Two NextGen Air Safety Tools: An ADS-B Equipped UAV and a Wake Turbulence Estimator

Ward A. Handley

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2016

Reading Committee:

Robert E. Breidenthal, Chair

Christopher W. M. Lum

Program Authorized to Offer Degree:
Aeronautics and Astronautics

University of Washington

Abstract

Two NextGen Air Safety Tools:
An ADS-B Equipped UAV and a Wake Turbulence Estimator

Ward A. Handley

Chair of the Supervisory Committee:
Prof. Robert E. Breidenthal
William E. Boeing Department of Aeronautics and Astronautics

Two air safety tools are developed in the context of the FAA's NextGen program. The first tool addresses the alarming increase in the frequency of near-collisions between manned and unmanned aircraft by equipping a common hobby class UAV with an ADS-B transponder that broadcasts its position, speed, heading and unique identification number to all local air traffic. The second tool estimates and outputs the location of dangerous wake vortex corridors in real time based on the ADS-B data collected and processed using a custom software package developed for this project. The TRansponder based Position Information System (TRAPIS) consists of data packet decoders, an aircraft database, Graphical User Interface (GUI) and the wake vortex extension application. Output from TRAPIS can be visualized in Google Earth and alleviates the problem of pilots being left to imagine where invisible wake vortex corridors are based solely on intuition or verbal warnings from ATC. The result of these two tools is the increased situational awareness, and hence safety, of human pilots in the National Airspace System (NAS).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathcal{AR}$	wing aspect ratio
ϵ	turbulence energy dissipation rate
η	dimensionless turbulence parameter
Γ	wake vortex circulation
Γ_0	initial wake vortex circulation
NS	dimensionless stratification parameter, $\frac{Nb}{V_0}$
QS	dimensionless turbulence parameter, $\frac{q}{V_0}$
τ	Crow Instability growth rate (e-folding time)
A	area of wake oval, $0.90\pi b^2$
b	wake vortex spacing, $\frac{\pi}{4}S$
C_D	wake oval viscous force coefficient
C_L	aircraft coefficient of lift
H	dimensionless wake descent distance, $\frac{z}{b}$
N	Brunt-Vaisala frequency
q	root-mean-square atmospheric turbulence parameter

S	aircraft wing span
T	dimensionless time, $\frac{tV_0}{b}$
t	time
T_L	dimensionless mean wake lifespan
U	aircraft forward speed
V	wake descent speed
V_0	initial wake descent speed
x	downstream (longitudinal) distance
y	distance along the wingspan
z	wake descent distance, positive downward

GLOSSARY

ICAO: International Civil Aviation Organization

FAA: Federal Aviation Administration

NTSB: National Transportation Safety Board

GPS: Global Positioning System

JCATI: Joint Center for Aerospace Technology Innovation

NAS: National Airspace System

UAS: Unmanned Aerial System, also known as Unmanned Aerial Vehicle (UAV), Remotely Piloted Aircraft Systems (RPAS) or "drone"

WV: wake vortices

WT: wake turbulence; the interaction between an aircraft and the wake from a preceding aircraft

ADS-B: Automatic Dependent Surveillance - Broadcast; a protocol and equipment standard for transmitting (Out) and receiving (In) local air traffic data

NEXTGEN: FAA NAS modernization program that mandates ADS-B Out equipage for most US aircraft by January 1st, 2020, among other improvements

SA: Situational Awareness

RF: Radio Frequency

RC: Radio Controlled

PWM: Pulse Width Modulation

PPM: Pulse Position Modulation

TTL: Transistor-Transistor Logic

SQUITTER: Radio frequency signal broadcast

SQUAWK: The broadcast or selection of a transponder code

GA: General Aviation; all non-commercial civil aviation

AGL: Above Ground Level

COTS: Commercial Off-The-Shelf

COA: Certificate of Authorization

SSR: Secondary Surveillance Radar

MLAT: Multilateration; localization of a target by RF triangulation

LAMS: Local Area Multilateration System

ANPC: Advanced Navigation and Positioning Corporation

TRAPIS: TRansponder based Position Information System

LIDAR: Light Detection and Ranging; similar to radar technology, but uses the visual part of the electromagnetic spectrum

HAPRA: Husky Adjustable Payload Research Aircraft

KML: Keyhole Markup Language

ACKNOWLEDGMENTS

I would like to recognize and thank the organizations, companies, and individuals who contributed to this research project. Professor Emeritus Juris Vagners and Dr. Andy von Flotow of Hood Technology Corp. originally instigated this research project. Funding was generously provided by the Joint Center for Aerospace Technology Innovation (JCATI) and administered by JCATI project manager Dr. Beth Hacker. Much guidance and in-kind support was provided by our four industry partners: Hood Technology Corp., Sagetech Corp., Advanced Navigation & Positioning Corp. (ANPC) and Insitu, Inc. Special thanks go to my committee chair Prof. Breidenthal for being open minded to this multidisciplinary endeavor, and my adviser Dr. Christopher Lum for giving me this opportunity and energetically spearheading the project - I learned a lot! Many members of the Autonomous Flight Systems Laboratory (AFSL) made important contributions to this project: LT Robert Larson, Gage Winde, Henry Qin, Ryan Valach, Emil Caga-anan, Alec Bueing, Marissa Reid, Anupam Gupta, Zach Williams, Scott An, Yifu Wang and Shida Xu. Ed Connery helped me find the teaching assistantship that funded my first year of study and Leah Panganiban helped me navigate through the MSAA program requirements. Jim Fox created and distributes this UW thesis template. Finally, the lovely Alia Westlund supported me throughout the roller coaster that is graduate school and made these years my happiest yet.

DEDICATION

to my father Bruce

it took me many years to appreciate the profound gift of napkin diagrams, weekend projects and late nights working on the Skymaster

Chapter 1
INTRODUCTION

The National Airspace System (NAS) is undergoing monumental changes: the Federal Aviation Administration (FAA) has begun implementing their NextGen NAS modernization program [18]; for the first time Unmanned Aerial Systems (UAS) are being issued N-numbers and formally integrated into the NAS [17]; and earlier this year the number of registered UAS in the NAS surpassed manned aircraft [33]. These changes significantly increase both the complexity and capability of the NAS and have profound implications for air safety. The present work leverages Automatic Dependent Surveillance - Broadcast (ADS-B) technology, one of the pillars of the NextGen project, to address two pressing air safety issues: 1) unplanned interactions between manned and unmanned aircraft stemming from a lack of situational awareness (SA) [52], and 2) the frequent and dangerous wake turbulence (WT) interactions between leading and following aircraft [22, 31].

1.1 Improving Situational Awareness

ADS-B can be utilized to dramatically increase the situational awareness of all NAS users. ADS-B is a managed peer-to-peer network where equipped aircraft squitter their 3D positions, velocities, vehicle type and identification information. Aircraft can also listen for ADS-B messages and then display local air traffic on a common tablet computer. ADS-B is already common on commercial aircraft and equipage has been mandated for most manned aircraft operating in the United States by January 1st, 2020 [10].

Currently UAS operators are required to stay under 400 feet above ground level (AGL) and "well clear" of manned aircraft operations, but given the frequency of near misses and

the first credible reports of a drone striking a passenger jet [51] many UAS operators are either unaware of these flight restrictions or utterly ignorant of the ramifications of their actions. Either way, UAS operators, pilots, and Air Traffic Control (ATC) would all benefit by being able to precisely localize and identify all man-made objects in their local airspace. This can be achieved by extending commercial off-the-shelf (COTS) ADS-B technology to UAS and is the subject of Chapter 3.

1.2 Improving Wake Turbulence Management

Chapter 4 presents the second contribution of this thesis, which develops an ADS-B based tool for visualizing, and hence avoiding, the counter-rotating wake vortices (WV) shed by all aircraft in flight. WV encounters are common and by some estimates 1 in 150 aircraft arriving or departing at Heathrow International Airport experience Wake Turbulence (WT) caused by a preceding aircraft [31]. Generally these interactions are minor, but there is a significant danger to light ($< 13600kg$) aircraft and there have even been cases where large commercial aircraft have crashed following WV encounters [4, 3].

To date the use of wake management tools has been limited to ATC and other institutional entities (e.g. the U. S. Air Force). All other NAS users are limited to referring to the FAA-specified minimum aircraft separations in terminal areas and an Advisory Circular that diagrams potential WV hazards [22]. While undeniably useful, these resources leave pilots to imagine the extent and severity of WV zones. This is insufficient. A quick search of `youtube.com` yields numerous well-documented cases where ATC fails to warn pilots of WT or pilots do not appropriately respond to WT warnings [29, 28]. The NAS-wide roll out of ADS-B creates an unprecedented opportunity to enhance WT avoidance by putting an intuitive Wake Turbulence Estimator (WTE) tool into the hands of pilots.

1.3 Scope of Work

The current thesis should be viewed in the context of a larger project funded through a Joint Center for Aerospace Technology Innovation (JCATI) grant to research applications of

NextGen technologies. The primary JCATI project objectives were:

1. Equip a UAS with ADS-B technology
2. Augment the UAS navigation system with a ground station for seamless operation in GPS denied environments
3. Acquire and display local air traffic surveillance data
4. Utilize data with a "plug-in" or 3rd party app (i.e. the Wake Turbulence Estimator)

As stated above, the scope of this thesis includes items one and four. LT Robert Larson's upcoming thesis covers items two and three [36].

Chapter 2
BACKGROUND

This chapter provides useful background regarding the NAS, ADS-B, NextGen, and the system level architecture of the TRansponder based Position Information System (TRAPIS).

2.1 The National Airspace System

The National Airspace System includes everything associated with flight: the skies above the U.S., infrastructure (airports, runways, towers, etc.), communication systems, pilot training and protocols, equipment certifications, and so on. The NAS has entered an era of unprecedented use and hence congestion. Given the recent addition of *several hundred thousand* UAS [33] and modest (2.6%) perennial growth in air carrier operations the density of aircraft in U.S. skies will continue to increase for the foreseeable future [20]. To complicate matters further the NAS was designed for manned aircraft, as were the FAA regulations put in place to ensure flight safety. This has led to a dual crisis: how to safely integrate UAS, and how to safely increase the capacity of the NAS.

2.1.1 UAS Integration

The FAA's approach to UAS has been famously fraught with difficulty. The UAS operation and airworthiness standards have largely been grafted from the manned aircraft standards. This has resulted in a series of embarrassments and the adoption of an onerous system of formal exemptions for UAS. Case in point: a popular anecdote courtesy of Paul Applewhite, founder of Applewhite Aero, is how he had to ask the FAA for a waiver so his unmanned aircraft would not have to carry an on-board flight manual despite the self evident fact that no pilot would be on-board to read it.

To further complicate matters identical airframes with identical avionics are certified differently depending on whether the user is a hobbyist, commercial entity, or a public entity. All three must register their UAS with the FAA, but the hobbyist is allowed to fly immediately, while a for-profit user must apply for a Section 333 Exemption, and a public entity can apply for Certificate Of Authorization (COA). Each authorization carries its own particular rights, responsibilities and limitations that will not be enumerated here, but suffice it to say the current regulatory environment poses a substantial hurdle to flight operations by both public and private entities. For this reason all data presented below are simulated or were obtained in surface tests.

2.1.2 NextGen

The NextGen NAS modernization initiative is the FAA's plan to safely increase the capacity and capability of the NAS. As befits the complexity and scope of the NAS, NextGen is a sweeping collection of improvements promising $133 billion in benefits for the modest investment of $29 billion from 2013 to 2030 [23]. Many of the improvements are efficiency based and pertain to commercial aviation: achieve more take-off and landings per runway per hour by using algorithms to determine optimal lineup and runway allocation; streamline surface operations; mitigate backups at a given airport by altering routes in mid-flight; etc.

2.1.3 ADS-B

ADS-B is the NextGen linchpin because it allows for more accurate flight tracking across the entirety of the United States and much of the rest of the world. At its root, ADS-B is a communications protocol and electronic hardware standard that specifies the radio frequencies, and the format in which data messages are sent [8, 5]. Most data messages are simply comprised of the unique identification, position and velocity of an aircraft so clearly the technology required is not particularly advanced: a GPS, an altimeter and an RF transceiver. Due to the challenging and rigorous certification process ADS-B installations are relatively expensive though. A typical GA aircraft owner can expect to spend $4-10,000

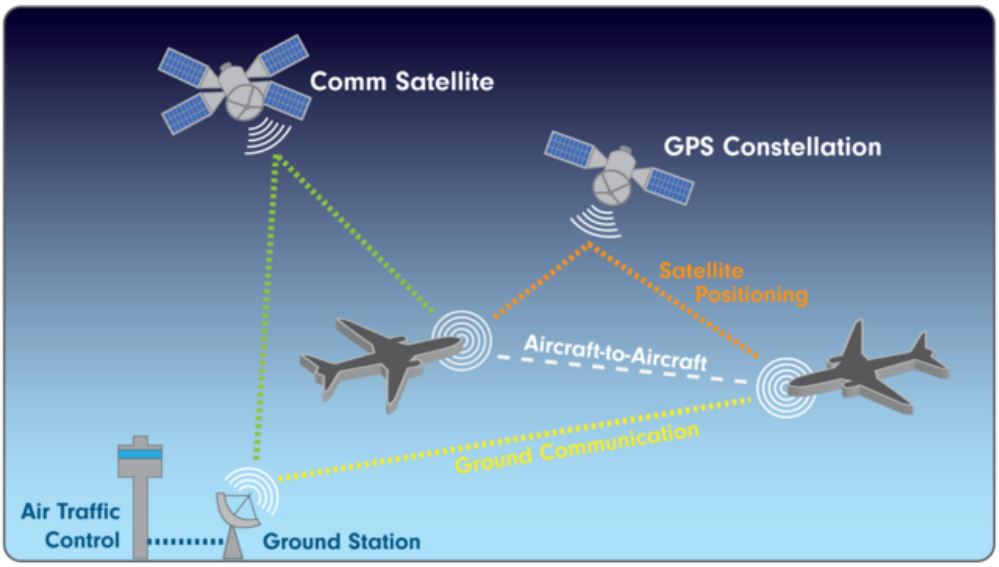

Figure 2.1: System level view of ADS-B, adopted from [13]

depending on their airframe type and if a certified WAAS GPS is already installed [38]. Perhaps for this reason alone the widespread adoption of ADS-B has been a divisive and slow process. The benefits of ADS-B are undeniable however. The ADS-B architecture allows appropriately equipped aircraft to directly send and receive surveillance data peer-to-peer, as well as through rebroadcasts managed by ATC ground stations. Pilots in the cockpit now have access to a high quality data stream giving the position, heading and velocity of cooperative local air traffic. Previously, such information was only available to ATC at well-equipped airports.

2.2 Safe Separation

One of the obvious benefits of GPS-based navigation (i.e. ADS-B) in the NAS is ATC's ability to confidently decrease the separation between aircraft[1] and hence safely increase the efficiency of busy airports. The powerful wake vortices shed by preceding aircraft can cause

[1]Naturally, aircraft separation in each of the three body-fitted coordinates (horizontal, longitudinal, and vertical) along with other factors such as speed, heading and altitude are considered. Longitudinal separation is often 10-15 minutes to allow for WV breakdown.

Leader	Follower				
	Super	Heavy	B757	Large	Small
Super	3	6	7	7	8
Heavy	3	4	5	5	6
B757	3	4	4	4	5
Large	3	3	3	3	4
Small	3	3	3	3	3

Table 2.1: Horizontal separation matrix (nm), from [22]

the loss of control or even the structural failure of following aircraft. WV breakdown is stochastic in nature and in calm atmospheric conditions WV can persist for several minutes [15, 16]. To prevent WV disasters the FAA has mandated that ATC utilize a safe separation matrix [21, 24]. The ever increasing air traffic congestion in the NAS has caused the FAA to (unofficially) backslide on its commitment to WV safety, however. This should not be surprising because slight increases in airport efficiency are immediately felt in concrete ways such as lower fares, fewer delays, greater airline fleet utilization and enormous savings on infrastructure improvements such as additional runways. The net result is that the FAA's safe separation matrix, see Table 2.1, is routinely violated based on the personal experience of air traffic controllers or the desires of ATC administrators, occasionally with tragic consequences [11]. The most notable tragedy occurred on November 12, 2001. American Airlines Flight 587 encountered WT shortly after takeoff resulting in the separation of the vertical stabilizer. All 260 passengers and crew aboard AA587 were killed, along with 5 more victims on the ground. Interestingly, the National Transportation Safety Board (NTSB) report determined the crash was instigated by WT, but faulted the co-pilot for the crash instead of recommending a review of the FAA's separation standards [4].

2.2.1 ReCAT

Clearly ad hoc modifications to the safe separation standards are undesirable so effective June 1st, 2014 the FAA enacted order 7110.659A, which updates their separation matrix

and procedure [25]. This effort is known as ReCAT and has successfully been tested at the Memphis International Airport since 2013 [21].

2.2.2 Wake Prediction and Avoidance

As a supplement to the new separation matrix ATC has access to advanced WV prediction tools like the Aircraft Vortex Spacing System (AVOSS) and Wake Vortex Advisory System (WakeVOS) [9][41]. AVOSS and WakeVOS are incredibly useful systems, but as noted by Guerin [31], "There are few, if any, technological tools to measure and display in real time the strength of a preceding aircraft's wake turbulence and its probable consequence on a particular aircraft type operating behind."

Furthermore there is a unidirectional information bottle neck between ATC and pilots with only verbal WT warnings reaching the pilots. Since 2007 when the FAA began to allow Electronic Flight Bags (EFBs) [19] tablet computers have become commonplace in the cockpit. This phenomenon paired with the rise in ADS-B equipage presents a novel opportunity to effectively tackle the WT information problem: build a simple and conservative Wake Turbulence Estimator (WTE) that processes surveillance data in real-time to predict and visualize wake vortex corridors. Pilots could then make more informed decisions about how best to respond to a verbal wake turbulence warning from ATC. Earlier works on this subject exist, but were severely hampered by the absence of reliable local air traffic data [34, 32], or required specialized on-board sensors such as LIDAR [43].

2.3 System Architecture

This thesis exists within the larger context of a JCATI-funded project, and is in cooperation with four industry partners: Hood Technology Corp., Sagetech Corp., Advanced Navigation & Positioning Corp. (ANPC) and Insitu, Inc. The architecture pictured in Figure 2.2 was conceived to meet the objectives enumerated in Section 1.3. The UAS ADS-B payload is described in Chapter 3. In the event of GPS denial due to jamming or malfunction then primary UAS localization will shift to the ANPC Local Area Multilateration

System (LAMS) and all stake holders will maintain situational awareness. See Figure 2.2. Detailed information on the system when GPS is denied is available in LT Robert Larson's forthcoming thesis [36].

2.3.1 TRAPIS

The TRansponder based Position Information System (TRAPIS) is the software cornerstone of this project. TRAPIS was developed using Microsoft Visual Studio, is written in the C# programming language and is thoroughly vetted using a large body of unit tests. There are also realistic simulators developed by our industry partners for both the Clarity (Sagetech) and LAMS (ANPC) data streams to assist with development and testing. Additional simulation scenarios and tools such as data replay were developed in-house.

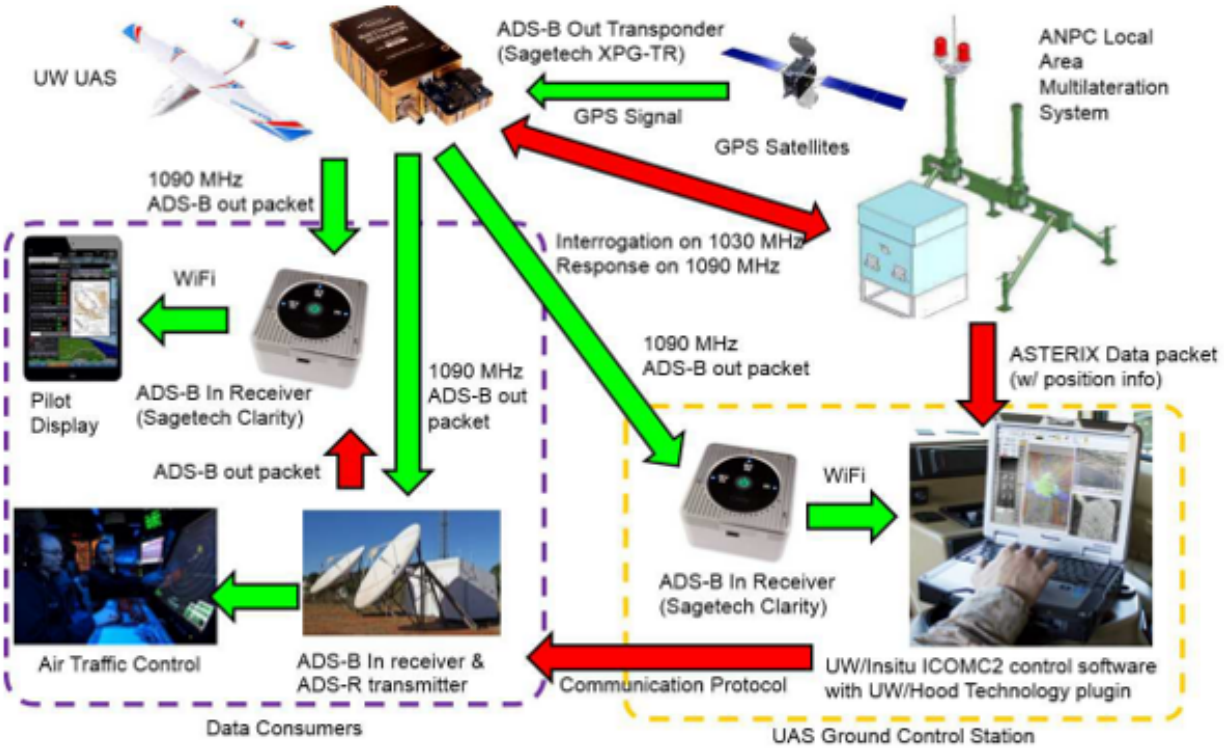

Figure 2.2: Architecture for nominal (green) and GPS-denied (red) operations

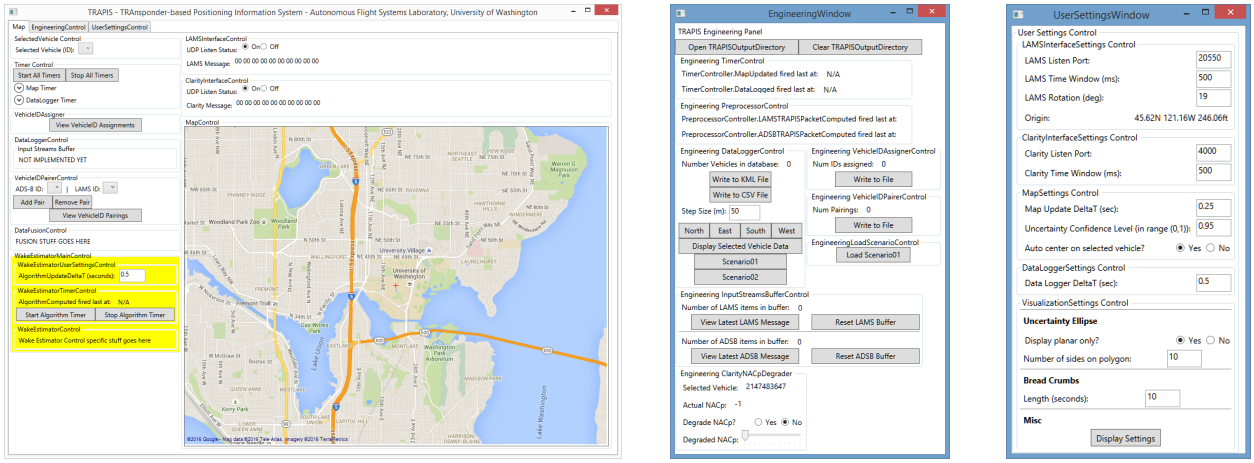

Figure 2.3: TRAPIS GUI and controls

The essential functions of TRAPIS are manifold: ingesting surveillance data from the ANPC LAMS and ADS-B via the Sagetech Clarity receiver, data handling (buffering, transfer, storage, display), data processing using a custom-built Kalman Filter, and a graphical user interface (GUI). See Figure 2.3 for screenshots of the TRAPIS front end.

The TRAPIS GUI intuitively displays local air traffic data using the `GMap.NET` framework [42]. Multiple position measurements for each aircraft are visible due to the dual input data streams. After pairing two data streams TRAPIS then fuses the position measurements into an enhanced position estimate. Again, the details of this process are detailed in [36]. Finally, TRAPIS passes the position, velocity, and aircraft wake category data to the Wake Turbulence Estimator that is the subject of Chapter 4.

Chapter 3

ADS-B AVIONICS PAYLOAD FOR UAS

This chapter details the custom ADS-B avionics payload developed for use with a Pixhawk UAV autopilot [2]. The Pixhawk is a sophisticated, but popular and economical COTS autopilot that is used on unmanned rotorcraft, fixed wing aircraft and ground vehicles. The Pixhawk itself is simply a hardware platform that typically runs ArduPilot software [1]. The Pixhawk and ArduPilot coordinate all of the UAS infrastructure: RC control signals, telemetry from GPS and inertial measurement units, actuation of the control surfaces, motor throttle, and autonomous functionality such as waypoint-based navigation and autolanding.

3.1 ADS-B Transponder

3.1.1 1090MHz ES

The ADS-B transponder used for this project is a Sagetech model XPS-TR, which is shown in Figure 3.1. ADS-B has grown out of the legacy Mode A, C and S transponders that already operate at 1090MHz, but utilizes "Extended Squitter" to transmit much more information than just squawk code and altitude [13]. To maintain backwards compatibility the XPS-TR is an ADS-B Out transponder that broadcasts at the international standard 1090MHz, but also responds to Mode S interrogations at 1030MHz [6, 5]. Common interrogators include the LAMS, Secondary Surveillance Radar (SSR) and Traffic Collision Avoidance Systems (TCAS) on other aircraft. Electrical specifications for the XPS-TR, connections depicted in Figure 3.3, power output shown in Table 3.2, and other configuration settings can be found in the XPS-TR User Guide [45].

Figure 3.1: Sagetech XPS-TR

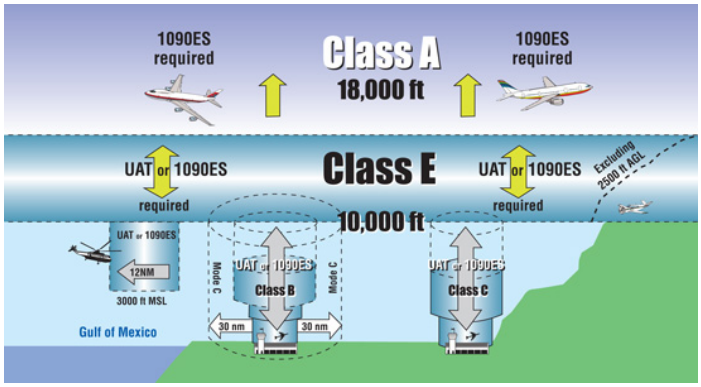

Figure 3.2: ADS-B airspace requirements from [13]

3.1.2 978MHz UAT

To prevent saturation of the 1090MHz band in the NAS Universal Access Transceivers (UAT) are also used for ADS-B. UAT transcievers operate at 978MHz and are specific to the United States. They are largely targeted at GA users because they are limited to use below 18,000 feet [13] and they have the attractive additional capability to receive Traffic Information Service - Broadcast (TIS-B) and Flight Information Service - Broadcast (FIS-B) messages [7, 44]. Figure 3.2 depicts where and what type of ADS-B Out transponder will be required for manned aircraft after January 1st, 2020 [10].

3.2 Autopilot Integration

As shown in Figure 3.3 a Pixhawk autopilot determines the 3D position and velocity of the UAS using GPS and an on-board inertial measurement unit, which is then translated and relayed to the XPS-TR via an Arduino Mega2560 board. It is technically feasible to modify the Pixhawk ArduPilot software to format and output messages directly to the XPS-TR, but ArduPilot is a large flight critical code base so much care would have to be taken to ensure that the customizations would not adversely affect the functionality of the *entire* code base. Therefore, for this initial project all customizations were segregated from the ArduPilot code and implemented on an Arduino Mega2560 board.

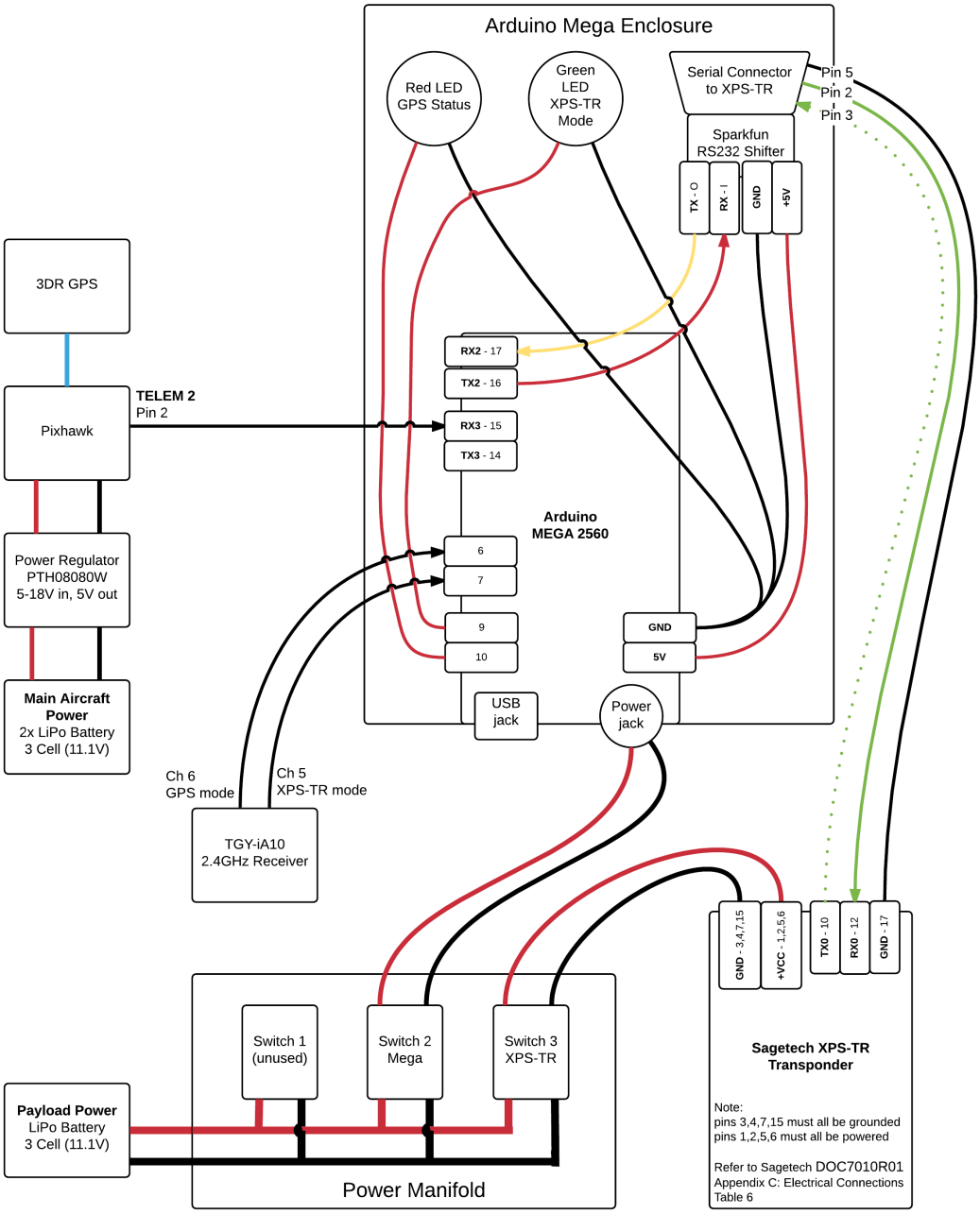

Figure 3.3: TRAPIS avionics wiring diagram

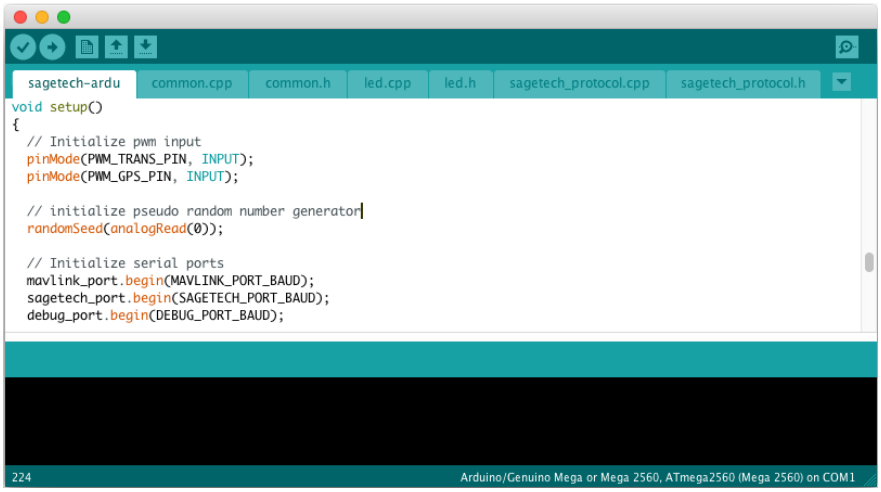

Figure 3.4: Arduino integrated development environment

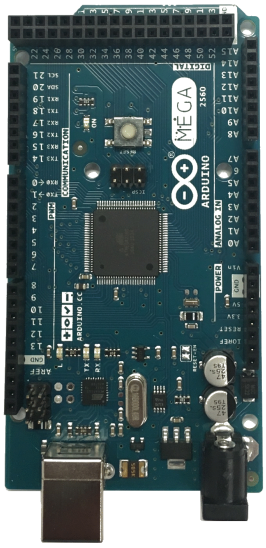

Figure 3.5: Mega2560

3.2.1 Arduino Mega 2560

The Arduino Mega2560 is a relatively powerful stand alone computer with a processor, nonvolatile memory, numerous analog and digital inputs, and programmable outputs. The Arduino is programmed in C++ using the integrated development environment shown in Figure 3.4. The master file is known as a "sketch," which contains three types of code:

1. precompiler commands
2. initialization code that is run once when the board powers up
3. loop code that is iteratively executed so long as the board is powered

For example: the precompiler commands can be used to load extensive external libraries and define global variables, then the initialization code runs once to setup a serial communications port, or blink and LED to signal startup, and finally the loop code repeatedly executes a sequence of tasks like measuring the voltage on a pin and then sending a serial message containing the most recent voltage reading. The initialization and loop code can call linked

C++ files containing subfunctions, communications protocols, etc. The Arduino sketch used in this project is located in Appendix B and is based on prior work by SPH Engineering [46] that is distributed under a very permissive copyright. Also, Sagetech Corp. has granted permission to publish the small subset of their proprietary communications protocol that appears in the Arduino sketch in Appendix B. See Appendix A for the relevant licenses and permissions.

3.3 Testing and Verification

3.3.1 Hardware-in-the-Loop Simulator

Initially a Hardware-in-the-Loop (HiL) simulator was used for bench top and "walking" tests of the ADS-B payload. The HiL contains all of the electrical components found in a fully functional UAS (Pixhawk, RF receivers, GPS, motor, ailerons, etc.) laid out flat for ease of reconfiguration and modification. See Figure 3.6. This facilitated rapid code iteration and troubleshooting while still in the laboratory.

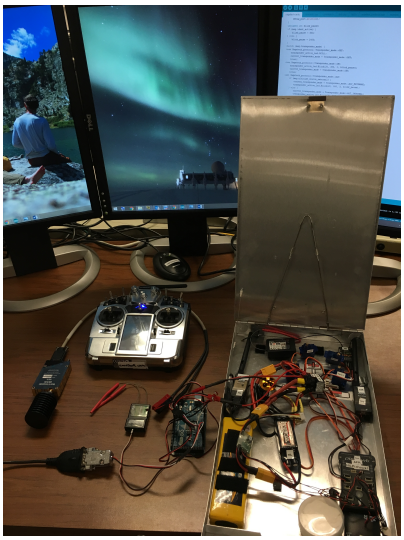

Figure 3.6: HiL simulator

Figure 3.7: HAPRA with ADS-B payload

Date	Location	Outcome	Comment
8/20/2015	The Dalles	success	ADS-B equipped manned aircraft demo
1/15/2016	Drumheller Fountain	success	HiL setup
2/6/2016	Meadowbrook Farm	failure	HiL setup; EMI and software issues
3/17/2016	Drumheller Fountain	success	payload integrated with UAS
3/19/2016	Meadowbrook Farm	failure	integrated with UAS; EMI and software issues
4/1/2016	Crescent Lake	success	HiL setup
4/2/2016	Meadowbrook Farm	success	HiL setup with EMI shielding
4/6/2016	Drumheller Fountain	success	LAMS simulator and Clarity with TRAPIS
4/16/2016	Meadowbrook Farm	success	payload integrated with UAS, LAMS simulator and Clarity with TRAPIS, car mounted
5/6/2016	The Dalles (KDLS)	success	full system test: payload integrated with UAS, LAMS and Clarity with TRAPIS, car mounted

Table 3.1: Avionics field testing regime

3.3.2 Field Testing

Extensive field tests were conducted with the ADS-B payload in successively more complicated configurations culminating with integration into a fully functional UAS dubbed the Husky Adjustable Payload Research Aircraft (HAPRA) shown in Figure 3.7. See Table 3.1 for a full list of field tests and their outcome. HAPRA is a COTS Skywalker X8 flying wing popular with hobbyists and researchers around the world for its large payload bay and long (30 minute) flight times. The HAPRA has been formally registered with the FAA as N676ZL and the unique hexidecimal ICAO identifier A8F44C [27], although the N-number occasionally used for ground tests was TEST1234 as shown in Figure 3.9. Figure 3.9 is an important result because it demonstrates that the HAPRA UAS can be identified and located by any pilot with the following common cockpit tools: an ADS-B In receiver such as the Clarity and a tablet computer such as an iPad running the WingX Pro app. The squawk code 1253 was used to easily distinguish HAPRA from GA aircraft flying VFR (squawk 1200).

Figure 3.8: HAPRA mounted for a driving test on April 16th at Meadowbrook Farm

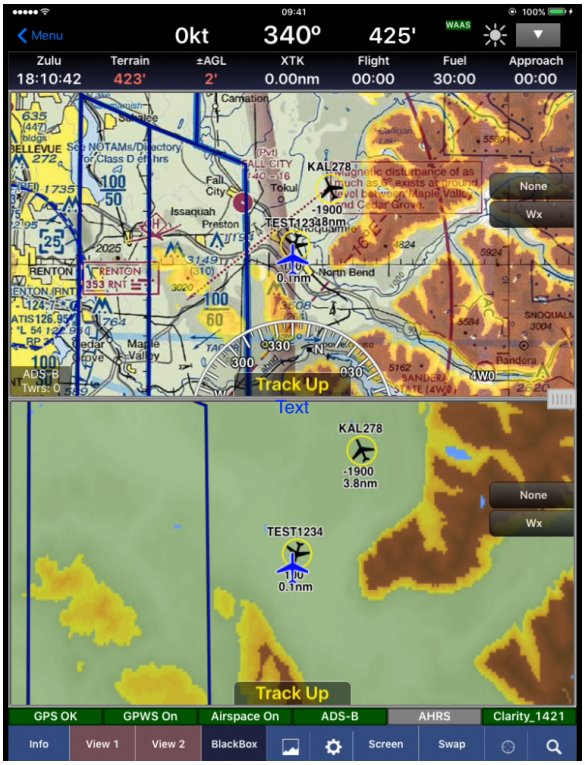

Figure 3.9: WingX Pro screen capture showing ADS-B data transmitted by HAPRA

3.3.3 Regulatory Hurdles

As discussed in Section 2.1.1, the FAA certification process is long, slow and often non-intuitive. This posed an insurmountable legal obstacle to executing live flight operations so field testing was limited to ground-based "walking" or "driving" tests. Ironically it would have been perfectly legal for a private citizen "hobbyist" to execute identical flights, whereas public and commercial entities must obtain an exemption or authorization from the FAA.

The FAA's interpretation of its own rules and the scope of exemptions it grants are rapidly changing. On May 4th, 2016 the FAA issued a memorandum relaxing the prohibitions on students flying UAS for educational purposes [26]. The memorandum clarifies that flying UAS for research and development purposes such as this project still requires a COA however. Fortunately in April the FAA began to offer "blanket" COAs to educational institutions. As

the name suggests, a blanket COA is much more permissive and allows operation of all small (< 55 lbs) university-owned UAS below 400' AGL and within class G airspace. There are significant reporting requirements however, including issuing a Notice to Airmen (NOTAM) between 24 and 72 hours before missions and reporting all accidents and mishaps to the FAA. As a result of an application submitted by the Autonomous Flight System Laboratory the University of Washington was issued a blanket COA on May 6th, 2016. Under this COA HAPRA flight tests are planned for the summer of 2016.

3.3.4 Challenges Overcome

Electromagnetic Interference

Most manned aircraft are aluminum skinned and act as Faraday cages that protect their avionics from Electromagnetic Interference (EMI). With few exceptions manned aircraft are also significantly larger than UAS, allowing for the physical separation of emitting components such as antennas. In contrast HAPRA's Skywalker X8 airframe is constructed with RF transparent Styrofoam and maximum antenna separations are limited to about 18 inches. The XPS-TR ADS-B transponder was designed for manned aircraft and its 250 Watt power output is so high that its useful range is 120 nautical miles [45]. As shown in Table 3.2, the power level of the transponder and LAMS is three orders of magnitude higher than the other transmitters and receivers onboard HAPRA, and an astounding 19 orders of magnitude higher than GPS signals received on the surface of the Earth. Even though each antenna

Component	Purpose	Frequency (MHz)	Power (W)
C2 link	ground station uplink	915	0.1
Transmitter	radio remote control	2400	0.1
XPS-TR	ADS-B Out	1090	250 (pulsed)
LAMS	Mode S interrogation	1030	500 (pulsed)
GPS	position and timing	1575 (L1), 1227 (L2)	1E-16 (on surface of Earth)

Table 3.2: RF signal strength

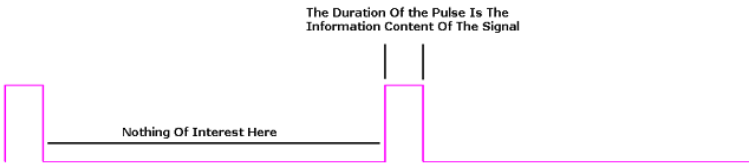

Figure 3.10: PWM signal content, adopted from [50]

is sized for its particular design frequency and has low gain outside of that band, the very high power of the ADS-B transponder can still saturate receivers. As discovered during the May 6th field test at The Dalles the same is true for operations near to the 500 Watt LAMS interrogator.

EMI mitigation is straightforward in theory, if still difficult and frustrating in practice. First, simply separating antennas helps because RF power is diminished as $1/r^2$. I.e. the power emitted by an omni-directional antenna will be spread over the surface of a larger sphere. This approach was successfully employed on May 6th to eliminate EMI from the LAMS - we simply drove the UAS several nautical miles away from the LAMS. Second, the ADS-B antenna cannot be moved far away from other electronic so metal shielding must be used to prevent EMI from interrupting HAPRA's other avionics. To this end the enclosure housing the Arduino Mega2560 is lined with metal, signal cables are shielded, and the omni-directional antenna is mounted to a flat plate that separates it from the payload bay.

PWM Signals and Timing

A large part of the reason why the HAPRA avionics are susceptible to EMI is because the RC receiver encodes all control input as Pulse Width Modulation (PWM) signals [2]. Figure 3.10 shows a single PWM pulse where the duration of that pulse contains the information content of the signal [50]. The case-specific interpretation of the PWM duration is defined by the user and Figure 3.11 shows how a long duration (2ms) PWM signal could equate to full throttle, whereas a short (1ms) PWM signal could equate to a full break or reverse

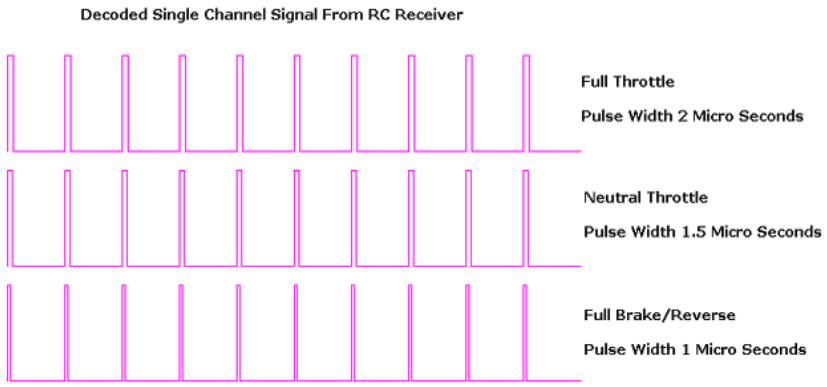

Figure 3.11: PWM signal train, adopted from [50]

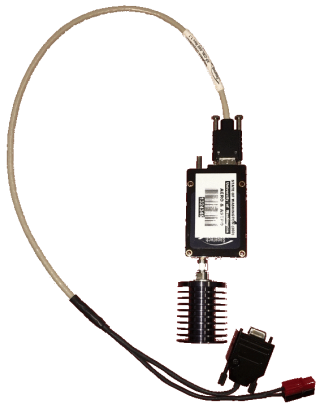

Figure 3.12: XPS-TR with resistive load

command [50]. The same PWM signal train could set the aileron deflection or the XPS-TR transmit mode.

PWM is a digital signal and the rising and falling edges of the pulse are used to trigger (start or stop) the Arduino Mega2560 timers monitoring the RC signals that control the XPS-TR mode. As should be expected, when a 250 Watt RF signal is emitted very near the Arduino, then the PWM input can be saturated - resulting an erroneously long PWM measurement, or the PWM input can be interrupted - resulting in an erroneously short PWM measurement.

Geographic Limitations

An additional logistical challenge to testing the ADS-B system at the University of Washington is that the UW campus lies directly beneath the North arrivals portal into SeaTac International Airport. The use of an incorrect altitude mode could result in TCAS warnings and the emergency diversion of commercial aircraft with hundreds of people on board. To prevent inadvertent broadcast errors all tests at UW were conducted using a resistive load in place of the ADS-B monopole antenna as shown in Figure 3.12. Simply powering on the XPS-TR without either an antenna or load could damage it and is therefore inadvisable [45].

The load is a 50 Ohm RF terminator with a maximum sustained power rating of 20 Watts. The XPS-TR power output specified in Table 3.2 is 250 Watts, but is pulsed so the time averaged output is only 5-10 Watts [45]. Interestingly, the broadcast power of the XPS-TR is so high that despite the RF inefficiency of the resistive load it *still* transmits ADS-B messages with a useful range of approximately 50m, which is actually quite advantageous for "live" ground tests on the UW campus.

The off-campus field test sites at Meadowbrook Farm, Crescent Lake, and The Dalles were all carefully selected to stay outside of Class B airspace (see Figure 3.2) and adhere to the UAS airspace restrictions listed in Table 3.3.

3.4 Results

3.4.1 System Level Demonstration of Functionality at Meadowbrook Farm

During the 4/16/2016 trip to Meadowbrook Farm the functionality of the complete TRAPIS system was demonstrated. The ADS-B avionics payload was integrated into the HAPRA UAS, which was then mounted to a vehicle (see Figure 3.8) and driven at realistic flight speeds for a short distance (1/2 mile). Throughout the test HAPRA transmitted surveillance data in real time to the Clarity ADS-B receiver, which then passed the data to the TRAPIS software suite for processing and display. Figure 3.13 shows the Keyhole Markup Language (KML) output from TRAPIS displayed natively in Google Earth [39].

Distance	Operational Area
5 nm	from an airport having an operational control tower
3 nm	from an airport having a published instrument flight procedure, but not having an operational control tower
2 nm	from an airport not having a published instrument flight procedure or an operational control tower
2 nm	from a heliport

Table 3.3: UAS airspace restrictions

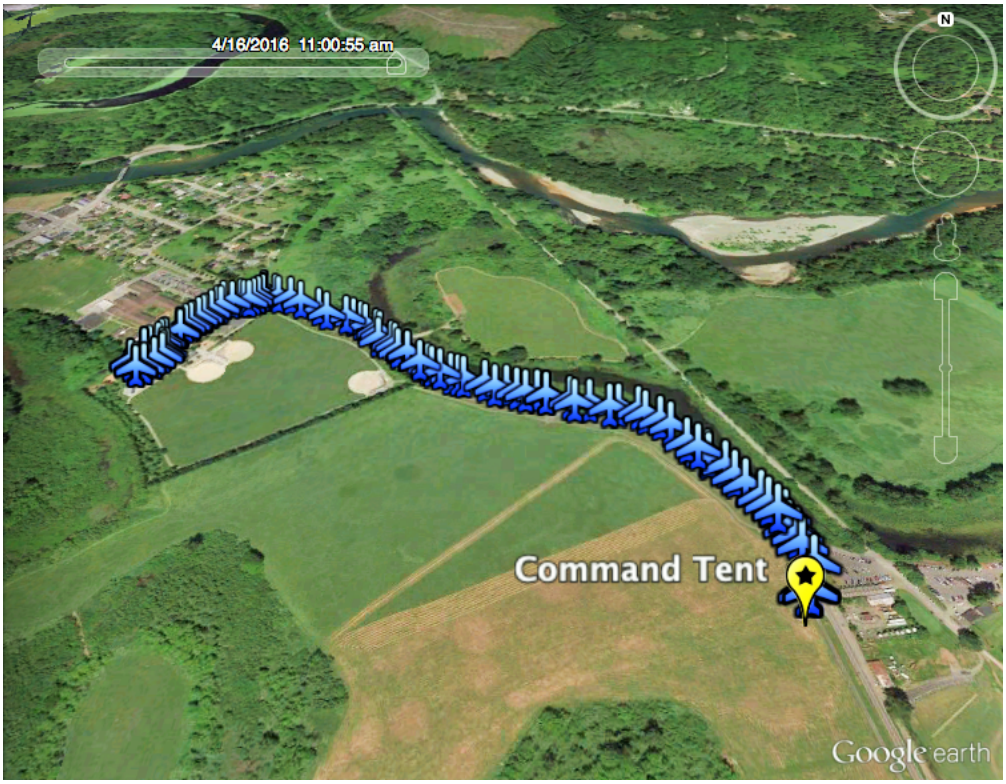

Figure 3.13: Test data from 4/16/2016 showing short distance ADS-B payload functionality

During the same test Figure 3.9 was captured. As discussed in Section 3.3.2 this is an important result because it demonstrates that *anyone* within range who is monitoring ADS-B transmissions using COTS hardware and software can uniquely identify and locate HAPRA.

3.4.2 System Level Demonstration of Functionality at The Dalles

A more ambitious field test took place on 5/6/2016 near The Dalles, OR. Once again HAPRA was car mounted, but was taken over 5.5 miles away from the command station. During this test the LAMS installation located at the KDLS airport was also used to interrogate the XPS-TR transponder at 1030MHz and received Mode S responses at 1090MHz. The LAMS independently identified and localized HAPRA using only the Mode S replies

Figure 3.14: Test data from 5/6/2016 showing long distance ADS-B payload functionality

while the Sagetech Clarity simultaneously received full ADS-B messages from HAPRA. Both data streams were ingested by TRAPIS and Figure 3.14 was generated with the `.kml` output. Figure 3.15 is a detail view of the driving route executed during this test. Unfortunately, a longer route was not feasible due to topography limitations on direct line-of-sight with the command station 5.5 miles away.

With two independent sources of position data coming from the LAMS and the Clarity additional test runs were conducted where HAPRA was accurately identified and localized despite a degraded or denied GPS signal compromising nominal ADS-B functionality. This demonstrated that even in challenging environments (e.g. tactical or urban canyon) the ADS-B payload could reliably be used to identify, locate, and, if necessary, deconflict manned and unmanned aircraft.

Figure 3.15: Detail view of the 5/6/2016 driving route

Chapter 4
WAKE TURBULENCE ESTIMATOR

Once TRAPIS could collect, store, process and display ADS-B data the obvious extension was to generate a valuable data product for pilots: the Wake Turbulence Estimator (WTE). The idea is simply to help pilots visualize likely wake vortex corridors and display those corridors in an intuitive way on a pilot's tablet computer.

4.1 Wake Vortex Models

4.1.1 Limitations

Unfortunately, from a fluid dynamics standpoint accurate calculation of wake vortex evolution is extremely challenging. In the words of the great Philippe Spalart [47], "a theory giving the trajectory and lifespan of the vortices behind an airplane in a given weight class to within - 30%, for instance, would be very impressive." Currently the most successful models are 3D LES or hybrid κ-ϵ RANS, which are extremely useful from a research perspective, but are highly idealized (e.g. constant boundary conditions, simplified airplane geometry) and require substantial computing resources such as clusters or a super computer. So even if such a theory were known it is hard to imagine that it could be implemented as a real-time wake management tool available to ATC, much less pilots in the cockpit.

Furthermore, only a few of many parameters are likely to be known with any degree of confidence such as the position, velocity, wingspan, and a rough weight estimate of an aircraft. Whereas critical model inputs such as wind profile, flap configuration, angle of attack, atmospheric turbulence and atmospheric stratification can only be grossly approximated or neglected altogether. Given these limitations it is reasonable to calculate conservative estimates for WV corridors using a simple WV model.

4.1.2 Assumptions

The WV models presented below assume the following:

- Elliptically loaded wing (i.e. uniform downwash)
- Incompressible, laminar, and inviscid flow
- Vortex cores are viscous and can be turbulent, in which case they are modeled with an enhanced turbulence viscosity
- Wake is a perturbation to the free stream flow ($\bar{u} = U + \bar{u}'$)
- 2D cross sectional dynamics dominate the longitudinal dynamics
- Free stream atmosphere has a small amount of background turbulence and is uniformly stratified

4.1.3 Historical Progression

Wake vortices exhibit a frustrating diversity of behavior. Early works describe the deterministic decay of wakes as the counter rotating vortices grew due to viscous effects and eventually contacted one another [48, 49]. By the 1970's however, it had become clear that wake vortices persisted for long periods in stable atmospheric conditions and that their destruction was stochastic and due to longitudinal instabilities [15, 16]. In 1986 Greene advanced an empirical model that combined elements of Crow & Bate's Stochastic Collapse model with earlier Predictable Decay models [30]. In 1998 Kantha extended Greene's model to include ground effect and cross wind [35]. Unfortunately there has been little theoretical progress since Greene and Kantha even as numerical simulations yield higher fidelity flow fields of particular situations [47, 11, 41, 31].

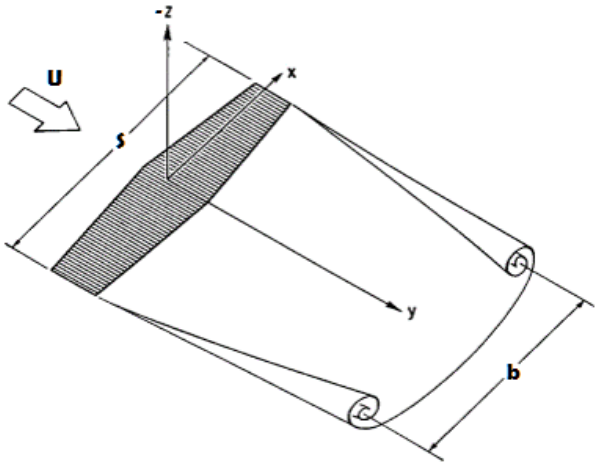

Figure 4.1: Vortex sheet roll up, adapted from [48]

4.1.4 Lifting Line Theory and the Finite Wing

Prandtl's Lifting Line Theory is an appropriate starting point when discussing WV models because it was the first successful theory of a finite wing [40]. Prandtl's insight was to model the wing as the superposition of flat horse-shoe vortex lines, thereby relating the lift of the wing to the circulation and the downwash velocity of the wake.

$$Lift = \rho U \int_S \Gamma(x)\, dx$$

Where $\Gamma(x)$ is a function of the wing cross section geometry and S is the wingspan.

Vortex Sheet Roll Up

While Prandtl's theory was very successful for calculating aerodynamic properties of real wings, it did not describe the known roll up of the vortex sheet behind the aircraft. Recognizing this, Spreiter & Sacks (1951) made the first concerted effort to describe the roll-up and descent of wakes in a constant density fluid [48]. Their main contribution was to show that while the wake initially descends at

$$V_0 = \frac{\Gamma_0}{S} \tag{4.1}$$

(recall that $+z$ is defined down), the velocity of the center of gravity of the vortices just behind the trailing edge reduces to

$$V_{T.E.} = \left(1 - \frac{\pi}{4}\right) \frac{\Gamma_0}{S} = 0.215 \frac{\Gamma_0}{S} \tag{4.2}$$

and ultimately declines to

$$V_\infty = \frac{2}{\pi^2} \frac{\Gamma_0}{S} = 0.203 \frac{\Gamma_0}{S}$$

far behind the wing. In a uniformly stratified fluid V_∞ will decrease to zero due to buoyancy as discussed below.

4.1.5 Predictable Decay

Around the same time Squire laid the groundwork for the Predictable Decay of WV due to viscous diffusion [49]. In his 1954 "back of the envelope" calculation that was widely circulated, but not published until 1965, he treats the turbulent flow in the cores of the WV as having the enhanced viscosity $\nu_{effective} = \nu + a\Gamma$. Noting that

$$\Gamma = \frac{2S}{\pi \mathcal{AR}} C_L U = \frac{L}{\rho U b} \tag{4.3}$$

and making the assumption that $\nu \ll a\Gamma$, the vortex cores grow as

$$\sqrt{\nu_{effective} t} \sim \sqrt{x}$$

and will touch at a distance

$$\frac{x}{S} = 0.02 \frac{\mathcal{AR}}{a C_L}$$

Given reasonable aerodynamic properties $C_L = 0.5$, $\mathcal{AR} = 5$ and guessing $a = 10^{-3}$ we have:

$$\frac{x}{S} \approx 200 \tag{4.4}$$

Applying Equation 4.4 to an A380-800 we have $x \approx 10$ miles, which corresponds to a 4 minute separation at approach speeds. *This is suspiciously (or shockingly) close to the FAA's mandated separation time.* Given that this calculation dates to the fifties Spalart's

observation is apt [47]: "A concerted scientific effort towards an accepted and rational process by which to set the distances appears very worthwhile, but 30 years of good scientific work have had little impact on ATC practice."

Near Field Evolution

As shown by Cotel & Breidenthal [14] the growth rate of the vortex cores in the near field (approximately the first 5 seconds after the aircraft passes) proportional to $Re^{-1/2}$ and therefore astonishingly small as shown below. Essentially the interface between the vortex and the free stream is so highly stratified that it is very smooth and there is little entrainment of free stream fluid or transfer of momentum.

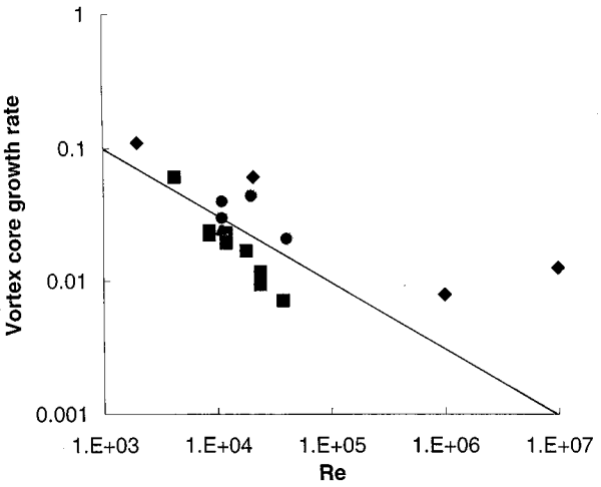

Figure 4.2: Adopted from [14]

Take the Airbus A380-800 in cruise as an example: Using Spalart's [47] +11% correction of Spreiter & Sacks' [48] formula #17 for the vortex core radius: $r_c = 0.086S \approx 7m$. The initial circulation calculated using Equation 4.3 is $\Gamma_0 = 730^{m^2}/_s$ (see Figure 4.10) and given a wingspan of 80m the initial wake descent velocity from Equation 4.1 is $V_0 = {\Gamma_0}/_S \approx 9^m/_s$. At an altitude of $33,000ft$ $\rho = 0.41^{kg}/_{m^3}$ and $\mu = 1.46 \times 10^{-5 Ns}/_{m^2}$.

$$Re = \frac{\rho V_0 (2r_c)}{\mu} \tag{4.5}$$

Applying Equation 4.5 the Reynolds number of the descending vortex is initially on the order of 4×10^6 and the growth rate of the vortex core is quite small according to Figure 4.2. As the vortex descends it slows, thereby lowering the Reynolds number and increasing the growth rate.

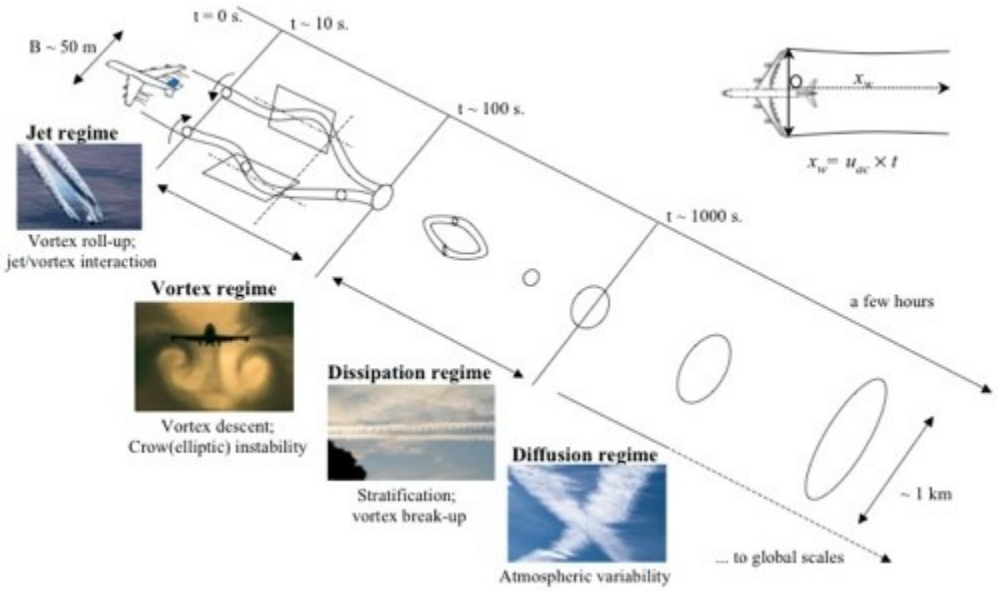

Figure 4.3: Evolution of aircraft wakes, adopted from `www.alcf.anl.gov`

4.1.6 Stochastic Collapse

The first line of Crow's seminal 1970 paper [15] was an unequivocal attack on the prevailing Predictable Decay models: "Trailing vortices do not decay by simple diffusion." Instead he proposed a theory for the (nearly) sinusoidal instability that can lead to vortex destruction and bears his name. Crow argued that observed vortex destruction was faster than predicted by viscous decay models, and that the shorter life times could be attributed to the Crow Instability that caused the two wakes to reconnect and form rings. As with most vortical instabilities, slight perturbations are amplified as they further perturb the flow via the Biot-Savart Law, which then causes yet more perturbations. The longitudinal wavelength of the Crow Instability is

$$\lambda \approx 6.8S$$

which grows with the e-folding time:

$$e^{\frac{t}{\tau}},\ \tau = 7.38\frac{ÆS}{C_L V_0}$$

Crow & Bate extended Crow's original work to include the effects of atmospheric turbulence and meteorological conditions on the statistical lifetime of wakes [16]. They looked at the effect of the dimensionless turbulence parameter

$$\eta = \frac{(\epsilon b)^{1/3}}{\Gamma/2\pi b}$$

on the dimensionless mean wake lifespan T_L and created a piecewise function T_L by taking the limit where η is small and large. The conservative limit when η is small is of the most interest for aircraft safety and can be expressed as

$$T_L = \frac{1.6}{QS} \tag{4.6}$$

where QS is a dimensionless turbulence parameter related to η and falls within the range specified in Table 4.1.

4.2 Wake Zone

As noted earlier, applying the above models to real wakes is fraught with difficulty because many parameters such as QS are crudely estimated, and even if all the parameters were known the resultant mean lifetime would still be very difficult to translate into ATC actions:

"...such an average is meaningless in the field of safety. The probability that the strength is still over the acceptable level is meaningful. However, the time for that probability to reach useful values, such as 10^{-6}, is both too long to explain the current [FAA separation] matrix entries and extremely difficult to calculate. It appears that (unless maybe the following aircraft is heavier than the leading one) safety is obtained by avoiding encounters through flight-path control, rather than by flying so far back that no active wake could ever be encountered." [47]

The above quote is the inspiration for my desire to give pilots in the cockpit a visual tool to assess the WT environment around their aircraft. A conservative approach would be to

estimate a Wake Zone (WZ) that conservatively contains the stochastic WV. A reasonable model for calculating such a region is given by Greene (1986) and includes Crow's work as well as the effects of buoyancy in a stably stratified atmosphere [30].

$$\frac{d^2H}{dT^2} + \frac{C_D L}{4\pi b}\left(\frac{dH}{dT}\right)^2 + 0.82QS\left(\frac{dH}{dT}\right) + \frac{ANS^2}{2\pi b^2}H = 0 \tag{4.7}$$

Where the non-dimensional variables H, T, and $^{dH}/_{dT}$ are defined as follows:

$$T \equiv \frac{V_0}{b}t \tag{4.8}$$

$$H \equiv \frac{z}{b} \tag{4.9}$$

$$\frac{dH}{dT} \equiv \frac{V}{V_0} = \frac{\Gamma}{\Gamma_0} \tag{4.10}$$

4.2.1 Atmospheric Stratification

Greene's equation is analogous to the damped harmonic oscillator with a quadratic drag term:

$$\frac{d^2x}{dt^2} + D\left(\frac{dx}{dt}\right)^2 + 2\zeta\omega_0\left(\frac{dx}{dt}\right) + {\omega_0}^2 x = 0$$

By analogy the natural frequency ω_0 of Equation 4.7 is then:

$${\omega_0}^2 = \frac{ANS^2}{2\pi b^2} = \frac{A}{2\pi {V_0}^2}N^2$$

$$\boxed{\omega_0 \propto N}$$

N is the Brunt-Vaisala Frequency, which is the frequency at which a parcel of fluid displaced in a stably stratified fluid will oscillate due to buoyancy and gravity as sketched in Figure 4.4. The Brunt-Vaisala Frequency can be expressed as [30]:

$$N = \left(\frac{g}{\rho}\frac{d\rho}{dz} - \frac{g^2}{c^2}\right)^{1/2}$$

Obviously in this application the density gradient $\frac{d\rho}{dz}$ and speed of sound c are a function of the atmospheric lapse rate (temperature gradient), humidity, etc. In general treating the atmosphere as a stably stratified fluid is a reasonable assumption because the

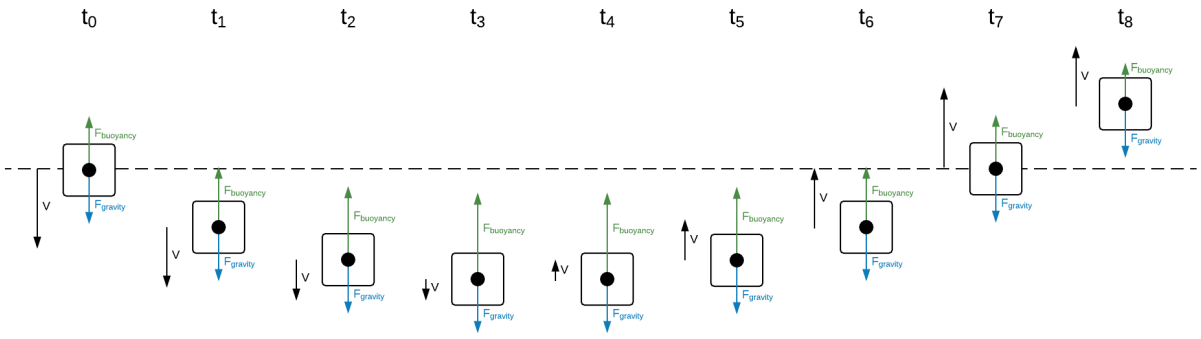

Figure 4.4: Element displaced in stably stratified fluid

lapse rate in the troposphere ($<$ 10km) is nearly constant. This idealization is subject to daily variations and common phenomena such as inversion layers are necessarily ignored, but it is sufficiently general to be one of the pillars of the widely used and internationally recognized "standard day" conditions detailed in [12] and shown in Figure 4.5.

4.2.2 Model Limitations Revisited

The limitation of Greene's model is that it does not include ground effect or cross winds, both of which are critical near runways. It is well known that WV can stall or be pushed onto a neighboring runway due to these effects [22], but the addition of those data sources is left to future work, not to mention adding the effect of trees, structures and local topography. Kantha's model would be an excellent place to start [35].

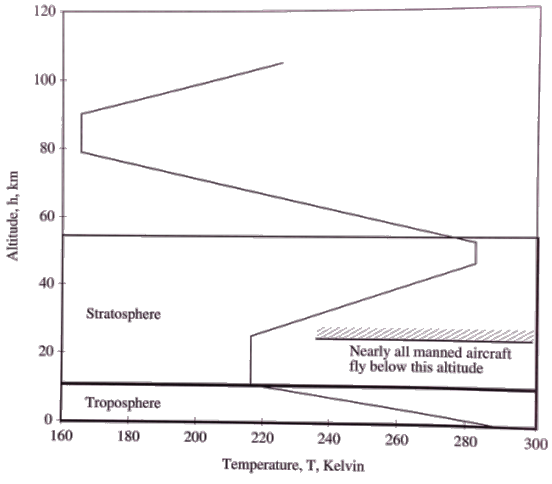

Figure 4.5: Standard atmosphere model, adopted from [12]

Substituting in the width and area of the

NS	Lapse Rate
0	isothermal
.2	std. atmosphere
.4	$.25°C/100m$
.8	$4°C/100m$

QS	Turbulence
$< .2$	negligible
.2	light
.4	moderate
.8	heavy

C_D	R_e	Applicability
.2	$> 6 \times 10^5$	passenger jet
1.4	$< 4 \times 10^5$	GA aircraft

Table 4.1: Wake model parameters from [30]

wake oval $L = 2.09b$, $A = \pi(1.73) \times 2.09b^2$ [30], and the estimated coefficient of drag for the descending wake oval based on its Reynolds number $C_D = 0.2$ [37] simplifies the above equation equation:

$$\frac{d^2H}{dT^2} + 0.033\left(\frac{dH}{dT}\right)^2 + 0.82QS\left(\frac{dH}{dT}\right) + 0.45NS^2H = 0$$

The resulting non-dimensional nonlinear ODE is dependent on the atmospheric turbulence and stratification parameters QS and NS, and independent of aircraft specifics such as weight and air speed. From Table 4.1 $QS = 0.2$ corresponds to "Light" turbulence and $NS = 0.2$ corresponds to the stratification of a standard atmosphere so these values produce a conservative estimate of the wake lifetime and motion. Plugging these values in yields the general non-dimensional nonlinear ODE:

$$\frac{d^2H}{dT^2} + 0.033\left(\frac{dH}{dT}\right)^2 + 0.16\left(\frac{dH}{dT}\right) + 0.018H = 0 \tag{4.11}$$

As discussed in subsection 4.3.1 a single numerical solution to the above ODE can readily be scaled to wake vortices being shed by a wide variety of aircraft in cruise or on approach. Note that this solution is only valid in the limit of large aircraft (e.g. passenger jets) because the Reynolds number of the descending wake oval for a small GA aircraft will be much smaller and thus invalidate the estimated C_D [37].

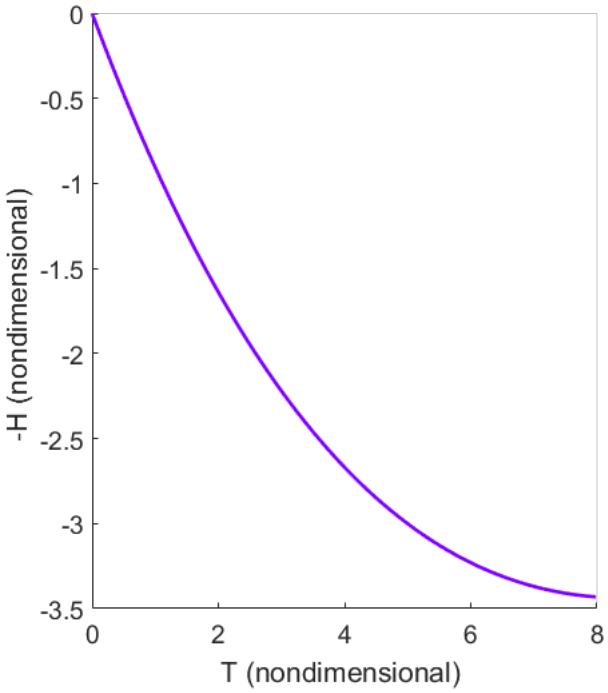

Figure 4.6

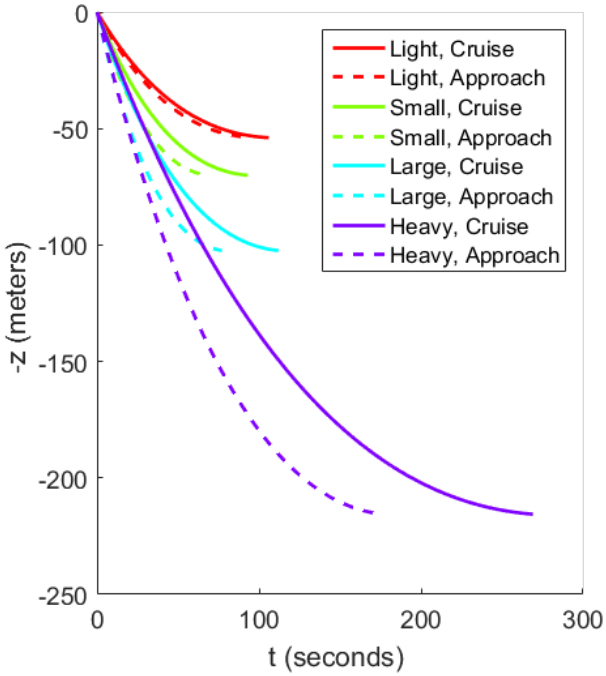

Figure 4.7

4.3 Simulation

Equation 4.11 is readily solved numerically and yields the non-dimensional solutions shown in Figure 4.6 and Figure 4.8. As discussed above, these non-dimensional solutions *only* apply to situations when $QS = NS = 0.2$, which is conservative.

4.3.1 Scaling Factors

The non-dimensional solutions can then dimensionalized by inverting Equations 4.8, 4.9 and 4.10. The coefficients are then found by combining Equation 4.3 with:

$$V_0 = \frac{\Gamma_0}{2\pi b} = \frac{2\Gamma_0}{\pi^2 S}$$

which yields an equation for the initial downwash velocity V_0:

$$\boxed{V_0 = \frac{8L}{\pi^3 \rho U S^2}} \tag{4.12}$$

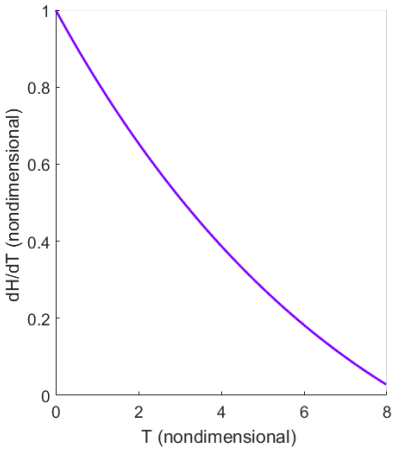

Figure 4.8

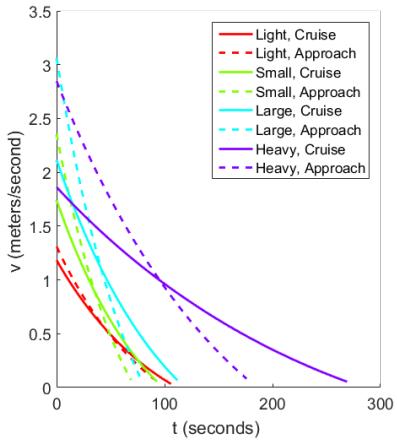

Figure 4.9

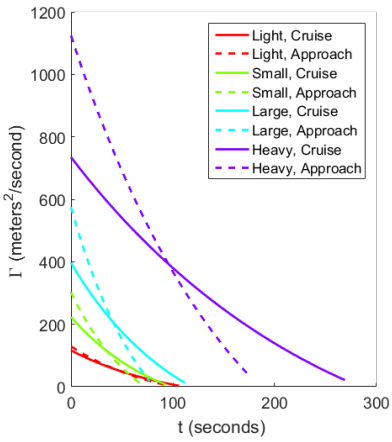

Figure 4.10

Emitter Category	Typical Airframe	Mass (kg)	Wingspan (m)	U_{cruise} (m/s)	$U_{approach}$ (m/s)
Light	EMB120	11500	20	150	50
Small	RJ100	42000	26	220	60
Large[1]	B757	136000	38	275	70
Heavy	A380	560000	80	290	70

Table 4.2: Emitter category parameters

where $L \approx mg$. Note that this "initial" downwash velocity is historically defined and corresponds to 4.2; not the correct near-field velocity discussed in subsection 4.1.5.

Figures 4.7, 4.9 and 4.10 are dimensionalized for the passenger jet categories defined in ICAO and FAA documents [44, 22, 5]. The specific parameters used to create these figures are in Table 4.2 assuming an altitude of $10,000m$ for cruise and $1,000m$ for approach.

4.3.2 Discussion

A striking feature of the dimensionalized figures is that the wake vortices shed by Heavy category aircraft descend much farther, are much more powerful (higher circulation) and

[1] "High Vortex Large" is a special designation for the B757 only so it is used as a worst-case estimate for the Large emitter category

persist much longer than wakes generated by smaller category aircraft. Interestingly, in the context of increasing airport throughput it may actually be counter productive to use super heavy aircraft because of the time overhead required to maintain safe separation [47].

It is also interesting to note that the wake shed by a B757 initially descends more rapidly than any other category. Perhaps this is part of the justification for placing the B757 in its own "High Vortex Large" category.

4.4 Results

Greene's model was published 30 years ago. Given that fact it is unlikely that any observation, figure, or point made above is novel. Therefore the contribution of this project is in leveraging the recent modernization of the NAS (ADS-B in particular) to build a useful wake vortex visualization tool.

4.4.1 C# Implementation

As discussed in Section 2.3.1, TRAPIS is the software cornerstone of this project. In addition to collecting, processing, displaying and storing ADS-B surveillance data TRAPIS exposes its database to extension applications. The Wake Turbulence Estimator is the first of perhaps many extension applications to come that has been built to further process and then visualize the TRAPIS database.

First, a cubic polynomial fit to the non-dimensional solution shown in Figure 4.6 was used as a quick and accurate method for accessing continuous points along the non-dimensional solution curve. Next a function dimensionalizes the solution based on the wake category, altitude and speed of the aircraft according to Equation 4.12 and the "worst case" mass and wingspan specified in Table 4.2. The location history of each aircraft is used to define the top of the wake corridor and then the bottom of the wake corridor is vertically offset based on the age of the vortex. Finally, the WTE generates KML polygon objects that can natively be read and displayed in Google Earth [39].

Figure 4.11

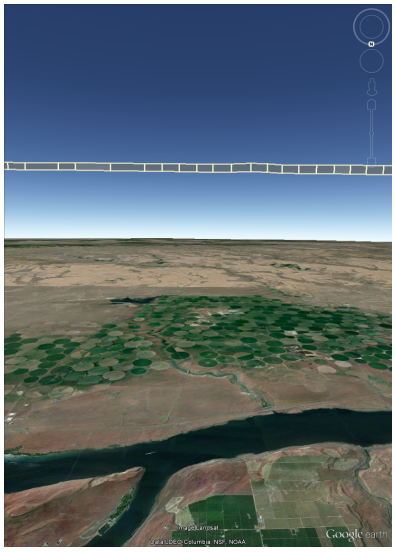

Figure 4.12

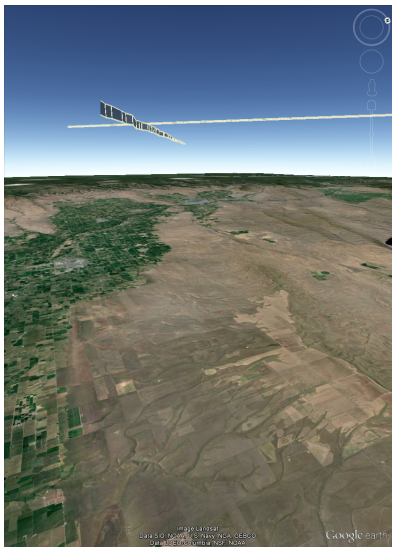

Figure 4.13

All of these calculations occur in a small fraction of a second with only 2% of the processing power of a standard desktop computer even though the WTE consists of over 3000 lines of code - not including preexisting utilities or main TRAPIS functionality. These performance benchmarks come from a data replay of a field test run at The Dalles on 5/6/2016 where over one dozen unique aircraft tracks were recorded over a five minute period. Therefore it is reasonable to expect that the WTE could operate in real time on a pilot's tablet computer in the presence of local air traffic.

4.4.2 Wake Zone Visualization

Figures 4.11, 4.12 and 4.13 show the KML output from the WTE in Google Earth. The wake corridors have very high aspect ratios (> 100) because even for a Heavy category A380 on approach the wake will descend only about 220 meters over 5 minutes, whereas the aircraft will have traveled about 21 km over the same interval. Figure 4.11 depicts a safe crossing above a wake; Figure 4.12 depicts what a pilot would see while on an intersecting course with the wake zone; and 4.13 shows a safe parallel course with a wake.

Chapter 5
CONCLUSION

Two air safety tools have been developed within the context of the FAA's NextGen NAS modernization initiative. Both tools utilize ADS-B data to warn pilots of essentially invisible dangers within their local airspace: small UAS and wake vortices shed by preceding aircraft. Furthermore both tools have been vetted in real world scenarios - either physically in the field or by processing real world data in real time.

5.1 Outlook and Extensions

5.1.1 ADS-B Payload

Admittedly equipping a $1000 hobby drone with a $10,000 ADS-B transponder is unlikely to become a common activity in the near future, but technologies like the ADS-B payload do facilitate a seamless integration of UAS into the NAS and are likely to serve an unmet need on larger commercial, public or military UAS. This would be particularly valuable where joint manned and unmanned aircraft operations are advantageous such as law enforcement, border patrol, wildfire fighting, and search and rescue.

Refinements to the ADS-B payload include matching the RF power output to distance scales appropriate to UAS and making the antenna retractable to protect it during takeoff and landing. The current 120nm range broadcasts are absurd when many UAS are limited to speeds well below 100 knots. 10nm is a more reasonable broadcast range for UAS and would alleviate some of the EMI challenges while simultaneously reducing power draw. A longer range, but publicly relevant and useful extension is building a custom Pixhawk flight mode to directly interface with the XPS-TR. Generating an official pull request for ArduPlane and ArduCopter is a complex and involved task, but it would eliminate the need for peripherals

like the Arduino Mega2560 board and make ADS-B integration with UAS plug-and-play.

5.1.2 Wake Turbulence Estimator

The core functionality of the WTE is sound, and the wake corridor polygon objects implemented in C# are sufficiently flexible to support advanced display options like painting the WV corridors according to an intuitive color coded scheme where more dangerous or likely volumes are red and solid whereas less dangerous or likely volumes are green and more transparent.

To fully support extensions like the WTE the TRAPIS GUI needs to support three dimensional visualization. `GMap.NET` was an excellent starting place, but it is limited to planar maps viewed from above, whereas much of the value of the WTE hinges upon a synthetic vision display that reveals the location of invisible wake zones.

There are also synergistic extensions that tie into existing software tools developed at the AFSL such as the Forward State Estimator and the Airspace Conflict Calculator. In short, aircraft trajectories can be projected into the future and the likelihood of an intersection with wake corridors could be calculated. If that likelihood is above a particular threshold then the software would notify the pilot.

Clearly more software and UI refinements are necessary before the WTE can be deployed to pilots in the cockpit, but most importantly a more detailed physics model including ground effect and cross winds is essential for visualizing WV positions in the final moments before landing. This is a challenging task because accurate, site-specific meteorological details are not necessarily known, and even if they are known in a global sense getting that data into the cockpit and onto the pilot's tablet computer is no small feat.

BIBLIOGRAPHY

[1] Ardupilot autopilot suite. URL http://copter.ardupilot.org/ardupilot/index.html.

[2] Pixhawk autopilot. URL https://pixhawk.org/modules/pixhawk.

[3] Aircraft accident report NTSB/AAR-73/03. Technical report, Oct. 1973. URL http://www.ntsb.gov/investigations/AccidentReports/Reports/AAR7303.pdf.

[4] Aircraft accident report NTSB/AAR-04/04. Technical report, Oct. 2004. URL http://www.ntsb.gov/investigations/AccidentReports/Reports/AAR0404.pdf.

[5] Doc 9871: Technical provisions for mode s services and extended squitter. Technical report, 2008.

[6] Technical standard order C166b: Extended squitter automatic dependent surveillance - broadcast (ADS-B) and traffic information service - broadcast (TIS-B) equipment operating on the radio frequency of 1090 megahertz (mhz), Dec. 2009.

[7] Technical standard order C154c: Universal access transceiver (UAT) automatic dependent surveillance-broadcast (ADS-B) equipment operating on frequency of 978 mhz, Dec. 2009.

[8] Guide on technical and operational considerations for the implementation of ADS-B in the SAM region. Technical report, May 2013.

[9] Air safety: Germany develops wake turbulence warning system, July 2014. URL http://www.thehindubusinessline.com/news/science/

air-safety-germany-develops-wake-turbulence-warning-system/article6167146.ece.

[10] Automatic dependent surveillance-broadcast (ADS-B) out equipment and use, Apr. 2016. URL http://www.ecfr.gov/cgi-bin/text-idx?node=se14.2.91_1225&rgn=div8.

[11] A. Bobylev, V. Vyshinsky, G. Soudakov, and V. Yaroshevsky. Aircraft vortex wake and flight safety problems. *Journal of Aircraft*, 47(2):663, 2010.

[12] S. A. Brandt, R. J. Stiles, J. J. Bertin, and R. Whitford. *Introduction to Aeronautics: A Design Perspective*. American Institute of Aeronautics and Astronautics, Inc., second edition edition, 2004.

[13] Cincinnati Avionics. ADS-B 101: what you need to know, April 2016. URL http://sportysnetwork.com/avionics/ads-b-101-what-you-need-to-know/.

[14] A. J. Cotel and R. E. Breidenthal. Turbulence inside a vortex. *Physics of Fluids*, 11(10): 3026–3029, 1999. doi: http://dx.doi.org/10.1063/1.870161. URL http://scitation.aip.org/content/aip/journal/pof2/11/10/10.1063/1.870161.

[15] S. C. Crow. Stability theory for a pair of trailing vortices. *AIAA Journal*, 8(12):2173–2179, 1970.

[16] S. C. Crow and E. R. Bate Jr. Lifespan of trailing vortices in a turbulent atmosphere. *Journal of Aircraft*, 13(7):476–482, 1976.

[17] Federal Aviation Administration. Unmanned aircraft systems (UAS) registration, . URL https://www.faa.gov/uas/registration/.

[18] Federal Aviation Administration. NextGen, . URL https://www.faa.gov/nextgen/.

[19] *Advisory Circular 91-78*. Federal Aviation Administration, July 2007.

[20] *Terminal Area Forecast Summary: Fiscal Years 2013 - 2040*. Federal Aviation Administration, 2013.

[21] Federal Aviation Administration. FAA wakes up separation standards, Apr. 2013. URL https://www.faa.gov/nextgen/snapshots/stories/?slide=18.

[22] *Advisory Circular 90-23G*. Federal Aviation Administration, February 2014.

[23] *The Business Case for the Next Generation Air Transportation System*. Federal Aviation Administration, Dec. 2014.

[24] Federal Aviation Administration. NextGen portfolio - separation management, Dec. 2015. URL https://www.faa.gov/nextgen/snapshots/portfolios/?portfolioId=13.

[25] *Wake Turbulence Recategorization*. Federal Aviation Administration, Feb. 2015. URL http://www.faa.gov/documentLibrary/media/Order/JO_7110_659C.pdf.

[26] Federal Aviation Administration. FAA administrator makes two major drone announcements, May 2016. URL http://www.faa.gov/news/updates/?newsId=85528/.

[27] Federal Aviation Administration. HAPRA FAA Registry, Mar. 2016. URL http://registry.faa.gov/aircraftinquiry/NNum_Results.aspx?NNumbertxt=N676ZL.

[28] FlightChops. Wake turbulence encounter - POV - flying. URL https://youtu.be/sjifp5oi6dE.

[29] gochase85. Comair3049 wake turbulence and boston atc. URL https://youtu.be/3BYAAmd9scA.

[30] G. Greene. An approximate model of vortex decay in the atmosphere. *Journal of Aircraft*, 7(23):566–573, 1986. URL http://dx.doi.org/10.2514/3.45345.

[31] D. Guerin. Consideration of wake turbulence during the integration of remotely piloted aircraft into the air traffic management system. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 926–935, Denver, CO, 2015.

[32] W. L. Holforty. *Flight-Deck Display of Neighboring Aircraft Wake Vortices*. PhD thesis, Stanford University, June 2003.

[33] B. Jansen. FAA: Drone registration eclipses that of regular planes, Feb. 2016. URL `http://www.usatoday.com/story/news/2016/02/08/faa-drone-registration-eclipses-regular-planes/80002730/`.

[34] C. W. Jennings. *Threat Displays for Final Approach*. PhD thesis, Stanford University, May 2003.

[35] L. H. Kantha. Empirical model of transport and decay of aircraft wake vortices. *Journal of Aircraft*, 35(4):649–652, July 1998.

[36] R. Larson. Robust UAV guidance, navigation and control in GPS-denied environments. Master's thesis, University of Washington, 2016.

[37] B. W. McCormick. *Aerodynamics of V/STOL Flight*. Academic Press, New York, 1967.

[38] J. Moore. Many choices for ads-b equipage. on-line, Mar. 2013. URL `http://www.aopa.org/News-and-Video/All-News/2013/March/28/Many-choices-for-ADS-B-equipage`.

[39] Open Geospatial Consortium. OGC Standards: KML. URL `http://www.opengeospatial.org/standards/kml/`.

[40] R. L. Panton. *Incompressible Flow*. Wiley, Hoboken, 4 edition, 2013.

[41] F. H. Proctor and D. W. Hamilton. Evaluation of fast-time wake vortex prediction models. In *47th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Orlando, FL, 2009.

[42] Radioman. GMap.NET - great maps for windows forms & presentation, Apr. 2015. URL https://greatmaps.codeplex.com/.

[43] W. Rouwhorst, J. Groeneweg, G. Winckelmans, and R. Luckner. Piloted simulation results of an onboard wake vortex detection, warning and avoidance system and cockpit display aspects. In *ICAS2008*. 26TH International Congress of the Aeronautical Sciences, 2008.

[44] *DO-282B: Minimum Operational Performance Standards for Universal Access Transceiver (UAT) Automatic Dependent Surveillance - Broadcast (ADS-B)*. RTCA, Inc., December 2009.

[45] Sagetech Corp. *DOC710R01 XP Transponder User Guide Rev 1.3*, July 2014.

[46] Smart Projects Holdings, Ltd. Sagetech ADS-B transponder integration with UAV. URL https://github.com/UgCS/sagetech-ardu.

[47] P. R. Spalart. Airplane trailing vortices. *Annual Review of Fluid Mechanics*, 30(1): 107–138, 1998. ISSN 00664189.

[48] J. R. Spreiter and A. H. Sacks. The rolling up of the trailing vortex sheet and its effect on the downwash behind wings. *Journal of the Aeronautical Sciences*, 18(1):21–32, 1951.

[49] H. B. Squire. The growth of a vortex in turbulent flow. *The Aeronautical Quarterly*, 16:302–306, 1965.

[50] C. I. Trade. How to read an rc receiver with a microcontroller - part 1, Jan. 2012. URL http://rcarduino.blogspot.co.uk/2012/01/how-to-read-rc-receiver-with.html.

[51] R. Wall. London police say airliner at heathrow may have hit a drone, Apr. 2016. URL http://www.wsj.com/articles/london-police-say-airliner-at-heathrow-may-have-hit-a-drone-1460921037.

[52] C. Whitlock. Near-collisions between drones, airliners surge, new FAA reports show, Nov. 2014. URL https://www.washingtonpost.com/world/national-security/near-collisions-between-drones-airliners-surge-new-faa-reports-show/2014/11/26/9a8c1716-758c-11e4-bd1b-03009bd3e984_story.html.

Appendix A

LICENSES AND PERMISSIONS

A.1 uwthesis template

License for the UW LaTeX thesis template style courtesy of Jim Fox (fox@washington.edu) and freely distributed through `http://staff.washington.edu/fox/tex/uwthesis.shtml`.

A.2 GMap.NET

License for the `GMap.NET` framework coordinated by `radioman` and freely distributed under a MIT license via `https://github.com/radioman/greatmaps`.

GMap.NET License

```
Copyright (c) 2008-2011 Universe, WARNING: This software can access some
map providers and may viotile their Terms of Service, you use it at your
own risk, nothing is forcing you to accept this ;} Source itself is legal!

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
```

A.3 sagetech-ardu

License for the original Arduino code provided by Smart Project Holdings Ltd. and freely distributed through `https://github.com/UgCS/sagetech-ardu`.

SPH Engineering License

```
Copyright (c) 2015, Smart Projects Holdings Ltd
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of the Smart Projects Holdings Ltd nor the
      names of its contributors may be used to endorse or promote products
      derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL SMART PROJECTS HOLDINGS LTD BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.4 Permission to Publish Proprietary Sagetech Corp. Intellectual Property

Ward Handley

From: Jim Davis <Jim.Davis@sagetechcorp.com>
Sent: Monday, April 11, 2016 11:04 AM
To: Ward Handley
Cc: 'Christopher Lum'; Tom Furey
Subject: RE: Permission to publish Sagetech Protocol

Hi Ward,

I discussed your query with Kelvin Scribner (President) and Tom Furey (COO). You have our permission to publish a small subset of the proprietary Sagetech serial communications protocol, much like what was done by SPH Engineering. If the reader needs additional information regarding the protocol, they should be referred to Sagetech.

Best regards,

Jim Davis | Director of Business Development

Sagetech Corporation
509-493-2185 ,105 (Office)
404-333-1718 (Mobile)
SagetechCorp.com

From: Ward Handley [mailto:ward.handley@gmail.com]
Sent: Thursday, March 31, 2016 2:38 PM
To: Jim Davis <Jim.Davis@sagetechcorp.com>
Cc: 'Christopher Lum' <lum@u.washington.edu>
Subject: Permission to publish Sagetech Protocol

Hi Dr. Davis,
Thank you for taking the time to talk with us this afternoon.

As I mentioned, the UW Graduate School takes copyright issues very seriously so I am asking for official permission to publish a small subset of the proprietary Sagetech serial communications protocol. This covers a nearly identical subset of the protocol that was previously published by SPH Engineering (https://github.com/UgCS/sagetech-ardu) with Sagetech's permission.

Thank you,
Ward

Ward Handley
Graduate Research Assistant | Autonomous Flight Systems Laboratory
William E. Boeing Department of Aeronautics & Astronautics | University of Washington
(303) 956-5785 | ward.handley@gmail.com | https://www.linkedin.com/in/handleyw

Appendix B

CODE

B.1 sagetech-ardu.ino

sagetech-ardu.ino

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #include "Arduino.h"
6  #include "led.h"
7  #include "common.h"
8
9  // Led config
10 #define TRANSPONDER_LED 9      // Led to signal that transponder radio transmitter is ON
11 #define MAVLINK_LED     10      // Led to signal that mavlink data is received
12 bool mavlink_led_state
13 // PWM TRANSPONDER MODE reader config
14 #define PWM_TRANS_PIN     7      // PWM input used to select transponder mode.
15 int16_t pwm_trans_duration bool toggle_mode bool toggle_mode_old bool ident_on bool ident_on_old
16 // PWM_GPS_INTERRUPT reader config
17 #define PWM_GPS_PIN      6      // PWM input used to deny GPS data.
18 int16_t pwm_gps_duration // 100ndeg Lat int32_t randomLatScale int32_t randomLonScale
19 // serial port config
20 #define MAVLINK_PORT_BAUD   57600   // baud rate for mavlink port
21 #define SAGETECH_PORT_BAUD  57600   // baud rate for sagetech port
22 #define DEBUG_PORT_BAUD  57600      // baud rate for debug serial port
23 HardwareSerial& mavlink_port HardwareSerial& sagetech_port HardwareSerial& debug_port bool debug_output
24 // Do not poll PWM inputs more often than 3.3Hz, otherwise pulseIn() call blocks serial communications
```

```
25 static constexpr uint32_t PWM_PERIOD  uint32_t last_pwm_poll_time
26 // Do not send coordinates and altitude more often than 5Hz
27 static constexpr uint32_t MAVLINK_PERIOD  uint32_t last_mavlink_change_time
28 // 1200 in octal. SQUAWK code for VFR flights
29 #define SQUAWK_CODE_VFR 0x2AB //0x280
30
31 // Tail number to send via ADSB. Max len static const char TAIL_NUMBER
32 // Current altitude from Ardupilot.
33 long current_altitude_in100ft
34 enum GPS_mode {
35   NORMAL,
36   DEGRADED,
37   DENIED
38 };
39 GPS_mode gps_mode_current GPS_mode gps_mode_old
40 enum class Transponder_mode : uint8_t {
41   UNKNOWN           STANDBY           ALT_INTERNAL    } current_transponder_mode
42 Sagetech_protocol sagetech_protocol;
43 Mavlink_protocol mavlink_protocol;
44
45 // This led is on when transponder is transmitting: ON or ALT mode.
46 Led transponder_active_led(TRANSPONDER_LED, false);
47
48 // This led is on if valid GPS coordinates are present.
49 Led mavlink_active_led(MAVLINK_LED, false);
50
51 void Set_transponder_mode(Transponder_mode mode, bool start_ident {
52   uint8_t sagetech_packetSagetech_protocol::MAX_FRAME_LENGTH;
53   Sagetech_protocol::Operating_message oper;
54   oper.power_up_mode   oper.squawk_code
55   switch (mode) {
56     case Transponder_mode::STANDBY:
57       oper.pressure_altitude       oper.transponder_mode       oper.start_ident       break;
```

```
58      case Transponder_mode::ALT_INTERNAL:
59        oper.pressure_altitude       oper.transponder_mode       oper.start_ident       break;
60      default:
61        return;
62    }
63
64    auto len   sagetech_port.write(sagetech_packet, len);
65  }
66
67  void On_mavlink_gps(Mavlink_protocol::Message_gps_int msg)
68  {
69    if (msg.lat     current_altitude_in100ft     return; // GPS data invalid. Ignore.
70    }
71
72  switch (mavlink_led_state)
73  {
74    case true:
75      mavlink_active_led.On();
76      mavlink_led_state     break;
77    case false:
78      mavlink_active_led.Off();
79      mavlink_led_state     break;
80  }
81
82    current_altitude_in100ft   auto clock
83    if ((clock - last_mavlink_change_time) < MAVLINK_PERIOD) {
84      return;
85    }
86
87    last_mavlink_change_time
88    if (current_transponder_mode !    return;
89    }
90
```

```
91    Sagetech_protocol::Gps_message gps_message;
92    uint8_t sagetech_packetSagetech_protocol::MAX_FRAME_LENGTH;
93
94    switch (gps_mode_current) {
95      case GPS_mode::DENIED:
96        msg.lat       msg.lon       msg.vx       msg.vy       msg.hdg       break;
97      case GPS_mode::DEGRADED:
98        msg.lat       msg.lon       msg.vx       msg.vy       msg.hdg       break;
99      default:
100       // do nothing to the gps data
101       break;
102   }
103
104   gps_message.lat   gps_message.lon   gps_message.alt   gps_message.speed_over_ground   gps_message.cou
105   auto len   sagetech_port.write(sagetech_packet, len);
106
107 if (debug_output) {
108   debug_port.print(",");
109   debug_port.print((long)msg.lat);
110   debug_port.print(",");
111   debug_port.print((long)msg.lon);
112   debug_port.print(",");
113   debug_port.print(sqrt((long)msg.vx * (long)msg.vx + (long)msg.vy * (long)msg.vy));
114   debug_port.print(",");
115   debug_port.println((long)msg.hdg);
116   debug_port.print((uint32_t)msg.time_since_boot);
117 }
118 }
119
120
121 void On_sagetech_ack(Sagetech_protocol::Acknowledge_message msg)
122 {
123   if (debug_output) {
```

```
124   debug_port.print(",");
125   debug_port.print((int)msg.transponder_mode);
126   debug_port.print(",");
127   debug_port.print((uint8_t) gps_mode_current);
128   debug_port.print(",");
129   debug_port.print(msg.ident_active);
130   debug_port.print(",");
131   debug_port.print(msg.altitude_source_external);
132   debug_port.print(",");
133   debug_port.print(msg.extended_squitter_error);
134   debug_port.print(msg.gps_error);
135   debug_port.print(msg.icao_error);
136   debug_port.print((int)msg.power_up_mode);
137   debug_port.print(msg.temperature_error);
138   debug_port.print(msg.transponder_error);
139   debug_port.print(",");
140   debug_port.print(msg.pressure_altitude);
141   }
142
143   switch (msg.transponder_mode) {
144     case Sagetech_protocol::Transponder_mode::STANDBY:
145       current_transponder_mode       transponder_active_led.Off();
146       break;
147     case Sagetech_protocol::Transponder_mode::ALT:
148       current_transponder_mode       transponder_active_led.On();
149       break;
150   }
151 }
152
153 void setup()
154 {
155   // Initialize pwm input
156   pinMode(PWM_TRANS_PIN, INPUT);
```

```
157   pinMode(PWM_GPS_PIN, INPUT);
158
159   // initialize pseudo random number generator
160   randomSeed(analogRead(0));
161
162   // Initialize serial ports
163   mavlink_port.begin(MAVLINK_PORT_BAUD);
164   sagetech_port.begin(SAGETECH_PORT_BAUD);
165   debug_port.begin(DEBUG_PORT_BAUD);
166
167   // Set flight number (Send preflight message to transponder)
168   uint8_t sagetech_packetSagetech_protocol::MAX_FRAME_LENGTH;
169   auto len   sagetech_port.write(sagetech_packet, len);
170   debug_port.println("preflight message sent to transponder");
171   debug_port.println("time_since_boot,tx_mode,gps_mode,ident,alt_ext,err_flags,alt,lat,lon,gnd_speed,hdg
172 }
173
174 void loop()
175 {
176   uint32_t clock
177   if (debug_port.available()) {
178     auto cmd     switch (cmd)
179     {
180       case 'p':
181         debug_output         break;
182       case 'q':
183         debug_output         break;
184     }
185   }
186
187   // Read telemetry data from port and feed it into mavlink protocol parser
188   if (mavlink_port.available()) {
189     // this will call On_mavlink_gps() eventually.
```

```
190       mavlink_protocol.On_data_byte(mavlink_port.read());
191     }
192
193     // Read transponder data from port and feed it into sagetech protocol parser
194     if (sagetech_port.available()) {
195       auto data_byte     // this will call On_sagetech_ack() eventually.
196       sagetech_protocol.On_data_byte(data_byte);
197     }
198
199     // Periodically read pwm for transponder mode, ident, and gps_mode input
200     if ((clock - last_pwm_poll_time) > PWM_PERIOD)
201     {
202       pwm_trans_duration     pwm_gps_duration
203       // GPS MODE
204       if (pwm_gps_duration > 1750) {
205         gps_mode_current     } else if (pwm_gps_duration > 1250) {
206         gps_mode_current     } else {
207         gps_mode_current     }
208
209       if (gps_mode_current     {
210         gps_mode_old     }
211
212       // TRANSPONDER MODE (incl IDENT)
213       toggle_mode     ident_on     if ( (toggle_mode     {
214         toggle_mode_old
215         if (ident_on       {
216           ident_on_old       }
217
218         if (toggle_mode) // toggle_mode       {
219           Set_transponder_mode(Transponder_mode::ALT_INTERNAL, ident_on);
220         }
221         else // toggle_mode       {
222           Set_transponder_mode(Transponder_mode::STANDBY);
```

```
223         }
224       }
225       last_pwm_poll_time   }
226
227   }
```

B.2 common.h

common.h

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #ifndef UTIL_H_
6  #define UTIL_H_
7
8  #include "mavlink_protocol.h"
9  #include "sagetech_protocol.h"
10 #include "Arduino.h"
11
12 void
13 Set_le_i16(long val, uint8_t* data);
14
15 void
16 Set_le_i32(long val, uint8_t* data);
17
18 long
19 Get_le_i16(uint8_t* data);
20
21 long
22 Get_le_i32(uint8_t* data);
23
24 unsigned long
25 Get_le_u16(uint8_t* data);
26
27 unsigned long
28 Get_le_u32(uint8_t* data);
29
30 void
```

```
31 On_mavlink_gps(Mavlink_protocol::Message_gps_int msg);
32
33 void
34 On_sagetech_ack(Sagetech_protocol::Acknowledge_message msg);
35
36 #endif /* UTIL_H_ */
```

B.3 common.cpp

common.cpp

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #include "common.h"
6
7  void
8  Set_le_i16(long val, uint8_t* data)
9  {
10     *(data + 0)     *(data + 1) }
11
12 void
13 Set_le_i32(long val, uint8_t* data)
14 {
15     *(data + 0)     *(data + 1)     *(data + 2)     *(data + 3) }
16
17 long
18 Get_le_i16(uint8_t* data)
19 {
20     return  (static_cast<long>(*(data + 0)) <<  0) +
21             (static_cast<long>(*(data + 1)) <<  8);
22 }
23
24 long
25 Get_le_i32(uint8_t* data)
26 {
27     return  (static_cast<long>(*(data + 0)) <<  0) +
28             (static_cast<long>(*(data + 1)) <<  8) +
29             (static_cast<long>(*(data + 2)) << 16) +
30             (static_cast<long>(*(data + 3)) << 24);
```

```
31 }
32
33 unsigned long
34 Get_le_u16(uint8_t* data)
35 {
36     return  (static_cast<unsigned long>(*(data + 0)) <<  0) +
37             (static_cast<unsigned long>(*(data + 1)) <<  8);
38 }
39
40 unsigned long
41 Get_le_u32(uint8_t* data)
42 {
43     return  (static_cast<unsigned long>(*(data + 0)) <<  0) +
44             (static_cast<unsigned long>(*(data + 1)) <<  8) +
45             (static_cast<unsigned long>(*(data + 2)) << 16) +
46             (static_cast<unsigned long>(*(data + 3)) << 24);
47 }
```

B.4 sagetech_protocol.h

sagetech_protocol.h

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #ifndef SAGETECH_PROTOCOL_H_
6  #define SAGETECH_PROTOCOL_H_
7  #include "Arduino.h"
8
9  class Sagetech_protocol {
10 public:
11     typedef uint8_t Packet_len_size_t;
12
13     enum class Message_type: uint8_t {
14         INVALID                         INSTALLATION                    PREFLIGHT_DATA
15
16     enum class Transponder_mode: uint8_t {
17         OFF             STANDBY //        ON              ALT         };
18
19     enum class Transponder_type: uint8_t {
20         MODE_C                      MODE_S_ADSB_OUT             MODE_S_ADSB_IN_OUT      };
21
22     enum class Power_up_mode: uint8_t {
23         OFF_SQUAWK_1200         USE_NVRAM_MODE      };
24
25     typedef struct {
26         Message_type acked_type;
27         uint8_t acked_id;
28         bool transponder_error;
29         bool altitude_source_external;
30         bool gps_error;
```

```
31          bool icao_error;
32          bool temperature_error;
33          bool extended_squitter_error;
34          Transponder_mode transponder_mode;
35          int pressure_altitude;
36          Transponder_type transponder_type;
37          Power_up_mode power_up_mode;
38          bool ident_active;
39          int squawk_code;
40      } Acknowledge_message;
41
42      typedef struct {
43          int32_t lat;                  // 100nanodegrees
44          int32_t lon;                  // 100nanodegrees
45          int32_t alt;                  // millimeters
46          int32_t speed_over_ground;    // centimeters/second
47          int32_t course_over_ground;   // centidegrees from North
48      } Gps_message;
49
50      typedef struct {
51          long squawk_code;
52          long pressure_altitude;
53          Transponder_mode transponder_mode;
54          Power_up_mode power_up_mode;
55          bool start_ident;
56      } Operating_message;
57
58      static constexpr Packet_len_size_t MAX_FRAME_LENGTH     static constexpr uint32_t ALTITUDE_USE_BUIL
59      virtual
60      ~Sagetech_protocol();
61
62      void On_data_byte(uint8_t data);
63
```

```
64      // assumes buffer has enough space.
65      Packet_len_size_t
66      Create_gps_message(Gps_message msg, uint8_t* buffer);
67
68      // assumes buffer has enough space.
69      Packet_len_size_t
70      Create_operating_message(Operating_message msg, uint8_t* buffer);
71
72      // assumes buffer has enough space.
73      Packet_len_size_t
74      Create_preflight_message(const char* flight_id, uint8_t* buffer);
75
76      // Convert coordinates from 100nanodegrees to Sagetech ascii format.
77      static void Gps_int_to_gprmc(long v, uint8_t *b);
78
79      // Convert speed from centimeters/s to Sagetech ascii format.
80      static void Cms_to_knots(long v, uint8_t *b);
81
82      // Convert course over ground from centidegrees to Sagetech ascii format.
83      static void Cog(long v, uint8_t *b);
84
85  private:
86
87      void
88      On_message();
89
90      Message_type
91      Get_message_type() {return static_cast<Message_type>(incoming_packet1);};
92
93      void
94      Calc_checksum(const uint8_t* data, Packet_len_size_t len, uint8_t& fletcher, uint8_t& additive);
95
96      void
```

```
97      Slice_to_next_start_byte(Packet_len_size_t start
98      uint8_t
99      Get_next_id() {return next_message_id++;};
100
101     Packet_len_size_t
102     Build_frame(Message_type type, uint8_t* frame);
103
104     static constexpr uint8_t START_BYTE     static constexpr uint8_t STOP_BYTE      static constexpr ui
105     static constexpr Packet_len_size_t FOOTER_LENGTH     /** header length (address, type, id, len). */
106     static constexpr Packet_len_size_t HEADER_LENGTH     /** Minimal length of frame (assembly_addr, ty
107     static constexpr Packet_len_size_t MIN_FRAME_LENGTH
108     /** Current protocol state. */
109     bool receiver_waiting_start_byte
110     uint8_t incoming_packetMAX_FRAME_LENGTH;
111     Packet_len_size_t incoming_packet_len
112     uint8_t next_message_id };
113
114 #endif /* SAGETECH_PROTOCOL_H_ */
```

B.5 sagetech_protocol.cpp

sagetech_protocol.cpp

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #include "sagetech_protocol.h"
6  #include "common.h"
7
8  Sagetech_protocol::~Sagetech_protocol()
9  {
10 }
11
12 void
13 Sagetech_protocol::Cms_to_knots(long v, uint8_t *buf)
14 {
15     char tmp10;
16     uint8_t len;
17     if (v < 0) {
18         v     }
19     if (v > 51444) {
20         v     }
21     v     long full     long frac
22     buf0     buf1     buf2     buf3     buf4     buf5
23     ltoa(full, tmp, 10);
24     len     memcpy(buf + 3 - len, tmp, len);
25
26     ltoa(frac, tmp, 10);
27     len     memcpy(buf + 6 - len, tmp, len);
28 }
29
30 void
```

```
31 Sagetech_protocol::Cog(long v, uint8_t *buf)
32 {
33     char tmp10;
34     uint8_t len;
35     if (v < 0) {
36         v     }
37     if (v > 36000) {
38         v     }
39     long full     long frac
40     buf0     buf1     buf2     buf3     buf4     buf5     buf6     buf7
41     ltoa(full, tmp, 10);
42     len     memcpy(buf + 3 - len, tmp, len);
43
44     ltoa(frac, tmp, 10);
45     len     memcpy(buf + 6 - len, tmp, len);
46 }
47
48 void
49 Sagetech_protocol::Gps_int_to_gprmc(long v, uint8_t *buf)
50 {
51     char tmp10;
52     uint8_t len;
53     if (v < 0) {
54         v     }
55     long deg     long min     long dec     min
56     buf0      buf1      buf2      buf3      buf4      buf5      buf6      buf7      buf8      buf9
57     ltoa(deg, tmp, 10);
58     len     memcpy(buf + 3 - len, tmp, len);
59
60     ltoa(min, tmp, 10);
61     len     memcpy(buf + 5 - len, tmp, len);
62
63     ltoa(dec, tmp, 10);
```

```
64      len     memcpy(buf + 11 - len, tmp, len);
65  }
66
67  Sagetech_protocol::Packet_len_size_t
68  Sagetech_protocol::Build_frame(Sagetech_protocol::Message_type type, uint8_t* frame)
69  {
70      Packet_len_size_t payload_len, frame_len;
71      switch (type) {
72      case Message_type::PREFLIGHT_DATA:
73          payload_len         break;
74      case Message_type::OPERATING:
75          payload_len         break;
76      case Message_type::GPS_DATA:
77          payload_len         break;
78      default:
79          break;
80      }
81      frame_len     frame0     frame1     frame2     frame3     frame4
82      Calc_checksum(
83              frame + 1,
84              HEADER_LENGTH - 1 + payload_len,
85              frameframe_len - 3,
86              frameframe_len - 2);
87      frameframe_len - 1
88      return frame_len;
89  }
90
91  Sagetech_protocol::Packet_len_size_t
92  Sagetech_protocol::Create_preflight_message(const char* flight_id, uint8_t* buffer)
93  {
94      uint8_t i;
95      if (flight_id) {
96          for (i             buffer5 + i         }
```

```
97          for (; i < 8; i++) {
98              buffer5 + i         }
99      } else {
100         for (i              buffer5 + i         }
101     }
102
103     buffer13     buffer14
104     return Build_frame(Message_type::PREFLIGHT_DATA, buffer);
105 }
106
107 Sagetech_protocol::Packet_len_size_t
108 Sagetech_protocol::Create_operating_message(Operating_message m, uint8_t* buffer)
109 {
110     buffer5     buffer6     buffer7     buffer8     buffer9     if (m.start_ident) {
111         buffer9 ^    }
112     buffer10     buffer11     buffer12
113     return Build_frame(Message_type::OPERATING, buffer);
114 }
115
116 Sagetech_protocol::Packet_len_size_t
117 Sagetech_protocol::Create_gps_message(Gps_message m, uint8_t* buffer)
118 {
119     // put latitude first because it has 2 digit integer part
120     Gps_int_to_gprmc(m.lat, buffer + 15);
121     // this overwrites the first digit (0) of lat.
122     Gps_int_to_gprmc(m.lon, buffer + 5);
123     Cms_to_knots(m.speed_over_ground, buffer + 26);
124     Cog(m.course_over_ground, buffer + 32);
125     buffer40     if (m.lat >        buffer40 ^    }
126     if (m.lon >        buffer40 ^    }
127
128     // We do not know the gps time.
129     // Docs say there it should be filled with spaces (0x20) but
```

```
130     // that does not work. Transponder responds with error.
131     // So, we fill with invalid data: 99:99:99.999
132     for (int i         bufferi     }
133     buffer47
134     // Zero last 6 bytes (reserved)
135     for (int i         bufferi     }
136
137     return Build_frame(Message_type::GPS_DATA, buffer);
138 }
139
140 void
141 Sagetech_protocol::On_message()
142 {
143     // We need to process only ack message for now.
144     if (Get_message_type()         Acknowledge_message m;
145         if (incoming_packet3 !            return;
146         }
147         m.acked_type         m.acked_id         m.transponder_error         m.altitude_source_external
148                 static_cast<int>(*(incoming_packet + 8)) * 256 +
149                 static_cast<int>(*(incoming_packet + 9));
150         m.transponder_type         m.power_up_mode         m.ident_active         m.squawk_code
151     }
152 }
153
154 void
155 Sagetech_protocol::On_data_byte(uint8_t data_byte)
156 {
157     if (receiver_waiting_start_byte) {
158         if (data_byte             receiver_waiting_start_byte         }
159         return;
160     }
161
162     incoming_packetincoming_packet_len     incoming_packet_len++;
```

```
163
164     while (incoming_packet_len) {
165
166         // need at least minimal packet before processing;
167         if (incoming_packet_len < MIN_FRAME_LENGTH) {
168             return;
169         }
170
171         // packet must fit in our buffer and must have 0x01 as first byte.
172         if (    incoming_packet_len > MAX_FRAME_LENGTH - MIN_FRAME_LENGTH
173             ^^  incoming_packet0 !            Slice_to_next_start_byte();
174             continue;
175         }
176
177         auto payload_length
178         // need the whole packet before processing;
179         if (incoming_packet_len < HEADER_LENGTH - 1 + payload_length + FOOTER_LENGTH) {
180             return;
181         }
182
183         uint8_t cs1, cs2;
184         Calc_checksum(incoming_packet, payload_length + 4, cs1, cs2);
185
186         // packet footer must have valid checksum and stop byte.
187         if (    incoming_packetpayload_length + 4             &&  incoming_packetpayload_length + 5
188             Slice_to_next_start_byte(HEADER_LENGTH - 1 + payload_length + FOOTER_LENGTH);
189         } else {
190             Slice_to_next_start_byte();
191         }
192     }
193     receiver_waiting_start_byte }
194
195 void
```

```
Sagetech_protocol::Slice_to_next_start_byte(Packet_len_size_t startindex)
{
    Packet_len_size_t sbindex;
    for (sbindex         if (incoming_packetsbindex             sbindex++;
            incoming_packet_len -            for (Packet_len_size_t i                 incoming_packeti
            return;
        }
    }
    incoming_packet_len }

void
Sagetech_protocol::Calc_checksum(const uint8_t* data, Packet_len_size_t len, uint8_t& fletcher, uint8_t&
{
    additive     fletcher     for (Packet_len_size_t i     {
        additive +        fletcher +    }
}
```

B.6 mavlink_protocol.h

mavlink_protocol.h

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #ifndef MAVLINK_PROTOCOL_H_
6  #define MAVLINK_PROTOCOL_H_
7
8  #include <Arduino.h>
9
10 class Mavlink_protocol {
11 public:
12     typedef uint16_t Packet_len_size_t;
13     static constexpr Packet_len_size_t MAX_FRAME_LENGTH
14     typedef struct {
15         uint32_t time_since_boot;   // milliseconds
16         int32_t lat;                // 100nanodegrees
17         int32_t lon;                // 100nanodegrees
18         int32_t alt;                // millimeters
19         int32_t relative_alt;       // millimeters
20         int16_t vx;                 // cm/s
21         int16_t vy;                 // cm/s
22         int16_t vz;                 // cm/s
23         uint16_t hdg;                // centidegrees
24     } Message_gps_int;
25
26     virtual ~Mavlink_protocol();
27
28     void
29     On_data_byte(uint8_t data);
30
```

```
31 private:
32     typedef enum {
33         HEARTBEAT         GLOBAL_POSITION_INT     } Message_type;
34 
35     void
36     On_message();
37 
38     bool
39     Calc_checksum(uint8_t* data, bool save_into_packet
40     void
41     Slice_to_next_start_byte(Packet_len_size_t start
42     static bool
43     Get_extra_crc_data(uint8_t* packet, uint8_t& extra_byte);
44 
45     static constexpr uint8_t START_BYTE     /** frame footer length (csum, csum, trailer byte). */
46     static constexpr Packet_len_size_t FOOTER_LENGTH     /** header length (address, type, id, len). */
47     static constexpr Packet_len_size_t HEADER_LENGTH     /** Minimal length of frame (assembly_addr, ty
48     static constexpr Packet_len_size_t MIN_FRAME_LENGTH
49     /** Current protocol state. */
50     bool receiver_waiting_start_byte
51     uint8_t packetMAX_FRAME_LENGTH;
52     Packet_len_size_t packet_len };
53 
54 #endif /* MAVLINK_PROTOCOL_H_ */
```

B.7 mavlink_protocol.cpp

mavlink_protocol.cpp

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #include "mavlink_protocol.h"
6  #include "common.h"
7
8  Mavlink_protocol::~Mavlink_protocol()
9  {
10 }
11
12 void
13 Mavlink_protocol::On_message()
14 {
15     Message_gps_int msg_gps;
16     switch (packet4) {
17     case HEARTBEAT:
18         break;
19     case GLOBAL_POSITION_INT:
20         msg_gps.time_since_boot         msg_gps.lat         msg_gps.lon         msg_gps.alt         msg
21         break;
22     default:
23         return;
24     }
25 }
26
27 void
28 Mavlink_protocol::On_data_byte(uint8_t data_byte)
29 {
30     if (receiver_waiting_start_byte) {
```

```
31          if (data_byte             receiver_waiting_start_byte         }
32          return;
33      }
34
35      packetpacket_len     packet_len++;
36
37      while (packet_len) {
38
39          auto payload_length
40          // need the whole packet before processing;
41          if (packet_len < HEADER_LENGTH + payload_length + FOOTER_LENGTH) {
42              return;
43          }
44
45          // packet must fit in our buffer.
46          if (packet_len             Slice_to_next_start_byte();
47              continue;
48          }
49
50          // packet footer must have valid checksum.
51          if (Calc_checksum(packet)) {
52              On_message();
53              Slice_to_next_start_byte(HEADER_LENGTH + payload_length + FOOTER_LENGTH);
54          } else {
55              Slice_to_next_start_byte();
56          }
57      }
58      receiver_waiting_start_byte }
59
60  void
61  Mavlink_protocol::Slice_to_next_start_byte(Packet_len_size_t startindex)
62  {
63      Packet_len_size_t sbindex;
```

```
64      for (sbindex         if (packetsbindex             sbindex++;
65              packet_len -            for (Packet_len_size_t i                 packeti             }
66              return;
67          }
68      }
69      packet_len }
70
71  bool
72  Mavlink_protocol::Get_extra_crc_data(uint8_t* packet, uint8_t& extra_data_out)
73  {
74      Packet_len_size_t extra_len     switch (packet4) {
75      case HEARTBEAT:
76          extra_len         extra_data         break;
77      case GLOBAL_POSITION_INT:
78          extra_len         extra_data         break;
79      default:
80          return false;
81      }
82      if (packet0 !        return false;
83      }
84      extra_data_out     return true;
85  }
86
87  bool
88  Mavlink_protocol::Calc_checksum(uint8_t* packet, bool save_into_packet)
89  {
90      uint16_t csum     Packet_len_size_t len     uint8_t tmp;
91      uint8_t* data
92      uint8_t extra_crc_byte;
93
94      if (!Get_extra_crc_data(packet, extra_crc_byte)) {
95          return false;
96      }
```

```
97
98      while (len--) {
99          tmp         tmp ^        csum     }
100
101     tmp     tmp ^    csum
102     if (save_into_packet) {
103         *(data++)         *(data++)     } else {
104         return (*(data++)     }
105     return true;
106 }
```

B.8 led.h

led.h

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #ifndef LED_H_
6  #define LED_H_
7
8  #include "Arduino.h"
9
10 class Led {
11 public:
12     Led(uint8_t pin, bool initial_state_on);
13
14     void
15     On();
16
17     void
18     Off();
19
20     // Blink the led. Should call Loop() in the loop().
21     // on  - On time in milliseconds
22     // off - Off time in milliseconds
23     void
24     Blink(unsigned int on, unsigned int off
25     void Loop();
26 private:
27     void
28     Turn_on();
29
30     void
```

```
31      Turn_off();
32
33      unsigned long off_time     unsigned long on_time     unsigned long off_interval     unsigned long o
34      bool is_on;
35  };
36
37  #endif /* LED_H_ */
```

B.9 led.cpp

led.cpp

```
1  // Copyright (c) 2015, Smart Projects Holdings Ltd
2  // All rights reserved.
3  // See LICENSE file for license details.
4
5  #include "led.h"
6
7  Led::Led(uint8_t pin, bool initial_state_on)
8  :pin(pin){
9      pinMode(pin, OUTPUT);
10     if (initial_state_on) {
11         On();
12     } else {
13         Off();
14     }
15 }
16
17 void
18 Led::On()
19 {
20     Turn_on();
21     blink_count }
22
23 void
24 Led::Off()
25 {
26     Turn_off();
27     blink_count }
28
29 void
30 Led::Turn_on()
```

```
31 {
32     digitalWrite(pin, HIGH);
33     on_time     is_on }
34 
35 void
36 Led::Turn_off()
37 {
38     digitalWrite(pin, LOW);
39     off_time     is_on }
40 
41 void
42 Led::Blink(unsigned int ontime, unsigned int offtime, unsigned int n, unsigned int pause)
43 {
44     on_interval     off_interval     pause_time     blink_count }
45 
46 void
47 Led::Loop()
48 {
49     if (blink_count         return;
50     }
51 
52     unsigned long clock
53     if (is_on) {
54         if (clock - on_time > on_interval) {
55             current_blink_count++;
56             if (off_interval                blink_count             }
57             Turn_off();
58         }
59     } else {
60         unsigned long pause;
61         if (current_blink_count >            pause         } else {
62             pause         }
63         if (clock - off_time > pause) {
```

```
64              if (current_blink_count >                current_blink_count             }
65              Turn_on();
66          }
67      }
68  }
```