Scenario-Driven Development and Testing of ATC Conflict Detection

Kevin Richard[1], Shafagh Jafer[2] and Mohammad Moallemi[3], Krishan Patel[4]
Embry-Riddle Aeronautical University, Daytona Beach, FL, 32114-3900

Neal C. Thigpen[5]
Federal Aviation Administration (FAA) Academy, Oklahoma City, OK, 73169

Conflict detection in Air Traffic Control (ATC) is inherently a complex issue to solve, as there are many factors that go into determining whether two (or more) aircraft are going to collide, or at the very least, violate their minimum separation requirements. In the most basic form, there are four dimensions to be concerned with: the three dimensions of position, and time. Tracking the aircraft's heading, turn rate, airspeed, vertical speed, and any accelerations or decelerations are all vital to more accurately predict the trajectory of each aircraft. Furthermore, the planned route versus the aircraft's actual path needs to be considered, as deviations from the planned route will almost certainly occur. These deviations add uncertainty to the predictability of a conflict.

The Federal Aviation Administration (FAA) Academy has tasked the team with implementing an ATC simulator that can run various scenarios for their ATC training program. Conflict detection is the core algorithm underlying such a simulator and is the focus of this paper. There are three different levels of conflict detection that are typically performed; they consist of: long range, mid-range, and short-range. In this paper methods of calculating conflict detection at multiple levels will be discussed with a focus on long range and mid-range.

Aside from the aircraft attributes and intent, other factors need to be accounted for in the calculation of the trajectories. Aircraft, especially those on an Instrument Flight Rule (IFR) flight plan, fly over a series of waypoints that form their route. These waypoints can have specific airspeed and altitude limitations associated with them. Additionally, it is in the interest of an Air Traffic Controller to have conflict detection limited to only pointing out conflicts within a specific area. As such geofencing concepts will also play a part. To implement and test detection algorithms, scenario-driven development and testing is used. Selective categories of scenarios will be dissected to identify key factors that must be considered for the design and implementation of such algorithms.

In this paper a scenario-driven approach will be proposed to implement such algorithms and a discussion on how tradeoffs are accomplished between response time and precision of conflict detection will be conducted. Sample simulation runs will also be included, testing selective ATC scenarios.

I. Air Traffic Control Systems

In order to adequately prepare controllers for their tasks at their facilities, simulators are utilized for training in the ATC domain. This section will talk about the following ATC tracks and associated simulation technologies: (1)

[1] Research Assistant, Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach FL 32114-3900, richa084@my.erau.edu
[2] Assistant Professor, Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach FL 32114-3900, jafers@erau.edu
[3] Research Associate, Next Generation Applied Research Lab., Embry-Riddle Aeronautical University, Daytona Beach FL 32114-3900, moallemm@erau.edu
[4] Research Assistant, Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach FL 32114-3900, patelk22@my.erau.edu
[5] Enroute Course Coordinator, Air Traffic Division, AMA-511C Staff Office, Training Branch, Enroute Section Advanced Enroute Operations Unit, Mike Monroney Aeronautical Center, 6500 South MacArthur Blvd. Oklahoma City, OK 73169, neal.thigpen@faa.gov

Terminal, which includes Tower and TRACON (2) En Route. Fig. 1 depicts the usage of each track during the various stages of an aircraft's flight.

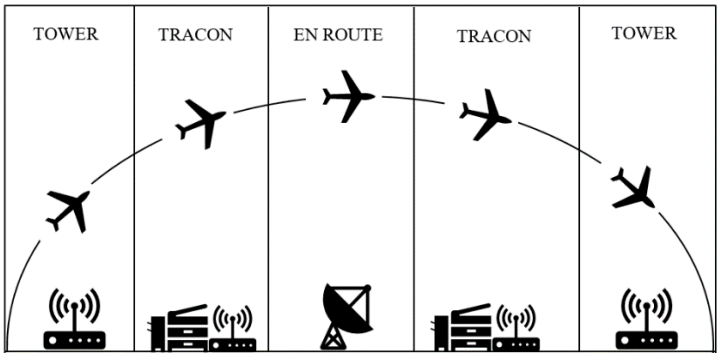

Fig. 1 ATC facilities and responsibilities

A. Tower

The Tower track manages aircraft that are maneuvering within a few miles of the airport at which the ATC is located. The major responsibilities of tower controllers include providing take-off and taxiing instructions, as well as clearance deliveries. [1] Simulators are used to replicate real ATC towers and train students. [2] The Tower simulator relies heavily on line-of-sight with all ground operations of an airport and the active aircraft in the vicinity, therefore the simulators for this track are highly complex. Relatively simple types of tower simulators consist of 4 big TV screens with a 3D model of the airport, a set of computer screens with radar, and other equipment. The larger ones are full-size 360-degree simulators where the view from the tower windows is created by an array of projectors or large screens. [3]

The scenarios associated with Tower operations include the control of aircraft on the ground, departure and arrival clearances, and airport operations. The visual display of the airport is essential for a Tower simulator as flight clearances are issued only after a visual inspection of the appropriate runway. Aircraft are managed by the TRACON controllers directly after take-off and before arrival.

B. TRACON

The training for TRACON controllers is carried out during the Terminal Basic Radar Training Course, which includes classroom training and simulation. These controllers primarily focus on managing aircraft that are just outside of an airport's Terminal control zone and extends up to 40 miles from the arrival/departure airport. Major responsibilities of specialists in this track include ascending departing aircraft, descending arriving aircraft, maintaining separation distances, and transferring control to En Route Center controllers or Tower controllers. [1] Since the control area is much larger than that controlled by a Terminal controller and TRACON control isn't associated with any ground operations, visual access to the airport is not required making the simulators simpler. Training is done in regular radar labs and the scenarios revolve around arriving aircraft, departing aircraft, and hand off communication with En Route and Tower controllers.

C. En Route

The En Route course requires the most detailed training and is only provided at the FAA Academy. The region managed by a single sector of an Air Route Traffic Control Center (ARTCC) is much larger than that of a Tower or TRACON controller. [3] This course consists of classroom instruction, medium-fidelity simulation for practice utilizing interactive computer-based instructional systems, and full fidelity En Route Automation Modernization (ERAM) simulation in an En Route lab. [1]

The ERAM system requires coordination between two controllers managing the same space and looking at two different monitors: (1) the Radar-Position (R-Position) which contains an interface with a radar display, and (2) the Radar Associate-Position (RA-Position) which has an interface to the ERAM Decision Support Tool (EDST). [3]

The R-Position interface shows all aircraft in the vicinity of a specific sector as blocks of data that contain information about the flight such as the aircraft's ID, speed, altitude, etc. Fig. 2 shows the graphical representation of the R-Position interface with a number of flights and their associated data blocks.

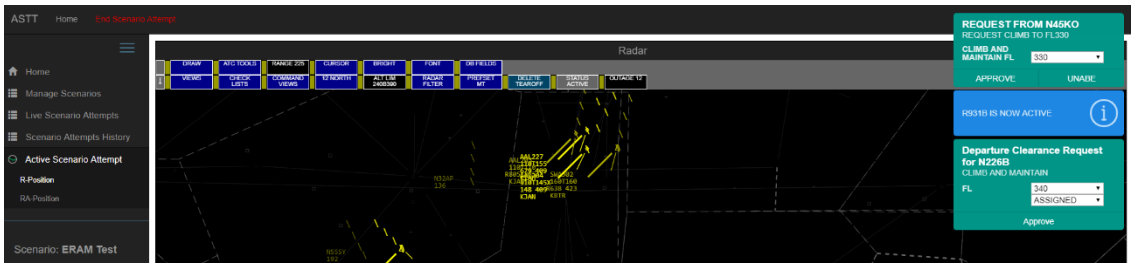

Fig. 2 View of ERAM R-Position interface

The RA-Position interface allows the toggling of various other windows that each contain different information depending on what the radar associate would like to view. The type of information that can be retrieved ranges from a list of all flights that are shown in the R-Position interface with a table of their altitude, direction, route, etc. to a Graphical Plans Display. Other views enable the radar associate to plan a modification to an aircraft's trajectory, amend heading, and receive notifications of potential issues such as conflict alerts. It also gives the controller the option to edit any of the fields or to trial plan a change in altitude or route. [5] An image of this interface screen can be seen in Fig. 3.

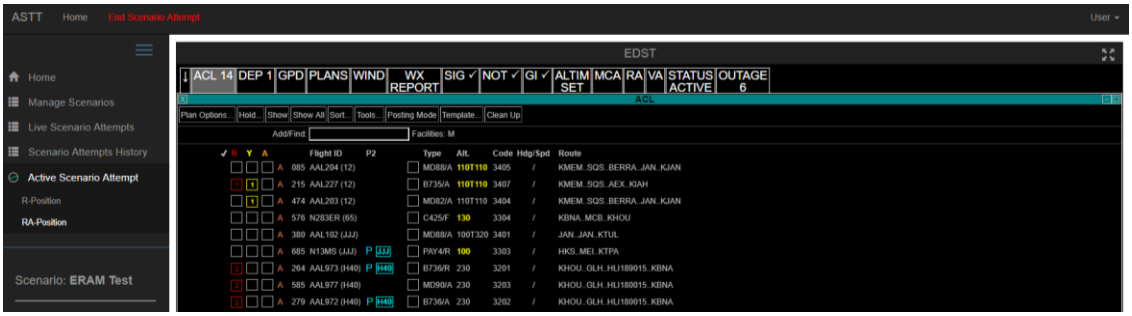

Fig. 3 View of ERAM RA-Position interface

II. ATC Conflict Detection

As previously discussed there are three different levels of conflict detection that are typically performed; they consist of: long range, (daily – for flight plans and airline schedules); mid-range (multiple attempts per hour – for modifications to flight plans based on actual flight information); and short-range (every few seconds – carried out at both the ATC level and onboard aircraft that are equipped with Collision Avoidance System (CAS)).

A. Long Range

Long-range looks over the entire route (from departure airport to arrival airport) to see if the aircraft's planned route will cause any issues with other traffic.

B. Medium Range

Medium-range occurs multiple times per hour and looks for conflicts up to a specific point in the future, typically 20 minutes. Detection occurs en route, so the first element in the route should be the current location of the aircraft, with only the future waypoints listed afterwards.

III. Conflict Detection Implementation

The section describes the methodology that was followed whilst designing the algorithms used for Conflict Detection (CD). Section A focuses on aspects related to the trajectory creation. Section B focuses on the aspects related to GeoFencing. Section C focuses on the aspects related to Conflict Detection.

A. Trajectory Creation

The process of computing conflict detection is split into two parts, the first of which is building the trajectories for each of the flights. Trajectories are simply just the paths that the aircraft will be flying as they travel between the departure and arrival airports.

The first step involved with the trajectory creation is segmenting the route and focusing on its horizontal profile. Initially, each segment makes up a span of the flight from one waypoint to the next. These segments will be further divided in future steps.

Segments contain enough detail to indicate the initial position, orientation, and the relative time that the segment starts. They also include the details to transition to the next segment, including distance, airspeed and vertical speed. Additionally, a segment contains a Boolean to indicate the outcome of the GeoFencing computation, which will be discussed in a Section B.

Each segment gets instantiated with the waypoint that makes up the starting point of that segment. All the segment instances get saved to a Linked List. This is important because it not only allows the segments to remain in a specific order, it also allows additional segments to be added at specific points in the list. The segments will get further divided in the upcoming steps and this allows everything to stay organized.

With the initial segments built the next step is to determine which part of the route passes through a specific two-dimensional area, which will be introduced later as an Area Element. This essentially defines a polygon using Waypoints as vertices and allows conflict detection to be limited to a specific area. The innerworkings of this feature will be discussed in a future section, but for now, each segment is sent off to find out if it resides inside of the specified area. There are three possible results. The segment could fall entirely inside the area, it could fall entirely outside the area, or it could partially pass through the area.

The segment will need to be split into two at the first intersection along the segment. A new segment is created with its starting waypoint at the intersection point previously determined. This new segment gets inserted in the Linked List immediately after the segment that it was just split from. A check is then done on the initial piece of the original segment to determine if it is now fully in the area, or fully out of the area.

Future steps are going to require knowing the distances (and bearings) between each waypoint, so the next logical step is to calculate those. The Haversine Formula [6] is used to calculate the distances. The bearing is found with a separate formula [6]. Remember that this bearing is just the initial bearing to fly at the start of the segment. When flying a great-circle path, the heading will vary.

With the distances and bearings complete, the vertical component of the trajectory can now be completed. Similar to the area, segments most likely will require getting split into two at the Top-of-Climb (TOC) and Top-of-Descent (TOD) points, as required. For this pass, altitudes are calculated, and inputting airspeeds and vertical speeds based on the aircraft's performance numbers.

The vertical component of the trajectory is split into two passes. The first one covers getting the aircraft from its initial altitude to its target altitude. The second one deals with descending from cruise into the destination airport. With the appropriate performance values assigned, the horizontal distance it would take to reach the target altitude must be found. First, find the difference in altitude between the current and targets. Then use the aircraft performance vertical speed (in feet per minute) to calculate how long (in minutes) it would take to travel that vertical distance. Finally, based on that time, use the aircraft performance airspeed to calculate how much ground would be covered.

The distance until the level-off point is now known; but where is that point, and where does it fall in the list of segments? These will be accomplished while the altitudes, airspeeds, and vertical speeds are assigned to each segment. Starting from the beginning, each segment is assigned the appropriate vertical speed and airspeed. Altitudes at the start of the segment are also determined, except for the first segment where the altitude has already been assigned. This is done by starting with the previous segment's starting altitude, and then using that previous segment's vertical speed (and determined duration) to calculate the new altitude's offset.

As it progresses through the segments of the route, the distances are summed into an accumulated distance. If the accumulated distance plus the distance of the current leg exceeds the distance to level-off, the segment where the level-off happens has been found. This segment needs to be split into two, like before.

The length of the current segment is reset to a new value, that being the difference between the total distance to the level-off point, and the accumulated distance, not counting the current segment. A new segment now needs to be inserted in the list, which starts at the level-off position, and at the target altitude. The starting position of the new segment is calculated by another formula [6], related to Haversine. Given a starting point, a bearing (in radians), and a distance, the destination point can be calculated.

Now that the new segment has been found, this pass aborts, and goes through a similar process to calculate the TOD position, if the final waypoint is an airport. The main difference with the TOD calculation and the one just covered, is that the segments are iterated through backwards.

With the vertical profile of the trajectory calculated, the final step is to calculate the times. Using the same logic as before, the duration for each leg is calculated by using the leg's distance and airspeed in the common physics formula: *Time = Distance / Velocity*. Time is important for conflict detection because you need to know where the aircraft are in relation to each other at a specific point in time. With all these steps done, the aircraft trajectory is complete.

B. GeoFencing

As previously explained in the trajectory creation, one of the abilities of the conflict detection feature is to limit the detection to a specific area. Each segment of an airplane's route is analyzed to determine whether or not it is inside a specific area. Segments that are mixed, are split up into distinct segments that either are or are not inside the defined boundary area. The implementation on how this is achieved is now discussed. Determining whether a segment is inside an area was implemented by answering three questions:

- Is the start of the segment inside the area?
- Is the end of the segment inside the area?
- Are there any intersections the segment makes with the area?

First is the discussion on whether a geographic location is in an area. This is fundamentally a geometry problem, centered around the concept of inclusion. The Winding Number [7] algorithm was chosen to solve this problem. It works by counting the number of times a polygon "winds" around a point. If the number is zero, then the point is outside; otherwise, its inside.

The Winding Number method works by taking each segment of the area and comparing the direction of the segment with the relative position of the point. If the segment starts below, crosses upward, and the point is left of the segment, increment a counter. If the segment starts above, crosses downward, and the point is right of the segment, decrement the counter. Other segments, including horizontal segments are ignored, as it wouldn't change the answer.

Now, to find out if the point is left or right of the segment, another calculation needs to happen. This is accomplished with a vector cross-product operation [7]. The sign of P tells whether point p3 lies to the left or to the right of a line with points p1, p2. A positive value indicates the point is left of the line; a value of zero would indicate it's on the line; and a negative value indicates the right side.

In addition to the inclusion problem for the segment endpoints, intersections also need to be checked. Finding intersections between two segments is done yet again through the use of cross-product operations.

The first cross-product determines the angle between the two segments. This will be used in further equations as the denominator for fractions. However, if the angle between the two is found to be zero, it aborts right there, as that means the lines are parallel and won't intersect. Two more cross-products are calculated, for two unique numerators for fractions. The two numerators are divided by the previously mentioned denominator, and the resulting values indicate at one point along the two segments the intersection happens at. If either of the values indicate less than zero, or greater than one, the intersection happens outside of the boundaries of the segments, so there is no actual intersection. If both values are within the limits, those values are used to interpolate between the appropriate segment endpoints, to find the actual position of the intersection.

C. Conflict Detection

Aside from the trajectories, the Conflict Detection functionality requires a few more pieces of information. This information includes a look-ahead time, the minimum-allowable horizontal separation (in nautical miles), the minimum-allowable vertical separation (in feet), and the previously-discussed Area Element to define the area for GeoFencing…although this element is optional.

The look-ahead time defines a time limit as to how long into the future the conflict detection should be running. A common time is 20 minutes for medium-range, but if you wanted to check the entire route, then you could not include a specific time. Minimum horizontal and vertical separations define what would trigger a conflict. Both values would have to be less than their minimums to trigger the conflict. For the en route environment, five and twelve nautical miles are common, horizontally, and 1000 feet vertically.

Depending on the scenario, the lists of trajectories will get broken up into unique pairs so two flights can be compared against each other at a time.

With the trajectories created, the next step involves finding the span of time where the flights overlap. For the start time of CD, the later of two options is used: the earliest time both flights are in the air, and the time the first plane enters the area. For the stop time of CD, the first of the two options is used: the latest time both flights are in the air,

and the time the last plane exists the area. One last check that wasn't mentioned was that the end time cannot be after the look-ahead time. So, the earlier of the two values is used.

Now that the timespan to check through is known, the next step is to find the segment start times that fall within that span for both flights. These are referred to as pertinent times. The start time, end time, and all the additional pertinent times that were just found are joined into a single list, sorted from earliest to latest, with redundant times removed.

For the conflict detection to work, linear vectors of each segment of the trajectory need to be created. However, both vectors need to start at the same time, so additional sub-segments will need to be created, since each flight will most-likely not start their segments at the same exact time.

The algorithm [8] used to calculate the conflict detection is referred to as Closest Point of Approach (CPA). CPA considers two moving targets, each with their own initial positions, and linear velocity vectors, and finds the point where they are closest together. This algorithm works with vectors, and 3D coordinates, but X, Y, and Z; not lat/long and altitude. As such, position and velocity vectors will need to be created in the cartesian coordinate system.

With the vectors created, the CPA times can now be calculated. CPA works off dot-product operations [8]. It first checks if the vectors are parallel, since if they were, the time wouldn't matter (we used a time of zero). Otherwise, it essentially tells you how many instances it takes the vectors to get their closest. Since the vectors were converted to a minute scale, the value it calculates represents the number of minutes after the start of the segment that the CPA would occur at.

With the time of the CPA found, the positions of the planes can be found at that time, to see whether there is a conflict. The CPA calculation does not know anything about the limitations of the segments. The time returned could be less than zero, or after the end of the segment. Therefore, the time needs to be capped to within those boundaries.

The altitudes are the first to be calculated, as those are simpler, and potentially more efficient. Using the same interpolation techniques as before, the altitudes at the CPA are found, and compared to see if they are within the vertical separation requirements. If they are too close vertically, it continues. Otherwise, it moves on to the next segments.

The horizontal positions of the aircraft are also found by the same techniques previously mentioned. The distance the planes travel is found based on the time and their speed. Then, knowing the segment's starting point, their bearing, and the distance traveled, their position at the CPA can be found. The horizontal distance can now be found between the two planes. If the planes are not within the minimum horizontal separation specified, a conflict has been found.

IV. Conflict Detection Testing Scenarios

This section describes the sample scenarios that were used to verify the accuracy of the process. For each scenario, an overview of the events, expectations, and actual results are described. After that, there is a conceptual drawing of the routes of the aircraft, and where the conflict is supposed to occur. After that is a table of the scenario input data. The two cells in the first row of the table describe the trajectory skeletons for both aircraft involved. The second two describes the Area Element boundaries, and the conflict specifications. It finally indicates the result of the conflict, which indicates the position (lat/long) of the two aircraft, and the distance between their two positions.

A. Scenario 1

The first scenario that was created was a simple verification that features were working. There were two flights, starting at the same time, going the same speed, and flying at the same altitude, on a 90-degree converging angle. This was a blatant attempt to force a conflict. After the conflict point, the routes continued to random fixes/waypoints. Coordinates around the 0° N, 0° E area were used so that the creation of the cartesian vectors could be easily checked over.

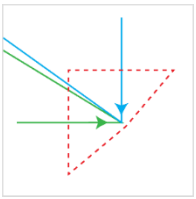

Figure 4 – Concept for Scenario 1

Flight 1: Route: 00.00.00N/02.00.00W → 00.00.00N/00.00.00W → CLT Starting Altitude: 33000 feet Target Altitude 33000 feet Cruise Speed: 340 knots Start Time: 01:05:00 Performance Numbers: MMO: 0.84 Climb Norm: 2000 Descent Best: 1500	**Flight 2:** Route: 02.00.00N/00.00.00W → 00.00.00N/00.00.00W → HBG Starting Altitude: 33000 feet Target Altitude: 33000 feet Cruise Speed: 340 knots Start Time: 01:05:00 Performance Numbers: MMO: 0.84 Climb Norm: 2000 Descent Best: 1500
Area: Points: 01.00.00S/01.00.00W → 01.00.00N/01.00.00W → 01.00.00N/01.00.00E → 00.00.01S/00.00.01E	Look-Ahead Time: 24 hours Minimum Horizontal Separation: 5 nm Minimum Vertical Separation: 1000 feet

CONFLICT FOUND:
Position 1: $6.12292781247509x10^{-17}$ | $7.65391954641678x10^{-07}$ → 00.00.00N/00.00.00E
Position 2: $-7.65391954641678x10^{-07}$ | 1.22458556249502x10-16 → 00.00.00N/00.00.00E
Distance Between: 6.49882288063301x10-05 nm → 0 nm

B. Scenario 2

This scenario is based on an actual scenario that end-users would be interacting with. The guidance was such that a conflict with less than 5 nm of separation should be triggered. It did get triggered. No GeoFencing was implemented in this scenario.

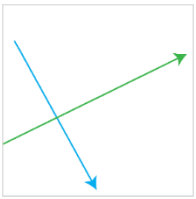

Figure 5 – Concept for Scenario 2

Flight 1: Route: 31.53.35N/092.22.49W → IGB → KEWR Starting Altitude: 35000 feet Target Altitude 23000 feet Cruise Speed: 325 knots Start Time: 01:03:00 Performance Numbers: MMO: 0.85 Climb Norm: 2500 Descent Best: 3500	**Flight 2:** Route: 33.41.19N/091.49.49W → MON --> MCB → KMOB Starting Altitude: 37000 feet Target Altitude: 23000 feet Cruise Speed: 325 knots Start Time: 01:02:00 Performance Numbers: MMO: 0.85 Climb Norm: 2500 Descent Best: 3500
Area: None	Look-Ahead Time: 24 hours Minimum Horizontal Separation: 5 nm Minimum Vertical Separation: 1000 feet

CONFLICT FOUND:
Position 1: 32.5020305160726 | -90.9513639855641
Position 2: 32.4352917567446 | -90.9776809910168
Distance Between: 4.223 nm

C. Scenario 3

This is another scenario that could be faced by end-users. This CD was described as having a yellow alert, which means the conflict point should be greater than 5nm, but no more than 12. One aircraft should pass in front of the other's path. A conflict was found, but it ended up claiming that the conflict was less than 5nm. The culprit is most likely inaccurate performance numbers being used in the scenario, so one or both are going too fast or too slow, which is skewing the location where the conflict happens. The best course of action would be to compare the 4D trajectory created from this CD feature with the 4D log of the aircraft flight simulator being used to drive the aircraft in these scenarios and find what is causing the discrepancy.

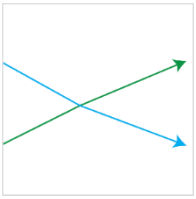

Figure 6 – Concept for Scenario 3

Flight 1: Callsign: SWA552 Route: 31.37.14N/091.42.00W → JAN → CLT Starting Altitude: 23000 feet Target Altitude 23000 feet Cruise Speed: 340 knots Start Time: 01:01:00 Performance Numbers: MMO: 0.82 Climb Norm: 2000 Descent Best: 3500	**Flight 2:** Callsign: SWA840 Route: 331016N/0915811W → JAN --> TLH Starting Altitude: 35000 feet Target Altitude: 23000 feet Cruise Speed: 340 knots Start Time: 01:00:00 Performance Numbers: MMO: 0.82 Climb Norm: 2000 Descent Norm: 3500
Area: None	Look-Ahead Time: 24 hours Minimum Horizontal Separation: 12 nm Minimum Vertical Separation: 1000 feet

CONFLICT FOUND:
Position 1: 32.5059107285791| -90.1852137853859
Position 2: 32.5049589441487| -90.1601316766267
Distance Between: 1.271 nm

V. Discussion and Future Work

This paper addressed a solution to the complexity of the Conflict Detection problem as there are many factors that can influence whether an aircraft is on a collision course with another. To tackle this problem three separate solutions were discussed and presented; those being trajectory creation, GeoFencing through polygon inclusion, and Closest Point of Approach (CPA) analysis. Trajectories were created in piecemeal, consisting of distinct linear segments that approximate the routes taken by the aircraft. Segments were split at points where aircraft accelerations (airspeed changes, heading changes, and vertical speed changes) happened. Segments were also split at points where the aircraft entered and exited a defined area of interest, so conflict detection could optionally only operate in specific areas.

Further work in this area can pertain to efficiency improvements that could be implemented on GeoFencing methods being used. For example, before even calculating the inclusion of a point, that point should be checked to see if it falls within the rectangular boundary box of the area with simple comparisons. Additionally, the endpoints of neighboring segments are needlessly getting checked twice and this process could also be optimized. A quicker solution would be to take the midpoint of the line segment of the route and see if that one point is in the area. However, this only works if there are no intersections. For the scope of this paper, the solutions provided are sufficient to resolve the problem statement of reducing the complexity of Conflict Detection in Air Traffic Control.

References

[1] J. Updegrove and S. Jafer, "Recommendations for Next Generation Air Traffic Control Training," in *Digital Avionics Systems Conference (DASC), 2017 IEEE/AIAA 36th*, St. Petersburg, FL, 2017.

[2] Federal Aviation Administration, "Review and Evaluation of Air Traffic Controller Training at the FAA Academy," 2013.

[3] B. Chhaya, S. Jafer, W. B. Coyne, N. C. Thigpen and U. Durak, "Enhancing Scenario-Centric Air Traffic Control Training," in *AIAA Modeling and Simulation Technologies Conference*, Kissimmee, FL, 2018.

[4] Federal Aviation Administration, "EAA121L0 En Route Automation Modernization (ERAM) Air Traffic Manual (ATM): R-Position User Manual," 2009.

[5] Federal Aviation Administration, "EAC11000 En Route Automation Modernization (ERAM) Air Traffic Manual (ATM): RA-Position User Manual," 2012.

[6] C. Veness, "Calculate Distance and bearing between Two Latitude/Longitude Points Using Haversine Formula in JavaScript," [Online]. Available: www.movable-type.co.uk/scripts/latlong.html. [Accessed March 2018].

[7] D. Sunday, "Inclusion of a Point in a Polygon," 2012. [Online]. Available: geomalgorithms.com/a03-_inclusion.html. [Accessed March 2018].

[8] D. Sunday, "Distance Between 3D Lines and Segments," 2012. [Online]. Available: geomalgorithms.com/a07-_distance.html. [Accessed March 2018].