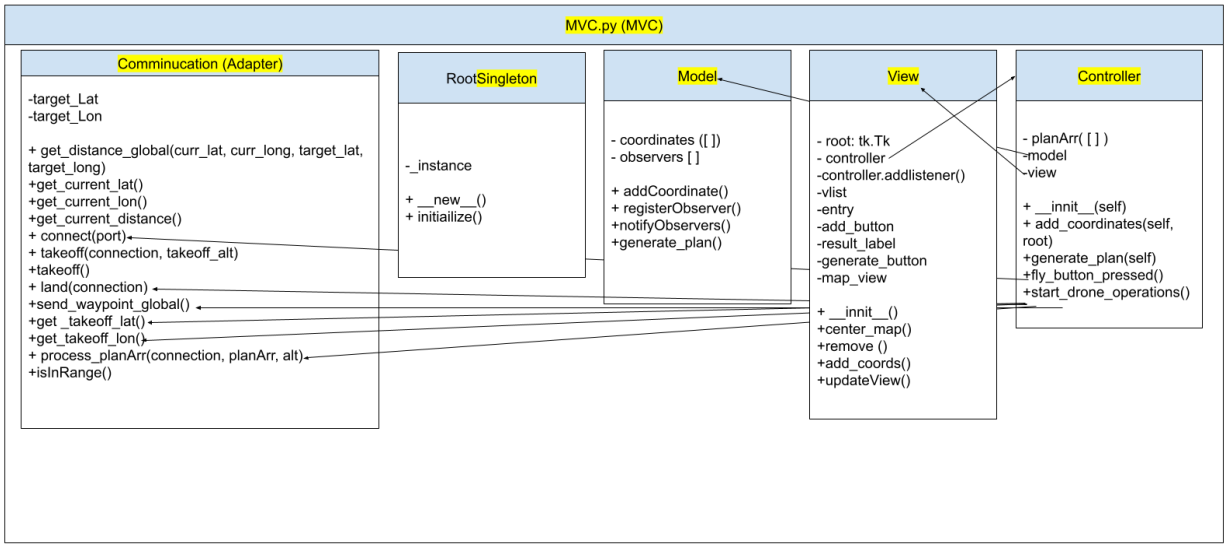
Spot-On Drone UI
Jack Lay and Charles Rothbaum
Object Oriented Analysis and Design

Final State of System

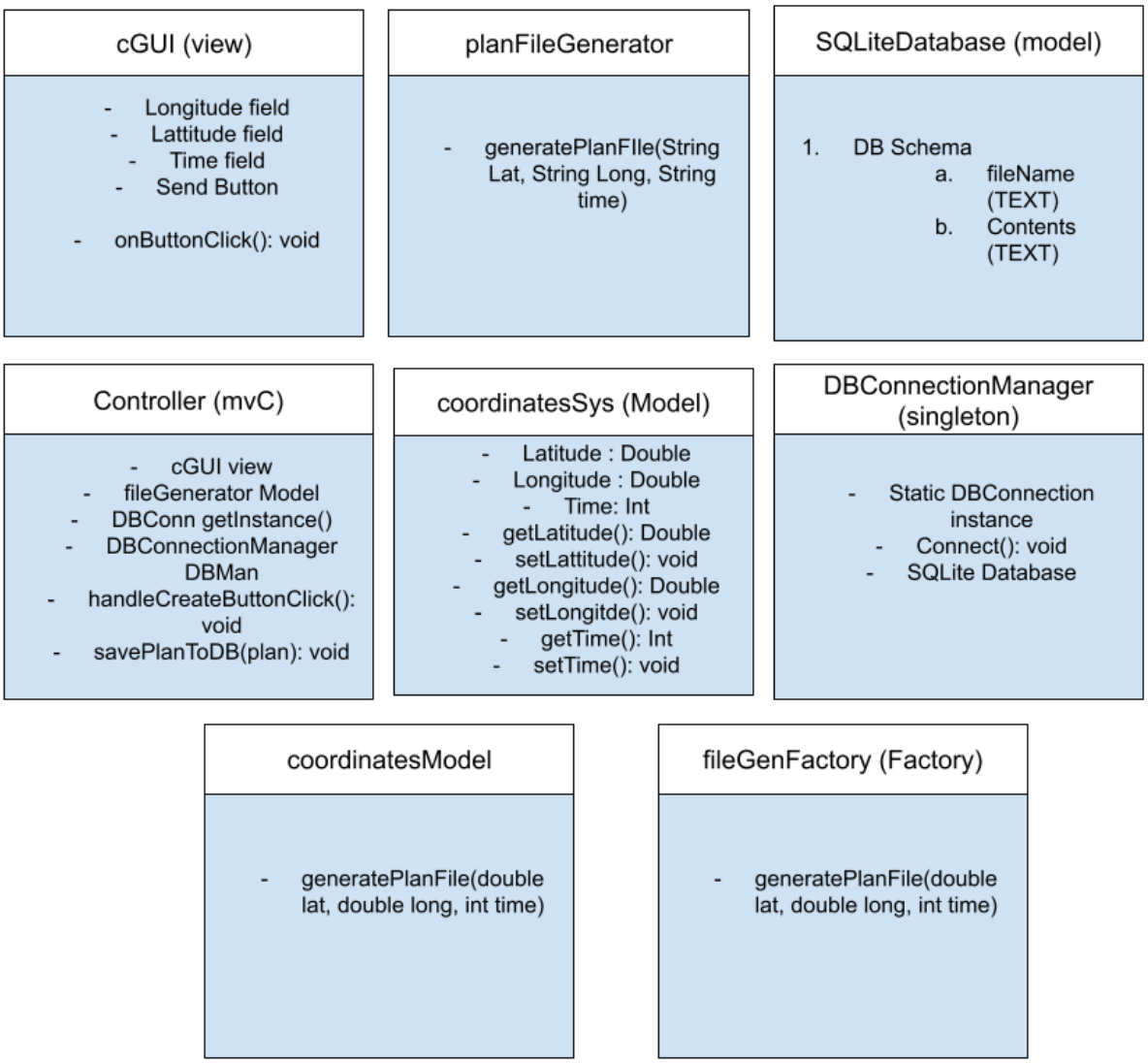
Within this project, we used python to implement a Graphical Interface to send commands to a drone. Initially, we were going to build a C# app to potentially interface with Autodesk, but we did not go through with this, due to a lack of documentation, and then moved on to Java. We had issues using the generated mavlink library with our Java project, so to finally get it off the ground, we switched to Python, because this was the easiest to set up and get going. Jack Implemented most of the gui, which initially was text fields for manual entry of points, and buttons to add the points to the plan, and another to generate the plan and send it to the drone. Next, Charles implemented a clickable map that would take locations from your mouse-click and add them to a visual list on the gui. Another cool thing about this map is, that we were able to enter a location, and view it on the map. Using this map did not only solve the problem of point entry, going from manual entry to now automatic entries, it also allows for the user to view and alter the points in the list in that moment. Charles implemented the “back end” for this app, which entailed taking the generated array of coordinates from the gui, and converting them into instructions for the drone. To do this he used mavlink, which is a communication protocol for Rc vehicles. For practicality, we sent the instructions to a simulated drone, showing it traversing the area on the map in the gui, but this program can be used with real drones, with minor configuration.

UML:

P7 UML:



P5 UML:



Between the start of this project in Project 5 (P5) and its completion in Project 7 (P7), we underwent several key changes. Initially considering C#, we later shifted to Java due to documentation issues but faced challenges with the mavlink library. Finally, we settled on Python for its simplicity. In our UML, a major update involved consolidating classes into five, merging all drone instructions into one adapter pattern. We revamped our Model-View-Controller (MVC) structure, introducing an observer and a new singleton pattern while ditching unnecessary elements like the factory pattern. This time, the root of the view window acted as the singleton instead of a database connection, which we didn't need. We simplified our design by replacing the coordinatesModel class with a tuple and removed the SQLite database wrapper class since we didn't use a database.

Third-Party code vs. original code Statement

We did not take code directly from sources such as ChatGPT, but did look at websites such as GeeksforGeeks, StackOverflow, and refactoring.guru for structural examples, and
Singleton Example:
https://www.geeksforgeeks.org/singleton-pattern-in-python-a-complete-guide/
https://refactoring.guru/design-patterns/singleton/python/example

MVC Example:
https://gist.github.com/ajfigueroa/c2af555630d1db3efb5178ece728b017
^This is from this question: https://stackoverflow.com/questions/7638139/how-to-implement-the-mvc-pattern-in-tkinter
https://www.geeksforgeeks.org/mvc-design-pattern/
https://www.giacomodebidda.com/posts/mvc-pattern-in-python-introduction-and-basicmodel/
^ This was also referenced for the visual CRUD list, not for the observer part though

Observer Example:
https://refactoring.guru/design-patterns/observer/python/example
https://www.geeksforgeeks.org/observer-method-python-design-patterns/

Statement on the OOAD process for your overall Semester Project:

The main hurdle we had to get over was finding a language to use, we bounced through 3 different languages before we were able to get it working. This was due to our need to use a mavnet library, which usually came in the form of a library that was generated automatically. The C# library had no documentation, and we could not get the Java mavgen library to work, so we instead settled on python. Another issue we had was with threading. Due to the way our GUI was setup, the only way to reach out to the adapter and tell it to make a plan was to close the gui and then send the array of coordinates off after it closed. To fix this, we moved everything into the same file, and set up threading, so that when asked to, the gui will generate and send points without having to close first. The final hurdle we had was about filling the project requirements for the number of patterns. Our app was simple enough that we struggled to fill in the number of patterns required for this project and ended up throwing in a singleton just to round out the requirements. Not necessarily a negative though.