Hardware-in-the-Loop Simulation of an Autogyro UAV Control System

Rory Greig supervised by Peter Baxendale

***Abstract*—This paper details development of an autonomous control system for a lightweight autogyro UAV, based on an off-the-shelf Arduino autopilot system designed for control of lightweight UAVs. Starting from firmware intended for fixed-wing aircraft the control algorithms were modified to be capable of piloting an autogyro. A hardware-in-the-loop simulation environment was set up, allowing the control algorithms to be verified and tested in a flexible environment which didn't risk damage to valuable hardware. The flight simulator 'FlightGear' was used to replicate the performance of the autogyro UAV in real time. A model of the dynamic behaviour of the Durham University autogyro UAV was developed for FlightGear based on existing designs. Stable autonomous flight and successful navigation to waypoints were achieved in the HIL simulation through new pitch and throttle control algorithms as well as correct tuning of control parameters such as the PID controllers. By providing a test bed for control algorithm development and predicting the flight performance of the autogyro structure this HIL simulation environment has demonstrated its usefulness as a design tool for both the control system and the physical design of the UAV.**

***Index Terms*—Autogyro, UAV, Autopilot, HIL Simulation, Ardupilot.**

I. Introduction

Unmanned aerial vehicles (UAVs) have found widespread use in applications which are considered too "dull, dirty or dangerous" for piloted aircraft. The advent of inexpensive embedded electronics with powerful processing capabilities has seen a boom in low-cost, light-weight UAVs for civilian purposes such as environmental surveillance [1]. An autogyro UAV offers several potential benefits over both fixed-wing and helicopter alternatives by exploiting the inherent qualities of autogyros, such as low stalling speed and fuel efficiency [2]. This could result in better fuel consumption as well as more stable flight at low speeds, both of which are an advantage in a surveillance context.

This project aims to develop stabilisation and autopilot algorithms for a lightweight autogyro, as part of ongoing development work on an autogyro UAV at Durham University. In order to comply with UK aviation laws it must weigh less than 7kg to be flown without a license. There are no available examples of any similarly sized autogyros in development, so designing control algorithms for such a lightweight autogyro presents a novel challenge.

Little is known about the stability characteristics of such a lightweight autogyro. This means that the first test flight poses a significant risk of a crash, resulting in expensive damage to hardware. It is therefore desirable to develop the control system before a physical flight test to reduce this risk. The mechanical design of this UAV has not yet reached the stage of being able to fly, so flight testing is currently not possible. For these reasons it was decided to develop a hardware-in-the-loop (HIL) simulation environment to allow the performance of the autogyro to be tested in a controlled, risk-free way.

Hardware-in-the-loop simulation is a method for testing physical control hardware inside a simulated environment, in this sense it differs from pure software simulation. A simulation model is run on a separate computer which takes the output signals from the hardware, as well as emulating input signals to be fed back in to the control system under test. HIL simulation can provide realistic evaluation of an embedded system while minimizing development time and cost [3].

Being able to develop control algorithms in a simulation reduces the risk of a crash when the first real test flights are performed. It also allows pilots to develop a feel for the aircraft before they fly it. Another benefit is the ability to analyse the physical design of the autogyro and its effects on stability before any real test flight data is available.

II. Theory

A. Autogyro Control

There are three controls on an autogyro; throttle, rudder and joystick. The throttle controls the power of the propeller. The yaw of the craft is controlled by the rudder and the joystick governs the pitch and roll of the rotor, which in turn can be used to control the pitch and roll of the autogyro itself. Smooth turning is achieved by a mix of rotor roll and rudder yaw.

Techniques for piloting autogyros differ from those used for flying other types of aircraft, though the handling and controls are similar to a fixed-wing aeroplane [4]. One specific difference is the way the altitude of the craft is controlled using a balance of throttle and pitch adjustments [2]. Speed should be kept at a constant rate by adusting the pitch, there will be an optimum speed depending on whether the aircraft is climbing, flying at a steady altitude or in a powerless glide. Climb rate is then controlled by changing the throttle. This avoids a negative pitch attitude which can cause severe instability in autogyros.

One characteristic of autogyro flight dynamics is the occurrence of a low frequency pitch oscillation, known as the phugoid mode [5]. For this reason it is advisable for the pitch to be held as steady as possible. A phenomenon reffered to as 'Pilot Induced Oscillation' can occur when an over-correcting input from the pilot on the joystick can cause a positive feedback loop and result in runaway pitch oscillations [6]. This can result in complete loss of control of the aircraft.

Several roll moments affect autogyro stability. There is a moment in the roll axis due to propeller rotation, which can

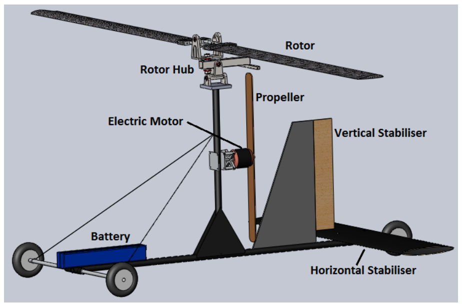

Fig. 1. Durham autogyro design schematic

cause a constant roll offset. In addition the rotor can give rise to a gyroscopic precession moment in the roll axis when it is tilted forward or backward in the pitch axis [4].

B. *Durham University Autogyro Design*

The Durham University Autogyro UAV is an ongoing design project, so currently there is no version which can be considered a final, authoritative design, nor has there been a version of the UAV capable of a flight test. The version of the physical design used throughout this project was taken from CAD models developed this academic year by Charlie Paterson. A labelled diagram can be seen in figure 1.

C. *PID Controllers*

Proportional-integral-derivative (PID) control is a method of negative feedback control. Properly tuned PID controllers give a rapid response without causing undue oscillation. There are three separate terms in a PID controller, referred to as the proportional, integral and derivative terms [7]. All three are calculated from the error between the current position and the desired setpoint, defined as $y_s(t) - y(t)$, where $y_s(t)$ is the setpoint and $y(t)$ is the current system output. The proportional term is directly proportional to the current error:

$$u_p(t) = P(y_s(t) - y(t)), \tag{1}$$

The integral term is found from a summation of previous errors:

$$u_i(t) = \frac{I}{\tau_i}\int_0^t (y_s(t) - y(t))d\tau, \tag{2}$$

where τ_i represents the sampling period for integration.

The derivative term is the rate of change of the error term:

$$u_d(t) = D\tau_d\frac{(y_s(t) - y(t))}{dt}, \tag{3}$$

τ_d represents the sampling period for differentiation.

These terms are summed together and subtracted from the current control output to the servos, to provide a negative feedback control system. It is implemented using a control loop running on a microprocessor [8].

The PID controller can be tuned by modifying the constants P, I and D to change the magnitude of the effect of each term. A simple way to think of the effect of each term is that the proportional term controls the magnitude of the response, the integral term can prevent steady state error (constant offsets) and the derivative term provides a damping effect which can prevent oscillation around the setpoint.

D. *Ardupilot*

Ardupilot is an autopilot system for UAVs, based on an Arduino microprocessor board [9]. The firmware is open-source, and can be edited using the Arduino compiler. Arduplane, the firmware package designed for fixed-wing aircraft, was used as a starting point for control algorithm development in this project. The Ardupilot Mega board used has a separate oilpan board with an IMU, GPS and other sensor sub-systems.

The Ardupilot autopilot system uses cascaded PID controllers to adjust the aircraft servos and actuators, meaning an outer PID loop controls the setpoint of an inner PID loop. PID controllers are used for both navigation and stability [10].

III. Hardware-in-the-loop Simulation Setup

A. *FlightGear*

FlightGear, an open source flight simulator package, was used for the HIL simulation. It was chosen over commercial alternatives such as XPlane as it gives greater flexibility over implementation and aircraft model design.

FlightGear is comprised of two parts; the flight environment and the flight dynamics model. The former replicates the real world conditions of the flight environment, giving scenery, position in geodesic coordinates, weather conditions and a visual representation of the simulation. The flight dynamics model (FDM) describes the dynamic behaviour of the aircraft and its response to control inputs. This means it is the most important part of creating a realistic testing environment for the control system. One advantage of FlightGear is that the dynamic behaviour of the craft is completely separate from the visual model used to represent its appearance. This meant development effort could be concentrated on the FDM rather than the visual model, which is of low importance with regards to investigating aircraft control.

FlightGear supports several different FDMs, however the FDM *YASim* was chosen as it supports rotorcraft and includes examples of autogyro models. All parameters which affect the dynamic behaviour of the aircraft are specified in a YASim XML document.

B. *Ardupilot Mission Planner*

One advantage of using the Ardupilot control system is the versatile groundstation software available. The 'Ardupilot Mission (APM) Planner' is a groundstation program developed solely for use with Ardupilot. It was selected for this project due to its built-in support for HIL simulation, as well as other capabilities such as the ability to upload custom firmware to the Ardupilot board, vary parameters and settings, set waypoint coordinates on the board memory and receive telemetry in real-time.

The APM planner runs on a PC and connects using a serial link over USB cable to the Ardupilot Mega board. It communicates using the MAVLink (Micro Air Vehicle)

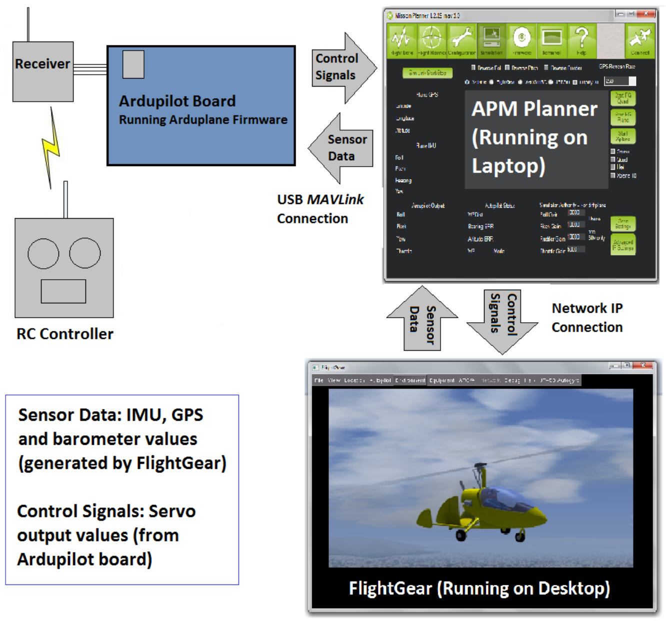
Receiver
Ardupilot Board
Running Arduplane Firmware
Control Signals
Sensor Data
USB MAVLink Connection
APM Planner (Running on Laptop)
RC Controller
Sensor Data
Control Signals
Network IP Connection
Sensor Data: IMU, GPS and barometer values (generated by FlightGear)
Control Signals: Servo output values (from Ardupilot board)
FlightGear (Running on Desktop)

Fig. 2. HIL system block diagram

protocol [11], which was designed specifically for real-time communication with light-weight UAVs and is emerging as the industry standard [1].

C. *HIL Simulation Configuration*

HIL simulation was first implemented with the Ardupilot board controlling a model of a full-sized autogyro, the 'Hornet', which had already been developed for FlightGear. The complete HIL system is shown in figure 2. As can be seen the flight simulation runs on the desktop. All communication between the Ardupilot control board and the simulation goes through the APM planner software running on the laptop. Control outputs, such as servo values, from the Ardupilot board are sent to FlightGear, which models the behaviour of the aircraft as well as providing a visual representation of the simulation. The position data generated by the flight simulator, such as GPS and IMU attitude values, are then sent back to the Ardupilot board through the APM planner. The whole HIL environment runs and communicates in real time so that the Ardupilot board can operate as though it were controlling a real aircraft.

Several steps were necessary to set up and run the simulation. The Arduplane firmware code was compiled in HIL mode using the Arduino compiler and uploaded to the Ardupilot board as a hexadecimal file. The flight simulator was run on a separate computer to make use of better graphics processing capabilities, although it is possible for it to be run on the same machine as the APM Planner by communicating using the loopback IP address (127.0.0.1). They communicate over a UDP protocol connection handled by the APM Planner. It was necessary to modify the firewalls of both machines to allow this connection through. A batch file was created to launch FlightGear on the desktop, in which the IP addresses for the input and output UDP sockets were specified; in this case the IP address of the laptop. This batch file was updated after every reboot of the system in case of a change of IP address. The batch file allows the aircraft model to be specified, as well as weather, light conditions and other environmental variables. Adverse weather conditions such as wind and turbulence were removed during development, as these factors were not deemed to be fundamental to control algorithm development, however they could be the subject of future investigation.

A MAVLink protocol configuration file was added to the FlightGear folder on the desktop. Finally inside the APM planner options were selected for running a HIL simulation with FlightGear and the IP address of the simulator PC and the UDP socket numbers were entered.

The Ardupilot board was prepared for testing by connecting the radio receiver and turning on the RC transmitter. The board was physically connected to the laptop by USB cable. Once FlightGear had been launched on the desktop the simulation was started from the APM Planner.

IV. Dynamic Model Development and Stability

The Arduplane control firmware running on the Ardupilot board consists of an autopilot system with four main flight modes; 'manual', 'stabilise', 'fly-by-wire' and 'autopilot'. Manual mode allows the human pilot complete control over the craft. Stabilise mode is identical except that the Ardupilot board uses servo PID control and data from the inertial measurement unit (IMU) to keep the aircraft level automatically. 'Fly-by-wire' modes uses the same stabilisation PIDs but the Ardupilot system also controls the attitude and airspeed, with the pilot responsible only for the overall direction of the aircraft. In autopilot mode complete control is handed over to the Ardupilot board, it uses the same attitude and airspeed control loops but with the addition of navigation control. The navigation pitch, roll and rudder values are set by navigation PID controllers, which in turn control the setpoints of the stabilisation PIDs; an example of 'cascaded PID controllers'. In autopilot mode the aircraft navigates around a pre-determined route which can be modified with the APM planner software. The only mode not used in this project was 'Fly-by-wire', as the development was focused on the stabilisation and navigation algorithms, which can be isolated using the stabilise and autopilot modes. However 'Fly-by-wire' mode could provide an effective way of controlling an autogyro UAV and therefore merits further research and development.

The aim of this part of the project was to develop a flight dynamics model of the Durham autogyro which accurately replicated the dimensions and characteristics of the Durham UAV design, as seen in figure 1, and was capable of steady flight in the HIL simulation, aided by control inputs from the Ardupilot board in stabilise mode.

A. *Flight Dynamics Model*

An aircraft model for FlightGear consists of a 3D model of the physical appearance, a flight dynamics model and scripting files in the FlightGear scripting language *nasal* which describe aircraft behaviour such as reaction to control inputs.

A YASim FDM is described in an XML document. Components which affect the dynamic behaviour of the aircraft, such as wings, rotors, propellers and fuselages, are entered as XML elements. Positions are described in metres relative to an arbitrary origin, in this case set to be the nose of the craft. A YASim XML file was created for the Durham Autogyro UAV based on dimensions and weights taken from the solidworks model. Since no flight tests have been performed there is currently no empirical data regarding the performance of the Durham UAV design. The YASim XML format allows for easy editing of parameters, meaning the Durham autogyro FDM can be changed in future to incorporate more accurate data, or to investigate the effect of varying certain parameters as will be detailed later in this report.

The rotor element allows various rotor parameters to be specified, such as blade dimensions, weight per blade, number of blades, chord, taper, minimum and maximum collective angle, flap angles, drag factors and lift coefficients among others (see appendix A for a full list). In the absence of sufficient wind tunnel data for the autogyro most of the rotor parameters were based on the ‘Hornet’ autogyro YASim model, however these could be modified to make use of data from wind tunnel experiments or flight tests. YASim also allows the slew rates of the control inputs to be limited. These limits were obtained from the data sheet of the Firgelli L12 actuator used in the real Durham autogyro design.

The stabilisers were entered as horizontal and vertical stabiliser elements. Parameters such as length, taper, chord and sweep angle were specified based on the solidworks model dimensions. A flap was added to the vertical stabiliser which was linked to the rudder control input.

The body of an aircraft in YASim is created from tubular ‘fuselage’ structures, which have both a mass distribution as well as aerodynamic properties. Ballast weights were added to represent various components of the Durham autogyro such as the propeller motor and the battery. The overall weight is specified at the top of the file, the ballast elements merely redistribute this weight so the YASim solver can correctly calculate the moments of inertia around different axes.

The propeller mass and radius were specified. The moment of inertia of the propeller was entered based on a value calculated in solidworks. The propeller power was taken from the data sheet of the electric motor used, but reduced on the assumption that power would be lost to inefficiencies in a real model and on the premise that it was better to underestimate any factors which would improve the model’s performance.

The creation of the YASim FDM was an iterative process, with continual editing of the model after successive flights. The result was an FDM which replicates the dimensions of the solidworks model, however certain factors had to be changed as initially the model was highly unstable. In the first few flights a high moment in the roll axis was observed immediately on take-off, causing the autogyro to crash. It was discovered that this was due to the gyroscopic moment of the propeller, as the effect could be reduced by lowering the propeller moment of inertia.

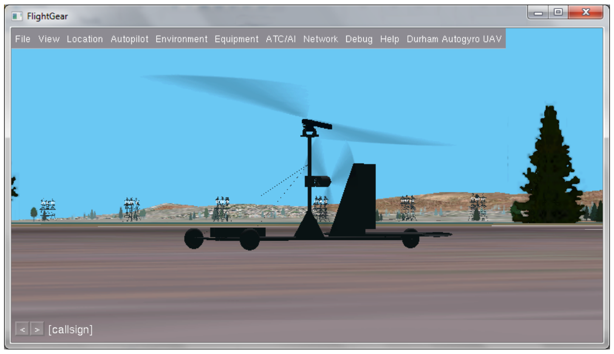
FlightGear
File View Location Autopilot Environment Equipment ATC/AI Network Debug Help Durham Autogyro UAV
[callsign]

Fig. 3. Model of Durham University autogyro taking off from a runway in FlightGear

There are two ways of explaining the highly destabilising nature of this propeller moment; one possibility is that the YASim FDM is exaggerating the propeller moment due to inaccuracy in the calculations at a small scale, since the autogyro’s roll moment of inertia will be small compared to a typical YASim model. It could alternatively be attributed to the structure of the Durham autogyro design. Most of the mass is distributed close to the x axis running from the front to rear of the craft, which could result in an insufficient resistance to roll moments. Either way the simulation has flagged up a potential design problem which may result in instability in a real-world flight of the autogyro. It can be said with some degree of confidence that, even if the simulator is exaggerating the scale of the effect, this is an issue which requires further design consideration. This phemomenon was effectively preventing any form of sustained flight, so it was deemed acceptable to remove it for the rest of the project by setting a flag in the YASim model indicating that the propeller was a contra-rotating design, therefore cancelling out any propeller-induced gyroscopic moment.

A 3D visual model was created by converting the Solidworks model to a file format readable by FlightGear; ‘.ac’. It can be seen in action in figure 3. Even though the appearance of the aircraft in FlightGear is completely separate from its dynamic behaviour, having an animated visual representation of the aircraft is useful for analysing its performance, not to mention making it easier to pilot.

B. Servo PID Tuning

When the model of the Durham autogyro UAV was first flown in stabilise mode it was apparent that the default stabilisation PID settings were not sufficient for stable flight, as the pitch and roll behaviour was unstable and level flight was not possible for any sustained period of time. It was necessary to tune these PID coefficients to achieve stable flight. This was done using heuristic methods during repeated simulated flights. The mission planner software allows for PID coefficients, among other parameters, to be altered while the simulation is running, so the effects of any change can be seen immediately. The coefficient of the proportional term, P, (from equation 1) was of primary concern, as this determines the magnitude of the negative feedback. For both the roll and pitch servo

PIDs the value of P was adjusted until the system gave a fast stabilising response. As P was increased oscillations around the setpoint were observed, above a certain value the aircraft was destabilised by these oscillations, resulting in an upper bound for the value of 'P'. The value of 'I' was kept low and used only to counter the constant offset observed in the roll axis. The 'D' coefficient was used to reduce oscillation around the setpoint, effectively increasing the damping of the system, but was kept low relative to the 'P' coefficient.

The PID settings arrived at in this project gave satisfactory stable flight, however this is effectively a 3-dimensional optimisation problem with the possibility of complicated interactions between the terms, so these settings are not fully optimised. More sophisticated tuning methods, such as automatic PID tuning, are possible [12], however since stable flight has already been achieved this would provide only marginal improvements over the current tuning method and requires significant changes to the simulation system. Attitude information would need to be output in real time and linked to a program capable of modifying the PID settings automatically. A set of libraries called *Aerospace Blockset* are provided by Mathworks to link FlightGear to MATLAB Simulink [13], which could be used for automatic tuning. Automatic control parameter optimisation in this way was beyond the scope of this project but could be an area for future development.

C. Stability Experiment Procedure

Experiments were performed in order to illustrate the effects on stability of varying both the PID settings and the autogyro structure. The simulation consists of free, unconstrained flight, where the aircraft position and velocity depend on numerous different inputs and conditions at any point in time. This presented a significant challenge in devising experiments which were repeatable and produced results where the effect of a single parameter or input was identifiable. Two methods of generating repeatable results were used. The first was a takeoff experiment; the conditions immediately after the autogyro has lifted off from the runway were found to be acceptably repeatable, as the control inputs up until this point can be replicated exactly. When varying a parameter separate take-off manoeuvres were performed for each value. While travelling along the runway the throttle input was set to maximum and the pitch was pulled back fully, allowing the rotor to start autorotating, as is standard procedure for an autogyro takeoff. As soon as the autogyro had left the runway the rotor joystick was returned to a neutral position and the throttle was kept at maximum. The simulator was allowed to run for several seconds after take-off to record the immediate response of the control system to the forces resulting from leaving the ground; primarily the gyroscopic precession moment of the rotor which causes the aircraft to roll to the right.

The second experiment measured the impulse response of the aircraft in both the pitch and roll axes. This 'impulse response' experiment was performed after the aircraft had taken off and was in stable, level flight. A sharp impulse input was applied on the relevant axis by pushing the rotor joystick to its limit then quickly restoring it to a neutral state,

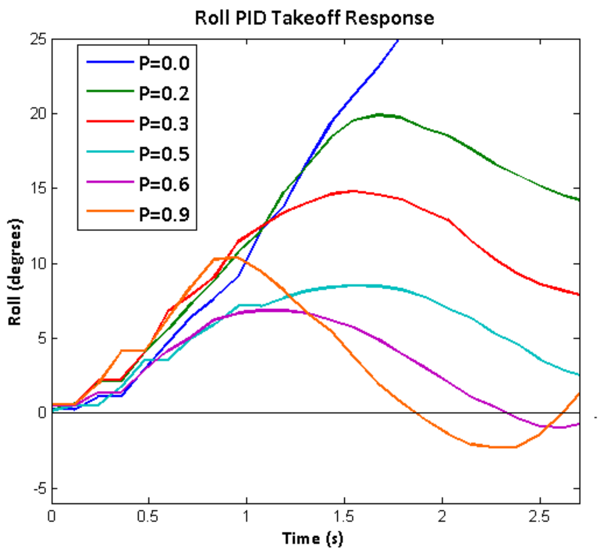

Fig. 4. Roll response immediately after takeoff. Each line represents a different value of 'P' for the roll servo PID controller. The tendency for the aircraft to roll to the right (caused by the rotor gyroscopic precession moment) can be seen clearly as a positive roll value. The value of 'P' determines the amount of negative control input feedback applied.

giving an approximation of the 'Dirac Delta' function. The response is then an approximation of the impulse response of the system in one axis, however it is not possible to completely isolate this from effects in other axes. In stable, level flight the aircraft can be approximated as a linear, time-invariant (LTI) system, and so this impulse response represents the transfer function of the system in the respective axis. This is useful in predicting the response of the system to external inputs. For an aircraft this could consist of destabilising gusts of wind, turbulence or rapidly changing inputs from navigation controllers. Understanding the response to these factors is important for ensuring the stability of the aircraft under different conditions, and could be used for future optimisation of control parameters.

The mission planner software allows telemetry logs to be read from the Ardupilot board. These were exported as comma-separated-value files, from which the relevant results (arrays of roll and pitch values with corresponding time stamps) were extracted using Microsoft Excel. These extracted arrays were processed and graphed in MATLAB. The repeatability of the experiments was confirmed by graphing repeated results for the same values and ensuring there was a high correlation.

D. Stability Experiment Results

Figure 4 shows that the value of 'P' corresponds directly with the speed of the corrective response to return the autogyro to level. With a value of 0.0, equivalent to no PID feedback, the aircraft is unstable; the roll response is unbounded. For values higher than 0.5 an overshoot is observed, with oscillations around a level attitude.

The tendency for an oscillatory response is clearer in figure 5, which shows the results of a sharp impulse in the roll axis.

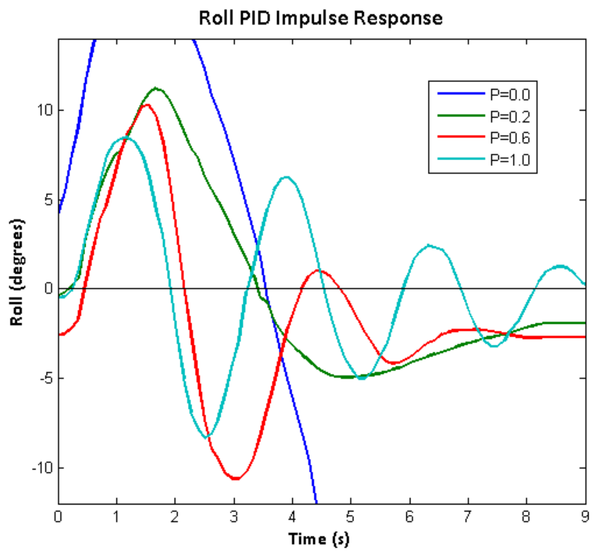

Fig. 5. Roll response immediately after an impulse input in the roll axis, varying the ‘P’ coefficient of the roll servo PID controller

An autogyro can be thought of as a pendulum hanging from its rotor. In this graph some of the inherent stability afforded by this structure is apparent, as even for a value of P=0.0 the aircraft manages to self-right before rolling in an uncontrolled manner to the opposite direction. For high values of ‘P’ it takes the oscillations a long time, roughly 10 seconds, to decay. For low values it can be observed that there is a constant offset after the response has stabilised, this can be removed by altering the ‘I’ term of the PID controller and does not detract from the stability of the craft, so this is not a major concern here.

It was found through repeated flights that the most stable characteristics were achieved with a relatively slow response time to remove the tendency for oscillations, which can cause a destabilising effect. A value of P=0.3 for the P term of the roll servo PID was found to give the best performance and was used for the rest of the project.

The much longer period phugoid oscillations in the pitch axis can be seen in figure 6. In this case the oscillations are impossible to remove completely, even at low ‘P’ values, as they are a fundamental characteristic of autogyro flight. It should be noted that the values of ‘P’ required for stability are much higher than for the roll servo PID by around an order of magnitude. This is because instead of exacerbating any oscillations a strong PID response reduces the magnitude of the phugoid pitch oscillations and prevents their destabilising effects. Any downward pitch is to be avoided for stable autogyro flight [2], so it is important for the pitch servo PID to reduce the tendency to pitch down. It was found in the navigation experiments, detailed later in this report, that successful autonomous flight in autopilot mode was not possible without a high value of ‘P’ in the pitch servo PID (in the region of between 2.0 and 3.0).

The effects of the physical structure on stability were also investigated. The period and magnitude of the phugoid

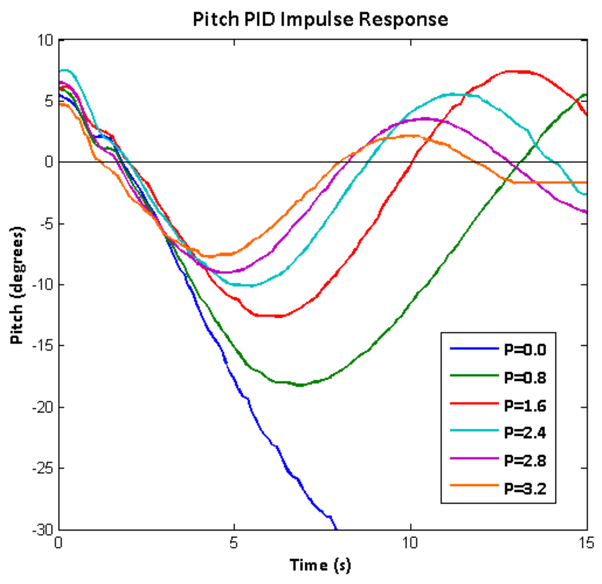

Fig. 6. Pitch response immediately after a downward impulse input in the pitch axis. This shows that increasing the value of ‘P’ reduces the magnitude of the long period pitch oscillation

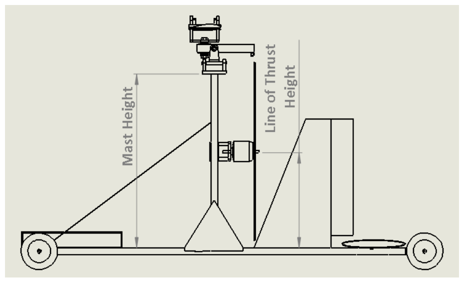

Fig. 7. Physical parameters varied to investigate stability

oscillations are a function of several parameters; the height of the mast, the height of the centre of gravity and the height of the line of thrust of the propeller, shown in figure 7. The height of the rotor mast was varied and results were taken. It is hard to isolate the effect of any one of these factors, since increasing the mast length increases the height of the rotor, which in turn increases the height of the centre of gravity relative to the line of thrust, since the rotor is one of the heaviest parts of the autogyro.

A frequency spectrum of the responses to a pitch impulse for different values of mast height showed a clear peak of around 0.075Hz for all values, so mast height did not appear to affect the period of pitch oscillations. However the magnitude of the pitch oscillations appeared to increase proportional to mast height. An important conclusion drawn from this experiment was that the PID controllers were observed to have a much larger effect on the stability of the aircraft than physical design parameters.

This project is concerned with developing a control system for the autogyro and less so with the mechanical design of the aircraft. However the mast height experiment demonstrated that this HIL simulation setup can also be used to investigate the effects of various physical design parameters on the dynamic behaviour of the autogyro.

V. Navigation Control and Firmware Development

A. Firmware Modifications

Autopilot algorithms were developed by modifying the *Arduplane* firmware designed for control of fixed wing aircraft. The algorithms controlling roll and yaw were transferrable to autogyros, as both aeroplanes and autogyros turn using a mix of rudder yaw and roll. It should be noted that both the rudder mix coefficient and the PID parameters need to be tuned for autogyro flight, but they are not not hard-coded into the firmware and can be modified at run-time. The main difference with piloting an autogyro lies in the way throttle, pitch and speed are adjusted relative to each other to give precise altitude control and stable pitch behaviour [2].

The Arduplane firmware is an Arduino project written in the Arduino programming language with supplementary C++ libraries. Two key functions to calculate the target pitch and throttle values, located in the ‘attitude’ arduino file, were relevant to autogyro control. The instantaneous errors in airspeed and altitude, relative to current target values, are calculated in a separate arduino file called ‘navigation’. The altitude is a combination of barometer and GPS readings, the airspeed is taken from an airspeed sensor. PID controllers are implemented with a C++ library, in the Arduino code they are represented as a simple function which takes an instantaneous error value as the only input, and returns the sum of the three individual PID terms.

In the case of the unmodified, or ‘stock’, Arduplane firmware designed for fixed-wing aircraft the altitude is controlled by adjusting the pitch as well as the throttle. The altitude error is the instantaneous difference between the current altitude and the altitude of the target waypoint. The default method for calculating the target pitch (which is used as the setpoint for the stabilisation pitch PID in a cascaded PID configuration) is by passing the altitude error in as an argument to the navigation pitch PID function. A simplified pseudocode version is shown in equation 4. The throttle is calculated from a PID function where the input is the instantaneous *energy* error; the sum of the altitude error and the airspeed error. The throttle is also adjusted depending on the pitch input by a pitch-to-throttle mix ratio. A simplified representation of these calculations can be seen in equations 5 and 6. ‘Cruise’ represents the cruise throttle value.

$$TargetPitch = PID(AltitudeError); \tag{4}$$

$$Throttle = Cruise + PID(EnergyError); \tag{5}$$

$$Throttle = Throttle + Pitch * K_{pitch2throttle}; \tag{6}$$

However for an autogyro it is recommended that altitude is controlled solely by throttle adjustment, allowing the aircraft to ‘float’ up and down at a constant pitch [2], which avoids pitch fluctuations and dangerous negative pitch values. The pitch is only adjusted to control the airspeed, which should be kept constant at a value which depends on whether the autogyro is climbing, sinking, cruising or in a powerless glide. Pitching up slows the aircraft and pitching down speeds it up. There is a single airspeed at which the best performance is achieved for each of these flight scenarios. Both the functions mentioned above were re-written. The pitch was calculated based solely on the airspeed error, as seen in equation 7. The Arduino code for this function can be seen in Appendix B, named ‘calc_nav_pitch()’.

$$TargetPitch = -PID(AirspeedError); \tag{7}$$

The altitude control could have been implemented simply by passing the altitude error into the throttle PID. However this method of throttle calculation was tested and did not give satisfactory results; it caused the throttle to swing immediately to the maximum or minimum value, depending on whether the altitude error was positive or negative. In the stock firmware the altitude error is ‘ramped’, meaning it is limited depending on the distance from the next waypoint. However this still did not prevent the size of the altitude error maxing out the throttle in either direction.

To counter this effect a novel approach was taken for the throttle control, in which the error in *climb rate* was passed into the throttle PID rather than the altitude error. A function was written to calculate the climb rate by differentiating the instantaneous altitude values, using a smooth, noise-robust differentiator filter [14]. The resulting throttle calculation can be seen below in equation 8. The code for the differentiator function and the throttle calculation can be seen in Appendix B. The target climb rate was calculated by multiplying the altitude error by a constant coefficient, but constrained by maximum and minimum limits. The climb rate error was calculated as the difference between this target climb rate and the instantaneous climb rate from the differentiator function.

$$Throttle = Cruise + PID(ClimbRateError); \tag{8}$$

This method produced superior results; the throttle stabilised at a value depending on the desired rate of climb rather than saturating at either the maximum or minimum limits. It is a more intuitive way of controlling altitude and allows the climb and sink rates to be specified explicitly. Currently these are hard-coded into the firmware but in future they could be implemented as parameters which can be adjusted at run-time.

One further modification to the firmware was to fix a bug which prevented the altitude values being read when the board was operating in HIL simulation mode. This fix was necessary for altitude control to work.

To achieve successful navigation in autopilot mode several parameters had to be tuned correctly. There were three relevant PID controllers; the navigation roll PID, the navigation pitch PID and the throttle/altitude PID, all of which were tuned in the same way as the stabilisation PIDs. This resulted in smooth navigation to waypoints and avoided ‘snaking’ between waypoint locations. Other parameters which affected the navigation performance included the maximum and minimum climb rates, the coefficient which relates the altitude error to the climb rate error, the target airspeed, the maximum, minimum and cruise throttle values and the roll-to-rudder mix coefficient.

The autopilot algorithms were initially developed with test flights of two separate full-size autogyro models; the 'Hornet' and the 'JT-5B'. At a later stage further development work was done with the model of the Durham autogyro UAV. The fact that these algorithms have been demonstrated to function well for three different autogyros, and that the stock firmware doesn't, gives confidence that these algorithms are appropriate for autogyro control.

B. Navigation Experiment Results

Several flights were performed to compare the navigation performance of the firmware developed in this project (the 'modified' firmware) with the 'stock' Arduplane firmware (which was unmodified apart from fixing the bug mentioned above). The model of the Durham autogyro UAV was used. In each flight the UAV was taken off in stabilise mode then switched to autopilot to navigate its way around a course of 4 waypoints at different altitudes. Altitude and pitch results were extracted from the telemetry logs.

The improvement in altitude performance can be seen clearly in figure 8. The top graph shows the altitude and pitch performance of the stock algorithms while navigating to the four waypoints. As described previously the altitude is being controlled by the pitch. This causes pitch oscillations which appear to grow in magnitude throughout the duration of the flight. Several flights were run with these stock algorithms, in the majority of these flights eventually the pitch oscillations grew so large that the aircraft was completely destabilised and all control was lost, resulting in a crash.

The lower graph shows the performance of the algorithms developed in this project. Practically no pitch oscillations are observed when in autopilot mode (the mode was switched to autopilot after around 100 seconds, the period where the pilot is in control is indicated by the circle on the graph). The system is capable of navigating to the correct altitude at a constant climb/sink rate, then remaining at this altitude until the waypoint is reached. Some overshoot is observed before arriving at the desired altitude, however no oscillations occur beyond this initial overshoot. The pitch remains close to constant, with a small amount of noise, for the whole flight. Negative pitch values are avoided entirely. For the latest iteration of firmware and tuned parameters no crashes were observed.

The throttle values, although not shown here, also differed significantly between the two firmware versions. For the stock version the throttle fluctuated rapidly between its maximum and minimum values for the whole flight, whereas in the modified firmware the throttle moved very gradually and changed only to adjust the climb rate. This ensured that the autogyro was kept within its *flight envelope*; an acceptable range of throttle, pitch and altitude values which are considered to constitute safe flight.

In both flights a brief period of manual control in stabilise mode can be seen at the start, circled on the graphs. Several large pitch oscillations are present, which is an important observation as it suggests that the algorithms developed in this project are actually better at flying the Durham autogyro UAV than a human pilot. This is because they can be tuned to eliminate any pilot induced pitch oscillations and are more able to maintain a steady pitch.

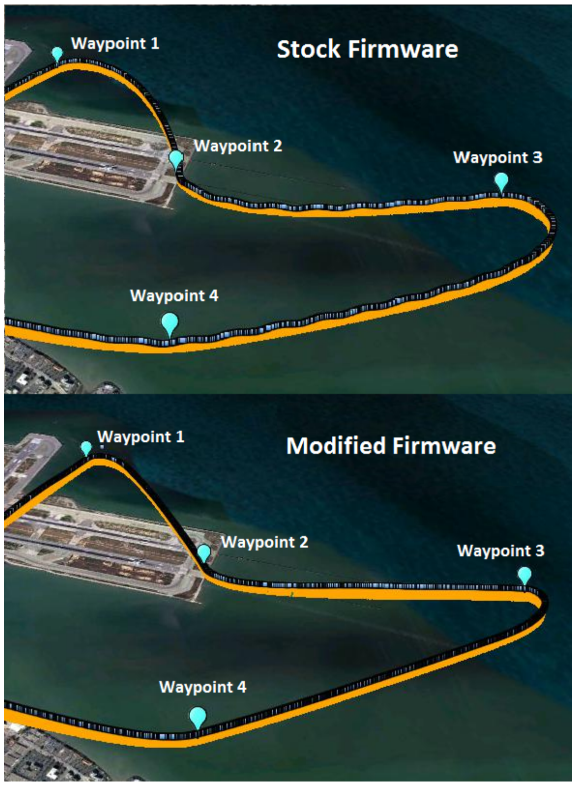

Fig. 9. Comparison of turning and navigation performance for both the stock (unmodified) Arduplane algorithms and the algorithms developed in this project.

A comparison of the turning navigation performance is shown in figure 9, created using Google Earth to map the routes of two flights around the same waypoints. The modified firmware shows an improvement; the turns are sharper, but it is a more modest improvement than the difference in altitude performance, since both firmware versions use the same algorithms to calculate the roll and yaw required for turning. The change can be attributed to the fact that the modified firmware allows for much more stable flight, which lets the roll navigation PID operate more effectively.

To demonstrate the effect of PID tuning and other parameters on autonomous flight several attempts were made to fly the aircraft in autopilot mode (using the modified firmware) with the default Arduplane parameter values. In every flight under these conditions the aircraft proved to be completely unstable and crashed within moments of being switched to autopilot mode, showing that correct tuning of parameters is critical for successful autonomous flight.

One proviso attached to these navigation results is that in this HIL simulation environment the position data the board receives is completely smooth and accurate, which would not be the case in a real flight. For example real GPS position data is generally accurate to around 5m [15]. However this discrepancy is not considered to be crucial to the navigation performance. The navigation algorithms do not rely on highly

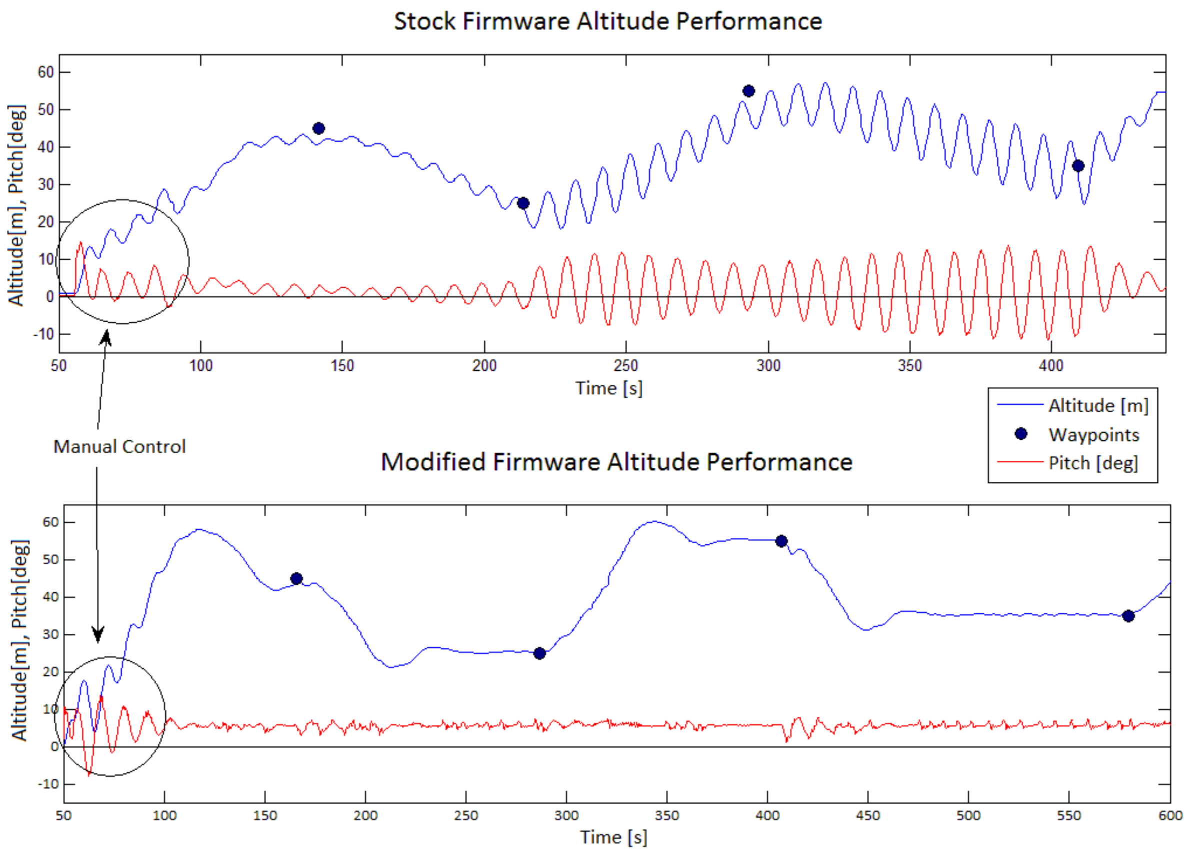

Fig. 8. Comparison of altitude control for both the stock (unmodified) Arduplane algorithms and the algorithms developed in this project.

accurate position data, in fact only large errors passed into the PIDs will have any effect at all, and a waypoint is considered to be reached if the aircraft is within a 15m radius of it. The differentiating algorithm used to calculate climb rate is also designed to be noise robust. The position data is updated at the same frequency regardless of whether the board is in a HIL simulation or in real-life operation.

Short video clips were captured of the autogyro turning the same corner with the control system running both the modified and stock firmware, which can be accessed at the following web address: tinyurl.com/ctnykoq

The difference in pitch stability between the two firmware versions is clearly observable.

VI. Simulation Accuracy

A central question surrounding this project is the level of confidence that can be placed in the simulation, and to what extent it replicates the performance of the real design, were it to be flown. It should be noted that the main concern here is not that it replicates the dynamic behaviour exactly, but that it is sufficiently accurate to be *useful* for control system development. With that in mind there are several considerations which lend credence to this simulation.

The accuracy of the simulation rests on the YASim FDM, so it is necessary to gain an insight into how it functions. YASim is an iterative solver, which runs when FlightGear starts up to produce a number of values for a given aircraft model, including the centre of gravity and an inertia tensor which describes the rotational behaviour of the aircraft in 3 dimensions, calculated from the mass distribution of the elements in the XML document [16]. Rotor performance coefficients are calculated, such as lift and drag factors. A body drag factor is also given for the physical structure of the aircraft. While the simulation is running YASim repeatedly calculates the instantaneous velocities of the aircraft, both rotational and translational, depending on inputs from the simulation environment and the numerical model of the aircraft it has generated.

YASim allows almost every conceivable rotor parameter to be specified (for an exhaustive list see appendix B). However some of these parameters were either based on YASim default values or taken from the existing examples of autogyro models, and do not accurately describe the rotor of the Durham autogyro UAV. With the propeller, even though the nominal power output was known, there was insufficient data on what rpm and power it would achieve in real operation. On the other hand the dimensions and mass distribution could be specified in the XML document with near complete conformity to the solidworks model.

In this project the most important factor is the dynamic stability of the model as determined by its structure, mass distribution and the directions of various lift and thrust forces.

This is because understanding the dynamic stability is crucial to the design of the stabilisation algorithms needed for a successful control system. The accuracy of the stability performance rests primarily on correct calculation of the model's moments of inertia in each axis, which are described exactly by YASim in the form of an inertia tensor, and were in turn calculated from dimensions and mass distributions specified with a large degree of confidence in the XML file.

The areas where the model is much less accurate are the lift characteristics of the rotor and the thrust characteristics of the propeller. However there is only uncertainty around the *magnitude* of the forces these produce, whereas the directions of these forces are well determined, as they are dependent only on the model dimensions. It is the directions of these vectors which are most relevant to the overall stability, and so even though the rotor and propeller characteristics are clear areas for improvement, the current lack of accuracy does not undermine the usefulness of the model as a way of testing the stability of the autogyro.

FlightGear has already been used as a development tool in many engineering design projects in both professional and academic environments, the most relevant being a project at Simon Fraser University which used FlightGear to develop autonomous control algorithms for a UAV [17]. FlightGear is also used to train pilots, which backs up the idea that the flight dynamics models it employs are capable of replicating real aircraft performance to a satisfactory extent. This validates FlightGear as the correct tool for the simulation. The model still needs further verification that the performance is similar to the real design of the Durham UAV. FlightGear provides the flexibility to achieve this, both in the YASim FDM or with the possibility of creating a custom FDM, by allowing data obtained from real-world analysis, such as wind tunnel results, to be incorporated in future.

The Durham UAV model exhibits several behaviours which are characteristic of real autogyro flight, such as the tendency for pilot-induced pitch oscillations and the low cruise speed achievable without losing altitude. This can be taken as further confirmation that the simulation reliably simulates the dynamics of an autogyro.

VII. Transition to Real UAV

The ultimate aim for this control system is for it to be used for autonomous control of the real Durham autogyro UAV when it is first flown. In order to implement this project's navigation algorithms in real life an airspeed sensor and compass would need to be purchased, as these are not included in the current Ardupilot setup. Another recommendation is the purchase of the updated Ardupilot Mega 2 board. This is necessary as the current APM1 board has insufficient onboard memory to load the latest firmware when compiled for real flight, it is only capable of HIL simulation as compiling the firmware in HIL mode results in a smaller hexadecimal file.

In addition to these hardware changes it will be necessary to modify the firmware slightly to accommodate the rotor actuator configuration on the real autogyro. In the HIL configuration the rotor control signals are sent from the Ardupilot board as separate roll and pitch values. However in the real autogyro there are only two rotor actuators; a left and right actuator. A simple calculation would need to be performed to convert the roll/pitch output values to left/right actuator values. It is suggested that all algorithms deal with roll/pitch values internally and the conversion is made at the output. A simple conversion calculation is proposed in equations 9 and 10.

$$LeftActuator = Roll - Pitch; \tag{9}$$

$$RightActuator = -Roll - Pitch; \tag{10}$$

VIII. Conclusions

Stable autonomous flight and successful navigation to waypoints, with precise altitude control, were achieved using a model of the Durham autogyro UAV inside a HIL simulation.

Simulated flights to test stability and navigation control gave several significant findings:

- Tuning of several control parameters, especially PID coefficients, is critical for stable flight. In fact the PID controllers appear to have a much larger impact on stability than variations in certain physical parameters.
- Structural design issues have been flagged up, meriting further investigation, such as the moment of inertia in the roll axis being insufficient to prevent the aircraft being destabilised by propeller rotation.
- Some form of stabilisation algorithms will be necessary when the real Durham autogyro UAV is flown for the first time, emphasising the need for prior HIL simulation testing of the control system.
- Successful autonomous flight requires stabilising PID feedback in the pitch axis, with a high value of 'P', to overcome the inherent pitch instability of autogyro flight.
- The autopilot control system developed in this project is capable of out-performing a human pilot in flying a lightweight autogyro UAV in certain scenarios in the HIL simulation, specifically in minimising pilot-induced oscillations and maintaining a stable pitch.
- There is a need for pitch and altitude control algorithms designed specifically for autogyros, which was demonstrated by the large improvement in stability gained from using the algorithms developed in this project. However other control algorithms designed for fixed-wing aircraft, such as for roll and yaw control, are transferrable.

These results need to be verified outside of this simulation, there is also significant potential for improving the accuracy of the flight dynamics model, for example by incorporating wind tunnel measurements from the real autogyro design. However results suggest the simulation model is accurate enough to make it a useful method for testing control algorithms. This project is a large step towards being able to implement this control system in real world flight. It also serves as proof of concept that HIL simulation is a valuable design tool in the development of both the control system and the structural design of a lightweight autogyro UAV.

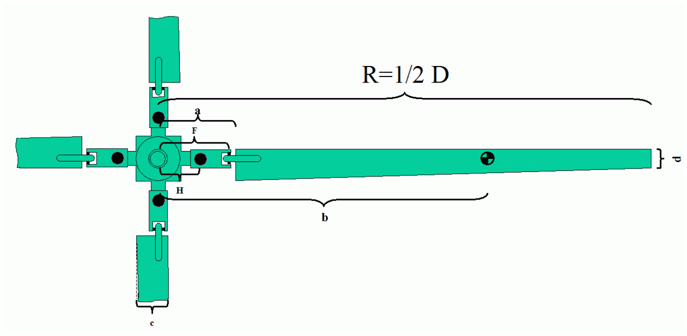

Fig. 10. YASim rotor dimensions. Image courtesy of FlightGear, published under the creative commons license.

IX. Appendix A - YASim FDM

This appendix provides further detail on the YASim flight dynamics model of the Durham autogyro UAV. It outlines the main elements included in the YASim XML document and lists the parameters which can be specified for each one.

A. Rotor

See figure 10 for specific dimensions.

- Position of centre
- Normal vector to rotor (ie. direction of lift)
- Diameter [D]
- Number of blades
- Weight per blade
- Relative centre of blade [b/R]
- Blade chord [c]
- Blade angle of incidence twist
- Blade chord taper
- Relative length where blade incidence is measured
- Relative length where blade starts [a/R]
- Initial rotor position.
- Direction of rotation (clockwise or counter-clockwise)
- Maximum and minimum collective angles
- Maximum and minimum cyclic incidence angles
- Angle of incidence where aerofoil produces no lift
- Stall angle of incidence
- Stall lift factor
- Stall drag factor
- Aerofoil lift coefficient
- Aerofoil drag coefficient
- Maximum and minimum flap angles
- Flap angle at zero rotation
- Relative length of flap hinge [F/r]
- Flap damping constant (delta)
- Translational lift factor
- Number of elements used for blade calculation
- Balance factor
- Centre of rotor tilt
- Rotor tilt angle limits

B. Propeller

- Centre of mass
- Action point - where the thrust force is applied.
- Mass
- Radius
- Cruise speed
- Cruise power
- Cruise altitude
- Takeoff power
- Takeoff RPM
- Maximum and minimum RPM
- Gear ratio
- Contra-rotating propeller; a flag which was set to 1 to negate any gyroscopic moment

C. Engine

Currently only combustion engines, not electric motors, are supported in YASim, however in future an electric motor could be added. It shouldn't make any difference to the dynamic behaviour as the only relevant factor from the engine is the power produced.

- Power
- RPM (related to propeller RPM by propeller gear ratio)

D. Fuselage

A tubelike structure with even mass and aerodynamic force distribution.

- Position of both ends
- Width
- Taper
- Midpoint (location of the widest part of the fuselage)
- Induced drag multiplier
- Generated drag factors

E. Stabiliser

Both horizontal and vertical stabilisers can be specified, horizontal stabilisers are automatically mirrored in the x axis so two symmetrical stabilisers can be defined by specifying one. These are identical to a wing element in YASim.

- Base position
- Length
- Chord
- Angle of incidence
- Twist angle
- Chord Taper
- Sweep angle
- Dihedral angle
- Induced drag multiplier
- Camber

F. Ballast

A ballast is a point mass used to control the mass distribution of the aircraft. They do not affect the overall weight, which is specified at the top of the file, but merely redistribute it.

- Position
- Mass

X. Appendix B - Arduino Code for Selected Firmware Functions

```
static void calc_altitude_error ()
{
    target_altitude_cm = next_WP.alt;
    altitude_error_cm = target_altitude_cm − current_loc . alt ;
}

static void calc_climb_rate_error ()
{
    // limit climb rate for large altitude errors
    target_climb_rate = constrain ( altitude_error_cm ,
        min_climb, max_climb);
    climb_rate_error = K_climb∗( target_climb_rate −
        current_climb_rate ) ;
}

static void calc_airspeed_errors ()
{
    float aspeed_cm = airspeed . get_airspeed_cm () ;
    target_airspeed_cm = g . airspeed_cruise_cm ;
    airspeed_error_cm = target_airspeed_cm − aspeed_cm;
}

static void calc_nav_pitch ()
{
    // Speed is dictated by pitch and kept constant , so
        airspeed error entered into pitch PID
    nav_pitch_cd =
        −g.pidNavPitchAirspeed.get_pid ( airspeed_error_cm ) ;
    nav_pitch_cd = constrain (nav_pitch_cd ,
        g . pitch_limit_min_cd . get () , g . pitch_limit_max_cd . get () ) ;
}

static void calc_throttle ()
{
    // climb rate controlled by throttle , calculated from climb
        rate error
    g . channel_throttle . servo_out = g . throttle_cruise +
        g . pidTeThrottle . get_pid ( climb_rate_error ) ;
}

// 1 st order noise robust differentiator function ,
    differentiates GPS altitude to get climb rate
static float calc_gps_climb( int32_t new_alt)
{
    int32_t slope = 0.0;
    int i ;
    for (i = 0; i < 6; i = i + 1) {
        altitudes [ i ] = altitudes [ i+1];
    }
    altitudes [6] = new_alt;
    slope = (5∗( altitudes [4]− altitudes [2]) +
        4∗( altitudes [5]− altitudes [1]) +
        ( altitudes [6]− altitudes [0]) ) /3.2;
    return slope ;
}
```

XI. Acknowledgment

The author would like to thank Peter Baxendale for his continued support, Charlie Paterson for the CAD models, and the enthusiastic DIY drones community for their work on the Arduplane project and their invaluable help forums.

References

[1] M. Brooks, *Welcome to the personal drone revolution*. New Scientist, 2012.
[2] U. F. A. Administration", *Rotorcraft Flying Handbook*. Washington, D.C.: US Federal Aviation Administration, 2000.
[3] Mathworks, “Hardware-in-the-loop simulation for embedded systems,” http://www.mathworks.co.uk/, (Accessed January 2013).
[4] D. McCoy, “Precession and gyroplane control,” http://www.asra.org.au/Control.htm, (Accessed January 2013).
[5] B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics*. New York: John Wiley and Sons, 1999.
[6] S. Flying, “How to fly autogyros,” http://www.startflying.com, (Accessed January 2013).
[7] S. W. Sung, *Process identification and PID control*. Hoboken, New Jersey: Wiley, 2009.
[8] A. . Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, New Jersey: Princeton University Press, 2011.
[9] “Ardupilot mega,” http://www.ardupilot.co.uk, (Accessed January 2013).
[10] “Ardupilot mega pid tuning,” https://code.google.com/p/ardupilot-mega/wiki/Tuning, (Accessed January 2013).
[11] “Diy drones,” http://diydrones.com/, (Accessed January 2013).
[12] C. Yu, *Autotuning of PID Controllers*. London: Springer, 1999.
[13] Mathworks, “Aerospace blockset,” www.mathworks.co.uk/help/aeroblks, (Accessed March 2013).
[14] P. Holoborodko, “Smooth noise-robust differentiators,” http://www.holoborodko.com/pavel/numerical-methods/numerical-derivative/smooth-low-noise-differentiators/, (Accessed February 2013).
[15] G. . W. . Andrews, *Global Positioning Systems, Inertial Navigation, and Integration*. Hoboken, New Jersey: Wiley, 2007.
[16] A. Ross, “Yasim design notes,” ftp://ftp.uni-duisburg.de/FlightGear/Docs/YASim-simnotes.pdf, (Accessed March 2013).
[17] Flightgear, “Professional and educational flightgear users,” http://wiki.flightgear.org/Professional_and_educational_FlightGear_users, (Accessed March 2013).