Federal Aviation Administration

DOT/FAA/AM-15/16
Office of Aerospace Medicine
Washington, DC 20591

The Influence of Looping Next-Generation Radar on General Aviation Pilots' Flight Into Adverse Weather

William R. Knecht
Civil Aerospace Medical Institute
Federal Aviation Administration
Oklahoma City, OK 73125

Eldridge Frazier
Office of Advanced Concepts &
Technology Development
Federal Aviation Administration
Washington, DC 20024

August 2015

Final Report

NOTICE

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/AM-15/16		

4. Title and Subtitle	5. Report Date
The Influence of Looping Next-Generation Radar on General Aviation Pilots' Flight Into Adverse Weather	August 2015
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Knecht WR,[1] Frazier E[2]	

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
[1]FAA Civil Aerospace Medical Institute, P.O. Box 25082 Oklahoma City, OK 73125 [2]FAA Office of Advanced Concepts & Technology Development 800 Independence Ave., SW, Washington, DC 20024	11. Contract or Grant No.

12. Sponsoring Agency name and Address	13. Type of Report and Period Covered
Office of Aerospace Medicine Federal Aviation Administration 800 Independence Ave., S.W. Washington, DC 20591	14. Sponsoring Agency Code

15. Supplemental Notes

Work was accomplished under approved task AHRR521

16. Abstract

Looping Next-Generation Radar (NEXRAD) is currently finding its way into general aviation (GA) aircraft cockpits. Pilots may use this to try to pick their way through convective weather cells, with fatal results. This suggests a need to understand how well can pilots maintain safe clearance from severe weather when using this type of technology.

A looping NEXRAD-type cockpit display was created as a part-task simulation. NEXRAD-type images of various types of simulated adverse convective weather were created using a mathematical model of weather to precisely control the placement and movement of weather cells, and to measure the effect of the *opening and closing of gaps*, as well as weather system *depth*.

Results indicated that weather system depth made no difference for the values tested. In contrast, moving weather appeared to greatly decrease pilots' ability to judge how closely their planned flightpath would approach hazardous weather (measured as *point-of-closest-approach*, PCA). Moreover, it did not seem to matter if weather movement was relatively fast or slow, nor whether gaps were opening or closing. The results indicate judgment of PCA with *any* movement of weather was clearly difficult.

This has considerable implications for in-cockpit use of looping NEXRAD. First, it is not anticipated that first-generation looping NEXRAD will be adequate for safe penetration of gaps between storm systems. Second, while training could be expected to improve performance, training alone may not be sufficient. The display concept itself may require considerable human factors modification and testing to overcome multiple safety issues, not only those revealed in this study but also known additional issues of data latency, weather-movement prediction, and the inherently more unpredictable nature of natural weather itself.

The results suggest that in-cockpit NEXRAD could benefit from a) a zoom feature to allow a bigger picture of the weather hazards, b) a range ring to show how far the aircraft could travel in ½ hour, c) a method of projecting how far hazardous weather might be expected to travel in ½ hour, and d) a planning feature to estimate PCA of intended flightpath to hazardous weather.

17. Key Words	18. Distribution Statement
Aviation Weather, NEXRAD, Weather Flight, Weather in the Cockpit, Weather Avoidance, Convective Weather	Document is available to the public through the Internet: www.faa.gov/go/oamtechreports

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	47	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

ACKNOWLEDGMENTS

This study is sponsored by FAA ANG-C61, Weather Technology in the Cockpit Program, and supported through the FAA NextGen Human Factors Division, ANG-C1. Special thanks go to Andy Mead and Gena Drechsler, AAM-510, for help with data collection, to Peter King, for information about XM cockpit data feed, and, most particularly, to David McClurkin, Chief Flight Instructor, and Kenneth Carson, Program Director, both of the Department of Aviation, University of Oklahoma, without whose assistance this study would have been far more difficult. Finally, we thank Mike Wayda for his thoughtful editing and technical support.

Contents

The Influence of Looping Next-Generation Radar on General Aviation Pilots' Flight Into Adverse Weather

Looping Next-Generation Radar (NEXRAD) is currently finding its way into general aviation (GA) aircraft cockpits, embodied in technologies ranging all the way from subscription aviation weather services such as XM satellite radio, on down to smartphone applications. Pilots may use this technology to try to pick their way through convective weather cells, with fatal results. This suggests a need for research to understand any underlying characteristics of the display that might encourage excessive risk-taking. Most particularly, how good are pilots at determining safe clearance distances from severe weather when using this type of technology? We address this question and how it relates to NEXRAD's potential for increased or decreased safety.

A looping NEXRAD-type cockpit display was created as a low-cost, part-task simulation. Various types of simulated adverse convective weather were modeled, and NEXRAD-type images created. A mathematical model of weather was used to precisely control the placement and movement of weather cells, and to measure the effect of the *opening and closing of gaps*, as well as weather system *depth* (defined as "shallow," for weather systems 19 nm in depth, and "deep," for systems 40 nm in depth).

Results indicated that weather system depth made no difference for these values.

In contrast, moving weather appeared to greatly decrease pilots' ability to judge how closely their planned flight path would approach hazardous weather (point-of-closest-approach, PCA). Moreover, it did not seem to matter if movement was relatively fast or slow, nor whether gaps were opening or closing. The results indicate judgment of PCA with *any* movement of weather was clearly a difficult task.

This has considerable implications for in-cockpit use of looping NEXRAD. First, it is not anticipated that first-generation looping NEXRAD will be adequate for safe penetration of gaps between storm systems. Second, while training could reasonably be expected to improve performance, training alone may not be sufficient. The display concept itself may require considerable human factors modification and testing to overcome multiple safety issues, not only those revealed in this study but also known additional issues of data latency, weather-movement prediction, and the inherently more unpredictable nature of natural weather itself.

The results suggest that in-cockpit NEXRAD could benefit from a) a zoom feature allowing pilots to get a bigger strategic picture of the weather hazards, b) a range ring showing how far their aircraft could travel in ½ hour, c) some method of projecting how far the hazardous weather might be expected to travel in ½ hour, and d) a planning feature estimating PCA of intended flight path to hazardous weather. Some of these features are clearly easier to instantiate than others, and that fact is discussed.

The Influence of Looping Next-Generation Radar on General Aviation Pilots' Flight Into Adverse Weather

INTRODUCTION

Adverse weather remains a persistent challenge for all general aviation (GA) pilots and, therefore, a high priority for the Federal Aviation Administration (FAA). While figures vary, a U.S. Congressional Joint Economic Committee report estimated that domestic commercial weather delays cost the American economy "as much as $41 billion for 2007" (JECMS, 2008, p. 1). Moreover, convective weather is particularly problematic to GA because those aircraft are not equipped for adverse weather conditions, and pilots maybe not be experienced for the weather conditions (Batt & O'Hare, 2005; NTSB, 2005).

Data linking weather (DLW) information is widely seen as a leading contender for promoting efficient flight while minimizing weather-related risk. As such, the FAA Next Generation (NextGen) Air Transportation System Implementation Plan specifically discusses "up-to-date weather and airspace status information delivered directly to the cockpit" as part of its vision (FAA, 2012, p. 8).

Research for this capability is being led by groups such as the FAA's WTIC Program (JPDO, 2010), which has sponsored a variety of research projects such as surveying pilots to identify their preferred weather information products, identifying weather-related accident/incident causation for datalinked aircraft, and investigating probabilistic weather displays (ATSC, 2013).

One area of DLW information involves pilot interpretation and use of color-coded weather-risk displays. For example, Next-Generation Radar (NEXRAD) employs a network of more than 150 National Weather Service radar units to detect, process, combine, and distribute radar reflectivity information that indicates areas of precipitation. The resulting weather images are what we commonly see on television and the Internet, intended to help us visualize complex, dynamic behavior of nature that, otherwise, would be impossible to describe. Figure 1 shows three sequential NEXRAD images.GA pilots are now being offered this capability in the cockpit, for instance via XM satellite radio, and on handheld devices like tablet computers and smartphones.

From a human-factors perspective, NEXRAD display is more than just a "weather display." Its WSR-88D Weather Radar Precipitation Intensity scale and Weather Radar Echo Intensity Legend are effectively *risk-proxy gradients*—graphical representations of relative weather-related risk (FAA, 2013). These gradients contain important perceptual cues that pilots can utilize to make decisions (Wiggins, Azar, & Loveday, 2012).

At issue is whether looping NEXRAD can be used to predict where adverse weather will likely be in the near future. Pilots are using looping NEXRAD for tactical weather avoidance, and we anticipate problems associated with that use (Beringer & Ball, 2004).

For instance, the U.S. National Transportation Safety Board has cautioned pilots that the data presented by NEXRAD are not real-time, and may, in fact, be as old as 15-20 minutes (NTSB, 2012). Research is needed to identify such potential data-latency issues. Work so far in this area has largely been proprietary (e.g., Kelley, Kronfeld, & Rand, 2000).

It is natural to wonder exactly *what the critical stimulus information is* in these risk-proxy gradients and *how well pilots understand and can use it*. Recent WTIC Program-sponsored research shows that GA pilots vary widely in how they perceive, understand, and use *static* weather-risk gradients (Knecht & Frazier, 2015). Since misunderstanding those principles can increase flight risk, we proposed to investigate the issue further by computerizing the static displays used in that study to make them dynamic and more realistic. This resulted in the current follow-on study of looping NEXRAD-type display, with a goal of developing flexibility in its design and capabilities.

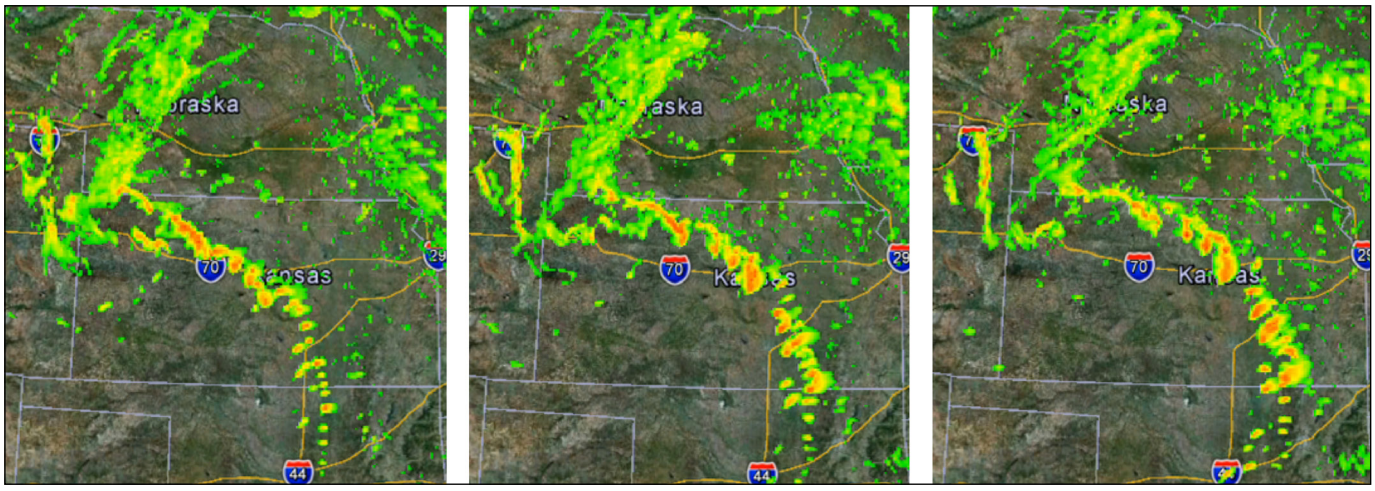


Figure 1. Three looping NEXRAD movie frames, 5 minutes apart.

This research was limited in scope to first address two aspects of looping NEXRAD suspected of modulating risk taking, namely a) depth of the weather system, and, b) gap closure speed. Recall that "depth" was defined as "shallow," if a gap between cells were 19 nm, and could therefore be flown through relatively quickly, and "deep," if the gap were 40 nm.

A mathematical model of weather was first created, capable of generating a series of 12 plausible-looking NEXRAD-type still images (frames) on a laptop computer. The frames were then combined to create dynamic, looping "one-hour movies" of storm-cell activity running at 2 frames per second (fps). We recruited GA pilots from a local Oklahoma City area flight school. These pilots viewed the various NEXRAD movies and then decided how they would fly from a departure point to a destination point during convective weather conditions. Using a standard computer mouse, pilots were able to click on *inflection points*—places onscreen where their planned flightpath would change direction—to draw an overall flightpath. The computer recorded a picture of that flightpath, plus the screen coordinates of all its inflection points, which were then mathematically scored for attributes such as *path length* (a measure of efficiency) and *point of closest approach (PCA)* to hazardous weather (a measure of safety).

It is important to note that these simulated NEXRAD loops ran only up to the critical place and time when the pilot would normally make a "go/no-go" decision either to shoot the gap or to divert around the entire storm system. These were not long, slow scenarios, following the pilot all the way to the "Destination." They were shown rapidly, one after the other, 10 scenarios in about 15 minutes' time. Therefore, each scenario was testing the pilot's "mental snapshot" estimate of future weather position, aircraft position, and weather-clearance distances, taken precisely at that go/no-go point.

The reason for this method was straightforward: If a decision is made to shoot a gap, one is generally committed. There is no turning back, because turning back will usually involve more time and danger than staying the course and getting out of the gap as quickly as possible. Therefore, it makes sense to examine pilot perception and behavior at this critical point in the decision-making timeline.

MATERIALS AND METHODS

Hardware/Software

Hardware consisted of a laptop computer (Dell Latitude D630), with an optical mouse and 12" (30.5 cm width) × 7.5" (19 cm height) screen, running *Mathematica* software (Wolfram, 2013). Pilots self-seated at a comfortable distance, approximately 22" (56 cm), resulting in a total display viewing angle of about 31×19 degrees.

Mathematica provided image-processing of each individual frame (e.g., overlaying a semi-transparent weather image on top of a terrain/road map, so that the map was still visible underneath the weather). *Mathematica* also displayed images with event handling (e.g., reading onscreen mouse clicks), to let participants draw their flightpath on-screen, capture the results, compute the PCA to hazardous weather, and store the results to files readable by *Excel* or SPSS statistical software.

Mathematical Generation of Plausible-Looking Weather

Real weather is chaotic in the strictest technical sense (Gleick, 1987). Where time and cost are no factors, weather can be modeled on a supercomputer to whatever accuracy is necessary. But, for the current research, time and cost were critical. Moderate visual complexity of weather cells was necessary, but not at the cost of excessive computational time. This required a 2D image that loosely resembled NEXRAD but which could be drawn by fast algorithms based on fully controllable parameters.

Specifically, it was important to *independently control at least two weather cells'* a) initial location, b) size, c) shape, d) angle of orientation, e) movement, f) color scheme, and g) evolution (i.e., how each cell changed shape, or morphed, over time). Moreover, it was essential to have a color scheme *contiguously rank-ordered by risk*. In other words, weather cells had to maintain contiguity, meaning the same *order of colors*, even while morphing across the screen. Red always had to sit next to orange, orange next to yellow, and so forth (see Fig. 2).

A number of approaches were initially tried. The one finally chosen began with two independent 2D Gaussian probability density functions, the product *f(x)g(y)* of which forms a 3D structure (Fig. 2a). The height (*z*) of this 3D structure could

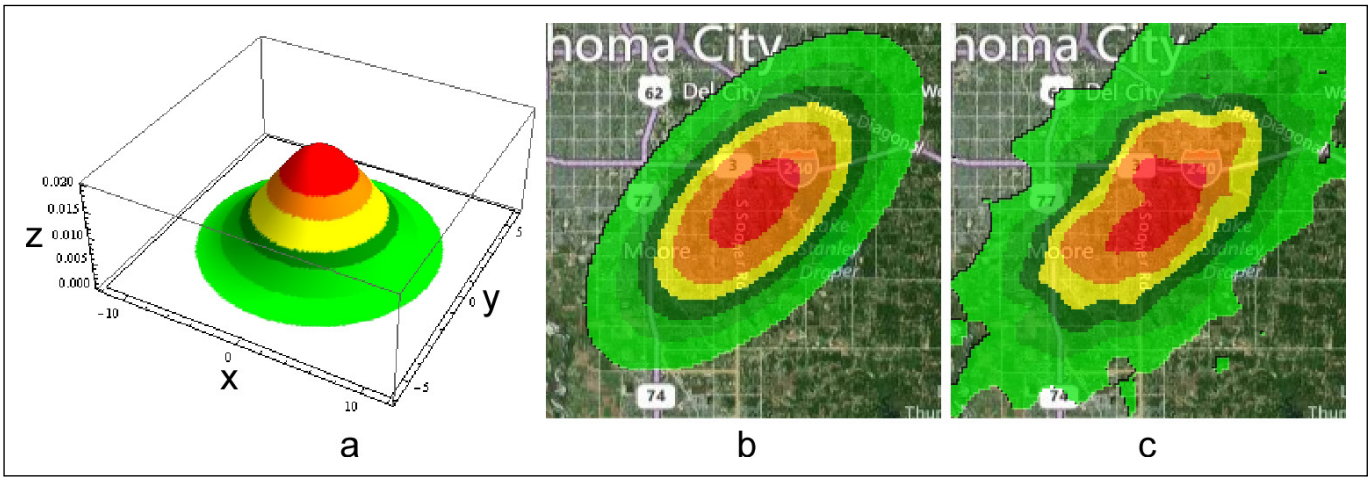


Figure 2. Fast-algorithmic emulation of simplified NEXRAD-like weather. a) the basic 3D structure, b) "a" as 2D, stretched, rotated, and partially transparent, overlaying a terrain map showing roads, c) "a" with added z-noise.

represent continuous, monotonic[1] values of weather risk, which could then be stratified to represent discrete, contiguous color selections from the NWS WSR-88D Intensity Legend recommended in FAA's *Advisory Circular 00-24C* (FAA, 2013). The 3D structure lent itself to being mathematically translated (moved sideways or up and down), rotated, and stretched in x and y via standard linear affine transforms.

Horizontal slices through this basic 3D figure produced a series of ellipses (see Appendix A for proof). Ellipses are sufficiently simple to allow direct computation of minimum distance from aircraft to any given weather risk boundary (*orange* is what *AC 00-24-C* advises pilots to clear by at least 20 nautical miles). Adding z-noise, followed by polynomial smoothing, roughened the edges of each colored region. Flattening (collapsing across z, Figs. 2 b,c) added realism without much computational overhead. The result was not fully realistic, of course, but looked reasonable enough for experimental purposes.

Numerical methods[2] were used to estimate PCA. These are detailed in Appendix A. The trick here was to measure distance to the underlying elliptical basis function (Fig. 2b), rather than the noisy function (2c). Given that the ellipse boundary was the center of the noise probability density function, this method was consistent with an ideal observer approach[3] (Geisler, 2011), and logically defensible on that basis.

Three separate computer programs were written, a) *Weather Generator*, b) *Viewer*, and c) *Autoscorer*. *Mathematica* code for each is presented in Appendix B.

Weather Generator. As the name suggests, *Weather Generator* allowed independent control over the sizes, aspect ratios, placement locations, rotation angles, movement direction, and speed of the ellipse-based weather cells. Two cells were generated for each scenario. The program generated 12 .jpg images per scenario, representing one hour's worth of "weather history" at 5-minute intervals.

Viewer. The second program, *Viewer,* served as the experimental data-gathering platform. *Viewer* sequenced each scenario's 12 images into a looping movie, displayed that movie at 2.2 fps, and allowed data collection on each pilot's performance. Figure 3a shows sample screenshots of "shallow" weather cells. Figure 3b shows "deep" cells.

Viewer displayed a scale of miles at lower left, a "Freeze" button, which stopped the movie at the most-current point in time, and a "Resume" button, which allowed looping to resume.

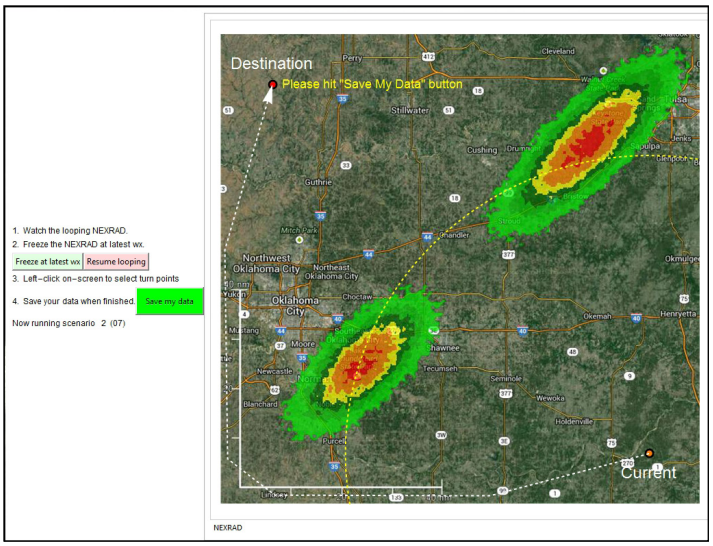


Figure 3a. Viewer, showing "shallow" weather cells, and a sample flightpath with four inflection points (dashed white line, starting at "Current," ending at "Destination"). The dashed, yellow line is a ½-hour range ring.

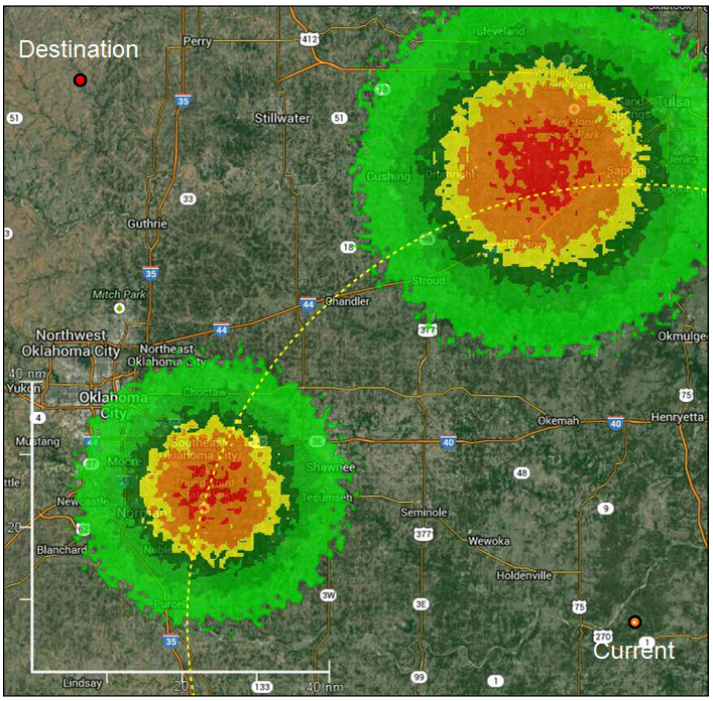


Figure 3b. A partial screenshot showing "deep" weather cells.

A yellow, circular dashed range ring depicted where the aircraft was expected to be in ½ hour, given its stated speed. Together, these features were designed to help pilots estimate clearance from weather as they planned a flight, but without adding overly complex technology to an otherwise standard NEXRAD-type display.

Pilots used a standard optical computer mouse to create a flightplan. Each flightplan automatically began at the map point labelled "Current," representing the aircraft's current position at the most recent weather frame.

To draw the flightplan, the pilot had simply to move a mouse cursor to a desired *inflection point*—a location on the map where

[1] Monotonic functions preserve numerical ordering along an axis. For example, as x increases, y might continually increase (or, at worst, remain constant), but would never decrease.

[2] Numerical methods are mathematical techniques for estimating the value of functions too complex to be simplified down to a "closed-form solution" (one that can be solved with a single equation).

[3] An "ideal observer" is a hypothetical observer that presumably knows the statistical properties of a situation, and can therefore make the absolute best possible guess about chance outcomes. In the present context, an ideal observer could "see through the noise" added to each ellipse to determine exactly where each level of weather intensity would be, on average. Remarkably, humans are reasonably good at this type of task.

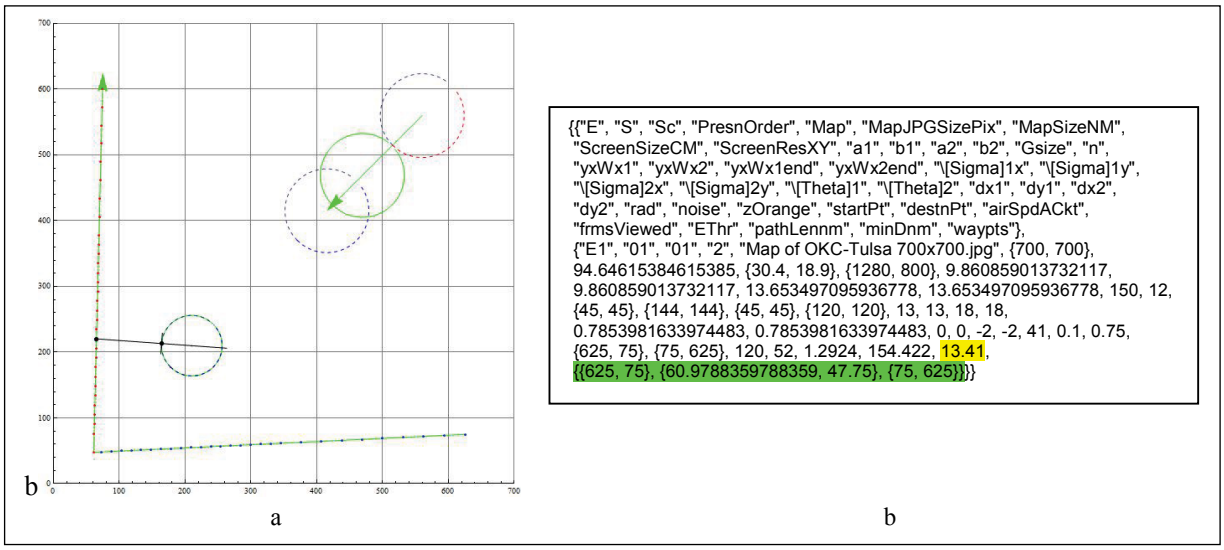


Figure 4, a) sample Autoscorer graphical output, b) the associated text output file. Minimum distance to orange here was 13.41 nm (highlighted yellow). Flightpath inflection points (in screen coordinate units) are highlighted green.

they planned to execute a turn—and then left-click. The computer drew a straight line from each point to the next, and recorded the *x,y* screen coordinates of each inflection point.

Viewer was set up on the assumption that "current" weather meant "real-time" weather. That is, it contained no *data latency*—in this case, no 15-minute minimum time delay such as seen in current NEXRAD (NTSB, 2012). The reason for this will be discussed shortly.

The challenge of this task was that the display could show the position of the aircraft and weather *up to* the current time. Just like real NEXRAD, it would still be up to the pilot to judge where both the aircraft and weather would be *in the future*, and whether or not a planned flightpath might get closer than 20 nm from "orange" weather.

The reason for having zero data latency and extremely simple-looking weather was straightforward: If the pilots' task of maintaining safe clearance from weather proved difficult under these nearly ideal conditions, then it can safely be assumed that the real-life task of using present-generation looping NEXRAD to navigate through weather will only be that much harder. And, that would be baseline information that policy makers, pilots, manufacturers, and the public want to know.

Autoscorer. The function of Autoscorer was to score each flightplan for PCA. Keep in mind that pilots' data consisted of flightplans based on their snapshot estimates of future-projected PCA. In reality, both their aircraft and the weather would change position as time went on and that flightplan was executed. This movement of aircraft and weather would make hand-scoring of the PCA difficult and inaccurate. Therefore, a computer program was written to do this.

Appendix A describes the mathematics of this *Autoscorer* program, and Appendix B shows the *Mathematica* code. Figure 4a shows a sample autoscored "Deep" weather scenario. The three dashed circles[4] represent the mean likelihood estimator (MLE) boundaries of the "orange, must-avoid" weather at scenario's beginning, PCA, and end, respectively. During the experiment, note that the lower-left weather cell always remained static, while the upper-right cell moved along a straight-line path (denoted by green arrow). In this example, the upper-right cell's movement resulted in the gap closing over time. Scored PCA is represented by a line drawn from the flightpath to the lower-left-hand circle, intersecting at a point tangent to that curve.

As stated previously, the technical challenge for the pilot making a go/no-go decision was that the looping NEXRAD ended about 20-30 minutes before the aircraft would actually reach hazardous weather. So, judging PCA requires pilots to mentally future-project the position of both weather and their aircraft. This looks like a challenging task, and is precisely what this experiment was designed to measure.

By design, the mathematical structures chosen to generate the weather (i.e., ellipses) were sufficiently regular to permit computerized scoring of PCA by numerical methods, given the assumption that the weather moved along essentially unchanged in shape and/or direction. Naturally, this was not fully realistic but, again, was "ideal" in the ideal-observer sense, and such circumstances had to be tested first.

Appendix A shows how scoring was conducted. Essentially, *Autoscorer* started at the "Current" point, finding PCA to orange from that point. It then moved forward in time, step-by-step, along the flightpath, calculating where both the aircraft and the weather would ideally be at each new point in time, given their

[4] Consider that a circle ($k_1=x^2+y^2$) is just a special case of an ellipse ($k_2=(x/a)^2+(y/b)^2$) where $a=b$.

known velocities. At each time step, the smallest PCA encountered to date was stored. Sampling density increased as PCA approached 20 nm and continued increasing as PCA approached zero. This allowed computational speed to remain high, while still maintaining PCA accuracy to at least three decimal places.

Experimental Design

Statistical paradigm. Prior research supports that *repeated-measures* designs work well for this particular kind of task (Knecht & Frazier, 2015). Repeated-measures present multiple trials within a single experiment, each trial testing a different combination of independent variables (IV). Each participant sees all trials, allowing change scores to be analyzed. This sharply increases statistical power, our ability to detect true effects where they indeed exist (Keppel & Wickens, 2004).

Nonetheless, repeated measures can sometimes induce treatment order effects, in which conditions in a particular trial may bias the outcome of the subsequent trial(s). These biases can be statistically controlled by either counterbalancing or randomizing the trial presentation order. Randomization was chosen here, since 10 trials would have required 10! (3,628,800) presentation orders for a full counterbalance.

Independent variables. Independent variables are those the experimenter controls to see which ones influence the outcome measures (i.e., the *dependent variables*, or DV). There were quite a few IVs that could be examined, including

1. Weather cells'
   a. initial *x,y* location
   b. length
   c. depth
   d. angle of rotation
   e. overall severity
   f. speed of movement
2. Aircraft's
   a. initial distance from closest weather
   b. nominal angle of attack with weather
   c. speed

To simplify this initial experiment, only two weather depths and five gap-opening speeds were set up (a 2x5 design). Additionally, the lower-left weather cell was kept static, always located to guarantee at least one safe option for flying around the weather. The remaining weather cell was dynamic, with positive speeds representing a gap that opened over time, zero denoting a static gap, and negative speeds representing a closing gap. At 22" (56 cm) viewing distance, this produced visual angular displacements of -1.3, -0.65, 0, 0.65, and 1.3 deg/sec, and corresponded to simulated weather-movement speeds from 0-14 kt. In the Figures and tables that follow, these are labelled by their effect on the size of the gap between cells, as "closing-fast," "closing-slow," "static," "opening-slow," and "opening-fast," respectively.

Experimental hypotheses. This approach shared the goals of cue utilization (Wiggins et al., 2012). Generally, the goal was to find the information in the stimulus (cues) upon which perception and cognition act. Specifically, it would be useful to know if any particular information produced difficulty in task execution.

First, the *storm depth* might be a factor. "Shallow" storms might reasonably tempt more penetrations than "deep" storms, because gaps could be slipped through quicker, therefore risk exposure would be shorter-duration.

Second, *gap-opening speed* might also be a factor. Gaps increasing in size might tempt more penetrations than static gaps, which in turn, might tempt more penetrations than gaps becoming narrower. It was critically important to avoid the trivial, unwanted situation where everyone either always diverted around the weather or else always flew straight to the "Destination." To accomplish this, the initial location and speed of each scenario's moving weather cell was carefully set to match the aircraft speed, in order to always produce just a small fraction more than the 20 nm required clearance from "orange" weather, should the pilot choose to "fly direct." However, none of the pilots were told that it was always safe to fly direct. They were only told that it was acceptable to go direct *when* it was safe. This was expected to be the best way to probe the "go/no-go impulse" to test the experimental hypotheses (and, as we shall see, did appear to work effectively).

Participants

Recruitment. GA pilots were recruited from a local Oklahoma City, OK area flight school. Researchers travelled there with portable equipment, making participation easy for pilots, and greatly shortening the time required to collect data.

Quick "shakedown" test. This was a complex system. Therefore, four pilots were first tested, and their data analyzed, to better ensure that the settings for the weather parameters produced good response variability.

Instructions. The four "shakedown" pilots sometimes approached weather too closely when diverting around the static cell. Therefore, the instructions were slightly modified to emphasize the need for maintaining clearance, whether flying direct-to-destination or diverting. However, care was taken not to overemphasize this, since it might prove to be an interesting result on its own.

Pilot demographics. The main test group consisted of 21 pilots. Table 1 shows certificates, ratings, age, and total flight hours (TFH). This was a relatively young group, only four of whom were instrument-rated. Nonetheless, lack of an instrument rating was considered acceptable, since this was a study

Table 1. Pilot demographics (*N*=21).

Student	8	**CFII**	2	**Age-mean**	22.7	**TFH-mean**	176.5
Private pilot	13	**Commercial**	2	**Age-median**	21.5	**TFH-median**	137.5
Instrument-rated	4	**ATP**	0	**Age-SD**	4.0	**TFH-SD**	185.4
CFI	3	**Multi-engine**	2				

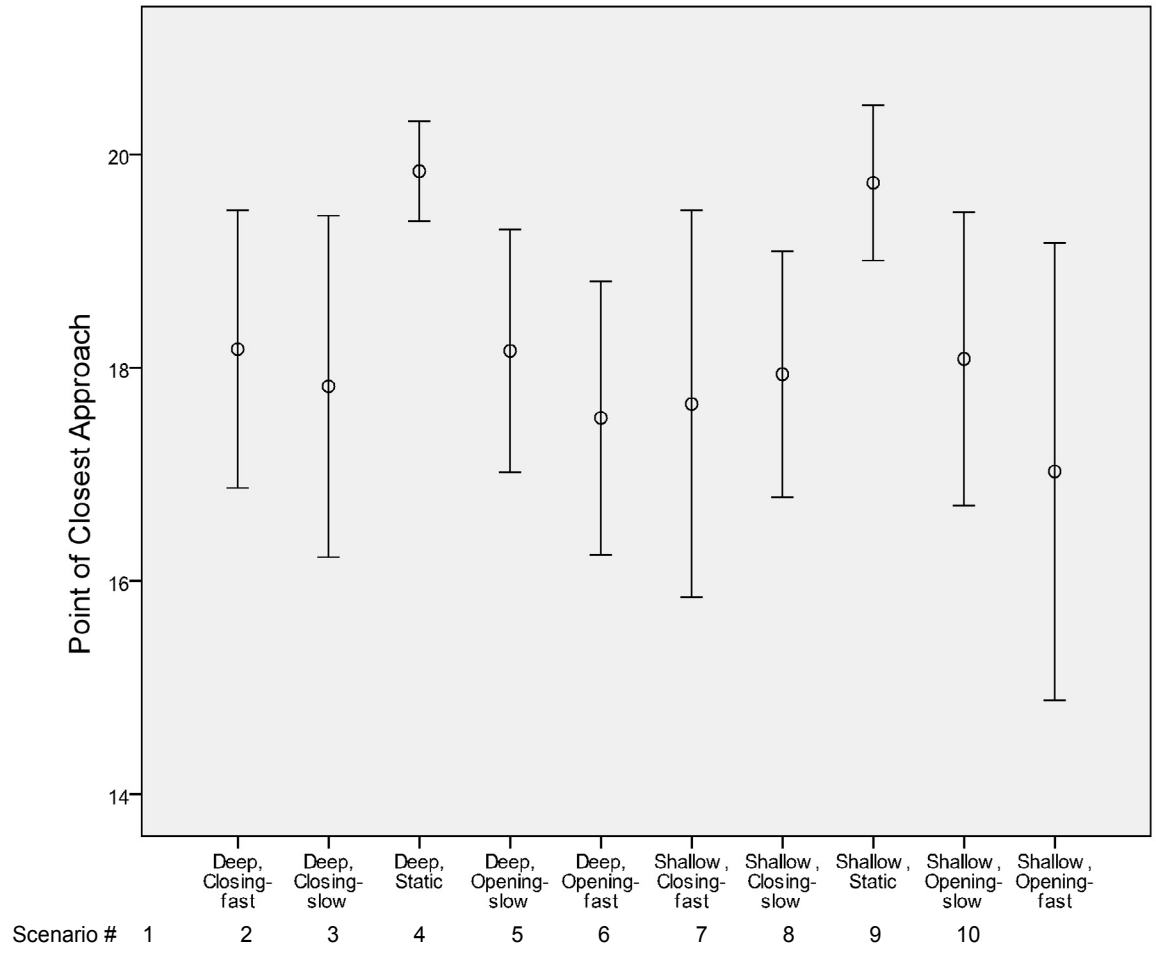


Figure 5. PCA (vertical axis, in nm) between aircraft and "orange" weather for two values of weather depth ("deep" vs "shallow"), and five weather gap-closure speeds (-14,-7,0,7, and 14 kts). Each group mean is shown as an open circle. Error bars show the range encompassed by the 95% confidence interval.

focusing on human perceptual judgment of times and distances. Moreover, pilots were specifically instructed to fly direct-to-destination whenever it looked safe, regardless of whether they were instrument-rated. A safe flight, by definition, should not take them through hazardous weather.

RESULTS

Main effects of weather width and speed. Figure 5 summarizes the 21 pilots' points-of-closest-approach for the 10 scenarios. Recall that PCA to "orange" weather was the main performance variable (DV), and that weather cells were defined as "shallow" if the weather cells were 19 nm deep and the gap could be flown through quickly, and "deep" if the cells were 40 nm deep and the gap would take longer to fly through.

First, note how relatively few cases maintained 20 nm clearance, and that these were concentrated in the "deep static" (#3) and "shallow static" (#8) weather conditions, where neither weather cell moved.

Second, note the great variability in PCA, as shown by the tall confidence intervals. Note that, again, variability was least for the two completely static-weather conditions.

Repeated-measures analysis of variance (RM-ANOVA) showed weather depth to be nonsignificant ($p=.636$). Weather speed was significant ($p=.003$), but very nearly failed Mauchly's test of sphericity ($p=.058$)[5], and did fail it for the interaction of depth × speed ($p=.007$).

Mauchly's test implied it prudent to collapse (i.e., average) the data across width for better stability, and then use a more conservative test based on rank-ordering, for example, the non-parametric Friedman test.

[5] In Mauchly's test, $p>.05$ suggests that the data meet assumptions of ANOVA, and that parametric statistics can be used. Failing Mauchly's test suggests that nonparametric ("distribution-free") statistics should be used.

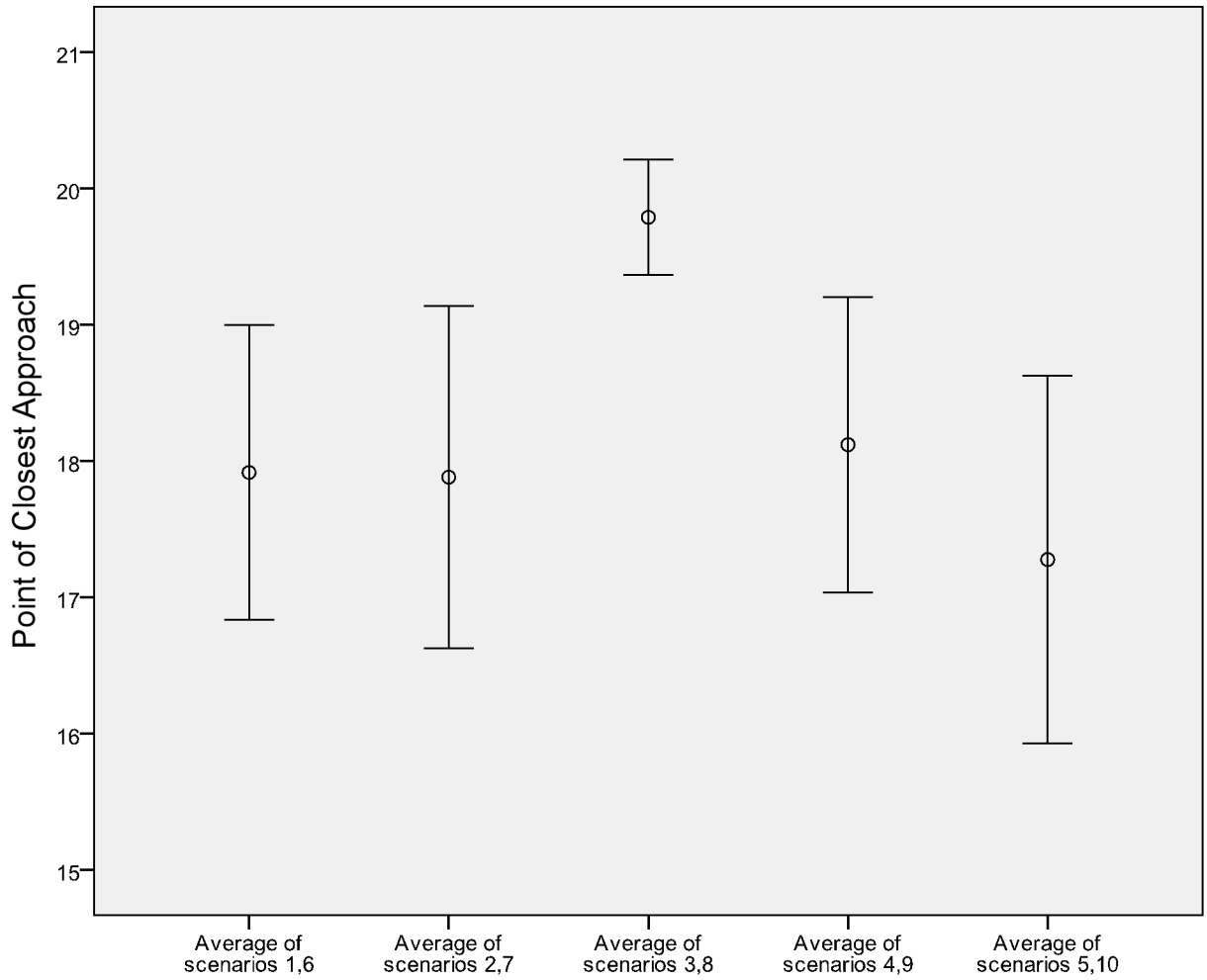


Figure 6. The same data from Figure 5, collapsed across weather depth (i.e., "shallow" and "deep" combined). "Average of scenarios 3,8" represents the static condition, where weather cells did not move.

Shown in Figure 6, this approach clearly visualizes the significantly better performance ($p=.016$) on the completely static scenarios (scenarios 3 and 8).

Shooting the gap between weather cells. A second way to examine these data is to count the numbers of pilots who shot the gap between weather cells (which, recall, was always technically safe). From the perspective of signal detection theory (SDT), having safe, sufficient clearance from hazardous weather is a signal that can be detected. And, shooting the gap therefore implies a "Hit," which is defined in SDT as a correct detection of the signal (Green & Swets, 1966).

Here, a pattern in "Hits" emerged similar to the pattern in Figure 5. Therefore, results were again collapsed across width (Fig. 7). A chi-square test showed that these results also differed significantly ($pX^2 = .03$), with the greatest number of gaps shot—again—when both weather cells were static.

Summary of main results. Given the agreement between PCA and gaps-shot, it appears that—for the two levels of weather depth tested here—successful judgment of PCA with looping NEXRAD did not seem to depend on whether the weather system was shallow or considerably deeper. Nonetheless, this should be interpreted cautiously, since extremely deep weather cells were not tested.

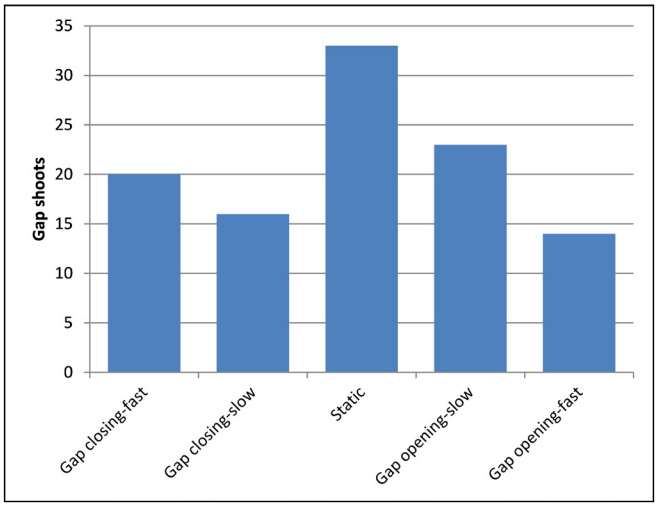


Figure 7. Numbers of gaps shot.

Table 2. PCA to closest "orange" weather (Scenarios 3 and 8 wx were fully static). See color key at bottom.

	Deep weather					Shallow weather				
S	Gap closing fast	Gap closing slow	Static	Gap opening slow	Gap opening fast	Gap closing fast	Gap closing slow	Static	Gap opening slow	Gap opening fast
5	16.144	16.145	20.554	17.182	16.507	15.320	17.058	15.680	14.135	15.865
6	9.819	6.151	20.554	20.556	20.560	20.079	12.721	20.077	20.079	20.085
7	21.899	22.077	18.177	18.430	21.699	20.193	22.421	22.504	20.860	21.484
8	20.562	12.927	19.982	14.900	14.815	13.735	17.538	19.186	12.797	16.078
9	20.559	20.267	20.554	14.877	16.145	19.275	19.622	20.077	17.019	15.076
10	16.359	19.682	20.158	19.425	21.119	19.316	20.079	22.270	22.194	21.714
11	18.322	19.551	18.132	18.713	17.832	2.153	17.540	20.146	18.670	0.004
12	20.559	17.786	20.554	20.556	16.152	16.497	18.174	20.077	20.079	15.761
13	20.559	20.556	20.554	16.653	16.352	20.079	20.079	20.077	20.079	17.884
14	20.559	19.925	20.554	20.008	19.936	20.704	18.438	20.077	20.308	14.970
15	15.726	20.556	20.554	16.384	15.947	19.272	20.079	20.077	14.541	14.692
16	17.703	16.492	20.554	20.556	16.859	20.079	20.079	20.077	20.079	20.085
17	14.621	14.692	17.662	15.424	15.216	17.578	16.154	17.303	16.274	12.980
18	17.104	20.965	19.368	20.431	10.104	18.236	17.862	21.233	19.755	20.476
19	14.681	18.224	19.041	13.435	15.044	20.022	12.923	20.077	13.048	20.085
20	18.951	19.410	19.656	19.629	19.857	17.111	21.166	20.938	17.521	17.071
21	18.692	18.545	20.554	18.784	18.558	19.021	19.314	18.789	19.790	18.720
22	19.807	19.177	17.907	20.556	20.560	17.361	18.545	18.626	20.079	20.085
23	18.718	18.977	20.554	20.556	18.797	17.086	16.253	16.982	12.268	20.085
24	20.559	15.603	20.554	13.722	15.484	20.079	16.304	20.077	20.079	14.276
25	19.751	16.635	20.554	20.556	20.560	17.666	14.376	20.077	20.079	20.085
Means	**18.2**	**17.8**		**18.2**	**17.5**	**17.7**	**17.9**		**18.1**	**17.0**

Color Key

"Direct" = pilot flew directly to "Destination" (i.e., shot the gap). "Divert" = pilot avoided the gap, & flew around the wx.

Outcome	Direct-success	Direct-failure	Divert-success	Divert-failure	Divert—cell 2

More importantly—given equal gap widths at expected time of arrival—it did not seem to matter if weather was moving relatively fast or slowly, nor whether gaps were opening or closing. The mere fact that weather was moving at all seemed to greatly interfere with pilots' judgment of PCA.

Additional details. Several additional results emerged. Table 2 shows these in detail.

First, the overall task seemed quite difficult since only 85 of the 210 total scenarios produced separation greater than 20 nm (Table 2, white and light-gray cells). Moreover, even though all scenarios were set up so that direct-to-destination flight was technically safe and pilots were encouraged to fly direct when they judged it was safe, they chose to divert around weather 103 times (light-gray, gray, and red cells). Both these results support the conclusion of high task difficulty.

Second, as Figure 4a showed earlier, most pilots seemed to misjudge distance to the static weather cell during diversions. Table 2 (light-gray and gray cells) shows that all but three of the 103 diversions were made around the static cell (red cells show diversion around the moving cell). Of those 100 diversions made around the static cell, 80 were technically failures.

This high PCA failure rate might indicate a kind of "boundary shyness," that is, a tendency of users to avoid hugging the very edge of the display when planning a maneuver. If so, it could be easily overcome, either by instruction and practice, or by adding a zoom feature to the display to allow shrinking the scene, thereby increasing the apparent distance between weather images and the edge of the display.

Finally, as mentioned, there were three attempts to divert by going around the moving weather cell (Table 2, red cells). All three were serious failures. This probably indicates that, had there been *two* moving weather cells instead of just one, the task of judging clearance would have been even more difficult than the task presented here.

DISCUSSION

Looping NEXRAD is now finding its way into GA aircraft cockpits. Experience suggests that pilots may use this technology to try to pick their way through convective weather cells. This suggests a need for research, hazard identification, and prophylactic "immunization" of pilots through training, policy, and regulation *before* pilots begin to have accidents.

Little is known about how well pilots can use the information in looping NEXRAD to future-project where both they and the weather will be, in order to maintain a safe distance from hazard. We therefore ask ourselves what are the cues in the display itself that might tend to promote safety versus encouraging excessive risk taking?

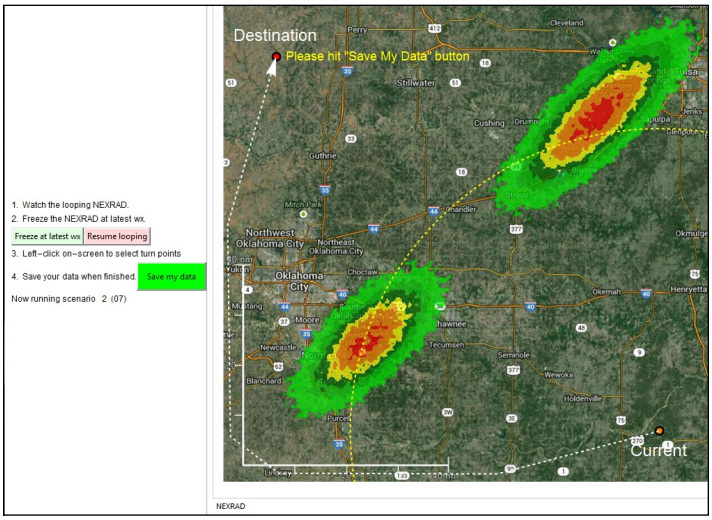


Figure 8. Viewer, showing "shallow" weather cells, and a sample flightpath with four inflection points (dashed white line, starting at "Current," ending at "Destination"). The dashed yellow line is a ½-hour range ring.

To this end, a looping NEXRAD-type cockpit display was created as a low-cost, part-task simulation. Figure 8 illustrates. Mathematically generated "convective weather radar images" allowed precise control over the placement and movement of weather cells, letting us test the effects on flight safety of two potentially important variables—weather system *depth*, and the *opening and closing of gaps* between cells.

Results of this preliminary test indicated that, for shallow (19 nm deep) and moderately (40 nm) deep weather systems, judgment of PCA to hazardous weather with looping NEXRAD was about the same. More importantly, the mere fact that weather was moving at all seemed to greatly interfere with pilots' judgment of PCA. When scenarios were equilibrated for gap width at expected time of arrival, judgment of PCA was best for completely static weather. But, even when one cell remained static and the other moved, PCA judgment degraded considerably. Moreover, it did not seem to matter if weather movement was relatively fast or slow, nor whether gaps were opening or closing. Judgment of PCA with any moving weather was clearly a difficult task.

RECOMMENDATIONS

Further testing would be desirable. A first test could measure pilot performance under more realistic circumstances—omit the range ring that was present here, have both weather cells move (rather than just one), and run the trials with 15-minute data latency, rather than the zero-latency assumed here. In that event, one would expect performance to decline even further from the levels reported here.

A second test could measure performance after pilots were trained with feedback on PCA after every trial. Training would probably improve performance (Vincent, Blickensderfer, Thomas, Smith, & Lanicci, 2013). The question is how much. Could trained pilots reliably maintain safe clearance with current NEXRAD technology?

If not, a third test could measure the effects of relatively simple, creative additions to the current technology (e.g., an ownship 30-minute, future-projected "flight corridor" providing a simple way to estimate 20-nm clearance. Or, edge-detection technology that could find the boundaries of hazardous weather, and future-project them by 45 minutes). Future-projected weather display appears to aid safe weather flight (ATSC, 2013). At issue is whether future-projection can be done relatively simply, or will it require complex, computationally intensive weather models.

As far as specific safety recommendations are concerned, a prudent observer considering these findings must call into question the utility and safety of using current-generation looping NEXRAD to navigate safely through convective weather. Given the twin issues of a) *data latency* (i.e., the most-current data being as much as 15 minutes old), and b) the lack of *ranging* capability (i.e., no means of accurately determining how far away severe weather is), it seems probable that looping NEXRAD as currently displayed will only present difficulties even greater than those observed in this study.

REFERENCES

Atmospheric Technology Services Company. (2013). *Demonstration comparing the effects of probabilistic and deterministic forecast guidance on pilot decision-making and performance.* (Report no. FAA-20130902.1). Washington, DC: Federal Aviation Administration.

Batt, R., & O'Hare, D. (2005). *General aviation pilot behaviors in the face of adverse weather.* (Report no. B2005/0127). ACT, Australia: Australian Transport Safety Bureau.

Beringer, D.B., & Ball, J.D. (2004). *The effects of NEXRAD graphical data resolution and direct weather viewing on pilots' judgments of weather severity and their willingness to continue a flight.* (Technical Report no. DOT/FAA/AM-04/5). Washington, DC: Federal Aviation Administration. Downloaded 1, Dec., 2014 from http://www.faa.gov/data_research/research/med_humanfacs/oamtechreports/2000s/media/0405.pdf

Federal Aviation Administration. (2012). *NextGen implementation plan, March 2012*. Downloaded 15 Feb., 2013 from http://www.faa.gov/nextgen/implementation/media/NextGen_Implementation_Plan_2012.pdf

Federal Aviation Administration (Feb. 19, 2013). *Advisory Circular 00-24C*. Downloaded 14 Aug., 2013 from http://www.faa.gov/documentlibrary/media/advisory_circular/ac%2000-24c.pdf

Geisler, W.S. (2011). Contributions of ideal observer theory to vision research. *Vision Research, 51*(7), 771-781.

Gleick, J. (1987). *Chaos: Making a new science*. New York: Viking.

Green, D.M., & Swets J.A. (1966). *Signal Detection Theory and Psychophysics*. New York: Wiley.

Joint Economic Committee Majority Staff (2008). *Your flight has been delayed again.* Washington, DC. Downloaded 14 Feb., 2013 from http://www.jec.senate.gov/public/?a=Files.Serve&File_id=47e8d8a7-661d-4e6b-ae72-0f1831dd1207

Joint Planning and Development Office. (2010). *Next Generation Air Transportation System (NextGen) ATM-Weather Integration Plan, V2.0*. Washington, DC: JPDO. Downloaded 14 Feb., 2013 from http://www.jpdo.gov/library/JPDO_ATM_Weather_Integration_Plan_V2-0_Complete_Plan_with_Appendices.pdf

Kelley, W., Kronfeld, K., & Rand, T. (2000). Cockpit integration of uplinked weather radar imagery. *Proceedings of the 19th Digital Avionics Systems Conference, Vol 1,* 1-6. IEEE.

Keppel, G., & Wickens, T.D. (2004). *Design & Analysis—Researcher's Handbook.* (4th Ed.). Englewood Cliffs, NJ: Prentice-Hall.

Knecht, W.R.,& Frazier, E. (2015). *Pilots' risk perception and risk tolerance using graphical risk-proxy gradients*. (Technical Report no. DOT/FAA/AM-15/9). Washington, DC: Federal Aviation Administration. Downloaded 2 July, 2015 from http://www.faa.gov/data_research/research/med_humanfacs/oamtechreports/2010s/media/201509.pdf

National Transportation Safety Board. (2005). *Risk factors associated with weather-related general aviation accidents*. (Report no. NTSB/SS-05/01). Downloaded 14 Feb., 2013 from http://www.ntsb.gov/doclib/safetystudies/SS0501.pdf

National Transportation Safety Board. (2012). *NTSB Safety Alert SA_017*. Downloaded 5 May, 2014 from http://www.ntsb.gov/doclib/safetyalerts/SA_017.pdf

Vincent, M., Blickensderfer, E., Thomas, R., Smith, M., & Lanicci, J. (2013). In-cockpit NEXRAD products: Training general aviation pilots. *Proceedings of the Human Factors and Ergonomics Society 57th Annual Meeting,* 81-85.

Wiggins, M.W., Azar, D., & Loveday, T. (2012). The relationship between pre-flight decision-making and cue utilization. *Proc. Human Factors & Ergonomics Soc. 56*(1), 2417-21.

Wolfram Research. (2013). *Numerical nonlinear global optimization*. Downloaded 9 Oct., 2013 from http://reference.wolfram.com/mathematica/tutorial/ConstrainedOptimizationGlobalNumerical.html

APPENDIX A
Derivation of the Auto-Scoring Methodology

This appendix discusses the logical and mathematical rationale underlying the computer-generated "weather" used in this experiment, as well as the details of how that was created, and issues encountered along the way.

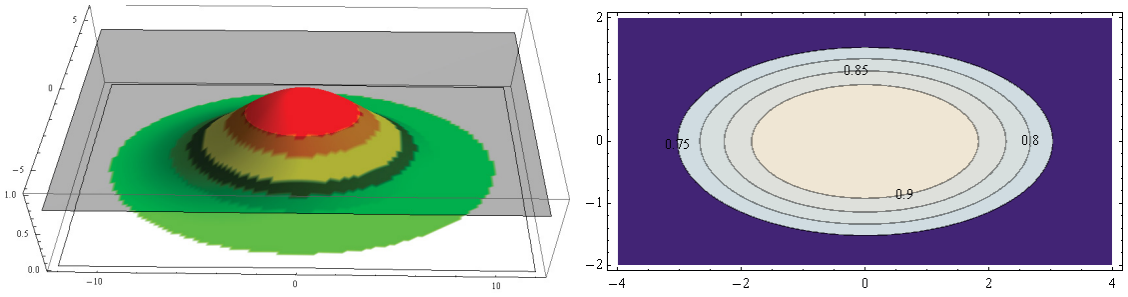
A.1. *Generating plausible-looking "scorable weather."* The experiment needed at-least-plausible-looking weather that could be quickly and accurately scored for point-of-closest-approach (PCA) between the pilot's moving aircraft and a moving, "orange" weather area. To avoid having to use a slow, frame-by-frame, pixel-by-pixel "brute-force" approach to calculating PCA, it was highly desirable that weather be represented by some kind of underlying regular, relatively tractable 2D mathematical function. To accomplish this, a number of problems had to be solved. First, was the *contiguity problem*.

A.2 *Solving the contiguity problem. Contiguity* involves maintaining the relative order of colors representing hazard levels. Red areas must always sit next to orange areas, which must sit next to yellow areas, and so forth. Maintaining this order is not as simple as it might sound.

The problem can be approached by imagining a 3D (*x,y,z*) *basis function*, for each weather cell. Visualize a mountain, with height (*z*) representing weather severity (here, decibels reflectivity in NEXRAD). Fig. A1a illustrates. Height will be greatest at one central point (the origin, representing maximum intensity within a red cell), and monotonic—always decreasing when moving outward, radially. Importantly, visualize that horizontal slices through this structure will always have constant height at their outer edge.

This simple idea guarantees 2D contiguity in *x* and *y* at each horizontal slice, ensuring that colors will remain in their proper order.

In solving the contiguity problem, note that finding PCA now merely involves following the 2D contour of the outer edge of a single 2D slice (here, "orange weather"), rather than having to check distances to hundreds of thousands of individual pixels per frame, as a brute-force method would require. This "side effect" will save much computational effort, provided we can find a 2D basis function that represents this new, "orange" weather contour.



a b

Fig. A1. a) a 3D Gaussian probability density function, b) horizontal slices through which form 2D ellipses.

A.3. *Choosing the geometric basis functions.* First, a 2D function must be chosen to represent each weather-intensity boundary. This 2D function has to look at least somewhat like a real weather cell (i.e., no triangles, rectangles, or any other regular angular structure). Moreover, it has to be able to be extracted from its parent 3D basis function.

For a useful 2D basis function, ellipses seem a fair compromise. Fig. A2 shows an ellipse, while Eq. A1 describes its generator.

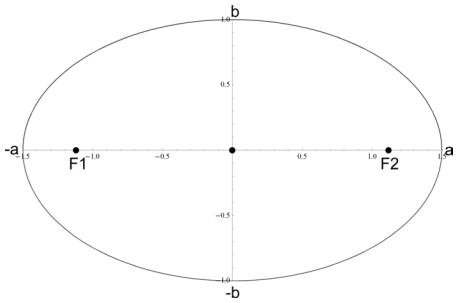


Fig. A2. An ellipse, showing the major axis ($-a \le x \le a$) and minor axis ($-b \le y \le b$). (*F1* and *F2* are the foci).

$$z = \frac{x^2}{a} + \frac{y^2}{b} \qquad \text{where } z, a, \text{ and } b \text{ are constants} \qquad \text{(A1)}$$

A.4 *Demonstrating ellipticality.* To end up with 2D horizontal cross-sections that are elliptical (i.e.,where threat level z equals a constant), we can start with a 3D Gaussian probability density function (pdf, Eq. A2), which is simply a Gaussian in x times another Gaussian in y.

$$z = \frac{e^{-\left(\frac{x^2}{2\sigma_x^2}\right)}}{\sigma_x\sqrt{2\pi}} \frac{e^{-\left(\frac{y^2}{2\sigma_y^2}\right)}}{\sigma_y\sqrt{2\pi}} \qquad \text{(A2)}$$

We can prove that horizontal cross-sections will be ellipses by demonstrating that the 3D Gaussian is equivalent to a standard ellipse at any fixed, constant height z.

$$z = \frac{e^{-\left(\frac{x^2}{2\sigma_x^2}\right)}}{\sigma_x\sqrt{2\pi}} \frac{e^{-\left(\frac{y^2}{2\sigma_y^2}\right)}}{\sigma_y\sqrt{2\pi}} \qquad \text{is our 3D Gaussian pdf, with height } z \qquad \text{(A3a)}$$

$$\text{So,} \quad zk_1 = e^{\left(\frac{x^2}{k_2}\right)} e^{\left(\frac{y^2}{k_3}\right)}, \qquad \text{where } k_1 = 2\pi\sigma_x\sigma_y,\ k_2 = -2\sigma_x^2,\ k_3 = -2\sigma_y^2 \qquad \text{(A3b)}$$

$$\text{Taking the natural log of both sides,} \quad \ln(zk_1) = \ln(e^{\left(\frac{x^2}{k_2} + \frac{y^2}{k_3}\right)}), \qquad \text{(A3c)}$$

$$\text{which gives} \quad k_4 = \frac{x^2}{k_2} + \frac{y^2}{k_3}, \qquad \text{where } k_4 = \ln(zk_1) \qquad \text{(A3d)}$$

$$\text{Finally, we divide by } k_4\text{, giving} \quad 1 = \frac{x^2}{a^2} + \frac{y^2}{b^2} \qquad \text{where } a^2 = k_2k_4,\ b^2 = k_3k_4 \qquad \text{(A3e)}$$

which is the standard ellipse. QED, as Fig. A1b showed, horizontal cross-sections through 3D Gaussians are always ellipses, regardless of z.

A.4 *Computing a and b.* Given a 3D Gaussian with specified values for σ_x, σ_y, and z, we need to know the major and minor axis values a and b in order to position and move our ellipses, and to compute PCA, which we shall formally call the minimum distance d_{min}.

We start by finding the maximum value for z, in order to normalize the 3D Gaussian, setting its maximum value equal to 1.0. This z_{max} will, of course, occur at position $(x,y)=(0,0)$.

$$z_{\max} = \frac{e^{-\left(\frac{0}{2\sigma_x^2}\right)}}{\sigma_x\sqrt{2\pi}} \frac{e^{-\left(\frac{0}{2\sigma_y^2}\right)}}{\sigma_y\sqrt{2\pi}} = \frac{1}{2\pi\sigma_x\sigma_y} \tag{A4a}$$

Dividing z by z_{max}, we get the normalized form

$$z_{norm} = \frac{z}{z_{\max}} = 2\pi\sigma_x\sigma_y \frac{e^{-\left(\frac{x^2}{2\sigma_x^2}\right)}}{\sigma_x\sqrt{2\pi}} \frac{e^{-\left(\frac{y^2}{2\sigma_y^2}\right)}}{\sigma_y\sqrt{2\pi}} \tag{A4b}$$

$$z_{norm} = e^{-\left(\frac{x^2}{2\sigma_x^2}\right)} e^{-\left(\frac{y^2}{2\sigma_y^2}\right)}, \quad \text{now on a scale of 0-1} \tag{A4c}$$

$$\text{So,} \quad z_{norm} = e^{\left(\frac{x^2}{k_2}\right)} e^{\left(\frac{y^2}{k_3}\right)}, \quad \text{where } k_2 = -2\sigma_x^2,\ k_3 = -2\sigma_y^2 \tag{A4d}$$

$$= e^{\left(\frac{x^2}{k_2} + \frac{y^2}{k_3}\right)} \tag{A4e}$$

$$\text{Taking the natural log of both sides, } \ln(z_{norm}) = \ln(e^{\left(\frac{x^2}{k_2} + \frac{y^2}{k_3}\right)}), \tag{A4f}$$

$$\text{giving} \quad k_4 = \frac{x^2}{k_2} + \frac{y^2}{k_3}, \quad \text{where } k_4 = ln(z_{norm}) \tag{A4g}$$

$$\text{Finally, dividing by } k_4 \text{ gives} \quad 1 = \frac{x^2}{a^2} + \frac{y^2}{b^2}, \quad \text{where } a^2 = k_2k_4,\ b^2 = k_3k_4, \tag{A4h}$$

which takes us into standard form. Back-substituting from A4d-h, we get workable formulae for a and b

$$a = \sqrt{k_2k_4} = \sqrt{-2\sigma_x^2 \ln(z_{norm})} \tag{A4i}$$

$$\text{and} \quad b = \sqrt{k_3k_4} = \sqrt{-2\sigma_y^2 \ln(z_{norm})}, \tag{A4j}$$

which is correct at any arbitrary value of z from 0-1. Below, we can graphically and numerically see this for an ellipse with σ_x=6 and σy=2 at z=.25, where A4i yields $a \rightarrow 9.99$ and A4j yields $b \rightarrow 3.33$.

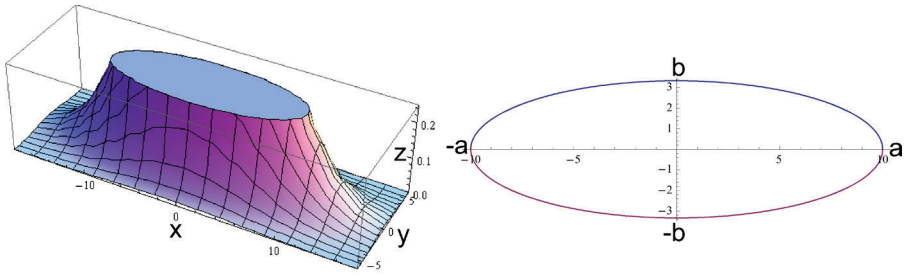


Fig. A3. Finding the elliptical parameters a and b from a section of our 2D Gaussian.

QED, given a 2D Gaussian, we can compute a and b for any arbitrary value of z.

A.5 *Computing minimum distance from aircraft to ellipse.* The simplest situation for computing d_{min} is where neither the aircraft nor the ellipse is moving, and where the ellipse is in standard position (i.e., non-rotated). There is value in examining that simplified situation, so we begin there.

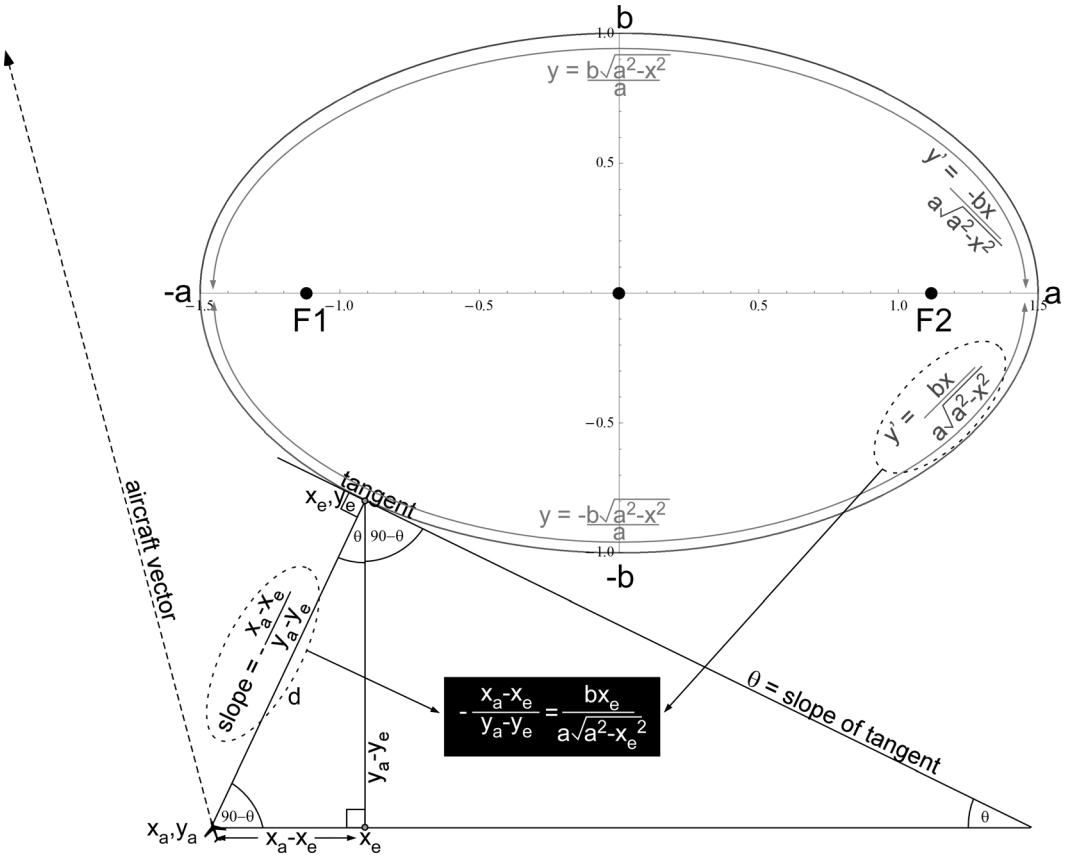


Fig. A4. Computing the distance from aircraft to the closest point on the ellipse boundary.

Given a fixed point for aircraft position (x_a,y_a), the minimum distance to the ellipse will be the length d of a line intersecting at right angles to a tangent line on the ellipse. Our task is to find the intersection point (x_e,y_e) on the ellipse. That will allow us to compute the distance

$$d = \sqrt{(x_a - x_e)^2 + (y_a - y_e)^2}\,, \tag{A5a}$$

We know the basic equations for the ellipse itself, and for its slope

$$\text{Ellipse} \qquad y_e = \pm\frac{b\sqrt{a^2 - x_e^2}}{a}, \text{ and} \tag{A5b}$$

$$\text{Slope of ellipse} \qquad y_e' = \mp\frac{bx_e}{a\sqrt{a^2 - x_e^2}}, \text{ respectively.} \tag{A5c}$$

If we can find either x_e, or y_e, we can compute the other term. We also know from basic trigonometry that the slope of the triangle containing d will be the same as the slope θ of the tangent line, except with opposite sign. That slope can be described as

$$y_e' = -\frac{x_a - x_e}{y_a - y_e}, \tag{A5d}$$

Therefore, from Eqs. A5c and A5d,

$$-\frac{x_a - x_e}{y_a - y_e} = \frac{bx_e}{a\sqrt{a^2 - x_e^2}}, \tag{A5d}$$

for which we critically know a, b, x_a, and y_a, but not x_e.

Using Eq. A5b, we can substitute out y_e, giving

$$-\frac{x_a - x_e}{y_a - \frac{b\sqrt{a^2 - x_e^2}}{a}} = \frac{bx_e}{a\sqrt{a^2 - x_e^2}} \tag{A5e}$$

A6. Preliminary proof of plausibility. We now know all terms but one, so, we can theoretically solve for x_e. Unfortunately, the closed solution proves so complicated, with so many terms—*even in this simplified case with no movement of aircraft or ellipse*—that rounding error will literally make it less accurate than computation by numerical methods. Therefore, we should expect to use a numerical technique such as Nelder-Mead (Nelder & Mead, 1965) to solve the distance equation, Eq. A5a (Wolfram, 2013). Our task is becoming harder, but is still manageable.

Next, we must examine the case where our weather and aircraft are both moving.

A7. Computing minimum distance from a moving aircraft to a moving, rotated ellipse. To accommodate aircraft and weather movement, we can reformulate our equations as parametric functions of time (t). We must also have the ability to present weather at angles other than horizontal.

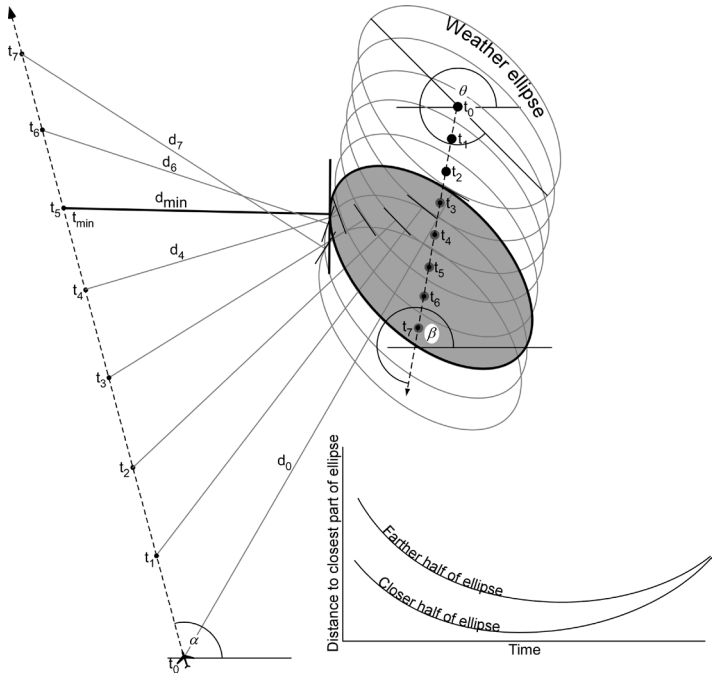


Fig. A5. Computing distance to a rotated ellipse, given straight-line aircraft and ellipse flightpaths. Note that d_{min} still intersects at a right angle to a tangent on the ellipse.

The aircraft flightpath can be represented in 2D simply by a series of straight-line segments. Within any given segment, the aircraft's parametric position (x_{act}, y_{act}) at elapsed time t is merely the original position (x_{ac0}, y_{ac0}) at time t_0, augmented by its speed S_{ac} parsed into x and y components for the angle α.

$$\begin{aligned} x_{ac(t)} &= x_{ac(0)} + v_{acx}t = x_{ac(0)} + S_{ac}\cos\alpha\, t \\ y_{ac(t)} &= y_{ac(0)} + v_{acy}t = y_{ac(0)} + S_{ac}\sin\alpha\, t \end{aligned} \tag{A7a}$$

The weather "flightpath" will follow similar logic. Note that we are going to limit weather to a single straight-line path during each experimental run. For angle β, all points on the ellipse e translate to

$$\begin{aligned} x_{e(t)} &= x_{e(0)} + v_{ex}t = x_{e(0)} + S_e \cos\beta\, t \\ y_{e(t)} &= y_{e(0)} + v_{ey}t = y_{e(0)} + S_e \sin\beta\, t \end{aligned} \tag{A7b}$$

The rotation matrix for weather is the standard affine transform

$$R(x, y, \theta) \text{ where } (x, y)_{new} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} . (x, y)_{old} \tag{A7c}$$

Given the standard ellipse (Eq. A3e), solving for y gives us two generators

$$y_{top\ half} = \frac{b\sqrt{a^2 - x^2}}{a} \text{ and } y_{bottom\ half} = -\frac{b\sqrt{a^2 - x^2}}{a}, \tag{A7d}$$

which we augment with a location parameter

$$y_{top\ half} = \frac{b\sqrt{a^2 - (x - x_{wx(0)})^2}}{a} \text{ and } y_{bottom\ half} = -\frac{b\sqrt{a^2 - (x - x_{wx(0)})^2}}{a} \tag{A7e}$$

All this allows us to create a *distance equation* at time t, from the moving aircraft to any given point (x_e, y_e) on the moving ellipse, although we must compute the top and bottom halves separately. For example,

$$d_{top\ half\,(t)} = \sqrt{\left(x_{ac(t)} - x_{e(t)}\right)^2 - \left(y_{ac(t)} - y_{e(t)}\right)^2} \tag{A7f}$$

$$\text{where } \left(x_{e(t)}, y_{e(t)}\right) = R\left(\left(x_{e(t)} - \left(x_{e(0)} + S_e \cos\beta\, t\right)\right), \left(y_{e(0)} - \left(y_{e(0)} + S_e \sin\beta\, t\right)\right), \theta\right) + \left(x_{e(t)}, y_{e(t)}\right) \tag{A7g}$$

Eq. A7g effectively translates the moving ellipse's center at time t back to (0,0), rotates it by θ, then retranslates it back to where it was originally at time t.

Ideally, we would like to have a closed-form solution to minimize the distance equation A7f over the entire range of t. However, A7f-g cannot be easily solved by the usual method of setting the function derivative to zero and solving for t. The function itself is too complicated. Fig. A6 shows that computation of d_{min} even to a single moving, rotated ellipse at a single value of t will be complicated, and that the overall error surface of Eq. A7f-g will be multidimensional, shallow, and scalloped.

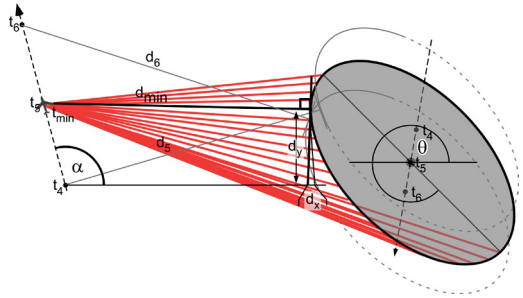


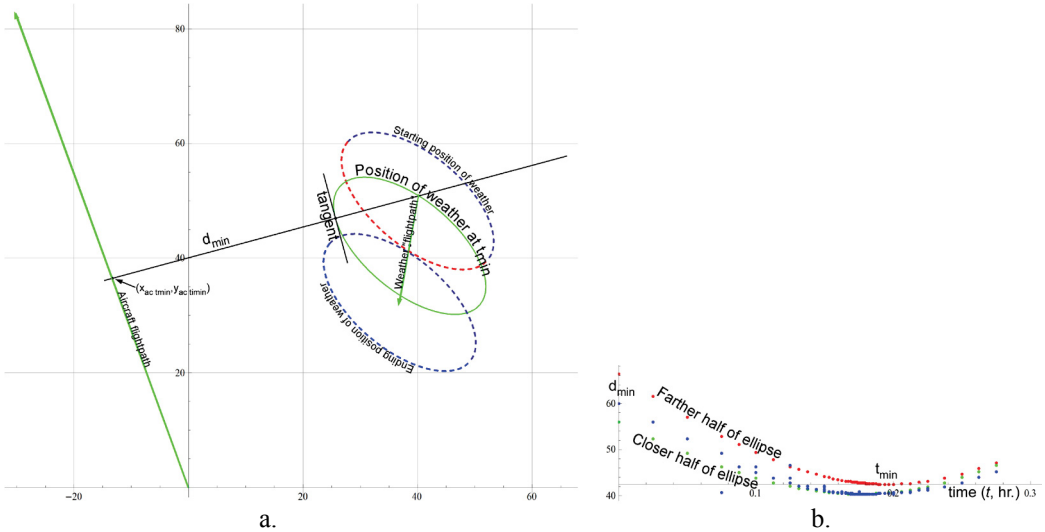
Fig. A6. Computing distance to a rotated ellipse results in a scalloped multi-D error surface at each value of t.

Fortunately, we can still solve the problem numerically, to within a reasonable error tolerance value.

A8. Verifying solution by numerical methods. It is always comforting to graphically see that solution by numerical methods looks correct. As Fig. A6 shows, we know that, for any given point on the aircraft flightpath, the minimum distance d_{min} to the ellipse will always form a ray that hits the ellipse boundary at a right angle to the tangent at that point. So, if we draw a picture of the situation at time t, we should see the normal to the tangent on the ellipse at our solution point running straight through the point on the aircraft vector where the aircraft would be at that time. That is a necessary condition of accuracy (but not sufficient, because every instantaneous value of t will have its own $d_{min(t)}$).

We therefore test this by having *Mathematica* draw sample plots. For the initial test cases, we can assume no winds aloft. Fig. A7a shows one with geometry similar to our example of Fig. A5. Note that the scenario-wide minimum distance $d_{min(tmin)}$ correctly follows the normal to the tangent, and correctly intersects the aircraft flightpath at the proper point ($x_{ac(tmin)}$, $y_{ac(tmin)}$).

We can verify sufficiency by simply plotting out the individual values of d_{min} as we move across t. Fig. A7b shows this done for Fig. A7a. Together, these two plots demonstrate the accuracy of our intended scoring method to a tolerance of about $t=t_{min}$ ±1 second.



a. b.

Fig. A7. Visually verifying the computation of d_{min} by numerical methods: a) *Mathematica* rendering of a moving aircraft and moving, rotated ellipse; b) The resultant sampling points for $d_{min(t)}$.

As Fig. A7b shows, sampling density is concentrated near the scenario-wide t_{min}. This can be accomplished iteratively, by sampling d_{min} first across the entire aircraft flightpath at a fixed time granularity dt, then focusing each subsequent iteration in on the lowest found value of d_{min}, halving both the scan range of t and the value of dt for that iteration. This process is repeated until $dt < 1/3600$ hr. This method, while far from efficient, does ensure finding a global minimum for d_{min}, meaning that the tests will be scored very accurately.

A9. Correction for winds aloft. We have to account for winds aloft, if we hope to achieve the highest degree of realism and accuracy. So, let us begin by calculating the effect that a single wind vector should have on the aircraft. Fig. A8 shows the angles and vectors involved. Essentially, what we want to calculate is the *effect of wind on aircraft groundspeed*, given that we wish to fly along some desired course with original airspeed s, (which, in the absence of wind, would produce the velocity component vectors V_x and V_y).

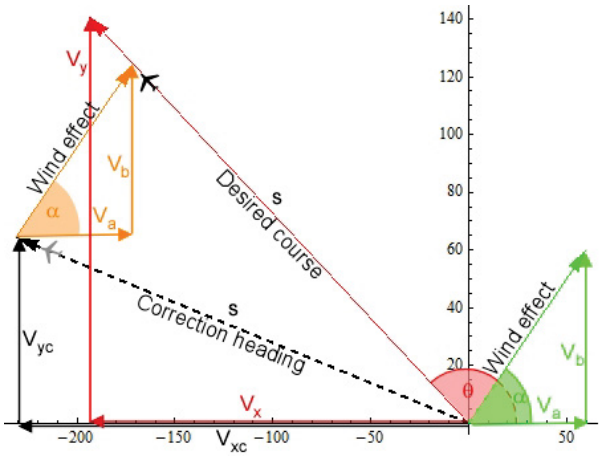


Fig. A8. Component single-source wind vectors affecting the aircraft's airspeed (hence, groundspeed).

The approach we take can be understood by imagining "turning off the wind," and flying along an appropriate correction heading—a vector $\{V_{xc},V_{yc}\}$—for a fixed period of time t (say, one hour) at airspeed s. After that, imagine "turning the wind back on," having the aircraft behaving like a balloon, and simply drifting along, pushed by the wind component vectors V_a and V_b for one hour. As Fig. A8 shows, the aircraft would end up back at some point on the desired course. That point would be:

$$\{V_a + V_{xc}, V_b + V_{yc}\}t \tag{A9a}$$

What we must do is figure out V_{xc} and V_{yc}, based on the available information. We know that

$$\tan\theta = \frac{V_y}{V_x} = \frac{V_{yc}+V_b}{V_{xc}+V_a} \text{ and } s^2 = V_x^2 + V_y^2 = V_{xc}^2 + V_{yc}^2 \tag{A9b}$$

so $$V_{yc} + V_b = \frac{V_y}{V_x}\left(V_{xc} + V_a\right) \tag{A9c}$$

making $$V_{yc} = \frac{V_y}{V_x}\left(V_{xc} + V_a\right) - V_b \tag{A9d}$$

Squaring both sides gives $$V_{yc}^2 = \left(\frac{V_y}{V_x}\left(V_{xc} + V_a\right) - V_b\right)^2 \tag{A9e}$$

Previously, we noted that $$s^2 = V_{xc}^2 + V_{yc}^2 \tag{A9f}$$

Substituting for V_{yc} gives $$s^2 = V_{xc}^2 + \left(\frac{V_y}{V_x}\left(V_{xc} + V_a\right) - V_b\right)^2 \tag{A9g}$$

This sets up the possibility of solution by the quadratic formula, provided we can rearrange terms. Expanding Eq. A9g gives

$$s^2 = V_{xc}^2 + V_b^2 - \frac{2V_aV_bV_y}{V_x} - \frac{2V_bV_{xc}V_y}{V_x} + \frac{V_a^2V_y^2}{V_x^2} + \frac{2V_aV_{xc}V_y^2}{V_x^2} + \frac{V_{xc}^2V_y^2}{V_x^2} \tag{A9h}$$

$$0 = -s^2 + V_{xc}^2 + V_b^2 - \frac{2V_aV_bV_y}{V_x} - \frac{2V_bV_{xc}V_y}{V_x} + \frac{V_a^2V_y^2}{V_x^2} + \frac{2V_aV_{xc}V_y^2}{V_x^2} + \frac{V_{xc}^2V_y^2}{V_x^2} \tag{A9i}$$

$$0 = \frac{V_x^2\left(-s^2 + V_{xc}^2 + V_b^2\right) - 2V_xV_aV_bV_y - 2V_xV_bV_{xc}V_y + V_a^2V_y^2 + 2V_aV_{xc}V_y^2 + V_{xc}^2V_y^2}{V_x^2} \quad \text{(A9j)}$$

$$0 = V_x^2\left(-s^2 + V_{xc}^2 + V_b^2\right) + V_{xc}^2V_y^2 + 2V_aV_{xc}V_y^2 - 2V_xV_bV_{xc}V_y + V_a^2V_y^2 - 2V_xV_aV_bV_y \quad \text{(A9k)}$$

$$0 = V_x^2V_{xc}^2 + V_x^2\left(-s^2 + V_b^2\right) + V_{xc}^2V_y^2 + 2V_aV_{xc}V_y^2 - 2V_xV_bV_{xc}V_y + V_a^2V_y^2 - 2V_xV_aV_bV_y \quad \text{(A9l)}$$

gathering terms, $$0 = \left(V_x^2V_{xc}^2 + V_{xc}^2V_y^2\right) + \left(2V_aV_{xc}V_y^2 - 2V_xV_bV_{xc}V_y\right) + \left(V_x^2\left(-s^2 + V_b^2\right) + V_a^2V_y^2 - 2V_xV_aV_bV_y\right) \quad \text{(A9m)}$$

$$0 = \left(V_x^2 + V_y^2\right)V_{xc}^2 + \left(2V_aV_y^2 - 2V_xV_bV_y\right)V_{xc} + \left(V_x^2\left(-s^2 + V_b^2\right) + V_a^2V_y^2 - 2V_xV_aV_bV_y\right) \quad \text{(A9n)}$$

which now fits the quadratic formula form $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ with $a = V_x^2 + V_y^2 = s^2$, (A9o)

$$b = 2\left(V_aV_y^2 - V_xV_bV_y\right),$$

and $$c = V_x^2\left(-s^2 + V_b^2\right) + V_a^2V_y^2 - 2V_xV_aV_bV_y$$

We can now solve for V_{xc} by substituting terms into the quadratic formula

$$V_{xc} = \frac{-2\left(V_aV_y^2 - V_xV_bV_y\right) \pm \sqrt{\left(2\left(V_aV_y^2 - V_xV_bV_y\right)\right)^2 - 4\left(V_x^2 + V_y^2\right)\left(V_x^2\left(-s^2 + V_b^2\right) + V_a^2V_y^2 - 2V_xV_aV_bV_y\right)}}{2\left(V_x^2 + V_y^2\right)} \quad \text{(A9p)}$$

which, after simplification, becomes

$$V_{xc} = \frac{V_y\left(V_xV_b - V_aV_y\right) \pm \sqrt{V_x^2\left(s^4 - \left(V_bV_x - V_aV_y\right)^2\right)}}{s^2} \quad \text{(A9q)}$$

with V_{yc} being $$V_{yc} = \sqrt{s^2 - V_{xc}^2} \quad \text{(A9r)}$$

The new wind-modified airspeed along the original flight vector $\{V_x, V_y\}$ will be

$$s = \sqrt{\left(V_{xc} + V_a\right)^2 + \left(V_{yc} + V_b\right)^2} \quad \text{(A9s)}$$

Unfortunately V_{xc} and V_{yc} are now quadrant-dependent, so we must keep track of that as we write our computer code (see Appendix B).

A10. Adjusting groundspeed for effect of multiple weather cells. We also have to try to account for the effect of having multiple weather cells aloft. This is no mean feat, since weather is technically chaotic, and any attempt we make at regularization is bound to be flawed. Nonetheless, we should note that even the most sophisticated flight simulators very, very rarely make any correction whatsoever for multiple convective cells. So, any attempt we make is arguably better than the vast majority of currently accepted test platforms, which do nothing more than represent large-scale winds as uniform, straight-line vectors.

We know a priori that the aircraft and the weather move independently, and, winds aloft—from the aircraft's perspective—change instantaneously as both the aircraft's position *and* the weather cells' positions change.

We therefore compute an estimate, based on a set of differential equations that weights the effect of each of two weather cells on the aircraft's airspeed, according to its instantaneous closest distance from "orange" boundary to the aircraft (which we have to compute anyway). Essentially, the farther the weather cell from the aircraft, the less effect it will have on the aircraft's airspeed. Fig. A9 illustrates.

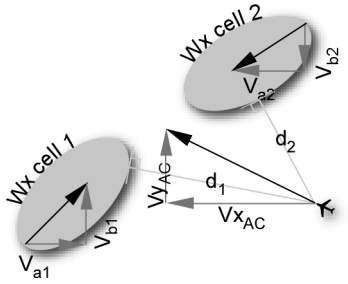


Fig. A9. Component multiple-source wind vectors affecting the aircraft's airspeed (hence, groundspeed) at time *t*.

This will result in an adjusted set of *x, y* components for "local weather velocity at the aircraft" at the *i*th time step *t(i)*,

$$Va_{adj(t(i))} = \frac{d_{2(t(i))}Va_{1(t(i))} + d_{1(t(i))}Va_{2(t(i))}}{d_{1(t(i))} + d_{2(t(i))}} \tag{A10a}$$

$$Vb_{adj(t(i))} = \frac{d_{2(t(1)}Vb_{1(t(i))} + d_{1(t(i))}Vb_{2(t(i))}}{d_{1(t(i))} + d_{2(t(i))}} \tag{A10b}$$

where $\{Va_{1t(i)}, Vb_{1t(i)}\}$ and $\{Va_{2t(i)}, Vb_{2t(i)}\}$ are the instantaneous velocity vectors for weather cell 1 and 2 at *t(i)*, and $d_{1(t(i))}$ and $d_{2(t(i))}$ are the respective minimum distances from the aircraft to the "orange" ellipses of weather cells 1 and 2, also at *t(i)*. You can see that this is simply a set of two normalized *x*- and *y*-components, each of which is weighted by the minimum distance to the *opposite* weather ellipse. This effectively weights higher the closer cell's effect on aircraft speed, then normalizes both effects by the denominator $d_{1(t(i))} + d_{2(t(i))}$.

Together, A10a and A10b can then behave as substitutes for V_a and V_b used formerly in Section A9.

A11. Correction for flight segment end effects. Additionally, we add a small correction to compensate for the error at the end of each flightpath segment, now introduced by using differential equations. Otherwise, the very last fraction of a time step incremented at each segment's end will cumulate, the more segments we examine. As we compute the starting time for the next segment, we simply subtract from the prior total time an estimate of how long it would take to traverse that tiny remaining distance, given the latest estimate of aircraft speed. Appendix B shows the code.

A12. Effect of A9-11 on overall method. Having to adjust for instantaneous airspeed effectively eliminates any elegant time-based methods of computing overall minimum distance to the weather cells. In the no-wind condition, we could home in on the one segment we knew contained the minimum distance, and then, decrease the size (granularity) of time steps (*dt*) while decreasing the sideways distance we examined on the flightpath. Now, A9-11 reduces us to brute-force computation, time step-by-time step. Nonetheless, albeit slow, we still expect the method to be quite accurate, and faster than any pixel-by-pixel computation of PCA.

A13. Validation of logical correctness. The odd-shaped flightpath in Fig. A10 is a quadrant check, designed to show that the program gives logically reasonable groundspeed estimates throughout all four geometric quadrants.

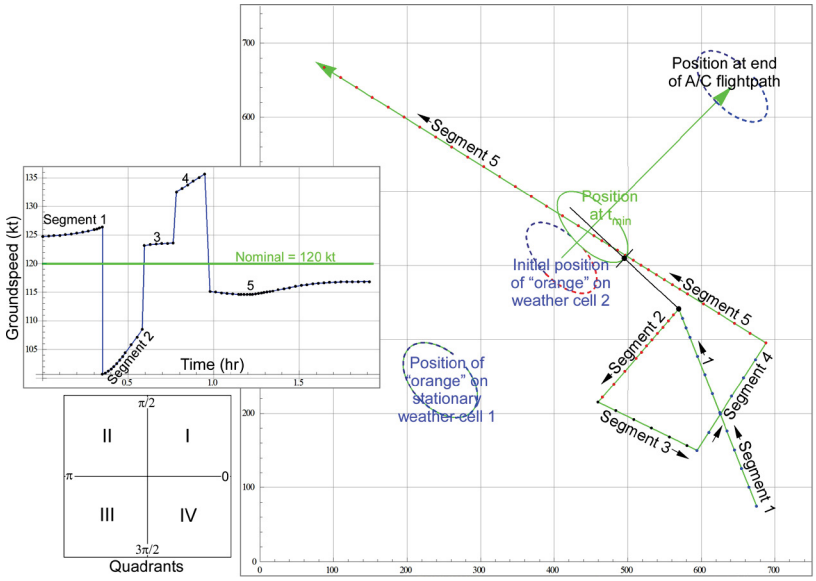


Fig. A10. A quadrant check shows logically expected effects of a moving weather cell on groundspeed.

Here, weather cell 1 is stationary, while weather cell 2 is moving at 23 kts at an angle of 45° ($\pi/4$, in standard Cartesian coordinates). Nominal aircraft airspeed/no-wind groundspeed is 120 kt.

Everything checks as logically expected. At an angle of approximately 105°, flight segment 1 tests logical correctness of Eqs. A10a-b in quadrant II. We see an expected slight increase in groundspeed, due to an oblique tailwind the aircraft would be experiencing.

Similarly, segment 2 (≈225°) tests quadrant III—straight into a headwind—and we see an expected severe drop in groundspeed. As expected, this drop starts out slightly less than the speed of weather cell 2, because the aircraft's distance from cell 2 mitigates it (see Eqs. A10a-b). And, the size of the drop decreases with time, because weather cell 2 and the aircraft are moving farther apart, so cell 2 exerts less of an effect on the aircraft.

Segment 3 doubles back (≈345°), testing quadrant IV, and we again see a slight expected oblique-tailwind speed boost. Segment 4 (≈50°) tests quadrant I, and has a strong tailwind boost that actually increases over time, because the aircraft is getting closer to cell 2.

Finally, segment 5 again tests quadrant II, but at a greater angle (≈140°), resulting in a slight oblique headwind, and expected slight speed drop.

To test correctness for both weather cells in motion simultaneously, Figs. A11b-c show that the program works properly with a variety of flightpaths in a scenario where a gap between two weather cells was closing at a rate of about 24 sm/hr. First, the algorithm is properly registered—weather appears where it was designed to, moves where it was designed to, and looks the same in all three programs, the Generator, the Viewer, and the Autoscorer. Meanwhile, the flightplan generated by the Viewer reoccurs properly in the Autoscorer.

Second, the geometry is correct, in that a normal ray drawn from the tangent on the ellipse at the point of closest approach properly intersects the position of the aircraft at time t_{min}.

Third, Runs 1-6 show that the algorithm can digest flightplans with as few as one segment (Run 3) and as many as 20 (Run 6).

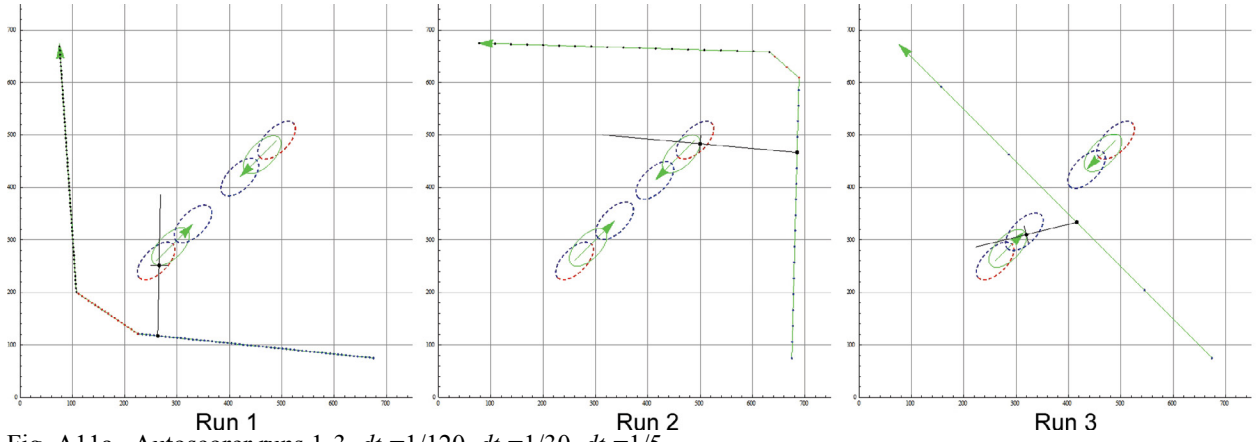


Fig. A11a. Autoscorer runs 1-3. dt_1=1/120, dt_2=1/30, dt_3=1/5.

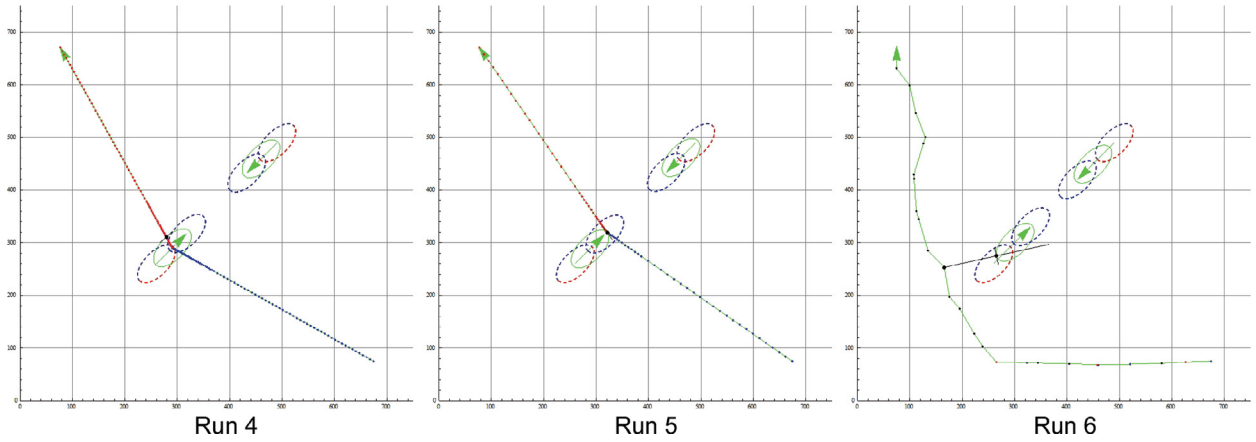


Fig. A11b. Autoscorer runs 4-6. dt_4=1/120, dt_5=1/60, dt_6=1/15.

A14. Selection of time-step size (dt). Our new method may be slower, but it does not have to be inaccurate. The issue centers around the size of *dt*. Infinitely small deltas produce infinitely good accuracy, but at the cost of infinite computation time. The issue in speed-accuracy tradeoff is always "How good is good enough?"

For our purposes, computational accuracy arguably needs to be only about ± 0.25 screen-coordinate unit, because that is about the size of one pixel, and sets the limit for visual accuracy. Moreover, we will eventually add noise to the weather, so subjects will never see perfect ellipses anyway. Ellipses are meant as mean likelihood estimates to an ideal observer. Finally, since our statistics will operate on rank-orders, individual measurements need only be sufficiently accurate to minimize changes in rank.

To bootstrap proper granularity, we initially scored a few expected flightpath types, shown as Fig. A11. These preliminary tests indicated that *dt* = 1/5 to 1/120 hr was a reasonable range to examine.

Next, we tweaked the algorithm. One way to approach speed-accuracy is to modulate *dt* as the virtual aircraft approaches the ellipse. The closer it gets, the smaller we make *dt*. That way, when the aircraft is far from weather, many large flightpath steps can be covered quickly with little sacrifice in accuracy, yet accuracy will still be high when d_{min} approaches 0.

Following this logic, Table A1 shows results of the simple hybrid algorithm underlying Runs 1-6, where aircraft proximity to "orange" is based on a log 3 scale, and *dt* is on a log 2 scale.

dt_0 = k;
If $d_{(t)}$ > 72.9 Then $dt = dt_0$,
Else If $d_{(t)}$ > 24.3 Then $dt = dt_0/2$,
Else If $d_{(t)}$ > 8.1 Then $dt = dt_0/4$,
Else If $d_{(t)}$ > 2.7 Then $dt = dt_0/8$,
Else If $d_{(t)}$ > .9 Then $dt = dt_0/16$,
Else If $d_{(t)}$ > .3 Then $dt = dt_0/32$,
Else If $d_{(t)}$ > .1 Then $dt = dt_0/64$,
Else $dt = dt_0/128$

Table A1 show results. Runs 1-3 show that, for "shallow" runs (flightpaths that stay more than about 25 screen coordinate units[1] from the ellipse), as *dt* changes, d_{min} and the total elapsed flight time *t* are relatively stable and change little. Therefore, *dt* ≥1/5 hr (i.e., greater than one sample per 12 minutes) is arguably accurate enough for flightpaths that never get too close to dangerous weather.

Table A1. Effect of time-step granularity (*dt*, in hr) on number of samples generated (*N*), minimum distance to weather (d_{min}, in screen-coordinate units), flight time (*t*, in hr) and scoring runtime (*T*, in sec).

	Run 1 (around the left)				**Run 2** (around the right)				**Run 3** (in between cells)			
dt	**N**	**d_{min}**	**t**	**T**	**N**	**d_{min}**	**t**	**T**	**N**	**d_{min}**	**t**	**T**
1/5	7	151.423	1.190	20	8	187.983	1.274	18	5	**98.818**	0.927	13
1/15	19	144.270	1.193	54	21	188.239	1.277	48	14	76.869	0.927	35
1/30	37	144.272	1.194	98	40	185.902	1.278	93	28	76.869	0.927	64
1/60	73	144.079	1.194	198	78	185.914	1.278	158	56	76.869	0.927	135
1/120	145	143.994	1.172	405	155	185.920	1.278	306	112	76.751	0.927	279

Table A1 (cont.).

	Run 4 (deep penetration)				**Run 5** (graze)				**Run 6** (20-segment flightpath)				**Run 7** (wind shear)			
dt	**N**	**d_{min}**	**t**	**T**	**N**	**d_{min}**	**t**	**T**	**N**	**d_{min}**	**t**	**T**	**N**	**d_{min}**	**t**	**T**
1/5	8	**22.323**	0.960	19	9	0.684	0.940	24	20	101.924	1.174	55	8	6.518	0.924	17
1/15	27	0.335	0.961	70	26	0.258	0.941	74	25	101.932	1.174	69	23	5.814	0.924	51
1/30	50	0.127	0.962	127	55	0.023	0.941	156	43	101.956	1.174	118	46	5.846	0.924	103
1/60	104	0.009	0.962	264	111	0.0012	0.941	307	78	101.967	1.175	218	90	5.856	0.924	202
1/120	208	0.003	0.962	**534**	224	0.0087	0.942	**636**	150	101.911	1.175	396	179	5.862	0.924	396

Run 4 is a harsher test, in that we know that actually penetrating our target (orange) area should result in d_{min} = 0. So, how close to 0 we score is a key indicator of acceptable size for *dt*. We expect this to be a rare event, but the algorithm must be able to handle it.

Run 4 shows evidence of failure for *dt* = 1/5. Flight time *t* is stable, but d_{min} = 22.323 is clearly inaccurate. There just are not enough sampling points. Nonetheless, decreasing *dt* all the way down to 1/120 carries a stiff execution time penalty (T=534 sec). Run 5—a "graze"—supports this conclusion (T = 636). Here, we might argue that *dt* = 1/30, carries us beyond our tolerance level of 0.25 screen units accuracy.

In the end, the secret is, of course, to run a hybrid algorithm such as the one previously described, which sets *dt* coarsely while the aircraft is far from weather, and progressively shrinks *dt* as the distance to weather decreases. In this way, we maximize computational speed where there is no great threat to accuracy, and temporarily maximize accuracy only where needed, minimizing the burden on computational speed.

Run 6 is a test of the special case where a pilot clicks many times onscreen, creating a flightpath with many short segments (here, 20). The algorithm appears to digest this and still give accurate results for *dt* ≥ 1/30.

[1] In this particular instance, 6.618 screen coordinate units equaled 1.0 sm on our map. Therefore, a "shallow" run is ≥ 4 sm from "orange," and our tolerance floor of 0.25 SCU equals about 0.378 sm, or 199.5' real-world accuracy.

APPENDIX B1
Weather Generator Mathematica Code

```
(* This allows generation of weather movies *) (* Paul's Valley to Yukon is about 70 statute miles. *)
Block[{(* Put any variables here that you want to remain local. However, they won't penetrate to the saveData function *)},
  (* File name convention will be  EXPT.SUBJ.SCENARIO.parms   for parameters,  e.g.   01.xx.03.parms
                                and  EXPT.SUBJ.SCENARIO.framesN  for frames,      e.g.,   01.xx.03.frame1.jpg   *)
  exptID = "E1"; (* experiment id--in a series of experiments, which was this? *)
  subjID = "Sxx";  (* which subject/participant?  *)
  scenID = "12";  (* which experimental scenario was this? *)
  time = Timing[
      dir = 1;
      Switch[dir,
        1, SetDirectory["\\\\Avsaaccama0\\aam500\\wknecht\\New Projects\\Creating programmable weather\\"],
        2, SetDirectory["C:\\Billy Bob\\Misc\\Junk\\"],
        3, SetDirectory["C:\\Looping NEXRAD\\"]
      ];
      (*FileNames[];*)
      (*------------------------BEGIN INPUT PARAMETERS------------------------*)
      Gsize = 150;        (* Granularity (grid size) of the overall square display (no. of cells in x and y) Type: INTEGER *)
      n = 12;             (* Number of frames to be generated to represent a 1-hr movie INTEGER *)
      (* Below, keep in mind that the positions of wx cells start out in grid coordinates, NOT screen coordinates *)
      yxWx1 = {45, 45};       (* y,x position (row, column) of the center of the 1st wx cell in the 1st frame INTEGER *)
      yxWx2 = {140, 140}; (* {0,0} is lower, L corner of the drawing window*)
      (*yxWx2 = {Gsize-yxWx1[[1]],Gsize-yxWx1[[2]]};   Position of the 2nd wx cell in the first frame  *)
      Print["yxWx2 = ", yxWx2];
      σ1x = 14;   σ1y = 6;      (* The x and y axes of the two unrotated wx cells REAL NUMBER*)
      σ2x = 18;   σ2y = 6;      (* This will control the a (major) and b (minor) axes of the wx cells *)
      θ1 = N[Pi/4];   θ2 = N[Pi/4]; (* Rotation of weather cells REAL *)
      dx1 = 0;    dy1 = 0;    (* THESE MUST BE INTEGERS !!! These delta values control horizontal & vertical movement of wx cells *)
      dx2 = -1;   dy2 = -1;  (* THESE MUST BE INTEGERS !!! dx1=2;   dy1=1;  dx2=-2;  dy2=-1; used for wind shear test *)
      rad = 36;               (* How big an area (radius) around each wx cell to add noise. Should be about 3σ INTEGER *)
      frames = Table[0, {i, 1, n}, {r, 1, Gsize}, {c, 1, Gsize}] ; (* r is rows (y), c is columns (x) *)
      noise = 0.10;           (* Amount of color noise to be added to our 3D wx cells *)
      zOrange = .75;          (* The "critical value" for z in the 2D Gaussian, that pilots must avoid.  0<z<1*)
      index = 1;   msg = ""; pathLength = 555.555; (* 555.555 would denote a save w/o any screen input *)
```

```
(*----------------------------END INPUT PARAMETERS-----------------------*)
mapName = "Map of OKC-Tulsa 700x700.jpg";  (*mapName="Map of OKC-Tulsa area 3.jpg"; 750x750 *)
gBackgrd = Import[mapName];
imgSize = Import[mapName, "ImageSize"]; (* Get image size, in pixels, {x,y} *)
offsetX = offsetY = 75; (* SCU. Used to position AC starting position (startPt) and destination (endPt) while generating images *)
startPt = {imgSize[[1]] - offsetX, offsetY};  destnPt = {offsetX, imgSize[[2]] - offsetY}; (* {0,0} is in lower left of the screen *)
(*movieID=Input["Enter the movie identifier",movieID=Null];   Get user input to create file identifier *)
mid = Round[Gsize/2]; (* Middle of the display, in grid-coordinate units *)
R[r_, c_, θ_] := N[{r, c}.{{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}}, 3];  (*Rotation matrix. Allows wx cells to be rotated *)
(*If[yxWx1[[1]]-rad<1||yxWx1[[2]]-rad<1||yxWx2[[1]]-rad>Gsize||yxWx2[[1]]+rad>Gsize,
    Print["Trying to write beyond bounds in frame 1. Exiting."]; Exit[]];*)
a1 = Sqrt[-2 σ1x^2 Log[zOrange]] ;  b1 = Sqrt[-2 σ1y^2 Log[zOrange]] ;  a2 = Sqrt[-2 σ2x^2 Log[zOrange]] ; b2 = Sqrt[-2 σ2y^2 Log[zOrange]] ;
(*----------------------------generate 1st wx cell-----------------------*)
incrX = incrY = 0;
maxF = N[PDF[NormalDistribution[0, σ1x], 0] * PDF[NormalDistribution[0, σ1y], 0 ] ] ; (* Find the highest point in the curve *)
For[i = 1, i ≤ n, i++, Print["Generating cell 1, table ", i];
    For[c = yxWx1[[1]] + incrX - rad, c ≤ yxWx1[[1]] + incrX + rad,  c++,
      (* Generate the first wx cell. "c" is "column" (i.e., x). "r" is "row" (i.e., y) *)
      For[r = yxWx1[[2]] + incrY - rad, r ≤ yxWx1[[2]] + incrY + rad,  r++, (* Translate the x, y position of the cell's center,
        and  bracket it by a in y, and b in x *)
        If[1 ≤ c ≤ Gsize && 1 ≤ r ≤ Gsize,  (* Protect against writing beyond array bounds *)
          x = R[c - yxWx1[[1]] - incrX, r - yxWx1[[2]] - incrY, θ1][[1]];   (* Calculate rotated x and y *)
          y = R[c - yxWx1[[1]] - incrX, r - yxWx1[[2]] - incrY, θ1][[2]];
          temp = N[PDF[NormalDistribution[0, σ1x], x] * PDF[NormalDistribution[0, σ1y], y ] / maxF];
          frames[[i, r, c]] = temp * (1 + RandomReal[{-0.3, noise}]) + RandomReal[{0, noise}]
          (* This loads noise more heavily toward higher values *)
        ]
      ]];
    incrX += dx1;  incrY += dy1;
  ];
yxWx1end = yxWx1 + {incrY, incrX}; (* The LAST wx cell generated here will be the FIRST one, for scoring purposes *)
```

```
(*----------------------------generate 2nd wx cell----------------------*)
incrX = incrY = 0;
maxF = N[PDF[NormalDistribution[0, σ2x], 0] * PDF[NormalDistribution[0, σ2y], 0]] ;
For[i = 1, i ≤ n, i++, Print["Generating cell 2, table ", i];
    For[c = yxWx2[[1]] + incrX - rad, c ≤ yxWx2[[1]] + incrX + rad, c++,   (* Generate the second wx cell *)
      For[r = yxWx2[[2]] + incrY - rad, r ≤ yxWx2[[2]] + incrY + rad, r++,
        If[1 ≤ c ≤ Gsize && 1 ≤ r ≤ Gsize,
          x = R[c - yxWx2[[1]] - incrX, r - yxWx2[[2]] - incrY, θ2][[1]];
          y = R[c - yxWx2[[1]] - incrX, r - yxWx2[[2]] - incrY, θ2][[2]];
          temp = N[PDF[NormalDistribution[0, σ2x], x] * PDF[NormalDistribution[0, σ2y], y ] / maxF];
          If[frames[[i, r, c]] < temp, frames[[i, r, c]] = temp * (1 + RandomReal[{-0.2, noise}]) + RandomReal[{0, noise}]]
        ]
      ]];
    incrX += dx2;  incrY += dy2
  ];
yxWx2end = yxWx2 + {incrY, incrX};
(*----------------------------------------------------------------------*)
For[i = 1, i ≤ n, i++, frames[[i]] /= Max[frames[[i]]];  Chop[frames[[i]], 10^-6]];  (* normalize & force small values to zero *)
myCF = Function[z,
      If[z < 0.03, RGBColor[1, 1, 1],
        If[z < 0.20, RGBColor[0, 0.9, 0],
          If[z < 0.3, RGBColor[0, 0.75, 0],
            If[z < 0.4, RGBColor[0, 0.4, 0],
              If[z < 0.6, RGBColor[1, 1, 0],
                If[z < zOrange, RGBColor[1, 0.5, 0],  (* This is orange, the color pilots must avoid *)
                  RGBColor[1, 0, 0]]]]]]];

images = {};
```

```
For[i = 1, i ≤ n, i++,
    Print["Generating frame ", i];
    gWx = Rasterize[ListContourPlot[frames[[i]], Contours → 8, ContourLines → False, ContourLabels → False, InterpolationOrder → 1,
          ColorFunctionScaling → True, PlotRange → All, PlotRangePadding → None, ColorFunction → myCF, ImageSize → imgSize,
          PerformanceGoal → "Quality", Frame → False], ImageResolution → 40];
    gWxb = Blur[gWx, 2];
    g1 = ImageApply[Min, gWxb];
    g2 = ColorNegate[g1];
    g3 = ImageSubtract[gBackgrd, g2];
    g4 = ImageSubtract[gWx, g1];
    g5 = ImageAdd[g4, g3];
    g6 = ImageCompose[gBackgrd, {g5, 0.7}, {Center, Center}]; (* ImageCompose[image,overlay,position] *)
    AppendTo[images, g6]
  ]; Print["Generation time = ", time]; (* End Timing *)

saveData[] := (* Always coordinate this Module with the Viewer and Scorer programs to ensure identical variables are being used *)
    Module[{}, (* This writes a header line with variable names, plus a data line with actual values for each variable *)
          filename1 = StringJoin["Wx movies\\", exptID, ".", subjID, ".", scenID, ".frame"];
          filename2 = StringJoin["Wx movies\\", exptID, ".", subjID, ".", scenID, ".parms"];
          header = {"E", "S", "Sc", "PresnOrder", "Map", "MapJPGSizePix", "MapSizeNM", "ScreenSizeCM", "ScreenResXY",
        "a1", "b1", "a2", "b2", "Gsize", "n", "yxWx1", "yxWx2", "yxWx1end", "yxWx2end", "σ1x", "σ1y", "σ2x", "σ2y", "θ1",
        "θ2", "dx1", "dy1", "dx2", "dy2", "rad", "noise", "zOrange", "startPt", "destnPt", "airSpdACkt", "frmsViewed", "EThr",
        "pathLennm", "minDnm", "waypts"};
          (* Always coordinate this header with the Viewer and Scorer programs to ensure identical variables are being used *)
          parms = {exptID, subjID, scenID, "999", mapName, imgSize, 0, {0, 0}, {0, 0}, a1, b1, a2, b2, Gsize, n, yxWx1,
        yxWx2, yxWx1end, yxWx2end, σ1x, σ1y, σ2x, σ2y, θ1, θ2, dx1, dy1, dx2, dy2, rad, noise, zOrange, startPt, destnPt,
        999, 999, 999, 999, 999, {}};
          data = {header, parms};
          ptr = OpenWrite[filename2];
          Write[ptr, data];
          Close[filename2];
          For[i = 1, i ≤ n, i++, Export[StringJoin[filename1, ToString[i], ".jpg"], images[[i]], "JPEG"]];(* Save movie frames *)
          ];
(* End Module *)
```

```
DialogInput[{},
      Column[{Button[StringJoin["Save Wx Movie ", scenID], DialogReturn[res], ImageSize → Automatic],
          Animate[
                    Show[images[[frame]], Graphics[{PointSize[0.02], Black, Point[startPt], Point[destnPt],
                                                                PointSize[0.012], Red, Point[destnPt],
                                                                PointSize[0.01], Orange, Point[startPt],
                                                                Text[Style[Current, Large, White, FontFamily –> "Arial"], startPt – {0, 30}],
                                                                Text[Style[Destination, Large, White, FontFamily –> "Arial"], destnPt + {0, 30}]
                                                      }]
                        ],
                    {frame, 1, n, 1, AppearanceElements → {"ProgressSlider", "ResetPlayButton"}},
                    AnimationRepetitions → 3, AnimationRunning → True,  AnimationRate → 2, Deinitialization :→ (saveData[])
                    }], (* Deinitialization:→  simply will not work unless movieID is a global variable*)
      WindowSize → {All, All}
    ] (* End DialogInput *)
    ];(* End Block *)
yxWx1end
yxWx2end
```

APPENDIX B2
Viewer Mathematica Code

```
(* Keep in mind that screen coordinates start with 0,0 at the lower lefthand side. sm = statute miles, kt = knots *)
dir = 1;
Switch[dir,
    1, SetDirectory["\\\\Avsaaccama0\\aam500\\wknecht\\New Projects\\Creating programmable weather\\"];
       screenActualXYcm = {47.3, 30.2}, (* Actual monitor width, height in cm, which you measure & enter here. Laptop→ {34.3, 19.4} *)
    2, SetDirectory["C:\\Billy Bob\\Misc\\Junk\\"]; (* My home machine *)
       screenActualXYcm = {47.7, 29.7},
    3, SetDirectory["C:\\Looping NEXRAD\\"];
       screenActualXYcm = {30.4, 18.9}(* Lab notebookS ADMNLOSA-PC and xxx *)
  ];
(* File name convention will be  EXPT.SUBJ.SCENARIO.parms   for parameters,  e.g.   01.xx.03.parms
                                 and  EXPT.SUBJ.SCENARIO.framesN for frames,      e.g.,  1201.xx.03.frame1.jpg   *)
sID = $Failed; (* Input the user ID *)
While[sID == $Failed || sID == "", (* Keep putting up ID box until a response is made *)
            sID = DialogInput[{res = ""},
            Column[{"Enter your ID #", InputField[Dynamic[res], Number, ImageSize → {75, 25}],
                    Button["Save", DialogReturn[res], ImageSize → Automatic]
                    }]
                ]
            ]; (* End While [id= *)
Block[{temp},
  numScenarios = 12;
  exptID = "E1"; (* experiment id--in a series of experiments, which was this? *)
  subjID = ToString[If[sID < 10, subjID = NumberForm[sID, NumberPadding → "0"], subjID = sID]]; (* Subject ID, left-zero-padded *)
  airspeedACkt = 120; (* *1.151;  aircraft airspeed, in kt,. Was sm in a previous version *)
  mapSizeNM = 7.69 * 40 / 3.25; (* width  (in inches) of the square terrain map we're displaying, in nm. CHANGE THIS WHEN CHANGING THE MAP SIZE *)
  frameRate = 2; (* frames / sec that the an211imtion will run. See also "numFrames" *)
  ϵ1 = 6; (* error-tolerance value to filter out double-clicking onscreen*)
  ϵ2 = 8; (* error-tolerance value to define "capture zone" around the final click representing "Destination" *)
  screenInfo = SystemInformation["Devices", "ScreenInformation"]; (* Need screen dimensions, {x, y}, in pixels *)
  screenResXY = {screenInfo[[1, 2, 2, 1, 2]] - screenInfo[[1, 2, 2, 1, 1]], screenInfo[[1, 2, 2, 2, 2]] - screenInfo[[1, 2, 2, 2, 1]]};
  (* Monitor width/height in pixels *)
  correctionFactor = (screenResXY[[2]] / screenActualXYcm[[2]]) / (screenResXY[[1]] / screenActualXYcm[[1]]);
  (* Used to ensure a square image onscreen *)
  mySet = myRandomizedSet = {}; (* Below, we randomize the scenario presentation order *)
```

```
For[scenarioNum = 1, scenarioNum ≤ numScenarios, scenarioNum++,
    subID = ToString[If[scenarioNum < 10, subID = NumberForm[scenarioNum, NumberPadding → "0"], subID = scenarioNum]];
    (* Subject ID, left-zero-padded *)
    AppendTo[mySet, subID]
  ];
numLeft = numScenarios;
For[numLeft = numScenarios, numLeft > 0, numLeft--,
    index = RandomInteger[{1, numLeft}];
    AppendTo[myRandomizedSet, mySet[[index]]];
    mySet = Delete[mySet, index];
  ]; (* now, myRandomizedSet contains zero-padded strings, arranged in random order from 1 to numScenarios *)
For[scen = 1, scen ≤ numScenarios, Pause[3]; scen++,
  scenID = myRandomizedSet[[scen]]; (* Select the randomized scenario *)
  movieRoot = StringJoin["Wx movies\\", exptID, ".Sxx.", scenID, ".frame"];
  inFilename = StringJoin["Wx movies\\", exptID, ".Sxx.", scenID, ".parms"];
  outFilename = StringJoin["Data\\", exptID, ".", subjID, ".", scenID, ".dat"];
  ptr = OpenRead[inFilename];
  parameters = Read[ptr];
  Close[inFilename];
  pixName = StringJoin["Wx movies\\", exptID, ".", subjID, ".", scenID, ".frame1.jpg"];
  frmSize = parameters[[2, 6]]; (* Frame size, in pixels, {x,y} *)
  parameters[[2, 2]] = subjID;
  parameters[[2, 3]] = scenID;
  parameters[[2, 4]] = ToString[scen];
  parameters[[2, 7]] = mapSizeNM;
  nmToSCU = frmSize[[1]] / mapSizeNM; (* A conversion factor = screen coord units (SCU) per nautical mile on x-axis *)
  rrDiam = Round[airspeedACkt * nmToSCU]; (* Range-ring diameter, i.e., how many nm can A/C nominally travel in 1 hr, disregarding winds *)
  rrHalfDiam = Round[airspeedACkt * nmToSCU / 2];
  parameters[[2, 8]] = screenActualXYcm;
  parameters[[2, 9]] = screenResXY;
  Gsize = parameters[[2, 14]]; (* Grid size of the underlying terrain map *)
  numFrames = parameters[[2, 15]]; (* This represents number of frames in a 1-hr movie *)
  dx1 = parameters[[2, 26]];  dy1 = parameters[[2, 27]];  dx2 = parameters[[2, 28]];  dy2 = parameters[[2, 29]];
  temp = N[numFrames mapSizeNM / Gsize]; (* Calculate weather cell speed *)
  VxWx1 = temp dx1;  VyWx1 = temp dy1;  VxWx2 = temp dx2;  VyWx2 = temp dy2; (* Component velocities of weather cells *)
  startPt = parameters[[2, 33]]; destnPt = parameters[[2, 34]];
  pts = {startPt}; (* Create a file to hold the pilot's "flightpath" *)
  movie = {}; (* Load the movie frames *)
```

```
For[i = 1, i ≤ numFrames, i++,
    frameName = StringJoin[movieRoot, ToString[i], ".jpg"];
    AppendTo[movie, Import[frameName]];
    If[i == 1, frmSize = Import[frameName, "ImageSize"]] (* Extract image size *)
  ];
index = 1;   msg = ""; pathLengthSCU = 0; (* 0 would denote a save w/o any screen input *)
saveData[nFrames_, outFile_] :=
    Module[{}, (* Always coordinate this Module with the Wx generator and Scorer programs to ensure identical variables are being used *)
      (*header={"E","S","Sc","PresnOrder","Map","MapJPGSizePix","MapSizeNM","ScreenSizeCM","ScreenResXY","a1","b1","a2",
            "b2","Gsize","n","yxWx1","yxWx2","yxWx1end", "yxWx2end","σ1x","σ1y","σ2x","σ2y","θ1","θ2","dx1","dy1","dx2","dy2",
            "rad","noise","zOrange","startPt","destnPt","airSpdACkt","frmsViewed", "EThr", "pathLennm", "minDnm","waypts"};*)
      parameters[[2, 35]] = airspeedACkt;
      parameters[[2, 36]] = nFrames;
      pathLengthSCU = 0;  (* Will be in screen units *)
      For[i = 1, i < Length[pts], i++, pathLengthSCU += N[Sqrt[(pts[[i + 1, 1]] - pts[[i, 1]])^2 + (pts[[i + 1, 2]] - pts[[i, 2]])^2]]];
      parameters[[2, 38]] = Round[pathLengthSCU / nmToSCU, .001]; (* The flight path length, in nm *)
      parameters[[2, 40]] = pts;
      data = {parameters[[1]], parameters[[2]]};
      ptr = OpenWrite[outFile];
      Write[ptr, data];
      Close[outFile];
    ]; (* End Module *)
frameCounter = 0;  okToSaveFlag = False; (* Flag that prevents saving data before certain conditions have been met *)
showLast = False;
(* xxx Can I record the length of time the movie played and how long it took pilot to make a decision? NO, but can record # of frames viewed *)
(* time=AbsoluteTiming[ *) DialogInput[{},
  Row[{ Column[{   Text["1.  Watch the looping NEXRAD.", BaseStyle → {Medium, FontFamily → "Arial"}],
                   Text["2.  Freeze the NEXRAD at latest wx.", BaseStyle → {Medium, FontFamily → "Arial"}],
                   Row[{ Dynamic[Button["Freeze at latest wx", showLast = True, ImageSize → Automatic, Background → LightGreen]],
                         Dynamic[Button["Resume looping", showLast = False, ImageSize → Automatic, Background → LightRed]] }],
                   Text["3.  Left−click on−screen to select turn points ",  BaseStyle → {Medium, FontFamily → "Arial"}],
                   Row[{Text["4.  Save your data when finished.", BaseStyle → {Medium, FontFamily → "Arial"}],
                         Button["Save my data", If[okToSaveFlag, DialogReturn[99]], ImageSize → {100, 40}, Background → Green] }],
                   Row[{Text[StringJoin["Now running scenario   ", ToString[scen], "  (", scenID, ")"], BaseStyle → {Medium, FontFamily → "Arial"}]   }]
                   }],  (* End Column *)
```

```
Animate[
    EventHandler[
        If[showLast == False, whichFrame = 1 + Mod[NEXRAD - 1, numFrames];  frameCounter++, whichFrame = numFrames];
        Show[movie[[whichFrame]],
            Graphics[{PointSize[0.02], Black, Point[startPt], Point[destnPt],
                PointSize[0.012], Red, Point[destnPt],
                PointSize[0.012], Orange, Point[startPt],
                White,  Thick, Dashed, Arrow[ pts],
                Yellow, Circle[startPt, rrHalfDiam],  Circle[startPt, rrDiam], (* Range rings *)
                Text[Style[Current, Large, White, FontFamily -> "Arial"], startPt - {0, 30}, {0, 0}],
                Text[Style[Destination, Large, White, FontFamily -> "Arial"], destnPt + {0, 30}, {0, 0}] ,
                Text[Style[msg, 18, Yellow, FontFamily -> "Arial"], destnPt + {15, 0}, {-1, 0}]  }],
              AspectRatio → correctionFactor
        ],
        "MouseClicked" :> ( temp = MousePosition["Graphics"]; (*Below, protects against double-clicking *)
            (* Below, protects against clicking outside the image boundaries *)
            If[ 0 ≤ temp[[1]] ≤ frmSize[[1]] && 0 ≤ temp[[2]] ≤ frmSize[[2]] ,
              (*  Below, protects against double-clicki
ng onscreen, i.e. unintentionally short flight segments *)
                If[ √((temp[[1]] - pts[[index, 1]])^2 + (temp[[2]] - pts[[index, 2]])^2  > ϵ1,
                  AppendTo[pts, temp];
                  index++
                  ]
                ]; (* Code below triggers if mouse is clicked near Destination *)
                If[Abs[pts[[index, 1]] - destnPt[[1]]] < ϵ2 && Abs[pts[[index, 2]] - destnPt[[2]]] < ϵ2,
                  pts[[index]] = destnPt;
                  msg = "Please hit \"Save My Data\" button";
                  okToSaveFlag = True
                  ];
            )
    ], (* End EventHandler *)
```

```
    {NEXRAD, 1, Infinity, 1, AppearanceElements → "None"},
                AnimationRepetitions → ∞, AnimationRunning → True, AnimationRate → frameRate, DisplayAllSteps → False,
                ControlPlacement → Bottom, Deinitialization :→ (saveData[frameCounter, outFilename])
          ] (* End Animate *)

        }], (* End Column *)
    (* The following AppearanceElements are included in a default animator:"ProgressSlider","PlayPauseButton","FasterSlowerButtons",
    "DirectionButton". Additional possible elements include:"PlayButton","ResetPlayButton","PauseButton","ResetButton".*)
    WindowSize → {All, All}, WindowMargins → {{0, Automatic}, {Automatic, 0}}, WindowFrame → "Frameless", WindowFloating → True
    ]; (* End DialogInput *)
  (*] End AbsoluteTiming, which will include the time MMCA takes to generate the display *)
  If[res == 99 && Length[pts] < 2,
    DialogInput[DialogNotebook[{TextCell["Error--no waypoints recorded. Please tell experimenter"], Button["Proceed", DialogReturn[1]]}]]];
  ](* End For scen=1 *)
];(* End Block *)
```

APPENDIX B3
Autoscorer Mathematica Code

```
(* A moving aircraft , TWO moving, ROTATED weather cells, AND read the A/C path from a subject response file, emulating a piecewise function for the flightpath. NOW, we read in values for speeds, etc, from a parameter file*)
dir = 1; (* THIS CURRENTLY READS FROM THE "DATA" FILE, SO BE SURE TO HAVE A COPY OF THE FILE IN THAT DIRECTORY *)
Switch[dir,
    1, SetDirectory["\\\\Avsaaccama0\\aam500\\wknecht\\New Projects\\Creating programmable weather\\"],
    2, SetDirectory["C:\\Billy Bob\\Misc\\Junk\\"],
    3, SetDirectory["C:\\Looping NEXRAD"]
  ];
Block[{temp, x, i, dx, dy},
  exptID = "E1"; (* experiment id--in a series of experiments, which was this? *)
  subjID = "15"; (* which subject/participant?  Be sure to include leading zero, if < 10 *)
  scenID = "03"; (* which experimental scenario was this? Be sure to include leading zero, if < 10 *)

  xACt[t_] := xAC0 + VxACt t; (* x-position of aircraft at time t on a straight-line path, in screen coordinate units (SCU) *)
  yACt[t_] := yAC0 + VyACt t; (* y-position of aircraft at time t *)
  xWx1t[t_] := xWx10 + VxWx1 t; (* x-position of center of weather cell 1 at time t on a straight-line path*)
  yWx1t[t_] := yWx10 + VyWx1 t; (* y-position of center of weather at time t *)
  xWx2t[t_] := xWx20 + VxWx2 t; (* x-position of center of weather cell 2 at time t on a straight-line path*)
  yWx2t[t_] := yWx20 + VyWx2 t;
  R[x_, y_, θ_] := N[{{Cos[θ], -Sin[θ]}, {Sin[θ], Cos[θ]}}.{x, y}]; (*Rotation matrix. Accepts (x,y), returns rotated (x_θ,y_θ)={ x Cos[θ] -y Sin[θ], x Sin[θ] +y Cos[θ] } *)
  Rc[x_, y_, θ_] := N[{{Cos[θ], Sin[θ]}, {-Sin[θ], Cos[θ]}}.{x, y}]; (*Counter-rotation matrix *)
  (* To rotate the ellipse, we translate it to the origin (0,0), rotate it, then translate it back to its original position *)
xyTop[a_, b_, xc_, yc_, x_, tWx_, θ_] := N[{xc, yc} + Re[R[x - xc, b Sqrt[a^2 - (x - xc)^2]/a, θ]]]; (* Returns (x,y) for top half of the un-translated/rotated/re-translated ellipse *)
  xyBtm[a_, b_, xc_, yc_, x_, tWx_, θ_] := N[{xc, yc} + Re[R[x - xc, -b Sqrt[a^2 - (x - xc)^2]/a, θ]]]; (* Occasionally throws a tiny imaginary component, so we filter that out *)
  dTopT[a_, b_, xc_, yc_, xet_, tAC_, tWx_, θ_] := Module[{xyRot, d}, xyRot = xyTop[a, b, xc, yc, xet, tWx, θ]; d = Sqrt[(xAC0 - xyRot[[1]])^2 + (yAC0 - xyRot[[2]])^2]];
  (* Dist eq, wrt top (positive) half of ellipse, w A/C at (xACt[tAC],yACt[tAC])*)
  dBtmT[a_, b_, xc_, yc_, xet_, tAC_, tWx_, θ_] := Module[{xyRot, d}, xyRot = xyBtm[a, b, xc, yc, xet, tWx, θ]; d = Sqrt[(xAC0 - xyRot[[1]])^2 + (yAC0 - xyRot[[2]])^2]];
```

```
(* Distance equation, wrt bottom half of ellipse *)

btmSlope[a_, b_, xc_, yc_, x_, y_, tWx_, θ_] := Module[{tmp, m0, m1}, tmp = Rc[x - xc, y - yc, θ]; m0 = -(tmp[[1]]/tmp[[2]]) b^2/a^2; m1 = (m0 Cos[θ] + Sin[θ])/(Cos[θ] - m0 Sin[θ])];

topSlope[a_, b_, xc_, yc_, x_, y_, tWx_, θ_] := Module[{tmp, m0, m1}, tmp = Rc[x - xc, y - yc, θ]; m0 = -(tmp[[1]]/tmp[[2]]) b^2/a^2; m1 = (m0 Cos[θ] + Sin[θ])/(Cos[θ] - m0 Sin[θ])];

vxNew1[Vx_, Vy_, Va_, Vb_, s_] := (Vy (Vx Vb - Va Vy) - Sqrt[Vx^2 (s^4 - (Vx Vb - Vy Va)^2)])/s^2; (* Receives VxAC0,VyAC0,VxWxComb,VyWxComb,airspeedAC0kt, all in nm/hr *)

vxNew2[Vx_, Vy_, Va_, Vb_, s_] := (Vy (Vx Vb - Va Vy) + Sqrt[Vx^2 (s^4 - (Vx Vb - Vy Va)^2)])/s^2;

saveData[] := Module[{}, (* Always coordinate this Module with the Wx generator and Viewer programs to ensure identical variables are being used *)
    (*header={"E","S","Sc","PresnOrder","Map","MapJPGSizePix","MapSizeNM","ScreenSizeCM","ScreenResXY","a1","b1","a2","b2","Gsize","n","yxWx1",
        "yxWx2","yxWx1end", "yxWx2end","σ1x","σ1y","σ2x","σ2y","θ1","θ2","dx1","dy1","dx2","dy2","rad","noise","zOrange","startPt","destnPt",
        "airSpdACkt","frmsViewed", "EThr", "pathLennm", "minDnm","waypts"};*)
    contents[[2, 37]] = Round[t, .0001];                    (* Elapsed time, in hours *)
    contents[[2, 39]] = Round[absMinD / nmToSCU, .001];     (* Minimum distance to ellipse (minD), in nm *)
    (*vars=contents[[3]]; *) (* {"nFrmsViewed","airSpdACkt","frmsViewed","EThrs","pathLennm","minDnm"."waypts"} *)
    data = {contents[[1]], contents[[2]] (*,vars *)};
    ptr = OpenWrite[inFilename];
    Write[ptr, data];
    Close[inFilename];
    Export[StringJoin[pixFilenameRoot, ".spds.jpg"], gSpds, "JPEG"];    (* Export .jpg of groundspeeds x time *)
    Export[StringJoin[pixFilenameRoot, ".score.jpg"], gScore, "JPEG"]; (* Export .jpg of flightpath, wx, and minD *)
  ]; (* End Module saveData[] *)

fileRoot = StringJoin[exptID, ".", subjID, ".", scenID];
inFilename = StringJoin["Data\\", fileRoot, ".dat"];
pixFilenameRoot = StringJoin["Data\\", fileRoot]; (* Used in saveData[] function *)
ptr = OpenRead[inFilename];
contents = Read[ptr];
Close[inFilename];
```

```
pixName = StringJoin["Wx movies\\", exptID, ".xx.", scenID, ".frame1.jpg"];
frmSizePix =  contents[[2, 6]]; (* Import[pixName,"ImageSize"]; Extract terrain map image (frame) size *)
mapSizeNM = contents[[2, 7]];  (* onscreen width of terrain map displayed in Viewer, in nautical miles (nm) *)
nmToSCU =  frmSizePix[[1]] / mapSizeNM; (* A conversion factor = screen coord units (SCU) per nautical mile on x-axis  *)
a1 = contents[[2, 10]];  b1 = contents[[2, 11]];  a2 = contents[[2, 12]];  b2 = contents[[2, 13]]; (* Raw wx-cell ellipse data,  before adjustment for screen size *)
Gsize = contents[[2, 14]]; (* Grid size of the underlying terrain map *)
numFrames = contents[[2, 15]]; (* Number of frames in a 1-hr movie *)
(* NOTE:  The starting position of the wx cells should equal their FINAL position from the Viewer program, i.e., yxWx1end and yxWx2end *)
yxWx1 = contents[[2, 18]];  yxWx2 = contents[[2, 19]]; (* Start position of wx cells 1 and 2 in grid coords, BUT 1st value is y, 2nd value is x *)
θWx1 = contents[[2, 24]]; θWx2 = contents[[2, 25]]; (* Rotation angle of weather cells 1 and 2*)
dx1 = contents[[2, 26]];  dy1 = contents[[2, 27]];  dx2 = contents[[2, 28]];  dy2 = contents[[2, 29]]; (* Per-frame deltas of wx, in grid units *)
startPt = contents[[2, 33]] ; destnPt = contents[[2, 34]]; (* Aircraft starting pt and finish pt, in screen coordinates *)
airspeedAC0kt = contents[[2, 35]]; (* No-wind aircraft airspeed, in nm/hr. Viewer controls this *)
(* pathLength=contents[[2,37]]; Total precomputed flightpath length, in screen units *)
wayPts = contents[[2, 40]];  numWayPts = Length[wayPts];

temp = N[numFrames / Gsize]; (* Recall that each "movie" is one hour long, has numFrames, and dx, dy are in grid coords, which were set in the Wx Generator *)
VxWx1 = temp dx1 frmSizePix[[1]];  VxWx2 = temp dx2 frmSizePix[[1]];  (* Wx speed, derived from grid delta values, here converted to SCU *)
VyWx1 =  temp dy1 frmSizePix[[2]];  VyWx2 = temp dy2 frmSizePix[[2]];
      (* Below, keep in mind that the positions of wx cells start out in grid coordinates, NOT screen coordinate units *)
xWx10 = yxWx1[[2]];  yWx10 = yxWx1[[1]]; (* Location of the center of weather cell 1 at time 0. Remember--x and y are reversed *)
xWx20 = yxWx2[[2]];  yWx20 = yxWx2[[1]]; (* Ditto for wx cell 2. Remember that initial location was in grid coords *)
convX = frmSizePix[[1]] / Gsize;  convY = frmSizePix[[2]] / Gsize; (* Conversion factors, xy grid units → SCU *)
xWx10 *= convX;  yWx10 *= convY;  xWx20 *= convX;  yWx20 *= convY; (* Convert initial grid locations of wx cells to SCU *)
a1 *= convX; b1 *= convY;  a2 *= convX;  b2 *= convY; (* Convert major and minor elliptical axes to SCU *)

myMethod = {Automatic, "DifferentialEvolution", "NelderMead", "RandomSearch", "SimulatedAnnealing"};
m = 3; (* Used to select myMethod above*)
minDs = windX = windY = {}; absMinD = 9999; tMin = 9999; dE = {0, 0, 0, 0};
recordSamples = True; (* Whether or not to record and show individual samples for dmin *)
segXYETs = grdSpds = {}; (* Will contain aircraft {x,y} position (in screen coordinates) at "universal" total elapsed scenario time *)
gSamplePoints = {}; totalDistTravelled = scenarioET = 0;  minSegLength  = 9999;
whatColor[i_] := Switch[i, 1, Blue, 2, Red, 3, Black, 4, Blue, 5, Red, 6, Black, 7, Blue, 8, Red, _, Black];
```

```
time = Timing[ (* Below, we will have to mark time separately for the aircraft ("segment time") and the weather ("universal time") *)
  (* AND, the component ground speed velocities of aircraft will have to be calculated segment-by-segment *)
  t = 0; (* scenario-wide elapsed time, used directly for weather movement *)
  δt0 = δtOld = N[1 / 30]; (* Initial time granularity, in hours *)
  pLTot = 0; (* Total flightpath length *)
  For[i = 1, i ≤ numWayPts - 1 , i++, (* Go through each flight segment, one by one *)
        xAC0 = wayPts[[i, 1]];    yAC0 = wayPts[[i, 2]]; xACT = wayPts[[i + 1, 1]];   yACT = wayPts[[i + 1, 2]];
        dx = xACT - xAC0;  dy = yACT - yAC0;
        αAC = Mod[ArcTan[dx, dy], 2 Pi]; (* Angle of the current flightpath segment, in radians.  ArcTan can throw negative values, so we ensure that αAC ≥ 0 *)
        VxAC0 = airspeedAC0kt Cos[αAC]; VyAC0 = airspeedAC0kt Sin[αAC]; (* No-wind x- and y-components of airspeed, in nm/hr *)
        segLength = √(dx^2 + dy^2) ;   (* Length of that flight segment, in screen coordinate units *)
        (* If[segLength<minSegLength,minSegLength=segLength];    Find the scenario-wide shortest segment *)
        segLengthSoFar = tAC = 0; (* tAC is the elapsed time for the aircraft on the ith segment *)
        While[segLengthSoFar < segLength,  (* Find minimum dist to ellipse halves (in SCU). NMinimize returns a structure of form {d, xd}, e.g. {462.613, {x→196.965}} *)
              If[recordSamples, AppendTo[gSamplePoints, Graphics[{whatColor[i], Point[{xAC0, yAC0}]}]]]; (* Record one pt on the A/C flightpath for each value of t *)
              dE[[1]] = NMinimize[{dTopT[a1, b1, xWx1t[t], yWx1t[t], x, tAC, t, θWx1], xWx1t[t] - a1 ≤ x ≤ xWx1t[t] + a1}, x, Method → myMethod[[m]] ]; (*Ellipse 1, top. *)
              dE[[2]] = NMinimize[{dBtmT[a1, b1, xWx1t[t], yWx1t[t], x, tAC, t, θWx1], xWx1t[t] - a1 ≤ x ≤ xWx1t[t] + a1}, x, Method → myMethod[[m]] ]; (*Ellipse 1, btm. *)
              dE[[3]] = NMinimize[{dTopT[a2, b2, xWx2t[t], yWx2t[t], x, tAC, t, θWx2], xWx2t[t] - a2 ≤ x ≤ xWx2t[t] + a2}, x, Method → myMethod[[m]] ]; (*Ellipse 2, top. *)
              dE[[4]] = NMinimize[{dBtmT[a2, b2, xWx2t[t], yWx2t[t], x, tAC, t, θWx2], xWx2t[t] - a2 ≤ x ≤ xWx2t[t] + a2}, x, Method → myMethod[[m]] ]; (*Ellipse 2, btm. *)
              minD = 9999; For[j = 1, j ≤ 4, j++, If[dE[[j, 1]] < minD, minD = dE[[j, 1]]; minIndex = j]]; (* Find the min dist to closest ellipse for this value of time, in SCU *)
                                                                                  (*minIndex can either be 1,2,3, or 4 *)
              If[minD < 10, AppendTo[minDs, {t, minD}]];
              If[minD / nmToSCU > 22, δt = δt0,  (* Check point-of-closest approach, in nm, and set time granularity δt *)
                If[minD / nmToSCU > 7.33, δt = δt0 / 2,
                  If[minD / nmToSCU > 2.44, δt = δt0 / 4,
                    If[minD / nmToSCU > 0.815, δt = δt0 / 8,
                      If[minD / nmToSCU > 0.272, δt = δt0 / 16,
                        If[minD / nmToSCU > 0.091, δt = δt0 / 32,
                          If[minD / nmToSCU > 0.030, δt = δt0 / 64,
                            δt = δt0 / 128
                  ]]]]]]];  If[δt != δtOld, Print["minD = ", minD, "   δt = ", δt];  δtOld = δt;];
```

```
If[minIndex == 1 || minIndex == 2,  (* Since we know the closest half-ellipse, we can also find the next-closest half-ellipse *)
    closerCellVxWx = VxWx1;  closerCellVyWx = VyWx1;   fartherCellVxWx = VxWx2;   fartherCellVyWx = VyWx2;  (* In SCU *)
    If[dE[[3, 1]] < dE[[4, 1]], minIndex2 = 3, minIndex2 = 4],  (* otherwise, minIndex must be 3 or 4, so... *)
    closerCellVxWx = VxWx2;  closerCellVyWx = VyWx2;  fartherCellVxWx = VxWx1;   fartherCellVyWx = VyWx1;
    If[dE[[1, 1]] < dE[[2, 1]], minIndex2 = 1, minIndex2 = 2]
    ];

If[minD < absMinD,
    absMinD = minD;  (* Capture the scenario-wide shortest distance to an ellipse *)
    tMin = t;            (* Capture the scenario-wide elapsed time of same *)
    xACMinD = xAC0;   yACMinD = yAC0;  (* Position of the A/C at that time *)
    xMin = dE[[minIndex, 2, 1, 2]];  (* x-value of the A/C, in SCU *)
    Switch[minIndex,     (* Calculate the x,y-point on the ellipse corresponding to minD *)
      1, temp = xyTop[a1, b1, xWx1t[t], yWx1t[t], xMin, tMin, θWx1]; xEMinD = temp[[1]]; yEMinD = temp[[2]];
                slope = topSlope[a1, b1, xWx1t[t], yWx1t[t], xEMinD, yEMinD, tMin, θWx1],
      2, temp = xyBtm[a1, b1, xWx1t[t], yWx1t[t], xMin, tMin, θWx1]; xEMinD = temp[[1]]; yEMinD = temp[[2]];
                slope = btmSlope[a1, b1, xWx1t[t], yWx1t[t], xEMinD, yEMinD, tMin, θWx1],
      3, temp = xyTop[a2, b2, xWx2t[t], yWx2t[t], xMin, tMin, θWx2]; xEMinD = temp[[1]]; yEMinD = temp[[2]];
                slope = topSlope[a2, b2, xWx2t[t], yWx2t[t], xEMinD, yEMinD, tMin, θWx2],
      4, temp = xyBtm[a2, b2, xWx2t[t], yWx2t[t], xMin, tMin, θWx2]; xEMinD = temp[[1]]; yEMinD = temp[[2]];
                slope = btmSlope[a2, b2, xWx2t[t], yWx2t[t], xEMinD, yEMinD, tMin, θWx2]
      ] (* End Switch *)
    ];   (* End If[minD<absMinD, *)

VxWxComb = (dE[[minIndex2, 1]] closerCellVxWx + dE[[minIndex, 1]] fartherCellVxWx) / nmToSCU (dE[[minIndex, 1]] + dE[[minIndex2, 1]]);(* Est. combined wind speed effect (converted from SCU to nm/hr) *)
VyWxComb = (dE[[minIndex2, 1]] closerCellVyWx + dE[[minIndex, 1]] fartherCellVyWx) / nmToSCU (dE[[minIndex, 1]] + dE[[minIndex2, 1]]);
AppendTo[windX, {t, VxWxComb}];  AppendTo[windY, {t, VyWxComb}]; (* nm/hr *)
(* If[VxWxComb==0,θWxComb=0,θWxComb=Mod[ArcTan[VxWxComb,VyWxComb],2 Pi]]; ArcTan can throw negative values, so ensure that θWxComb ≥ 0 *)
If[Pi / 2 < αAC ≤ 3 Pi / 2, VxNew = vxNew1[VxAC0, VyAC0, VxWxComb, VyWxComb, airspeedAC0kt],  (* All units in nm/hr *)
                            VxNew = vxNew2[VxAC0, VyAC0, VxWxComb, VyWxComb, airspeedAC0kt]
    ]; (*Be careful. This is quadrant-dependent *)
```

```
VyNew = Re[√(airspeedAC0kt^2 − VxNew^2)];(* in nm/hr.  Protect against complex−number anomalies along axes *)
        If[Pi ≤ αAC < 2 Pi || (αAC == 0 && VyWxComb > 0), VyNew = −VyNew]; (*This is also quadrant−dependent *)

        groundspeedACt = √((VxNew + VxWxComb)^2 + (VyNew + VyWxComb)^2) ; (* Est'd grndspd (nm/hr) at time t, accounting for normalized effect of the 2 wx cells *)
        AppendTo[grdSpds, {t, groundspeedACt}];
        VxACt = groundspeedACt Cos[αAC];  VyACt = groundspeedACt Sin[αAC];(* True groundspeed at  t, along orig. flightpath, accounting for est'd winds*)
        tempDX = VxACt δt nmToSCU;  tempDY = VyACt δt nmToSCU;(* Increment the position of the aircraft, converting from nm to SCU *)
        xAC0 += tempDX;  yAC0 += tempDY; (* Increment the starting x, y for the next time step *)
        segLengthSoFar += √(tempDX^2 + tempDY^2) ;
        t += δt; (* Increment the scenario−wide elapsed time, which also drives the weather *)
        tAC += δt; (* Increment the segment−specific elapsed time, which drives the aircraft *)
    ]; (* End While *)  (* Below, we correct for the small "overshoot" time error at the end of each segment *)
    t −= (segLengthSoFar − segLength) / (groundspeedACt nmToSCU); (*Simplification of t−= δt ( segLengthSoFar−segLength )/ ( δt  groundspeedACt )  *)
    pLTot += segLength;
    Print["Seqment ", i, "\tabsMinD = ", absMinD, " (SCU)  ", absMinD / nmToSCU, " (nm)"];
    (* tSeg=segLength/SpdAC;                                  Time to traverse this segment, in hr *)
    (* totalET+=tSeg;                                  Keep track of total elapsed time to complete scenario
      AppendTo[segXYETs,totalET ]; *)
  ]; (* End For i=1... *)
  saveData[];
  ];    (* End timing *)
Print[time, "  Number of sample points = ", Length[gSamplePoints], "   tMin = ", tMin];
slopeNormal = −1 / slope;  Print["slope = ", slope, "  slopeNormal = ", slopeNormal];
 Print["Took ", time[[1]], " seconds for NMinimize  abs min D = ", absMinD, " (SCU)  ", absMinD / nmToSCU, " (nm)",   "\n{x,y} of ellipse minpt = ", {xEMinD, yEMinD},
   "  {x,y} of A/C minpt = ", {xACMinD, yACMinD}];
gAC = Graphics[{Green, Arrow[wayPts]}];
gWx1 = Graphics[{Green, Arrow[{{xWx1t[0], yWx1t[0]}, {xWx1t[t], yWx1t[t]}}]}];
gWx2 = Graphics[{Green, Arrow[{{xWx2t[0], yWx2t[0]}, {xWx2t[t], yWx2t[t]}}]}];
```

```
topHalfgE10 = Table[xyTop[a1, b1, xWx10, yWx10, x, 0, θWx1], {x, xWx10 – a1 , xWx10 + a1 , 1 }];
    btmHalfgE10 = Table[xyBtm[a1, b1, xWx10, yWx10, x, 0, θWx1], {x, xWx10 – a1 , xWx10 + a1 , 1 }];
    topHalfgE1T = Table[xyTop[a1, b1, xWx1t[tMin], yWx1t[tMin], x, tMin, θWx1], {x, xWx1t[tMin] – a1 , xWx1t[tMin] + a1 , 1 }];
    btmHalfgE1T = Table[xyBtm[a1, b1, xWx1t[tMin], yWx1t[tMin], x, tMin, θWx1], {x, xWx1t[tMin] – a1 , xWx1t[tMin] + a1 , 1 }];
    topHalfgE1E = Table[xyTop[a1, b1, xWx1t[t], yWx1t[t], x, t, θWx1], {x, xWx1t[t] – a1 , xWx1t[t] + a1 , 1 }];
    btmHalfgE1E = Table[xyBtm[a1, b1, xWx1t[t], yWx1t[t], x, t, θWx1], {x, xWx1t[t] – a1 , xWx1t[t] + a1 , 1 }];
    topHalfgE20 = Table[xyTop[a2, b2, xWx20, yWx20, x, 0, θWx2], {x, xWx20 – a2 , xWx20 + a2, 1 }];
    btmHalfgE20 = Table[xyBtm[a2, b2, xWx20, yWx20, x, 0, θWx2], {x, xWx20 – a2 , xWx20 + a2, 1 }];
    topHalfgE2T = Table[xyTop[a2, b2, xWx2t[tMin], yWx2t[tMin], x, tMin, θWx2], {x, xWx2t[tMin] – a2 , xWx2t[tMin] + a2 , 1 }];
    btmHalfgE2T = Table[xyBtm[a2, b2, xWx2t[tMin], yWx2t[tMin], x, tMin, θWx2], {x, xWx2t[tMin] – a2 , xWx2t[tMin] + a2 , 1 }];
    topHalfgE2E = Table[xyTop[a2, b2, xWx2t[t], yWx2t[t], x, t, θWx2], {x, xWx2t[t] – a2 , xWx2t[t] + a2 , 1 }];
    btmHalfgE2E = Table[xyBtm[a2, b2, xWx2t[t], yWx2t[t], x, t, θWx2], {x, xWx2t[t] – a2 , xWx2t[t] + a2 , 1 }];
    gE10 = ListPlot[ {topHalfgE10, btmHalfgE10}, PlotStyle → {Dashed, Red}, Joined → True]; (* Draw the ellipse at t=0 *)
    gE1T = ListPlot[{topHalfgE1T, btmHalfgE1T}, PlotStyle → Green, Joined → True]; (* Draw the ellipse at t=tMin *)
    gE1E = ListPlot[{topHalfgE1E, btmHalfgE1E}, PlotStyle → {Dashed, Blue}, Joined → True]; (*Draw the ellipse at scenario's end *)
    gE20 = ListPlot[ {topHalfgE20, btmHalfgE20}, PlotStyle → {Dashed, Red}, Joined → True];
    gE2T = ListPlot[{topHalfgE2T, btmHalfgE2T}, PlotStyle → Green, Joined → True];
    gE2E = ListPlot[{topHalfgE2E, btmHalfgE2E}, PlotStyle → {Dashed, Blue}, Joined → True];
    tL = 16 Cos[ArcTan[slope]]; tN = absMinD Cos[ArcTan[slopeNormal]];
    gTan = Graphics[Line[{{xEMinD – tL, yEMinD – (tL slope) }, {xEMinD + tL , yEMinD + (tL slope) }}]];
    gNormal = Graphics[Line[{{xEMinD – tN, yEMinD – (tN slopeNormal)}, {xEMinD + tN , yEMinD + (tN slopeNormal)}} ]];
    gAC0 = Graphics[{Green, Thick, Line[{{0, airspeedAC0kt}, {t, airspeedAC0kt}}]}];
    gtMin = Graphics[{Red, Line[{{tMin, –5}, {tMin, airspeedAC0kt + 5}}]}];
    gs1 = ListPlot[grdSpds, PlotRange → All, PlotStyle → Blue, Joined → True, GridLines → Automatic];
    gs2 = ListPlot[grdSpds, PlotRange → All, PlotStyle → Black, Joined → False];
    gWindX = ListPlot[windX, PlotRange → All, PlotStyle → Red, Joined → True];
    gWindY = ListPlot[windY, PlotRange → All, PlotStyle → Green, Joined → True];
    gEMinXY = Graphics[{PointSize[0.01], Point[{xEMinD, yEMinD}]}];
    gACMinXY = Graphics[{PointSize[0.01], Point[{xACMinD, yACMinD}]}];
    Print[gSpds = Show[(*gWindX,gWindY,*)gs1, gAC0, gs2, (*gtMin,*) ImageSize → Medium, PlotRange → All, GridLines → Automatic]];
    Print[gScore = Show[gAC, gWx1, gWx2, gSamplePoints, gE10, gE1T, gE1E, gE20, gE2T, gE2E, gEMinXY, gACMinXY, gTan, gNormal, AspectRatio → Automatic,
          ImageSize → Large, PlotRange → {{0, frmSizePix[[1]] }, {0, frmSizePix[[2]] }}, Axes → True, AxesOrigin → {0, 0}, GridLines → Automatic]];
    Print["t = ", t, " (hr)"];
    ]; (* End Block *)
```