

Department of Electrical
and
Computer Systems Engineering

Technical Report
MECSE-27-2006

An Unmanned Aerial Vehicle Autopilot

G.K. Egan and R.J. Cooper

MONASH
UNIVERSITY

An Unmanned Aerial Vehicle Autopilot

G.K. Egan and R.J. Cooper
 Department of Electrical & Computer Systems Engineering
 Monash University 3800
 Melbourne, Australia

Abstract—this paper outlines the functions of a small unmanned aerial vehicle (SUAV) autopilot, intended for civil applications outside civilian airspace, and some of the design considerations.

Index Terms— autopilot, small unmanned aerial vehicle, SUAV, civilian airspace.

I. INTRODUCTION

FOR SUAVs to have wide use will require safe operation by relatively inexperienced operators. These are now also the attributes of the next generation of recreational almost ready to fly (ARTF) model aircraft. The days of a careful apprenticeship of construction of aircraft under the guidance of a skilled model aircraft constructor followed by a long period of flight training punctuated by expensive crashes are now largely over except for those aspiring to the highest levels of this sport, and even they crash sometimes. ARTF aircraft are widely available, inexpensive, and can be flown with some success within minutes.

It is our expectation that most model aircraft will, within a year or two, come equipped with integrated autopilots, including GPS navigation (return to origin), and spread spectrum communications if only to ensure safe use of recreational aircraft by unskilled pilots - "litigation mitigation" if you will. These features will serve to contain aircraft to designated safe flying spaces and return them to the vicinity of the pilot should they for any reason stray out of radio range.

The autopilot to be described briefly here has anticipated many of the characteristics above, which we expect to become commonplace. The core functions of the autopilot have been flown successfully for several years. The target applications for the autopilot are civilian below or outside normal civilian airspace that is the space accepted for the operation of recreational model aircraft.

I. RESEARCH AIRCRAFT

We operate a number of medium endurance (~2Hours) electrically powered research SUAVs in the 2-5Kg class with payloads to 1.5Kg. These endurances are without recourse to environmental energy sources discussed later in the report. Particular attention has been placed on flight safety including flight termination systems [1] that limit aircraft kinetic energy on landing.

Minimising mass is of vital importance in aircraft design if reasonable performance, including endurance, is

to be achieved. As an example, the mass of one of our SUAVs P16025 (Fig. 1) [2] empty is 2.2Kg with a typical payload of 1Kg. A significant part of the airframe mass is committed to batteries which for most missions is 0.7Kg. The power required to maintain this aircraft with full payload in cruising flight at 54Km/H is approximately 28W.

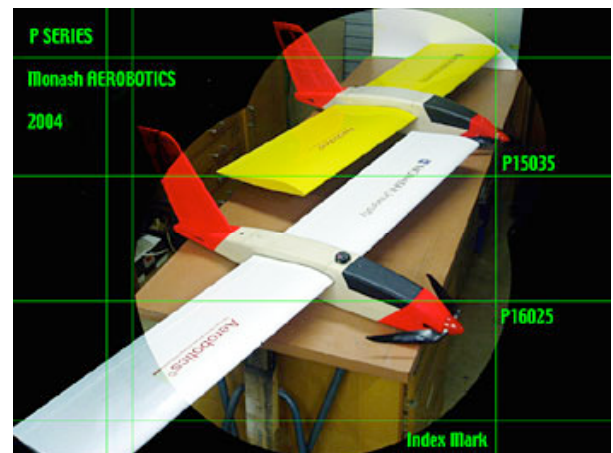


Figure 1: The P Series Aircraft.

TABLE I
 SPECIFICATIONS OF AIRCRAFT P16025

Span	160 cm	Motor	B4021L 5:1 13x11
Chord	25 cm	Duration	60-90 minutes
Length	106 cm	Speed	30-135 KpH
Controls	Elevon	Battery	24x1200mAH LiP
Weight	2.2 to 3.2 Kg	Autopilot	Non-inertial

We have also tested our autopilot on a number of airframes usually associated with the model aircraft fraternity of which we are also members. For many civil applications including those in primary industry and emergency services it is expected that the airframes will often need to be regarded as disposable items given the wear and tear they are likely to sustain. Current ARTF aircraft offer this prospect as they continue to fall in cost and adopt improved aerodynamic sophistication and crash survivability. The Multiplex EasyStar ARTF aircraft [3] capable of carrying our normal video telemetry systems has been flown using the autopilot.

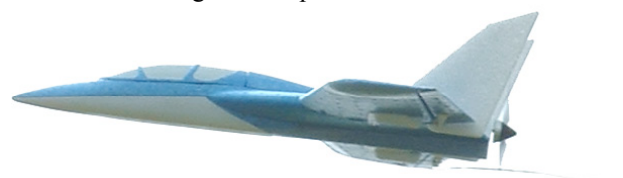


Figure 2: Multiplex MicroJet

The Paparazzi Group at the Ecole Nationale de l'Aviation Civile [4], basing their autopilot developments in part on our own work [5] have successfully flown aircraft as light as the Multiplex MicroJet (Fig. 2) which has a mass of 450gm + avionics.

A reasonable target for the additional avionics, over and above that required for simple radio control (RC), to support full autonomy including GPS antenna and video telemetry would be 50-75gm. In simple terms the mass of the avionics should be of the order required for small a GPS equipped cell-phone without case or batteries.

Model aircraft, and other commodity off the shelf (COTS) equipment are now widely used by the research community although there was some initial reticence and perhaps illogical need by researchers to build everything. Researchers flying model aircraft are often identified as simply having fun – which they generally are!

I. AUTOPILOT GENERAL OPERATION

Our autopilots are designed to support flight by one person in clear conditions outside of civilian airspace. The operator requires only minimal training (less than an hour) to operate an aircraft assuming its airframe configuration is chosen for intrinsically stable flight behavior. By this we mean that the aircraft will restore itself to normal trimmed flight when disturbed given sufficient time and altitude.

Operators may fly the aircraft in computer-assisted mode when the aircraft is within sight. If they lose sight of the aircraft the aircraft will return to the takeoff point and orbit at a safe altitude.

For missions outside visual range the aircraft's course is programmed in a manner no more complicated than programming a GPS for a cross-country trip. Most GPS units have associated trip planning and display tools, for example Garmin's MapSource [6].

The autopilot program is written entirely in C with the most usual aircraft configuration yielding 3200 lines of source. The source library comprises 11000 lines of source.

A. Flight conditions

The missions of interest to us usually involve photoreconnaissance at low altitude and are therefore applicable to Visual Flight Rules (VFR). Under VFR conditions the horizon remains visible or substantially visible at all times.

We fly outside civilian airspace by which we mean airspace that is open to model aircraft as specified by the Australian Civil Aviation Safety Authority (CASA) under CASR 101 [7]. It is worth noting here that the Model Aircraft Association of Australia (MAAA) currently prohibits the use of GPS based navigation systems in model aircraft although it does permit free-flight and flight stabilization in the form of gyros and various forms of wing levelers. With free-flight the aircraft is simply launched and will land where it will after a period of time. In Australia model aircraft airspace is away from built-up areas and airfields and usually below 125M altitude. Our autopilots currently have no

see and avoid capability.

A. Aircraft setup

Aircraft setup assumes that the aircraft has completed its commissioning flights. Other than a full briefing on safety issues, mainly relating to the danger of the propeller, there is no aircraft setup required.

If payloads are changed then the centre of gravity of the aircraft may need to be rechecked. In practice this requires ensuring that the aircraft will balance around clearly marked points on the wing, or in some cases, the fuselage.

A. Operator Control

Currently we use only three channels on a conventional RC transmitter for aircraft control. This will be replaced in due course by a spread-spectrum transceiver most likely with a simple game style hand controller or PDA, the sophistication of current RC transmitters is not required or compatible with two way telemetry.

Our autopilots have three operating modes:

- manual control;
- computer-assisted control;
- autonomous.

We are able to use other RC channels as required for manual payload control in manual and computer-assisted control modes.

1) Manual control

In this mode the RC transmitter's commands are relayed through the autopilot to the control surface servos and the motor control. This mode is normally used only as a failsafe for testing purposes. The autopilot does not limit the aircraft's pitch or roll angle in any way, but does provide mixing of the transmitter's controls to maintain consistency with the other control modes.

1) Computer-assisted Control

In computer-assisted control mode the autopilot will maintain the aircraft at the desired altitude and a bank angle appropriate to the desired turn rate. Advancing the throttle control beyond its mid-point increases the desired altitude. Moving the throttle below mid-point reduces the desired altitude setting. The increase or decrease in desired altitude is relative to the current altitude and is scaled to the aircraft's nominal climb rate.

The aileron control is used to select a desired turn rate. If the control is centred the aircraft will maintain its current heading. The bank angle is limited to an appropriate maximum at all times, typically 30°.

Takeoff can be commanded by simply advancing the throttle. The autopilot then maintains the aircraft heading and a nominal climb rate.

1) Autonomous

The aircraft is entirely under the control of the autopilot. This mode is selected explicitly using the RC transmitter. If the RC signal is lost the aircraft will follow its mission. If no mission is programmed it will return to the takeoff point and orbit at a safe altitude.

I. FLIGHT CONTROL SYSTEM

The flight control system (FCS) is configured from an

aircraft specific configuration file. Some initial parameters are tuned further during flight. An example of a configuration files is given in the appendices. This file may be generated using a simply GUI with associated consistency checks on the desired configuration.

The autopilot reconfigures depending on the sensors available to it. In principle it will still provide some computer-assisted control without any sensors at all, or at least control sufficient for failsafe landings. The minimum practical set of sensors required are those for the control of attitude. Primary configuration of the autopilot occurs when the autopilot program is compiled.

In what follows we will assume the aircraft is being flown in autonomous mode. As will be seen a substantial proportion of the autopilot program is devoted to heuristics or rules conditioning the relatively small amount of program devoted to conventional control. These heuristics are derived in part from considerable experience in flying model aircraft.

A. Airspeed and altitude determination

Conventional pressure sensors are used to determine airspeed and density altitude. GPS data is used to provide a sanity check for altitude values provided sufficient satellites are visible. GPS groundspeed as will be seen later is used in conjunction with airspeed to estimate wind vectors.

A. Attitude determination

In our work we take advantage of the VFR conditions to obtain absolute determination of aircraft attitude using Infrared sensors (IR) rather than the more common inertial reference systems (INR) [8, 9]. This significantly reduces the computational load of the autopilot over that required for INR based control. We are also investigating the use of video cameras to directly determine aircraft attitude [10].

Clouds and rising terrain can cause incorrect pitch and roll values. In almost all cases this is safe as the FCS will roll and/or pitch the aircraft away from the cloud or terrain.

A. Heading determination

For our current implementations we have included a heading gyro to hold heading between GPS updates. As our airframes are, for the most part, intrinsically stable the yaw gyro along with the airspeed and barometric altitude sensors may be used to identify inconsistencies.

In simplified terms if the IR sensors determine the aircraft is level but in fact the aircraft is rolling away from a cloud significant heading changes and yaw rates will be detected. If the airspeed is increasing faster than throttle settings and other factors dictate, then the IR pitch sensor may have failed.

These situations are relatively easy to detect permitting a degree of data fusion within our onboard computational constraints – we do not currently use Kalman Filtering.

The ultimate fall back is to take all control surfaces to trim position, close throttle, and initiate failsafe.

A. Control system tuning

The time taken to tune typical commercial autopilots

for a particular payload and airframe configuration is quoted as requiring several days of trial and error tuning to obtain satisfactory performance. Our own experience with one autopilot [11] confirms this. In practice of course, careful design of airframe and payload location can result in tuning parameters close to acceptable requiring only modest re-tuning with different payloads. Nonetheless the tuning process, be it through flight-testing or by simulation [12], requires considerable knowledge and experience in flying model aircraft.

Some of the less expensive autopilots claim minimal tuning requirements; we have not yet had the opportunity to verify the operation of these when used in conjunction with aircraft in the category of interest to us. In a number of cases these autopilots are a simple combination of aircraft-levellers with rudder/aileron control controlled by the cross track error of an associated GPS unit. As such heading is controlled by yawing the aircraft rather than through a properly coordinated turn. We favour adoption of well-understood, often empirical, design of the airframes coupled with automatic in-flight tuning of the FCS.

While sophisticated model identification systems can be used, to date we have found the older well understood but not necessarily optimum techniques, in our case Ziegler-Nichols (ZN) to be adequate. This has been used in the support of our camera based attitude control research [10][13]. As our aircraft are predominantly electrically powered, difficulties related to changing mass and associated inertial response do not need to be considered. We have found varying control gains based on airspeed, as a replacement for explicit gain scheduling, to provide acceptable performance. We are however testing in-flight model identification and tuning schemes [14].

The tuning procedure for new aircraft involves manually controlling the aircraft in sharp pitch and bank excursions. The aircraft samples the input control demand (approximating to a step function) and determines the PID constants using the well-understood formulae due to ZN. In practice we usually identify for only PI control.

A. Flight modes

Unlike more conventional autopilots we do not have distinct autopilot modes differentiating climb, hold, descent etc. We use instead a set of heuristics overlaying more traditional control strategies.

1) Airspeed

The aircraft's default airspeed is usually the best ratio of lift to drag (L/D) to maximize time aloft for given energy. This is qualified, however, by the desired time of arrival at the next waypoint in which case the desired airspeed may be increased.

The default control for airspeed is through airspeed-from-pitch. The alternative of controlling airspeed-from-throttle with altitude-from-pitch is not appropriate for aircraft which glide for a significant part of their mission. The effect of using airspeed-from-throttle and increasing pitch in an attempt to increase altitude when battery power is exhausted is rather obvious.

If no airspeed sensor is available, or it becomes

unavailable, the FCS maintains best L/D pitch for the aircraft using the attitude sensors and adopts an altitude-from-throttle strategy. Loss of airspeed and altitude sensor information can occur when the pitot or static tubes become blocked by for example water droplets or ice.

1) Altitude Hold

As may be seen from the following AltitudeHold function there is a significant amount of program logic conditioning the control loops including the use of propeller induced drag to aid descent should the aircraft stray above the current altitude hold band usually due to environmental lift.

Note also that if altitude data is not available due to say sensor failure then the throttle is closed.

```
void AltitudeHold()
{ // Throttle controls Altitude
  int16 Temp;

  if (Lev.F.UseAltitude)
  {
    Temp=DoPID(AltitudeV)+CruiseThrottle;
    Lev.F.PropellorDragOn=((Lev.V[AltitudeV].Error
    <(-AltitudeHoldBand*2))&&PropellorDragEnabled);
    if (Lev.F.PropellorDragOn)
      Temp=Limit(Temp, DescentThrottle, CurrMaxThrottle);
    Lev.Controls[ThrottleC]=ConstrainThrottle(Temp);
  }
  else
    Lev.Controls[ThrottleC]=0;

  Lev.F.Gliding=((Lev.V[AltitudeV].Error<=0)
  &&(Lev.Controls[ThrottleC]<ThrottleCutout));
}
```

If the aircraft is a powered glider then we have found it to be more efficient for the aircraft to climb at the most efficient power setting to the upper limit of the altitude hold band then to power down completely until the aircraft falls through the lower limit.

1) Heading

Heading is in general controlled by banking the aircraft within its maximum bank window as set by the aircraft configuration file.

The GPS used provides heading updates at one-second intervals. Some GPS units provide higher update rates but often these are provided by an internal extrapolation calculation. In our case a yaw gyro is used to extrapolate heading estimates from the current and previous yaw rate samples every 22.5mS.

Most of our aircraft have elevon (combined elevator and aileron) control only and so fine-tuning of heading from rudder is not available. Ailerons/elevons are controlled independently allowing control of adverse yaw. The yaw gyro may be used to condition this in more advanced control schemes [14] but for most aircraft this is not necessary. Where a rudder is available it is usually coupled simply to the aileron/elevon control. Feed forward control is used to apply up elevator offset in turns.

1) Takeoff

The aircraft is deemed to have been launched if autonomous mode is engaged, the desired altitude exceeds the origin altitude and the aircraft has not reached flying speed for the first time and the altitude has not

increased to some threshold above launch.

The desired altitude is set by the navigation system and will be that of the first waypoint or the failsafe return to origin setting if no mission plan is loaded.

The autopilot maintains a climb pitch, airspeed and the heading at launch until the launch altitude threshold is reached. Once the altitude threshold is reached the navigation system is enabled and other controls including spoilers, flaps, parachute etc are armed. If the navigation system has no mission loaded the aircraft will continue to climb to safe altitude and orbit the launch point.

1) Landing

If the current altitude is below the Approach Altitude and the desired altitude is below the current altitude or there is a Ground Proximity Alarm (as set by an ultrasonic or infrared detector) the aircraft is deemed to be landing. If the Proximity Alarm is not yet set flaps are deployed. When the Proximity Alarm is set the spoilers are also deployed, the flaps raised and the aircraft flares by maintaining distance from the ground using the proximity detector.

The program fragment that controls the landing sequence and also limits maximum altitude (excluding the thermal escape functions) is shown below:

```
void CheckAltitudeBoundaries()
{ // call to this function must immediately precede call to PitchHold
  int16 AboveAltLimit, Temp, RelAltitude;

  RelAltitude=Lev.V[AltitudeV].Current-OriginAltitude;
  Lev.F.Flying=Lev.F.Flying||(RelAltitude>AllArmingAltitude);
  Lev.F.Landing=Lev.F.Flying
  &&(RelAltitude<ApproachAltitude)

  &&((Lev.V[AltitudeV].Error<0)||Lev.F.GroundProximityAlarm);

  AboveAltLimit=RelAltitude-CurrMaxAltitude;
  Lev.F.AltitudeAlarm=(AboveAltLimit>0);

  if (Lev.F.AltitudeAlarm)
  {
    Temp=(Limit(AboveAltLimit, 0, AltitudeHoldBand)*
    MaxSpoilerServoThrow)/AltitudeHoldBand;
    SlewControls(SpoilersC, Temp, 1);
  }
  else
    if (Lev.F.Landing)
    {
      if (Lev.F.GroundProximityAlarm)
      {
        Lev.F.ThrottleArmed=false;
        SlewControls(SpoilersC, MaxSpoilerServoThrow, 1);
        SlewControls(FlapsC, MinFlapServoThrow, -1);
        Temp=BestPitch+(MaxCLPitch-
        BestPitch)*(LandingFlareAltitude-
        Lev.GroundProximity)/LandingFlareAltitude;
        Temp=Limit(Temp, BestPitch, MaxCLPitch);
        UpdateVarDesired(PitchV, Temp);
      }
      else
      {
        SlewControls(FlapsC, MaxFlapServoThrow, 1);
        SlewControls(SpoilersC, MaxSpoilerServoThrow, 1);
      }
    }
    else
      ResetFlapsAndSpoilers();
}
```

A. Use of integers

Numerical computations within the autopilot are integer based. Integer or fixed-point computations are used commonly for embedded applications because of

their energy efficiency and lower computation time compared with floating-point computations even when using dedicated floating-point hardware.

Two's complement integer arithmetic has the unfortunate characteristic of large positive numbers overflowing to large negative numbers if their range is exceeded. In an aircraft this can lead to full down-elevator being commanded when the aircraft is in fact asking for more than full up-elevator when recovering from a dive!

So called saturation arithmetic, which has been used for some time in graphics pipelines of mainstream processors (Pentium MMX and PowerPC AltiVec) is now appearing in processors intended for embedded applications (Arm, Microchip DSP). With saturation arithmetic the largest maximum (or minimum) number is generated depending on the sign of the operands¹. Some synthesis approaches suitable for FPGA implementations soften the saturation as the result approaches a maximum magnitude [15].

This still leaves the problem of scaling the variables in the calculation such that sufficient precision is maintained over several operations in the calculation of control surface settings. A naïve approach is to track the maximum and minimum possible values of the variables in the computation and scale accordingly. This usually leads to overly conservative scaling and insufficient precision. Another technique is to use floating-point numbers initially and run extensive simulations to determine the range required for intermediate values. Without hardware supported saturation arithmetic or C saturation arithmetic class libraries in the final implementation this can lead to disasters with the one set of values not covered in the simulation resulting in 'full down-elevator'! It may be observed that many of the techniques for dealing with these problems were developed for analog computers and have been partially lost over time or are at least outside Google's view!

For our autopilot we chose to scale variables to permit sufficient precision for fine control while explicitly clipping the larger excursions using a Limit Function seen frequently in the program fragments here. This was done such that maximum control surface actuation remained available.

I. NAVIGATION

The navigation system is 4D providing the ability to arrive at waypoints at a designated time (if feasible). It uses a simple waypoint list, with optional cycles, to define the mission. Each waypoint carries the desired coordinates, altitude, target arrival time and payload (camera) commands.

Routes are generated using MapSource and further annotated with payload control information and waypoint target arrival times.

The navigation system also tracks some environmental factors likely to affect missions. When the aircraft is gliding the GPS groundspeed and heading information is used to estimate windspeed and direction. Rate of Climb

is also monitored and used to map regions of rising or falling air (lift and sink) due to thermals and wind driven air masses moving upwards over rising ground. Ageing is used to decay values to a small nominal sink value of $-1MS^{-1}$ over time.

A. Waypoint approach strategies

There are a number of approach strategies for acquiring waypoints that include those where the aircraft must pass:

- through the waypoint within a defined ellipsoid (aircraft will persistently turn and climb/descend until it is within the arrival ellipsoid);
- through waypoint ground coordinates at any altitude above that specified (aircraft is maximizing lift use and will circle the waypoint until it is within the arrival circle);
- as close as possible with no go-around used mainly for landings and takeoffs (aircraft selects next waypoint when crossing line perpendicular to the desired track passing through the current waypoint);

Waypoint approaches usually attempt tangential arrival on the side away from the next waypoint permitting smoother turn transitions. Turns are always in the direction of minimum heading error. Of course if the waypoints are in a straight line as in landing the aircraft will fly directly through the waypoint coordinates (if possible).

Arrival ellipsoids and circles are dimensioned so as to be achievable with the aircraft's turn rate.

A. Containment

It is necessary to contain the aircraft within some operating area when in autonomous flight and often when in computer-assisted flight. For example an aircraft when accidentally, or deliberately flown outside a safe area should override manual or autonomous control and stay within the safe area. In most cases the safe volume may be inferred by the mission route but not always as the mission if not checked properly may stray into prohibited airspace.

The containment volumes may of course become very complex for some missions traps that an aircraft cannot escape if flown into. Obviously we must take into account the maneuvering capability of the particular aircraft and adjust the containment volumes to ensure the actual prohibited areas are not encroached. Unlike terrestrial robots that often may turn conveniently on their vertical axis, aircraft, other than helicopters cannot. Containment space feasibility is the subject of future work under mission planning.

The simplest (default) volume for our autopilot is a cylinder of some radius from the launch point with an absolute altitude limit. Under default conditions the aircraft will return to the takeoff point if it crosses the containment radius. Breaches of the altitude boundary engage the altitude limiting strategies previously outlined.

The now widespread use of GPS systems for vehicle navigation is leading to increasing sophistication in the

¹ We exclude here a discussion of modulus arithmetic applicable to two's complement representations.

information, including real-time data (traffic flows). GIS databases are being used to augment 3D views (building heights, vegetation etc) for 3D displays. We expect airspace classifications are already included in these databases making containment planning much simpler. For recreational flyers park boundaries including internal use classifications allow, or will allow, containment planning to be automatic when the aircraft is switched on.

A. Failsafes

There are two failsafe regimes. One is externally triggered either deliberately or by loss of radio communications. The circuits supporting this are independent of the Autopilot.

The second failsafe regime is initiated by the Autopilot specifically the core FCS.

1) Externally triggered

The prospect of runaway aircraft is quite real and as a consequence we have placed considerable emphasis on implementing and testing our failsafe strategies.

The pilot at all times has over-riding direct RC of our aircraft. If valid RC signals are lost for a period (2.5S) a braking parachute is released to contain the kinetic energy of the aircraft by limiting its airspeed to just above stall. Release of the parachute physically cuts power to the propulsion system.

The intention is not to lower the aircraft intact to the ground but to prevent potential runaway as some of our aircraft can comfortably exceed $50MS^{-1}$ and possibly double this in a full power dive before probable aircraft breakup.

For longer-range missions, which are outside normal RC range, the failsafe is triggered by loss of a low power VHF beacon signal. If no beacon is present then loss of RC signal triggers failsafe otherwise the Autopilot assumes autonomous control.

The monitoring of RC and beacon signal integrity is entirely independent of the FCS although the FCS can also trigger, but not override, failsafe.

1) FCS triggered

For the FCS itself there is a hierarchy of failsafe decisions leading ultimately to flight termination. This is described by changes to the FCS state. If a particular failsafe state persists for a timeout-period then the state machine moves to a new state triggering some action. 'Navigating' is the default state. If GPS signals are lost then the state is changed to 'Continuing' where the aircraft continues on the current heading (as determined by zero yaw) and altitude. 'Continuing' is designed to bridge over a few GPS signal losses. If GPS is not recovered then the state becomes 'HoldCourse' where the aircraft climbs (or descends) to the failsafe altitude followed by 'Orbiting' where the aircraft adopts a maximum turn rate at constant altitude followed by 'Terminating' with throttle off, full spoilers/flaps.

If the altitude continues to increase, usually because of thermal or ridge lift, the aircraft continues on the current heading for a period and then resumes the descent orbit. If the altitude still does not decrease the process is repeated this time in a different heading.

The parachute is deployed once the aircraft descends to a low altitude to minimize wind drift. Deployment

altitude is such that an aircraft that is traveling at high velocity will be slowed to acceptable velocities prior to impact. The 'Terminating' state cannot be exited by recovery of GPS signals.

A fragment of the FCS failsafe program function is shown below:

```
void UpdateLevMode()
{
  switch (Lev.Mode) {
  case Navigating:
    {
      if (ClockSeconds>=AbortTimeOut)
        {
          AbortTimeOut=ClockSeconds+AbortContinueTime;
          Lev.Mode=Continue;
        }
      break;
    }
  case Continue:
    {
      // stop any turn
      UpdateVarDesired(HeadingV, Lev.V[HeadingV].Current);
      // hold altitude
      UpdateVarDesired(AltitudeV, Lev.V[AltitudeV].Current);

      if (ClockSeconds>=AbortTimeOut)
        {
          AbortTimeOut=ClockSeconds+AbortHoldTime;
          Lev.Mode=HoldCourse;
        }
      break;
    }
  case HoldCourse:
    {
      if (ClockSeconds>=AbortTimeOut)
        {
          UpdateVarDesired(AltitudeV, DefaultAltitude+OriginAltitude);
          AbortTimeOut=ClockSeconds+AbortOrbitTime;
          Lev.Mode=Orbiting;
        }
      break;
    }
  case Orbiting:
    {
      // do orbit
      UpdateVarDesired(HeadingV, Lev.V[HeadingV].Current+
        TurnFlare/2);
      ContinueAbort();
      if (ClockSeconds>=AbortTimeOut)
        {
          Lev.F.FullSpoilers=true;
          Lev.F.FullFlaps=true;
          UpdateVarDesired(AltitudeV, -MaxAllowedAltitude);
          AbortEntryAltitude=Lev.V[AltitudeV].Current;
          Lev.Mode=Terminating;
        }
      break;
    }
  case Terminating:
  default:
    {
      ContinueAbort();
      if (Lev.V[AltitudeV].Current>
        (AbortEntryAltitude+AltitudeHoldBand))
        {
          // fly straight to attempt thermal exit
          UpdateVarDesired(HeadingV, Lev.V[HeadingV].Current);
          AbortTimeOut=ClockSeconds+AbortThermalTime;
          Lev.Mode=EscapeThermal;
        }
      break;
    }
  }
}
```

A. Flight extension and mission planning

This is an entire report in its own right. In short we have a number of opportunities to augment or extend

flight duration. To do this mission plans need to be sufficiently flexible and framed so that the autopilot can take advantage of naturally occurring energy sources autonomously.

For UAVs there is considerable opportunity to exploit naturally occurring lift and hopefully to avoid sink although as we have seen it may be necessary to escape lift, particularly thermals, in some circumstances.

In some cases the desired arrival time at the next waypoint may be relaxed such that the aircraft can take advantage of thermal and ridge lift to save power. For littoral missions we only have to observe birds ferry gliding along coastal dunes or cliffs to see the opportunities for extended patrol missions. The use of environmental lift for flight extension, including our own preliminary work [16] is seeing greater interest.

We are currently commissioning an in-flight solar recharging system for the flight batteries of our aircraft. Our initial computations indicate that the flight times of our main research aircraft, even with their relatively small wing areas, may be extended by a few hours.

I. AVIONICS POWER CONSUMPTION

A target mass of 50-75gm for avionics places constraints on the architecture and physical manifestation of the autopilot and associated sensors and control actuators.

The power budget for computation to navigate and control flight should ideally be less than 5% of the aircraft's cruise power consumption that in the case of the P16025 would be 1.2W; this is difficult to achieve.

A. Control surface actuation

The total power consumption for the two elevon servos on the P16025 averages approximately 0.6W with a peak of 1.6W in turbulent air with a normal update interval of 22.5mS. This already exceeds our budget.

To reduce power consumption in relatively still air we can lengthen the update interval or simply turn the servo power off, relying only on its mechanical holding torque.

A. GPS and other sensors

The Trimble SQ GPS receiver power consumption is 100mW. The balance of the sensors use an additional 30mW.

A. Computation

For larger aircraft the weight of wiring can quickly exceed the weight of the electronics being connected. For UAVs the situation is not as clear given their smaller dimensions.

Distributed architectures, with processors using CAN Bus as the interconnecting network, have now become well established in the automotive industry. They are also being used in larger UAVs such as the Mark 4 Aerosonde developed in conjunction with our university. The principal drivers are wiring weight and modular development.

For smaller UAVs such as those based on the ARTF MicroJet [3] it seems clear that the avionics will be integrated into a single package including sensors, GPS and data transceiver.

We ultimately chose to use a multiprocessor architecture based on two programmable interface controllers or PICs (Microchip 18F8722). The latest version of the autopilot has the PICs, altitude and airspeed sensors, GPS and yaw gyro tightly integrated. The choice of a PIC implementation was driven as much by curiosity as to their capability and the fact that they were in use in our teaching programs. The code/data space for the navigation and telemetry PIC is 12K/1K bytes and for the FCS PIC 13K/800 bytes.

Future versions of the autopilot will use a high-performance processor with adequate memory for navigation, telemetry and extended mission planning capability. Developers of processors for embedded control and telemetry applications have been aggressive in implementing power conservation schemes. By using short bursts of computation as required and then sleeping it is likely that the overall power consumption will be less than that for the PIC implementation.

A dedicated processor, or possibly a field programmable gate array (FPGA), will continue to be used for the core functions of the FCS although the principal strengths of current FPGAs lie more in high performance image processing pipelines [10].

The autopilot is highly modular, written in C, and already runs on a variety of platforms including workstations and so we foresee little if any difficulty in porting to a new processor architecture.

I. TELEMETRY

The autopilot transmits all of its internal state variables and flags to the ground once per second. Each major data structure in the program is sent as a separate tagged packet. The information transmitted is too large to document here and can be best seen in the data structure definitions in the appendices.

A simple pocket sized groundstation, which displays a small subset of the telemetry data, has proved invaluable and extremely practical for field trials. Telemetry data for most trials is recorded and may be monitored in real-time or replayed using a visualization tool [17].

I. CLOSING REMARKS

There are many elements of our autopilot that have not been detailed here. We hope however that some flavour of what we have developed emerges. We have found our autopilots to be practical in use after several years of flying and we believe they have a number of characteristics, particularly those relating to ease of use and safety, that will become commonplace.

There is a rapid increase in civilian applications of UAVs and the somewhat more practical and affordable UAVs. We expect the use of UAVs will expand quickly when it can be shown they can be flown safely outside of normal civilian airspace.

ACKNOWLEDGMENT

We wish to thank the members of the Aerobotics Research Group at Monash University [2], and in particular Ian Reynolds and Paul Jenkins, for their assistance with fabrication and in testing the various

revisions of the autopilot, and Professor John Bird who prompted the use of IR based flight control in 2001.

REFERENCES

- [1] Egan, G.K., Cooper, R.J., and Taylor, B., Unmanned Aerial Vehicle Research at Monash University, AIAC-11 Eleventh Australian International Aerospace Congress, February, 2005 also as MECSE-15-2006, Department of Electrical & Computer Systems Engineering, Monash University, 2006.
- [2] Monash University Aerobotics Group, <http://www.ctie.monash.edu.au/hargrave/aerobotics.html>
- [3] Multiplex Modellsport gmbH, <http://www.multiplex-rc.de/>
- [4] The Paparazzi Project at ENAC, <http://www.recherche.enac.fr/paparazzi/>
- [5] Taylor, B., Bil, C., Watkins, S., and Egan, G.K., Horizon Sensing Attitude Stabilisation: A VMC Autopilot, 18th International UAV Systems Conference, Bristol, England, Mar 2003.
- [6] Garmin Mapsource, <http://www.garmin.com/cartography/>
- [7] Civil Aviation Safety Regulations 1998 (CASR) Part 101, Office of the Legal Counsel, Civil Aviation Authority, 1st Edition, January 2003.
- [8] Egan, G.K., The Use of Infrared Sensors for Absolute Attitude Determination of Unmanned Aerial Vehicles, MECSE-22-2006, Department of Electrical & Computer Systems Engineering, Monash University, 2006.
- [9] Herrmann, P., Bil, C., Watkins, S., and Taylor, Simulation and Flight Test of a Temperature Sensing Stabilisation System', 19th International UAV Systems Conference, Bristol, England, Mar 2004
- [10] Cornall, T.D., Egan, G.K. and Price, A., Aircraft attitude estimation from horizon video, IEE, Electronics Letters, Volume 42, Issue 13, p. 744-745, 22 June 2006.
- [11] Micropilot MP2028 Installation and Operation, MicroPilot, Stony Mountain, Manitoba, Canada, October 2003. <http://www.micropilot.com/>
- [12] Hardware in the loop simulator for Piccolo avionics, Cloud Cap Technology, Hood River, Oregon, USA, Sept 2003., <http://www.cloudcaptech.com/>
- [13] Cornall, T., 'Using average sky and ground coordinates to determine aircraft roll angle and horizon position', PhD Thesis, Department of Electrical & Computer Systems Engineering, Monash University, *in preparation*.
- [14] Liu, M., Egan, G.K., and Yunjian Ge., Identification of Attitude Flight Dynamics for An Unconventional UAV, IROS06, Beijing, 2006.
- [15] Constantinides, G.A., Cheung, P.Y., and Luk, W., Synthesis of Saturation Arithmetic Architectures, ACM Transactions on Design Automation of Electronic Systems (TODAES) Volume 8, Issue 3, pp 334-354, 2003.
- [16] Goodwin, A., Egan, G.K., and Crusca, F., 'UAV Ridge Soaring in an Unknown Environment', MECSE-7-2006, Department of Electrical & Computer Systems Engineering, Monash University, 2006.
- [17] Price, E., and Egan, G.K., 'Real-time UAV Visualisation using a Flight Simulator', MECSE-9-2006, Department of Electrical & Computer Systems Engineering, Monash University, 2006.

Appendices

```

//*****
// UAV Autopilot - P16025 Hacker B40/21L
//      5:1 Maxon 16.5x13 Aeronaut
// Copyright (C) G.K. Egan 2001-2006
//*****

const uint8 AircraftID[8]="P16025 ";
// Configuration default - Ailerons,
// Rudder and Elevator
#define ELEVONS
//#define RUDDERELEVATOR

//#define HASFLAPERONS
//#define HASSPOILERONS

#define UseAltimeter true
#define UseAirspeedIndicator false
#define UsePropellorDrag false
#define UseGliderStrategies false
#define UseZAxis false

```

```

#define LiP 0
#define NiMH 1
#define NiCad 2

#define BatteryType LiP
#define BatteryCells 9

#define AVIONICSUSESMAINBATTERY

// Throttle and Airspeed
#define MaxThrottle 80 // 13.5x11
#define CruiseThrottle 60 // 16.5x13
#define BestClimbThrottle MaxThrottle
#define ThrottleCutout 5
#define RestartThrottle 7
#define DescentThrottle 10

#define StallAirspeed 82
#define MaxAirspeed 270
#define BestAirspeed 114
#define BestClimbAirspeed (BestAirspeed+10)
#define VNEAirspeed (MaxAirspeed*2)

// Pitch
#define PitchSensorSense (1)

#define MinDragPitchDeg (-2)
#define BestLDPitchDeg 5
#define LaunchPitchDeg 7
#define MaxCLPitchDeg 11
#define MaxFlarePitchDeg MaxCLPitchDeg
#define StallPitchDeg 13

#define MinPitchDeg MinDragPitchDeg
#define MaxManualPitchDeg 45
#define MinManualPitchDeg (-20)
#define StallDescentPitchDeg StallPitchDeg

#define BestClimbPitchDeg 16

// Roll
#define RollSensorSense (1)

#define MaxRollDeg 40
#define MaxManualRollDeg 45

#define AileronRudderCoupling 0
#define AileronDifferential 80
#define AileronElevatorCoupling 20

// heading rate from roll angle
// (%) extrapolation to compensate between
// GPS updates
#define RollToHeading 50

// Servo sense and mechanical throw limits
#define ThrottleServoSense 1

#define MaxAileronServoThrow 40
#define MinAileronServoThrow (-40)
#define LeftAileronServoSense (-1)
#define RightAileronServoSense (-1)

#define ElevatorServoThrow 40
#define LeftElevatorServoSense (-1)
#define RightElevatorServoSense (-1)

#define RudderServoThrow 0
#define RudderServoSense (1)

// Flap and Spoiler settings below are for
// where there are dedicated control surfaces
// for these functions
#define LeftFlapServoSense (1)
#define RightFlapServoSense (-1)
#define FlapServoSense 1 // for Flap mixer
#define MaxFlapServoThrow (100)
#define MinFlapServoThrow (-100)

#define SpoilerServoSense (1) // always Y
lead
#define MaxSpoilerServoThrow (0)
#define MinSpoilerServoThrow (-0)

#define Aux1ServoSense 1
#define MinAux1ServoThrow 0
#define MaxAux1ServoThrow 0

// Control gains
#define DefPitchGain 35
#define DefRollGain 45
// Abort strategy

```

```

#define DefHardAbort false
#define HasParachute true

//*****
// UAV Autopilot - Data Structures
// Copyright (C) G.K. Egan 2001-2006
//*****
-----

typedef struct {
    uint8 BadAlmanac:1;
    uint8 NoRTC:1;
    uint8 NoAntenna:1;
    uint8 NoBattery:1;

    uint8 No3DFix:1;
    uint8 Mode2D:1;
    uint8 Mode3D:1;
    uint8 Fault:1;
} TrimbleFlags;

typedef struct {
    uint8 DGPSMode;
    uint8 Status;
    TrimbleFlags F;
} TrimbleState;

typedef struct {int16 Current, Desired, Error,
IntError;} VarRec ;

typedef struct {
    uint8 UseAirspeed:1;
    uint8 UseAltitude:1;
    uint8 ThrottleArmed:1;
    uint8 ElevatorArmed:1;

    uint8 Flying:1;
    uint8 StallAlarm:1;
    uint8 Gliding:1;
    uint8 Landing:1;

    uint8 AltitudeAlarm:1;
    uint8 FullFlaps:1;
    uint8 FullSpoilers:1;
    uint8 GroundProximityAlarm:1;

    uint8 PropellorDragOn:1;
    uint8 f8:1;
    uint8 f9:1;
    uint8 f10:1;

    uint8 RCLinkUp:1;
    uint8 Glitch:1;
    uint8 NavValid:1;
    uint8 MustLandNow:1;

} LevFlags;

typedef struct {
    uint8 NavArrivalAlarm:1;
    uint8 ReturningToOrigin:1;
    uint8 GPSValid:1;
    uint8 NavValid:1;

    uint8 No3DFix:1;
    uint8 WayPointValid:1;
    uint8 Flying:1;
    uint8 Landing:1;

    uint8 CameraOn:1;
    uint8 CameraShutter:1;
    uint8 OutOfBounds:1;
    uint8 f11:1;

    uint8 f12:1;
    uint8 f13:1;
    uint8 f14:1;
    uint8 f15:1;
} NavFlags;

typedef struct {
    uint8 AvionicsBatteryVoltsAlarm:1;
    uint8 ServoBatteryVoltsAlarm:1;
    uint8 MotorBatteryVoltsAlarm:1;
    uint8 BatteryTemperatureAlarm:1;

    uint8 MotorTemperatureAlarm:1;
    uint8 MotorTemperatureNotAvailable:1;
    uint8 BatteryTemperatureNotAvailable:1;
    uint8 MotorCurrentNotAvailable:1;

} AirframeFlags;

typedef struct {
    uint8 NavValid:1;
    uint8 No3DFix:1;
    uint8 Abort:1;
    uint8 ReturningToOrigin:1;
    // etc.
} UpdateFlags;

typedef struct {
    real32 Latitude, Longitude;
    int16 Altitude;
    int16 GroundSpeed;
    int16 Heading;
    int16 RateOfClimb;
    int16 EstHorizontalError;
    int16 NoOfSats;
} GPSState;

typedef struct {
    int32 Time;
    real32 Latitude, Longitude;
    char PointID[8];
    int16 Altitude;
    uint8 Strategy; // + desired AS
    uint8 padding; // force 16 bit align
} WayState;

typedef struct {
    uint8 ID[8];
    VarRec V[StateVars];
    int16 Controls[MaxControls];
    int16 RCCommands[NoOfCommandInps];
    int16 AttitudeSensorSwing;
    int8 GroundProximity; // 120dM max
    uint8 CommandState;
    uint8 Mode;
    LevFlags F;
} LevState;

// inter processor FCS update
typedef struct {
    int16 GPSAltitude, GPSHeading,
    DesiredHeading, DesiredAltitude,
    AltitudeLimit;
    int8 Strategy;
    int8 AirspeedIncrease;
    UpdateFlags F;
} Update;

typedef struct {
    int32 MissionTime;
    GPSState GPS;
    WayState Way;
    int32 ClosingRange;
    int16 WayHeading;
    char TurnAdvice, ClimbAdvice;
    NavFlags F;
    uint8 AirspeedIncrease;
} NavState;

typedef struct {
    int16 MotorBatteryVolts, MotorCurrent,
    AvionicsBatteryVolts, ServoBatteryVolts;
    real32 BatteryDischarge;
    int8 BatteryTemperature, MotorTemperature;
    AirframeFlags F;
} AirframeState;

typedef struct {
    int16 Altitude, Airspeed, GlitchCount,
    RateOfClimb;
} BasicState;

typedef struct {
    uint8 S[MessageLength];
    int16 Value;
    boolean RaiseAlarm, Sent;
} MessageState;

typedef struct {
    int16 WindSpeed;
    uint8 NorthLift, EastLift;
    int8 MaxLift;
    uint8 WindSector;
    uint8 LiftAdvice;
} EnvironmentState;

```