Indian Journal of Science and Technology Vol.2 No 4 (Mar. 2009) ISSN: 0974- 6846

The design of an unmanned aerial vehicle based on the ArduPilot

He Bin and Amahah Justice
Beijing University of Chemical Technology, Beijing, China-100029.
justice.blackreverend@gmail.com

Abstract: This paper focuses on the design, modeling, implementation and testing of an autonomous unmanned aerial vehicle. The controller is based on an Ardupilot board which is a custom PCB with an embedded processor (ATMega168) combined with circuitry to switch control between the RC control and the autopilot control. It controls navigation (following GPS waypoints) and altitude by controlling the rudder and throttle. It uses flight stabilization system (co-pilot), a sensor pack, Global Positioning System (GPS) and an RF transceiver to monitor and report crucial parameters such as altitude, speed, pitch, roll, and position. An embedded software algorithm has been developed to enable the aerial vehicle accomplish the required autonomy and maintain satisfactory flight operation. The autopilot features an advanced, highly autonomous flight control system with an auto-launch and auto landing algorithms.

Keywords: Autopilot, GPS, MUX, UAV, FMA Copilot

Introduction

A growing area in aerospace engineering is the use and development of unmanned aerial vehicles (UAV) for military and civilian applications .There are difficulties in

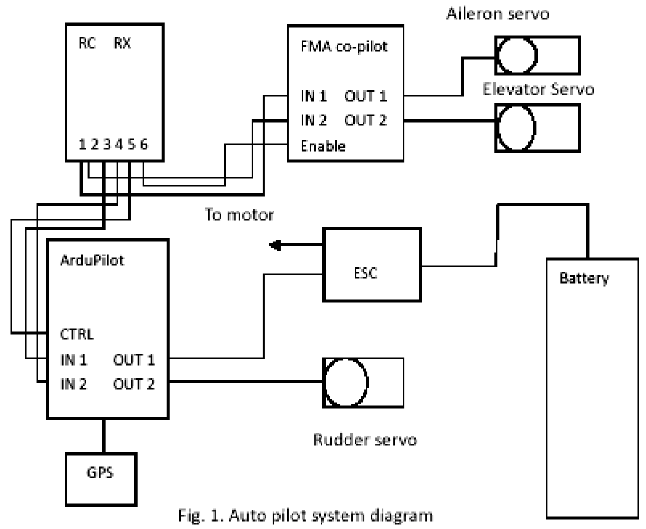


Fig. 1. Auto pilot system diagram

the design of these vehicles due to the varied and non-intuitive nature of the configurations and missions that can be performed (Krus & Andersson, 2003). Currently there has been a huge demand for UAVs and services for real time and remote sensing. Unmanned aerial vehicles can be deployed to solve a number of civilian tasks. It can be used as an effective means of search, detection and identifying of objects or subjects of interest as well as their precise coordinates (Paul G. Fahlstrom & Thomas J. Gleason: Introduction to UAV Systems). UAVs are also very useful in disaster management. In the occurrence of a forest fire, for instance, it is very difficult to have a precise data on the development of the situation. But with the deployment of a UAV which is capable of flying at low altitudes and able to navigate with GPS waypoints and machine vision, the situation can be controlled very efficiently.

UAV Autopilot Controller (Ardupilot)

ArduPilot is a full-featured autopilot based on the open-source Arduino platform (DIY Drones: http://www.diydrones.com). The Ardupilot is a custom PCB with an embedded processor (ATMega 168) combined with circuitry to switch between RC control and the autopilot control (i.e., the multiplexer/failsafe; otherwise known as a MUX). This controls navigation (following GPS waypoints) and altitude by controlling the rudder and throttle. These components are all open source. This autopilot is fully programmable and can have any number of GPS waypoints (including altitude) and trigger camera or other sensors.

Features of the ArduPilot

It can be used for an autonomous aircraft. The built-in hardware failsafe uses a separate circuit to transfer control from the RC system to the autopilot and back again. It includes the ability to reboot the main processor in mid-flight. It makes provision for the use of multiple waypoints. It also provides a 6-pin GPS connector for the 1hz EM406 GPS module.

Overview of the Autopilot system

The Autopilot system comprises of the Arduino compatible Ardupilot board, 1Hz EM406 GPS module and an Infrared Flight Stabilization System (FMA co-pilot). The EM-406A GPS module from Global Sat is based on the spectacular SiRF. This complete module is built upon the same technology as ET-301, but includes on-board RAM, and a built-in patch antenna. The Infrared Flight Stabilization System is used to sense the difference in infrared temperature between the Earth and the sky. The sky is always at a relatively lower infrared temperature, while the infrared signature of the earth is always relatively warmer. Co-pilot uses two pairs of infrared sensors: one pair points fore and aft and the other points left and right. When one pair of sensors sees a change in the aircraft's orientation relative to the earth's infrared horizon, co-pilot issues signals to the control system to bring the aircraft back into level flight.

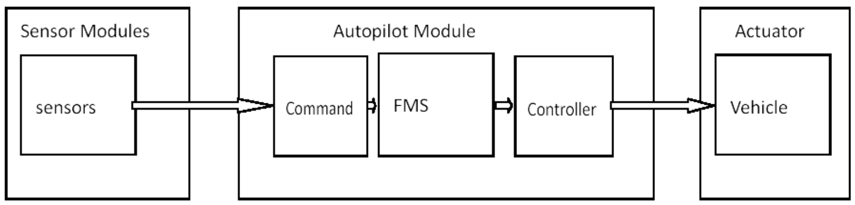


Fig. 2. Conceptual Data Flow and Components

Hardware (Airframe/Platform)

Plane

Deciding on the platform to use is very paramount in the building of a UAV. We used the Hobbico Superstar EP as my platform. The specification of the platform chosen is provide in Table 1.

Table 1. the specifications of the Hobbico Superstar

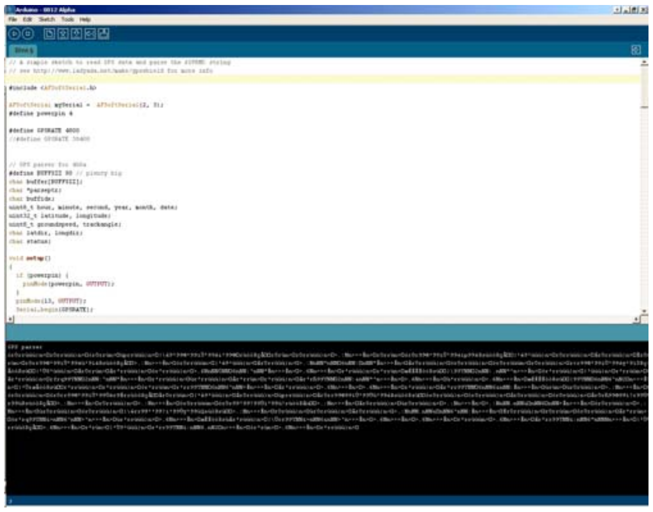
Wingspan	*48.5 in*
Wing area	*400 sq in*
Wing loading	*16.1 oz/sq ft*
Length	*36.1 in*
Weight	*3.11bs*
Radio	*4-channel with three servos*
Motor Battery	*8.4volt 1700mAh-3000mAh*

Center of gravity: 2.5” (63.5mm) -2-7/8” (73mm) back from the leading edge of the wing at the fuselage side

Control Throws :

	Low Rate	High Rate
Elevator, Up and Down:	6.35mm	11mm
Rudder, Right and Left:	9.5mm	15.9mm
Ailerons, Up and Down:	6.35mm	6.35mm

Fig. 3. Screen shot of the Arduino Environment



```
// A simple sketch to read GPS data and parse the $GPRMC string
// see http://www.ladyada.net/make/gpsshield for more info

#include <AFSoftSerial.h>

AFSoftSerial mySerial = AFSoftSerial(2, 3);
#define powerpin 4

#define GPSRATE 4800
//#define GPSRATE 38400


// GPS parser for 406a
#define BUFFSIZ 90 // plenty big
char buffer[BUFFSIZ];
char *parseptr;
char buffidx;
uint8_t hour, minute, second, year, month, date;
uint32_t latitude, longitude;
uint8_t groundspeed, trackangle;
char latdir, longdir;
char status;

void setup()
{
  if (powerpin) {
    pinMode(powerpin, OUTPUT);
  }
  pinMode(13, OUTPUT);
  Serial.begin(GPSRATE);
```

```
GPS parser
[illegible]
```

Software

Arduino Software: The open-source Arduino environment makes it easy to write code and upload it to the I/O board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java that serves as a code editor and compiler, and based on Processing and other open source software. The Arduino environment has the following functions: Verify/Compile, Check for errors in the code and uploads the already compiled code to the board.

Advantages of the Arduino software

1) Cross-Platform-The Arduino software runs on Windows, Macintosh OSX and Linux operating systems, 2) Simple, clear programming environment, 3) Open source and extensible software, 4) The language can be expanded through C++ libraries ,and AVR-C code can be added directly into Arduino programs.

Ground station

The ground station/monitor consists simply of a laptop computer that is used to display the video stream sent by the UAV. The Aerial video system (AVS) used is AVS-KX171 Camera. The AVS is a complete wireless video kit with 900MHz frequency, 500mW RF output power, 12V camera.

Aerial video system block diagram

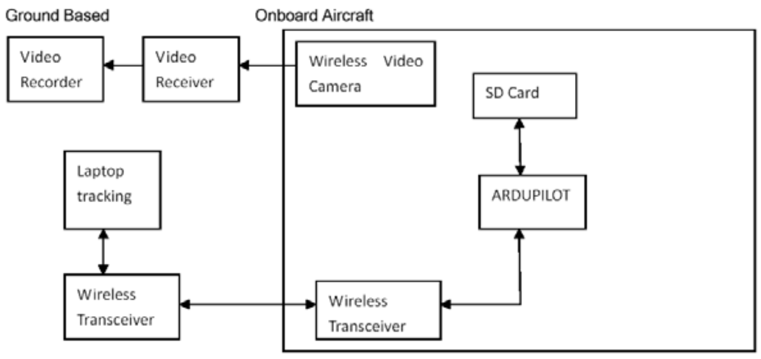
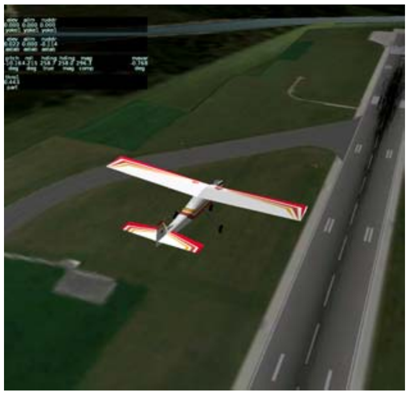


Fig. 4. Aerial Video System Block Diagram

Simulation / ground control.

Fig. 5. Screen Shot of the Simulation in XPlane



Screen shot of simulation in Google Earth is provided in Fig. 6. This method of simulation was developed by Jordi (DIYDRONES).

Required hardware: ArduPilot, FTDI cable and computer.

Required software: Modified ArduPilot code, X-Plane simulator, Google Earth and Ardu Simulator.

The simulation runs in X-plane simulator, while the ArduPilot which is connected via the FTDI cable to the pc receives simulated GPS data over serial and it returns back proposed servo positions back over serial as part of telemetry information. Variables that are recorded include longitude, latitude, altitude, waypoints and distance.

Functions of the Ardu simulator

1) Connects to ArduPilot over serial for sending and receiving data, 2) Connects to X-plane on localhost, 3) Reads data from X-plane (lat/lon/alt/ course), sending these to ArduPilot as GPS sentences, 4) Simulating FMA copilot stabilization on ailerons/elevator, 5) Reads and Displays telemetry and servo positions from ArduPIlot, 6) Sends servo positions to X-plane to control throttle and

rudder and 7) Records fly path and sends it to Google Earth to display.

Modified Ardu Pilot Code (samples)

```
//Defining ranges of the servos (and ESC), must be +-90 degrees.
#define max16_throttle 2100 //ESC max position, given in useconds, in my ungly servos 2100 is fine, you can try 2000..
#define min16_throttle 1000 //ESC position
#define max16_yaw 2100 //Servo max position
#define min16_yaw 1000 //Servo min position
#define reverse_yaw 1 // normal = 0 and reverse = 1
//PID max and mins
#define   heading_max 15
#define   heading_min -15
#define   altitude_max 40
#define   altitude_min -45
//Number of waypoints defined
#define   waypoints 6
#define distance_limit 4000 //The max distance allowed to travel from home.
/*****************************************************************
 * RTL, if it set as true by the user, the autopilot will always
 *****************************************************************/
#define RTL 1 //0 = waypoint mode, 1 = Return home mode
//PID gains
//At the beginning try to use only proportional.
//The original configuration works fine in my simulator.
#define Kp_heading 10
#define Ki_heading .01
#define Kd_heading 0.001
#define Kp_altitude 4
#define Ki_altitude 0.001
#define Kd_altitude 2
/******************************/
//Defining waypoints variables
float wp_lat[waypoints+1];
float wp_lon[waypoints+1];
int wp_alt[waypoints+1];
byte current_wp=1; //This variables stores the actual waypoint we are trying to reach..
byte jumplock_wp=0; //When switching waypoints this lock will allow only one transition..
byte wp_home_lock=0; //
int wp_bearing=0; //Stores the bearing from the current waypoint
unsigned int wp_distance=0; //Stores the distances from the current waypoint
/******************************/
//PID loop variables
unsigned int heading_PID_timer; //Timer to calculate the dt of the PID
int heading_previous_error;
float heading_I; //Stores the result of the integrator
float heading_D; //Stores the result of the derivator
int heading_output; //Stores the result of the PID loop
unsigned int altitude_PID_timer;//Timer to calculate the dt of the PID
int altitude_previous_error;
float altitude_I; //Stores the result of the integrator
float altitude_D; //Stores the result of the derivator
int altitude_output; //Stores the result of the PID loop
//PID K constants, defined at the begining of the code
float  kp[]={Kp_heading,Kp_altitude};
float  ki[]={Ki_heading,Ki_altitude};
float  kd[]={Kd_heading,Kd_altitude};
/******************************/
char buffer[90]; //Serial buffer to catch GPS data
char head_rmc[]="GPRMC"; //GPS NMEA header to look for
char head_gga[]="GPGGA"; //GPS NMEA header to look for
byte unlock=1; //some kind of event flag
byte checksum=0; //the checksum generated
byte checksum_received=0; //Checksum received
byte counter=0; //general counter
/*GPS Pointers*/
char *token;
char *search = ",";
char *brkb, *pEnd;
//Temporary variables for some tasks, specially used in the GPS parsing part (Look at the NMEA_Parser tab)
unsigned long temp=0;
unsigned long temp2=0;
unsigned long temp3=0;
//GPS obtained information
byte fix_position=0;//Valid gps position
float lat=0; //Current Latitude
float lon=0; //Current Longitude
byte ground_speed=0; //Ground speed? yes Ground Speed.
int  course=0; // Course over ground...
int alt=0; //Altitude,
//ACME variables
byte gps_new_data_flag=0; // A simple flag to know when we've got new gps data.
unsigned int launch_altitude =0; //launch altitude, altitude in waypoints is relative to starting altitude.
int middle_thr=90; //The central position
int middle_yaw=90; //The cnetral position of yaw
byte middle_measurement_lock=0; //Another lock to void resetting the middle measurement..
/******************************/
int test=0;
```

Simulation Results

The results of the simulation weren't exactly ideal because of certain problems such as the cumbersome nature of specifying individual waypoints manually. Another problem that arose was the altitude control by throttle with the use of the FMA copilot stabilization system which didn't work properly. Apart from this the simulation was successful. The plane flew smoothly and according to predetermined waypoints.

Navigation

The Ardupilot operates in two modes, first of all it can be programmed with desired waypoints (any number of waypoints), and also can be programmed to RTL (return-to-launch).

Methods

Programming the Ardupilot board for desired waypoints involving first obtaining the GPS Longitude and Latitude coordinates of the waypoints and then entering the GPS Longitude and Latitude coordinates into the code in the Arduino environment and uploading it to the board. The coordinates can be obtained from Google Maps, under satellite view. The waypoints ("wp-lat" and "wp-lon") declaration obtained would be pasted on the Mission Setup tab of the Arduino.

The altitude is relative to the initial launch position and not above sea level. For an example if the airfield is 500 meters above sea level, then "500" would be entered as the waypoint altitude. Hence the plane will fly at 1,000 meters above sea level but it would be just 500 meters above .

Code Sample:

```
void setup_waypoints(void)
{
 /*Declaring waypoints*/
 wp_lat[1]=  34.982613;
 wp_lon[1]= -118.443357;
 wp_alt[1]=50; //meters
 wp_lat[2]= 34.025136;
 wp_lon[2]=-118.445254;
 wp_alt[2]=100; //meters
 wp_lat[3]=34.018287;
 wp_lon[3]=-118.456048;
 wp_alt[3]=100; //meters
 wp_lat[4]= 34.009332;
 wp_lon[4]=-118.467672;
 wp_alt[4]=50; //meters
  wp_lat[5]=  34.006476;
 wp_lon[5]=-118.465413;
 wp_alt[5]=50; //meters
 wp_lat[6]=  34.009927;
 wp_lon[6]= -118.458320;
 wp_alt[6]= 20; //meters
```

RTL (Return-To-Launch)

For the UAV to return-to-launch, then the code still has to be modified. The code "#define RTL 0",this is the RTL flag and it should be set to 1 or just set to zero if the UAV is to follow the waypoints in the Mission Setup tab.

Autonomous Flight

For the first test flight of the UAV, we setup the system as described above i.e., connecting all the components to the board, modifying the code and uploading it to the board. Components include the EM-406A GPS, FMA Copilot Stabilization unit, Servos, Tower hobbies 72 MHz 8 channel RX and Tower Hobbies 6 XM TX and the AVS.

The UAV was launched manually and when it was about 150 feet it was switched into autopilot by flipping the control switch which corresponds to our channel 5. The UAV successfully started navigating to the predetermined first waypoint that we uploaded into the board. We then uploaded the RTL code (return -to-launch), the throttle behaved as expected and maintained steady altitude. It returned to launch point.

Results and conclusion

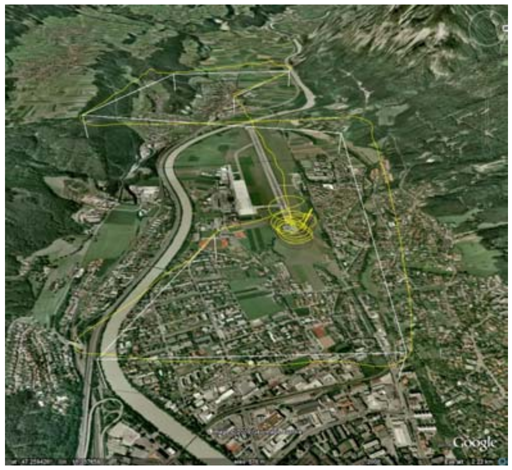
In this paper, we have discussed the design and implementation and simulation of an Ardupilot based UAV autopilot system.

The simple UAV has the ability to implement autonomous flight (automatic take and landing). The developed UAV has been tested successfully in both manual and automatic flight operations. Useful data has also been obtained by the use of computer vision, i.e. the AVS and valuable data has also been obtained by the use of telemetry.

Future work

Presently waypoints can only be entered pre-flight and manually, this limit the autonomous properties of the UAV .It would be very useful to develop a method to reprogram the waypoints on the controller board mid-flight. Power consumption is another issue worthy of considering, presently the power rating of the UAV will last only for several minutes without recharging, it would be great if the power rating of the UAV could be reasonably increased. Two methods worthy of consideration is 1) the use of solar panels for recharging mid-flight, 2) the use of automatic recharging, i.e. having the UAV taxiing into predetermined destination and recharging on its own then taking off. This would require the addition of extra electronic components which could drastically affect the weight of the aircraft and could also give rise to magnetic interference.

Fig. 6. Screen shot of simulation in Google Earth



Reference

Krus P and Andersson J (2003) Optimizing optimization for design optimization. In: *Proceedings of ASME Design Automation Conference*, Chicago, USA, September 2-6.