1),1}(1,2),1) +1));
        if nodes(fNumOfCen+2,:)~=Airport(f,1).sharemid{ind(2),1} % check again
            if nodes(2,:) == Airport(f,1).sharemid{ind(2),1}
                a = -atan2(nodes(:,2)-cy, nodes(:,1)-cx);
                [~,order] = sort(a);
                nodes(:,1) = nodes(order,1);
                nodes(:,2) = nodes(order,2);
                nodes(:,1) = circshift(nodes(:,1),(-find(nodes(:,1)
==Airport(f,1).sharemid{ind(1),1}(1,1),1) +1));
```

```
                nodes(:,2) = circshift(nodes(:,2),(-find(nodes(:,2)
==Airport(f,1).sharemid{ind(1),1}(1,2),1) +1));
            else
                angle =
angle2Points(Airport(f,1).sharemid{ind(1),1}(1,:),Airport(f,1).sharemid{ind(2),1}(1,:));
                cx = cx +500*sin(angle);
                cy = cy -500*cos(angle);
                a = atan2(nodes(:,2)-cy, nodes(:,1)-cx);
                [~,order] = sort(a);
                nodes(:,1) = nodes(order,1);
                nodes(:,2) = nodes(order,2);
                nodes(:,1) = circshift(nodes(:,1),(-find(nodes(:,1) ==
Airport(f,1).sharemid{ind(1),1}(1,1),1) +1));
                nodes(:,2) = circshift(nodes(:,2),(-find(nodes(:,2) ==
Airport(f,1).sharemid{ind(1),1}(1,2),1) +1));
            end
        end
    end
end

%draw the link between start of centerline and sharemid ind(1)
fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
fprintf(Output_File_ID, '%s\n', ['      <Airport>',Airport(1,1).ICAO{1,1}, '</Airport>']);
fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n), '</Name>']);
fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
fprintf(Output_File_ID, '%s\n', '           <DistributionType>number</DistributionType>');
fprintf(Output_File_ID, '%s\n', '           <DistributionParameters>0</DistributionParameters>');
fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T", num2str(f),'-
T',num2str(ind(1)), '" Position="', num2str(nodes(1,1)), ' ', num2str(nodes(1,2)), '"/>']);
fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T", num2str(f),'-1"
Position="', num2str(nodes(2,1)), ' ', num2str(nodes(2,2)), '"/>']);
fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
n = n+1;

% connect the centerline
for j = 1:1: fNumOfCen-1
    fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
    fprintf(Output_File_ID, '%s\n', ['      <Airport>',Airport(1,1).ICAO{1,1}, '</Airport>']);
    fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n), '</Name>']);
    fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
    fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
    fprintf(Output_File_ID, '%s\n', '           <DistributionType>number</DistributionType>');
    fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
    fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T",
num2str(f),'-',num2str(j), '" Position="', num2str(nodes(j+1,1)), ' ', num2str(nodes(j+1,2)),
'"/>']);
    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T",
num2str(f),'-',num2str(j+1) '" Position="', num2str(nodes(j+2,1)), ' ', num2str(nodes(j+2,2)),
'"/>']);
    fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
    n = n+1;
end

% draw the link between the end of centerline and the other sharemid, which
% is ind(2)
fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
fprintf(Output_File_ID, '%s\n', ['      <Airport>',Airport(1,1).ICAO{1,1}, '</Airport>']);
fprintf(Output_File_ID, '%s\n', ['      <Name>Link ', num2str(n), '</Name>']);
fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
fprintf(Output_File_ID, '%s\n', '           <DistributionType>number</DistributionType>');
fprintf(Output_File_ID, '%s\n', '           <DistributionParameters>0</DistributionParameters>');
fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T", num2str(f),'-
',num2str(fNumOfCen), '" Position="', num2str(nodes(fNumOfCen+1,1)), ' ',
num2str(nodes(fNumOfCen+1,2)), '"/>']);
```

```
fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="T', num2str(f),'-
T',num2str(ind(2)),'" Position="', num2str(Airport(f,1).sharemid{ind(2),1}(1,1)), ' ',
num2str(Airport(f,1).sharemid{ind(2),1}(1,2)), '"/>']);
fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
n = n+1;
end

function [AIRPORT_ID,RUNWAYEND_ID,LATITUDE,LONGITUDE,ELEVATION1,GLIDE_SLOPE_DEGREES,RW_WIDTH] =
importRunway(filename, startRow, endRow)
%IMPORTFILE1 Import numeric data from a text file as column vectors.
%
[AIRPORT_ID,RUNWAYEND_ID,LATITUDE,LONGITUDE,ELEVATION1,GLIDE_SLOPE_DEGREES,RW_WIDTH,FILE_RECORD_N
UM]
%   = IMPORTFILE1(FILENAME) Reads data from text file FILENAME for the
%   default selection.
%
%
[AIRPORT_ID,RUNWAYEND_ID,LATITUDE,LONGITUDE,ELEVATION1,GLIDE_SLOPE_DEGREES,RW_WIDTH,FILE_RECORD_N
UM]
%   = IMPORTFILE1(FILENAME, STARTROW, ENDROW) Reads data from rows STARTROW
%   through ENDROW of text file FILENAME.
%
% Example:
%
[AIRPORT_ID,RUNWAYEND_ID,LATITUDE,LONGITUDE,ELEVATION1,GLIDE_SLOPE_DEGREES,RW_WIDTH,FILE_RECORD_N
UM]
%   = importfile1('Runway.csv',2, 10583);
%
%    See also TEXTSCAN.

% Auto-generated by MATLAB on 2014/01/27 22:45:03

%% Initialize variables.
delimiter = ',';
if nargin<=2
    startRow = 2;
    endRow = inf;
end

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

%% Open the text file.
fileID = fopen(filename,'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 = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter', delimiter,
'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1, 'Delimiter',
delimiter, 'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));
```

```
for col=[3,4,5,6,7,8]
    % Converts strings in the input cell array to numbers. Replaced
    % non-numeric strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric
        % prefixes and suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-
+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [3,4,5,6,7,8]);
rawCellColumns = raw(:, [1,2]);


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); % Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
AIRPORT_ID = rawCellColumns(:, 1);
RUNWAYEND_ID = rawCellColumns(:, 2);
LATITUDE = cell2mat(rawNumericColumns(:, 1));
LONGITUDE = cell2mat(rawNumericColumns(:, 2));
ELEVATION1 = cell2mat(rawNumericColumns(:, 3));
GLIDE_SLOPE_DEGREES = cell2mat(rawNumericColumns(:, 4));
RW_WIDTH = cell2mat(rawNumericColumns(:, 5));
end

function [IATA_CODE,ICAO_CODE,FAA_CODE,LATITUDE,LONGITUDE] = importAirport(filename, startRow,
endRow)
%IMPORTAirport Import numeric data from a text file as column vectors.
%   [IATA_CODE,ICAO_CODE,FAA_CODE,ICAO_REGIONELEVATION,LATITUDE,LONGITUDE]
%   = importAirport(FILENAME) Reads data from text file FILENAME for the
%   default selection.
%
%   [IATA_CODE,ICAO_CODE,FAA_CODE,ICAO_REGIONELEVATION,LATITUDE,LONGITUDE]
%   = importAirport(FILENAME, STARTROW, ENDROW) Reads data from rows
%   STARTROW through ENDROW of text file FILENAME.
%
% Example:
%   [IATA_CODE,ICAO_CODE,FAA_CODE,ICAO_REGIONELEVATION,LATITUDE,LONGITUDE]
%   = importAirport('Airport.csv',1, 14277);
%
%    See also TEXTSCAN.

% Auto-generated by MATLAB on 2014/01/16 17:23:18
```

```
%% Initialize variables.
delimiter = ',';
if nargin<=2
    startRow = 1;
    endRow = inf;
end

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'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 = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1, 'Delimiter', delimiter,
'HeaderLines', startRow(1)-1, 'ReturnOnError', false);
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec, endRow(block)-startRow(block)+1, 'Delimiter',
delimiter, 'HeaderLines', startRow(block)-1, 'ReturnOnError', false);
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[11,12,13,14,15,16,17,22,23]
    % Converts strings in the input cell array to numbers. Replaced
    % non-numeric strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric
        % prefixes and suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-
+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
```

```
    end
end

%% Split data into numeric and cell columns.
rawNumericColumns = raw(:, [11,12,13,14,15,16,17,22,23]);
rawCellColumns = raw(:, [1,2,3,4,5,6,7,8,9,10,18,19,20,21]);


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),rawNumericColumns); % Find non-numeric cells
rawNumericColumns(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
IATA_CODE = rawCellColumns(:, 3);
ICAO_CODE = rawCellColumns(:, 4);
FAA_CODE = rawCellColumns(:, 5);
LATITUDE = cell2mat(rawNumericColumns(:, 2));
LONGITUDE = cell2mat(rawNumericColumns(:, 3));
end

function [p0] = pointProjectEdge(point, edge)
%edge has the form: [x1 y1 x2 y2], and point is [x y].
% direction vector of each edge
dx = edge(:, 3) - edge(:,1);
dy = edge(:, 4) - edge(:,2);

% compute position of points projected on the supporting line (Size of tp
% is the max number of edges or points)
delta = dx .* dx + dy .* dy;
tp = ((point(:, 1) - edge(:, 1)) .* dx + (point(:, 2) - edge(:, 2)) .* dy) ./ delta;

% ensure degenerated edges are correclty processed (consider the first
% vertex is the closest)
tp(delta < eps) = 0;

% change position to ensure projected point is located on the edge
tp(tp < 0) = 0;
tp(tp > 1) = 1;

% coordinates of projected point
p0 = [edge(:,1) + tp .* dx, edge(:,2) + tp .* dy];
end
```

```
function[edgeListSpf,edgeSpf,nodeListSpf,nodeSpf,adjMatrixSpf,runwayList] =
simplifyNet(Airport,runwayList, file)
%%
% Descrption:
%   1. read taxiways.xml into matlab structure file for network analysis purpose.
%   2. delete the node only connected to two edges, while the angle between
%      the two edges is around 180?
%   3. connect two node if they are close enough, via substitle either one.
%   4. add edges between runway exit inside runway.
%   5. rewrite the simplified network into taxiways.xml
% Input:
%   Airport: airport structure file processed by WGS2LCS()
%   runwayList: cell file containing all the runway info, whose col 1: runway
%   label, col 2: runway end name, col 3: runway end location info, col 4:
%   runway elevation info, col 5: runway slope, col 6: runway width.
%   file: the directory of the taxiways.xml
%   Airport.csv: Airport database from ADSIM+ built-in database.
% Output:
%   edgeListSpf(cell): Simplified edge list stored the edge name and the index of
%                      nodes on the edge.
%   edgeSpf(array): Simplified edge list stored the index of nodes on the edge.
%   nodeListSpf(cell): Simplified node list stored the node name.
%   nodeSpf: Simplified node list stored the node position info.
%   adjMatrixSpf: Adjacency matrix of simplified network.
%   runwayList: cell file containing runway info(required input files for
%   CreateRunways()), whose col 1: runway label, col 2: runway end name,
%   col 3: runway end location info, col 4: runway elevation info, col 5: runway slope,
%   col 6: runway width, col 7: runway index in Airport structure file, col 8:
%   eixt index in nodeListSpf file. col 9: airport ICAO.
% Subfunctions:
%   readEdgeList(), parseChildNodes(), getNodeData(), parseAttributes(), pointProjectEdge()

%%
[edgeList,edge,nodeList,node] = readEdgeList(file);     %read taxiways.xml created by
CreateTaxiways() to get the edge and node list
id1 = edge(:,1);    %'from' node index list of each edge
id2 = edge(:,2);    %'t' node index list of each edge
adj = sparse(id1,id2,1,size(node,1),size(node,1));      %build the node-edge list
adjMatrix = full(adj);      %convert node-edge list to adjacency matrix

edgeListSpf = edgeList;     %initiate the simplified edge name list
edgeSpf = edge;     %initiate the simplified edge info list
m = size(edgeSpf,1);    %measure the length of the initial simplified edge info list


h = waitbar(0,'Loading','Name','Simplifying Taxiway Network...');
%run a loop to simplify the network
for i = 1:1:size(node,1)        %i is the index of node f
    waitbar(i/(size(node,1)),h,sprintf('Preparing Index %g out of %g',i,size(node,1)))
    if nodeList{i,1}(1)~='G'    %'G'(gate) can't be deleted
        %find the edge contain the node f
        [a,b] = find(edgeSpf == i); %a is the edge index contain node f, b is the node f index or
the index of node t of node f on that edge
        if length(a) == 2
            %run a set of 'if...else' conditional operator to find the node
            %index of a P1----P0----P2 struct, and measure the angle of
            %P1-P0-P2. If the angle close to 180 degree, then delete node
            %P0 in the network.
            if cellfun(@isempty,strfind(edgeListSpf(a(1),1),'Exit'))&&
cellfun(@isempty,strfind(edgeListSpf(a(2),1),'Exit'))
                P0 = node(i,:);
                if b(1) == 1
                    indP1 = edgeSpf(a(1),2);
                    P1 = node(indP1,:);
                else
                    indP1 = edgeSpf(a(1),1);
                    P1 = node(indP1,:);
                end
                if b(2) == 1
                    indP2 = edgeSpf(a(2),2);
                    P2 = node(indP2,:);
```

```
                else
                    indP2 = edgeSpf(a(2),1);
                    P2 = node(indP2,:);
                end
                ang = atan2(abs(det([P2-P0;P1-P0])),dot(P2-P0,P1-P0));      %measure the angle of
P1-P0-P2
                if abs(pi - ang)<0.05*pi    %0.05pi(9 degree) is used as the default thershold to
judge if the node P0 can be deleted
                    edgeListSpf(a,:) = [];
                    edgeSpf = edgeSpf(~ismember(1:size(edgeSpf,1),a),:);
                    linkName = strcat('Link',32,num2str(m+1));
                    n = size(edgeSpf,1);
                    edgeListSpf(n+1,:) = {linkName,indP1,indP2,edgeList{1, 4}};
                    edgeSpf(n+1,:) = [indP1,indP2,0];
                    m = m+1;
                end
            end
        end
    end
end



for i = 1:1:size(node,1)        %combine two nodes if they are close enough
    waitbar(i/(size(node,1)),h,sprintf('Preparing Index %g out of %g',i,size(node,1)));
    [ai,bi] = find(edgeSpf == i);
    if length(ai)==1
        for j = (i+1):1:size(node,1)
            [aj,~] = find(edgeSpf == j);
            if length(aj)==1
                distanceExit = distancePoints(node(i,:),node(j,:));
                if distanceExit<15
                    if bi(1) == 1
                        indP1 = edgeSpf(ai(1),2);
                    else
                        indP1 = edgeSpf(ai(1),1);
                    end
                    linkName = edgeListSpf{ai,1};
                    edgeListSpf(ai,:) = [];
                    edgeSpf = edgeSpf(~ismember(1:size(edgeSpf,1),ai),:);
                    n = size(edgeSpf,1);
                    edgeListSpf(n+1,:) = {linkName,indP1,j,edgeList{1, 4}};
                    edgeSpf(n+1,:) = [indP1,j,0];
                    m = m+1;
                end
            end
        end
    end
end
%%
%write simplified taxiway network into ADSIM+ input file

%%
fileName = file;
Output_File_ID = fopen(fileName, 'w');
Output_File_ID = fopen(fileName, 'a+');

%% --------------------------------
% Create Header of XML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '<!DOCTYPE ADSIMpInput>');
fprintf(Output_File_ID, '%s\n', '<ADSIMpInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ADSIMpInput.xsd">');
%%
% create taxiways network

for i = 1:1:size(edgeSpf,1)
    waitbar(i/(size(edgeSpf,1)),h,sprintf('Preparing Index %g out of %g',i,size(edgeSpf,1)));
    fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
    fprintf(Output_File_ID, '%s\n', ['      <Airport>',edgeListSpf{i, 4}, '</Airport>']);
    fprintf(Output_File_ID, '%s\n', ['      <Name>', edgeListSpf{i, 1}, '</Name>']);
```

```
    fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
    fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
    fprintf(Output_File_ID, '%s\n', '           <DistributionType>number</DistributionType>');
    fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
    fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="',
nodeList{edgeSpf(i,1),1}, '" Position="', num2str(node(edgeSpf(i,1),1)), ' ',
num2str(node(edgeSpf(i,1),2)), '"/>']);
    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="',
nodeList{edgeSpf(i,2),1}, '" Position="', num2str(node(edgeSpf(i,2),1)), ' ',
num2str(node(edgeSpf(i,2),2)), '"/>']);
    fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
end
% Create footer of XML File
fprintf(Output_File_ID, '%s\n', '</ADSIMpInput>');
fclose all;

[edgeListSpf,edgeSpf,nodeListSpf,nodeSpf] = readEdgeList(fileName);

%connect runway exits inside runway

for i=1:1:length(Airport)
    waitbar(i/(length(Airport)),h,sprintf('Preparing Index %g out of %g',i,length(Airport)));
    if
strcmp(Airport(i,1).element_feature_type,'aixm_RunwayElement')&&strcmp(Airport(i,1).element_sub_t
ype,'NORMAL')
        for j = 1:1:size(runwayList,1)
            if strcmp(Airport(i,1).label,runwayList{j,1})
               runwayList{j,7} = cat(1,runwayList{j,7},i);
            end
        end
    end
end

for i =1:size(nodeSpf,1)
    waitbar(i/(size(nodeSpf,1)),h,sprintf('Preparing Index %g out of %g',i,size(nodeSpf,1)));
    if nodeListSpf{i,1}(1) == 'E'
        for j = 1:2:size(runwayList,1)
            for k =1:size(runwayList{j,7},1)
                if inpolygon(nodeSpf(i,1),nodeSpf(i,2),Airport(runwayList{j,7}(k)).extent{1,
1}(:,1),Airport(runwayList{j,7}(k)).extent{1, 1}(:,2))
                   runwayList{j,8} = cat(1,runwayList{j,8},i);
                   runwayList{j+1,8} = cat(1,runwayList{j+1,8},i);
                end
            end
        end
    end
end

n = 1;
m = size(edgeSpf,1);
for i =1:2:size(runwayList,1)
    sortval = nodeSpf(runwayList{i,8},1);
    [~,order] = sort(sortval);
    runwayList{i,8} = runwayList{i,8}(order,:);
    for j = 1:size(runwayList{i,8},1)-1
        edgeSpf(m+1,:) = [runwayList{i,8}(j),runwayList{i,8}(j+1),0];
        linkName = strcat('linkRWY',num2str(n));
        edgeListSpf(m+1,:) ={linkName,edgeSpf(m+1,1),edgeSpf(m+1,2),edgeListSpf{m,4}};
        m = m+1;
        n = n+1;
    end
    runwayList{i,9} = edgeList{1, 4};
    runwayList{i+1,9} = edgeList{1, 4};
end

%%
%write simplified taxiway network into ADSIM+ input file

%%
```

```
Output_File_ID = fopen(fileName, 'w');
Output_File_ID = fopen(fileName, 'a+');

%% --------------------------------
% Create Header of XML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '<!DOCTYPE ADSIMpInput>');
fprintf(Output_File_ID, '%s\n', '<ADSIMpInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ADSIMpInput.xsd">');
%%
% create taxiways network

for i = 1:1:size(edgeSpf,1)
    waitbar(i/(size(edgeSpf,1)),h,sprintf('Preparing Index %g out of %g',i,size(edgeSpf,1)));
    fprintf(Output_File_ID, '%s\n', '   <TaxiWay IncludeInStudy="true">');
    fprintf(Output_File_ID, '%s\n', ['      <Airport>',edgeListSpf{i, 4}, '</Airport>']);
    fprintf(Output_File_ID, '%s\n', ['      <Name>', edgeListSpf{i, 1}, '</Name>']);
    fprintf(Output_File_ID, '%s\n', '       <AircraftCategory></AircraftCategory>');
    fprintf(Output_File_ID, '%s\n', '       <AircraftSpeedMph>');
    fprintf(Output_File_ID, '%s\n', '           <DistributionType>number</DistributionType>');
    fprintf(Output_File_ID, '%s\n', '
<DistributionParameters>0</DistributionParameters>');
    fprintf(Output_File_ID, '%s\n', '       </AircraftSpeedMph>');
    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="',
nodeListSpf{edgeSpf(i,1),1}, '" Position="', num2str(nodeSpf(edgeSpf(i,1),1)), ' ',
num2str(nodeSpf(edgeSpf(i,1),2)), '"/>']);
    fprintf(Output_File_ID, '%s\n', ['       <EndPoint Intersecting="true" Name="',
nodeListSpf{edgeSpf(i,2),1}, '" Position="', num2str(nodeSpf(edgeSpf(i,2),1)), ' ',
num2str(nodeSpf(edgeSpf(i,2),2)), '"/>']);
    fprintf(Output_File_ID, '%s\n', '   </TaxiWay>');
end
% Create footer of XML File
fprintf(Output_File_ID, '%s\n', '</ADSIMpInput>');
fclose all;
delete(h);

[edgeListSpf,edgeSpf,nodeListSpf,nodeSpf] = readEdgeList(fileName);
hold on
id1 = edgeSpf(:,1);
id2 = edgeSpf(:,2);
adjSpf = sparse(id1,id2,1,size(nodeSpf,1),size(nodeSpf,1));
adjMatrixSpf = full(adjSpf);
gplot(adjMatrixSpf,nodeSpf,'or-');
%text(nodeSpf(:,1),nodeSpf(:,2),nodeListSpf);
end

function [edgeList,edge,nodeList,node] = readEdgeList(file)

xDoc = xmlread(file);
edges = parseChildNodes(xDoc);
edge= zeros(size(edges.ADSIMpInput{1, 2}.TaxiWay,2),3);
edgeList = cell(size(edges.ADSIMpInput{1, 2}.TaxiWay,2),3);
nodeList(1,:) = {' '};              %initilize nodeList cell file
nodeInd = 1;
for i = 1:1:size(edges.ADSIMpInput{1, 2}.TaxiWay,2)
    edgeList(i,1) = {edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.Name.Text};
    if isempty(find(strcmp(nodeList, edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
1}.Attributes.Name), 1))
        nodeList(nodeInd,1) = {edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
1}.Attributes.Name};
        node(nodeInd,:) = str2num(edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
1}.Attributes.Position);
        nodeInd = nodeInd +1;
    end
    if isempty(find(strcmp(nodeList, edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
2}.Attributes.Name), 1))
        nodeList(nodeInd,1) = {edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
2}.Attributes.Name};
        node(nodeInd,:) = str2num(edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
2}.Attributes.Position);
        nodeInd = nodeInd +1;
```

```
    end
    edge(i,1) = find(strcmp(nodeList, edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
1}.Attributes.Name),1);
    edgeList(i,2) = {find(strcmp(nodeList, edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
1}.Attributes.Name),1)};
    edge(i,2) = find(strcmp(nodeList, edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
2}.Attributes.Name),1);
    edgeList(i,3) = {find(strcmp(nodeList, edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.EndPoint{1,
2}.Attributes.Name),1)};
    edgeList(i,4) = {edges.ADSIMpInput{1, 2}.TaxiWay{1, i}.Airport.Text};
end

% text(node(:,1),node(:,2),nodeList);
end

% ----- Subfunction parseChildNodes -----
function [children,ptext] = parseChildNodes(theNode)
    % Recurse over node children.
    children = struct;
    ptext = [];
    if theNode.hasChildNodes
        childNodes = theNode.getChildNodes;
        numChildNodes = childNodes.getLength;

        for count = 1:numChildNodes
            theChild = childNodes.item(count-1);
            [text,name,attr,childs] = getNodeData(theChild);

            if (~strcmp(name,'#text') && ~strcmp(name,'#comment'))
                %XML allows the same elements to be defined multiple times,
                %put each in a different cell
                if (isfield(children,name))
                    if (~iscell(children.(name)))
                        %put existsing element into cell format
                        children.(name) = {children.(name)};
                    end
                    index = length(children.(name))+1;
                    %add new element
                    children.(name){index} = childs;
                    if(~isempty(text))
                        children.(name){index}.('Text') = text;
                    end
                    if(~isempty(attr))
                        children.(name){index}.('Attributes') = attr;
                    end
                else
                    %add previously unkown new element to the structure
                    children.(name) = childs;
                    if(~isempty(text))
                        children.(name).('Text') = text;
                    end
                    if(~isempty(attr))
                        children.(name).('Attributes') = attr;
                    end
                end
            elseif (strcmp(name,'#text'))
                %this is the text in an element (i.e. the parentNode)
                if (~isempty(regexprep(text,'[\s]*','')))
                    ptext = text;
                end
            end
        end
    end
end

% ----- Subfunction getNodeData -----
function [text,name,attr,childs] = getNodeData(theNode)
    % Create structure of node info.

    %make sure name is allowed as structure name
    name = regexprep(char(theNode.getNodeName),'[-:.]','_');
```

```
    attr = parseAttributes(theNode);
    if (isempty(fieldnames(attr)))
        attr = [];
    end

    %parse child nodes
    [childs,text] = parseChildNodes(theNode);

    if (isempty(fieldnames(childs)))
        %get the data of any childless nodes
        try
            %faster then if any(strcmp(methods(theNode), 'getData'))
            text = char(theNode.getData);
        catch
            %no data
        end
    end

end

% ----- Subfunction parseAttributes -----
function attributes = parseAttributes(theNode)
    % Create attributes structure.

    attributes = struct;
    if theNode.hasAttributes
       theAttributes = theNode.getAttributes;
       numAttributes = theAttributes.getLength;

       for count = 1:numAttributes
          attrib = theAttributes.item(count-1);
          attr_name = regexprep(char(attrib.getName),'[-:.]','_');
          attributes.(attr_name) = char(attrib.getValue);
       end
    end
end



function [p0] = pointProjectEdge(point, edge)
%edge has the form: [x1 y1 x2 y2], and point is [x y].
% direction vector of each edge
dx = edge(:, 3) - edge(:,1);
dy = edge(:, 4) - edge(:,2);

% compute position of points projected on the supporting line
% (Size of tp is the max number of edges or points)
delta = dx .* dx + dy .* dy;
tp = ((point(:, 1) - edge(:, 1)) .* dx + (point(:, 2) - edge(:, 2)) .* dy) ./ delta;

% ensure degenerated edges are correclty processed (consider the first
% vertex is the closest)
tp(delta < eps) = 0;

% change position to ensure projected point is located on the edge
tp(tp < 0) = 0;
tp(tp > 1) = 1;

% coordinates of projected point
p0 = [edge(:,1) + tp .* dx, edge(:,2) + tp .* dy];
end




function [fileName] = CreateRunways(nodeListSpf,nodeSpf,runwayList,ADSIM_dir)
%%
% Description:
```

```
%   write runway infomation into ADSIM+ input file runways.xml
% Input:
%   Airport: airport structure file processed by WGS2LCS(), SharePoint(),
%            GetCentroid()
%   runwayList: cell file containing runway info(required input files for
%   CreateRunways()), whose col 1: runway label, col 2: runway end name,
%   col 3: runway end location info, col 4: runway elevation info, col 5: runway slope,
%   col 6: runway width, col 7: runway index in Airport structure file, col 8:
%   eixt index in nodeListSpf file. col 9: airport ICAO.
% Output:
%   runways.xml: ADSIM+ required input file
%   fileName: runways.xml file directory (will be used as input for following function).
%
% Yang Zhang - 01/20/2014 - tested under MATLAB v8.1
%%
fileName = [ADSIM_dir,'\runways.xml'];      %ceate runways.xml directory
Output_File_ID = fopen(fileName, 'w');      %create or overwrite runways.xml file
Output_File_ID = fopen(fileName, 'a+');

%% --------------------------------
% Create Header of XML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '<!DOCTYPE ADSIMpInput>');
fprintf(Output_File_ID, '%s\n', '<ADSIMpInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ADSIMpInput.xsd">');

%%
h = waitbar(0,'Loading','Name','Creating Runways...');
%run a loop to create runways using the info from runwayList
for i=1:2:size(runwayList,1)
    waitbar(i/(size(runwayList,1)),h,sprintf('Preparing Index %g out of %g',i,size(runwayList,1)))
    if ~isempty(runwayList{i,2})
        RWY1 = i;
        RWY2 = i+1;
        fprintf(Output_File_ID, '%s\n', '   <RunWay IncludeInStudy="true">');
        fprintf(Output_File_ID, '%s\n', ['      <Airport>',runwayList{1,9},'</Airport>']);
        fprintf(Output_File_ID, '%s\n', ['      <RunWayEnd
AddNode="',runwayList{RWY1,2}{1,1},'">']);
        fprintf(Output_File_ID, '%s\n', ['          <Name>', runwayList{RWY1,2}{1,1}(3:end),
'</Name>']);
        fprintf(Output_File_ID, '%s\n', ['          <Position>', num2str(runwayList{RWY1,3}(1)),'
',num2str(runwayList{RWY1,3}(2)),'</Position>']);
        fprintf(Output_File_ID, '%s\n', ['          <Elevation>',
num2str(runwayList{RWY1,4}(1)),'</Elevation>']);
        fprintf(Output_File_ID, '%s\n', ['
<GlideSlopeDegrees>',num2str(runwayList{RWY1,5}(1)),'</GlideSlopeDegrees>']);
        for j = 1:1:size(runwayList{i,8})
            fprintf(Output_File_ID, '%s\n', ['          <Exit
Name="',nodeListSpf{runwayList{RWY1,8}(j)},'"
Position="',num2str(nodeSpf(runwayList{RWY1,8}(j),1)),'
',num2str(nodeSpf(runwayList{RWY1,8}(j),2)),'"/>']);
            fprintf(Output_File_ID, '%s\n', ['          <Entrance
Name="',nodeListSpf{runwayList{RWY1,8}(j)},'"
Position="',num2str(nodeSpf(runwayList{RWY1,8}(j),1)),'
',num2str(nodeSpf(runwayList{RWY1,8}(j),2)),'"/>']);
        end
        fprintf(Output_File_ID, '%s\n', '       </RunWayEnd>');
        fprintf(Output_File_ID, '%s\n', ['      <RunWayEnd
AddNode="',runwayList{RWY2,2}{1,1},'">']);
        fprintf(Output_File_ID, '%s\n', ['          <Name>', runwayList{RWY2,2}{1,1}(3:end),
'</Name>']);
        fprintf(Output_File_ID, '%s\n', ['          <Position>', num2str(runwayList{RWY2,3}(1)),'
',num2str(runwayList{RWY2,3}(2)),'</Position>']);
        fprintf(Output_File_ID, '%s\n', ['          <Elevation>',
num2str(runwayList{RWY2,4}(1)),'</Elevation>']);
        fprintf(Output_File_ID, '%s\n', ['
<GlideSlopeDegrees>',num2str(runwayList{RWY2,5}(1)),'</GlideSlopeDegrees>']);
        for j = 1:1:size(runwayList{i,8})
            fprintf(Output_File_ID, '%s\n', ['          <Exit
Name="',nodeListSpf{runwayList{RWY2,8}(j)},'"
```

```
Position="',num2str(nodeSpf(runwayList{RWY2,8}(j),1)),'
',num2str(nodeSpf(runwayList{RWY2,8}(j),2)),'"/>']);
            fprintf(Output_File_ID, '%s\n', ['          <Entrance
Name="',nodeListSpf{runwayList{RWY2,8}(j)},'"
Position="',num2str(nodeSpf(runwayList{RWY2,8}(j),1)),'
',num2str(nodeSpf(runwayList{RWY2,8}(j),2)),'"/>']);
        end
        fprintf(Output_File_ID, '%s\n', '       </RunWayEnd>');
        fprintf(Output_File_ID, '%s\n', '   </RunWay>');
    end
end
%%
% Create footer of XML File
fprintf(Output_File_ID, '%s\n', '</ADSIMpInput>');
fclose all;
delete(h);
end
```

```
function [fileName] = CreateGates2(Airport,nodeListSpf,nodeSpf,ADSIM_dir)
%%
% Description:
%   create ADSIM+ input file gates.xml
% Input:
%   Gate: gate structure file processed by gateinfo2struct()
% Output:
%   gates.xml: ADSIM+ required input file
% Yang Zhang - 01/19/2014 - tested under MATLAB v8.1
%%
fileName = [ADSIM_dir,'\gates.xml'];
Output_File_ID = fopen(fileName, 'w');

% --------------------------------
% Create Header of XML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '<!DOCTYPE ADSIMpInput>');
fprintf(Output_File_ID, '%s\n', '<ADSIMpInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ADSIMpInput.xsd">');

% --------------------------------
% Write Gate Info
% --------------------------------

index = length (Airport);
nodeListSpf(:,2) = {0};
h = waitbar(0,'Loading','Name','Creating Gates...');
%run a loop to find the mid point of the edge shared  by a taxiway element
%and an apron element. Use that mid point as the dummy gates.
for i = 1:1:index
    waitbar(i/index,h,sprintf('Preparing Index %g out of %g',i,index));
    if
strcmp(Airport(i,1).element_feature_type,'aixm_ApronElement')&&isempty(strfind(Airport(i).label{1
},'CARGO'))&&isempty(strfind(Airport(i).label{1},'GENERAL'))&&isempty(strfind(Airport(i).label{1}
,'HELICOPTER'))
        for j = 1:index
            if strcmp(Airport(j,1).element_feature_type,'aixm_TaxiwayElement')
                dist = distancePointPolygon(Airport(j,1).extent{1,1},
Airport(i,1).extent{1,1});    %recognize vertices on the edge shared by element i and j
                p = dist < 0.5;
                Share = Airport(j,1).extent{1,1}(p,:);      %vertices on shared edge
                if ~isempty (Share)
                    if size(Share,1)>1&&polylineLength(Share)>5
                        gateName = strcat('T',num2str(j));
                        nodeIndex = find(strcmp(nodeListSpf,gateName)~=0,1);
                        if ~isempty(nodeIndex)&& nodeListSpf{nodeIndex,2} == 0
                            if size(nodeIndex,1)==1
                                nodeListSpf{nodeIndex,2} = 1;
                                fprintf(Output_File_ID, '%s\n', '   <Gate
IncludeInStudy="true">');
                                fprintf(Output_File_ID, '%s\n', ['
<Airport>',Airport(1).ICAO{1}, '</Airport>']);
                                fprintf(Output_File_ID, '%s\n', ['      <Name>',
strcat('G',num2str(j)), '</Name>']);
                                fprintf(Output_File_ID, '%s\n', '       <Type>R</Type>');
                                fprintf(Output_File_ID, '%s\n', '       <Airline></Airline>');
                                fprintf(Output_File_ID, '%s\n', '
<AircraftCategory></AircraftCategory>');
                                fprintf(Output_File_ID, '%s\n', '       <Capacity>0</Capacity>');
                                fprintf(Output_File_ID, '%s\n', '       <Elevation></Elevation>');
                                fprintf(Output_File_ID, '%s\n', ['      <BoundingPoint
Intersecting="true" Name="', gateName, '" Position="', num2str(nodeSpf(nodeIndex,1)),'
',num2str(nodeSpf(nodeIndex,2)),'"/>']);
                                fprintf(Output_File_ID, '%s\n', '   </Gate>');
                                break
                            end
                        else
                            gateName = strcat('G',num2str(j));
                            nodeIndex = find(strcmp(nodeListSpf,gateName)~=0,1);
                            if ~isempty(nodeIndex)&& nodeListSpf{nodeIndex,2} == 0
                                if size(nodeIndex,1)==1
```

```
                                    nodeListSpf{nodeIndex,2} = 1;
                                    fprintf(Output_File_ID, '%s\n', '   <Gate
IncludeInStudy="true">');
                                    fprintf(Output_File_ID, '%s\n', ['
<Airport>',Airport(1).ICAO{1}, '</Airport>']);
                                    fprintf(Output_File_ID, '%s\n', ['      <Name>',gateName,
'</Name>']);
                                    fprintf(Output_File_ID, '%s\n', '       <Type>R</Type>');
                                    fprintf(Output_File_ID, '%s\n', '       <Airline></Airline>');
                                    fprintf(Output_File_ID, '%s\n', '
<AircraftCategory></AircraftCategory>');
                                    fprintf(Output_File_ID, '%s\n', '
<Capacity>0</Capacity>');
                                    fprintf(Output_File_ID, '%s\n', '
<Elevation></Elevation>');
                                    fprintf(Output_File_ID, '%s\n', ['      <BoundingPoint
Intersecting="true" Name="', gateName, '" Position="', num2str(nodeSpf(nodeIndex,1)),'
',num2str(nodeSpf(nodeIndex,2)),'"/>']);
                                    fprintf(Output_File_ID, '%s\n', '   </Gate>');
                                    break
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end

% --------------------------------
% Create footer of XML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '</ADSIMpInput>');
fclose all;
delete(h);
end
```

```
function [] =
CreateTaxipaths(runwayfile,gatefile,edgeListSpf,edgeSpf,nodeListSpf,nodeSpf,adjMatrixSpf,ADSIM_di
r)
%%
% Descrption:
%   1. read runways.xml and gates.xml as the start and end points of taxipaths
%   2. calculate shortest paths (based on acutal distance, which can be substituted
%      as travel time if related analysis is available  between all the gates and
%      runway exits with dijkstra's algorithm.
%   3. write taxipath into taxipaths.xml as input file for ADSIM+
% Input:
%   runwayfile: runways.xml file directory.
%   gatefile: gates.xml file directory.
%   edgeListSpf(cell): Simplified edge list stored the edge name and the index of
%                      nodes on the edge.
%   edgeSpf(array): Simplified edge list stored the index of nodes on the edge.
%   nodeListSpf(cell): Simplified node list stored the node name.
%   nodeSpf: Simplified node list stored the node position info.
%   adjMatrixSpf: Adjacency matrix of simplified network.
% Output:
%   taxipaths.xml
% Subfunctions:
%   parseChildNodes(), getNodeData(), parseAttributes()
%%
% create runway exit list
h = waitbar(0,'Loading','Name','Creating Taxiway Paths...');
xDoc = xmlread(runwayfile);     %read runways.xml to matlab struct file
runways = parseChildNodes(xDoc);
ind1 = 1;       % initiate the index of the taxiway path
exitList(:,1:2) = {' '};              %initiate exitList cell file
%run a loop to create the exit list from the runways struct file
for i = 1:size(runways.ADSIMpInput{1, 2}.RunWay,2)
    if size(runways.ADSIMpInput{1, 2}.RunWay,2) == 1
        for j = 1:size(runways.ADSIMpInput{1, 2}.RunWay.RunWayEnd{1, 1}.Exit,2)
            exitList(ind1,1) = {strcat(runways.ADSIMpInput{1, 2}.RunWay.RunWayEnd{1,
1}.Name.Text,'-',runways.ADSIMpInput{1, 2}.RunWay.RunWayEnd{1, 2}.Name.Text)};
            exitList(ind1,2) = {runways.ADSIMpInput{1, 2}.RunWay.RunWayEnd{1, 1}.Exit{1,
j}.Attributes.Name};
            ind1 = 1+ind1;
        end
    else
        for j = 1:size(runways.ADSIMpInput{1, 2}.RunWay{1, i}.RunWayEnd{1, 1}.Exit,2)
            exitList(ind1,1) = {strcat(runways.ADSIMpInput{1, 2}.RunWay{1, i}.RunWayEnd{1,
1}.Name.Text,'-',runways.ADSIMpInput{1, 2}.RunWay{1, i}.RunWayEnd{1, 2}.Name.Text)};
            exitList(ind1,2) = {runways.ADSIMpInput{1, 2}.RunWay{1, i}.RunWayEnd{1, 1}.Exit{1,
j}.Attributes.Name};
            ind1 = 1+ind1;
        end
    end
end
%create gate list
xDoc = xmlread(gatefile);       %read gates.xml into matlab struct file
gates = parseChildNodes(xDoc);
m = size(gates.ADSIMpInput{1, 2}.Gate,2);   %gateList demension
gateList(1:m,1:2) = {' '};
for i = 1:size(gates.ADSIMpInput{1, 2}.Gate,2)
    gateList(i,1) = {gates.ADSIMpInput{1, 2}.Gate{1, i}.Name.Text};
    gateList(i,2) = {gates.ADSIMpInput{1, 2}.Gate{1, i}.BoundingPoint.Attributes.Name};
end

%get distance adjacency matrix, which uses network distance to fill the cell fo connected edges
adjMatrixDist = adjMatrixSpf;
for i = 1:1:size(adjMatrixSpf,1)
    for j = 1:1:size(adjMatrixSpf,2)
        if adjMatrixSpf(i,j)==1;
            if nodeListSpf{i}(1) == 'E' || nodeListSpf{j}(1) == 'E'
                adjMatrixDist(i,j) = 100*distancePoints(nodeSpf(i,:),nodeSpf(j,:)); %default
penalty factor of crossing runway path is 100
            else
                adjMatrixDist(i,j) = distancePoints(nodeSpf(i,:),nodeSpf(j,:));
            end
```

```
        end
    end
end

%switch single direction adjacency matrix to bidirectional adjacency matrix
for i = 1:1:size(adjMatrixSpf,1)
    for j = 1:1:size(adjMatrixSpf,2)
        if adjMatrixSpf(i,j)==1;
            adjMatrixDist(j,i) = adjMatrixDist(i,j);
        end
    end
end
%sparse full adjacency matrix
adjDist = sparse(adjMatrixDist);
taxipath = struct;              %initialize the taxipath structure file
ind2 = 1;
for i = 1:1:size(exitList,1)
    waitbar(i/(size(exitList,1)),h,sprintf('Preparing Index %g out of %g',i,size(exitList,1)))
    for j = 1:size(gateList,1)
        taxipath(ind2).airport = edgeListSpf{1,4};
        taxipath(ind2).runway = exitList{i,1};
        taxipath(ind2).runwayexit = exitList{i,2};
        taxipath(ind2).gate = gateList{j,1};
        taxipath(ind2).taxiwayend = gateList{j,2};
        m = find(strcmp(nodeListSpf,exitList{i,2}));
        n = find(strcmp(nodeListSpf,gateList{j,2}));
        [dist,path] = graphshortestpath(adjDist,m,n);   %find the shortest path, while dist is
the distance of the path is the index of node on the path
        taxipath(ind2).distance = dist;
        taxipath(ind2).path = path';
        ind2 = ind2+1;
    end
end


%%
%write taxipath structure file into taxipaths.xml file
fileName = [ADSIM_dir,'\taxipaths.xml'];
Output_File_ID = fopen(fileName, 'w');
Output_File_ID = fopen(fileName, 'a+');

%% --------------------------------
% Create Header of XML File
% --------------------------------
fprintf(Output_File_ID, '%s\n', '<!DOCTYPE ADSIMpInput>');
fprintf(Output_File_ID, '%s\n', '<ADSIMpInput xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ADSIMpInput.xsd">');
%%
% create taxipaths
for i = 1:1:size(taxipath,2)
    waitbar(i/(size(taxipath,2)),h,sprintf('Preparing Index %g out of %g',i,size(taxipath,2)));
    if length(taxipath(1,i).path)>2
        %print forwards taxipaths
        fprintf(Output_File_ID, '%s\n', '   <TaxiPath IncludeInStudy="true">');
        fprintf(Output_File_ID, '%s\n', ['      <Airport>',taxipath(1,i).airport, '</Airport>']);
        fprintf(Output_File_ID, '%s\n', ['      <Name>txp',taxipath(1,i).runwayexit,'-
',taxipath(1,i).taxiwayend,'F</Name>']);
        fprintf(Output_File_ID, '%s\n', '       <Direction>F</Direction>');
        fprintf(Output_File_ID, '%s\n', ['      <StartUnit Unit="',taxipath(1,i).runway,'"
EndPoint="',taxipath(1,i).runwayexit,'"/>']);
        for j = 2:length(taxipath(1,i).path)
            a = find(ismember(edgeSpf(:,1:2),[taxipath(1,i).path(j-1),
taxipath(1,i).path(j)],'rows'),1);
            if isempty(a)
                a = find(ismember(edgeSpf(:,1:2),[taxipath(1,i).path(j), taxipath(1,i).path(j-
1)],'rows'),1);
            end
            fprintf(Output_File_ID, '%s\n', ['      <PathUnit Unit="',edgeListSpf{a,1},'"
EndPoint="',nodeListSpf{taxipath(1,i).path(j),1},'"/>']);
        end
```

```
        fprintf(Output_File_ID, '%s\n', ['      <EndUnit Unit="',taxipath(1,i).gate,'"
EndPoint="',taxipath(1,i).taxiwayend,'"/>']);
        fprintf(Output_File_ID, '%s\n', '   </TaxiPath>');
        %print backwards taxipaths
        fprintf(Output_File_ID, '%s\n', '   <TaxiPath IncludeInStudy="true">');
        fprintf(Output_File_ID, '%s\n', ['      <Airport>',taxipath(1,i).airport, '</Airport>']);
        fprintf(Output_File_ID, '%s\n', ['      <Name>txp',taxipath(1,i).runwayexit,'-
',taxipath(1,i).taxiwayend,'B</Name>']);
        fprintf(Output_File_ID, '%s\n', '       <Direction>B</Direction>');
        fprintf(Output_File_ID, '%s\n', ['      <StartUnit Unit="',taxipath(1,i).runway,'"
EndPoint="',taxipath(1,i).runwayexit,'"/>']);
        for j = 2:length(taxipath(1,i).path)
            a = find(ismember(edgeSpf(:,1:2),[taxipath(1,i).path(j-1),
taxipath(1,i).path(j)],'rows'),1);
            if isempty(a)
                a = find(ismember(edgeSpf(:,1:2),[taxipath(1,i).path(j), taxipath(1,i).path(j-
1)],'rows'),1);
            end
            fprintf(Output_File_ID, '%s\n', ['      <PathUnit Unit="',edgeListSpf{a,1},'"
EndPoint="',nodeListSpf{taxipath(1,i).path(j),1},'"/>']);
        end
        fprintf(Output_File_ID, '%s\n', ['      <EndUnit Unit="',taxipath(1,i).gate,'"
EndPoint="',taxipath(1,i).taxiwayend,'"/>']);
        fprintf(Output_File_ID, '%s\n', '   </TaxiPath>');
    end
end
%%
% Create footer of XML File
fprintf(Output_File_ID, '%s\n', '</ADSIMpInput>');
fclose all;
delete(h)
msgbox('ADSIM+ Input File Creation Completed','Success')
end

%% ----- Subfunction parseChildNodes -----
function [children,ptext] = parseChildNodes(theNode)
    % Recurse over node children.
    children = struct;
    ptext = [];
    if theNode.hasChildNodes
        childNodes = theNode.getChildNodes;
        numChildNodes = childNodes.getLength;

        for count = 1:numChildNodes
            theChild = childNodes.item(count-1);
            [text,name,attr,childs] = getNodeData(theChild);

            if (~strcmp(name,'#text') && ~strcmp(name,'#comment'))
                %XML allows the same elements to be defined multiple times,
                %put each in a different cell
                if (isfield(children,name))
                    if (~iscell(children.(name)))
                        %put existsing element into cell format
                        children.(name) = {children.(name)};
                    end
                    index = length(children.(name))+1;
                    %add new element
                    children.(name){index} = childs;
                    if(~isempty(text))
                        children.(name){index}.('Text') = text;
                    end
                    if(~isempty(attr))
                        children.(name){index}.('Attributes') = attr;
                    end
                else
                    %add previously unkown new element to the structure
                    children.(name) = childs;
                    if(~isempty(text))
                        children.(name).('Text') = text;
                    end
                    if(~isempty(attr))
```

```
                        children.(name).('Attributes') = attr;
                    end
                end
            elseif (strcmp(name,'#text'))
                %this is the text in an element (i.e. the parentNode)
                if (~isempty(regexprep(text,'[\s]*','')))
                    ptext = text;
                end
            end
        end
    end
end

% ----- Subfunction getNodeData -----
function [text,name,attr,childs] = getNodeData(theNode)
    % Create structure of node info.

    %make sure name is allowed as structure name
    name = regexprep(char(theNode.getNodeName),'[-:.]','_');

    attr = parseAttributes(theNode);
    if (isempty(fieldnames(attr)))
        attr = [];
    end

    %parse child nodes
    [childs,text] = parseChildNodes(theNode);

    if (isempty(fieldnames(childs)))
        %get the data of any childless nodes
        try
            %faster then if any(strcmp(methods(theNode), 'getData'))
            text = char(theNode.getData);
        catch
            %no data
        end
    end

end

% ----- Subfunction parseAttributes -----
function attributes = parseAttributes(theNode)
    % Create attributes structure.

    attributes = struct;
    if theNode.hasAttributes
       theAttributes = theNode.getAttributes;
       numAttributes = theAttributes.getLength;

       for count = 1:numAttributes
          attrib = theAttributes.item(count-1);
          attr_name = regexprep(char(attrib.getName),'[-:.]','_');
          attributes.(attr_name) = char(attrib.getValue);
       end
    end
end
```