

The Low Cost UAV Project

Brent Bryan Anthony Rowe

May 12, 2004

1 Introduction

Our project's goal is to design a low cost yet capable unmanned air vehicle which is controlled entirely by onboard processing. The project is largely targeted towards surveillance or search and rescue situations where there is a need for quickly deployable remote telepresence. Our goal of creating a simplistic yet functional design is an ideal research and teaching platform for aerial robotics. The UAV should be capable of autonomous stabilization, navigation, basic image processing and local wireless communication.

This is not by any means the first attempt to create such an autonomous aircraft. Groups at Brigham Young University¹ and Georgia Tech University² as well as hobbyists³ have designed and built prototype UAVs. While some of the projects used platforms similar to the one we chose, most projects turn commercial once they start being successful⁴, which means that few of the tricks are general knowledge and the resulting aircraft are far from "low cost"⁵

Thus, our goal is not only to create a working low-cost UAV, but to provide the comprehensive solution that really is "low cost" to the consumer, not just the manufacturer. Before, we discuss the current status of our project, let us first discuss the design of our system.

2 Hardware

2.1 Plane

Once set on building an autonomous aircraft, our first decision was what plane we should use. A previous test using a Wingo Porter⁶, which exploded on a mild turn in mid-flight due to torque on fuselage caused by the extra weight of the sensors in the nose of the plane, suggested that we needed an aircraft in which we could place most of our sensors in the wing and near the center of gravity. In addition, we also wanted an aircraft that

¹See: http://www.ee.byu.edu/grad1/users/beard/www_docs/papers/preprints/KingstonEtAl03

²See: <http://controls.ae.gatech.edu/uavrf/>

³See: <http://autopilot.sourceforge.net/>

⁴For instance: <http://www.magicctech.com/> or <http://www.rotomotion.com/>

⁵e.g. <http://www.magicctech.com/prices.html>

⁶<http://www.hobby-lobby.com/porter.htm>

was built out of EPP foam and thus more resilient to an occasional hard landing (read: numerous crashes during testing).

As such, we chose to use a Zagi ⁷, a common hobby model shaped like a flying wing or a B2 bomber. The plane can be bought as kit from many dealers, and requires some assembly. In the radio-controlled flying community, these aircraft are routinely used in combat situations (get together where pilots try to get their planes to hit other planes and cause them to crash), and thus were known to withstand multiple crashes without serious damage. Zagis have a wing span of about 4 feet, and have only two control surfaces: two ailerons that run the full length of each wing. Other benefits of using the Zagi include the fact that it has been designed to include additional payload beyond the necessary radio and servo gear, and, since it is a combat plane, the included motor has about twice the horsepower needed to get the plane aloft; this extra power proved to be very handy once we weighted the plane down with all of our electronic gear. One final reason for choosing the Zagi is that because it is a flying wing, it is nowhere close to being passively stable. Thus, it is very evident if the autopilot is working or not.

2.2 Electronics

The control board shown in Figure 1 is responsible for all of the real-time onboard aircraft control, sensor integration and mechanical actuation. Our control board design includes 3 MEMS rate gyros, 2 dual axis MEMS accelerometers, a temperature sensor, a high precision ADC and an onboard ARM processor. The main processor is a low power 80Mhz ARM7TDMI with 128Kb of “zero wait” flash memory and 64 Kb of RAM. The processor integrates signals from an 8 channel 14bit ADC that is responsible for collecting the output of the MEMS rate gyros, the temperature sensor and the dual axis accelerometers. The temperature sensor is included as the MEMS devices are known to drift proportionally to their temperature. The rate gyros monitor three different axes of rotation, while the accelerometers measure the linear acceleration of the plane in the “forward”, “left” and “down” (orthogonal) axes of the aircraft. The three gyros are read at 40Hz, while the accelerometers are read at 50Hz. Note that these update rates were calculated to be within the bandwidth of each of the sensors. To allow ample computational time to run a Kalman Filter, etc., aircraft control loop runs at 20Hz.

In addition to accelerometers and gyros, the main control board communicates with a GPS engine and a Bluetooth module using its two RS232 ports. The GPS engine provides position, elevation, bearing and velocity information that is updated once a second. The main control board then uses its inertial sensors to more accurately track the aircraft’s position between GPS updates. The Bluetooth module allows for bidirectional air to air or air to ground communication. Our current module has a range of approximately 150 meters, which is primarily used for local telemetry, or inter-aircraft communication. Each Bluetooth module can connect to up to 8 other nearby devices. This wireless channel of communication can be used to send digital commands such as new waypoints to the aircraft, as well as test the aircraft while in flight.

The primary interface between the main control board and the aircraft body is through eight bi-directional servo ports. The standard remote control receiver in the aircraft communicates to the main control board through 4 servo inputs. Two inputs for elevator

⁷<http://www.zagi.com/>

Block Diagram

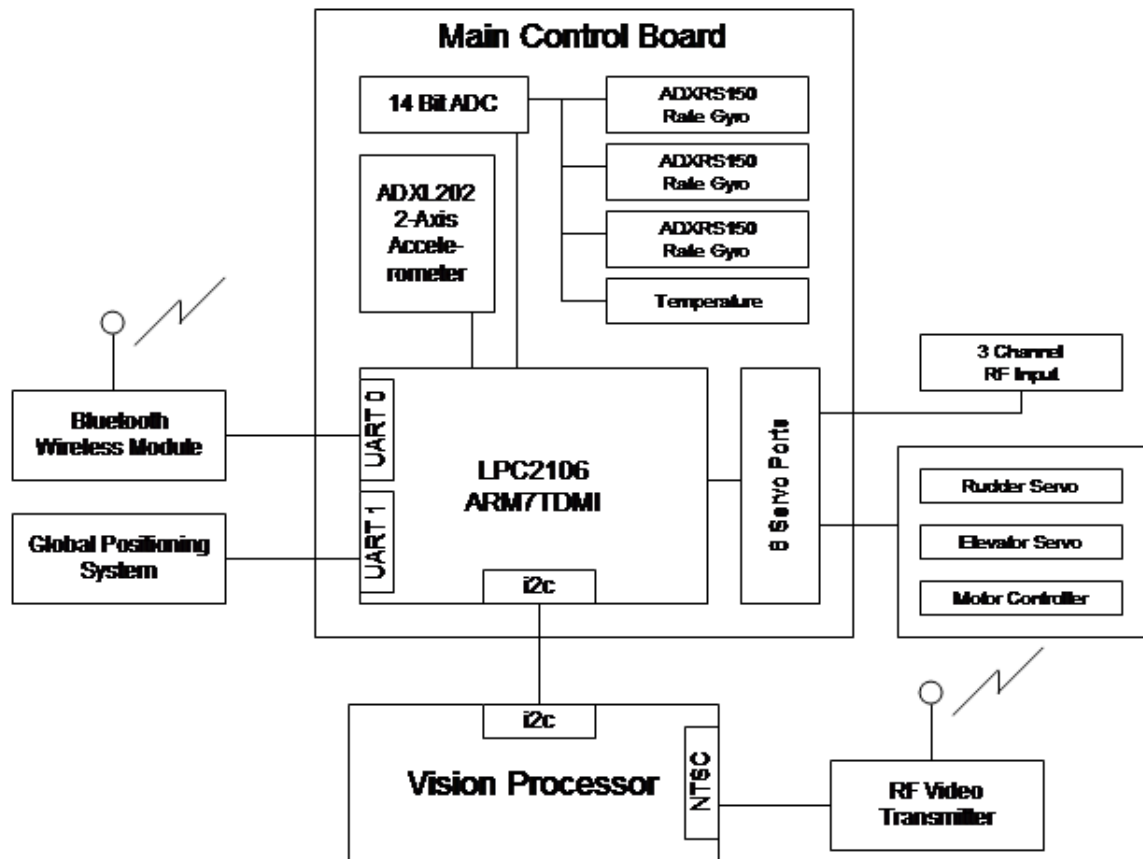


Figure 1: Block Diagram of Control Board

and rudder control, one channel for motor control and a remote override channel. The remote override channel allows the pilot to enable or disable the onboard automatic pilot.

We plan to add a custom embedded vision processor to do horizon tracking and possibly obstacle detection. This, vision processing board consisting of another identical ARM processor that is directly linked to a CMOS image sensor. The vision processor is capable of loading a 176 by 288 color image into memory at 16.9 frames per second. The vision processor board can also transmit an analog black and white NTSC image back to a ground based receiver. Communication between the video processing board and the main control board is achieved using a high speed i2c data bus.

3 Software

Software on the aircraft is currently divided into two broad categories: 1) low-level sensor reading and 2) localization and control code.

3.1 Sensor Reading

As mentioned in §2.2, the servos and accelerometers are polled at a rate of 145Khz, and the acceleration values are updated asynchronously whenever the pole determines it has reached the fall of the square-waveform. The 3 gyros are polled at 40Hz, and then every other sample is averaged with the one before it to compute the angular velocity at a rate of 20Hz, the same rate as the control loop.

3.2 Localization and Control Code

To determine the current state of the aircraft, we are using a standard technique of implementing a Kalman Filter⁸. Before we go into detail on the Kalman Filter implementation, let us define our coordinate systems and terminology.

First, we define world coordinates in terms of the ground, centered location where the plane was launched. We define the X_w axis as pointing North, the Y_w axis as pointing East, and the Z_w as pointing down, so as to follow the right-hand rule. We define ψ as the angle about Z_w , θ as the angle about Y_w and ϕ as the angle about X_w . We start the angle ψ , θ , and ϕ at zero when the plane is thrown, and assume that the plane is thrown level.

Next, we define body coordinates in terms of the aircraft. We do this facilitate easy translation of the values the sensors read (which are in body coordinates) into world coordinates (which we care about to do control). Thus, we define X to be out the nose of the aircraft, Y to be out the right wing of the aircraft and Z to point directly out of the bottom of the aircraft, as shown in Figure 2. We define P , Q and R as the angular velocities about X , Y and Z , respectively. Note that the gyros are a direct measure of P , Q and R (modulo the noise), while the accelerometers measure the acceleration in X , Y and Z .

⁸<http://www.cs.unc.edu/welch/kalman/>

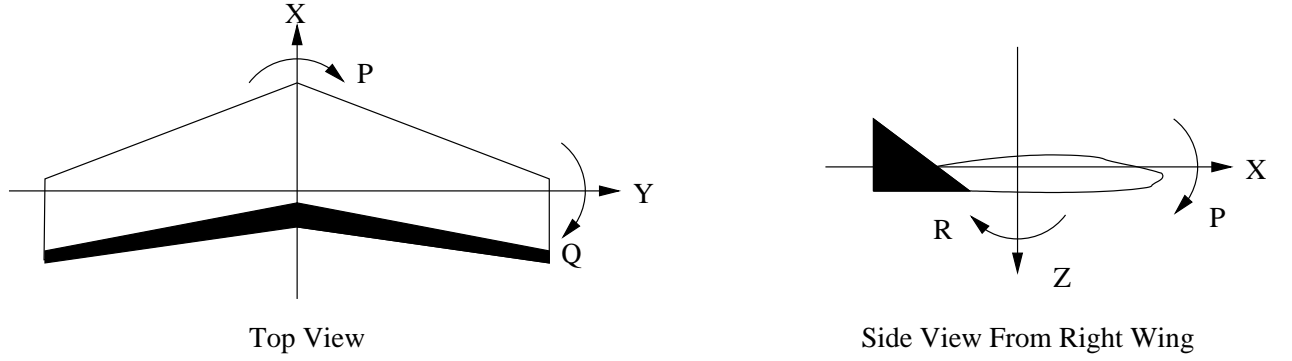


Figure 2: Aircraft Coordinate System

Now to translate the observations into world coordinates, it can be shown that⁹

$$\begin{aligned}
 \ddot{X} &= \dot{Y}R - \dot{Z}Q - g \sin(\theta) \\
 \ddot{Y} &= \dot{Z}P - \dot{X}R + g \sin(\phi) \cos(\theta) \\
 \ddot{Z} &= \dot{X}Q - \dot{Y}P + g \cos(\phi) \cos(\theta) \\
 \dot{\phi} &= P + Q \sin(\phi) \tan(\theta) + R \cos(\phi) \tan(\theta) \\
 \dot{\theta} &= Q \cos(\phi) - R \sin(\phi) \\
 \dot{\psi} &= Q \sin(\phi) \sec(\theta) + R \cos(\phi) \sec(\theta)
 \end{aligned}$$

These equations can be used to constrain $\psi, \theta, \phi, \dot{\psi}, \dot{\theta}$ and $\dot{\phi}$. Moreover, we can directly get $X_w, Y_w, Z_w, \dot{X}_w, \dot{Y}_w$ and \dot{Z}_w from the GPS.

Thus, our state variables for the Kalman Filter are $X_w, Y_w, Z_w, \dot{X}_w, \dot{Y}_w, \dot{Z}_w, \psi, \theta, \phi, \dot{\psi}, \dot{\theta}, \dot{\phi}$ where we note that all of these variables are respect to the ground. Update equations are given by

$$\begin{aligned}
 X_w^{(t+1)} &= X_w^{(t)} + \dot{X}_w^{(t)} dt \\
 Y_w^{(t+1)} &= Y_w^{(t)} + \dot{Y}_w^{(t)} dt \\
 Z_w^{(t+1)} &= Z_w^{(t)} + \dot{Z}_w^{(t)} dt \\
 \dot{X}_w^{(t+1)} &= \dot{X}_w^{(t)} \\
 \dot{Y}_w^{(t+1)} &= \dot{Y}_w^{(t)} \\
 \dot{Z}_w^{(t+1)} &= \dot{Z}_w^{(t)} \\
 \psi^{(t+1)} &= \psi^{(t)} + \dot{\psi}^{(t)} dt \\
 \theta^{(t+1)} &= \theta^{(t)} + \dot{\theta}^{(t)} dt \\
 \phi^{(t+1)} &= \phi^{(t)} + \dot{\phi}^{(t)} dt \\
 \dot{\psi}^{(t+1)} &= \dot{\psi}^{(t)} \\
 \dot{\theta}^{(t+1)} &= \dot{\theta}^{(t)} \\
 \dot{\phi}^{(t+1)} &= \dot{\phi}^{(t)}
 \end{aligned}$$

We then implemented a Kalman filter to predict our current state given our previous state and the sensor readings. Making this task tricker, we use all integer math, as it

⁹see: <http://www.movesinstitute.org/~zyda/pubs/Presence.1.4.pdf>

decreases the processing time on the ARM substantially over floating point processing. Currently we are using a PD controller to control the servos based upon our current state estimate, but we are thinking of learning the response given the state using neural networks or Gaussian naive Bayes classifiers.

4 Project History

We started this version of the project in late February 2004, after the unfortunate crash of the Wingo required the construction of a new, stronger plane. After a couple weeks, we had acquired and assembled the Zagi, and had it operational by radio control. On, March 10th, we glide tested the plane and assured ourselves that the plane was strong enough to holdup to minor crashes, and had enough lift to fly with the additional sensor payload. However, after a couple flights, we noticed that the processor seemed to have browned out, a fact that we attributed to the jarring incurred by hitting the ground. We went back to the shop to see if we could determine the source of our electrical problems.

The next week, after finding no loose wires, we tried flying the plane again. The plane was flying favorably for several minutes until we were about 300 feet above the road between Porter Hall and Phipps Conservatory. Then, all of a sudden the plane did a spiral of death, and crashed into the road, causing the ailerons to come flying off, and the payload hatch to spring open. To compound our problems, the plane was immediately hit by a car (fortunately we didn't hit a car during the death spiral!) resulting in even more structural damage.

We spent the following week rebuilding the aircraft and trying to determine just what went wrong. Since the processor was off by the time we got the plane (after it had crashed and got run over), we were unable to get data off the plane to determine the exact cause of the the crash (either software, processor, or pilot error). We have subsequently assumed that the error was caused by the same processor brownout noticed the week before and set out to eliminate all possible causes. We resoldered the board and added a capacitor over the leads to the electric motor. We also noticed that may have been a correlation between the battery switch position and crashes. Thus, we relayouted all of the components in the payload bay and added foam to keep them from moving. Tests in our offices have shown that at least one of these modifications seems to have fixed the problem most of the time; we have only observed one brown-out since March 20th, and we cannot find any way to cause the processor to choke, including throwing the plane at the ground as hard as possible.

Work in April and the beginning of May has been centered on calibrating the sensors, setting the timing of the control loop, and implementing the Kalman Filter. Following the lead of the autopilot project¹⁰, we experimented with integrating the rate gyros to get angular position and then trying to determine the angles ψ , θ and ϕ from the integrated rate gyros. After considering this solution for sometime (it was enticing because the documentation was quite good), it became apparent that their solution was actually ignoring the values from the rate gyros, and was basing angular positions solely on the accelerometers; from this experience, we learned that papers published by those who have working UAV's aren't always correct.

¹⁰<http://autopilot.sourceforge.net>

Another problem we are having with the Kalman Filter comes from using Euler Angles as was done by the BYU group. Using Euler Angles, there are at least two different ways of describing the same orientation, and this problem becomes exacerbated when the pitch of the airplane is either 90° or -90° . In this case, yaw and roll measure exactly the same angle, and implementations of the Kalman Filter which are trying to determine yaw and roll completely break down. We have implemented hacks in the code to constrain the pitch between $[-\pi/2, \pi/2]$, but a better solution to this problem is to use quaternions, which is what we are planning on doing in the near future.

5 Current Status

As of this writing, we have the control board, and the GPS working correctly. We have not been observing the brown-outs that caused the plane to crash in §4, but we can't eliminate the possibility that such brownout will occur in the future. We plan to rectify this problem by enabling the watchdog timer on the ARM, and rebooting the processor and attempting to restore the memory state if the ARM appears to have choked. We are also in the process of adding the Bluetooth module for better commutation with the plane while it is airbourne. Currently we find that a good deal of our time is sucked up download the data off of the aircraft after test flights. We hope to reduce this annoyance by downloading the data while the plane is flying using the Bluetooth module.

On the software side, we have an implementation of a Kalman Filter that seems to work during tests in our offices, as well as while the plane is in the air. However, the Kalman Filter gets really confused by the launch of the aircraft (which is one of us throwing it really hard in the X direction) and currently takes a few seconds to recover. However, we are worried that in the time necessary to recover, the plane will have met some ill-fated demise, so we haven't tried the autopilot yet. More tests are required to determine just how long the Kalman Filter is unstable. One solution we have been thinking about is open-loop control for the first second or so of flight. However, our launches are rarely exactly level or uniform, so developing a pre-defined sequence of control outputs that will keep the plane from stalling or crashing given the variance of our launches may be difficult.

We plan to continue working on the project and expect to have a working prototype by the end of the summer. Please note that the current status of the project can be obtained from our website <http://gs3636.sp.cs.cmu.edu/uav/>