Providing Metrics-Based Results to Student Pilots for Critical Phases of General Aviation Flights

by

Kelton Olson Karboviak
Bachelor of Science, University of North Dakota, 2016

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

May
2018

This thesis, submitted by Kelton Olson Karboviak in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

Dr. Travis Desell

Dr. Hassan Reza

Prof. Brandon Wild

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.

Wayne Swisher
Dean of the School of Graduate Studies

Date

PERMISSION

Title	Providing Metrics-Based Results to Student Pilots for Critical Phases of General Aviation Flights
Department	Department of Computer Science
Degree	Master of Science

Kelton Olson Karboviak
May 2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

For/Dedicated to/To my…

ABSTRACT

TODO LIST

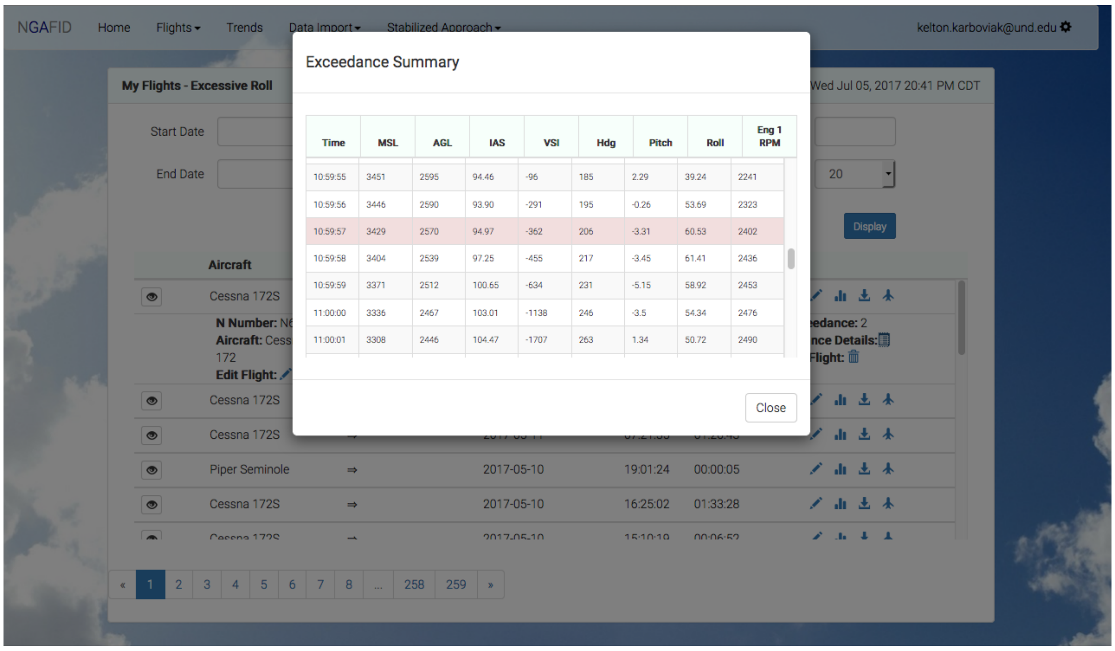
CHAPTER 1

INTRODUCTION

General Aviation (GA) is one of two branches of Civil Aviation, which pertains to the operation of all non-scheduled and non-military aircraft [2–5]. GA includes fixed-wing airplanes, helicopters (rotorcraft), balloons, dirigibles, gliders, etc.; and comprises 63% of all Civil Aviation activity within the U.S. [2, 4, 6]. Performing GA flight analysis is essential for making the GA community safer, as currently GA has the highest accident rates in Civil Aviation [5, 7]. As of 2014, the total accident and fatality rates for GA fixed-wing aircraft were 5.78 and 1.39 per 100,000 flight hours, respectively; and 75.3% of GA accidents were caused by pilot-related actions [5].

The National General Aviation Flight Information Database (NGAFID) has been developed at the University of North Dakota as a joint university-industry-FAA initiative that is responsible for the curation, dissemination, and analysis of flight data for the General Aviation (GA) sector of Civil Aviation [8, 9]. The objective of the NGAFID is to proactively identify accident precursors and mitigate risks associated with unsafe flight practices and aircraft maintenance issues within the GA community. This is achieved via non-punitive information sharing so as to educate operators on risks associated with their flights to encourage safer practices [8]. The analytical tools provided by the NGAFID are free and available to GA pilots who participate by uploading their flight data through the NGAFID web application[1] or the GAARD mobile application [10]. Subsequently, their flight data is preprocessed and analyzed using various queries. Many queries are based on threshold criteria called *exceedances*, which are predefined using known limitations of the

[1]http://www.ngafid.com



Time	MSL	AGL	IAS	VSI	Hdg	Pitch	Roll	Eng 1 RPM
10:59:55	3451	2595	94.46	-96	185	2.29	39.24	2241
10:59:56	3446	2590	93.90	-291	195	-0.26	53.69	2323
10:59:57	3429	2570	94.97	-362	206	-3.31	60.53	2402
10:59:58	3404	2539	97.25	-455	217	-3.45	61.41	2436
10:59:59	3371	2512	100.65	-634	231	-5.15	58.92	2453
11:00:00	3336	2467	103.01	-1138	246	-3.5	54.34	2476
11:00:01	3308	2446	104.47	-1707	263	1.34	50.72	2490

Figure 1: Screen-shot of a flight which has an excessive roll exceedance. The exceedance event is highlighted in red.

make/model aircraft or the phase of flight. However, recent work has focused on developing more advanced analytics through machine learning and other holistic techniques [11–16]. Upon logging into the web portal, the user is provided with summaries of any unsafe events (see Figure 1) and is able to reanimate their flight(s) using X-Plane[2] or Cesium[3] (see Figure 2). The intent is to educate participating pilots on any unsafe practices in their flight and maintenance issues with the aircraft which may contribute to an accident/incident. The overall goal of this initiative is to reduce the accident and fatality rates within the GA community.

Scope & Objectives

The goal of this research is to develop an automated grading system for analyzing quality of Approach and Landing phases with a low error rate, reasonable run-time, and that can handle the wide variation of GA flight. This

[2]http://www.x-plane.com
[3]http://www.cesiumjs.org

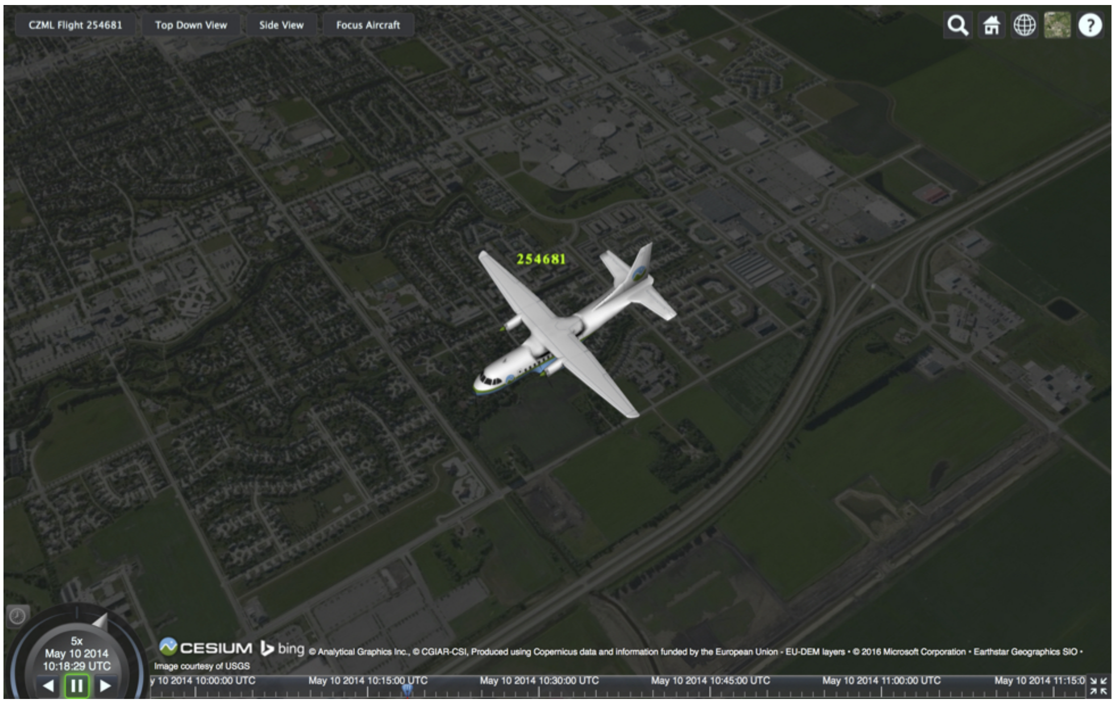


Figure 2: Example of in-browser flight reanimation using Cesium.

application will be useful in several different areas:

- provide student pilots with a grade/metric that they can use to gauge a flight's quality, which helps target different student learning techniques,
- help improve the flight training process for Certified Flight Instructors (CFI) by making post-flight evaluation more efficient,
- help reduce costs of training for the student and institution due to a lesser need for additional training flights, and
- help further reduce GA accident and fatality rates.

Motivation

Despite many safety efforts that have been recently introduced to the GA community, accident rates in the United States remain high. One way of characterizing flight safety is by identifying *exceedances*, or events that bring the aircraft into an unsafe state dependent on the phase of flight. By detecting these exceedances, we can identify areas of improvement at the pilot- or organizational-level and additionally educate the pilots on their unsafe practices. In particular, this research will be mainly focused on this aspect of improving teaching feedback for Certified Flight Instructors (CFI) and student pilots within flight training institutions (although it may be useful to individual private pilots as a side-effect). Furthermore, the scope of analysis is for the Approach and Landing phases of flight since these are some of the most critical phases in GA as they ranked #6 and #1, respectively, for number of pilot-related accident types in 2014 [5]. However, due to the high variability in GA operations and flight performance, phases of flight and exceedances are often difficult to detect [17–19].

Currently at the time of this writing, the feedback that a student pilot receives for their flying performance consists of a verbal debriefing given by the student's CFI. This process can be prone to errors as both the CFI and student must recall the flight from memory or utilize any notes they were able to take mid-flight. Additionally, a cross-country flight may last anywhere from 1 to 1.5 hours, which can be a lot to recall after they have landed and returned the aircraft. This may work well for some students, but not for all as many people do not have a great short-term memory. For these reasons, an automated system that can provide metrics-based results of their flying performance and allow them to replay their flight will benefit student pilots by providing another means of obtaining feedback, which will cater to students who learn more efficiently with visual materials.

Outline

This thesis is organized with related works in the areas of aircraft operations, post-flight evaluation tools, NGAFID related work, and data mining techniques in Chapter 2. The approach to phase of flight identification, exceedance detection, grading, and web interface are discussed in Chapter 3. The implementation; including the programming languages, libraries, and parallelization techniques used; are discussed in Chapter 4. Results of the flight analysis are given in Chapter 5. Finally, there is a conclusion of the research and a discussion of future work in Chapter 6.

CHAPTER 2

RELATED WORK

Aircraft Operations

In Dr. Ed Wischmeyer's paper, *The Myth of the Unstable Approach* [20], he discusses how the term "unstable approach" is now becoming too vague to be used in accident and incident reports. He argues there are too many factors that play into an approach; therefore, labeling it solely as an "unstable approach" is not sufficient. This aligns with one of the goals of the Go-Around Detection Tool in that it was developed to detect unstable approaches and be able to state what the specific parameter was that caused the approach to be unstable. In doing this, it allows for finer-grained statistics to be generated, which can reveal further patterns to be detected within an organization if it becomes a wide-spread problem.

Nazeri *et* al. [21] researched accident and incident data from several different commercial flight data sources in order to discover the factors that cause those events. They created eight high-level categories, each with sub-factors, for classification. They used an algorithm to analyze the data for correlations between different attribute-value pairs across the accident and incident data sets. A factor support ratio was calculated for each attribute-value pair and ranked in decreasing order to find the most significant factors. The following high-level factors were the four top ranked in order: company, air traffic control, pilot, and aircraft. They also did a time-series analysis of the data for the ten-year period in which the data was collected (1995-2004). This time-series data showed the pilot and aircraft factors are generally decreasing over time, while the air traffic control factors are generally increasing. By uncovering these patterns and analyzing them over time, they were able to find the factors that are leading causes for accidents/incidents and can address these factors for improvement.

Post-Flight Evaluation Tools

There are several software projects that have created post-flight evaluation tools, which a pilot can use to analyze their performance during various phases of flight. Each project has a similar methodology, but varying presentation techniques. Knighton and Claramunt [22] created a system that included hardware for collecting real-time flight data and an interactive graphical user interface (GUI). The GUI is capable of 2D flight re-animation and simple plots of the aircraft's vertical profile throughout the flight. The flight re-animation also has a panel with indicators showing the real-time sensor data at each time step. They found through experiments using volunteers that their interface was intuitive and encouraged exploration of different aspects of flight performance. Despite the usefulness of the flight parameter graphs, the downside to their research is that any analysis has to be performed manually by the user as there are no automated analysis results provided.

Masiulionis and Stankunas [23] provide a review of several software packages for flight analysis including *IGC Flight Replay*, *OziExplorer*, *GPS TrackMaker*, and *ArcGIS*. They found that none of the packages provided all the aspects they sought: interactively enabling/disabling map layers, graphing multiple flight parameters on a single plot, and high-resolution maps. Thus, they experimented with flight analysis and visualization using *Google Earth*. *Google Earth* met all of their standards, but the only disadvantage is that the altitude and speed graphs can become compressed for flights with an extended duration. It does not include any features to resize or drill-down into the graphs to make them more viewable.

Kelton: comment on how mine is different

Goblet *et* al. [17, 18] researched into automatically classifying phases of GA flights. The focus is on the Climb, Cruise, and Descent phases as they are the most difficult phases to identify in GA due to the variation of mission profiles and purposes of flight when compared to Commercial Aviation. Several methods were explored for identifying these phases: altitude-based and smoothing-and-differentiation-based (which includes down-sampling, moving average, and local regression). It was found the best method to use for a particular flight is determined by specific characteristics found within that flight. An algorithm is then given to automatically select the best method for each flight. Although the Climb, Cruise, and Descent phases are the most difficult to identify, it is known that the Approach, Landing, and Go-Around phases are the most critical and dangerous in GA (as discussed in Chapter 1), thus is the reason this work focuses on analysis of those phases.

Fala and Marais [19] developed a method of detecting unsafe aircraft parameters, termed "safety events", in GA flights. Similar to phase identification, detecting safety events in GA is difficult due to the variability in operations. In their research, they only focus on the Approach phase and provide the numerical parameter limits for a Cirrus SR20 aircraft during the Approach phase. For each parameter, they provide a Level 1 and Level 2 limit stating that Level 2 is more dangerous than Level 1. They analyzed the Approach phases from a sample of 23 flights using their defined thresholds and performed a one-way ANOVA analysis to evaluate whether the average number of instances for each type of safety event was similar. They found their initial threshold definitions were not similar enough across the parameters. They revised the definitions, re-analyzed the flights, and performed another ANOVA analysis to find that they were then significantly similar. When detecting safety events, Fala and Marais did not distinguish between sporadic exceedances of the thresholds and a span of consecutive exceedances. This differs from the work in this project, where an event can span multiple seconds instead of only single time steps.

explain why I chose

NGAFID Related Work

Other work on the NGAFID has focused in two different areas. In the first, Desell *et* al. have examined methods based on the prediction of flight data parameters using recurrent neural networks (RNNs). They have shown that training Jordan and Elman RNNs using evolutionary algorithms such as Particle Swarm Optimization and Differential Evolution can provide strong predictive results for flight data parameters such as airspeed, altitude, pitch, and roll [16]; and that these results can be further refined utilizing a novel neuroevolution technique based on ant colony optimization to evolve the structure of the RNNs [13]. Further work by ElSaid *et* al. has utilized Long Short-Term Memory (LSTM) RNNs to predict aircraft engine vibration events [14, 15].

The other area has been in utilizing unsupervised machine learning methods to detect anomalous flights. Clachar *et* al. have used self organizing maps (SOMs), a type of neural network, to both cluster time series flight data and identify anomalous flights [11, 12] based on approach phases, with SOMs providing significant benefits in terms of parallelism and performance over other clustering methods such as DBSCAN.

Data Mining Techniques

Harris *et* al. [24] of MITRE Corporation mined accident and incident reports provided by the International Civil Aviation Organization (ICAO) in order to determine the specific attributes that were the cause in each kind of report and also needed to be considered "interesting" (i.e., anything that is an exception to commonly accepted knowledge among aviation experts) by the aviation expert who collaborated with them. They discovered that using traditional data mining methods was not sufficient enough to find "interesting" results. This is because experts have studied the field of aviation so extensively that all the obvious rules and correlations have already been discovered. Next, they developed their own system called Smithers, which uses a technique called attribute focusing, that finally uncovered an interesting correlation which shows that having an advanced heads up display (HUD) can help reduce the amount of damage as a result of a runway incursion[1]. This shows that even though aviation safety has been studied extensively, there are still new correlations that can be made when applying several different data mining methods.

Matthews *et* al. [26] performed similar research in which their goal was to find anomalous data in flights. They differ in the fact that they used algorithms that could analyze at both a fleet-level and flight-level. Doing this allowed them to find anomalies for an entire organization or just a single flight, which makes it very useful in order to find patterns of problems. This idea is similar to the NGAFID project in which flight data can be analyzed on multiple levels while giving statistics for each.

[1]Defined by the Federal Aviation Administration (FAA) as, “any occurrence at an aerodome involving the incorrect presence of an aircraft, vehicle, or person on the protected area of a surface designated for the landing and take off of aircraft” [25].

CHAPTER 3

METHODOLOGY

The Go-Around Detection Tool provides several features: *(i)* phase of flight identification, *(ii)* quality analysis of each phase, *(iii)* grade assignment, and *(iv)* a web interface to display results. Since there are three separate phases of concern, there will be a separate subsection for each in both identification and quality analysis. These features are discussed in more detail in the rest of this Chapter.

Phase of Flight Identification

Kelton: Not trivial because multiple circuits in a single flight. Lack technology found in commercial aircraft. Flying patterns are highly variable in GA due to flight training.

Approach

The Approach phase is defined as the time between the aircraft entering the airport's traffic pattern (shown in Figure 3), or 1,000 feet above the runway elevation, to the beginning of the landing flare under Visual Flight Rules (VFR). For Instrument Flight Rules (IFR), it is the time from the Initial Approach Fix (IAF) to the beginning of the landing flare [27].

Along with detecting the Approach phase (Algorithm 1), this section also details the algorithms for detecting *(i)* the airport and runway that the aircraft is approaching and *(ii)* the Final Turn so it can later be analyzed for an undershoot or overshoot.

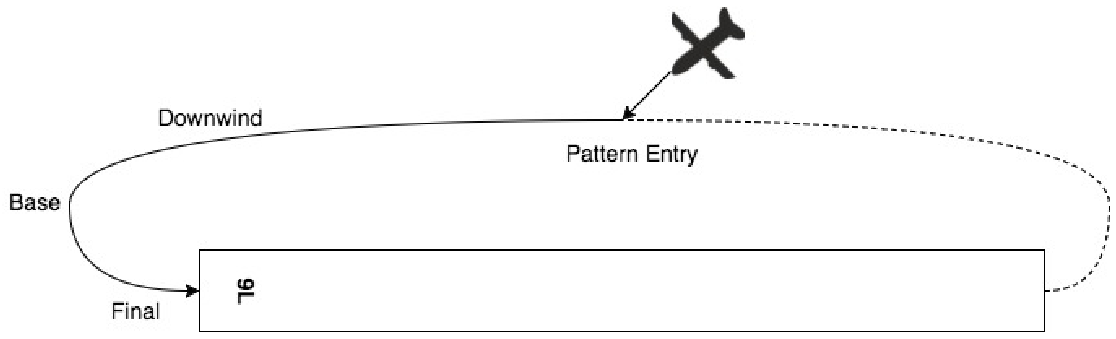


Figure 3: Example showing an airport's traffic pattern and the subphases of the Approach.

The algorithm for detecting an aircraft's approach needs to iterate through all of the time values since there can be multiple approaches within a single flight. Once the algorithm detects the aircraft is 1 mile away from an airport and is less than 500 feet above ground level (AGL) (Algorithm 1 Line 6), it is determined that the pilot is beginning an approach and a unique approach identifier is generated in order to store meta-data later in the process. Next, the algorithm continues to iterate through time values until either the aircraft goes under 200 ft AGL, or it goes back above 500 ft AGL, which will then be recorded as a go-around later in the process (Algorithm 1 Lines 8-12). If the aircraft goes under 200 ft AGL, then it is determined to be on the final approach. The aircraft is considered to be on the final approach while it is within 1 mile away from the airport and it is between 50 and 200 feet AGL inclusive (Algorithm 1 Lines 16-22).

Once the aircraft either goes above 200 feet AGL or goes below 50 feet AGL, then the final approach is marked as finished, and the critical meta-data associated with the approach is stored. At this point, the runway that is being approached can be detected using a combination of the aircraft's current geolocation and heading since the intended runway may not be closest to the aircraft depending on the degree of the Final Turn (Algorithm 1 Line 24).

Algorithm 1 Pseudo-code for function which detects when an aircraft is approaching a runway.

```
1: $airplanePoint \leftarrow data[i].geoPoint$
2: $airport \leftarrow detectAirport(airplanePoint)$
3: $airplaneAltitude \leftarrow data[i].altitude$
4: $heightAGL \leftarrow airplaneAltitude - airport.altitude$
5: $distance \leftarrow airplanePoint.distanceTo(airport.geoPoint)$
6: if $distance < 1\,\text{mi}$ and $heightAGL < 500\,\text{ft}$ then
7:     $apprID \leftarrow genNewApproachID()$
8:     while $200\,\text{ft} < heightAGL < 500\,\text{ft}$ and $i < data.length$ do
9:         $airplaneAltitude \leftarrow data[i].altitude$
10:        $heightAGL \leftarrow airplaneAltitude - airport.altitude$
11:        $i \leftarrow i + 1$
12:    end while
13:    $approachStartTime \leftarrow i$
14:    $airplaneHdg \leftarrow data[i].hdg$
15:    $airplanePoint \leftarrow data[i].geoPoint$
16:    while $distance < 1\,\text{mi}$ and $50\,\text{ft} \leq heightAGL \leq 200\,\text{ft}$ and $i < data.length$
           do
17:        $airplaneAltitude \leftarrow data[i].altitude$
18:        $airplanePoint \leftarrow data[i].geoPoint$
19:        $distance \leftarrow airplanePoint.distanceTo(airport.geoPoint)$
20:        $heightAGL \leftarrow airplaneAltitude - airport.altitude$
21:        $i \leftarrow i + 1$
22:    end while
23:    $approachEndTime \leftarrow i$
24:    $runway \leftarrow detectRunway(airplanePoint, airplaneHdg, airport)$
25:    $approaches[apprID] \leftarrow$ store approach meta-data
26:    return $(approachStartTime, approachEndTime)$
27: end if
```

Airport detection.

For identifying the airport that is being approached, a QuadTree [28] data structure was used. It was used due to the fact that a two-dimensional tree structure is needed in order to efficiently find the closest airport latitude and longitude point when given the aircraft's latitude and longitude. The QuadTree is constructed using a list of airport objects from a database then is optimized. The insertion algorithm yields $\mathcal{O}(n \log n)$ complexity when given random keys, and performs in $\mathcal{O}(\log n)$ time when searching an optimized tree.

Runway detection.

The algorithm used for finding the runway that is being approached is a simple sequential search with a constraint that the difference between the aircraft's heading and the runway's heading must be within an upper limit. The reasoning for this constraint is the fact that the runway closest to the aircraft may not necessarily be the one it is approaching depending on the arrangement of the runways and the degree of the Final Turn (see Figure 3). A value of 20° was used for the heading constraint since it is double the value used for detecting a heading exceedance (see Table 2). Thus if the runway returned by the algorithm is not the intended runway, it means the aircraft's heading is significantly off-center from the runway's heading and the pilot will need to perform severe corrections to get back on course.

In this case, using a sequential search is efficient enough since an airport has a very small number of runways, whereas there are thousands of airports within the United States which requires a more sophisticated algorithm. The runway detection algorithm is given in Algorithm 2.

Algorithm 2 Pseudo-code for *detectRunway* function which detects the runway an aircraft is approaching.

```
 1: function DetectRunway(airplanePoint, airplaneHdg, airport)
 2:     $theRunway \leftarrow NULL$
 3:     $closestDistance \leftarrow \infty$
 4:     for $runway$ in $airport.runways$ do
 5:         if $|\,headingDifference(runway.hdg,\ airplaneHdg)\,| \leq 20^\circ$ then
 6:             $distance \leftarrow airplanePoint.distanceTo(runway.geoPoint)$
 7:             if $distance \leq closestDistance$ then
 8:                 $theRunway \leftarrow runway$
 9:                 $closestDifference \leftarrow difference$
10:             end if
11:         end if
12:     end for
13:     return theRunway
14: end function
```

Final Turn detection.

In order to detect the Final Turn subphase of the Approach, we first get the previous three minutes of data before the Approach ends. The previous three minutes are used to reduce the search space and because it does not make logical sense for a Final Turn to occur greater than three minutes before the Approach ends. Next, the algorithm searches for the Final Turn start and end time. It finds these by searching for the points at which the aircraft's heading creates a 90° and 15° angle, respectively, to the runway's heading. A visualization of these reference points can be seen in Figure 4. The search is performed backwards through the slice of data in order to obtain the last occurrence of each angle difference. Once both points have been found, they are stored for later use in the analysis stage. If the aircraft did not have a heading difference greater than 90° in the final three minutes, the pilot performed a straight-in approach and, consequently, did not execute the Final Turn subphase. The Final Turn detection algorithm is given in Algorithm 3.

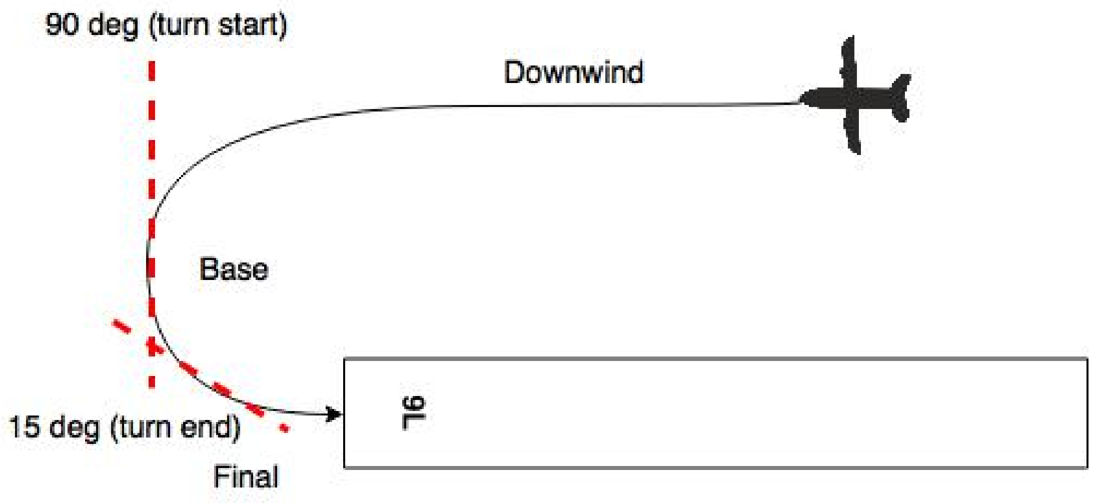


Figure 4: Example showing the Approach subphases and the slice of data used in the Final Turn analysis. The dashed lines represent when the Final Turn starts (90° heading difference) and ends (15° heading difference).

Algorithm 3 Pseudo-code for function which detects the Final Turn subphase of the Approach.

```
 1: function DetectFinalTurn(approachEndTime, runway)
 2:     $last3Mins \leftarrow$ get previous 3 mins of data before $approachEndTime$
 3:     $turnStartTime \leftarrow NULL$
 4:     $turnStartFound \leftarrow false$
 5:     $turnEndTime \leftarrow NULL$
 6:     $turnEndFound \leftarrow false$
 7:     $i \leftarrow last3Mins.length - 1$
                                ▷ Loop backwards through the last 3 mins of data
 8:     while not $turnStartFound$ and not $turnEndFound$ and $i \geq 0$ do
 9:         $headingError \leftarrow |\, headingDifference(runway.hdg,\ last3Mins[i].hdg)\, |$
10:         if $headingError \geq 90$ and not $turnStartFound$ then
11:             $turnStartTime \leftarrow i$
12:             $turnStartFound \leftarrow true$
13:         end if
14:         if $headingError \geq 15$ and not $turnEndFound$ then
15:             $turnEndTime \leftarrow i$
16:             $turnEndFound \leftarrow true$
17:         end if
18:         $i \leftarrow i - 1$
19:     end while
20:     $approaches[apprID] \leftarrow$ store final turn meta-data
21:     return $(turnStartTime,\ turnEndTime)$
22: end function
```

Landing

The Landing phase is defined as the time from the beginning of the landing flare until the aircraft performs one of the following actions: *(i)* exits the landing runway, *(ii)* comes to a complete stop on the runway (full-stop), or *(iii)* when power is applied for takeoff in the case of a touch-and-go landing [27].

The Landing phase and result detection is able to differentiate between a full-stop, touch-and-go, and a go-around[1]. Pseudo-code for this process is given in Algorithm 4.

This detection algorithm iterates through time values starting where the final approach analysis finished (Algorithm 4 Lines 10-29). It continues to iterate while the aircraft is below 500 feet AGL; or if it is the aircraft's final landing and the time values run out, then it stops analyzing. While the algorithm iterates through the time values, it checks if the aircraft's IAS is less than or equal to 35 knots (Algorithm 4 Line 13). If this is true, then it is determined the aircraft is no longer traveling at a flying speed, thus it is making a complete stop. The stall speed of a Cessna 172S aircraft is 40 KIAS [29]; therefore, the value of 35 knots guarantees the aircraft cannot be flying. In order to detect a touch-and-go landing, the previous five elevation readings are stored and their average is calculated (Algorithm 4 Lines 20-28). If it is found the aircraft is not making a stop-and-go landing, then the average elevation for the last five seconds is checked to see if it is less than five feet AGL (Algorithm 4 Line 15). This means the aircraft is still at a flying speed (above 35 knots) and is also maintaining a stable elevation of five feet or less for at least five seconds.

Once the aircraft goes above 500 feet AGL or the time values run out, then the landing result is determined from the conditions found during the analysis

[1]Go-around is included here as a possibility for the result of a Landing phase since only one of the full-stop, touch-and-go, and go-around maneuvers can be executed after an Approach/Landing even though the aircraft does not physically contact the ground.

Table 1: Landing result types and their conditions.

Type	**Condition**
full-stop	Aircraft's indicated airspeed speed (IAS) falls below 35 knots
touch-and-go	Aircraft is not making a complete stop and maintains a stable altitude of five feet AGL or less for at least five seconds
go-around	All other cases

(Algorithm 4 Line 32 and Algorithm 5). If it was found the aircraft was making a complete stop, then a value of "full-stop" is stored. If it was not making a complete stop and had a relatively stable elevation of 5 feet or less above the runway, then a value of "touch-and-go" is stored. The final result type, "go-around", is used as a fall-through since there are only three classifications, as mentioned previously. The three landing result types and how they are detected are summarized in Table 1.

After the landing is classified, then it is determined whether there is a Takeoff phase that follows the current Landing phase. If the end of the data has been reached or a go-around is being performed (Algorithm 4 Line 33), there will not be a subsequent Takeoff phase. Otherwise, we need to find the transition from Landing to Takeoff. This is done by finding the index of the minimum RPM value between *landingStartTime* and *landingEndTime* (Algorithm 4 Line 36 and Algorithm 6). By using the minimum RPM value, we know all RPM values afterwards will be greater, which means the pilot will be using more throttle in order to takeoff. The *landingEndTime* is then reset to this transition mark. Lastly, the critical meta-data found during the analysis is stored.

Algorithm 4 Pseudo-code for function which detects the landing from its associated approach.

```
 1: function DetectLanding(approachEndTime, runway)
 2:     landingStartTime ← approachEndTime
 3:     i ← approachEndTime
 4:     airplaneAltitude ← data[i].altitude
 5:     heightAGL ← airplaneAltitude − runway.altitude
 6:     isFullStop ← false
 7:     isTouchAndGo ← false
 8:     elevations ← []
 9:     avg5SecElevation ← $5\,\text{ft} + 1$          ▷ value to guarantee first check passes
10:     while heightAGL < $500\,\text{ft}$ and i < data.length do
11:         if not isFullStop then
12:             airplaneIAS ← data[i].ias
13:             if airplaneIAS ≤ $35\,\text{kts}$ then
14:                 isFullStop ← true
15:             else if avg5SecElevation ≤ $5\,\text{ft}$ then
16:                 isTouchAndGo ← true
17:             end if
18:         end if
19:         i ← i + 1
20:         airplaneAltitude ← data[i].altitude
21:         heightAGL ← airplaneAltitude − runway.altitude
22:         if elevations.length < 5 seconds then
23:             elevations.append(heightAGL)
24:         else
25:             elevations.pop()
26:             elevations.append(heightAGL)
27:             avg5SecElevation ← avg(elevations)
28:         end if
29:     end while
30:     landingEndTime ← i
31:     isEndOfData ← landingEndTime == data.length − 1
32:     landingResult ← getLandingResult(isFullStop, isTouchAndGo)
33:     isFollowedByTakeoff ← not(isEndOfData or landingType == ‘go-around’)
          ▷ If landing is followed by a takeoff, then we need to find the transition
            from landing to takeoff
34:     if isFollowedByTakeoff then
35:         landingDataSlice ← get slice of data between landingStartTime and
                landingEndTime
                  ▷ Marks where the pilot is transitioning from landing to takeoff
36:         lastOccurrence ← getLastOccurrenceOfMinRPM(landingDataSlice)
37:         landingEndTime ← lastOccurrence
38:     end if
39:     approaches[apprID] ← store landing meta-data
40:     return (landingStartTime, landingEndTime)
41: end function
```

Algorithm 5 Pseudo-code for *getLandingResult* helper function.

```
1: function GetLandingResult(isFullStop, isTouchAndGo)
2:     if isFullStop then
3:         landingResult ← ‘full-stop’
4:     else if isTouchAndGo then
5:         landingResult ← ‘touch-and-go’
6:     else
7:         landingResult ← ‘go-around’
8:     end if
9:     return landingResult
10: end function
```

Algorithm 6 Pseudo-code for *getLastOccurrenceOfMinRPM* helper function.

```
1: function GetLastOccurrenceOfMinRPM(dataSlice)
2:     minRPM ← min(dataSlice[‘rpm’])
3:     lastOccurrence ← 0
4:     i ← 0
5:     while i < dataSlice.length do      ▷ loop through slice of data to find last
           occurrence of minimum RPM
6:         if dataSlice[i].rpm == minRPM then
7:             lastOccurrence ← i
8:         end if
9:     end while
10:    return lastOccurrence
11: end function
```

Phase of Flight Quality Analysis & Exceedance Detection

Approach

Along with analyzing the Approach phase, this section also details the algorithms for analyzing *(i)* the Final Turn subphase for an undershoot or overshoot and *(ii)* the pilot's self-defined glide path angle.

The algorithm for analyzing an Approach phase iterates through all the time values found during the phase identification stage (Algorithm 7 Lines 4-17). For each time value, the analysis for unstableness is performed. During this analysis, several flight parameters are checked against predetermined thresholds to see if any were exceeded (Algorithm 7 Lines 8-11). The values used for the thresholds are summarized in Table 2. A *true* value for a condition means the parameter is stable. Thus, if any of the parameters are unstable, *isUnstable* will result to being *true*, meaning the entire aircraft is in an unstable state (Algorithm 7 Line 12). If the aircraft is found to be unstable, the corresponding time value is stored as well as the parameter values that caused the unstableness (Algorithm 7 Line 14).

Final Turn.

The Final Turn subphase is very critical for achieving a flight path aligned with the runway. Since the end of the turn occurs fairly late in the Approach phase, any mistakes can greatly detriment the pilot's ability to be stabilized by 200 ft AGL. If the pilot makes a turn that is too sharp (undershoot) or too wide (overshoot), they may have to make a large corrective maneuver to re-align themselves, which could stall the aircraft if performed incorrectly and potentially result in a loss of control (LOC). Stalls and loss of control events contributed to 52.0% and 17.4% of all landing accidents in 2017 [5], respectively.

Analyzing this subphase only requires the end time value and runway found

Table 2: Stabilized Approach criteria for Cessna 172S [1].

Parameter	Description	Value
F	Flight path correct	Less than 10° off runway heading, less than 50 ft left or right of the runway center line (cross track error)
L	Landing configuration correct	N.A.
A	Airspeed proper	Indicated airspeed (IAS) within 55-75 kts
P	Power setting appropriate	N.A.
S	Sink rate appropriate	Vertical speed indicated (VSI) does not exceed -1000 ft/min

Algorithm 7 Pseudo-code for function which analyzes an approach for unstableness.

```
1:  function AnalyzeApproach(startTime, endTime, runway)
2:      approachDataSlice ← get slice of data between startTime and endTime
3:      $i \leftarrow 0$
4:      while $i <$ approachDataSlice.length do
5:          airplaneHdg ← approachDataSlice[i].hdg
6:          airplaneIAS ← approachDataSlice[i].ias
7:          airplaneVSI ← approachDataSlice[i].vsi
8:          airplanePoint ← approachDataSlice[i].geoPoint
9:          headingIsStable ← $180° - \mid \mid$ runway.hdg − airplaneHdg $\mid - 180° \mid \leq 20°$
10:         crossTrackIsStable ← calculateCrossTrack(
                airplanePoint, airplaneHdg, runway) $\leq 50$ ft
11:         iasIsStable ← $55$ kts $\leq$ airplaneIAS $\leq 75$ kts
12:         vsiIsStable ← airplaneVSI $\geq -1000$ ft/min
13:         isUnstable ← not (headingIsStable and crossTrackIsStable and
                iasIsStable and vsiIsStable)
14:         if isUnstable then
15:             approaches[apprID] ← store index as unstable and corresponding
                unstable parameter values
16:         end if
17:         $i \leftarrow i + 1$
18:     end while
19: end function
```

Table 3: Final Turn matrix of the combinations of roll direction and cross track error.

Cross Track / Direction	< 0 ft	> 0 ft
Left	Undershoot	Overshoot
Right	Overshoot	Undershoot

during the identification stage. For this single time value, the aircraft's cross track error is calculated (Algorithm 8 Line 5). Next, the direction of the turn is determined by calculating which roll attitude direction was greater[2] (Algorithm 8 Lines 6-12). The severity of the cross track error is then determined (Algorithm 8 Lines 13-19). A Level 1 error is a value greater than 25 feet, while a Level 2 error is a value greater than 100 feet. The turn error is determined next based on the roll direction and direction of the cross track error (Algorithm 8 Lines 20-32). For example, if the pilot rolled left and had a negative cross track error[3], it is considered an "undershoot". See Table 3 for all possible combinations of roll direction and cross track error. However, if the cross track error is less than a Level 1 risk, then the turn is considered to be safe and a Risk Level 0 is stored (Algorithm 8 Line 18). Lastly, the turn error and severity are stored. See Figure 5 for visualizations of several different Final Turn scenarios.

Explain why these val- ues were cho- sen.

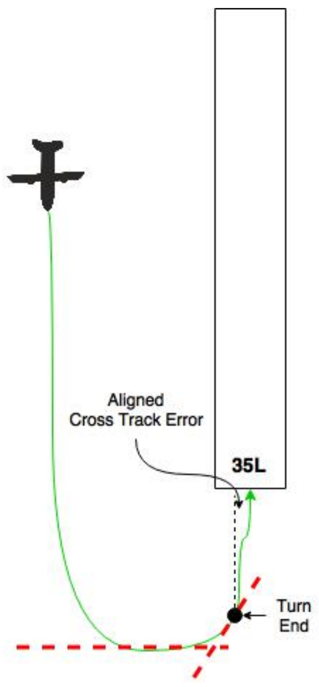
If a Final Turn was not found in the detection phase (due to the pilot performing a straight-in approach), the analysis stage will be skipped.

Self-defined glide path.

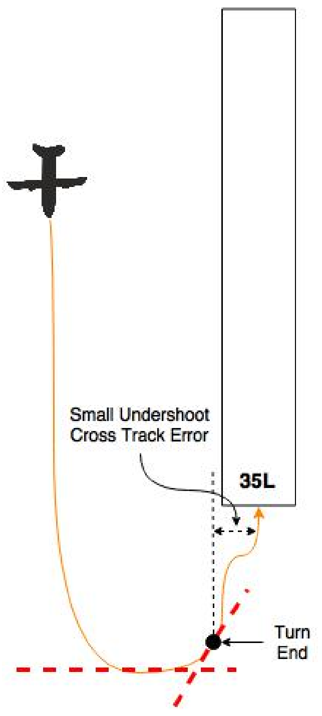
A majority of runways in the U.S. publish an ideal glide slope that all pilot's should adhere to. However, not all runways have a published glide slope. Therefore, a method for analyzing the aircraft's actual glide path angle (GPA) during the Final Approach is needed in order for the pilot to be able to see what

[2]If the aircraft rolls to the left, it is recorded as a negative degree and vice versa if the aircraft rolls to the right. This is why we find the minimum roll attitude as the highest degree in which aircraft rolled left, and the maximum roll attitude as the highest degree in which the aircraft rolled right.

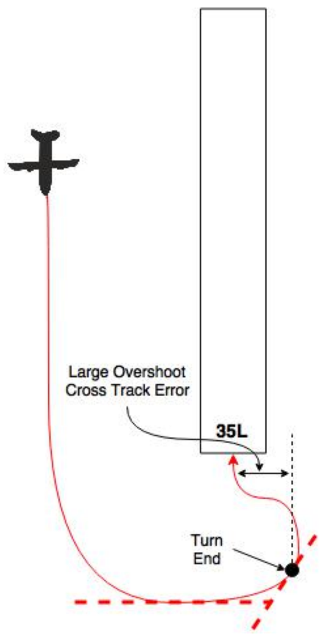
[3]Meaning they are left of the runway's centerline.



(a) Aligned.



(b) Undershoot. In this case, it is a small severity (Level 1) and color-coded as orange.



(c) Overshoot. In this case, it is a large severity (Level 2) and color-coded as red.

Figure 5: Examples showing various Final Turn qualities.

Algorithm 8 Pseudo-code for function which analyzes the quality of a Final Turn phase.

```
function AnalyzeFinalTurn(startTime, endTime, runway)
    turnDataSlice ← get slice of data between startTime and endTime
    airplaneHdg ← turnDataSlice[endTime].hdg
    airplanePoint ← turnDataSlice[endTime].geoPoint
    crossTrackError ← calculateCrossTrack(
            airplanePoint, airplaneHdg, runway)
    leftDirection ← | min(turnDataSlice['roll']) |
    rightDirection ← | max(turnDataSlice['roll']) |
    if leftDirection > rightDirection then
        rollDirection ← 'left'
    else
        rollDirection ← 'right'
    end if
    if | crossTrackError | > 100 ft then                ▷ Level 2
        severity ← 2
    else if | crossTrackError | > 25 ft then            ▷ Level 1
        severity ← 1
    else
        severity ← 0
    end if
    if rollDirection == 'left' then
        if crossTrackError < 0 then
            turnError ← 'undershoot'
        else
            turnError ← 'overshoot'
        end if
    else
        if crossTrackError > 0 then
            turnError ← 'undershoot'
        else
            turnError ← 'overshoot'
        end if
    end if
    approaches[apprID] ← store severity and error
    return (severity, turnError)
end function
```

their average GPA was and how well they adhered to it. This is why we termed this method a "self-defined glide path angle". Figure 6 shows an example of what the self-defined glide path analysis is performing.

Major deviations from the ideal glide slope can be very costly. For example, if the pilot is approaching at a steep angle, a hard landing or a landing short of the runway can occur. On the other hand, if the pilot is approaching at a shallow angle, a runway overrun can occur.

First, the slice of data for the corresponding Approach phase found during the detection stage is obtained (Algorithm 9 Line 2). Next, a simple linear regression using the least squares approach is calculated (Algorithm 9 Line 3) using the aircraft's height AGL (dependent variable) over all the time values (independent variable). From that calculation we obtain the y-intercept, slope, and r-value (correlation coefficient) of the linear regression model (Algorithm 9 Lines 4, 5, 15). Then, all the necessary values for computing the pilot's defined GPA are calculated (Algorithm 9 Lines 6-13). Once all the supporting values are found, then the actual GPA is calculated using the arctan of the predicted vertical distance dropped over the traveled horizontal distance (Algorithm 9 Line 14). Furthermore, the square of the r-value is calculated (Algorithm 9 Line 16), which explains how well the pilot's glide path "fit" the ideal glide path. Lastly, the calculated values and meta-data are stored

Landing

Kelton: Probably am going to remove this section as I don't really do any quality analysis of the Landing phase (only landing type detection as in previous Section).

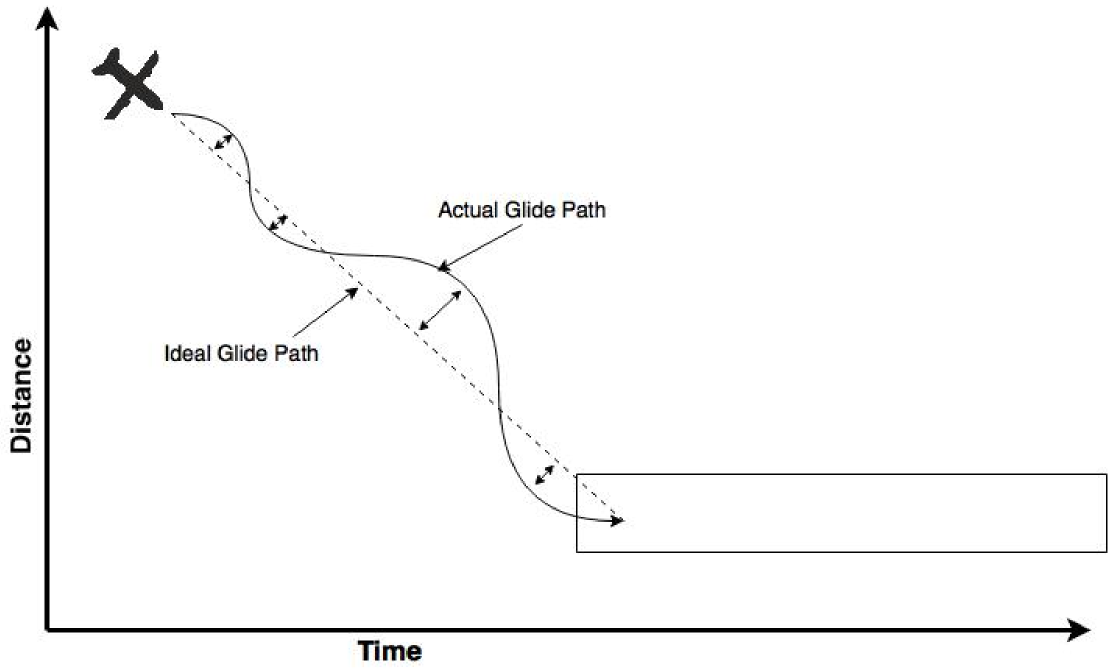


Figure 6: Example showing the self-defined glide path angle analysis. This shows a side view of the pilot oscillating about the glide slope during the Approach phase. The calculation uses a simple linear regression of the aircraft's vertical distance over time fitted using the least squares approach. The solid line is the aircraft's actual glide path while the dotted line is the ideal glide path.

Algorithm 9 Pseudo-code for function which analyzes the quality of the aircraft's glide path angle during the Approach phase.

```
1: function AnalyzeGlidePath(startTime, endTime, runway)
2:     approachDataSlice ← get slice of data between the approach startTime
           and endTime
3:     regressionResult ← linearRegression(
           approachDataSlice[‘time’], approachDataSlice[‘agl’])
4:     yIntercept ← regressionResult.intercept
5:     slope ← regressionResult.slope
6:     maxDistance ← max(approachDataSlice[‘distance’])
7:     minDistance ← min(approachDataSlice[‘distance’])
8:     horizontalDistance ← maxDistance − minDistance
9:     maxTime ← max(approachDataSlice[‘time’])
10:    minTime ← min(approachDataSlice[‘time’])
11:    predictedMaxAGL ← slope ∗ maxTime + yIntercept
12:    predictedMinAGL ← slope ∗ minTime + yIntercept
13:    predictedVerticalDistance ← predictedMaxAGL − predictedMinAGL
14:    actualGlidePathAngle ← degrees(
           atan(predictedVerticalDistance / horizontalDistance))
15:    pearsonsR ← regressionResult.rvalue
16:    rSquared ← pearsonsR ∗ pearsonsR
17:    approaches[apprID] ← store self-defined glide path meta-data
18:    return (actualGlidePathAngle, rSquared)
19: end function
```

Grading Metrics

When creating the risk level metrics to be used for grading the Approach analysis data, we wanted to ensure they were backed by statistics obtained from the results from the sample set of flights. Towards that goal, the risk level metrics have been created from the data found during the Approach quality analysis. For each parameter of concern, the recorded values across all Approach phases in the sample set were used to create a normalized histogram showing the probability density of each value range. From these histograms, the mean and standard deviation were calculated in order to create a best-fit line to overlay the histogram. The charts were then analyzed by an aviation statistics expert at the University of North Dakota who gave his opinion on reasonable values to use for Risk Level 1 and 2 value ranges based on each mean, standard deviation, and best-fit line. Even though those elements were created from the analysis statistics, the aviation expert wanted to also ensure the "safe value ranges" (Risk Level 0) did not conflict with the values published in UND's standardization manual [1] and the Cessna C172S Pilot's Operating Handbook (POH) [29].

After the risk level metrics have been established, the Approach quality analysis results will be re-processed and graded according to the metrics. The resulting grade is then stored along with the other generated Approach analysis data within the database. The specific details of the grading results from using the risk level metrics will be discussed further in Chapter 5.

Web Interface

New web visualization tools were developed to display the results generated by the Go-Around Detection Tool. There is a separate tool created for the Approach, Final Turn, and self-defined glide path analyses. Since the generated results can create numerous data-points for trend analysis, only the most useful charts will be implemented to begin with. Each visualization will utilize AJAX[4] to dynamically and asynchronously load the corresponding results from the back-end server. This will allow a user to quickly change their given inputs and receive an updated display with a minimal amount of wait time. New visualizations can be easily created in the future for different purposes since all of the front-end displays are modular from the generated data stored in the database.

[4]Asynchronous JavaScript and XML

CHAPTER 4

IMPLEMENTATION

Programming Languages and Libraries

The Python programming language was used for implementing the flight analysis due to its ease of use, its reputable scientific and graphing libraries, and the ability to quickly produce a viable application. The libraries utilized are MySQLdb[1] for interacting with the MySQL database; matplotlib[2] for graphing flight parameters in the early stages of the application; NumPy[3], Scipy[4], and Pandas[5] for their scientific and vectorized functions; and the geodesy scripts created by Chris Veness[6]. All source code is available at `https://github.com/KeltonKarboviak/NGAFID`.

For the back-end of the web interface, Laravel[7] was used as the PHP framework. For the front-end, the following technologies were used: jQuery[8] & jQuery UI[9], Bootstrap[10] for CSS styling, OpenLayers[11] for creating an interactive map, OpenStreetMap[12] for the images used by OpenLayers, and Turf.js[13] for its geodesy functions. All source code for the web interface is available at `https://github.com/travisdesell/ngafid`.

[1]v1.3.12: http://mysql-python.sourceforge.net/MySQLdb.html
[2]v2.2.2: https://matplotlib.org/
[3]v1.14.0: http://www.numpy.org/
[4]v1.0.0: https://docs.scipy.org/doc/scipy/reference/index.html
[5]v0.22.0: https://pandas.pydata.org/
[6]http://www.movable-type.co.uk/scripts/latlong.html
[7]v5.0: https://laravel.com/
[8]v2.2.4: https://jquery.com/
[9]v1.11.2: https://jqueryui.com/
[10]v3: https://getbootstrap.com/docs/3.3/
[11]v4.6.4: http://openlayers.org/
[12]https://www.openstreetmap.org
[13]v5.1.5: http://turfjs.org/

Hardware Specs

All experiments were performed on a 2013 Mac Pro running macOS 10.11.6 with a 3.5 GHz 6 hyper-threaded core Intel Xeon E5 processor (for a total of 12 logical processing cores). The machine also has 32 GBs of 1866 MHz DDR3 ECC RAM.

Parallelization

The application was originally created to process the flight data in a linear fashion. This proved to be fairly time consuming when running the application in batch mode with the significant number of flights contained in the NGAFID. In order to improve the performance and efficiency of the application, Python's built-in multiprocessing module was used. The parallel application uses the simple Producer-Consumer model in which the parent process acts at the Producer by enqueuing all of the unique flight identifiers onto a queue and the child subprocesses act as the Consumers by dequeuing a flight identifier then processing it. The multiprocessing module was chosen over the built-in threading module due to the issue with Python's Global Interpreter Lock (GIL) effectively restricting bytecode execution to a single core [30]. This makes the threading module unusable for long-running CPU-bound tasks, which this application heavily relies on.

CHAPTER 5

RESULTS

Experiments

The experiments were run using Cessna 172S flight data produced by students at the University of North Dakota during the month of September 2015. Student flight data is ideal for unstable analysis testing as it contains very noisy data, which provides a diverse array of flying patterns. A random sample of 100 flights was chosen for the experiments.

First, the application was run against the 100 flights to obtain the automated analysis results. The same 100 flights were then manually analyzed in order to get human results for the phase identification, which could be compared to the automated results then determine the accuracy of the application. The test of the 100 flights was also run ten times each with the single-process version and the multi-process version as previously described. This was done in order to compare and contrast the performance of the separate versions.

Accuracy of Phase Identification

The manual validation was performed using a combination of tools available on the NGAFID website: the Cesium flight reanimation tool and the Keyhole Markup Language (KML) generator to visualize the flight path in *Google Earth* [31] (see Figure 7).

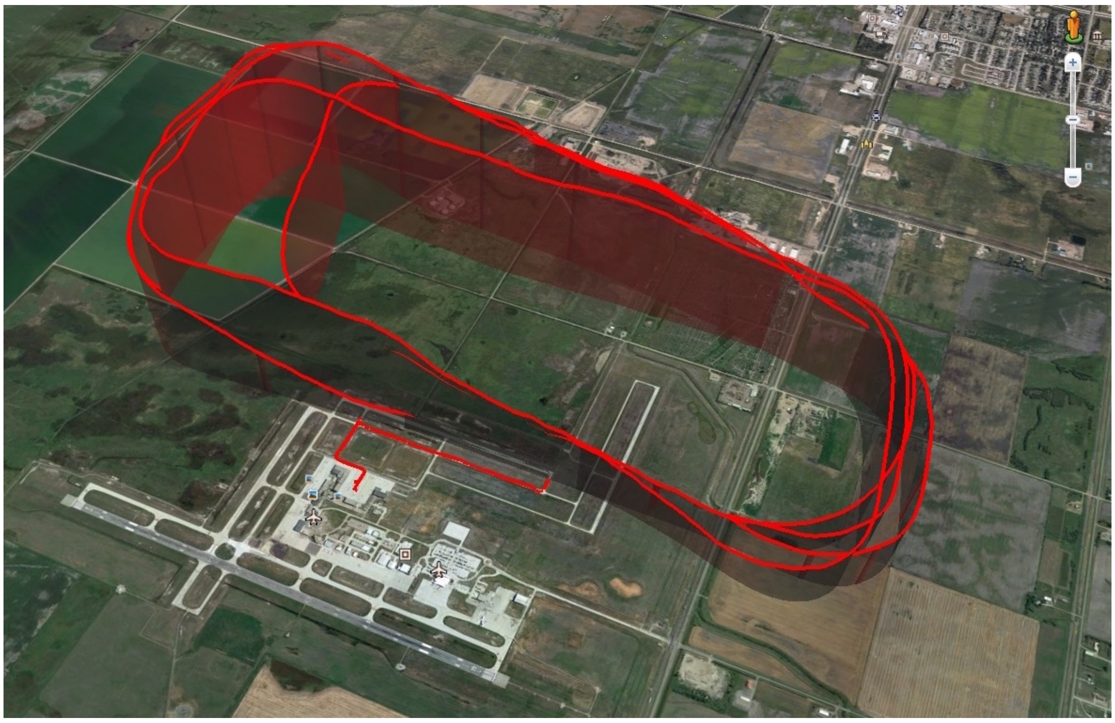


Figure 7: Example of using a KML file to visualize a flight path in *Google Earth*. This flight visualization is an example of a student flight that has multiple Approach phases.

Approach

The Go-Around Detection Tool generated a total of 380 approaches for the 100 flights that were tested. As seen in Figure 7, student flights typically consist of multiple approaches as this is something that needs to be practiced. Out of the total; there are 3730 (98.16%) true positives, five (1.32%) false positives, and two (0.53%) false negatives. These results can also be found in Figure 8.

In the context of this application, a true positive is a case where the tool correctly indicates that an approach is occurring during a specified time frame.

A false positive occurs when the tool indicates that an approach is occurring but is not in reality. Typically, a false positive occurs when the flight data has invalid values for about the first ten rows, which then throws off the beginning of the algorithms. This happens infrequently, but could be accounted for in a future work by sanitizing the data before analysis.

A false negative is the exact opposite where the tool indicates that an approach is not occurring but it is in reality. Typically, a false negative occurs when the approached runway's geological data is not contained within the database. These types of occurrences should stop once the airport and runway databases are expanded with more entries.

The tool misclassified the approached runway 13 times (3.42%). A runway is misclassified when the difference between the aircraft and runway headings is greater than 20°. This occurs during the runway detection portion of the approach analysis algorithm, and the algorithm either returns a *null* runway or an incorrect runway due to the large heading difference. Lastly, there was a total of 42 (11.05%) approaches that were given a *null* runway. This number overlaps with some of the 13 misclassifications, while the rest are due to a lack of runway information as mentioned previously.

In this same context, it is difficult to quantify the number of true negatives since these would be cases where the tool correctly indicates that an approach is not occurring. The difficulty lies in how to define a single occurrence. Should a single true negative be counted for every second the tool indicates that an approach is not occurring? If so, then this would create a numerous amount of true negatives and would dilute the percentages of the other statistics, which are more important in this application.

The validation results demonstrate that the Go-Around Detection Tool is exceptionally accurate in its ability to appropriately detect and classify most approaches in a flight.

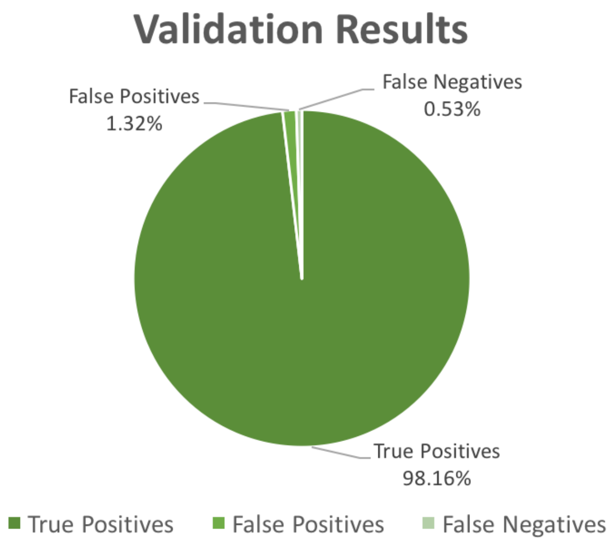


Figure 8: Pie chart showing the manual validation results including true positives, false positives, and false negatives.

Quality Analysis

Approach

The results of the application have provided many possibilities for statistical analysis since numerous statistics can be calculated from the generated approach data. This can be seen in Figures 9a to 9d in which a sample of the possible results were calculated from the experiments of the 100 flights used in this research. With these various results, trends can be found in the data that has been analyzed. For example, we can see in Figure 9a that out of the 380 approaches in the sample data, 57.11% (217) were stable and 42.89% (163) were unstable. By drilling down into that data, we can see the frequency for each of the landing types for stable and unstable approaches. Figure 9b depicts this more detailed information and shows that full-stop landings occur most frequently for both stable and unstable approaches. This result is not very surprising for stable approaches; however, it is very undesirable for unstable approaches. If we look even further into the proportions for unstable approaches alone (Figure 9c), we see that an unstable approach resulted in a go-around only 34.97% of the time.

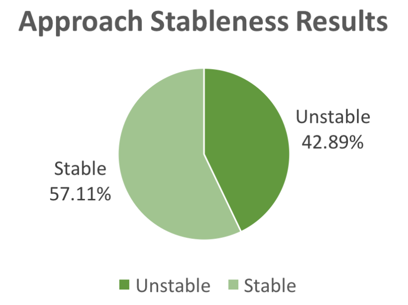
This is far lower than the hopeful 100%, but was expected to be approximately 20% by our aviation safety experts. As mentioned previously, this is largely due to pilot misjudgment since all the analyzed flights were piloted by aviation students; meaning they are still learning and are not yet professionals.

When looking at the unstable approaches and the parameters that caused them (Figure 9d), additional interesting results can be found. We found the parameter that was exceeded the most was heading with 91 occurrences. Heading was not predicted to be the leading cause of unstable approaches, but our safety experts believe the 10° threshold (as defined in Table 2) may be too strict. Indicated airspeed was the second highest, but was predicted to be the leading cause since it was stated by our aviation safety experts to be a trend for UND's student pilots to be going too fast on final approaches.

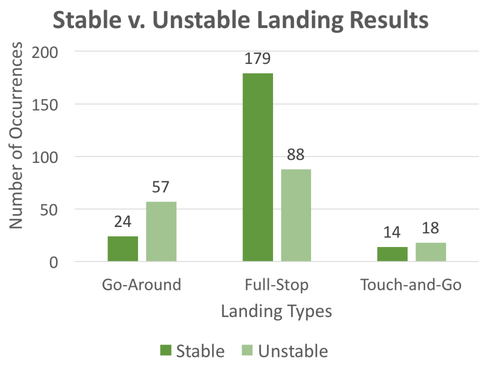
Another interesting set of statistics that can be drawn from the analysis are parameter value frequencies. Creating histograms of the values for each parameter during all Approach phases can show the values that occur most frequently (highest density). Figures 10a to 10d visualize these histograms and give the corresponding mean and standard deviation values. These graphs are able to show how well pilots are adhering to the published stabilized approach criteria (see Table 2). As mentioned previously, the standard deviations will be used in defining the grading metrics and will be discussed in more detail later in this Chapter.

Final Turn.

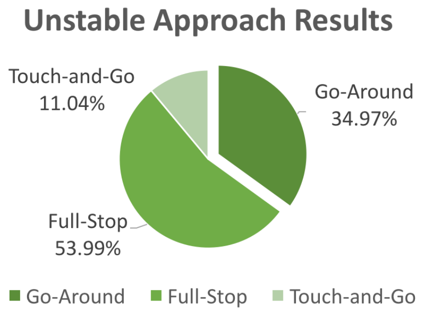
Out of the 380 detected approaches; 262 (68.95%) had a Final Turn subphase, 76 (20.00%) performed a straight-in approach, and 42 (11.05%) were not able to detect the runway and, therefore, could not detect whether a Final Turn was performed. Figure 11 depicts these ratios. As mentioned earlier, the 42 *null* runways will be drastically reduced in the future once additional airports and runways are added to the geological database. Once the airport and runway



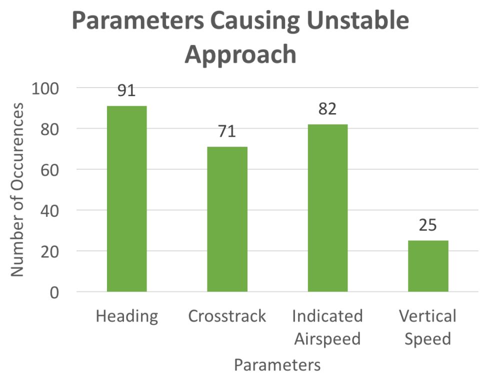
(a) Pie chart showing the number of stable approaches compared to the number of unstable approaches.



(b) Frequency of the occurrences of each landing type for stable and unstable approaches.

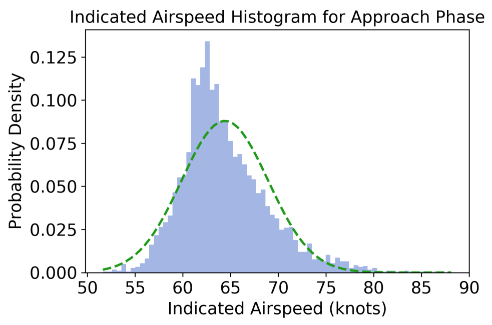


(c) Pie chart comparing the number of occurrences for each landing type after an unstable approach.

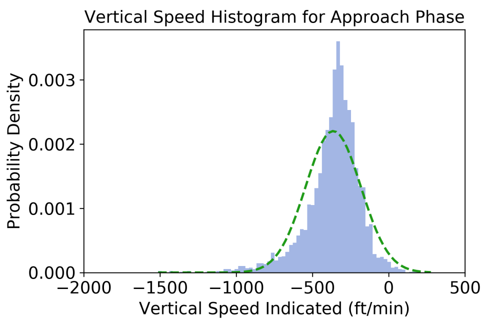


(d) Frequency of parameters that caused an aircraft to be unstable during an approach. Note that a single approach can have multiple unstable parameters, which causes the sum of the occurrences to not equal the total number of unstable approaches.

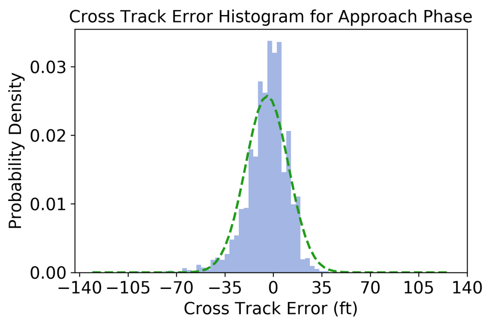
Figure 9: Sample set of the statistics and trends that can be found from the automated analysis results.



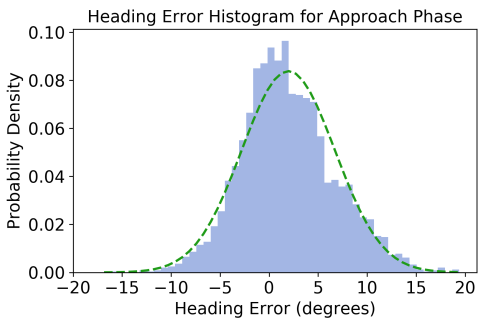
(a) Indicated airspeed.



(b) Vertical speed indicated.



(c) Cross track error.



(d) Heading error.

Figure 10: Histograms showing the frequencies of values for each parameter during all Approach phases. Each graph also has a dotted best-fit line to show how close the frequencies adhere to a normal distribution.

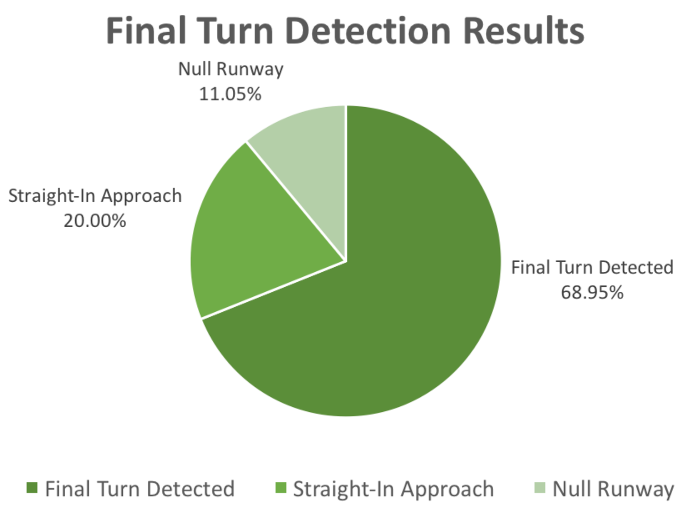


Figure 11: Pie chart showing the results from the Final Turn detection algorithm.

databases are more complete, the runway error rate should become much more acceptable.

Figure 12 gives a comparison of the number of occurrences for each turn error type and Risk Level classification. This graph is displaying the subset of 262 detected Final Turn subphases found in Figure 11. The figure shows that 76.34% of the Final Turns resulted in an undershoot and a large proportion (45.420%) resulted in a Risk Level 2 undershoot. Those statistics are definitely interesting and are an example of an anomaly that is worth looking into by an aviation expert. One explanation could be that there is frequently a strong wind component against the aircraft, which could cause the numerous undershoots. Further analysis such as this could be performed as a future work since wind and other meteorological factors were not taken into consideration in the analyses within this research.

Grading Metrics: Define From Parameter Frequencies

As mentioned in Chapter 3, the goal for creating the Risk Level metrics is to use statistical results from the Approach quality analysis to determine reasonable

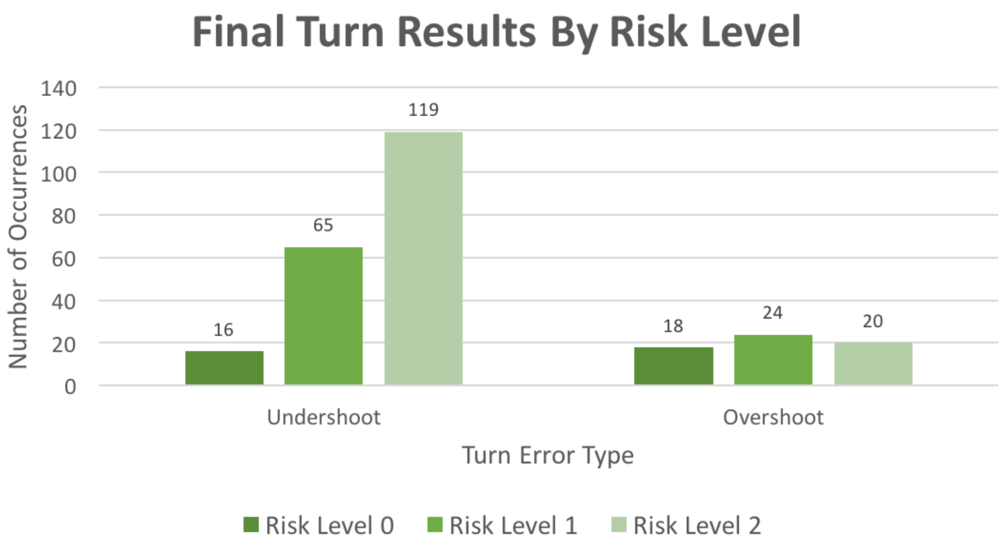


Figure 12: Frequency of the occurrences of each turn error type for Risk Levels 0, 1, and 2.

values that still adhere to the published stable value ranges. Figures 10a to 10d above contain normalized histograms showing the probability density for the parameter values. These graphs were analyzed by an aviation statistics expert from UND in order to create a safe range (Risk Level 0), a moderate risk range (Risk Level 1), and a high risk range (Risk Level 2) for each parameter of concern. These graphs will be re-used in the following Sections, but will have additional information showing the value ranges that were created. The Risk Level 1 values will be a yellow dotted line, while the Risk Level 2 values will be a red dotted line. A summary of the defined grading metrics can be seen in Table 4 at the end of this Section.

Indicated Airspeed Between 55 and 75 knots

The indicated airspeed values had a mean of 64.401 knots and a standard deviation of 4.535 knots. The aviation expert stated that the UND standardization manual [1] has a strict safe range of 61 -0/+5 knots, and thus anything less than 61 knots should be automatically classified as a Risk Level 2. He advised the higher Risk Level 1 should begin with any value greater than 66

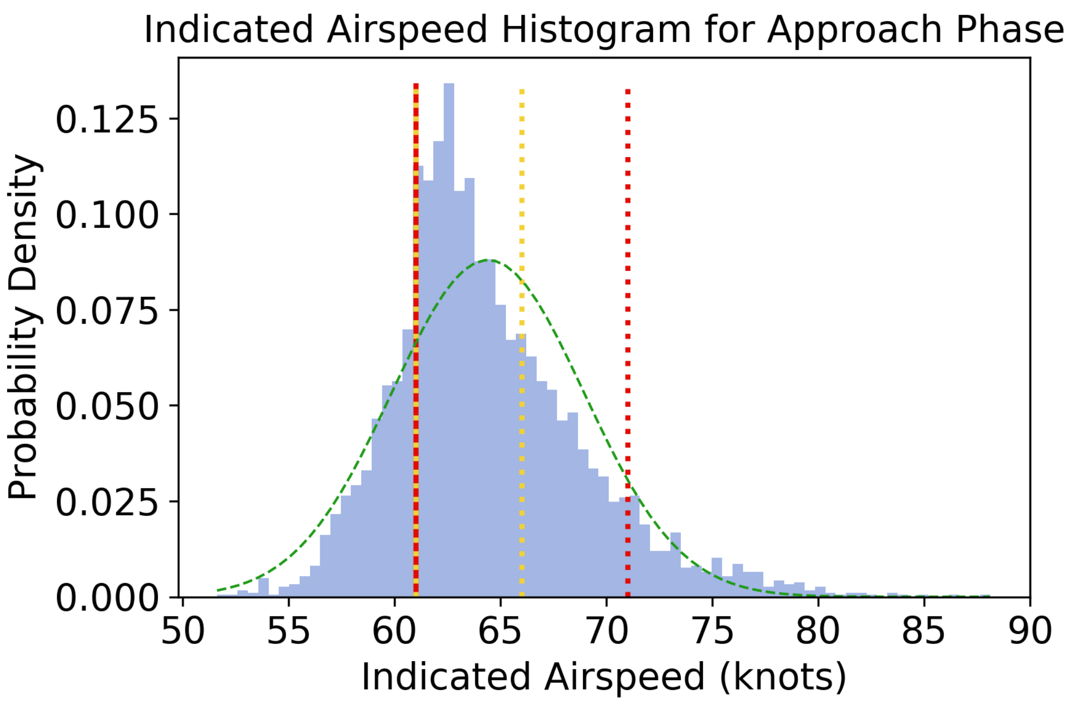


Figure 13: Histogram for indicated airspeed ($\mu = 64.401, \sigma = 4.535$). The safe range is between 61 and 66 knots. There is no lower Risk Level 1 since the Risk Level 2 is anything less than 61 knots. The higher Risk Level 1 range is between 66 and 71 knots, and the Risk Level 2 is anything greater than 71 knots.

knots due to the same rule. Lastly, he advised setting the higher Risk Level 2 at 71 knots in order to use the consistent 5 knots increment, which is also relatively close to one standard deviation. As is shown in Figure 13, the graph is slightly skewed to the right following the -0/+5 knots rule, which is more lenient towards faster speeds than slower speeds.

Vertical Speed Indicated Greater Than -1000 ft/min

The vertical speed indicated values had a mean of -364.528 ft/min and a standard deviation of 181.210 ft/min. Although the UND standardization manual states that a vertical speed greater than -1000 ft/min should be achieved for a stabilized approach, the aviation expert stated that a safe range of -800 to -500 ft/min is typically suggested instead of the wide range provided in the

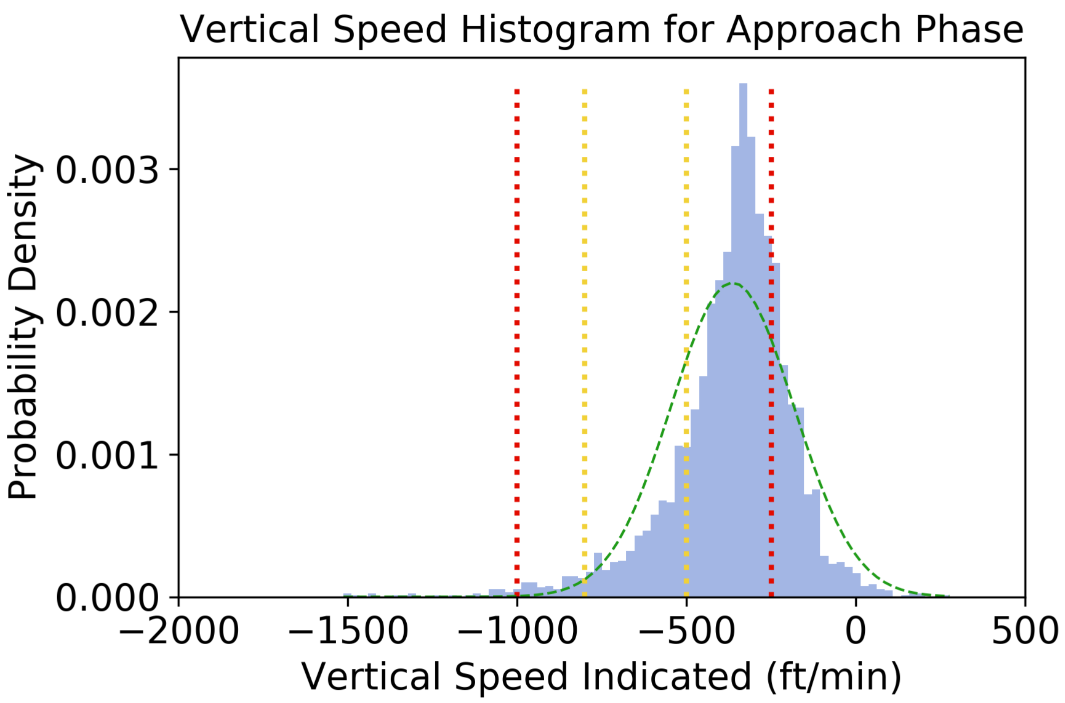


Figure 14: Histogram for vertical speed indicated ($\mu = -364.528, \sigma = 181.210$). The safe range is between -800 and -500 ft/min. The lower Risk Level 1 range is between -1000 and -800 ft/min, and the Risk Level 2 is anything less than -1000 ft/min. The higher Risk Level 1 range is between -500 and -250 ft/min, and the Risk Level 2 is anything greater than -250 ft/min.

manual. The lower Risk Level 1 is any value that is less than -800 ft/min, and the lower Risk Level 2 is any value less than -1000 ft/min in order to adhere to the stable limit given in the manual. The higher Risk Level 1 is any value greater than -500 ft/min, and the higher Risk Level 2 is any value greater than -250 ft/min. These higher limits were chosen because if the aircraft is descending at 250 ft/min, it will typically result in an unsafe and shallow glide slope. Figure 14 shows these limits and shows that the values are slightly skewed to the left, which corresponds to the risk limits that we've set.

Absolute Cross Track Error Less Than 50 ft

The cross track error values had a mean of -4.542 feet and a standard deviation of 15.499 feet. Figure 15 displays the histogram for these values and it can be

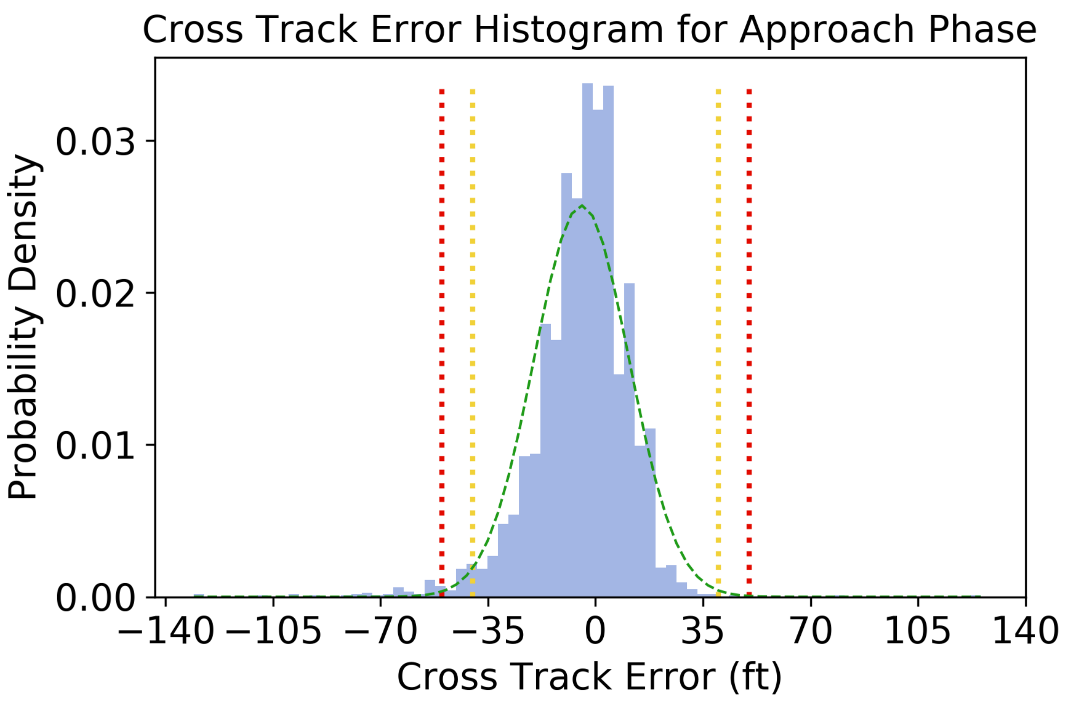


Figure 15: Histogram for cross track error ($\mu = -4.542, \sigma = 15.499$). The safe range is between -40 and 40 ft. The lower Risk Level 1 range is between -50 and -40 ft, and the Risk Level 2 is anything less than -50 ft. The higher Risk Level 1 range is between 40 and 50 ft, and the Risk Level 2 is anything greater than 50 ft.

seen that the graph is relatively normal with a slight skew to the left. The aviation expert stated that a deviation in cross track error is not as risky as a deviation in airspeed or vertical speed. Thus, a wider safe range was defined as -40 feet to 40 feet, which is about where the tails wane on both sides. Consequently, the lower and higher Risk Level 1 ranges are -50 feet to -40 feet and 40 feet to 50 feet, respectively. The lower and higher Risk Level 2 values are then -50 feet and 50 feet, respectively, in order to correspond with the original stable limits that were used.

Absolute Heading Error Less Than 10 degrees

The heading error values had a mean of 1.958° and a standard deviation of 4.761°. Similar to cross track error, the aviation expert stated that heading error

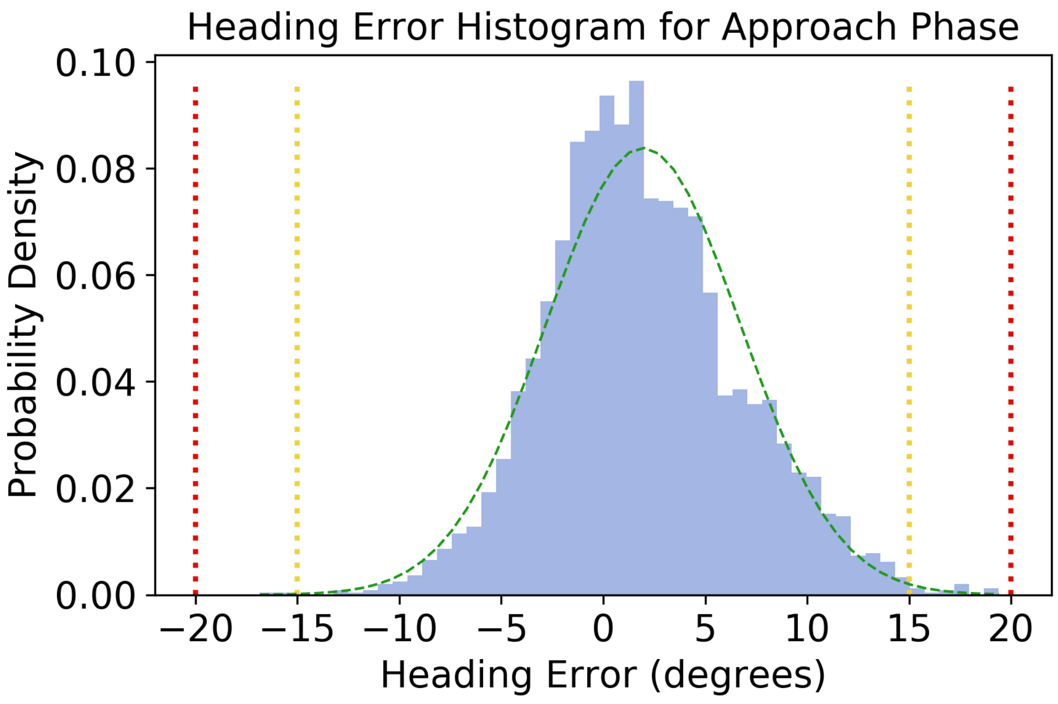


Figure 16: Histogram for heading error ($\mu = 1.958, \sigma = 4.761$). The safe range is between -15 and 15 degrees. The lower Risk Level 1 range is between -20 and -15 degrees, and the Risk Level 2 is anything less than -20 degrees. The higher Risk Level 1 is between 15 and 20 degrees, and the Risk Level 2 is anything greater than 20 degrees.

is not as risky as error in the other parameters. He also mentioned that heading error is slightly more difficult to judge without knowing the wind component on the aircraft since the pilot may have to purposely direct the aircraft several degrees off-center in order to counteract the push of the wind. Both of these facts means that the safe range for heading error will contain a majority of values. With that said, the expert advised using a safe range of $-15°$ to $15°$. The lower and higher Risk Level 1 ranges are from $-20°$ to $-15°$ and $15°$ to $20°$, respectively. Consequently, the lower and higher Risk Level 2 values are $-20°$ and $20°$, respectively. The histogram for heading error, as shown in Figure 16, appears to best fit a normal distribution.

Table 4: Defined Risk Levels during Approach for a Cessna C172S

Event		**Risk Level 1**	**Risk Level 2**
Indicated Airspeed	low	N.A.	61 knots
	high	66 knots	71 knots
Vertical speed	low	-800 ft/min	-1000 ft/min
	high	-500 ft/min	-250 ft/min
Cross track error	low	-40 ft	-50 ft
	high	40 ft	50 ft
Heading error	low	$-15°$	$-20°$
	high	15°	20°

Grading Metrics: Experiment Results

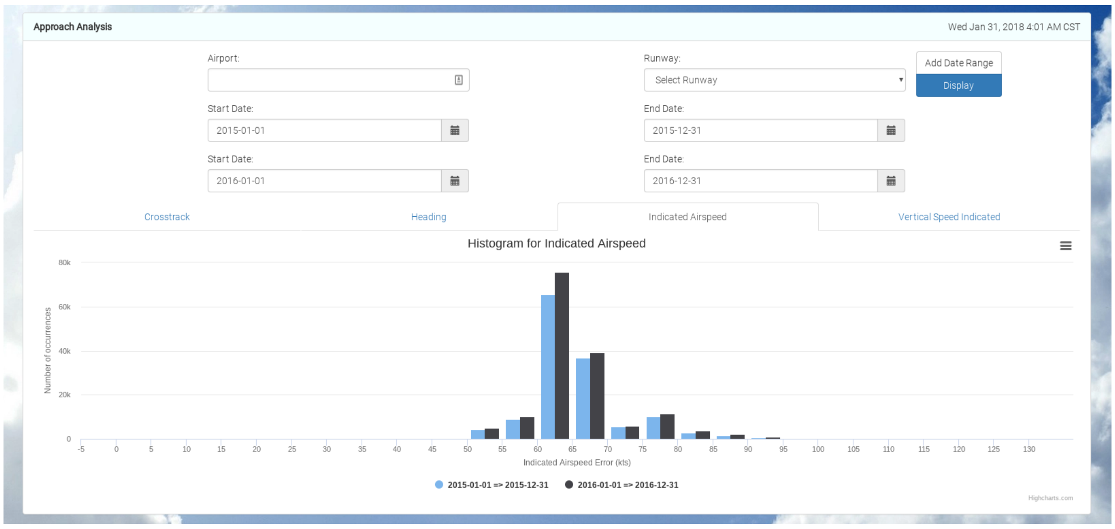
Kelton: Results found from using metrics on analysis results.

Web Interface

This Section details the newly developed web pages for the NGAFID, which dynamically display results based on the user's chosen filters. At the time of this writing, there have been new tools developed for the Approach, Final Turn, and self-defined glide path analyses. Each tool will be discussed further in the subsequent Subsections.

Approach

A new web page was implemented in the NGAFID for the purpose of dynamically displaying the Approach analysis results produced by the Go-Around Detection Tool to users (Figure 17). The results are given in four tabs, one for each parameter, as histograms over a specified date range. A user is able to dynamically add additional date ranges, which will create an additional


Approach Analysis

Wed Jan 31, 2018 4:01 AM CST

Airport:

Runway: Select Runway

Start Date: 2015-01-01

End Date: 2015-12-31

Start Date: 2016-01-01

End Date: 2016-12-31

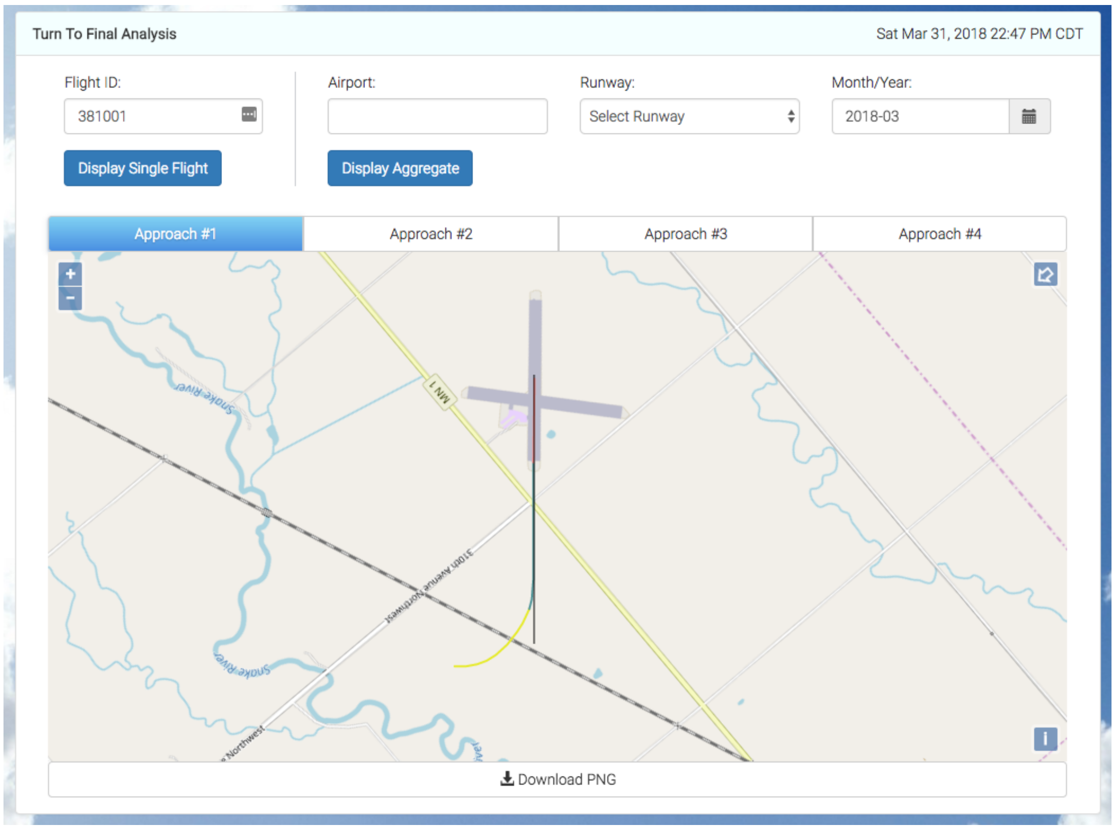
Figure 17: A screenshot of the Approach analysis tool on the NGAFID. It is showing the histogram for indicated airspeed error with two date range filters: 2015-01-01 to 2015-12-31 and 2016-01-01 to 2016-12-31. The frequency of exceedances can be seen with all values that fall outside of the 55-75 knots range.

series in the chart for comparison. This feature can be used to detect changes in trends over time. A user is also, optionally, able to filter the results to an airport and further filter to a single runway. This will allow users to identify trends that are potentially occurring at a specific runway but not at any other runways.

Final Turn

The tool developed for analyzing Final Turn phases in the NGAFID was implemented with two modes: *(i)* “Single Flight” and *(ii)* “Aggregate”.

For the “Single Flight” mode, the user can input an ID for a specific flight they’d like to analyze (Figure 18). Once the user clicks the “Display Single Flight” button, the interactive map then dynamically transitions to the first approach for that flight. The map will only display one approach at a time; although, there are tabs across the top for each approach which the user can choose. Once a different tab is chosen, the map automatically transitions the view to that corresponding approach. The flight path shows different color


Turn To Final Analysis

Sat Mar 31, 2018 22:47 PM CDT

Flight ID: 381001

Airport:

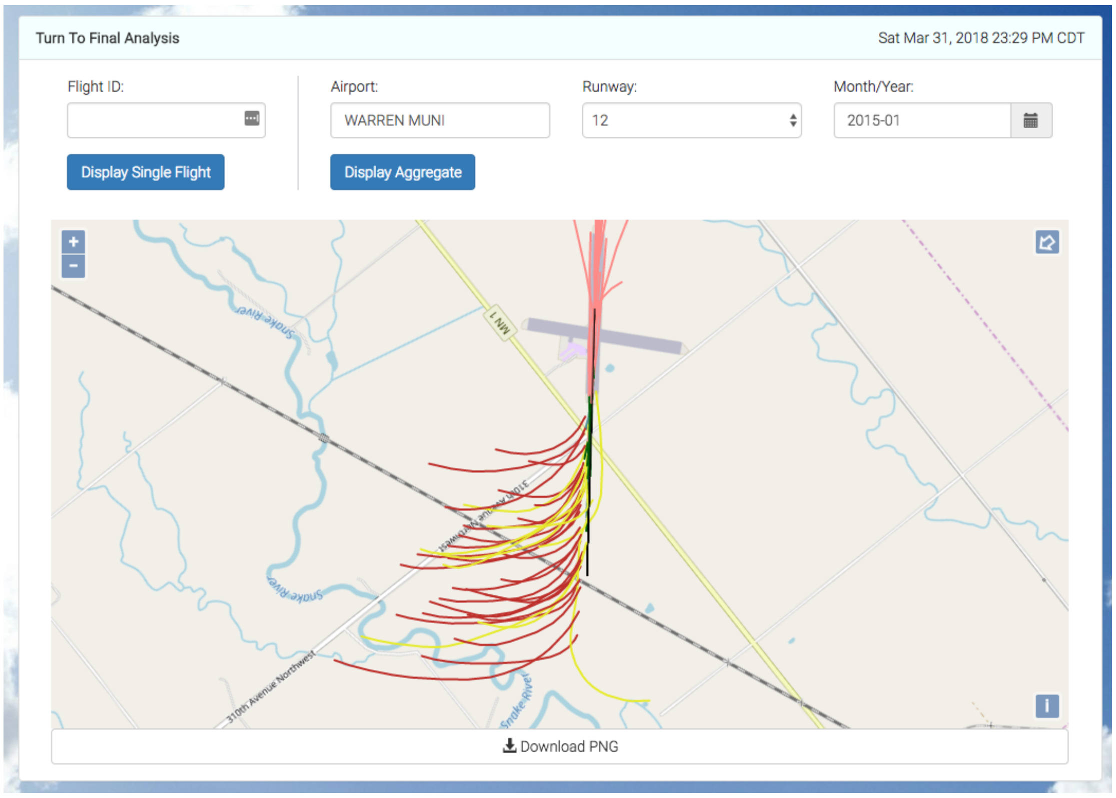
Runway: Select Runway

Month/Year: 2018-03

Figure 18: A screenshot of the Final Turn analysis tool on the NGAFID in “Single Flight” mode. It is currently showing Approach #1 for Flight ID #381001. Approach #1 shown here had a Level 1 (yellow color code) undershoot.

codings for the separate Final Turn, Approach, and Landing phases as well as different colors for the Final Turn specifically depending on the severity of the turn error. A Level 1 turn error will be colored yellow, while a Level 2 turn error will be colored red. If the turn error is less than the Level 1 criteria, it is colored green. The user is also able to download a PNG screenshot of the map by clicking the “Download PNG” button.

For the “Aggregate” mode; the user can choose a specific airport, runway, and month and year combination; which will then display all the approaches that occurred at the chosen runway during the chosen time-frame (Figure 19). This mode allows a user to see trends in Final Turn phases during a given time span. This mode displays the same color code scheme as the “Single Flight” mode.



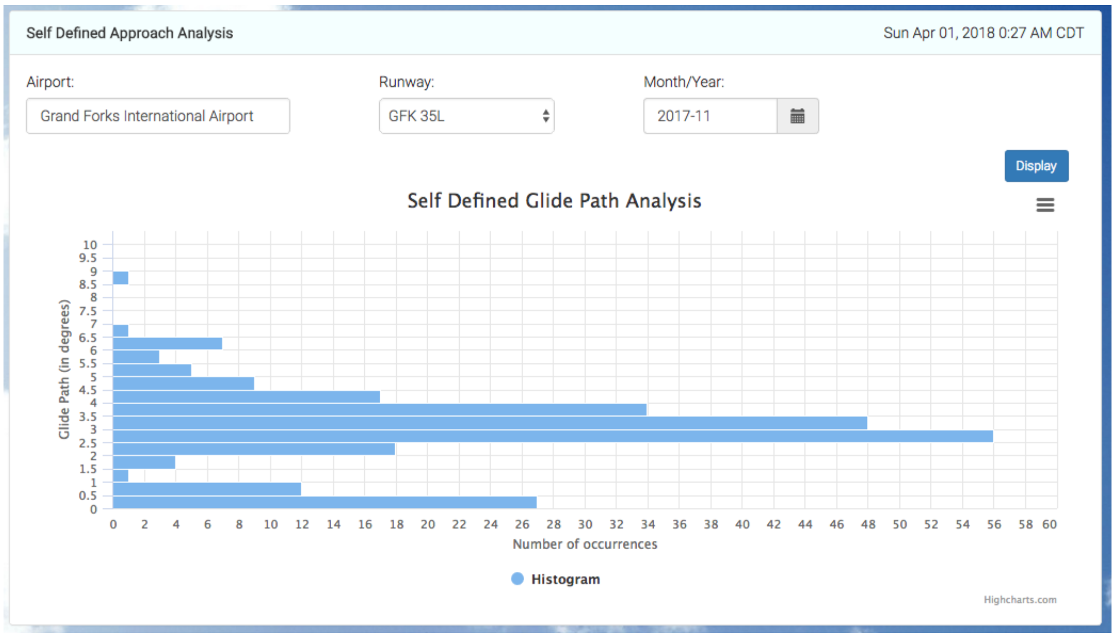
Flight ID:

Airport: WARREN MUNI

Runway: 12

Month/Year: 2015-01

Figure 19: A screenshot of the Final Turn analysis tool on the NGAFID in "Aggregate" mode. It is currently showing all approaches at the Warren Municipal Airport (KD37) for Runway 12 during the month of January 2015. The many red and yellow lines coming in from the left side mean that a majority of the turns were Level 1 & 2 undershoots.


Self Defined Approach Analysis

Sun Apr 01, 2018 0:27 AM CDT

Airport: Grand Forks International Airport

Runway: GFK 35L

Month/Year: 2017-11

Figure 20: A screenshot of the Self-Defined Approach analysis tool on the NGAFID. It is currently showing all approaches at the Grand Forks International Airport (KGFK) for Runway 35L during the month of November 2017. It displays a sideways histogram with glide path angles on the y-axis and the number of occurrences for each angle on the x-axis.

Self-Defined Glide Path

The tool implemented in the NGAFID for displaying the results of the self-defined glide path analysis currently only supports an aggregate mode (Figure 20). It works similarly to the Final Turn tool as the user chooses an airport, runway, and month and year combination. This will then display a sideways histogram of all the approaches at the given runway during the given time-frame. The y-axis shows glide path angles from 0° to 10° in 5° increments, and the x-axis shows the number of occurrences that fell within each angle bin. Lastly, the user can download an image of the displayed chart by clicking the “hamburger” menu button.

Performance

A secondary aspect of this research is to minimize the execution time so the analysis only adds a minimal amount of time to a flight being imported into the NGAFID system. The results of the benchmarking tests showed that the linearly executing application ran for an average of XXX.XXX seconds over the 100 randomly tested flights. On the other hand, the parallel application ran for an average of XX.XXX seconds over the same flights. This means the average per-flight execution times for the linear and parallel applications were X.XXX and X.XXX seconds, respectively. As a result, the parallelized application had a XX.XXX% speedup, which is fairly significant.

Kelton: Fill in timing results.

As further evidence, the parallel application was tested on a larger subset of flights to see if the average execution time remained stable, in which it was tested on 5,272 flights. For this test, the parallel application was able to analyze the data and insert all the results into the database in XXXX.XXX seconds. This gives a per-flight execution time of X.XXX seconds, which is slightly less than the average for 100 flights. The reasoning behind this can most likely be attributed to the fact that spinning up the sub-processes creates a substantial overhead. Thus, the longer the application is able to execute, the greater performance gain will be received. This will, of course, start to show diminishing returns as with any other parallel computing application.

Table 5: Performance of Linear v. Parallel Execution Times

Run	**Linear (sec)**	**Parallel (sec)**
1	591.935	57.295
2	576.774	60.282
3	586.009	57.830
4	597.643	62.489
5	591.170	57.578
6	585.834	62.702
7	593.711	65.064
8	587.177	66.167
9	586.059	58.108
10	590.012	56.501
Average	588.632	60.402
Latency / flight	5.886	0.604
Speedup		90.255%

CHAPTER 6

CONCLUSION

This thesis presented the Go-Around Detection Tool, an application designed to augment the existing features of the National General Aviation Flight Information Database (NGAFID). The purpose of creating the application is to provide student pilots and Certified Flight Instructors (CFI) with metrics-based feedback on flight performance during critical phases of flight. The desired effects of this are *(i)* target different student learning techniques, *(ii)* improve the efficiency and reduce the cost of flight training, and *(iii)* reduce General Aviation (GA) accident and fatality rates since GA is the most dangerous branch of Civil Aviation. Additionally, the application is currently geared towards analyzing the Approach and Landing phases as these phases of flight are where a majority of pilot-related accidents occur.

Using flight data recorder (FDR) data generated by a Garmin G1000 from Cessna C172S aircraft; the application can detect safety exceedances for indicated airspeed, vertical speed, cross track, and heading during the Approach phase as well as classify the result of each approach as a full-stop landing, touch-and-go landing, or a go-around during the Landing phase. For the event-driven approach to successfully characterize the safety of an Approach, the safety exceedance definitions needed to be internally consistent (*i*.e., the parameter limits need to correspond to the same level of risk to the pilot). In this research, the safety exceedances were re-defined in a way that makes them more consistent by an aviation statistics expert who used the distributions of parameter values found during the initial experiments. These new definitions were then used in the newly created grading system for the purpose of scoring the pilot's flight performance based on any exceedances found after the

Approach analysis stage.

Kelton: Talk about Web tools

Future Work

This research has provided many avenues for further work and refinement. First, the greatest constraint on the accuracy of the application is the accuracy of the instrument recording the flight data, whether that be a traditional FDR or a smartphone. This means that if data is recorded inaccurately, it is useless to the application and cannot be recovered. For example, in several of the sample flights, the first 10 to 20 rows of data can have missing and/or spurious values due to the aircraft's sensors calibrating after first starting the FDR. Invalid data rows can occur during the middle of a flight as well; not only when the FDR is initially turned on. Thus, further work into filter, sanitizing, or normalizing faulty data would be very beneficial.

Second, it would be beneficial to make the algorithms more modular in order to analyze data from different sources. For example, a source with a limited number of parameters it records, such as a smartphone, or a completely different brand of FDR.

Third, this research focused solely on the analysis of flight performance and generation of metrics describing the performance, thus further work in the area of UI/UX would be a great next step. This future research would ideally focus on the most effective way to display the metrics to the user and improve upon the user-friendliness of the web interfaces introduced in this work.

Lastly, the algorithms introduced in this research can be extended to analyze other phases of flight (*i.*e., Takeoff, Climb, Cruise, etc.). This means that new risk level values would need to be defined as well to fit the data found in the new analyses.

Once the Go-Around Detection Tool is fully integrated into the NGAFID, it will provide even more possibilities for data visualization and be easily accessible for both novice and experienced pilots. This will allow pilots on an individual or organizational level to become more aware of bad flight habits so they may correct them in future flights and help make General Aviation safer.

REFERENCES

[1] *Cessna 172S Standardization Manual*, UND Aerospace Foundation, Aug 2015.

[2] AOPA, "What is general aviation," 2009.

[3] W. B. Allen, D. L. Blond, A. J. Gellman, G. A. M. Association, N. A. of State Aviation Officials (U.S.), and I. MergeGlobal, "General aviation's contribution to the u.s. economy." General Aviation Manufacturers' Association, May 2006.

[4] Federal Aviation Administration, "The economic impact of civil aviation on the u.s. economy," November 2016.

[5] D. J. Kenny, "26th joseph t. nall report: General aviation accidents in 2014," AOPA Air Safety Institute, 421 Aviation Way, Frederick, MD 21701, Tech. Rep., 2017.

[6] K. I. Shetty and R. J. Hansman, "Current and historical trends in general aviation in the united states," Master's thesis, Massachusetts Institute of Technology, August 2012.

[7] AOPA Air Safety Institute, "2015-2016 ga accident scorecard," AOPA Air Safety Institute, 421 Aviation Way, Frederick, MD 21701, Tech. Rep., 2017.

[8] S. Clachar, J. Higgins, B. Wild, and T. Desell, "Large-scale data analysis for proactive anomaly detection in heterogeneous aircraft data," unpublished.

[9] National General Aviation Flight Information Database, "Welcome to the national general aviation flight information database (ngafid)." [Online]. Available: http://ngafid.org/

[10] MITRE, "Gaard–general aviation airborne recording device." [Online]. Available: https://www.mitre.org/research/technology-transfer/technology-licensing/gaard-general-aviation-airborne-recording-device

[11] S. A. Clachar, "Identifying and analyzing atypical flights using supervised and unsupervised approaches," *Journal of the Transportation Research Board*, 2014, published as part of an ACRP: Graduate Research Award.

[12] S. Clachar, "Novelty detection and cluster analysis in time series data using variational autoencoder feature maps," Ph.D. dissertation, University of North Dakota, December 2016.

[13] T. Desell, S. Clachar, J. Higgins, and B. Wild, *Evolving Deep Recurrent Neural Networks Using Ant Colony Optimization.* Cham: Springer International Publishing, 2015, pp. 86–98.

[14] A. ElSaid, B. Wild, J. Higgins, and T. Desell, "Using lstm recurrent neural networks to predict excess vibration events in aircraft engines," in *The IEEE 12th International Conference on eScience (eScience 2016)*, Baltimore, Maryland, USA, October 2016.

[15] A. ElSaid, "Using long-short-term-memory recurrent neural networks to predict aviation engine vibrations," Master's thesis, University of North Dakota, December 2016.

[16] T. Desell, S. Clachar, J. Higgins, and B. Wild, *Evolving Neural Network Weights for Time-Series Prediction of General Aviation Flight Data.* Cham: Springer International Publishing, 2014, pp. 771–781.

[17] V. Goblet, N. Fala, and K. Marais, "Identifying phases of flight in general aviation operations," in *15th AIAA Aviation Technology, Integration, and Operations Conference*, 2015, p. 2851.

[18] V. P. Goblet, "Phase of flight identification in general aviation operations," Ph.D. dissertation, Purdue University, 2016.

[19] N. Fala and K. Marais, "Detecting safety events during approach in general aviation operations," in *16th AIAA Aviation Technology, Integration, and Operations Conference*, 2016, p. 3914.

[20] E. Wischmeyer, "The myth of the unstable approach," *International Society of Air Safety Investigators*, August 2004.

[21] Z. Nazeri, G. Donohue, and L. Sherry, "Analyzing relationships between aircraft accidents and incidents," in *International Conference on Research in Air Transportation*, Feb 2008.

[22] R. Knighton and C. Claramunt, "An aeronautical temporal gis for post-flight assessment of navigation performance." *Transactions in GIS*, vol. 5, no. 1, p. 53, 2001.

[23] T. Masiulionis and J. Stankūnas, "Review of equipment of flight analysis and development of interactive aeronautical chart using google earth's software," *Transport*, vol. 0, no. 0, pp. 1–9, 2017.

[24] E. Harris Jr., E. Bloedorn, and N. J. Rothleder, "Recent experiences with data mining in aviation safety," in *SIGMOD Record*, Seattle, WA, June 1998.

[25] Federal Aviation Administration, "Runway safety: Runway incursions." [Online]. Available: http://www.faa.gov/airports/runway_safety/news/runway_incursions/

[26] B. Matthews, S. Das, K. Bhaduri, K. Das, R. Martin, and N. Oza, "Discovering anomalous aviation safety events using scalable data mining algorithms," *Journal of Aerospace Information Systems*, vol. 10, no. 10, pp. 467–475, 2013.

[27] C. C. T. Team, "Phase of flight definitions and usage notes," CAST/ICAO Common Taxonomy Team, Tech. Rep., 2013.

[28] R. A. Finkel and J. L. Bentley, “Quad trees a data structure for retrieval on composite keys,” *Acta Informatica*, vol. 4, no. 1, pp. 1–9, Mar 1974. [Online]. Available: https://doi.org/10.1007/BF00288933

[29] *Pilot’s Operating Handbook and FAA Approved Airplane Flight Manual: Cessna Model 172S*, 2nd ed., Cessna Aircraft Company, November 2010.

[30] D. Beazley, “Understanding the python gil,” in *PyCON Python Conference. Atlanta, Georgia*, 2010.

[31] D. Nolan and D. T. Lang, *Keyhole Markup Language.* New York, NY: Springer New York, 2014, pp. 581–618. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-7900-0_17