Real-Time G-Force Monitoring System for Prepar3D

C S 448 Senior Project
Final Report

Karina Gonzales
Candy Arce
Mauricio Muñoz
Jason Peña
Nate Le

Table of Contents

Introduction

Prepar3D is a highly advanced flight simulation software widely used in the aviation industry for aircraft training and development. It plays a crucial role in modeling aircraft flight, offering significant cost and time savings for training and design processes. Despite its robust features, there's an opportunity to enhance Prepar3D by improving how critical in-flight data, such as G-force values, are monitored and displayed. This enhancement can provide better situational awareness and safety for pilots in training. Precision Skies addressed the need for advanced flight metric visualization, but it didn't focus on G-force indicators that could enhance situational awareness during real-time simulations. Our project will contribute to Prepar3D by adding new features to the simulator's dashboard, specifically a G-Force Monitoring and Visualization System that uses visual indicators to alert pilots when G-force levels exceed safe thresholds.

Needs and Requirements

The user will be able to track their speed, altitude, magnetic heading, longitude, latitude, and pitch the same way they could with the previous team's work, once they connect an instance of the Prepar3D simulator to the application. We will extend the interface to include the tracking of G-Forces as Steven recommended along with the option to start the flight assessment. We know that the current code base offers these capabilities based on the provided code that Steven gave us to work with.

Functional Requirements

The overall functional requirements mainly include retaining the current features that the previous team implemented, while extending the GUI with G-Force tracking. The user will retain the ability to connect or disconnect the application from whatever instance of Prepar3D they are using to send aircraft data. The user will be able to record a flight and complete a list of goals to analyze their skill. The recorded flight will be stored with the important information on their performance and used for reflection of growth.

Non-Functional Requirements

The non-functional requirements that need to be met mainly include maintenance and refactoring of the code base of the previous team. We will need to work in C# using Visual Studio 2017 or newer using the .NET 8.0 framework using the Nuget package manager to restore and maintain dependencies. In order to connect to the Prepar3D simulator we need to use the Prepar3D SDK and SimConnect SDK to obtain aircraft data to process and display. Besides the

SDKs we need to communicate with Prepar3D, we will also need to update and utilize the following dependencies that the code base uses:

- HarfBuzzSharp - cross-platform OpenType text shaping engine.
- LiveCharts - data visualization library.
- Newtonsoft.Json - JSON framework.
- OpenTK - low-level C# bindings for OpenGL.
- SkiaSharp - cross-platform 2D graphics API.
- System.Buffers - resource pooling library.
- System.Drawing.Common - .NET Core and .NET 6+ GDI+ interoperability layer library.
- System.Memory - types for resource pooling.
- System.Numerics.Vectors - hardware accelerated numeric types library.
- System.Runtime.CompilerServices.Unsafe - low-level API for manipulating pointers.

All of these library dependencies were obtained through the “packages.config” file that the previous team utilized to configure NuGet. Thus, we will be using Nuget to manage these library dependencies.

Expected Area of Use

Our project is expected to be utilized in the aviation industry. Specifically with flight simulator technology like Prepar3D. Given that Sandia Labs is the main stakeholder, where Steven Smith is speaking on their behalf. We know that our project will be utilized by Sandia Labs for some kind of flight simulation extension product for Prepar3D itself.

Broad Impact

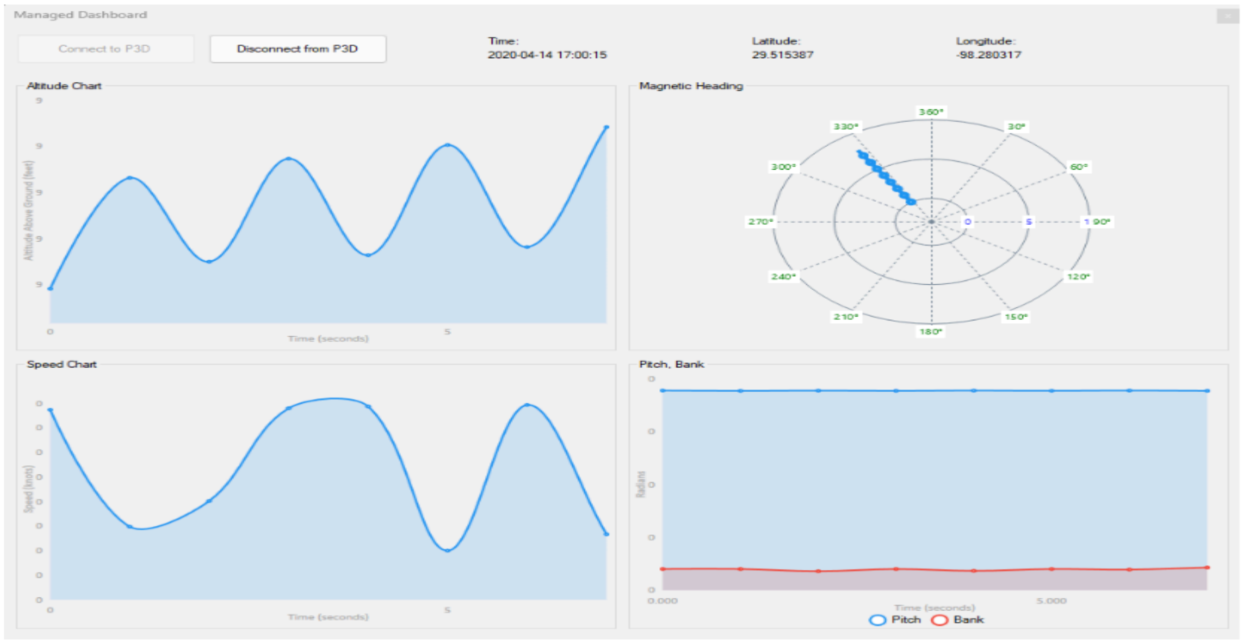
Our project development with Prepar3D will add accuracy to the simulator for precise training. We will implement G-Force indications to assist the users in their flight simulations training them to fly safely. We will refactor the GUI Dashboard to be accurate to a flight experience. We will also ensure that the Aircraft is accurate in movement in certain conditions to train the user's response time in the conditions of a real flight experience.

These implementations will benefit the target users by increasing accuracy to train the user's response time during a flight experience. This will benefit their time because the training is easily accessible compared to training and scheduling with instructors as they are not always available. Additionally, this will be cost-effective compared to paying for the real flight experience more times than needed. The goal is for the simulator to be accurate enough that users training with it consistently, can be fully prepared and have realistic expectations of what their flight experience will be like.

The problem that an effective flight simulator can solve is to help in the training of pilots. Our project will be helpful to flight instructors as they have many people to teach. Student pilots will also gain a better understanding of G-Forces and how to handle their aircraft. Plus, instructors and students can experiment with aircraft maneuvers without putting themselves in danger. Thus, our project will benefit both flight instructors and students.

Final Product

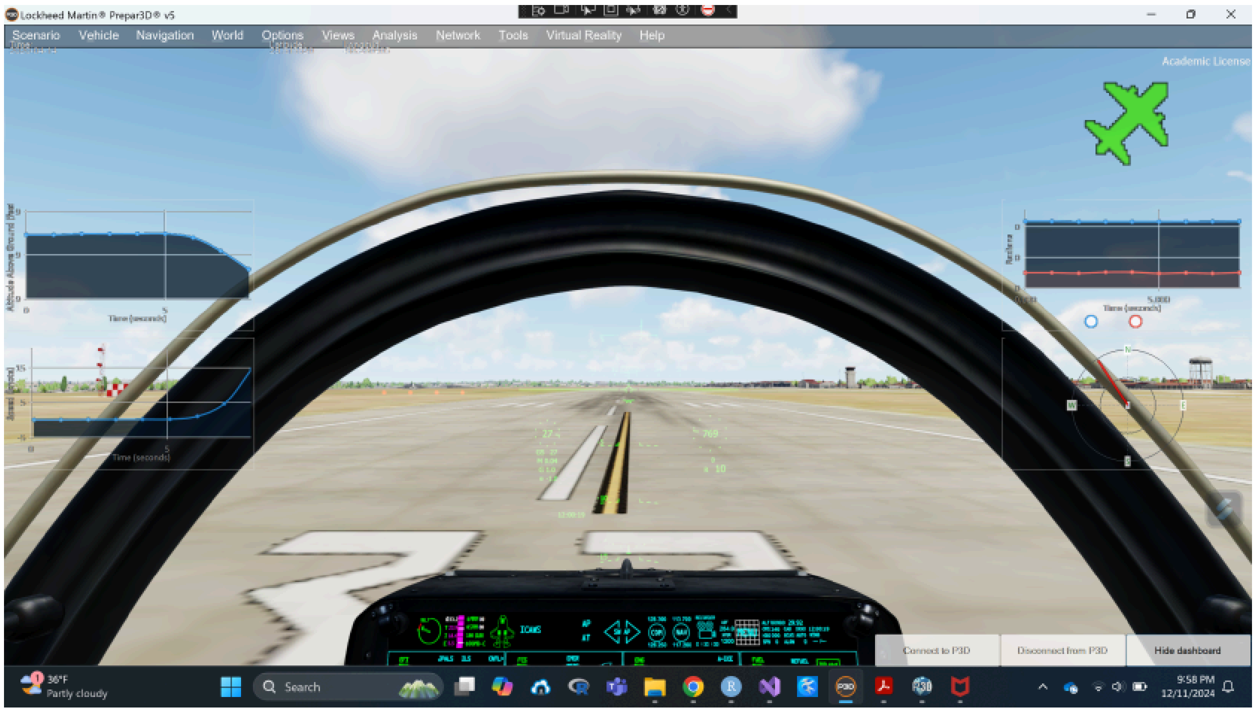
Massive strides have been made in improving the UI of the managed dashboard. We were able to convert the dashboard from a separate window that offered data visualization, to an integrated transparent overlay that snaps to the Prepar3D simulator window when opened. The following screenshots represent a before and after:



Time:
2020-04-14 17:00:15

Latitude:
29.515387

Longitude:
-98.280317



As shown above, the first screen shot was the simulator *before* our updates. It used to be a separate window form separate from the simulator. The second screenshot showcases what the project looks like now. It is a fully functioning overlay over the simulator window. It follows the simulator window everywhere it goes, and automatically resizes itself to fit the simulator window whenever the user changes the size of the simulator window itself. The overlay is fully transparent, and still maintains the data visualization graphs.

Improvements were made to the compass, by adding cardinal directions and making the needle more distinct instead of a collection of smaller circles. The G-force indicator showcases whether or not the user is experiencing safe G-forces. It should turn red once G-forces surpass a value of 3. Finally, the other groupboxes show the altitude, speed, and pitch as the simulation is running. The user still needs to first connect to the simulator in order for data visualization to begin. Once they disconnect, a special CSV file will be generated in the "Output" folder of the project containing the flight data of the session recorded by seconds. A PDF file will also be generated containing serialized CSV data. Finally, a third button was added that allows the user to hide the overlay when they no longer wish to see it on screen, but still want it to be connected during the session.

Design and Development Process

We utilized trunk based development for our project that was hosted on GitHub. We used the branch "main" as our trunk branch such that any and all pull requests would be created from that branch. Each team member made a fork of the project. We would create a new branch based off of “main” and add our changes into it. Once we were done, we pushed this branch into a given fork and created a pull request from there. This pull request would be reviewed and later merged back into the “main” branch. This cycle would repeat until the end of the project. This method of trunk based development helped avoid merge conflicts during development, since any new changes made to “main” could be rebased onto a given pull request with ease. This led to fast code reviews whose changes could easily propagate to each of the forks with a simple pull or fetch request from Git for each team member.

The project itself was developed through C# using the SimConnect SDK provided by the Prepar3D simulator. We tried to get the project fully integrated with the simulator as a plugin or “addon”, but later realized that we would have to rewrite the project in C++ in order to do that. Thus, we decided to turn our project into a basic window overlay that would be transparent and follow and snap itself on top of the Prepar3D window on the user’s screen. Improving the appearance of the overlay was rather difficult, because WinForms does not offer a lot of options for window form customization. Most of the difficulties we encountered came from WinForms either being very particular about how to render things, or doing things that we did not expect at all requiring clever workarounds that took a bit to figure out. In hindsight, we should have moved the project away from WinForms, and towards WPF.

An example of WinForms causing trouble was trying to improve the look of the overlay. Initially we wanted to add a transparent gradient over the overlay to improve its readability and contrast, but WinForms does not offer a lot in the way of gradient manipulation. The initial attempts to do this made the overlay even more unreadable since the gradient would reset for each group box instead of being applied uniformly over the entire overlay. This issue was not figured out in favor of trying to figure out how to produce PDF output from CSV data.

Overally, we learned a couple of things during the development of the project. Firstly, we all had to adapt to C#. None of us had any real experience with C# or Visual Studio at all. Thus, trying to understand the code base provided its own challenges. A lesson learned was that we should have put more effort in understanding the original code base better. We simply began to add features through trial and error in order to get where the project currently is. We also should have put more effort in refactoring the code base to use a much better design.

Currently, the code base is essentially one giant god class called “Form1”. All logic and GUI rendering is stuffed into this single class. This is the same design that the previous team utilized, and we decided to keep it in fear of breaking existing features. Now that we have a better overall understanding of the code base regarding the intricacies of function interactions in “Form1”, we roughly know what can be refactored without issue. However, most efforts were taken to add features as time was spent improving the build system. Thus, we felt we needed to try and get close to what we promised to deliver instead of trying to fix up the code base. In reality, we should have taken the time to fix up the code base more than we did to make development much smoother.

Future Work

To begin, the current plan in distributing the project is to have people first obtain a copy of the Prepar3D simulator with the Prepar3D SDK version 5.4.9, and clone the project from GitHub at https://github.com/awkless/SandiaLabsPrepar3D. The user will need to open the project through Visual Studio IDE, and run it along with the Prepar3D simulator. This is the only method of deployment we can currently achieve. Another method is to turn our overlay into an “addon” for Prepar3D, but that would require us to rewrite the entire project in C++, which would take time that we no longer have.

Moving on, we ran into many issues with the UI due to us using WinForms. It may be in the best interest for future implementers and maintainers of the project to move away from WinForms in favor of alternatives like WPF. Future maintainers may also want to improve the implementation of the PDF output function so graphs and other visuals can be presented instead of just barfing the same CSV output in PDF form. Improving the G-force detection logic is also needed since it is currently broken. Improvements to the organization of the source code is also largely needed given that the Form1 object is a god class that houses all the implementation details of the overlay. It may be beneficial to separate out the logic into separate files instead of shoving everything into the Form1 class. Implementers may also want to try and implement our plan for providing grades to the user for how well they did in flying the aircraft in the simulator. We ran out of time to do this such that our efforts were mostly spent improving the UI as much as possible despite the difficulties WinForm imposed on us.

All in all, despite the many improvements to our code that can be made, we did make great progress in improving the UI that was offered by the previous team that worked on this project. Hopefully, future teams can build off of what we did much like how we built off of the previous team despite the challenges.