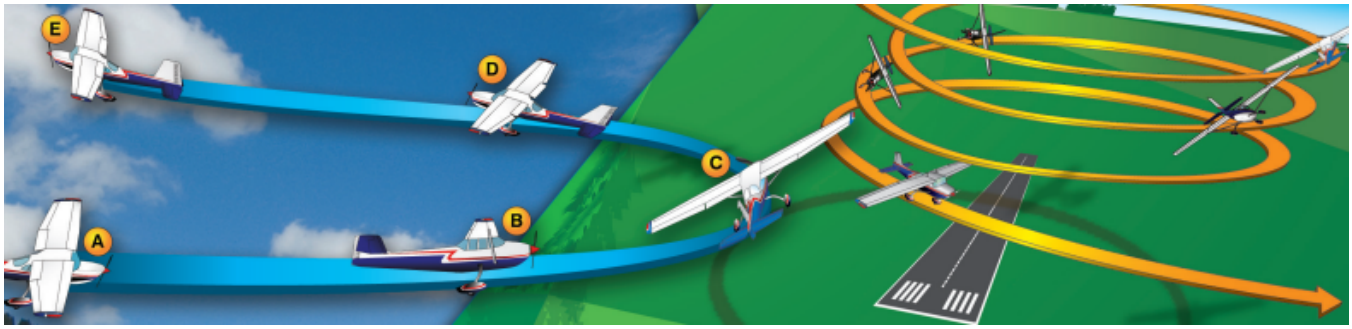
AI-r Force Maneuvers

UC Davis CS Master's Capstone Project

Siena Saltzen
Lucas Moehlenbrock
lwmoehlenbrock@ucdavis.edu
scsaltzen@ucdavis.edu
University of California, Davis
Davis, California, USA



1 Introduction

The U.S. Air Force released a dataset from Pilot Training Next (PTN) through the AI Accelerator of Air Force pilots and trainees flying in virtual reality simulators. In an effort to enable AI coaching and automatic maneuver grading in pilot training, the Air Force seeks to automatically identify and label each maneuver flown in this dataset from a catalog of around 30 maneuvers. This effort to automatically classify flight maneuvers is called the Maneuver Identification (ID) Challenge.

For this project we have taken up the Maneuver ID Challenge and analyzed several different approaches to detecting and labeling the maneuvers. The data provided has many dimensions with continuous classification problems and this project will aim to identify subsets of flight data with intervals along which a maneuver is performed and provide a label for the maneuver. Additionally, in doing so, we hope to also lay the foundations for extending our work to provide automatic labeling for self supervised algorithms and eventually real time identification.

In this report, we detail the different approaches that we applied to the maneuver identification problem.

1.1 Motivation

AI algorithms that identify maneuvers from trajectory data could play an important role in improving flight safety and pilot training. The Maneuver ID Challenge has the potential to enhance and personalize simulator training by providing mid-flight and post-flight feedback to students. As there continues to be a large pilot shortage, improving technologies to individualize and automate some aspects of training present an opportunity to alleviate the burden of training by reducing the load on the limited resource of instructor time. Further refinement and exploration of real time maneuver identification may be useful in safety and warning systems for private aircraft. Additionally this challenge provides a difficult and unique problem space that is a large scale unlabeled or sanitized data set that is dependant across time.

2 The Data

Source: Motivation | https://maneuver-id.mit.edu
Data: Data | https://maneuver-id.mit.edu/data
Info: 2.89GB/6,661 files of .tsv files containing aircraft position, rotation, and velocity data over various intervals

2.1 Attributes

The data consists of nearly 3GB of tsv files with each row of a file containing a timestamp, the xyz positional data, xyz directional velocity, and pitch, roll and yaw. Pitch, roll and yaw describe the axes of rotation of the aircraft during flight. Pitch is the rotation of the aircraft about the transverse axis, or the angle of the nose, and mostly governs the rate and change of altitude. Roll is the rotation about the longitudinal axis, across the wings. Yaw is the rotation about the vertical axis controlled by the rudder, which directly affects the heading. In the data this seems to be referred to directly as heading.

Each file contains a time-dependent sequence of rows containing the above-mentioned data. Each index is separated by approximately 0.2 seconds, so the number of indices in a file directly correlates to the total duration of the flight.

Throughout this report we will use the number of indices when discussing the length of a file.

Maneuvers are independent of the location that they are performed in the x-y plane, so we do not use the x and y positional data. There are some maneuvers that have specific altitude requirements, so they y position of the plane would be useful for the classifier, however the dataset is normalized such that the lowest y value in the file is 1. This removes any information that would be helpful for determining a maneuver from it's specific altitude requirements. Maneuvers are also independent of their xy velocity so we exclude the x and y-velocity data as well. This leaves us with the features that we have chosen to use for our classification approach: z-velocity, and the pitch, roll, and heading

There are considerable inconsistencies within the data from time skips, repetitions, and impossible aircraft positioning present in some of the files. Additionally the flight logs from the data set are incredibly imbalanced. The amount of "turn" maneuvers greatly outnumbers any other type of maneuver and the representation of some of the more complicated maneuvers was such they did not not appear during manual inspection. Unfortunately due to the unlabeled nature of the data we were not able to mitigate this with either under or over sampling.

2.2 Cleaning the Data

A large portion of our work on this project was dedicated to sanitizing and managing the data set. Without provided labeling and with a large amount of inconsistencies present in the data, it was very difficult to determine what was good and usable. We also are working around suspicions that segments of the flight files were removed to hide information about the military aircraft's acceleration.

Additional issues present in the data set include:

- Duplicate Files
- Files of Static Airplane
- Inconsistencies with Time Stamps
- Duplicate lines of data in a file
- Missing data points/cells within a file

Once we identified the flaws in the data, we created a script that would remove files containing errors or duplication. This can be run on any copy of the data to create a clean set of log files.

2.3 Visualization and Labeling

In order to address the issues in the data and understand the format, we created a flight path visualiser that "flies" a model aircraft along the flight paths contained in each file. This also allows for manual verification of any results or inconsistencies. We also found that the length, number, and types of maneuvers in a file varies, and can range from one single maneuver to many chained or repeated maneuvers. There are also many files that consist entirely of steady flight and do not contain any maneuvers at all.

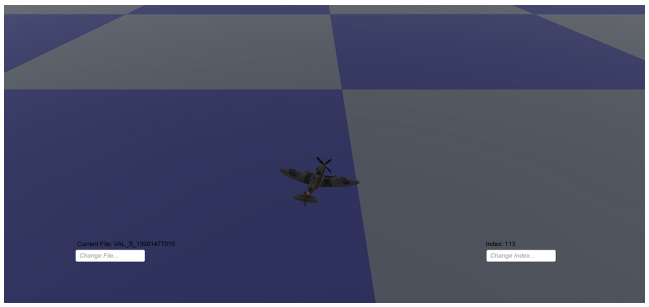


Figure 1. Screenshot of the Flight Path Visualizer in action

We developed a visualizer in Unity, which proved to be an extremely useful tool for the project. The visualizer takes in a file from the dataset as input and allows the user to control the flight path of the aircraft by increasing or decreasing the current index and mapping the plane to the positional and rotational data at that index.

The visualizer also has a user interface that allows the user to input a file name from the dataset to load that file for visualization. There is also an interface for jumping to a specific index along the flight path, which is useful for validating the results of the different classification approaches we applied.

Using this tool we were able to manually mark the transitions from steady flight to "maneuvers" and synthesize a basic formula that we used to find threshold values of the change in pitch/roll/heading that differentiate a "maneuver" from steady level flight based on the change in pitch/roll/yaw. From here we collected a subset of hand labeled data to be used as the training data for our classification algorithm.

Additional early attempts at understanding the data included graphing out the trajectories on a grid. While helpful, this unfortunately did not convey enough information on the actual 3D orientation or the speed to the human reviewer, and thus was forgone in favor of the moving model.

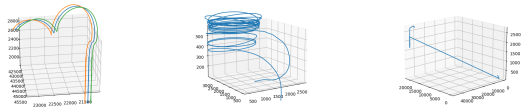


Figure 2. Examples of previous attempts to visualize flight paths with a 3D plot

3 Background and Related Work

3.1 Maneuvers and Terminology

There are 29 valid maneuvers present in the data set: *Steep Turn, Aileron Roll, Barrel Roll, Cuban 8, Immelman, Intentional*

Spin, Landing Attitude TP Stall, Lazy 8, Loop, Overshooting TP Stall, Split S, Undershooting TP Stall, ILS, Localizer, Nose High Recovery, Nose Low Recovery, Straight In Pattern, Closed Pull Up, ELP-PEL, ELP-SEFL, Overhead Pattern, Slow Flight, Unusual Alt Nose High, Power On Stall, Power On Stall Nose High, Unusual Alt Nose Low, Verticle Salpha, Verticle Sbravo.

Many of these maneuvers are very similar to each other or based on chained actions from smaller maneuvers. For example the only difference between an aileron roll and a barrel roll is that an aileron roll's centre of rotation is very close to or on the aircraft. A barrel roll has its centre of rotation around a point further away from the aircraft itself. How much further away depends on the quality of the roll. As this is training data from inexperienced pilots, in many cases, the two maneuvers can be difficult to distinguish.

3.2 Approaches for Classifying Time Series Data

We started analyzing the problem from the same point of view as [12] and [2]. Unfortunately their approaches relied on expert annotation in the case of Wang's work and access to more features such as altitude, throttle inputs, roll rate, pedal controls, etc. As such we were unable to follow their more sophisticated methods and were instead reliant on a more generic and limited set of algorithms that can work with time dependent data across very few features. It should be noted that while the dataset we are using does have an altitude feature in the form of the position along the z-axis, all of the files are normalized such that the lowest z value is 1 which excludes any possibility of using the altitude to assist in identifying maneuvers.

Our initial approaches were to use modified clustering algorithms that can handle a time series. We investigated the implementation of HMMs and Covariance-Based Clustering. Our final solution does not involve clustering, but instead applies Dynamic Time Warping with a graph search approach to identify maneuvers using a training dataset that we constructed.

4 Project Overview

For the full project, we have created a working prototype of an automatic flight maneuver identifier that runs in real time on a simulator. While the plugin and the classifier are simultaneously executed, any flight log that begins triggers automatic classification of maneuvers performed in between steady flight. This is then written to a file and displayed back on the simulator for the pilot to view. While on the surface it performs a rather unhelpful task, this is hopefully a baseline for a tool that can help identity and grade quality of new pilots while they practice maneuvers without instructor supervision. Repeated practice with distinct feedback on adherence to maneuver standards will be an invaluable tool to let pilots train on their own while mitigating reinforcement of bad behaviors they may not be aware of.

4.1 Tools and Resources

4.1.1 XPlane and The Plugin X-Plane is a flight simulation engine series developed and published by Laminar Research in 1995 that is pre-packaged with several commercial and military aircraft, as well as global scenery which covers most of the Earth. X-Plane also has a plugin architecture that allows users to create their own modules, extending its customization and behaviors. The professional version allows FAA certified flight training hours to be logged, as long as the computer system running X-Plane is tested to meet minimum frame-rate requirements and all the associated hardware has been approved. This makes it the perfect platform to interface with our classifier and begin creating a baseline of a real working tool to help train pilots.

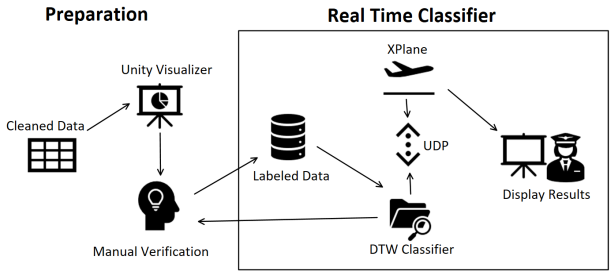
XPlane runs a highly detailed simulation with over 4,000 readable variables controlling the state of the airplane. These are available to access through a plugin as datarefs. Plugins are DLLs that are strong-linked inside the XPlane Plugin Manager. Because of the non permanent nature of the plugin's memory, and the plugin running synchronously inside the X-Plane process, we had to reshape the flow of data from the original plan, which included the plugin monitoring the intervals of a flight maneuver into handling this in the classifier program. Since the plugin and the simulator will not be running at the same time, If the plugin is slow, the simulator's frame rate will slow down too. This was a factor in considering how often to send UDP packets containing aircraft information. After running several tests, the optimal time for reliable packet receiving without hang ups ended up being 1 packet per flight loop. The Plugin also has two callback functions, one which is executed every flight loop for the UDP, and one which uses the XPLMDisplay to create the window reporting the values and results. The final plugin is not an overly complex piece of code, however setting up the environment, linking the necessary libraries and matching versions of the SDK and APIs were very time consuming.

4.1.2 PFC We have been lucky enough to have support from the flight simulator company Precision Flight Controls, in the form of correspondence with a developer on their custom builds based around XPlane. They have also graciously allowed us to run a full demo of our project on their motion enhanced flight sim platform, and provided a pilot who will test several more complicated maneuvers. We will be formally testing the project with expert oversight in the week of May 23rd, and will add a demo to the repository as soon as possible.

4.2 The Pipeline

The flow of our work started in the maneuver classification. We began with manually identifying a set of base maneuvers through identification using the unity visualizer and built up a small set of labeled maneuvers. This becomes the basis

for our work on the classification algorithm, which cross examines sections of flight time against the labeled data. Next XPlane is launched running the plugin. This begins streaming UDP packets containing the timestamp, Z-Velocity, Pitch, Roll and Yaw as soon as the user starts a new flight. Once the classifier is run, it listens for the UDP packets and applies our threshold for distinguishing steady flight in the change in parameters across a certain time stamp. Once the threshold has been passed, the airplane states are recorded until the change drops again below the steady flight limit. This continuous flight record is then passed through the classifier, which writes its results to a file to be displayed back into XPlane. This is a continuous operation until either process is terminated.



4.3 Installation and Code

All material relating to this project can be found on github at https://github.com/ssaltzen/MastersProject. A link to download the executable version of the visualizer will be provided separately.

To run the entirely of this project, you will need to download the XPlane demo (https://www.x-plane.com/), our plugin (GitHub), the classifier (GitHub), and the labeled dataset (link provided separately).

1. Set up the python environment for the classifier
2. Make sure the paths in the classifier are correct for the dataset location
3. Make sure that port 8888 is not in use, if so, you will have to recomplile the plugin or clear the port
4. Copy the plugin .xpl into the X-Plane 11-Resources-plugins
5. Run XPlane and Classifier program
6. Start Flying

5 Identification Algorithms

5.1 Clustering

The challenge of identifying and labeling maneuvers from the positional and rotational data of a flight path can be broken up into two major aspects. The first is finding the subsequences of the flight path that are maneuvers and the second is classifying those subsequences with the appropriate maneuver label.

The first approach we used to find the maneuver subsequences of the flight path was clustering. The clustering method that we used is Toeplitz Inverse Covariance-Based Clustering (TICC) [6]. This clustering approach first divides the time-series data into windows and then assigns each window to a cluster. Each cluster is defined by a Gaussian inverse covariance matrix. The TICC algorithm has a regularization parameter for adjusting the sparsity of the covariance matrix and another parameter for increasing the likelihood that adjacent windows will get assigned to the same cluster.

We clustered several files, adjusting the number of features from the dataset that were used. We found that using the three rotational features (yaw, pitch, and roll) and the z velocity to perform the best. After determining the optimal features to use, we ran the TICC algorithm on 25 random files from the dataset. The TICC algorithm requires the number of clusters to be determined before performing clustering and since we don't know how many clusters there are in each file, we ran the TICC algorithm on each file multiple times across a range of clusters.

The TICC algorithm performs well for grouping together sequences of flight with similiar behavior, however the downside is that the algorithm performs poorly when it comes to finding the boundaries between maneuvers. For example, a flight sequence with a turn followed by a period of steady flight followed by another turn might get assigned to the same cluster, where ideally the turns would get assigned to the same cluster and the steady flight would get assigned to a different cluster. Another example is that two maneuvers in a row, such as a turn into an aileron roll, may get added to the same cluster.

Another issue with the TICC approach is that the cluster assignments across different files are meaningless. For example, cluster three for a certain file may contain mostly turns whereas cluster three in another file may contain steady flight. The TICC algorithm outputs the inverse covariance matrix for each cluster. We explored the idea of using these matrices to measure the similarity between clusters in different files to try to group similar clusters from different files into the same cluster, however taking into consideration the difficulty the TICC algorithm has in finding the boundaries between maneuvers, we decided to take a different approach to maneuver identification.

6 Shortest Path Classification

Shortest Path Classification is the algorithm that we developed to classify maneuvers from the flight path time series data. This algorithm uses Dynamic Time Warping to determine the similarity between maneuvers. Dynamic Time Warping is a popular approach to time series classification, often combined with K-Nearest Neighbors approaches [11]. In

our investigation of different approaches to maneuver identification and classification, we found dynamic time warping (DTW) to be an extremely effective measure for comparing the similarity of two flight sequences. DTW finds the optimal index pairing between two sequences that gives the minimal cost, where the cost is the sum of the differences of each index pair. This technique allows for regions with similar behavior to be matched even if they are shifted over or of different lengths [7].

We manually labeled examples of each maneuver of interest in the dataset by analyzing flight paths in the Unity visualizer and looking for examples of each maneuvers. When one was found, the start and end index of the maneuver were recorded and the subsequence of the flight path that contained the maneuver was saved as a separate file. These manually labeled examples are referred to as the training data for the rest of this section.

For our solution to the maneuver classification problem, we take as input a file from the flight path dataset and first we perform the following for each file in the training data:

Algorithm 1 DTW Comparisons

```
comparison_list = []
label = training_file.label
for i = 0 to len(input_file) − len(training_file):
    start = i
    end = i + len(training_file)
    input_section = input_file[start : end]
    distance = DTW(input_section, training_file)
    comparison = (distance, label, start, end)
    comparison_list.append(comparison)
```

This algorithm effectively slides each of the training files along the input file and calculates the DTW distance at each index to compare the training file to every possible section of the input file. Once this comparison process has been completed for all training files, we construct a graph of the comparisons using the following criteria:

- create a "start" node and "end" node for the graph.
- Add an edge between the start node and every comparison that has a start index of 0, the edge weight is equal to the DTW distance for the comparison.
- Add an edge between the end node and every comparison that has an end index equal to the length of the input file, the edge weight to the end node for each comparison is the same and is chosen arbitrarily as these comparisons mark the end of the file.
- Add an edge from $comparison_a$ to $comparison_b$ if the start index of $comparison_b$ is equal to the end index of $comparison_a$, the edge weight is equal to the DTW distance for $comparison_b$.

After this process is complete, each path from the start node to the end node of the graph represents a set of comparisons that span the full length of the input file with no overlap between comparisons. Therefore, every path through the graph is a set of labels, and the distance for each label indicates how similar that label was to the section of the file that was used for the comparison. The shortest path through the graph is then the set of labels that most closely matches the input file. We use Dijkstra's algorithm to find the shortest path between the start node and end node. For each comparison along the shortest path, the label of the comparison becomes the predicted label for the region of the input file matching the start and end index of that comparison.

We use four features for this classification: z velocity, Δheading, pitch, and roll, where Δheading is calculated at each index by taking $heading[i] - heading[i-1]$. Since we are using four features, we calculate the DTW distance separately for each feature and add the distances together to get the total DTW distance.

The reason that the change in heading is used rather than the real value of the heading is because the heading represents the plane's rotation in the horizontal plane, so identical maneuvers performed while traveling in a different direction (e.g. east vs west) would have a large difference in the heading value which would affect the accuracy of the predicted labels.

For the training data, it is important to have labeled examples of varying lengths for each maneuver of interest. The algorithm compares each training file to a section of the input file that is the same length as the training file, so if all examples of a specific maneuver were the same length then any occurrences of that maneuver in the input file that are of a different length may either be incorrectly classified or classified with inaccurate start and stop indices. The training data is constructed with this in mind; for each maneuver of interest, the labeled examples are chosen so that there is a large variation in the lengths.

It is possible that there may be maneuvers within the dataset that have lengths outside of the range of lengths for the corresponding maneuvers in the training data. To account for this, at each index we perform multiple comparisons by changing the bounds of the region of the input file that is used in the DTW calculation. We incrementally increase the length of this region by 2, decreasing the start index by 1 and increasing the end index by 1. This process is repeated 5 times at each index, and then the same thing is done to decrease the length of the region, however decreasing the length is only performed 4 times. Therefore, at each index, the training file is compared to a range of lengths in $[length(trainingfile) - 8, length(trainingfile) + 20]$ rather than just a single comparison the same length as the training file.

For the DTW calculations, we use the fastdtw library in python which approximates the optimal DTW distance in

$O(N)$ time. A DTW algorithm that calculates the exact optimal DTW distance would have a time complexity of $O(N^2)$, so the potential loss of accuracy due to using an approximate DTW distance is acceptable. The time complexity of the DTW calculation part of the Shortest Path Classification algorithm is

$$N \sum_{i=1}^{L} \ell_i$$

where N represents the length of the input file, L represents the number of labeled maneuvers that are in the training data, and ℓ_i is the length of the ith labeled maneuver. This summation can be simplified to $NL\bar{\ell}$, where $\bar{\ell}$ is the average length of the training data files.

This algorithm is rather computationally intensive so we applied some optimizations. The first optimization is to preprocess files, looking at the change in heading, pitch, and roll, and removing any regions of the file where the value falls below a threshold. This technique removes periods of steady flight which are not of interest for classification and often make up the majority of files. The regions of the input file that remain are potential maneuvers or series of maneuvers. We then run the Shortest Path Classification algorithm on each of these regions. Additionally, we implement the python multiprocessing library to perform the DTW calculations for each training file in parallel in order to improve the classification time.

Another algorithmic improvement came from the graph construction. When the comparisons are made between a training data file and the input file, the result is a list containing a tuple with the DTW distance, label of the training file, and start and end index of the region of the input file that was compared to the training file. The process of computing the comparisons for each training file in parallel results in a list of lists, with the nested lists containing all tuples for the respective training file.

We initially used a naive approach to construct the graph, first putting all comparison tuples into a single list, then using nested loops to check each comparison tuple against every other comparison tuple to see if they should share an edge. The number of vertices in the graph is roughly $N * L$ so this naive approach had a time complexity of $O(N^2L^2)$.

To improve this time complexity, we perform the same first step as the naive approach. That is, to put all comparison tuples into a single list. Then we sort the list by comparison start index using the python list.sort() method, which has a time complexity of $O(NL \log NL)$.

With the single sorted list of all comparison tuples, the process to construct the graph is to loop through each $comparison_i$ in the list and perform a binary search of the list to find all comparisons with a start index matching the end index of $comparison_i$. The time complexity of this search is $O(NL \log NL)$. Therefore, the overall time complexity of constructing the graph is $O(NL \log NL)$, which is a large improvement from the naive implementation with $O(N^2L^2)$.

The final part of the algorithm to analyze is the Dijkstra's algorithm shortest path search. The time complexity of Dijkstra's algorithm is $O(E \log V)$ [3]. As stated above, the number of vertices in the graph is about $N * L$. The number of edges from each vertex is equal to the number of comparisons with a start index equal to the end index of the vertex. Therefore, the number of edges connected to each vertex is proportional to L since the DTW calculation is performed at each index for each of the L training files. The total number of edges in the graph is then $N * L^2$ and the time complexity of the shortest path search part of the algorithm is $NL^2 \log NL$.

The time complexity of the shortest path search clearly dominates the time complexity of the graph construction. Comparing the time complexity of the shortest path search to the DTW calculation part of the algorithm is a little more difficult. The DTW calculations depend on the average length of the training files, $\bar{\ell}$. If $\bar{\ell} > L \log NL$, then the DTW calculations time complexity dominates that of the shortest path search.

Although L is a constant for the purpose of this project (there are 108 training files), it is still interesting to analyze how the Shortest Path Classification algorithm would scale with the size of the training data, as increasing the size of the training set would improve the classification accuracy. When letting L be a constant, the time complexity is $N \log N$. We empirically verified this runtime by timing each portion of the algorithm, the results of which can be seen in Figure 3.

6.1 Labeled Training Data

The maneuvers of interest that we were able to find in the dataset include: Loop, Immelman, Split-s, Aileron Roll, Barrel Roll, Lazy-8, and Cloverleaf. A link to videos containing an example of each type of maneuver can be found in the readme of the Github repository for this project. Steady flight is also included in the labeled training files, as the pre-processing can still leave in regions of mostly steady flight where the plane rolls slightly. We also classify turns, however the labeled training data for turns is different from the other maneuvers.

For the maneuvers mentioned above, there is a small variation in how the maneuver can be performed. For example, a loop may vary slightly in duration from the loops in the training data, but this is manageable by including loops of varying durations within the training data to account for this. There is much more variation in how turns can be performed; the radius of the turn can vary quite a bit from sharp turns to shallow turns that are almost indiscernible from steady flight. Turns also vary greatly in length, as a turn can be sustained indefinitely which isn't true for any of the other maneuvers. Instead of including full turns in the training

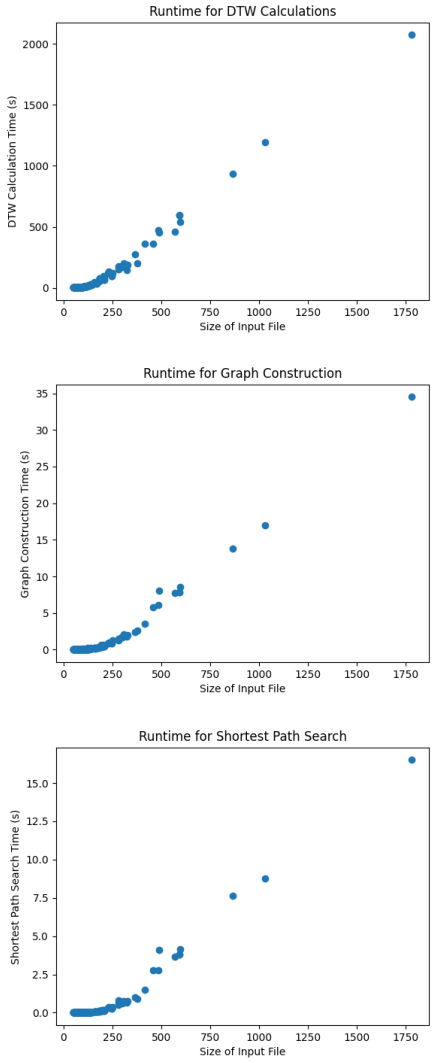


Figure 3. Empirical values for the runtime of the shortest path classification algorithm

data we include small parts of turns. The labels for these turn components also specify whether each turn component is turning clockwise or counterclockwise. This enables the algorithm to properly identify the boundary between back-to-back turns, which would otherwise be merged into a large single turn if the components weren't separated by turn direction.

There is some post-processing done on the list of predicted labels returned by the Shortest Path Classification algorithm. First, any labels of Steady Flight are removed as we are only interested in the parts of the file that are not Steady Flight. Next, any sequences of turn component labels with the same turn direction are merged into a single predicted turn. Any other predicted labels are in the set of maneuvers of interest listed above and are left as-is. The end result is a list of predicted labels with start and end index of the predicted maneuver.

The labeled training data for Cloverleafs are also slightly different from the other maneuvers. A Cloverleaf consists of 4 identical maneuvers back-to-back. These maneuvers can also be performed individually, so we include Quarter Cloverleafs in the training data rather than full Cloverleafs.

6.2 Results

We ran the Shortest Path Classification algorithm on 256 random files from the dataset and validated the output of 50 of these files. We prioritized validating files containing multiple maneuvers over files containing only turns, as the majority of files in the dataset only contain turns. The validated files contained a total of 481 maneuvers. 429 of these maneuvers were turns and the remaining 52 were the maneuvers of interest. Table 1 shows the results with how many of each maneuver were correctly labeled and how many were incorrectly labeled by the Shortest Path Classification algorithm.

Table 1 - Shortest Path Classification Results

Predicted Label	Correctly Labeled	Incorrectly Labeled
Turn	401	28
Loop	5	0
Barrel Roll	1	2
Aileron Roll	14	3
Immelman	6	2
Lazy-8	9	3
Quarter Cloverleaf	3	0
Split-S	3	1
Total	442	39

Out of a total of 481 predicted labels, 442 were correctly labeled. This gives us an accuracy of 91.8%. Out of the 429 predicted turns, 401 were correctly labeled giving a turn prediction accuracy of 93.5%. Out of the 52 non-turn maneuvers, 41 were correctly labeled giving a non-turn prediction accuracy of 78.8%. The accuracy for non-turn maneuvers is quite a bit lower than the accuracy for turns, however the sample size is much smaller. Some possible improvements would be to increase the size of the training data. Another possibility to explore in future work is breaking each maneuver down into components in the training data, as was done with turns.

7 Additional Efforts

7.1 Hidden Markov Model

We briefly explored the possibility of implementing a HMM model as it would greatly aid our long term goal of online predictions. Without label for our data set we started working on the third general problem solvable with Markov models: "Given just the observed data, estimate the model parameters." Unfortunately we ran into some issues as it is unclear

what constitutes a "state" as the definitions of a maneuver are not well defined as there is a lot of room for variation. Additionally the number of observation symbols would be too large if we consider each observation to be a row in the file, as they are continuous values across several features. The the large space of possibilities is very difficult to pair down and we were unsure how to break the data into meaningful instances to be of use in an HMM. Potentially with more domain knowledge, or understanding of HMMs, we could identify where a sequence of flight time is relevant to be carved out. We intend to continue to pursue the HMM as the next model in our work as continuous predictions from the given sequences would be extremely helpful in solving our real time classification goals.

7.2 Shared Memory

The original plan for passing data between the simulator and the classifier was to create a shared memory with mutex locking between the processes. While we were able to set up a working memory space from the plugin, we decided to switch to UDP communication to allow for the classifier and XPlane to run on separate computers, to limit the amount of time we would have to spend creating python executable or environment and worrying about path placements. This also allows us to interface with any running setup of XPlane regardless of the hardware, with minimal changes to the host computer.

8 Conclusion

Given the complexities of working with an unlabeled multivariate time series data set, we are pleased with the results we have achieved. We believe we have developed a novel approach to time series classification with the Shortest Path Classification algorithm, although it is similar to a Nearest Neighbors approach. With the effort we devoted to understanding the domain specifics of the data and our work managing the messy data set we are confident that we have made significant progress in solving the challenge. We believe that our methods are sufficient to identify the most represented simple maneuvers.

The variety of tools that we implanted created a unique set of challenges from working with real world applications. Much effort was spent dealing with version issues, library dependencies, and data sharing/transformation. We found several novel workarounds and tested the limits of our patience. We also learned a great deal about dealing with real data sets, time dependant sequences and unsupervised learning. This project provided an opportunity for us to full explore the limits of our skills and better our understanding of the development process.

We will need a great deal more processing power and/or time to fully apply our solution to the entirety of the maneuver space, but we feel confident with the performance of the current model. Much of our time was also spent research other practical options and gathering more in-depth knowledge on the particulars of aviation exercises. With this understanding we are excited to continue our research and expand the scope of our work.

9 Future Work

While the breadth of this project was limited, there is a very real need for automatic detection of aircraft maneuvers both for training and eventually safety or auto piloting capabilities. The goal of this work was to lay a foundation for expanding into maneuver quality grading, better data-set creation and more accurate validation. With our integration into the XPlane environment, we have a demonstrable classifier that is capable of distinguishing between common maneuvers with only a slight delay during analysis. This confirms that further work in refining the system could be applied to real training situations.

There are many areas for continued improvement that would be large but worthwhile undertakings. Gathering more quality and feature rich data would be a priority, as many intricacies among the maneuvers are lost with the exclusion of important aircraft variables. Previous work in automatic aircraft maneuver identification has shown that there are additional important features such as the normal load factor is extremely valuable for maneuver identification. A data-set containing the normal load factor along with altitude, throttle inputs, roll rate, pedal controls and acceleration would help propel our work from a novelty into a reliable tool. Additionally we are interested in exploring the space of realistic unlabeled data sets dependant over time as there are not many available or obvious solutions. Adding other classifiers or verification algorithms would only serve to bolster our results and practicality of the program.

It would also be beneficial to extend some research into a user study to determine if flight instructors would appreciate and trust the use of this tool to facilitate training and how they would interact with it. Some research into FAA complacency and standards could guide the next steps into practical application.

As for the Maneuver Identification Challenge, we hope that our work will provide a valuable contribution to the effort. While we diverged slightly from the original expectations of the challenges, we hope that we have preserved the ultimate motivation in pioneering a tool that will eventually serve as a valuable assistance in a new pilots journey.

10 Contributions

Lucas Moehlenbrock: Labeling files, Visualizer, Classification algorithm.
Siena Saltzen: Data Cleaning, Project Design, XPlane Plugin, Domain Knowledge and Documentation

The idea, set up and additional work for this project was developed between both authors.

References

[1] Mobyen Uddin Ahmed and Shahina Begum. 2020. Convolutional Neural Network for Driving Maneuver Identification Based on Inertial Measurement Unit (IMU) and Global Positioning System (GPS). *Frontiers in Sustainable Cities* 2 (2020). https://doi.org/10.3389/frsc.2020.00034

[2] Camilla Bodin. [n.d.]. *Automatic Flight Maneuver Identification Using Machine Learning Methods.* arXiv:http://liu.diva-portal.org/smash/get/diva2:1432871/FULLTEXT01.pdf http://liu.diva-portal.org/smash/get/diva2:1432871/FULLTEXT01.pdf

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[4] Curtis E. Ewbank, Randall J. Mumaw, and Michael P. Snow. 2016. Development of the enhanced bank angle warning. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. 1–9. https://doi.org/10.1109/DASC.2016.7778061

[5] FAA. [n.d.]. Performance Maneuvers. https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/airplane_handbook/media/11_afh_ch9.pdf.

[6] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. 2017. Toeplitz Inverse Covariance-Based Clustering of Multivariate Time Series Data. *KDD : proceedings. International Conference on Knowledge Discovery Data Mining* 2017, 215–223. https://doi.org/10.1145/3097983.3098060

[7] Rohit Kate. 2015. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery* 30 (05 2015). https://doi.org/10.1007/s10618-015-0418-x

[8] Tim Oates, Laura Firoiu, and Paul Cohen. 1999. Clustering Time Series with Hidden Markov Models and Dynamic Time Warping. (06 1999).

[9] Kimberlee H. Shish, John Kaneshige, Diana M. Acosta, Stefan Schuet, Thomas Lombaerts, Lynne Martin, and Avinash N. Madavan. [n.d.]. *Trajectory Prediction and Alerting for Aircraft Mode and Energy State Awareness.* https://doi.org/10.2514/6.2015-1113 arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2015-1113

[10] Peer Ulbig, David Müller, Christoph Torens, Carlos C. Insaurralde, Timo Stripf, and Umut Durak. [n.d.]. Flight Simulator-Based Verification for Model-Based Avionics Applications on Multi-Core Targets. https://doi.org/10.2514/6.2019-1976 arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2019-1976

[11] Jinghui Wang and Yuanchao Zhao. 2021. Time Series K-Nearest Neighbors Classifier Based on Fast Dynamic Time Warping. In *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. 751–754. https://doi.org/10.1109/ICAICA52286.2021.9497898

[12] Yongjun Wang, Jiang Dong, Xiaodong Liu, and Lixin Zhang. 2015. Identification and standardization of maneuvers based upon operational flight data. *Chinese Journal of Aeronautics* 28, 1 (2015), 133–140. https://doi.org/10.1016/j.cja.2014.12.026