UNIVERSITY of York

Department of Electronics

MEng Project Report

2015/2016

Student Name: Alexander Cash

Project Title: Autopilot for Aerial Photography

Supervisors: Dr. Andrew Pomfret and Tim Clarke

University of York
Department of Electronics
Heslington
York
YO10 5DD

AUTOPILOT FOR AERIAL PHOTOGRAPHY

Alexander Cash

James College
University of York

May 2016

4th Year Project Report for degree of
Master of Electronic and Computer Engineering With a Year in Industry

I would like to dedicate this report to my school teachers who said I was too lazy to amount to anything.

Acknowledgements

And I would like to acknowledge firstly Dr. Andrew Pomfret

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 An Introduction to Unmanned Aerial Vehicles

The term Unmanned Aerial Vehicle (UAV) can be used to describe any form of aircraft that does not require a pilot on board. This term includes aircraft that are controlled remotely, autonomous vehicles capable of navigating themselves, and any aircraft that are a combination of the two. Other terms that may be familiar are "drone" and Unmanned Aerial System (UAS), which can be used interchangeably with UAV within the context of this project.

UAVs come in a wide range of form factors but can be split into two main classes; fixed-wing UAVs and rotary wing UAVs. Fixed-wing UAVs are generally only capable of horizontal take-off and landing (HTOL), whilst rotary wing UAVs generally employ vertical take-off and landing (VTOL). There are, however, other forms of UAV which incorporate features of both main classes; for example, quadplane UAVS use rotary wing rotors to allow VTOL, but do the main bulk of their flying with forward facing rotors much like a fixed-wing UAV. Across these categories are a very wide range of UAV platforms, from hobbyist quadcopters to military attack vehicles, and many many others in between. Although the range of UAV form factors and their uses is vast, the vast majority are suitable for candidates for complete autonomous flight control, as shall be described in Section

Although there were a large number of suitable candidates for the following work, it was decided that this project would be aimed at the control of strictly fixed-wing UAVs for the purposes of aerial photography. For that reason, we shall not discuss any other style of UAV.

1.2 UAVs for Aerial Photography

The first recorded use of aerial photography was in 1858 by a French balloonist and photographer known as "Nadar" at a height of 80 metres using a tethered hot air balloon. Since then aerial photography has of course massively progressed, both in terms of the aerial vehicles in use and the cameras available. As such, it is now an activity that can be partaken in by both organisations, companies, and individuals alike, for a range of purposes. For example, aerial photography of course plays a very large role in intelligence gathering by government agencies and militaries, but can also be a hobby for otherwise earth-bound photography enthusiasts.

Different applications will require different equipment and styles of photography; a hobbyist may want a wide panorama, whilst a mapping application will require a consistent vertical camera angle. The use of aerial photography for mapping or measurement is known as aerial photogrammetry, and will be the chosen field of interest for this project. Aerial photogrammetry is a specific subset of aerial photography in that it not only requires the capturing of images, but additional information about where the image was captured. For this project, we are looking at the use of aerial photogrammetry to create an image of the land below our UAV, either for mapping or general surveying purposes. In order to be able to do this, we need to be able to capture a set of images, and know the location of the UAV when the images were captured, to allow the images to be stitched together into one larger image of the ground beneath. In this scenario we can typically use GPS co-ordinates and a pre-planned imaging path to define which images relate to which areas of the ground below.

This project will primarily focus on the mapping of land beneath the UAV, and as such we need to define a specified flightpath over the chosen area. There are a number of typical flightpaths for surveying or mapping land, two forms are lawnmower pattern paths and spiral pattern paths. Fig. 1.1 shows the general form of a lawnmower pattern, and Fig. 1.2 shows us the general form of a spiral pattern.

Using these patterns, we can define what shall be referred to henceforth as "imaging paths" or "imaging runs". These are the sections of the flightpath for which we are capturing images of the ground below, and as such we very much care about the position and orientation of the UAV whilst following these paths. For the lawnmower pattern the imaging paths are the vertical sections of the path seen in Fig. 1.1, whereas for the spiral path all sections of the flight plan are considered imaging paths.

As the lawnmower pattern does not require the horizontal sections of the path in Fig. 1.1 to be used for capturing images, we are not particularly concerned with how we fly between the imaging paths. Knowing this, we can specify our flightpath in terms of simply our imaging paths, as can be seen in Fig. 1.3

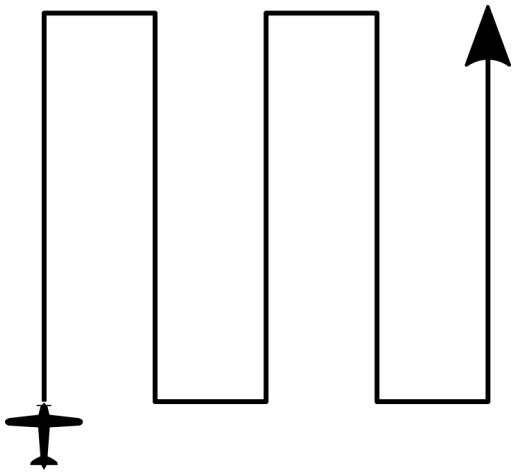

Fig. 1.1 The general form of a lawnmower pattern aerial imaging flightpath

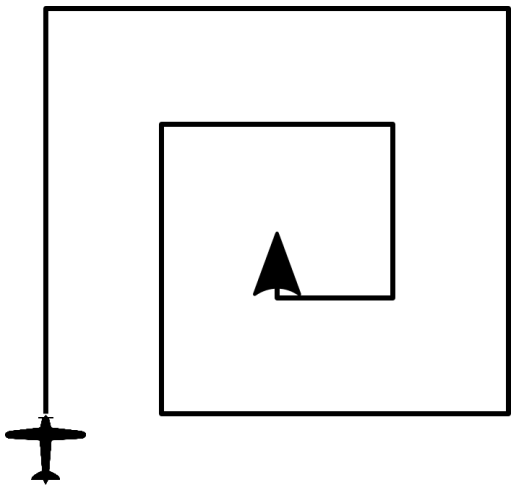

Fig. 1.2 The general form of a spiral pattern aerial imaging flightpath

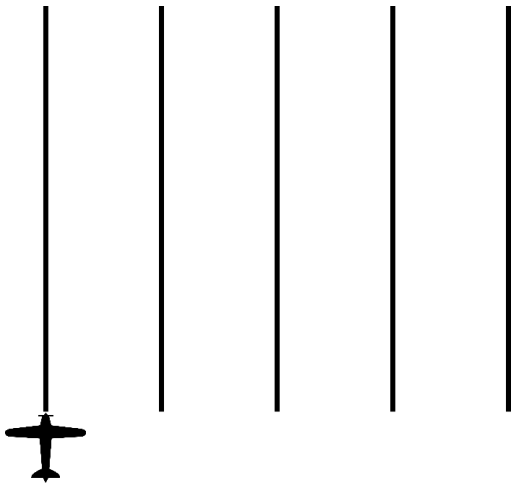

Fig. 1.3 A version of the lawnmower pattern, using only imaging paths

As the imaging paths shown in Fig. 1.3 are separate of one another, it is possible to complete each imaging path in any order and in either direction. This does however require the user to plan accordingly and to record location data accurately so as to be able to stitch together the resulting photographs correctly. One of the resulting patterns that we can create from this is called the "Zamboni pattern", named after the route a Zamboni machine takes around an ice rink to smooth out the ice. A Zamboni pattern describes a path whereby we d onot traverse up and then down adjacent imaging paths, but instead perform a series of overlapping paths by travelling up one imaging path, and then down a path a few rows over. This is necessary for a Zamboni due to its large minimum turning radius, and would be equivalent to a UAV with a large turning radius being required to traverse a number of very close imaging paths.

The required flightpath will have to be calculated based on a number of factors including:

- The size of the area we are wishing to photograph
- Changes in ground elevation over the chosen area
- The operational range of the UAV based on battery life or fuel amount
- The desired resolution of the resulting images - we must fly lower for higher detail images
- The distance between imaging paths will be a function of the desired flying height and the angle of the lens on the camera
- The path to follow between the imaging paths will be a function of the turning performance of the UAV
- Environmental factors such as wind during the flight

These factors shall be considered and discussed later in this report where relevant, and have been used to inform decisions throughout the work of this project.

Throughout this work we shall be referring to a flightpath as a "mission", which describes a full, pre-planned flight path from take-off to landing. To define these missions we require particular software products that shall be discussed in Section 1.3.3.

1.3 ArduPilot

ArduPilot, also known as ArduPilotMega (APM), is an open-source suite of autopilot products aimed at hobbyists and professionals alike. ArduPilot includes the ArduPlane, ArduCopter,

and ArduRover autopilots, each aimed at aeroplane style UAVS, helicopter and multirotor UAVS, and ground based rover platforms respectively. As previously mentioned, this project is aimed at fixed-wing aeroplane UAVs, and as such this report is looking at the ArduPlane product.

ArduPilot has been around since late 2009 and was originally designed to operate using the Arduino development ecosystem. Since then, the codebases have evolved in both size and functionality, in part due to hardware improvements which will be discussed in 1.3.3. This has lead to both the software and hardware components of these ArduPilot autopilot solutions moving away from Arduino almost entirely.

ArduPilot grew out of an organisation called DIYDrones, which was set up as an online community for enthusiasts to discuss and create solutions for UAV use; this community still exists and operates as one of the many hubs for discussion regarding UAVs, autopilots, and route planning. In 2014, the group of people maintaining the ArduPilot project teamed up with The Linux Foundation to create a non-profit organisation called the Dronecode Project. The Dronecode Project is an effort to collaborate a number of open-source hardware and software products under one structure, which includes the ArduPilot project. In February 2016, it was announced that the core team behind ArduPilot would be creating their own non-profit organisation to best suit the needs of the users of the ArduPilot products, however at the time of writing this is still a work in progress.

1.3.1 ArduPlane

ArduPlane is designed so that when paired with a suitable controller board (see Section 1.4), it enables autonomous flight on almost any fixed-wing UAV. To do this, the ArduPlane software is compiled to create the firmware, which is then programmed onto the controller board alongside the mission plan. Once the controller board is connected to all of the necessary control outputs and sensor inputs, the UAV must be configured and tuned for flight, and then is ready for take-off. The configuration and tuning are of course capabilities provided by ArduPlane, however are not relevant for discussion within the context of this project. ArduPlane handles both automatic take-off and landings, as well as providing the option to change between flight modes. The flight modes available include fly-by-wire modes, an acrobatics mode, a training mode, and an automatic flight mode. For this project we are only concerned with the automatic flight mode, and shall not be dealing with any other mode.

The ArduPlane codebase uses a very large collection of libraries, of which the majority are also used by the ArduCopter and ArduRover products. This, combined with the fact that ArduPlane is designed to work with any fixed-wind UAV means that a lot of the code is very generic and reusable. The codebase for the whole ArduPilot project currently consists of

roughly 11,000 files in total, and the specific ArduPlane code (excluding any shared libraries or resources) is made up of only 37 files. The sheer number of shared files, generic coding style, and open-source nature of this project combine to mean that in many respects the code commenting and documentation is very lacking and unspecific. That being said, there are a number of instructional guides and references that can be found on the ArduPilot website for anyone looking to modify the code or get involved with contribution. In addition to this, there are a number of forums that can provide many answers to common problems. In addition to there being three main autopilot platforms, there are also a large number of differing released versions and code branches, which are often platform specific or hardware specific.

1.3.2 JSBSim

As the ArduPilot project is open source, there are a number of associated open-source utilities available to download for testing purposes. One of the main tools used in ArduPlane development is called JSBSim, an open-source Flight Dynamics Model (FDM) which can be used as part of a flight simulator.

We are able to use this FDM to provide a Software In The Loop (SITL) simulator, which does not require any additional hardware besides a computer. This enables us to modify the ArduPilot code and test it with a range of flight missions without the need to purchase a control board or have a suitable UAV available.

The ArduPlane project includes a utility to simulate and view a fixed-wing UAV in flight, starting at any given GPS location. Fig. 1.4 shows one example of this, where we have the command window, an output console, and a map view. The map view shows us the planned mission and the actual path the UAV will follow in the given scenario, the console window displays output and communication messages from the UAV, and the command window allows us to control the simulator.

This simulator forms the bulk of the testing for this project, and is used to map flight paths and record performance. Each simulated UAV flight generates a log in the form of a *.BIN* file. We can then use the MAVExplorer utility and our ground control software to analyse the log files.

MAVExplorer

MAVExplorer is a log analysis tool, provided in the form of a python script suitably named *MAVExplorer.py*. This allows us to generate a map of the area over which the UAV has flown with white lines displaying the planned mission, and a green line plotting the actual route the UAV flew. An example of this can be seen in Fig.

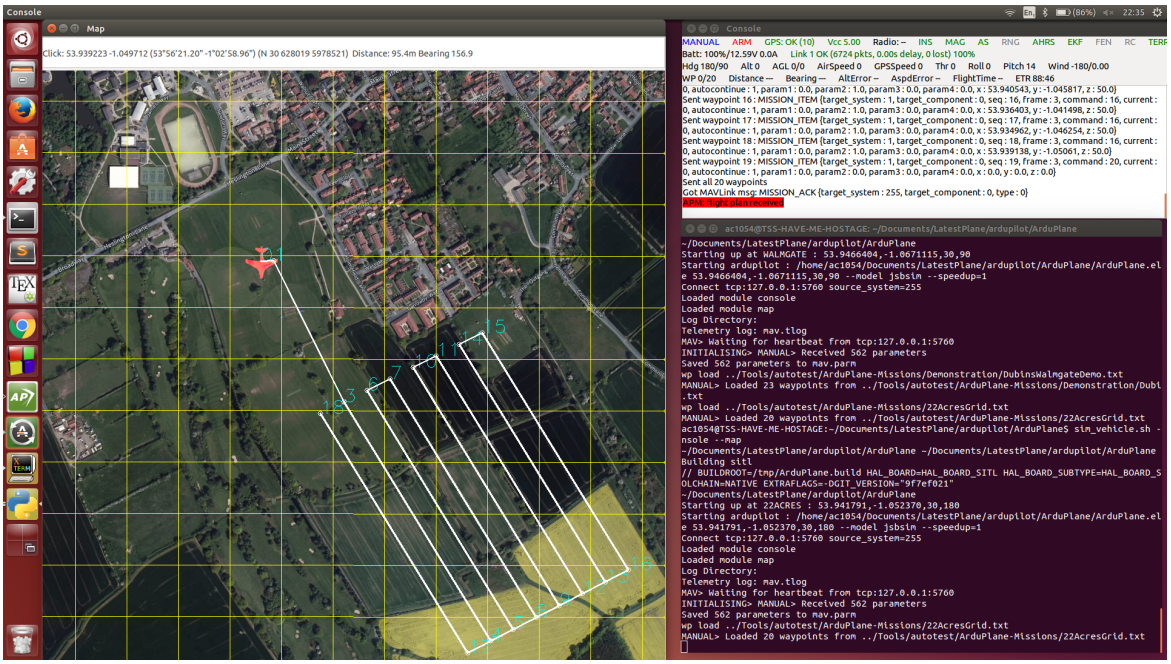

Fig. 1.4 A view of the JSBSim based flight simulator, showing a command window, output console, and simulator map view

1.3.3 MissionPlanner and APM Planner

As previously mentioned, for autonomous UAV flight we need to plan out a mission before flight. There are a number of software products capable of this, however the main two used by the ArduPilot project are called APM Planner 2 and MissionPlanner. Both of these products are open source however APM Planner 2 runs on Linux, Windows, and Mac OSX, whilst MissionPlanner is a Windows only application. Both products enable mission planning however the MissionPlanner product is more feature rich, reliable, and quick to use.

The missions plans generated using either of these programs are of the same format; a text file with a number of commands, parameters, and often GPS locations. This is very useful as it allows manual editing of these text files where necessary, and allows us to easily work out how the autopilot itself reads its commands from the provided mission plan.

There are a wide range of commands that can be included in a mission including take off commands, land commands, and navigation commands. For this project the most relevant navigation commands are those conducted using waypoints. We are able to instruct he UAV to loiter around a point at a specified radius for a number of rotations (complete circles around a point), a duration, or indefinitely. The other form of navigation command is simply an instruction to travel to a point in space, with the option to define how close to that point we need the UAV to travel.

Fig. 1.5 The mapping view provided by MAVExplorer utilising log files from flight simulations

Fig. 1.5 shows a typical mission plan in MissionPlanner, where we are commanding a simple lawnmower pattern imaging run.

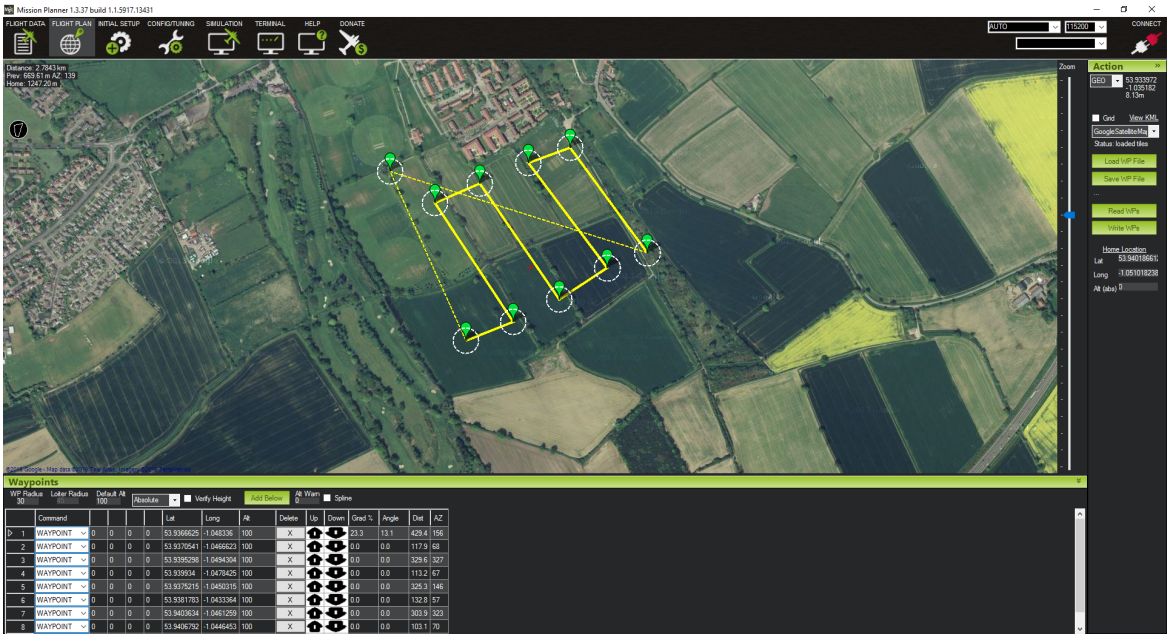

Fig. 1.6 The mission planning view of the Windows based MissionPlanner software

1.4 Autopilot Hardware

Chapter 2

Literature Review

2.1 Dubin's Paths

2.2 Path Following in Wind

Chapter 3

Task One: Path Planning

3.1 The Problem

3.2 Solution

Chapter 4

Task Two: Path Following

4.1 The Problem

4.2 Solution

Chapter 5

Future Work

5.1 Proposal 1: Incorporation into MissionPlanner

Chapter 6

Project Planning and Management

6.1 Agile Planning

Chapter 7

Summary and Conclusions

7.1 Conclusions

References

[1] Abramovich, Y. A., Aliprantis, C. D., and Burkinshaw, O. (1995). Another characterization of the invariant subspace problem. *Operator Theory in Function Spaces and Banach Lattices.* The A.C. Zaanen Anniversary Volume*, Operator Theory: Advances and Applications*, 75:15–31. Birkhäuser Verlag.

[2] Ancey, C., Coussot, P., and Evesque, P. (1996). Examination of the possibility of a fluid-mechanics treatment of dense granular flows. *Mechanics of Cohesive-frictional Materials*, 1(4):385–403.

[3] Aupetit, B. (1991). *A Primer on Spectral Theory*. Springer-Verlag, New York.

[4] Conway, J. B. (1990). *A Course in Functional Analysis*. Springer-Verlag, New York, second edition.

[5] Ljubič, J. I. and Macaev, V. I. (1965). On operators with a separable spectrum. *Amer. Math. Soc. Transl. (2)*, 47:89–129.

[6] Read, C. J. (1985). A solution to the invariant subspace problem on the space l_1. *Bull. London Math. Soc.*, 17:305–317.

Appendix A

Testing Results

Appendix B

MATLAB Outputs

Index