NASA/TM–2013-218055

A Turn-Projected State-Based Conflict Resolution Algorithm

Ricky W. Butler and Timothy A. Lewis
Langley Research Center, Hampton, Virginia

November 2013

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 443-757-5803

- Phone the NASA STI Information Desk at 443-757-5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TM–2013-218055

A Turn-Projected State-Based Conflict Resolution Algorithm

Ricky W. Butler and Timothy A. Lewis
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

November 2013

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Abstract

State-based conflict detection and resolution (CD&R) algorithms detect conflicts and resolve them on the basis on current state information without the use of additional intent information from aircraft flight plans. Therefore, the prediction of the trajectory of aircraft is based solely upon the position and velocity vectors of the traffic aircraft. Most CD&R algorithms project the traffic state using only the current state vectors. However, the past state vectors can be used to make a better prediction of the future trajectory of the traffic aircraft. This paper explores the idea of using past state vectors to detect traffic turns and resolve conflicts caused by these turns using a non-linear projection of the traffic state. A new algorithm based on this idea is presented and validated using a fast-time simulator developed for this study.

Contents

1 Introduction

Conflict detection and resolution (CD&R) algorithms for air traffic separation assurance can be divided into *intent*-based and *state*-based categories. Intent-based algorithms operate on information about an aircraft's future intended trajectory, i.e., upcoming turns and altitude changes within a certain time horizon. CD&R systems that rely on intent information typically perform strategic functions, recommending longer-term route changes to avoid traffic as well as maintaining an optimal flight plan to the destination. However, due to the nature of the heuristic optimization approach taken by many of these systems, it can be extremely difficult to establish that a strategic, intent-based system will always produce a solution to any conflict.

In response, state-based systems are often proposed as a backup to strategic solvers. This approach is motivated by the philosophy that a simplified, highly-reliable, formally-verified CD&R layer operating only on the state vector information can provide separation assurance during short-term encounters in which an intent-based system cannot resolve a particular conflict.

CD&R algorithms based upon the current state perform extremely well where the velocity vectors of aircraft are constant or nearly so. However, in the presence of a turning aircraft or an aircraft that is leveling-out after an altitude change, the performance of these algorithms can be very poor. This presents a conundrum: if a state-based system is to be the backup for an intent-based system, it must perform flawlessly in the presence of maneuvering traffic, even though in the absence of intent information it is impossible to know the intruder's future trajectory with any certainty.

This paper is motivated by the results of a human-in-the-loop simulation experiment conducted to address this issue and presents a new state-based conflict resolution algorithm designed to provide much improved performance in response to conflict encounters with maneuvering traffic.

2 Background

Throughout this paper, *ownship* will refer to the focus aircraft running a given instance of the CD&R algorithm, and *traffic* or *intruder* will refer to an aircraft of interest in the vicinity of the ownship.

To investigate the issue of state-based algorithm performance in the presence of maneuvering traffic, the Separation Allocations in Shared Airspace (SALSA) experiment was conducted at the NASA Langley Air Traffic Operations Laboratory [6]. In this experiment, airline pilot crews were presented with a battery of conflict encounters to evaluate the performance of a research prototype airborne separation assurance system. This prototype, which features both strategic, intent-based functions as well as a state-based safety layer, is incorporated into a desktop flight simulator flown by pilots in the study.

In particular, the simulation focused on events known as *pop-up* conflicts. In a pop-up conflict, an aircraft that is not broadcasting intent information begins a turn or altitude change that is unknown to the ownship beforehand. The trajectory

change creates a conflict with little warning; it "pops-up" on the pilot's display with a varying amount of time until loss of separation (LoS). The experiment focused on the pilot's ability to detect and resolve these conflicts while maintaining separation using the provided CD&R tools.

An example pop-up is illustrated in Figure 1. In this example, the ownship and

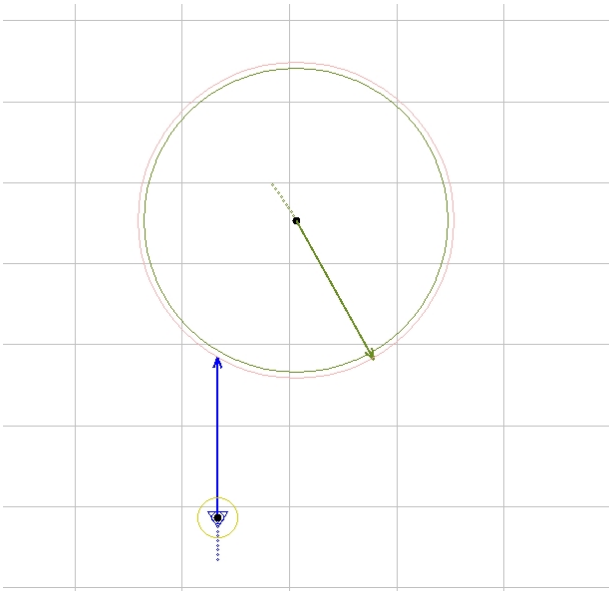

Figure 1: Pop-up Conflict Created by Turning Intruder

intruder are initially 12 nm apart, each traveling at a ground speed of 550 kn, and are not in conflict. The intruder begins a right turn (with a bank angle of 25° in accordance with the model described in section 6.1) without warning the ownship. Nine seconds later, the aircraft have closed to 10 nm and the ownship first detects the conflict. At this point the ownship initiates a horizontal resolution maneuver (with same bank angle) but there is insufficient time to complete it before LoS occurs. This LoS is illustrated in Figure 2.

Note that in this case, the ownship initiated the resolution maneuver as soon as it detected the conflict. However, in a practical scenario, some amount of pilot delay can be expected. Figure 3 shows how quickly the time to LoS decreases if the pilot hesitates in the execution of the turn resolution maneuver. In a dynamic situation like this, the actual time to loss of separation can be much shorter than an initial state-based estimate.

To minimize the impact of a turning intruder and provide additional reaction time to the ownship, the concept of an avoidance buffer can be used. In this approach, the ownship algorithm uses a larger separation standard for the traffic aircraft; e.g. detecting and resolving conflicts to a lateral distance of 8 nm rather than the standard separation standard of 5 nm. Entrance into this zone is not a separation violation, although the ownship's pilot will attempt to stay out of the zone if

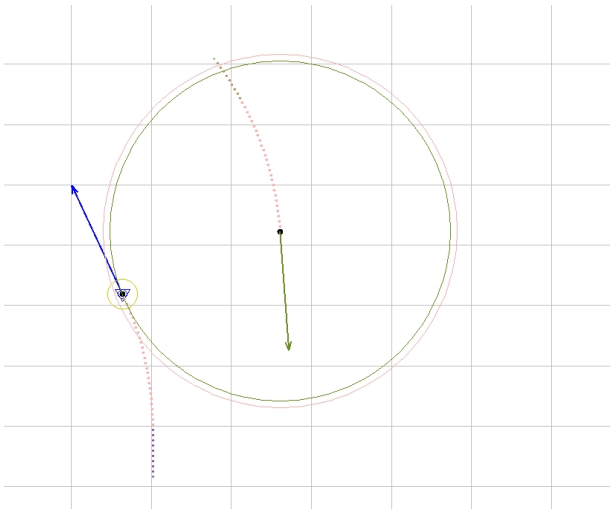

Figure 2: Failed Conflict Resolution and Loss of Separation Resulting from Conflict

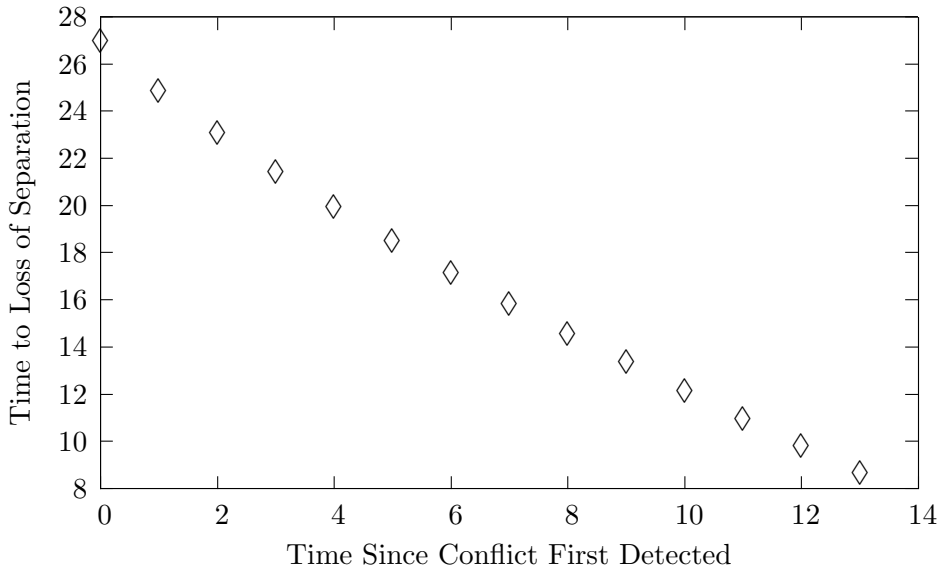

Figure 3: Time to Loss of Separation in the Presence of a Turn

possible.

Much of our previous work on state-based CD&R algorithms has been concerned with the problem of implicit coordination using a concept known as the criteria approach [2, 4, 5]. A mathematical proof has established that any algorithm that satisfies the implicit cordination criteria will inherit two key safety properties: that separation will be maintained when i) only one aircraft maneuvers, and ii) when both aircraft maneuver at the same time. An important consequence is that there

is no requirement that both aircraft execute the same resolution algorithm, just that both algorithms satisfy the same criteria. Therefore, the criteria approach allows different avionics vendors to implement different algorithms, each optimized according to their own proprietary concerns. This approach, however, can break down in the presence of non-compliant intruders.

The state-based algorithm used in the SALSA experiment was the `resolutionKinematic` method of Chorus [1]. This algorithm was designed to support the Autonomous Flight Rules (AFR) concept of operation [7] using implicit coordination. However, in this experiment, the traffic aircraft were flying under standard Instrument Flight Rules (IFR), and not conforming to the mathematical implicit coordination criteria. The fact that an IFR aircraft could turn in any direction and not be restricted to following the criteria violated a central assumption of the Chorus algorithm. Losses of separation were observed that could be partially attributed to this contradiction.

Kuchar and Yang [3] provided a well-cited taxonomy for classifying a vast assortment of state-based and other CD&R algorithms in literature. However, experience in the SALSA experiment highlighted several critical characteristics of state-based CD&R systems not captured by this taxonomy, including:

- Continuity of the solutions over time: through experience with human pilot test subjects, we have learned that it is highly desirable that the resolution maneuvers provided by any CD&R system evolve smoothly during turns and altitude changes, rather than jumping discontinuously between alternate solutions.

- Transition to/from strategic solvers: state-based algorithms must work in concert with other CD&R layers, especially those providing strategic, intent-based functions. Providing smooth transitions between these algorithm layers when each is producing potentially incompatible solutions based on different information and assumptions is a nontrivial problem.

- Initiation of the algorithm as a consequence of a changing velocity vector: Development of state-based CD&R algorithms has traditionally focused on aircraft whose state vectors are not substantially changing over the detection and resolution cycle. However, in practical operations we have found the need to quickly respond to conflicts arising with an intruder aircraft that is in the middle of a turn.

In this paper we develop a new algorithm, named `resolutionTurnProj`, that is designed to better respond to an intruder aircraft turning into the ownship in encounters with a short time remaining until loss of separation. We have also developed a fast-time simulator to test this algorithm over its entire input range with a fairly small grid size that produced 100,000s of test cases. In this fast time simulator, we have not sought to imitate the exact trajectory generated by a particular aircraft's Flight Management System (FMS). Instead we have used a standard model that produces circular turns. The advantage of this approach is that we can stress test the algorithm and the results are not limited to a particular aircraft type.

3 Notation

This paper uses pseudocode to document the `resolutionTurnProj` algorithm rather than its Java or C++ implementation. This pseudocode is a hybrid language incorporating features of both functional and object-oriented languages. The purpose of the pseudocode is to capture the major logic of the algorithms without the distracting details of a normal programming language.

The implementation employs standard mathematical vectors: `Vect2` for a 2-dimensional vector and `Vect3` for a 3-dimensional vector. In addition, basic vector arithmetic operations are assumed. For example: for Vect3 `u`, `v`, `w` and double `a`, the vector assignment `v = u+w*a` is defined by:

```
v.x = u.x + w.x * a
v.y = u.y + w.y * a
v.z = u.z + w.z * a
```

The pseudocode assumes the existence of an n-ary tuple data type, denoted by `(x1,..,xn)`, which allow piecewise assignment,

```
(x1,...,xn) = (y1,...,yn)
```

thus assigning `x1` the value `y1`, etc. In the Java and C++ source code, these operations are often implemented as separate instance variables using appropriate accessor functions, and sometimes may be set as side effects as opposed to being returned explicitly in a function.

The definitions of the operators (`+`, `-`, `*`, `/`, `==`, `!=`, etc.) are the definitions from Java or C++ without relying on exotic behavior such as overflow. We use `AND` and `OR` for the boolean connectives rather than `&&` and `||` used in Java and C++. The operator `~=` is defined to mean "almost equals," which means that the values are compared and if they are within a fixed floating point precision of each other, then they are considered to be equal.

The for loop statement syntax is taken from Java and C++:

```
for (initialization; test; increment) { ... }
```

Our notation for function definitions departs from Java/C++ syntax in that the return type of the function follows the parameters. For example, a function that computes the sine function would be declared as follows:

```
sin(x): double  { ... }
```

Note that the types of the function parameters are not listed. We also allow an n-tuple return type:

```
func(x,y,z):  (a,b)  { ... }
```

Here the function `func` has three parameters and returns two values with types `a` and `b`.

4 Common Variables and Functions

The following variables are commonly used in the Chorus algorithms:

`so`	initial ownship position ($\mathbf{s_o}$)
`vo`	initial ownship velocity ($\mathbf{v_o}$)
`si`	initial intruder position ($\mathbf{s_i}$)
`vi`	initial intruder velocity ($\mathbf{v_i}$)
`s`	initial relative position ($\mathbf{s}$)
`v`	initial relative velocity ($\mathbf{v}$)

We use a prime notation to represent updated values of variables, e.g. `s'` is a newly calculated value of `s`.

The algorithms use a vector library which provides the ability to algebraically manipulate vectors. The following are common functions:

`v.x`	x component of vector $\mathbf{v}$
`v.y`	y component of vector $\mathbf{v}$
`v.z`	z component of vector $\mathbf{v}$
`vo.trk()`	returns track angle component of velocity $\mathbf{v_o}$
`vo.gs()`	returns ground speed component (2-D vector magnitude) of velocity $\mathbf{v_o}$
`vo.vs()`	returns vertical speed component of velocity $\mathbf{v_o}$
`vo.mkTrk(trk)`	creates a 3-dimensional velocity vector from $\mathbf{v_o}$, where the track component is changed to `trk`
`vo.mkGs(gs)`	creates a 3-dimensional velocity vector from $\mathbf{v_o}$, where the ground speed component is changed to `gs`
`vo.mkVs(vs)`	creates a 3-dimensional velocity vector from $\mathbf{v_o}$, where the vertical speed component is changed to `vs`
`mkTrkGsVs(t,g,v)`	creates a 3-dimensional velocity vector from track, ground speed and vertical speed components
`v.vect2()`	returns a 2-dimensional vector from the first two components of a three-dimensional vector

Additionally, the following function is used to define loss of separation between two aircraft:

`LoS(s,D,H)`	returns `TRUE` if the relative position `s` is in loss of separation with respect to a protection zone with horizontal separation D and vertical separation H. This is defined as: `s.x*s.x + s.y*s.y < D*D AND |s.z| < H`

The following are parameters that are assumed to be globally accessible and therefore these variables are not explicitly passed in the pseudocode:

`D`	minimum horizontal separation
`H`	minimum vertical separation
`Tres`	lookahead time for resolution

5 The Fast Time Dir2Sim Simulator

To study this problem and evaluate the performance of the `resolutionTurnProj` software, we developed a fast-time simulator that generates hundreds of thousands of pairwise encounters, each featuring an intruder who executes an unbroadcasted turn into the ownship. The simulator places the ownship at the center of a Euclidean frame and positions the intruder on a uniform grid of points relative to the ownship.

5.1 Simulation Approach and Parameters

In the Dir2Sim simulator, the buffer approach is implemented by constraining the initial location of each intruder to be outside of circle of radius `bufferSize` around the ownship as illustrated in Figure 4. The dimensions of the grid were -90 nm to 90

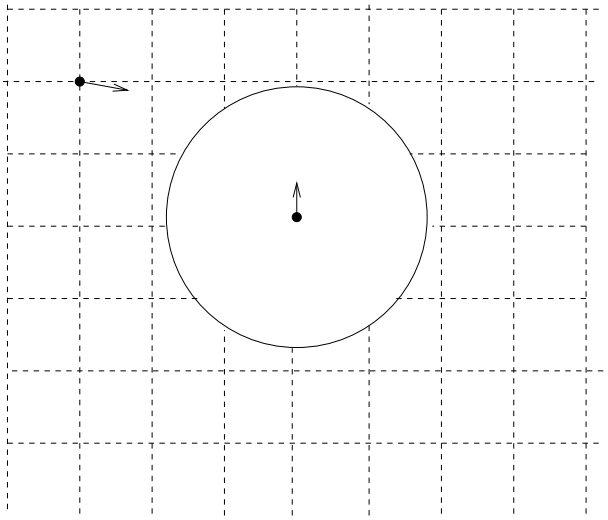

Figure 4: Initial conditions for a given ownship-intruder pair.

nm for the x dimension and -30 nm to 90 nm for the y-coordinate. The increment in both dimensions was 5 nm. Cases where the aircraft were in conflict at time 0 were discarded. Also, cases where the turn was completed without a conflict ever occuring were also discarded.

The initial velocity vector of the ownship and intruder are

```
vo = mkTrkGsVs(ownInitTrack,gso,0);
vi = mkTrkGsVs(trki0, gsi, 0);
```

where the variables `ownInitTrack`, `gso` , `trki0` and `gsi` are varied over a range of values described in Table 1. The time step for the simulation was 1 second.

In each simulation run, the intruder begins an immediate turn with a constant bank angle (in all cases in this paper, 25°) and constant ground speed. The bank angle is deliberately large to stress the algorithm. The turn continues until a specified track angle, up to 45° left or right of the initial track, is reached. The ownship takes

variable	units	starting value	ending value	step size
`ownInitTrack`	deg	0	0	0
`gso`	kn	350	550	50
`trki0`	deg	0	360	5
`gsi`	kn	350	550	50

Table 1: Dir2Sim Velocity Parameters

no action until it first detects the conflict, at which point it exectues a resolution maneuver provided by the algorithm. Results using solutions provided by the original `resolutionKinematic` algorithm and the newly-developed `resolutionTurnProj` algorithm are presented in the subsequent sections.

5.2 Baseline Results: Using the Chorus `resolutionKinematic` Track Algorithm

To provide a baseline for comparison, `Dir2Sim` was executed using the resolutions from the original Chorus `resolutionKinematic` solver. These results are shown in Table 2.

`bufferSize` [nm]	# Conflict Cases	# LoS cases	# no Solution
6	61065	11369 (18.62%)	518 (0.85%)
7	57545	9923 (17.24%)	511 (0.89%)
8	53990	8467 (15.68%)	503 (0.93%)
9	50880	7205 (14.16%)	494 (0.97%)
10	48103	6143 (12.77%)	486 (1.01%)
11	44873	4892 (10.90%)	476 (1.06%)
12	41979	3809 (9.07%)	470 (1.12%)
13	39404	3014 (7.65%)	462 (1.17%)
14	37040	2307 (6.23%)	455 (1.23%)

Table 2: The Dir2Sim Results Using `resolutionKinematic`

As the buffer size increases, the number of conflict cases decreases. This is a consequence of the fact that the `Dir2Sim` intruder position grid is of constant size, and that as the buffer gets larger, there are fewer valid grid points for intruder aircraft. However, the specific number of conflicts, LoS cases, etc., is less important than the percentage trends (shown in parentheses).

As the buffer size increases, the percentage of conflict cases which result in LoS decreases. This shows the beneficial effect of the buffer: since we restrict intruders to be initially outside the buffer, then larger buffers provide extra reaction time and "breathing room" for CD&R. Doubling the buffer from 6 nm to 12 nm cuts the LoS percentage in half.

A converse effect can be seen in the percentage of conflicts in which the algorithm produces no solution. As the buffer gets larger and consumes more airspace, it becomes slightly more difficult for the algorithm to return any solution at all. However, this effect is overcome by the drastic reduction in LoS percentage, resulting in a net benefit for larger buffers in this context.

The percentage of LoS cases is very high in these results; even with a significant buffer (e.g. 12 nm), the LoS rate is unacceptably high (9%). This performance is most certainly due to the fact the Chorus `resolutionKinematic` solver is constrained to produce resolutions in only one direction, under the assumption that the intruder is also following the same implicit coordination criteria. In these simulations, this is not true.

These simulations, as well as practical experience with `resolutionKinematic` in SALSA, motivate the development of an improved resolution algorithm.

6 An Intrusive-Turn Resilient Algorithm

In this section we present a new algorithm that uses recent state vector history from the intruder to infer its rate of turn.

The algorithm predicts the future trajectory of the intruder assuming it continues this turn. Without accurate intent information, the algorithm does not know when the intruder will complete its turn. However, it can make an heuristic assumption that a given intruder during cruise flight is not likely to make a turn greater than $\pi/4$ rad or $45°$.

The algorithm iteratively explores the result of the ownship turning left or right with a specified maximum bank angle. It continues the iterative search in both directions until it finds a solution in which the ownship is conflict free for multiple successive time steps into the future. In the case when both directions result in conflict free velocities, the algorithm uses a simple heuristic to pick between the two.

The algorithm is provided the following input values:

`Vect3 so` — initial position of the ownship
`Vect3 vo` — initial velocity of the ownship
`Vect3 si` — initial position of the intruder
`Vect3 vi` — initial velocity of the intruder

It is also assumed that a recent history of past velocity vectors of the intruder have been stored to enable a calculation of the turn rate of the intruder, denoted `trackRate`. The ownship aircraft is assumed to be able to turn at a turn rate of `omega` when executing the resolution maneuver.

6.1 Kinematic Subfunctions

The algorithm finds solutions based on a simple kinematic turn model. Before we present the algorithm, we will present some of the kinematic subfunctions used in the algorithm.

We begin we a simple linear projection in time:

```
linear(s, v, t): (Vect3, Vect3) = (s + t*v, v)
```

This function returns a pair which contains the new position and the velocity.

Next, we look at a kinematic turn with radius R.

$$R = \frac{v^2}{g \tan \phi}$$

where R is the turn radius of an aircraft turning with bank angle ϕ, and v is the ground speed of the aircraft. Using the relationship $v = R|\omega|$, where ω is the turn rate of angular velocity of the aircraft, we obtain

$$\omega = \frac{g \tan \phi}{v}$$

We assume that the aircraft turns with this constant angular velocity ω (i.e. $\frac{d\theta}{dt} = \omega$). This will result in time based velocity vector of $\mathbf{v}(t) = (v_x(t), v_y(t))$, where

$$\begin{aligned} v_x(t) &= v \sin(\theta + \omega t) \\ v_y(t) &= v \cos(\theta + \omega t) \end{aligned} \tag{1}$$

Integrating, we obtain position

$$\begin{aligned} s_x(t) &= s_x(0) - \epsilon R[\cos\theta - \cos(\theta + \omega t)] \\ s_y(t) &= s_y(0) + \epsilon R[\sin\theta - \sin(\theta + \omega t)] \end{aligned} \tag{2}$$

This is directly coded into the function `turnOmega` which takes the following parameters:

`Vect3 s0`	starting position
`Vect3 v0`	initial velocity
`t`	time of turn
`omega`	rate of change of track, sign indicates direction

The `turnOmega` function is defined:

```
turnOmega(s0, v0, t, omega): (Vect3, Vect3) {
   if (omega ~= 0) return linear(s0,v0,t)
   v = v0.gs();
   theta = v0.trk();
   xT = s0.x + (v/omega)*(cos(theta) - cos(omega*t+theta));
   yT = s0.y - (v/omega)*(sin(theta) - sin(omega*t+theta));
   zT = s0.z + v0.z*t;
   s' = (xT,yT,zT);
   v' = v0.mkTrk(v0.track()+omega*t)
   return (s',v');
}
```

The function `turnTime` computes the time it takes to complete a turn given a ground speed, the magnitude of the track change and the maximum bank angle:

```
turnTime(groundSpeed, deltaTrack, bankAngle): double {
        omega = turnRate(groundSpeed, bankAngle);
        if (omega == 0.0) return MAX_VALUE;
        return |deltaTrack/omega|;
}

turnRate(speed, bank): double {
        if (bank ~= 0.0) return 0.0;
        return g*tan(bank)/speed;
}
```

Another useful turn function which turns for a specified time `turnTime` and then goes straight:

```
turnUntilTime(so, vo, t, turnTime, omega): (Vect3, Vect3) {
   if (t <= turnTime) {
      return turnOmega(so, vo, t, omega);
   } else {
      (nso,nvo) = turnOmega(so, vo, turnTime, omega);
      return linear(nso, nvo, t-turnTime);
   }
}
```

6.2 Track Angle Search Component

The algorithm searches both directions iteratively using the search component. This component is provided the following parameters

`so`	Initial ownship position
`vo`	Initial ownship velocity
`si`	Initial intruder position
`vi`	Initial intruder velocity
`remainingTime`	The estimated time remaining before the intruder's turn will be complete
`trkDir`	search direction (-1 = left, +1 = right)
`trackRate`	The estimated turn rate of the intruder
`numIterConfFree`	algorithm parameter that governs how many conflict free iterations are needed

It returns a triple of values:

`nDetKin`	-1 if no conflict was found, 1 if a conflict is found, 2 if there was a loss of separation with a secondary aircraft, 3 if there is a loss of separation with the most urgent aircraft
`trkDelta`	the change in track angle that first elicits a conflict
`nvTrk`	the final conflict-free track when `nDetkin` = -1. Otherwise, it is the last track analyzed.

The following Java-like pseudo code describes the search component of the algorithm:

```
confTrkSearch(so, vo, si, vi, remainingTime,
              trkDir, trackRate, numIterConfFree): (int, double, double)  {
   omega = turnRate(vo.gs(),trkDir*maxBankAngle);
   for (trkDelta = 0.0; trkDelta < PI; trkDelta = trkDelta + step) {
      tm = trkDir*trkDelta/omega;
      nvTrk = vo.track()+trkDir*trkDelta;
      (soAtTm, vo') = turnOmega(so, vo, tm, omega);
      if (|trackRate| > trackRateThreshold) {
          (siAtTm, vi') = turnUntilTime(si, vi , tm, remainingTime, trackRate);
      else
          (siAtTm, vi') = linear(si0,vi0,tm);
      Vect3 s = soAtTm - siAtTm;
      if (LoS(s,Dr,Hr)) {
         nDetKin = 3;         // Los with primary
      } else if (CDSS.conflict(s, vo', vi', Dr, Hr, Tres)) {
          nDetKin = 1;
          confFreeIterations = 0;
      } else {
          nDetKin = -1; // conflict free
      }
      if (nDetKin >= 2) break;
      if (nDetKin < 0) {
          confFreeIterations++;
          if (confFreeIterations >= numIterConfFree) break;
      }
   }
return (nDetKin,trkDelta,nvTrk);
```

If the traffic aircraft's rate of turn (i.e., `trackRate`) is greater than a specified threshold (i.e., `trackRateThreshold`), then the traffic aircraft's future trajectory is calculated using a kinematic turn function. If the track rate does not exceed the threshold a linear trajectory is used. Note that the algorithm projects the intruder turn for a maximum of $\pi/4$ rad (45°). The search continues until there are `confFreeIteration` successive, conflict-free iterations. This was necessary because it is possible for the aircraft state to be conflict free for a while and then subsequently revert back to conflict status in the presence of a turning aircraft. Furthermore, if the ownship accepts and executes the resolution before the intruders turn is complete, the resolution can be inadequate. This is illustrated in Figure 5. In this figure, the intruder's turn has not yet completed so the computed resolution of 266.5° is not adequate. Even if the execution of the resolution maneuver begins immediately,

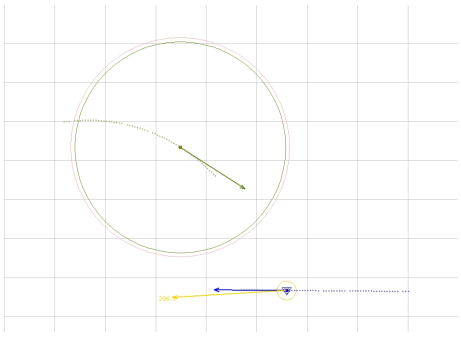

Figure 5: Resolution Too Early

the conflict reappears a few seconds later as shown in Figure 6, and a loss of sep-

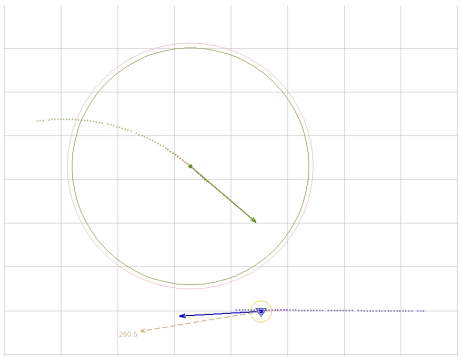

Figure 6: Resolution Too Early: Subsequent Conflict State

aration occurs soon thereafter as shown in Figure 7. The parameters `Dr` and `Hr`

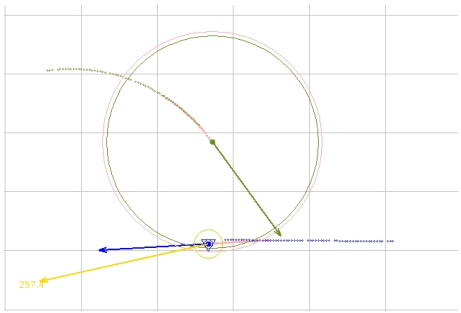

Figure 7: Resolution Too Early: Subsequent LoS State

are the protection zone parameters (e.g. `D` and `H`) with a small buffer added for resolution. Typical valyes are: `Dr` = 5.2 nm and `Hr` = 1100 ft. The `CD3D.LoS` and `CDSS.conflict` algorithms are presented in the appendix A.

The main body of the algorithm first makes an estimate of the turn time of the intruder based on a computed `trackRate`. The `trackRate` is computed using a running average of the three most recent velocity vectors. If this track rate is greater than a specified threshold `trackRateThreshold`, then the intruder is assumed to be turning.

```
remainingTime = 0.0;
if (|trackRate| > trackRateThreshold) {
   maxTurnTime = | (PI/4)/trackRate |;
   remainingTime = maxTurnTime;
   if (useRemainingTime) {
      remainingTime = estimateRemainingTime();
   }
} else {
   numIterConfFree = 3;
}
```

and then calls the search function twice:

```
int trkDir = -1;
(nDetKinLeft, ... )  = confTrkSearch(cdc, so, vo, si, vi,
                          remainingTime, trkDir, trackRate, numIterConfFree);
trkDir = 1;
(nDetKinRight, ... ) = confTrkSearch(cdc, so, vo, si, vi,
                          remainingTime, trkDir, trackRate, numIterConfFree);
nvTrk = 0;
```

The boolean variable `useRemainingTime` is a configuration parameter that controls whether the algorithm should try to more accurately predict the end of the intruder's turn. This option is explored in section 6.4.

6.3 Choosing Between the Two Potential Solutions

After searching in both directions, the algorithm must choose the appropriate direction (left or right) to return for the resolution. When only one direction is conflict free, the decision is simple. However, when the search finds that both directions are conflict free, the choice becomes more subtle. While it may at first seem reasonable to simply flip a coin and pick either direction with equal probability, experience shows that the two directions can produce very different results. Several different approaches were pursued to choose the solution when both search directions appear to be conflict-free. We present two of the methods that have been studied: (1) use the turn that is coordinated with the ownship's criteria, or (2) use the turn that is consistent with the intruder's observed behavior, even if it is contrary to the implicit criteria.

Using the ownship's criteria the following results were obtained:

bufferSize	Conflict Cases	LoS cases	no Solution
6.0	61065	963 (1.58%)	101 (0.17%)
7.0	57545	681 (1.18%)	58 (0.10%)
8.0	53990	473 (0.88%)	31 (0.06%)
9.0	50880	351 (0.69%)	15 (0.03%)
10.0	48103	255 (0.53%)	5 (0.01%)
11.0	44873	207 (0.46%)	1 (0.00%)
12.0	41979	167 (0.40%)	0 (0.00%)
13.0	39404	141 (0.36%)	0 (0.00%)
14.0	37040	115 (0.31%)	0 (0.00%)

Results using the new algorithm are significantly improved over the previous algorithm's results presented in Table 2. For example, the LoS percentage for a 12 nm buffer was reduced from 9% to 0.4%. The cause of this significant improvement is mostly due to the fact that the new algorithm searches both directions rather than just the one given by the criteria.

Using the intruder's turn behavior to make the left-right decision, we obtain:

bufferSize	Conflict Cases	LoS cases	no Solution
6.0	61065	2093 (3.43%)	101 (0.17%)
7.0	57545	1775 (3.08%)	58 (0.10%)
8.0	53990	1511 (2.80%)	31 (0.06%)
9.0	50880	1305 (2.56%)	15 (0.03%)
10.0	48103	1099 (2.28%)	5 (0.01%)
11.0	44873	896 (2.00%)	1 (0.00%)
12.0	41979	712 (1.70%)	0 (0.00%)
13.0	39404	566 (1.44%)	0 (0.00%)
14.0	37040	419 (1.13%)	0 (0.00%)

Surprisingly, using the ownship's mathematical criteria obtains slightly better results than using the intruder criteria. It should be noted that for most scenarios, only one search direction has a conflict-free solution, so this selection is infrequently made.

6.4 Improvement Using the Time When the Intruder Turn Began

The success of the bi-directional search algorithm depends upon a reasonable estimate of the turn characteristics of the intruder. Unfortunately, the duration of the intruder's turn is not known. However, it is reasonable to assume that the turn will not be greater than some maximum value, typically 45°. The time required to reach this maximum value, `maxTurnTime`, is easily computed. At the time of the conflict, the magnitude of the intruder's turn that has already taken place can be subtracted from the maximum turn to obtain a better prediction of the intruder's trajectory. This is accomplished by the subfunction `estimateRemainingTime`:

```
estimateRemainingTime(maxTurnTime): double {
    tmTurnStarted = lastStraightTime();
    turnTimeSoFar = (timeLast() - tmTurnStarted);
```

```
        remainingTime = maxTurnTime  -  turnTimeSoFar;
        if (remainingTime < 0) remainingTime = 0;
        return remainingTime;
    }
```

The function `lastStraightTime()` returns that last time when intruder aircraft was not turning and the function `timeLast()` returns the intruder aircraft's latest timestamp (i.e. the current time). Using this strategy, the results were improved for all values of the initial separation buffer, `bufferSize` (see figure 4):

`bufferSize`	Conflict Cases	LoS cases	no Solution
6.0	61065	233 (0.38%)	48 (0.08%)
7.0	57545	58 (0.10%)	13 (0.02%)
8.0	53990	20 (0.04%)	6 (0.01%)
9.0	50880	9 (0.02%)	2 (0.00%)
10.0	48103	1 (0.00%)	1 (0.00%)
11.0	44873	0 (0.00%)	0 (0.00%)
12.0	41979	0 (0.00%)	0 (0.00%)
13.0	39404	0 (0.00%)	0 (0.00%)
14.0	37040	0 (0.00%)	0 (0.00%)

Note that for a bufferSize of 8 nm, the LoS cases were reduced from 0.88% to 0.04% (using ownship criteria).

7 Exploring The Use of `turnDetector`

State-based conflict detection typically has relied upon linear projection of the intruder trajectories. It is also possible to use turn projection for conflict detection, as well as for resolution.

7.1 The `turnDetector` algorithm

The `turnDetector` algorithm retrieves position and velocity vectors from a list of intruder information named `aircraftList`. It is provided a parameter `deltaTime` that specifies how far into the future the conflict detection should be made in the presence of the projected turn. This should be a relatively small time, e.g. 6 seconds, to prevent an unacceptably-high false alarm rate. The parameter `T` is the lookahead time for the conflict probe, typically 300 seconds. The `turnDetector` algorithm is defined as follows:

```
turnDetector(deltaTime, T, trackRateThreshold): int  {
   int acIxEarliestIn = -1;
   double earliestTmIn = MAX_VALUE;
   for (int j = 0; j < aircraftList.size(); j++) {
      (so', vo') = linear(so,vo,deltaTime);
      (si, vi)    = predLinear(aircraftList[j])
      if (CD3D.LoS(so-si), D, H)) tmIn = 0;
```

```
        double trackRate = avgTrackRate(NumPtsTrackrate);
        if (|trackRate| > trackRateThreshold ) {
            (si', vi') = turnOmega(si, vi, deltaTime, trackRate);
        } else {
           (si', vi') = linear(si,vi,deltaTime);
        }
        ns = so' - si';
        conf = CDSS.conflict(ns, vo', vi', D, H, T-deltaTime);
        if (conf) tmIn = CDSS.timeIntoLoS(ns, vo', vi', D, H);
        else tmIn = -1;
        if (tmIn >= 0 AND tmIn < earliestTmIn) {
           earliestTmIn = tmIn;
           acIxEarliestIn = j;
        }
     }
     return acIxEarliestIn;
  }
```

The function `turnDetector` returns the index of the aircraft with the smallest time to loss of separation. The function `predLinear(ac)` estimates the position and velocity vectors for the intruder `ac` that corresponds to the latest data of the ownship, using a linear extrapolation from the aircraft's last position and velocity. The function `avgTrackRate` computes an average rate of change of a intruder's track angle using the most recent data values. If this rate exceeds `trackRateThreshold`, then the traffic aircraft is determined to be turning and its trajectory is continued in accordance with this track rate. Otherwise, the traffic aircraft's state is projected using a linear function. The `CDSS.conflict` function performs the conflict detection and it is defined in Appendix A. If there is a conflict with any aircraft the `nDetector` function returns `TRUE`, otherwise it returns `FALSE`. The time parameter, `T`, indicates how far in the future conflicts should be checked.

7.2 `turnDetector` Results

The `turnDetector` algorithm can be used to get an early indication of a conflict. In the `Dir2Sim` simulator this detection function was used to obtain an earlier detection of a turn. Surprisingly, this was not found to improve the overall performance of the turn projection resolution algorithm.

Using a value of 6 for the `deltaTime` parameter of `turnDetector`, the following results were obtained:

bufferSize	Conflict Cases	LoS cases	no Solution
6.0	66019	155 (0.23%)	76 (0.12%)
7.0	62558	72 (0.12%)	23 (0.04%)
8.0	59013	43 (0.07%)	11 (0.02%)
9.0	55907	29 (0.05%)	7 (0.01%)
10.0	53118	18 (0.03%)	3 (0.01%)
11.0	49837	7 (0.01%)	0 (0.00%)
12.0	46851	4 (0.01%)	0 (0.00%)
13.0	44208	2 (0.00%)	0 (0.00%)
14.0	41757	1 (0.00%)	0 (0.00%)

For a `bufferSize` value of 8 nm, the LoS percentage increased from 0.04% to 0.07%. This suggests that this type of capability should be used primarily as a yellow alert, to get pilots prepared for action. Future work will explore modifications to this approach, to see if the LoS rate can be improved.

8 Conclusions

In this paper a new algorithm for resolving conflicts in the presence of a turning aircraft is presented and experimentally analyzed. This work was motivated by recent human-in-the-loop experiments where short-term pop-up conflicts found deficiencies in the existing state-based algorithms. The new algorithm detects turning aircraft and performs a non-linear projection of their future state. The future trajectory is generated as a kinematic turn with the same turn rate that has been recently observed. The new algorithm was designed with several internal parameters that were varied in a fast time simulator to find appropriate values for them. The results from these simulations are presented in detail.

References

1. Ricky Butler, George Hagen, and Jeffrey Maddalon. The Chorus conflict and loss of separation resolution algorithms. Technical Memorandum NASA/TM-2013-218030, NASA, Langley Research Center, Hampton VA 23681-2199, USA, August 2013.

2. Ricky Butler and César Muñoz. Formally verified practical algorithms for recovery from loss of separation. Technical Memorandum NASA/TM-2009-215726, NASA, Langley Research Center, Hampton VA 23681-2199, USA, June 2009.

3. James K. Kuchar and Lee C. Yang. A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems,*, 1(4):179–189, December 2000.

4. César Muñoz, Ricky Butler, Anthony Narkawicz, Jeffrey Maddalon, and George Hagen. A criteria standard for conflict resolution: A vision for guaranteeing

the safety of self-separation in NextGen. Technical Memorandum NASA/TM-2010-216862, NASA, Langley Research Center, Hampton VA 23681-2199, USA, October 2010.

5. Anthony Narkawicz and César Muñoz. State-based implicit coordination and applications. Technical Publication NASA/TP-2011-217067, NASA, Langley Research Center, Hampton VA 23681-2199, USA, March 2011.

6. David Wing, Thomas Prevot, Timothy Lewis, Lynne Martin, Sally Johnson, Christopher Cabrall, Sean Commo, Jeffrey Homola, Manasi Sheth-Chandra, Joey Mercer, and Susan Morey. Pilot and controller evaluations of separation function allocation in air traffic management. In *Tenth USA/Europe Air Traffic Managmeent Research and Development Seminar*, June 2013.

7. David J. Wing and William Cotton. Autonomous flight rules: A concept for self-separation in U.S. domestic airspace. Technical Report NASA/TP-2011-217174, NASA Langley Research Center, 2011.

Appendix A

The CDSS Conflict Probe

The `CDSS.conflict` function takes the following parameters

`s`	the relative position of the aircraft
`vo`	the ownship's velocity
`vi`	the intruder's velocity
`D`	the minimum horizontal distance
`H`	the minimum vertical distance
`B`	the the lower bound of the lookahead time (B ¿= 0)
`T`	the upper bound of the lookahead time (T ¡ 0 means infinite lookahead time)

It is defined as follows:

```
conflict(s, vo, vi, D, H, B, T): boolean {
  if (T >= 0 AND B >= T) return false;
  Vect2 vo2 = vo.vect2();
  Vect2 vi2 = vi.vect2();
  if (vo.z ~= vi.z) AND |s.z| < H) {
     return CD2D.cd2d(s.vect2(),vo2,vi2,D,B,T);
  }
  vz = vo.z - vi.z;
  m1 = max(-H - sign(vz)*s.z, B*|vz|);
  m2 = T < 0 ? H-sign(vz)*s.z :
               min(H-sign(vz)*s.z,T*|vz|);
  if (!(vo.z ~= vi.z) AND m1 < m2) {
    return CD2D.cd2d(|vz|s.vect2(), vo2, vi2, D*|vz|,m1,m2);
  } else {
    return false;
  }
}
```

where `cd2d` is defined as follows

```
cd2d(s, vo, vi, D, B, T): boolean {
  if (T < 0) {
      v = vo - vi;
      return almost_horizontal_los(s,D) OR Delta(s,v,D) > 0 AND s*v < 0;
  }
  if (B >= T) return false;
  v = vo - vi;
  return almost_horizontal_los(s+Bv,D) OR  omega_vv(s,v,D,B,T) < 0;
```

and `almost_horizontal_los(s,D)` is defined as

```
   almost_horizontal_los(s,D): boolean {
      return !(s*s ~= D*D) AND s*s < D*D;
}
```

and `omega_vv(s,v,D,B,T)` is defined as follows:

```
   omega_vv(s,v,D,B,T): double {
      if (s*s ~= D*D) AND B ~= 0) {
         return s*v;
      } else {
         tau = min(max(B*v*v,-(s*v)),T*v*v);
      }
      return v*v*s*s + (2*tau)*(s*v) + tau*tau - D*D*v*v;
   }
```

and `Delta(s,v,D)` and `det(s,v)` are defined as

```
  Delta(s,v,D)  = D*D*V*V - det(s,v)*det(s,v);

  det(s,v) = s.x*v.y - s.y*v.x
```

Note. In the above formulas, the notation $=$ is used for approximately equal. That is equal within a specified floating point precision.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 11 - 2013	Technical Memorandum	

4. TITLE AND SUBTITLE

A Turn-Projected State-Based Conflict Resolution Algorithm

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Butler, Ricky W.; Lewis, Timothy A.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

411931.02.02.07.13.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L-20327

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/TM-2013-218055

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 61
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT

State-based conflict detection and resolution (CD&R) algorithms detect conflicts and resolve them on the basis on current state information without the use of additional intent information from aircraft flight plans. Therefore, the prediction of the trajectory of aircraft is based solely upon the position and velocity vectors of the traffic aircraft. Most CD&R algorithms project the traffic state using only the current state vectors. However, the past state vectors can be used to make a better prediction of the future trajectory of the traffic aircraft. This paper explores the idea of using past state vectors to detect traffic turns and resolve conflicts caused by these turns using a non-linear projection of the traffic state. A new algorithm based on this idea is presented and validated using a fast-time simulator developed for this study.

15. SUBJECT TERMS

Air Traffic Management; Aircraft Trajectory; Algorithms; Autonomy; Conflict Detection and Resolution

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	28	(443) 757-5802