DEPARTMENT OF COMPUTER SCIENCE

COLLEGE OF ARTS AND SCIENCES

CSCI 4961/4962 CAPSTONE DELIVERABLE ONE

Title of Project: Multi-User Virtual Reality Interface for Control of Aerial Drones
Client: Dr. Srikanth Gururajan
Supervisor: Dr. David Ferry
Student(s): Kyle Coleman, Woo Seok Yang, Logan Leavel

1 EXISTING SYSTEM

- The system is made of three distinct components
  - Master PC
  - Drone/PixHawk/Pi
  - Streaming/VR System

1.1 MASTER PC

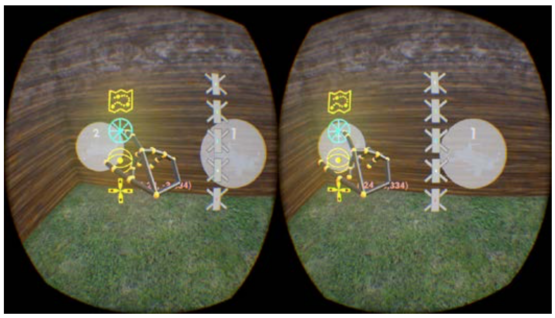
- Runs the main system software
  - Facilitates communication between VR system and drone for control instructions.
  - Translates controls from VR in a way that can be read by PixHawk

1.2 DRONE/PIXHAWK/PI

- Handles flight logic
  - Interprets flight controls received from Master PC and handles execution
  - Handles telemetry of drone flight typical of drone flight controllers
  - Can communicate back to Master PC when flight errors occur (invalid flight instructions for example)

1.3 STREAMING/VR SYSTEM

- Creates the VR interface for the user
  - The VR interface renders the drone lab flight space
  - Renders the drone itself
  - Provides controls for the drone which you interact with by hand
- Interfaces with an infra-red camera tracking system
  - The camera tracks the location of the drone using infra-red lasers
  - The camera returns positioning coordinates to the streaming PC
  - The streaming PC uses MatLab to translate those coordinates into a different style of coordinates usable by the VR software.



2 Project Goals

We mainly aim to transition the current system into a functional multi-user system. This will allow for multiple users to control one or more drones within a shared VR space. The system must accurately update the VR space for both users with proper positioning of rendered drones. Ideally, the system will allow users to "hand off" their drone to another user and will prevent users from taking control of another user's drone without their permission.

To do this, we plan to implement a client-server architecture. The server will receive and forward all data that was previously passed from the Master PC to the drone. This includes flight control instructions generated by the VR machine. By doing this, we can avoid major modification of the existing code.

The Master PC will now transition to acting as a client. It will retain all of its previous functionality, but for clarity's sake, we will refer to it as a client now.

The new system will consist of:

- One server
- N clients
- M drones

3 Server

We plan to model the functionality of a server based on that of a real life flight control tower(FTC). A FTC serves 4 main purposes.

- Retain current positioning of all aircraft
- Record predicted flight path of all aircraft
- Maintain communications with all aircraft
- Predict and prevent potential collisions of aircraft.

We aim to develop functionality based around the first 3 with the 4th being a stretch goal.

The importance of retaining current positioning data of all of the drones is 2-fold.

First, it needs this positioning data so that it can relay the coordinates of every drone in the space to each client. Currently, this is less of an issue because every drone is being flown in one enclosed and mapped space and each drone is being tracked by the same infrared system. This means that each VR environment will receive real time positioning data on each drone. However, this will not be the case if the system is further developed to support different environments or transitions to using GPS for position tracking. Therefore, it is necessary for the server to maintain the positioning data for each drone so that it can send this data to each client so that the drones can be rendered in each user's VR space.

Secondly, one of the biggest benefits to the client-server architecture is that we can aggregate this flight data on the server to perform data processing. This data processing will facilitate functionality like boundary enforcement (it will determine if the user's commands will result in the drone leaving the rendered VR space), hand-off of drones from one user to another, and prediction of drone collision. Other functionality likely exists, but given our lack of experience in this kind of project, it won't be fully realized until we are faced with a problem that can be solved by data processing on the server. The benefit of data processing on the server is not the main motivation for the server-client architecture, but it is definitely a secondary benefit of the design.

The server's main purpose will be to facilitate communication. The client-server system allows for us to easily send necessary data about any particular drone (such as positioning data) to any of the connected clients via a broadcast. This is how we plan to update and render the position of drones in every user's VR environment. Flight instructions are inputted by the user in the VR interface and sent to the user's client. The client will then forward these instructions the the server. The instructions will contain some kind of header data that specifies which drone will receive these instructions. The server can then send the instructions to the specified drone.

4 CLIENT

The former Master PC will transition into a client role. The client's only real responsibility now is the run the code that establishes communication between the VR environment and the drone. It will establish communication with the FTC and provide a list of the drones that the client is currently in charge of. This will allow the server to communicate with the client's drones directly. It may seem counter-intuitive to add another step (the FTC) into the system when the client is already capable of communicating with the drone. However, this is necessary to prevent each client from being directly responsible for communicating with other clients. A peer-to-peer setup would quickly get messy as the number of clients grows and also eliminates the possibility of doing concise data-preprocessing to detect possible collisions. It makes more sense for one (likely more powerful) machine to handle any difficult computation than having all involved clients either handle the computation independently, or depend on one client to do the computation and relay the results to every other client. The consideration of a peer-to-peer or "mesh" network will be discussed in a later section.

5 DRONES/PIXHAWK/PI

The drone's role in the system remains virtually unchanged. It is only responsible for interpreting and executing flight controls. Since we are not going with a mesh network, the drones will not be directly responsible for communicating with other drones or users.

6 MODIFICATIONS + ADDITIONS

As of right now, we have developed a definite list of necessary modifications and additions to the system.

6.1 SERVER ADDITIONS

This is by far the largest addition we will be making.

- Since the server currently doesn't exist, we will need to implement everything from scratch
  - Basic communication between a server and client(s)
  - Functionality for data processing
  - Drone indexing to keep track of which drone is which

6.2 CLIENTS

The biggest benefit of the server-client architecture is not needing to modify much of the code working on the current Master PC. We do need to modify some things though.

- Networking
  - Each client needs to establish a connection to a server, which means adding in the functionality to do so
  - Each client needs to establish connections with multiple drones. Currently, the Master PC only connects to one drone (using a hard-coded IP address)
- Multiple Drone Support
  - Because the system is only designed to work with one drone, we will need to modify any initialization programs to work for N number of drones. This includes things like our Drone class and various Listener classes.

  More than likely, we will discover more problems with multi-user support that can be solved by making modifications to the existing code that we haven't already thought of in our initial assessment.

6.3 VR Environment

Currently, the application for the VR environment is only rendering drones based off of positioning data it is receiving directly from the infrared camera setup. Because all of the drones in our project will be in the same lab and will be tracked by the same camera setup, this will not be a problem. However, we do want to develop functionality for receiving positioning data of all drones from the server. This is because we can not expect to use an infrared camera system in future development if tracking is done by GPS. So, we need to make sure the VR system can communicate with the server and steadily update it's drone renderings based on a collection of positioning data received from the server. We also plan to make some GUI adjustments to reflect the change to a multi-user system. These things may include labeling for each drone (Kyle's Drone 1, Kyle's Drone 2, Austin's Drone 1, Logan's Drone 1, etc.) as well as an interface for releasing control of a drone so that you can give control to another user.

7 Consideration of a P2P or Mesh Network

Dr. Esposito and I (Kyle) talked briefly about the benefits and efficacy of using a mesh network architecture instead of the client-server setup we have now. I consolidated the argument into a list of pros and cons.

7.1 P2P Pros

- A P2P network removes an intermediate step in communication, theoretically reducing latency in drone-to-drone communication.
- Systems for this exact thing currently exist and could reduce a lot of the workload on the group.
- Deconfliction and coordination would be easier if each drone was immediately aware of the position of other drones

7.2 P2P Cons

- More computational responsibility is placed on the drones
- The Pi on the drone is not really designed for any kind of complex computation (collision prediction)
- Additional computation requires more power draw. The drone is already running on a small power source, that being a power bank designed to charge cell phones.
- P2P networks introduce more points of failure in communication. If we don't have a complete web of connections, then we will have to daisy chain communication from one drone to another. This would mean that if one drone fails to communicate, multiple drones will be affected

In my opinion, I feel that the pros are outweighed by the cons. There seems to be much more to gain in terms of reliability and simplicity by going with a server-client architecture.

8 Timeline

Friday, October 13, 2017	Evaluate the existing system and determine where modifications need to be made
Friday, December 15, 2017	Implementing client and server, define failure modes and handling, form architecture plan
Friday, December 15, 2017	Make sure the system works on actual hardware not flying on a drone
TBD	Full system integration and flight test
TBD	Polishing, reviewing documentation, stretch goals (outdoor flying)