I. Introduction

The following describes in detail the setup of a fixed wing aircraft platform for student use. Both physical and software configuration will be discussed. For the purposes of testing the platform, a basic path following controller was implemented. The airframe for the platform is an electric powered Hobbico NexSTAR, which is a high wing conventional gear aircraft with a wingspan of 1.75 m. Inside the frame lies a 3D Robotics Pixhawk autopilot and an ODROID U3 for extra computing power. The Pixhawk is capable of running several versions of firmware,[1] however was configured for testing with a customized version of ArduPlane.[2] Similarly the ODROID U3 can run Xubuntu or Android operating systems, but was configured with Xubuntu for testing.

II. Hardware Configuration

Hardware on the aircraft can be grouped into three distinct categories: The Pixhawk autopilot and its peripherals, the ODROID and its peripherals, and the power distribution system. Each group interfaces with the other two. Below are figures showing in detail each of the three categories, as well as a figure summarizing the communication interfaces between them in various scenarios of use. The section is then ended with a labeled photograph of the hardware mounted in the aircraft. Figures 1 through 3 contain letters a-k associated with various components, which appear in figure 5 to aid in identification.

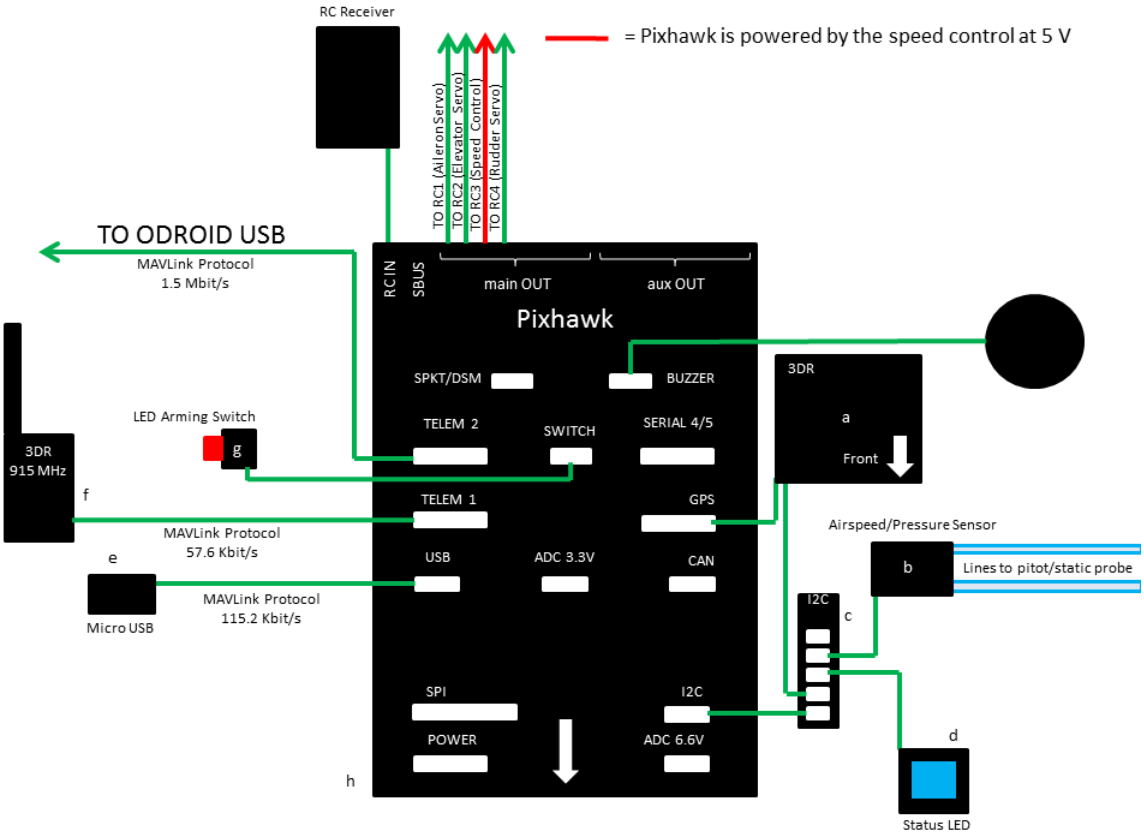

Figure 1: The Pixhawk autopilot and associated peripherals

In figure 1, it should be noted that the power port on the Pixhawk is not connected. There are three power sources that can be used simultaneously: the power port, the servo power rail, and the USB port. All operate nominally at 5 V, and provide the opportunity for redundancy. The power port is not used because the regulator supplied by 3D Robotics is not compatible with the chosen flight battery. To learn more about Pixhawk setup, see the 3D Robotics manual.[3]

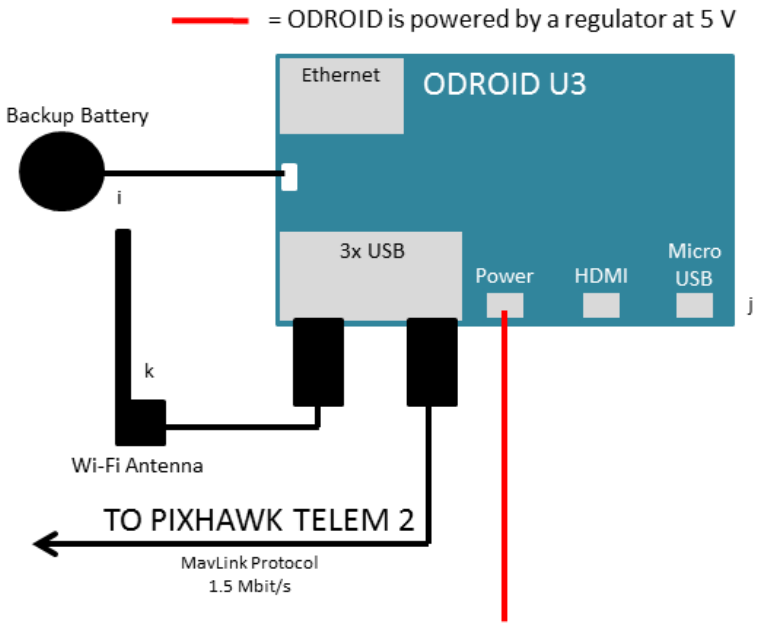

Figure 2: The ODROID single board computer and associated peripherals

Several comments should be made regarding the ODROID board and extra hardware which may be required for development and testing. Xubuntu includes a conventional graphical desktop environment which can be extremely useful in debugging and development for those not experienced with the Ubuntu terminal. Using the desktop requires only connecting the board to a monitor with an HDMI cable and keyboard/mouse. Since there are three USB ports on the ODROID, the addition of a mouse and keyboard allows the wireless antenna to stay available for use. If necessary, there is also an Ethernet port. To access it the board must be removed by unscrewing the two forward most bolts on the ODROID tray and then sliding the tray forward and up.

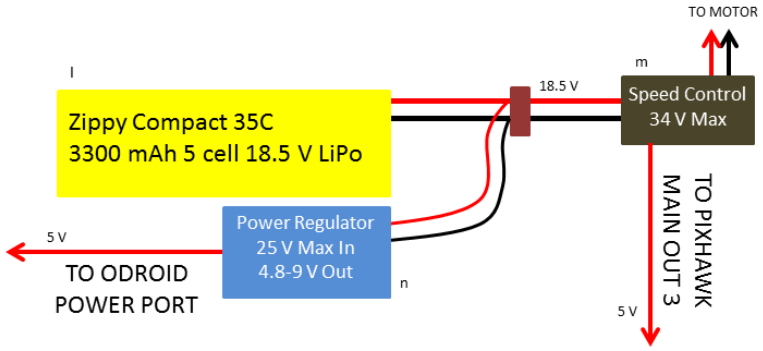

Figure 3: Power Distribution System

Figure 4 summarizes the communication between all of the main hardware components. ODROID to ground station communication is done via SSH, but the underlying network may take one of two variations: an ad hoc network or a conventional wireless network. If the latter is used a wireless router, which is not pictured, is required.

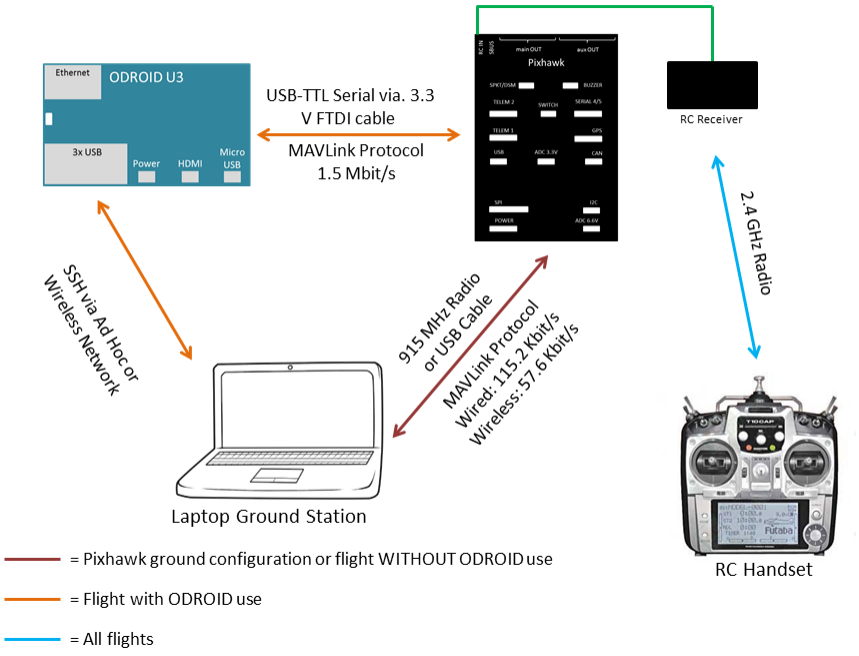

Figure 4: Full System Communications

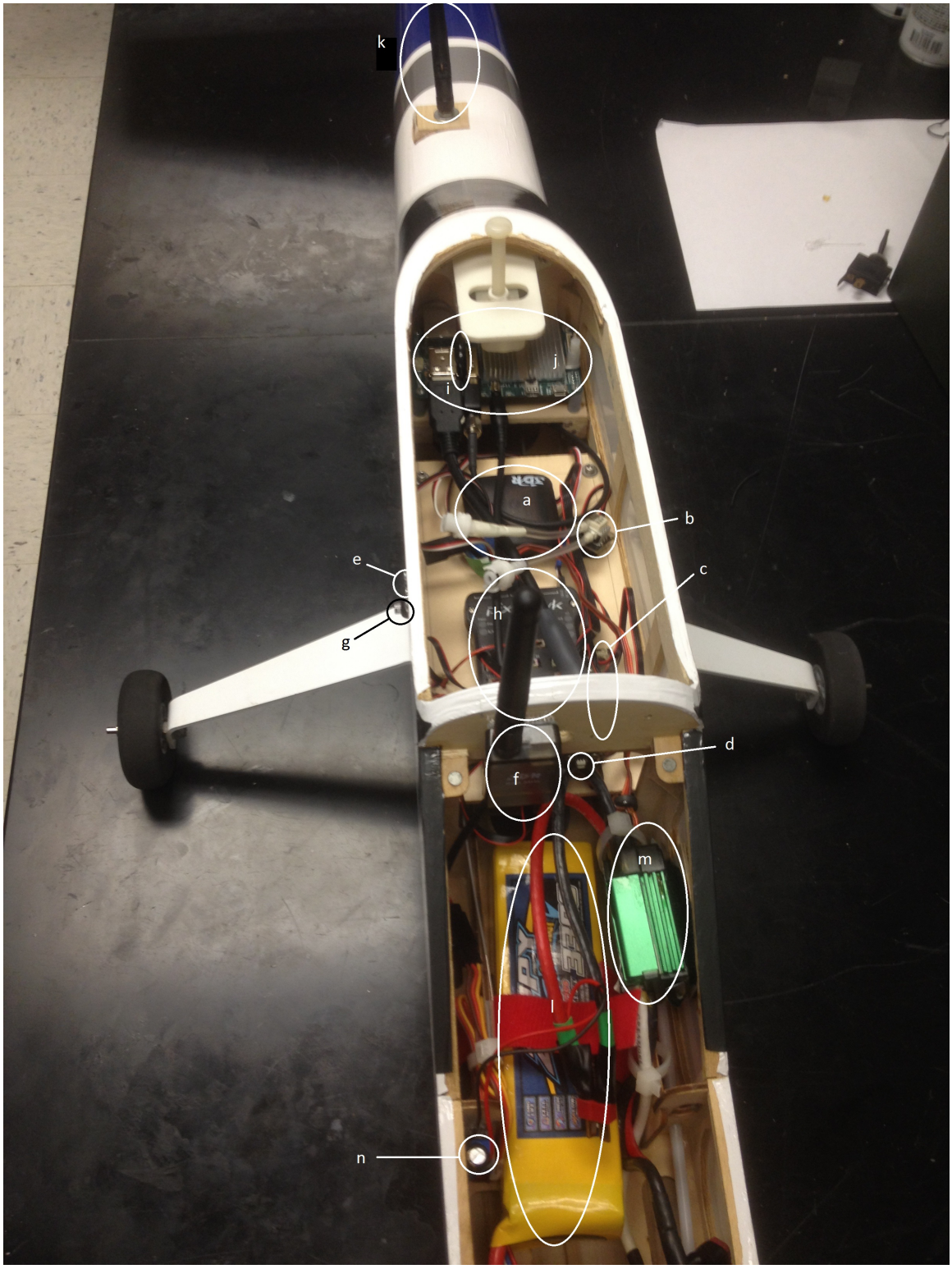

Figure 5: Installation Photograph

Several components which are shown in figure 1 are not visible in figure 5 as they are located underneath the tray supporting the Pixhawk. These are the buzzer and the RC receiver, and can be accessed by removing the four bolts securing the tray and lifting it out.

III. Software Configuration

Software implemented on the aircraft will be discussed in two sections. One section describing the general infrastructure of aircraft control and interfacing, both through the Pixhawk and through the ODROID, and one which examines how to use the infrastructure with a path following controller implementation example. Figure 4 will be useful to keep in mind through this section: the general infrastructure can be thought of as unmodified software running on the laptop ground station, the ODROID, and the Pixhawk. The example will then show one way that software can be molded and used to attack a specific problem.

A. General Infrastructure

1. Pixhawk

As mentioned previously, the Pixhawk is capable of running several versions of firmware but was loaded with ArduPlane for the example. ArduPlane is open-source, and consequently scattered, poorly documented, and sometimes confusing. On the other hand however, there is a large community of enthusiastic developers and users who are generally willing to answer questions.[4] A complete understanding of the code is not necessary for its use, so this section will include only a brief description. See the developer section of the ArduPilot website[2] for information on obtaining and working with the code. ArduPlane is configured to be operated in different flight modes with varying levels of autonomy. Table 1 describes three of the main modes. See the website for complete descriptions of all other modes.

Table 1: Major ArduPlane Flight Modes

Mode	Description
Manual	ArduPlane reads the RC handset control positions and sets the servos directly from those positions. The RC pilot has full control as if the Pixhawk were not there.
FBWA	Stands for fly-by-wire-A. RC handset aileron and elevator stick positions are translated to roll and pitch hold commands and the throttle is directly controlled by the RC pilot.
Auto	Pixhawk has full control of the aircraft. In this mode it is possible to perform waypoint navigation and autonomous takeoff, landing, and various other tasks

By examining the FBWA mode, the general method of handling control that ArduPlane takes can be understood. There are two levels of code, looking from a control perspective: the "what to do" level (/ardupilot/ArduPlane, if you have downloaded the code) and the "how to do it" level (/ardupilot/libraries). "What to do" handles the scheduling of tasks, what is done in each autopilot mode, the gathering of sensor data, checking for failsafe conditions, etc. For example, the function update_flight_mode() is a task run every so often by the scheduler. This task checks which mode the autopilot is currently in and performs high level control accordingly. For FBWA, the RC handset control positions are read and those associated with aileron and elevator are converted to bank and pitch angle commands between predefined limits. With desired bank and pitch, the lower level "how to do it" code level drives the aileron and elevator servos to achieve the demanded bank and pitch. ArduPlane communicates with ground stations and other software using the MAVLink protocol, which can be read about on the Qgroundcontrol website.[5]

2. Laptop Ground Station

Communicating with ArduPlane can be done in several ways from a laptop. The simplest way is connecting directly to the Pixhawk using a program called Mission Planner, which can be found and downloaded on the ardupilot website. Through this program it is easy to upload firmware to the Pixhawk, tune parameters like the FBWA roll and pitch limits, create auto mode missions, and review logged data. Connections are made with a USB cable or with the 915 MHz radio. With the ODROID sending commands to the Pixhawk, however, the laptop needs to be connected to the ODROID. Using an SSH client is a good way to do this, through either an ad hoc or traditional network. Cygwin[6] is an excellent program for this if the laptop operating system is windows, with the added benefit of Xwindows as an option. Xwindows allows for the editing of code on the ODROID in a graphical environment remotely from the ground station laptop. If a traditional wireless network is used, connecting is straightforward. Simply power on the ODROID, wait a moment for it to connect automatically (configured for networks named *flightops*), and run *ping odroid* to determine the proper IP address x.x.x.x. Then run *ssh -Y odroid@x.x.x.x* with password *odroid* and the connection will be established. If running an ad hoc network, the network must be set up on the laptop. Windows provides an easy way to do this: open the control panel and go to "Network and Internet", then "Network and Sharing Center" and select "Set up new connection or network." Choose the ad hoc network option, name the network "odroid", select no security, and check the save network box. Then, each time before switching the laptop to the "odroid" network open the control panel and go to "Network and Internet", then "Manage Wireless Networks." Click "adapter properties", then select "Internet Protocol Version 4 (TCP/IPv4)" and edit the properties. Click "Use the following IP address" and change the IP address field to 10.0.0.3 and the subnet mask field to 255.0.0.0. Click ok, and connect to the "odroid" network. The ODROID is configured to connect to "odroid" with the IP 10.0.0.2, so a connection can be established by running *ssh -Y ordoid@10.0.0.2* with password *odroid*.

3. ODROID

Infrastructure software running on the ODROID consists of two parts, MAVProxy and DroneAPI. MAVProxy provides ways of encoding and decoding MAVLink messages, and can be thought of as a Mission Planner-like link to the aircraft but interfacing with the user through the command line. DroneAPI runs as a module in MAVProxy and provides a convenient object oriented way to access aircraft data and send some limited commands using Python scripts. Installation instructions for MAVProxy and DroneAPI can be found in the developer section of the ardupilot website.[2] Kevin Hester is in charge of the project, and has been tremendously helpful and enthusiastic in adding vehicle attributes when needed. MAVProxy is fairly well documented and has its own webpage.[7] It is through DroneAPI and MAVProxy that a user is able to write a custom Python script for aircraft control, and how the path following algorithm mentioned previously was implemented.

B. Infrastructure Use

A lateral and vertical path following controller was implemented to test the ODROID to Pixhawk infrastructure, and to show one way of achieving customized high level control. The goal of the path following controller was to track arbitrary curved paths using the lateral navigation described by Park, Deyst, and How[8] with a proportional and feed-forward controller on climb rate as the vertical navigation. In all, the controller determined a desired bank angle and elevator position in a python script running on the ORDOID to obtain the desired turn and climb rates. Commands were then sent via DroneAPI and MAVProxy to the Pixhawk. Airspeed hold with throttle control was also implemented, however on the ArduPlane firmware side rather than on the ODROID. Figure 6 depicts the software components of the controller and the information passed back and forth. The modified ArduPlane firmware was run in FBWA mode, so bank and climb rate commands were sent in the form of RC channel 1 and 2 override values. FBWA mode then took those to be bank and pitch as described previously, and the modified portion of the code implemented airspeed hold.

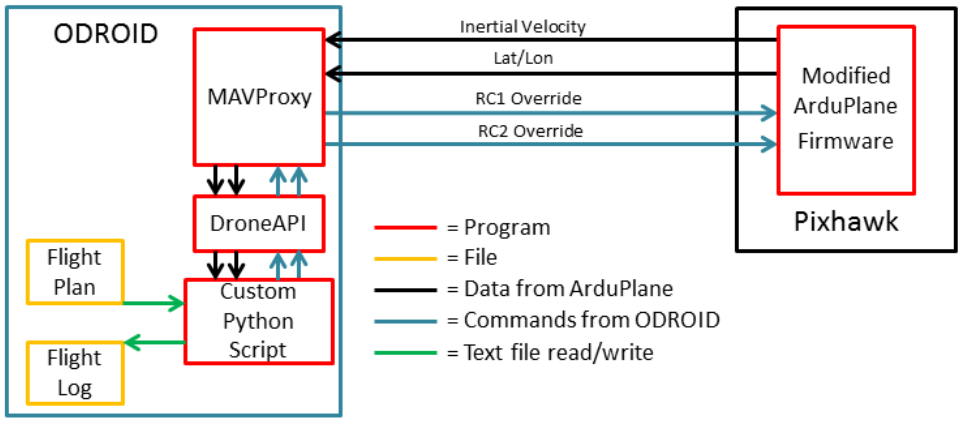

Figure 6: Path Following Controller Overview

Figure 7 displays a flow diagram of the program "Custom Python Script" in figure 6. Table 2 summarizes the changes made to ArduPlane to implement airspeed hold. In addition to the changes, the parameter TECS_SPDWEIGHT was changed from 1, meaning the speed and height controller of ArduPlane uses a combination of pitch and throttle to control airspeed, to 0, meaning that airspeed is controlled only by throttle.

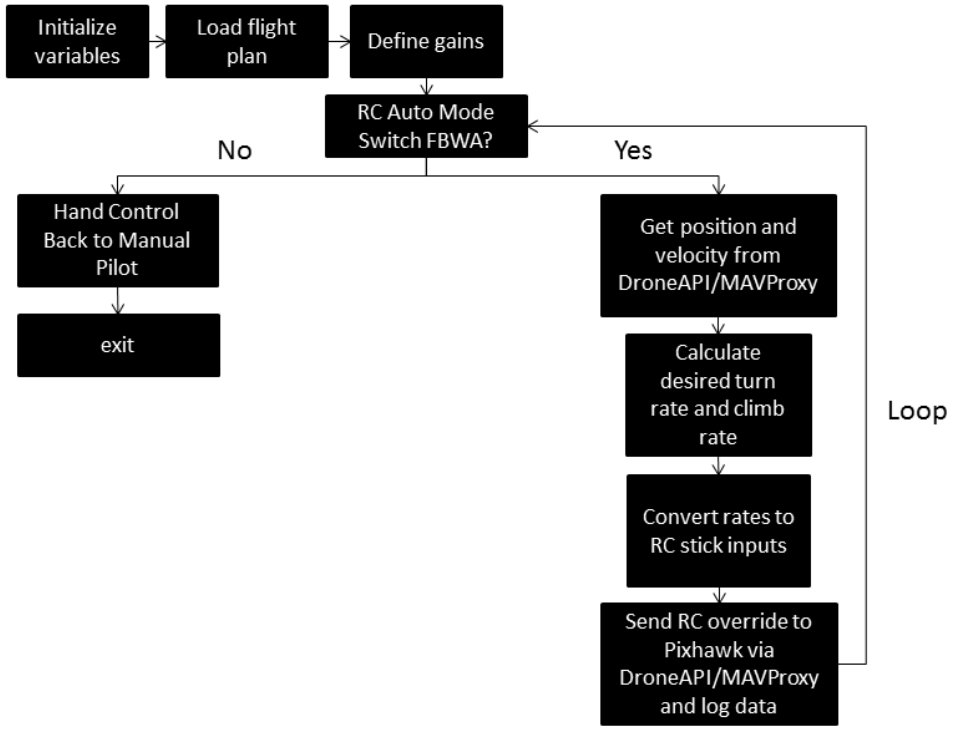

Figure 7: Python Script Flow Chart to Determine Turn/Climb Rate

Running the path following controller was very straight forward: The connection to the plane was made using an ad hoc network and SSH from a laptop running Windows 7 with Cygwin. With MAVProxy and DroneAPI started on the ground, a manual pilot performed the takeoff in manual mode. Once airborne the pilot switched into FBWA mode, starting the airspeed hold with auto-throttle and allowing the custom python script to start. Starting the script was done by running *api start Script.py*, which began lateral and vertical tracking of the flight plan.

Table 2: Summary of Modifications to ArduPlane for Airspeed Hold

Modified Function	File	Lines of Code	Purpose
update_flight_mode()	ArduPlane.pde	1	Cause speed and height controller to calculate required throttle
suppress_throttle()	attitude.pde	3	Allow throttle to be tested on the ground
failsafe_on_long_event()	events.pde	2	Change action on long failsafe to comply with COA
set_mode()	system.pde	2	Enable auto-throttle in FBWA mode

References

[1]PX4 Project, [pixhawk.org/firmware/start]

[2]APM, [ardupilot.com]

[3]3D Robotics, "Pixhawk Autopilot Quick Start Guide", March 20, 2014 [3drobotics.com/wp_content/uploads/2014/03/pixhawk-manual-rev7.pdf]

[4]DIY Drones, [diydrones.com]

[5]Qgroundcontrol, [qgroundcontrol.org/mavlink/start]

[6]Cygwin, Red Hat Inc, 2014. [www.cygwin.com]

[7]Tridgell, A., Dade, S., "MAVProxy: A UAV ground station software package for MAVLink based systems" [tridge.github.io/MAVProxy]

[8]Park, S., Deyst, J., How, J., "A New Nonlinear Guidance Logic for Trajectory Tracking," Proceedings of the AIAA Guidance, Navigation and Control Conference, Aug 2004.