CSM#2 - Quadcopter

Chase Tyree, Matthew Childers, Mykala Miller, Sean Zeiler, Tri Nguyen

June 17, 2014

I. Introduction

CSM#2's client was Dr. Cameron Turner, a Mechanical Engineering professor at the Colorado School of Mines. Dr. Turner's area of expertise is in the areas of design and computational engineering methods, which allows engineers to utilize the power and efficiency of computers when designing, modifying, and manufacturing projects.

Dr. Turner is involved in certain robotics projects within Mines. One of his visions is to have a small quadcopter swarm that can accomplish certain tasks with or without a participating human. This project was a starting point for this vision, and required the team to control a Parrot AR-Drone 2.0 quadcopter through Google Glass. It is anticipated that further development will expand control from the Glass to include more than one quadcopter, and perhaps different brands of quadcopter.

II. Requirements

The functional requirements for this project include:

- Receive information from the quadcopter
- Send proper commands to the quadcopter
- Display video footage from quadcopter on Glass
- (Preferable) Commands are voice commands
- (Optional) Implement control for GPS functionality
- Interpret commands from the Glass into commands for the quadcopter. Commands for the quadcopter include:
  - Takeoff/Land
  - Fly forward, backward, left and right
  - Rotate left or right
  - Hover
  - Gain/lose altitude
  - Flip
  - Emergency Stop (Kill/Reset)

The non-functional requirements for this project include:

- Build an application for the Google Glass
- Minimize damage to quadcopter and Glass

III. System Architecture

This projects rests on two systems, the Parrot AR-Drone 2.0 quadcopter and the Google Glass. The quadcopter emits it's own WiFi network, navigation and health telemetry as well as receive commands over this network. It also sends over raw video data over the network. The navigation and health telemetry is contained within a "navigation stream" (nav stream) that is essentially a bitstream of many different navigation messages, one of which is GPS information. The nav stream also has a "demo" mode that is formatted in a known structure, but contains limited information.

The Google Glass was initially chosen for it's heads-up display capabilities that allows a user to potentially multi-task a little easier. It contains voice recognition technologies, GPS, and sensors

such as accelerometers and gyroscopes, and the sensors associated with the touchpad. This makes the Glass a versatile tool able to accomplish many tasks.

Initially, the team thought it would be necessary to have the Google Glass and quadcopter send and receive information through a cell phone application that would essentially act as a router between the two devices. This architecture is shown in Figure 1 (below).

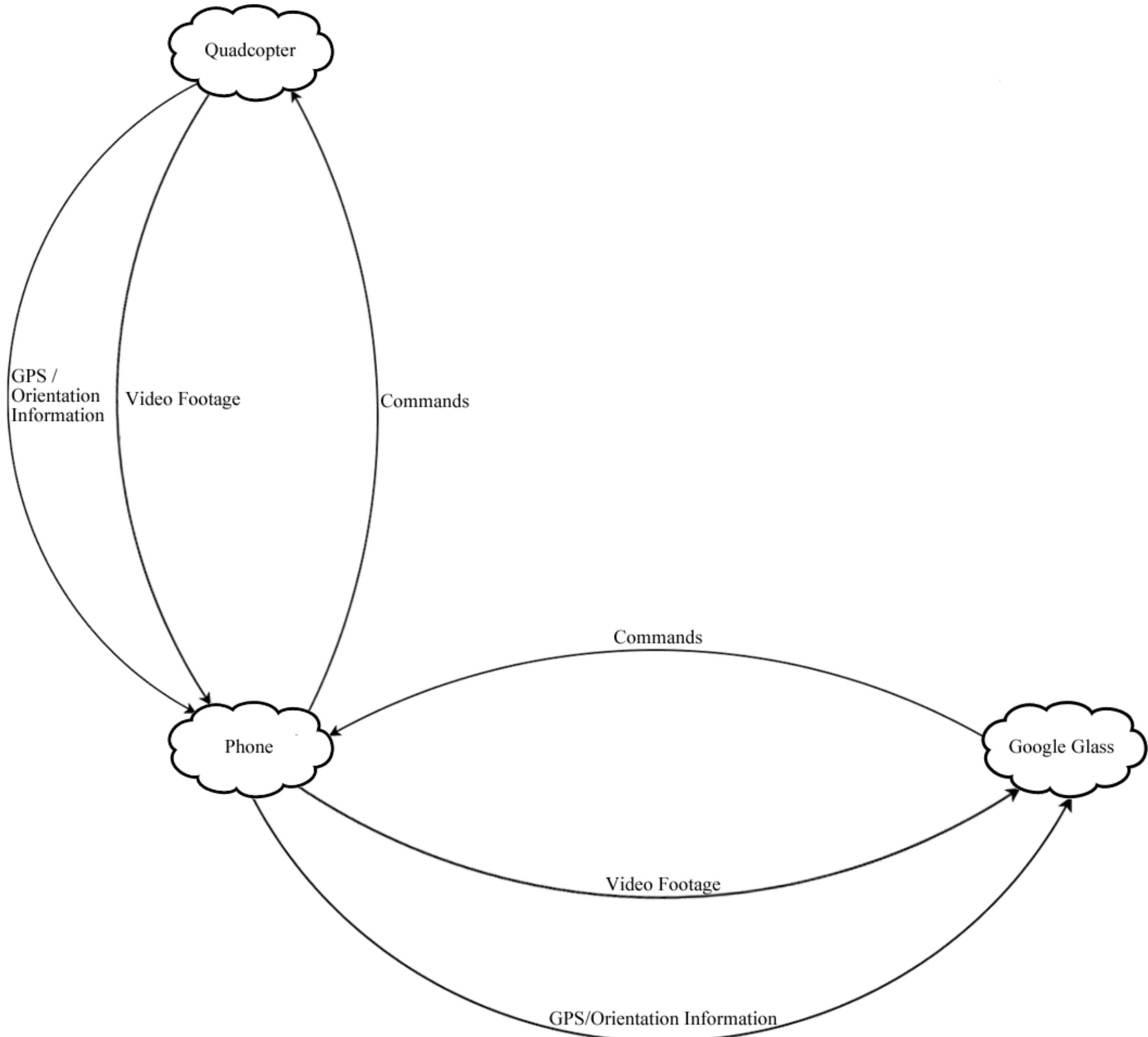

Figure 1: Initial Application Concept

Upon further research, the team discovered that the router application is not necessary - the quadcopter can be controlled through the Glass itself. This simplified our data flow to what is seen in Figure 2 (below).

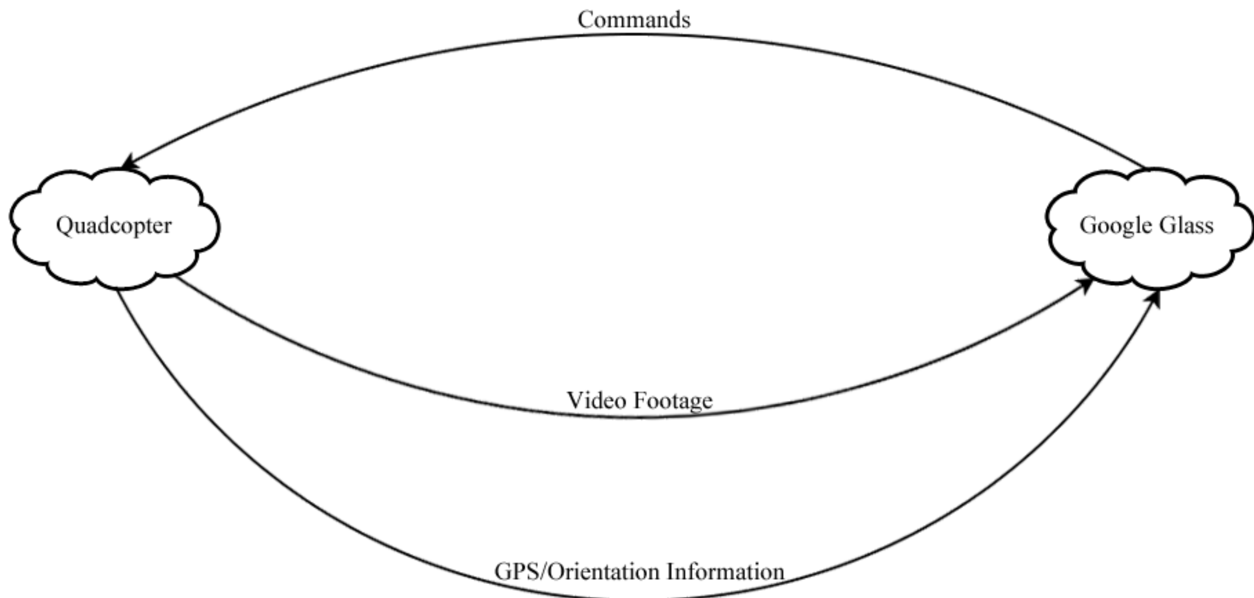

Figure 2: Data Flow Between Glass and Quadcopter

As the team attempted to implement this architecture, we ran into several issues, one of which was that Google Glass does not have the processing power to display video and send commands to the quadcopter often enough to ensure smooth flight. We also found that the "Nav Stream", which contains the GPS and orientation information, is largely unreadable without a significant amount of extra processing. These two factors combined led to the final architecture shown in Figure 3 (below).

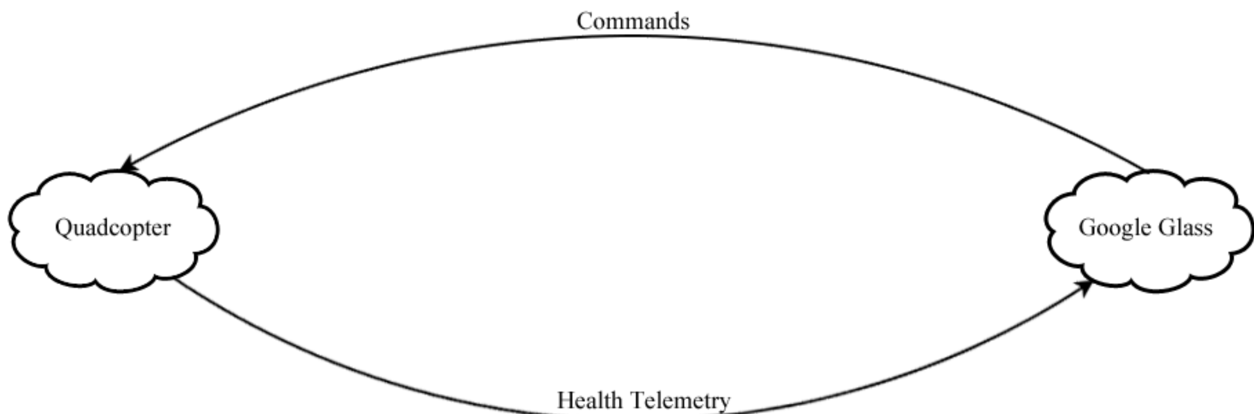

Figure 3: Final Data Flow Between Glass and Quadcopter

IV. Technical Design

Within the Glass application, tilt and touchpad commands will control which commands are sent to the quadcopter. Tilt commands include the forward, backward, left and right commands that cause the quadcopter to go forward, backward, left or right. To invoke these commands, the user needs to tilt their head in the appropriate direction.

The touchpad commands include commands that cause the quadcopter to takeoff, land, flip, and emergency stop (kill). A set of “Toggle Commands” is also included in the touchpad commands. The Toggle Commands include Gain Altitude (Up), Loose Altitude (Down), Rotate Left, Rotate Right, and Pause commands. When they are activated, they disable the communication of any other command until they are toggled “off”. The touchpad commands are mapped to the following gestures:

Takeoff:	One finger tap on touchpad (if not flying)
Land:	One finger tap on touchpad (if flying)
Pause:	Two finger tap on touchpad (again to toggle off)
Flip:	Three finger tap on touchpad
Rotate Left:	One finger swipe forwards (again to toggle off)
Rotate Right:	One finger swipe backwards (again to toggle off)
Up:	Two finger swipe forward (again to toggle off)
Down:	Two finger swipe backwards (again to toggle off)
Kill/Reset:	Swipe up on touchpad

The command flow within the application is shown in the finite state automata in Figure 2 (below). A takeoff command must be issued before any subsequent commands can be issued. Once the takeoff command has been issued, any of the commands illustrated above can be issued. If the land command is issued, the user can either exit the program, or issue another takeoff command. If the kill command is issued, the user must issue the command again before the quadcopter can be used again. It should be noted that the “Wait for Command” state shown in the figure does not transmit anything to the quadcopter.

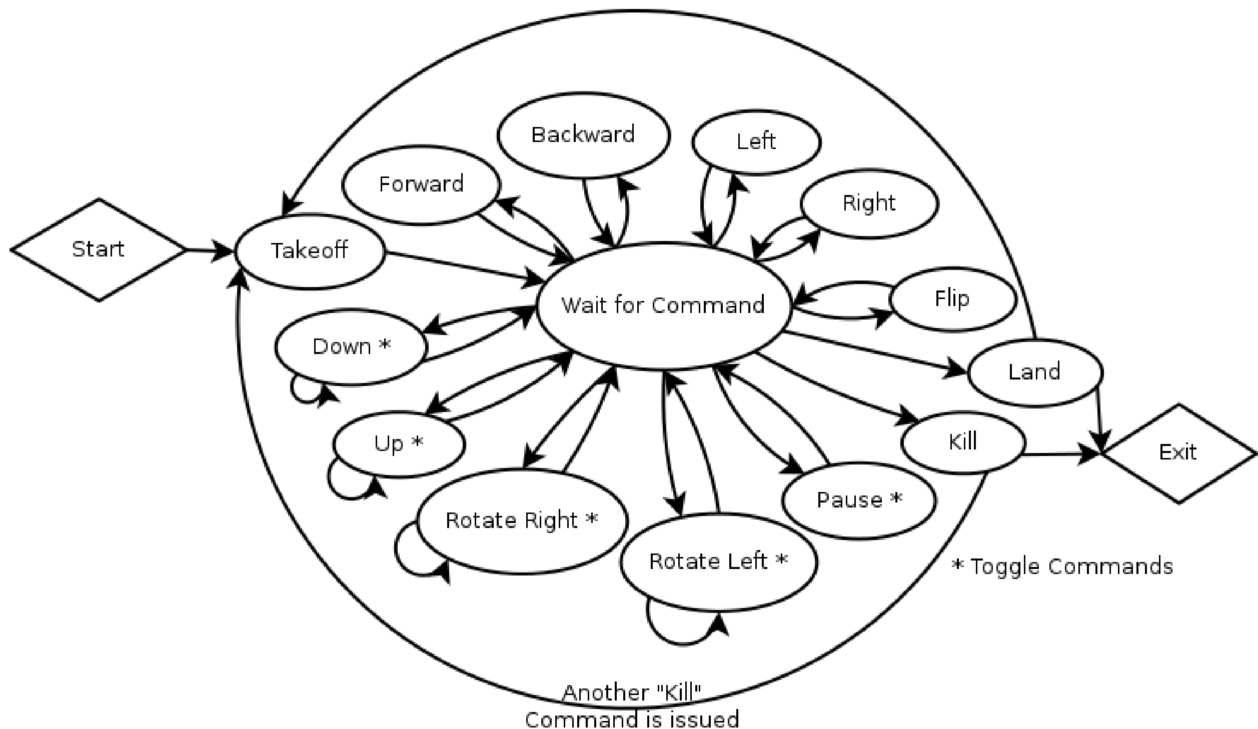

Figure 4: Finite State Automata of Commands for the Quadcopter

The code for the project has been divided into 4 files, described below:

ARdrone.java:

Contains functions that map the commands discussed above to drone commands. Drone commands are simply bit patterns that the quadcopter can recognize and execute.

NavData.java:

Parses the drone output to give real-time data about the quadcopter to the Glass application.

UDPPacketSender.java:

Contains code to send commands to the drone via the drone's WiFi network.

MainActivity.java:

Contains the core app functionality including mapping Glass inputs to the appropriate commands, the visualization of the app on the Glass, and listeners for the drone data.

V. Design & Implementation Decisions

The first major design decision was to interface directly from the Glass to the quadcopter without an intermediate phone interface. This simplifies the necessary network connections and also eliminates the need to develop an application for the phone. This decision overrides one of the initial requirements we had for the phone app, as we didn't realize that the Glass could connect directly to the quadcopter.

The most popular IDE for Android development that we could find was Eclipse. However, the team had issues with the Eclipse IDE. It had trouble supporting all the libraries necessary for this particular project. The Intellij IDEA IDE was chosen to develop our Glass Application over Eclipse as it is the only one we can find that can support all the necessary libraries.

Initially, the team intended to map the quadcopter commands to voice commands that the Glass would interpret. However, we found that implementing voice recognition for the Glass impossible, due to a bug in the Glass's firmware. We were unwilling to flash the Glass back to a previous firmware version, as that had potential to permanently break the Glass. Due to these problems, the team decided to use the tilt and touchpad sensors on the Glass.

Another error that the team discovered during development was that the Glass was not powerful enough to decode and display the video from the quadcopter as well as issue the quadcopter commands frequently enough to control the quadcopter reliably. This meant that we could not use projects already written that incorporated video, as they were usually so large that we couldn't decipher how to remove the video. Because these projects also decoded the raw nav stream coming from the quadcopter, we also didn't benefit from listeners such as the GPS listeners already being implemented. The team discovered these errors too late to be able to implement suitable workarounds, such as having the quadcopter take pictures instead of video, and simply displaying a picture or two every second. It also meant that we didn't have time to implement listeners for the raw nav stream, making GPS functionality impossible to develop.

As we tested and used the application, design flaws with certain commands were found. The most severe of which was that initially the kill/reset command was mapped to a two-finger tap, and the pause feature was mapped to the swipe up gesture. This worked well until we started testing the quadcopter's flip (mapped to the three-finger tap) abilities. In order to flip, the

quadcopter needed to be at a high altitude, and occasionally when the user issued the flip command, the third finger was not registered, in which case a kill command was issued instead. The kill command cuts of power to the motors, so the quadcopter ended up falling uncontrollably out of the air from a high altitude. This resulted in significant damage to the quadcopter, and as a result, the team decided to move the kill command to a gesture that was unique, and not likely to be misinterpreted. For this purpose, the pause and kill command gestures were swapped, so that if a flip command was again misinterpreted, the quadcopter would instead hover.

Less severe command issues related to the ease of control of the quadcopter. The Toggle commands were introduced to allow the user to focus on one thing instead of many. For example, the up command initially did not disable any other commands while it was being used. This meant that while the quadcopter was gaining altitude, the user could not watch it. If they did, their head would tilt backwards, telling the quadcopter to go backwards. This made it remarkably difficult for the user to control the quadcopter, and decide whether or not it had gained the necessary altitude.

VI. Results

While the team was able to control the quadcopter with the Google Glass, we were unable to complete all the desired functionality. Any future work with the project should include finishing some of the tasks that the team was unable to accomplish. This includes displaying the video on the glass, and decoding the nav stream such that the full functionality of the quadcopter can be unlocked. In time, Google will correct the firmware issue that prevented the team from using voice commands, and that too can be corrected.

For the video, future work could mean adding in intermediate hardware such as a Beaglebone system that intercepts and decodes the raw video footage into something that the Glass could display easier. A quadcopter other than the Parrot AR-Drone 2.0 could also be considered - there are some that don’t require the user to decode the video stream.

Additionally, in order to comply with the original intent of the project of being able to complete a swarm, future development will occur. This includes adding semi-autonomous commands to complete small tasks such as moving forward a certain distance, and turning to a certain angle, and expanding the controls such that multiple quadcopters can be controlled at any one time.