Title:
An Approach to Enhance an Economical COTS FCS for UAV

Authors:
Shing Chi Chan
Maria Luviano
Adrienne Lam
Dr. Helen Boussalis
Dr. Chivey Wu
Dr. Khosrow Rad

Affiliations:
SPACE NASA-URC (University Research Center)
California State University Los Angeles
College of Engineering Computer Science and Technology
Department of Electrical Engineering

Address:
5151 State University Drive, Los Angeles, CA 90032

Contact:
Shing Chi Chan – intel815e@gmail.com
Maria Luviano – coneylu2004@yahoo.com
Adrienne Lam – ALam4@calstatela.edu
Dr. Helen Boussalis – HBoussa@calstatela.edu
Dr. Chivey Wu – CWu@calstatela.edu
Dr. Khosrow Rad – KRad@calstatela.edu

Introduction:
Most Flight Control Systems nowadays are available in the form of embedded system platforms due to its complexity yet robust performance. In our UAV Flight Control System (FCS) development, we employed a Commercial-Off-The-Shelf (COTS) system developed in the Arduino programming environment. This paper details our methods and approach to developing a fully autonomous flight control system for our aircraft, with a fundamental and inexpensive system as a starting point. Within the given capability of the open source codes and limitations of an embedded system, our team managed to integrate and amplify specific features involving autonomous take-off, waypoint flight and autonomous landing. Although these features were introduced by the open source codes, they provide general support for small scale aircraft and are not optimized for our specific application. To achieve full autonomous flight for our aircraft requires us to address all requirements of proper takeoff, flight and landing.

Open source software:
As we develop these required features, we must acknowledge the limitations of the embedded system. The presence of landing gear, for instance, was not part of the given feature of the open source codes. Due to that fact, proper landing procedures are necessary for our aircraft to perform a safe autonomous landing. The challenge of implementing this particular feature was that a significant portion of aerodynamics computations was required to be integrated with the existing FCS functionality. Furthermore, it is not possible to implement such a design by "adding" a separate module in the algorithm; it needs to be embedded in the existing algorithm. The requirements of a proper landing are described in more detail below.

System Identification:
In any FCS development of control algorithm, system identification is often times a vital piece of required task. Performing system identification enables developers to model the system in simulation environment; fine-tuning to such system can then also be made possible and becomes rather simple with modern tools like Matlab & Simulink. The drawback of system identification is time; it is a rather time consuming task, because a deep understanding of the system is required before a representation of such can be carried out. Without the luxury of time, system identification is simply not an option. In our FCS development, we employed to use trial and error to fine tune the existing PID controller. This debug method of control algorithm tuning is not our preferred solution, yet it shortens the time required for minor fix. The FCS hardware performance was then observed through Software-In-the-Loop (SIL) by using the X-Plane Simulator. Upon many simulation iterations, we obtained the result with our desired FCS performance. For major changes made to the FCS, however, system identification must be performed.

Software and Hardware Architecture:
To develop a controller that meets the requirements for proper autonomous takeoff and landing, we require access to variables and functions within the existing code that contain valuable navigation/flight data or perform such calculations. The information is necessary to determine the current state of the UAV and is utilized as controller inputs. The controller outputs are connected to variables in code wherein adjustments can be made to

the UAV's control surfaces and throttle, providing corrections to speed, altitude, pitch, yaw, roll, etc.

To trace these variables and functions, the existing code libraries were transferred to a Linux system for convenience. This enabled us to parse through the entire library for a particular variable or function name to determine where it was defined, how it was used by other functions and if it had any dependents. When traced down far enough, the functions that perform calculations based on raw sensor inputs or GPS data can be found. Each function in the code performs specific calculations and each output is intended for a specific use. As to not interfere with the existing flight control system, we use these output variables and take them into separate functions we create. These functions then yield the exact information needed as controller inputs.

A critical part of our system is the interface between hardware and software systems. In order to operate an uninhabited vehicle, hardware considerations have to be taken into account along with the structure of the software. A real-time structure must be used in order to satisfy these requirements. An embedded system is ideal for our purposes since it interacts at a low level with physical hardware. This cuts down on software overhead. We are using the ATMega328 8-bit RISC microprocessor with a 6-channel 10-bit ADC at 20MHz along with a GPS module and transmitting/receiving modules to communicate between the vehicle and ground station. The software is structured as a real-time system where certain tasks must be performed within a time period. This enables the software to schedule critical tasks according to the priorities assigned to them.

The takeoff and landing stages of autonomous flight require more accuracy since the slightest error can be fatal. Due to this reason it is important to know how often calculations are being performed and how often variables are being updated. Increasing this frequency is preferred as would increase the accuracy. However, this can cause a problem similar to the interference issue mentioned previously.

Takeoff:

The takeoff procedure begins once the aircraft at position zero on the runway; the brakes are released and the throttle is pushed to maximum takeoff power. There are two stages in the takeoff process that sum up to the total take off distance, the ground roll and the obstacle clearance distance also known as airborne distance. During ground roll the angle of attack is held constant at a set value. Once the airplane reaches a takeoff speed, 1.2 ratio of the stall speed the elevator is deflected to rotate the plane nose up in order to increase the angle of attack and consequently increase its lift. Precaution must be taken during this phase. The angle of attack should not exceed the stalling angle of attack and must be within the limits set by the ground-tail clearance. However, the angle of attack must be enough to produce a lift larger than the negative forces acting on the plane such as weight and drag. The airplane should continue to accelerate until reaches takeoff speed. At this point the aircraft begins its second and last takeoff stage. Once airborne it will continue to climb until it clears a specified obstacle height usually set at 50ft above the ground. Once this point is reached it is considered a successful takeoff. If the takeoff

distance is limited by the runway clearance, the takeoff distance can be controlled with an increased elevator deflection and the velocity.

Landing

There are three stages during landing that the plane has to go through to successfully land in a defined distance on the runway. The first phase of the landing is the approach. During this stage the plane has to reach a defined altitude of 50 ft minimum. At this instance the plane should begin a steady glide approach with a constant glide angle to help reduce the energy that will be dissipated during ground impact. The second stage of landing is the flare stage. Here the nose of the plane must be pitched up to achieve level flight just before touching the ground. The rear landing gear must touchdown first. The ground clearance is an important factor in at this point. The pitch of the plane should always be kept within the limits of the ground-tail clearance to prevent the tail from coming into contact with ground. The nose should only fall when the elevator is unable to hold the nose up. The perfect landing would be having the wheels touch the ground just at stall speed. The third and last stage of the landing sequence begins right at touchdown (sequence known as ground roll).The plane should maintain a certain positive pitch angle after touchdown using its aerodynamic drag, the brakes and throttle to slowdown and have better control of the landing distance.

PID for Rudder Control:

A PID controller is added to the existing code in order to maintain rudder control during auto-landing. The PID controller would be inactive during flight and would be activated only during the auto-landing sequence. The last two waypoints, n and n-1 are used to calculate the reference heading. The current heading which is also the feedback is calculated by the pitch, roll, yaw and GPS signals which are extracted from the existing code. The rudder also sends a feedback signal which is converted to a current heading signal through aerodynamic equations. The bearing error is calculated by subtracting the current heading from the reference heading. Tuning the PID is a matter of trial and error until optimal control is achieved. The output of the PID is amplified and then sent to control the rudder.