COAST OPTIMIZATION SYSTEM

Frank Hermes, NAR 88930 TRA 12122 L3

Introduction

When I first got back into model rocketry (January 2009), I was enamored with the high power rockets I saw at my first launch. However, I was less impressed by the 2-stage rockets that I saw. Most of them either did not ignite the second stage, or they flew in crazy arcs or blew up or did not fly at all. Most of the more successful ones were the smaller, sport-rocket types that seemed to work okay using simple instant, motor-to-motor ignition. Though the concept of staging seemed kind of interesting, I did not think too much about it since it seemed either too difficult, or conversely, too simplistic. Instead, I focused on working toward my TRA Level 3 with single stage rockets.

Somewhere around the time I was working on the build of my Level 2 project and reading like mad to try and assimilate all that I could to be sure things worked correctly, I came across G. Henry and Bill Steins' book, *Handbook of Model Rocketry*. In it they discussed several aspects of staging, but again I did not pay much attention at the time, nor did I when, sometime later, I read Mark Canepa's excellent book *Modern High-Powered Rocketry 2*, only glancing at the sections on staging.

After a successful Level 2 flight (March 2009), I rapidly worked toward my Level 3, all the while really enjoying the learning process and the excitement of each of our club's monthly launch days. I was an avid reader of everything I could find about the hobby, and very grateful to have the power of the Internet to assist me in my research. When I first started out I thought I would be one of the rocketeers to always keep things in sight, but over time I became challenged by seeing others do higher altitude flights and recovery. I realized that I particularly love all the electronic gadgetry involved and the more technical mechanical aspects one must master in order to fly straight and recover high-power projects intact.

One day when I was searching for some information in Tim Van Milligan's *Apogee Newsletter* archives, I stumbled across an article on staging. In it Tim discussed how to achieve maximum altitude. He was discussing a *coasting period* between booster burn out (BBO) and sustainer ignition…hmmm…what he was saying contrasted with what I remembered reading in Steins' book. I remembered vaguely that Stein advocated performing sustainer ignition right after BBO, in this way you would be coupling the maximum velocity of the booster driven rocket with the maximum velocity of the sustainer rocket in order to achieve maximum altitude. However, when I went back and reread Stein, I realized he qualified his statement and indicated he was "neglecting aerodynamic drag". I was a bit confused at this point so I went back and reread Canepa – he drew the distinction of attempting to achieve maximum velocity versus maximum altitude and that you needed two very different approaches.

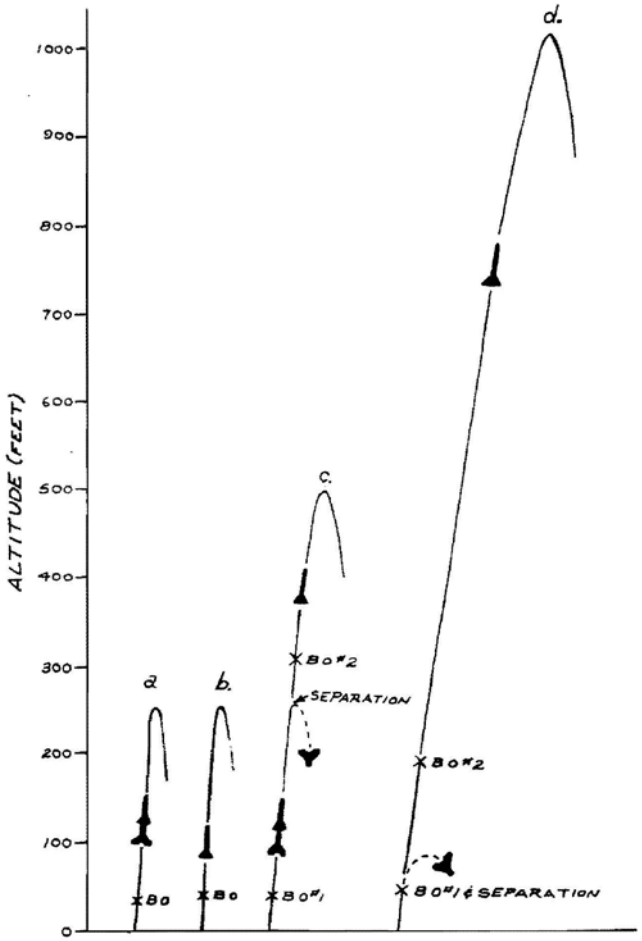


FIGURE 11-3 **Series staging** at various points in a model rocket flight. (a) Lower stage **only ignites.** (b) Top stage only. (c) Model stages at peak altitude of **lower-stage flight.** (d) Model stages at lower-stage burnout (BO) point and **therefore at maximum**-stage velocity.

Graphic from: *Handbook of Model Rocketry*

Van Milligan pointed out that Stein's instant ignition would achieve maximum altitude only if you were in a vacuum since parasitic drag is induced by our atmosphere. In fact, such drag is proportional to the *square* of the velocity, so the higher velocity gained by instant coupling is precisely the wrong approach to maximize altitude. What Canepa, Van Milligan and others point out is that *minimizing* velocity, and therefore drag, will generally achieve the higher altitude for the same total amount of thrust.

My simulations in RockSim show an average increase in altitude of about 30+% when the coast period is maximized verses igniting the sustainer motor immediately after booster motor burnout. That is a significant difference.

Since I was developing this urge to go high and realizing that staging was one way to accomplish this, I started thinking more about the coast period between sustainer ignition and BBO. I was intrigued. Just how would you go about maximizing the coast period? After a successful Level 3 certification (June 2009), I began to wonder just where to go next. Maybe I had the answer of how to maintain my interest

in high power model rocketry. Maybe staging could prove interesting after all! It obviously was very challenging.

From what I could determine, most high power staging ignition is done with timers. But that seemed rather crude to me. If my goal is to achieve maximum altitude, how much time should I dial in to maximize the coast period? I could guess at how much time to allow after booster burn out. Or I could run simulations in a program such as Apogee's *RockSim*, or, and probably the most expensive route, I could just experiment with test flights. However, to really attempt to maximize the coast period I would have to have something better than a good guess. Even if I were able to perform a very accurate estimate of the coast period, it would still be assuming the actual flight went exactly according to my assumptions. How would I allow for such things as inconsistent motor burns, erratic flight patterns, the weather and other assorted conditions that I could not reliably predict?

I thought about current day electronic altimeters. Prior to altimeters being adapted for use in high powered model rockets, most deployment was also based on timers, either the relatively simple motor ejection charge delay used as the igniter, or mechanical or electric timers. The basic problem with timers is that they are not dynamic in nature. That is, if everything goes according to plan in regards to a flight's performance, then the timer works well. However, as we have often seen out on the launch site, things do not always go according to plan – maybe seldom is a better word! For example, if a motor's burn is shorter than expected, or produces less thrust that expected, apogee will occur sooner than expected. Any delay that was designed into a timer to deploy at apogee will be later than desired and deployment will not occur at the slowest point in the rocket's flight, but rather at some higher speed as it is accelerating on its downward path. However, if an altimeter was used in that same scenario, it would be looking for the apogee event in real-time, based upon its sensor readings, not a prescribed time – it is able to *dynamically* adjust the deployment point to suit the actual flight conditions incurred.

I wondered if I would be able to develop some sort of sensor system that was analogous to the altimeter to dynamically maximize the coast interval. Could I find something to replace a simple timer for staged ignition? The following article describes my trial–and-error journey to produce just such a coast sensing system!

The Problem

So what factors or issues would I need to consider in order to replace the staging timer? What is it about the rocket's flight that characterizes an ideal coast period anyway? Just how would you describe to someone what has to happen between booster burnout (BBO) and a point in the flight path that would be considered as the longest possible coasting interval that would result in the rocket achieving maximum altitude?

In the ideal world, after BBO, your sustainer rocket would fly perfectly vertical all the way to what would be apogee, but, just prior to it "falling over", you would ignite an "instant on" motor that would come up to pressure immediately and send the sustainer hurtling upward.

From a more realistic point-of-view, we want to initiate sustainer ignition just at the point that we have enough time to bring the motor up to pressure while the rocket is still flying fast enough to maintain its "relatively" vertical attitude. So, we need to be able to monitor both velocity and verticality on a real time basis and make decisions based upon that information.

Let's go back to that ideal world picture and look at only at verticality aspect. Let's assume that we've been able to design the rocket so it flies perfectly in the vertical all the way to apogee. And let's assume we have a special sensor on board that is able to monitor the flight in real time and predict well ahead of time just how many seconds into the flight that the rocket will attain apogee. Then all we would have to do is ignite the motor at precisely the point in time in the vertical flight path that leaves us enough time to get the motor up to pressure before the rocket falls over at apogee. In other words, we simply subtract the time in seconds that it takes the motor to ignite and pressurize from the time it takes it to reach apogee.

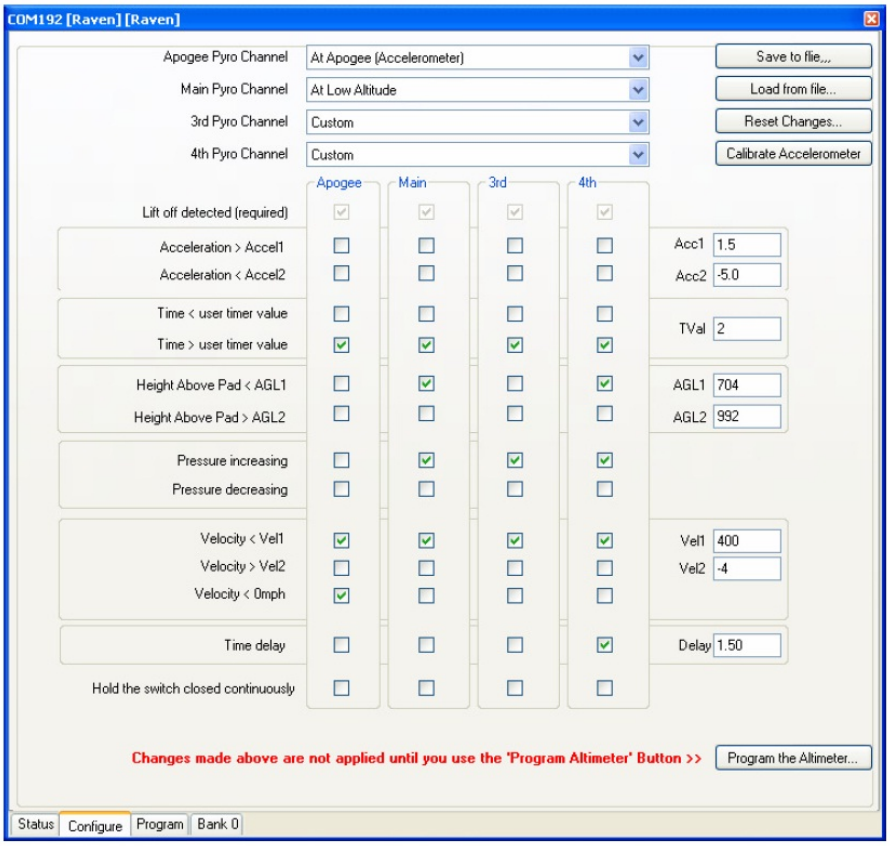
Hmmm…that's sounds straightforward enough. The problem is that nothing is that perfect. Aside from the fact that seldom does a rocket fly vertical all the way to apogee, we never know for sure ahead of time how long it will take a rocket to reach apogee due to variances in real performance versus specifications or simulations, or the current weather, booster motor performance, etc. If we could characterize and determine or predict what that value was however, such a predictive apogee sensor system would offer a *dynamic* trigger point, analogous to an altimeter.

Looking at it from the velocity aspect, if we knew that our magic rocket would fly vertical all the way to apogee, i.e., where it reaches zero velocity, we could pick a velocity point, e.g., decreasing through 200 MPH, prior to that which allowed us to pressurize the motor in time before the rocket arrived at apogee. Many of the popular commercial altimeters, or *flight computers*, out there today have the ability, either through on board accelerometers and their micro-controllers, to be calculating velocity in real-time. So, if we knew we would stay vertical throughout the coast period, we could fairly easily determine through flight test trial-and-error or simulation, a velocity-based ignition trigger point.

The problem remains that in the real world seldom do flights stay that vertical that long. So, the reality is that we can only get so close to that perfect flight path and therefore a coast maximization system needs to be a bit more complicated.

Verticality

Adrian Adamson at Featherweight Altimeters has provided some access to verticality and velocity as a trigger with both his Parrot altimeter and the newer Raven. With the Parrot or Raven you can access a parameter in the Featherweight Interface Program (FIP) "decreasing through nnn MPH" as an event to control the output of any of the output/pyro channels. The particular velocity is fixed in the Parrot, but configurable in the Raven. Black Magic Missile Works' *UFC* series also makes such an event readily available.



Featherweight Altimeters - Raven™

Additionally, flight computers such as the Parrot/Raven and the UFC can offer a few more indicators that the rocket is probably still flying upward. Equipped with an accelerometer and timer, as well as a barometer, such flight computers can assure that the pressure is still decreasing (lesser air pressure as you ascend) and that the rocket is at a predictable/configurable minimum altitude after a certain amount of time – all indications that the rocket is still ascending. Assuming that the flight is perfectly vertical all the way to apogee, use of one of these "vertical-check" flight computers is basically all you need.

However, a more likely scenario is that the rocket will not fly perfectly straight and will be arcing over to some degree or another. What is missing from the vertical-check flight computer is a monitoring of the angle off the vertical that the rocket is experiencing.

The rocket may still be ascending per the vertical-check parameters, but do you want to trigger ignition if the rocket is flying at an angle of say, 30 degrees off vertical? What about 45, or 60? High altitude recovery is tough enough without setting yourself up for a very long retrieval or loss of the rocket, or even worse, a flight into the crowd. To be in even a more dynamic triggering environment, we need to be able to measure the angle off vertical in conjunction with the other information before deciding to trigger sustainer ignition.

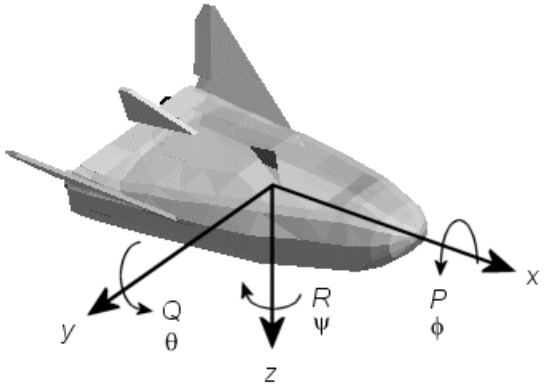
How do you determine tilt? Sensors that may come to your mind might include mechanical sensors, accelerometers, gyros or magnetometers. Since I did not really know what I was doing at the time, nor much about the more sophisticated sensors, I initially only considered mechanical tilt meters, such as you might find on “tilt-over” shipping containers to record any damage or adverse handling during the shipping process or maybe recreational vehicle levelers. I had also seen some sort of electronic tilt meter in the tools catalog of MicroMark. Could one of these mechanical devices reliably indicate tilt when bolted inside a high power rocket? I thought so.

I looked for inexpensive tilt meters with some sort of useable output like a relay or switched contact and came across some from a company named Rieker. Rieker produces various commercial tilt meters for many different applications. These sensors are normally used to detect whether an article has been subject to tilt outside a prescribed range, for example, to detect shipment damage or as a heavy–equipment vehicle “tilt-over” safety device. They can be ordered with various options, including specifically what degree of tilt is required to trigger their output. They have both single-axis and dual-axis devices. A configurable dual-axis, X and Y, sensor seemed perfect for what I wanted to do.

Rieker SlopeAlert™

<u>Coordinate Systems</u>

Envision tilt detectors placed in the avionics bay of a rocket sitting on the launch pad. You are facing the rocket, which is perfectly vertical, and looking straight ahead to the north. Though there are no pre-defined absolute model rocket coordinate reference systems, for purposes of illustration, we will refer to the axis that runs through the rocket from south to north as the X-axis, and the one that runs through it from east to west as the Y-axis. The Z-axis of the rocket runs through the middle of it from top to bottom. This will be our rocket’s X, Y, and Z coordinate frame of reference, and it is for our purposes defined with respect to the earth. So, with our rocket sitting vertical on the pad, we are arbitrarily defining that the rocket’s and the earth’s two reference systems coincide…we will discuss those frames of reference in more detail later on.



Aircraft Coordinate System

The Plan

I ordered the Rieker tilt meters to trigger an output whenever they exceeded 15 degrees of tilt. I placed two of the one-axis Rieker sensors one atop the other and rotated one by 90 degrees. So now I had a two-axis, X and Y tilt meter that gave me an output signal whenever the tilt exceeded the 15 degrees in any direction (or so I thought at the time). My idea was that once the angle of the rocket became 15 degrees from vertical, or, if the vertical-sensing flight computer triggered its output channel, I would trigger sustainer ignition. My thinking changed over time, in particular the use of the tilt angle as an abort mechanism rather than a trigger mechanism, but I will explain all of that as we continue to work through my learning process.

Gravity

I discussed my plans with a physicist friend of mine. I explained to him what I was doing. He asked me to explain the specific technology that the Rieker units were based upon. I did not know exactly so I went to the Rieker web site and deduced (never confirmed) that the devices contained a upward-pointed curved tube running along the orientation axis with some viscous substance to dampen out the travel of a ball that would travel inside the tube and cause something to trip once the prescribed angle was reached. He thought about things for awhile and made me realize that during the period of interest, i.e., coasting, the sensors would be experiencing negative gravity - as soon as booster burnout occurs, the positive g force the rocket was under quickly switches to negative g. The ball then, rather than staying in the bottom of the tube, as it should, would be floating up (think of yourself floating in a falling elevator whose cable had become unattached) and very prone to travel up the curved tube as soon as the slightest tilt occurred. That is when a serious recognition of the role of gravity began to enter the picture.

My first thought was ingenious (or so it seemed for a moment!) – just invert the Rieker sensors so that the tube would be re-oriented into the proper position for negative gravity. We thought about this for a while and came to realize that mechanical technology similar to the Rieker units would be troublesome at best.

Dealing With The Data

Even while I was struggling with identifying a sensor to use, I knew that eventually I would have to somehow be able to actually use the output of whatever sensors I decided on. I started thinking about how to do that. I knew I would have some electrical output from the sensors. I was very familiar with electro-mechanical relay logic, but also knew that mechanical devices in a high g environment were prone to failure. I started thinking about using solid-state digital logic devices and at first thought I would use solid-state relays, but then I thought maybe solid-sate logic could work, particularly since I only had a few outputs to deal with. I was only slightly familiar with transistor-to-transistor logic (TTL), but assumed I could figure it out. I read about the various configurations of the available chips and then ordered a bunch of AND, NOR, and NAND chips.

Before I could get too far into the TTL stuff, I stumbled across the Parallax web site and saw that they offered a tutorial starter kit for microcontrollers. A microcontroller is a device that includes a microprocessor (a small version of the microprocessor in your home or office computer) along with other embedded peripheral devices such as input and output ports, power regulators, and memory. I knew of microcontrollers for a long time, but always thought they would be far too complex to learn. But now I had a mission that seemed to be a perfect fit for such a device, so I thought I would try to learn how to use them. I ordered one of the starter kits and quickly was immersed in the tutorial.

As it turns out, microcontrollers are fairly straightforward. I had a basic understanding of electronics and had done a bit of programming at various times in my life, so it was not too much of a stretch to combine the two and become familiar with the microcontroller. After going through the exercises in the book I felt relatively comfortable doing the simple things I thought I would need for my project. Later on, I took a Saturday morning microcontroller course at the local junior college to get a more in depth experience.

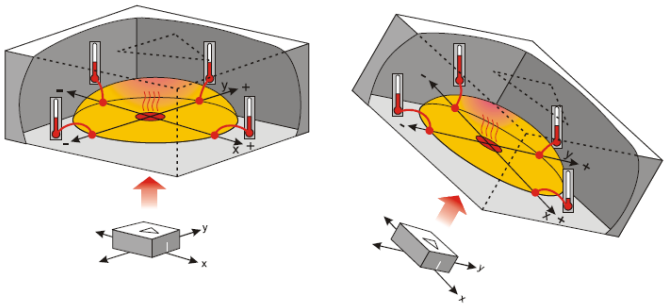
Simply stated, the microcontroller allows you to take various inputs from sensors, set up some programming to deal with what you sensed on the inputs, and then control output devices based upon those decisions. Perfect for what I needed to do to combine the vertical-sensing flight computer and my tilt meters and output a signal to the flight computer for motor ignition!

Accelerometer-based Tiltmeter

One of the exercises I came across while learning the Parallax BASIC Stamp microcontroller was an experiment using an accelerometer to sense tilt! Now that's what I'm talkin' about! Wow, something handed to me on a platter! The sensor device used in the exercise was a Memsic 2125 2-axis (X and Y) accelerometer that used a temperature differential inside a closed chamber to sense changes in gravity to its orientation. I started thinking that maybe this was the answer to the concerns about the mechanical "rolling balls in a tube", since this sensor seemed more "solid state".

Theory of Operation

The MX2125 has a chamber of gas with a heating element in the center and four temperature sensors around its edge. When the accelerometer is level, the hot gas pocket rises to the top-center of the chamber, and all the sensors will measure the same temperature.



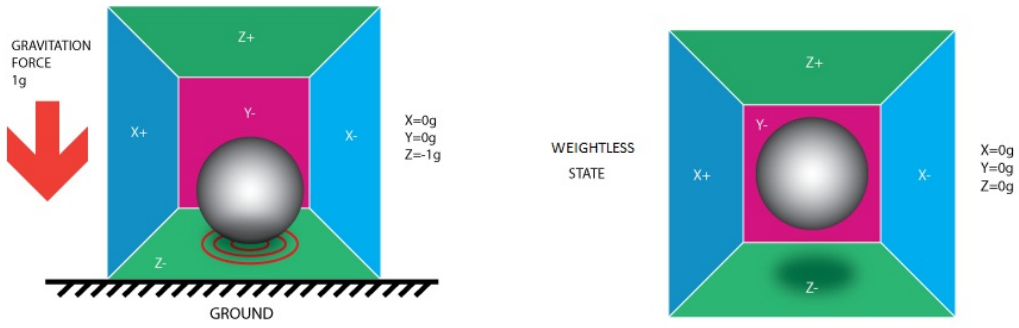
By tilting the accelerometer, the hot gas will collect closer to some of temperature sensors. By comparing the sensor temperatures, both static acceleration (gravity and tilt) and dynamic acceleration (like taking a ride in a car) can be detected. The MX2125 converts the temperature measurements into signals (pulse durations) that are easy for microcontrollers to measure and decipher.

I assembled the MX2125 circuit, copied and revised the code a bit, compiled it and downloaded the code into the microcontroller. The essential code was provided thankfully in the exercise. It used changes in the sensor output as the gas ball traveled along the walls in response to the tilt I applied to the chip. The code included the trigonometry calculations to develop the angles from the change of values presented at the chip output. It worked great. I set it up to trigger an LED whenever the absolute value of the tilt angle exceeded 15 degrees on either the X or Y-axis. Thought I was home free!

Gravity, Another Warning

About this same time, I was at a ROC launch in Lucerne Valley prepping my 2-stage for a test flight. A fellow wandered over and watched while I was assembling one of the Roush Tech CD3 CO2 charges I was using for deployment. I mentioned that I was investigating maximizing the coast interval and told him about my Rieker units and the Memsic accelerometer-based tilt meter I had recently created. He paused for a moment and shook his head. I asked him what was wrong. His reply was that any sensor that is affected by gravity would not work. I was busy trying to get launched before the winds came up so his comment did not fully register at the time. Later, when discussing the topic again with my physicist friend, he had come to the same conclusion.

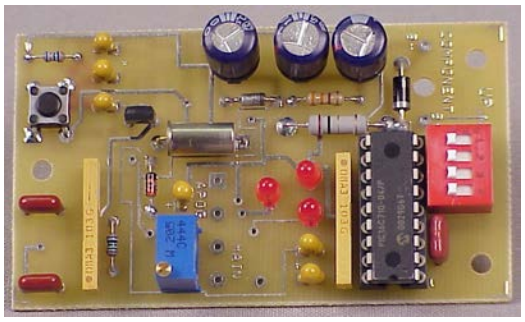
The problem gets back to the two different frames of reference I mentioned previously – the rocket's coordinates and the earth's coordinates. When the rocket is sitting on the pad vertical, the two systems are aligned. It is the earth's frame of reference that is important, since it is in that frame of reference that tilt has its basis – i.e., the 15 degrees of tilt that is of interest is in the earth's frame of reference relative to the Z axis (verticality).



Once the rocket leaves the pad it loses its physical reference to earth. If it has a sensor on board that can sense the gravity vector that it felt while it was sitting on the pad, it would still know whether it was vertical or not. However, once the motor ignites and the rocket lifts off, it is under a g force that exceeds 1 g (earth's gravity). Any sensor that is monitoring gravity will lose any sense of the earth's pull and its associated reference – the sensor cannot distinguish the difference between gravity and acceleration. Therefore the tilt in the earth's frame of reference cannot be measured any longer. And, the converse is true – when the rocket reaches motor burnout the rocket is weightless for an instant and begins to experience negative g, the same phenomenon occurs but in the reverse. Unfortunately, even my thermal-based Memsic accelerometer is affected in this way. Hmmm…I was back to square one. This was getting harder all the time!

Alternatives

So now what? What other sensors could I try that would not be affected by gravity? I knew a bit about magnetometers and had seen some ads for the Transolve Flux Capacitor that relied on a magnetometer to sense apogee.



Transolve Flux Capacitor

Magnetometers. My physicist friend was very familiar with magnetometers and after realizing the issue with accelerometers, had come to the conclusion that magnetometers might be an ideal solution. The magnetic flux lines that extend from the earth are very stable and fairly easy to read. The problem with them is that they are very weak in magnitude and subject to interference from any nearby metallic influences – such as a long metal launch rail!

Anything that can influence the magnetic reading can be calibrated out if its influence is constant – e.g., if you house the sensor in an avionics bay with steel connecting rods, the magnetic distortion caused by the rods can be deducted from the reading effectively canceling out the rods' influence - they always stay in the same orientation relative to the magnetometer. However, the launch rail is there while the rocket is sitting on the pad, but once it is launched its influence goes away. This makes the problem of dealing with a magnetic sensor a bit troublesome. But, it became a clear candidate in my mind at that point. You could after all, calibrate the sensor away from the pad just prior to launch, load the rocket on the rail, then when it lifted off the calibration you did would take effect.

GPS. GPS sensors also came to mind, but the resolution afforded does not really match up to the requirements for a rocket moving so quickly in such a small relative flight path footprint. I discounted use of a GPS sensor fairly quickly.

Gyros. Having been a jet fighter crew chief while I was in the Air Force in the late 60's and later getting my private pilot's license, I was aware of gyroscopes, but my vision at the time of gyroscopes was that of a huge spinning ball, and a very expensive ball at that. I quickly discounted gyros at that point too.

I was clearly running out of options.

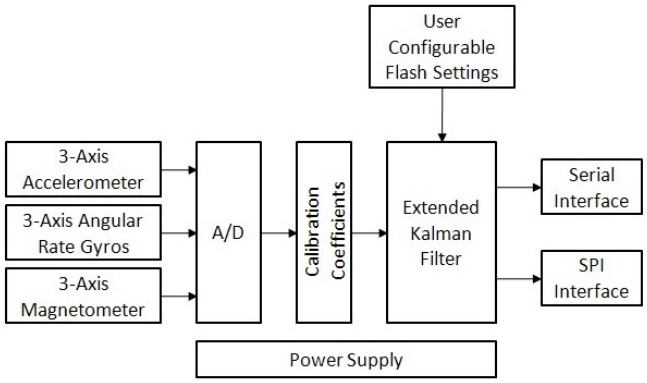
Thermopiles. I did some more research on line and found several references to "level sensors", inclinometers and tiltmeters, but once again, they all seemed to be based on accelerometers or balls-in-a-tube or similar gravity based sensors. Then one day after continually changing the arrangement of my search words in Google, I came across a reference to "thermopiles" being used to sense the horizon by contrasting the sky's IR signature with the earth's. I eventually was drawn away from using the thermopiles and never really explored them for a couple of reasons. For one, it did not seem that the thermopile sensors would give me the accuracy or preciseness of angle that I wanted. Additionally, it was recognized that the thermopile sensors did not work very well under a cloudy sky.

IMU. One of the references I saw on the internet mentioned something called an IMU to replace the thermopile sensor. Seems that it was difficult to fly the autopilot when it got cloudy, or when they had to come inside if the weather was bad - obviously a horizon sensor was not much good in those environments.

What's an IMU I wondered? I seemed to remember something about IMU's mentioned while I was watching some programs on smart bombs and cruise missiles. Off to Google again!

An IMU is an inertial measurement unit. It usually uses combinations of sensors to determine changes in a vehicles orientation. The sensors are typically accelerometers, gyroscopes and magnetometers. How many sensors incorporated into an IMU is referred to as its number of degrees of freedom (DOF). For example, if you have an IMU with accelerometers for the X, Y and Z-axis and a same number of gyroscopes, you have an IMU with 6 degrees of freedom, commonly abbreviated as 6DOF. Add three axes of magnetometers and you have a 9DOF IMU. Using the IMU to transform the vehicles orientation to the earth's orientation turns an IMU into an AHRS – an attitude/heading reference system. Now I seemed to be getting somewhere!

Some of them appeared to output only the rate of rotation from the gyros (IMU), while others actually output Euler angles or vector references. One of my favorites among the more sophisticated ones is the VectorNav VN-100. The VN-100 is a great, 9DOF IMU/AHRS with an easy to use interface. As all the technology involved in getting to where I wanted to go got more and more complicated, discovering these all-in-one devices started me thinking it might be a lot easier to just bite the bullet and order one that output angles already – I would be nearly done!



VectorNav VN-100 IMU/AHRS

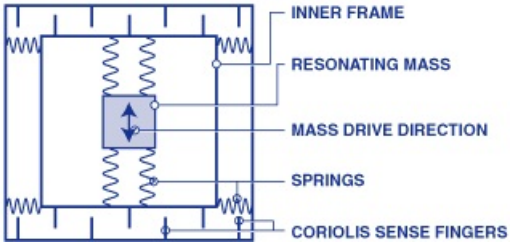
But, they are very expensive (for good reason, as I would better learn later on). I thought I should be able to come up with what I needed relatively easily using a couple of gyros. I did not need all the sophisticated sensor systems offered. All I needed were angles. Gyros were designed to sense rotation and be integrated into angles, right? I should be able to grab a gyro, do a bit of coding, and develop my own tiltmeter! Seemed easy enough and the individual gyros were not terribly expensive.

Micro-electro-mechanical-system - MEMS

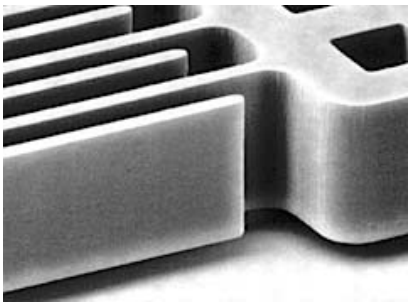
More Internet searching gave me references to several different IMU and AHRS devices, as well as individual gyroscopes. I was surprised however at how small the gyros were – again, I envisioned gyroscopes as large round objects the size of grapefruits. I noticed mention of the use of MEMS technology for the gyros.

MEMS – micro-electro-mechanical-systems – this refers to a relatively new (early 1970's) technology allowing extremely small mechanical devices to be manufactured using technology similar to making electronic chips. Layers of material are photo etched onto a substrate and actual micro-machines are made! Rather than relying on large spinning balls, these gyros use MEMS technology to fabricate devices that are smaller than a match head.

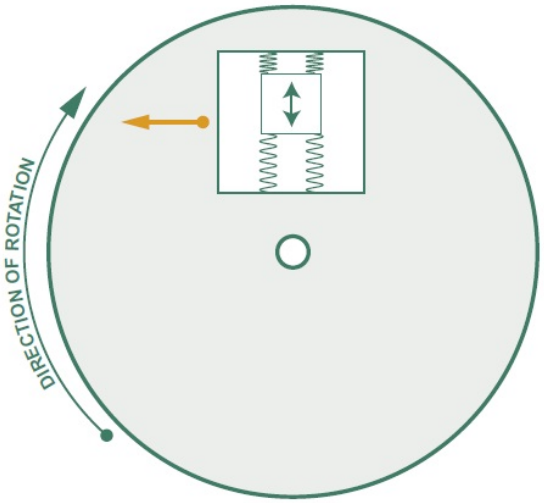
Rotating the gyro sensor subjects its structure to the Coriolis force. You are familiar with the centrifugal force that causes objects to move away from you on an outward trajectory – the Coriolis force an associated lateral component. The Coriolis force causes a mechanical structure to vibrate inside the MEMS chip. The resulting vibration is detected by measuring the capacitance change between the mechanical structure and some fixed electrodes, i.e., mechanical fingers move together and apart as the sensor turns about its axis. The associated change in capacity between the fingers is output from the sensor via voltage that varies with the amount or rate of rotation experienced by the sensor.

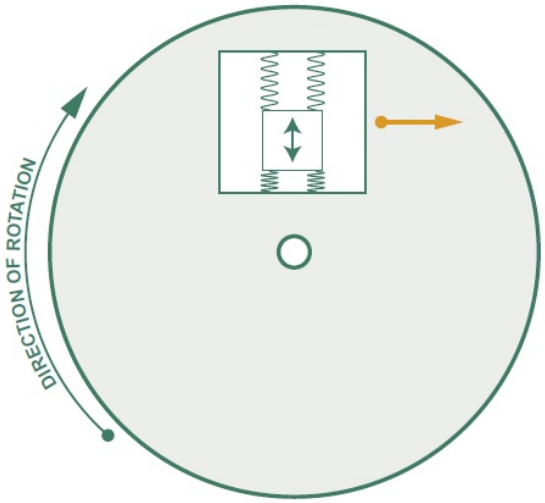


MEMS Gyro Mechanical Structure



MEMS "Tuning Fork"





Gyroscopes

OK, how do I go about actually using these MEMS gyros? I was not even sure where to start. I went back to SparkFun and saw that they had gyros on what they called "breakout boards". A breakout board usually mounts the device of interest along with a voltage regulator, capacitors, resistors and the like to allow you to actually access the particular device – they are similar to the development boards for the microcontrollers I had been learning about.

Typical Breakout board

But which gyro to use? What were the criteria for gyros to build an effective tiltmeter? And, just how does one go about turning the output of a rate-of-rotation sensor into an angle? I slowly started to

realize what I did not know and seriously re-considered just ordering a developed AHRS with angles already available to me - I decided I still had a long ways to go in this project.

I pushed on. I began by reading the specification sheets that were on the web store at SparkFun. SparkFun is a wonderful site for experimenters and those dabbling in electronics, and microcontrollers and sensors in particular. Each device has its own page listing the device, documentation for the device and references to other devices that you may want to use along with the primary one. There is a wealth of useful information, tutorials, and even sample projects.

It seemed that the relatively inexpensive gyros that were on breakout boards from SparkFun were analog output devices. After reading the specifications and thinking about what they might indicate for my application, e.g., their rated output scales and sensitivity to linear acceleration, I decided that I would most likely use gyros from either Invensense (the IDGnnnn series) or Analog Devices (the ADXRS6nn series), both of which had analog output. Since the BASIC Stamp microcontroller from Parallax that I had learned about in my course of study only had digital inputs, I went back to the Parallax site, researched some more and ordered a supplemental course on analog-to- digital conversion (ADC). I also started investigating just how those IMU/AHRS devices turned the output of a rate of rotation sensor into an angle.

Analog Sensors and Analog-to-Digital Conversion (ADC)

Digital devices can be compared to a toggle switch that controls a light in your home – it is either on or off. An analog signal can be compared to a rotary dimmer controlling another light in your home – it can have an infinite number of positions between on or off. In electronics, variations of signal for a digital device are defined by incrementing or decrementing discrete numbers as opposed to the infinitely variable signals associated with analog devices.

The microprocessor embedded in a microcontroller can only directly work with digital signals. It can only distinguish between a signal that is either off, or on – a binary bit (or in binary, an 0 or a 1). If we need to have the microprocessor deal with an analog device such as the output of an analog rate-of-rotation gyro, we have to somehow convert the analog output signal into a digital signal in order for the microcontroller to be able to use it. Some microcontrollers incorporate analog inputs, i.e., they integrate and embed ADC into their digital inputs within the microcontroller chip itself. But at this point in my journey I was not familiar with anything but the BASIC Stamp series. That series of microcontrollers only has analog inputs, so, I needed to learn about ADC.

Analog Gyroscopes

Bias Analog sensors are usually standardized in their output. They typically output in a range of industry standard voltages of 0 to +3.3, or 0 to +5 volts. A gyro for example might output 2.5 volts when standing still and +4.5 volts when turned clockwise at a rate as fast as it was designed to detect. If you turned it counter-clockwise at that same rate, its output would be 0.5 volts. This gyro would be considered to

have an output of 2.5 volts, +/- 2 volts full range. The voltage output from a gyro that is standing still will usually have a voltage greater than zero in order to keep the output voltage in both directions of rotation a positive voltage (note our example above). The value of the steady state output is called the bias voltage or the DC offset voltage.

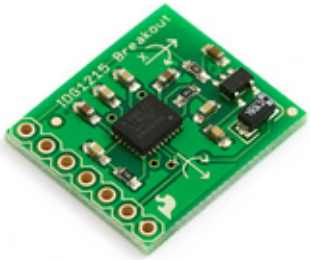
Gyros are usually linear in nature, so if you rotated our example gyro clockwise at *half* its design rate, its would output +3.50 volts (4.5 volts max rate output, minus 2.5 volts offset, divided in half, equals 1 volt – adding the 2.5 volt offset back in gives you an output of 2.5 volts plus 1 volt, or 3.5 volts).

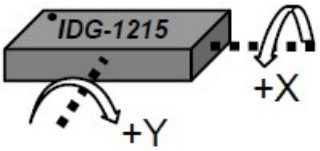
Saturation There are an infinite number of rates at which you could turn the gyro within its design parameter and a corresponding infinite number of output voltages that would be output – hence it is considered an analog device. If you exceed the rate at which the gyro was designed to operate, it will "saturate" and the output would indicate incorrectly. For example, if you had a gyro rated at 100 degrees per second, and you rotated it at 125 degrees per second, it would saturate.

Resolution Connecting the output of an analog gyro into the input of an analog-to-digital converter would give you a digital output that would correspond to the rate of output from the gyro, but it would come in discrete (digital) steps that the microcontroller could understand. ADC's are defined by their "word" length, or the maximum number of bits they will output for each reading of the gyro – the bigger the word length, i.e., the more bits, the better the resolution of the information being converted from analog to digital.

Standard ADC resolutions are usually in a range of 8, 10, 12 or 16 bits. When the ADC chip samples an analog signal at its input, it divides the signal into a number corresponding to the chip's word or bit structure design, then outputs that information digitally for reading by the microcontroller. An 8-bit ADC divides the input signal into 256 parts (low resolution), a 10-bit ADC is 1024 parts, 12-bit is 4096, and a 16-bit chip divides the signal into 16,384 parts (very high resolution). In other words, if you had a full range analog signal of 1 volt into an 8-bit ADC that was moving between 0 and 1 volt, the resolution would be changing in the range in steps of 0 to 256. For example, a reading of 0.5 volts would translate into a value of 128. However, that same voltage applied to a 12-bit ADC would be changing in steps of from 0 to 4096. That 0.5 volt analog signal would translate into a value of 2048 – much higher resolution, or accuracy of the data being read. The amount of resolution you need is a function of the application you are working on.

Sensitivity Gyros are also rated for their sensitivity, or the degrees per second that corresponds to a given output voltage. I was experimenting with an Invensense IDG1215 gyro. The IDG1215 is a dual axis (X and Y) gyro rated for a full-scale output of 67 degrees of rotation per second. Another version of the IDG gyro, the IDG500, is rated at 500 degrees per second. Since I do not need that amount of rotational sensing for my rocket, I used the lesser-rated sensor, which, for the same absolute value of voltage output will result in more sensitivity per volt output. That is, I would see a 1.0 volt swing for a rotation of 67 degrees, rather than a 1.0 volt swing for a 500 degree rotation. The IDG1251 would be considered more sensitive than the IDG500. This greater sensitivity would result in greater precision when used in the calculations used later on to create actual angles from the rate-of-rotation.

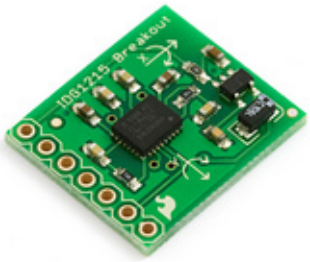


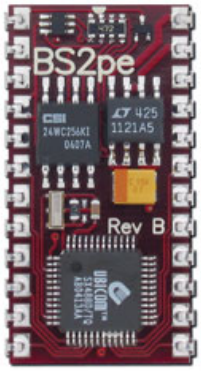


Invensense IDG1215 2-axis MEMS Gyro SparkFun Breakout Board

Output, Sensitivity and Resolution

So, armed with all of the device parameters and specifications, we can get back to the actual output of an analog sensor and its relationship to the ADC and the microcontroller. I will use the example that I was working with initially. My setup included the IDG1215 gyro, a 12-bit analog-to-digital converter and the BASIC Stamp 2pe (BS2pe) microcontroller, all mounted on a Parallax BASIC Stamp development board.





Gyro ADC Microcontroller

If the IDG1215 gyro were standing still, i.e., no rotation, its output would be +1.35 volts, since its bias or DC offset is just that. If though we constantly rotated the gyro clockwise at its maximum rate of 67 degrees per second, the output would be a constant +2.35 volts – the DC offset of +1.35 volts and +1.0 volts, which it adds when rotating at its maximum specified rate. At a constant rate clockwise of half that rate, i.e., 33.5 degrees per second, the output would be +1.85 volts (+1.35 V plus +0.5 V); at half that rate, the output would be +1.60 volts (+1.35 V plus 0.25 V), and so on. If we rotated the gyro counter-clockwise at a constant maximum specified rate of 67 degrees per second, the output would be 0.35 volts (+1.35 V plus -1.0 V), half that rate counter-clockwise would output +0.85 volts (+1.35 V plus -.0.5 V), and half that rate would produce an output of +1.10 volts (+1.35 V plus -.0.25 V). The output is

said to be linear. So, the overall output range of our gyro would be a minimum of +.35 volts and a maximum of +2.350 volts (1.35 V DC offset, +/- 1.0 V full range rotation).

Considering that the incremental output of the gyro at its maximum rating of 67 degrees per second is 1.0 volts (2.35 V minus the DC offset of 1.35 V), the gyro is said to have a sensitivity of 15 mV/degree/sec or 15 mV/d/s (i.e., the 1 volt full-range output ÷ 67 degrees/second, or 1000 mV ÷ 67 degrees/sec = 15 mV/degree/second = 15 mV/d/s, or, 0.015 V/d/s).

Reference voltage. ADC's have what is called a reference voltage. The reference voltage is the minimum and maximum voltage that will be presented at the input of the ADC. Typical reference voltages are 0-5 volts and 0 to 3.3 volts. In our example, we will assume the reference voltage for our 12-bit ADC is 0-5 volts. If you remember from the foregoing, a 12-bit ADC divides an input signal into 4096 parts, a 0 input signal would output 0, and 5 volts would output 4096 from the ADC. A 2.5 volt signal (half the reference voltage) would output 2056 (half the maximum output of 4096). As you can see, the ADC output is linear also in regards to the input. The microcontroller's input therefore sees a linear output from the ADC that represents a range of 0 to 4096 as the input to the ADC ranges from 0 to 5.0 volts, or 0 to 5000 mV.

If you divide the 5000 mV by the 4096 ADU (analog-to-digital unit), you will see that each increment of the ADC output, or ADU, is equal to 1.22 mV, or, 0.00122 V – this is called the conversion factor.

In the case of the IDG gyro, whose range is 0.35 volts to 2.35 volts, the ADU output range of a 12-bit ADC will be approximately 287 to 1926 ADU (0.35 volts / 1.22 mV = 287 and 2.35 volts / 1.22 mV = 1926). We have converted the analog output of the gyro into a digital format that the microcontroller can handle. The microcontroller is able to accept these digital signals. Once read by the microcontroller, we can store the gyro sensor data and use it in the microcontroller's programming.

Converting Rate to Angles

Now that we can sense and store gyro output, we need the microcontroller to convert that rotational rate information into an angle. This is done by "integrating" the rotational rate information over time. To illustrate, let's say that the gyro is in a body that is at rest (no rotation), then it instantly starts to rotate at a rate of 1 degree per second for 10 seconds and then instantly stops rotating. The body would have changed its orientation by 10 degrees. We determined this by measuring the rotation at a certain rate over periods of time – we took samples. We then integrated these samples into an angle by simply adding the samples together. The microcontroller is very capable of such repetitive measurements and calculations.

Sample rate. As stated above, the integration of an angular rate gyro is straightforward. We measure the output of the gyro at regular intervals, or sample periods. The samples have to be in a sufficient quantity, or at a rapid enough rate, to give you the resolution or accuracy you want. Typical rates are on the order of 20 to 200 samples per second, or 20 to 200 "Hertz" (Hz), again, dependent upon how precise you need to be relative to the sensitivity of the gyro.

Rate integration. So, armed with all of the above, I was able to determine the formula I needed to use in order to convert angular rate into angles. The derived angle is simply the aggregation of the samples:

Angle_new = Angle_existing +

Angle_incremental

[simply adding the amount of angle captured by the new sample onto the existing angle]

Or, using our specific setup of the IDG1215 gyro and the 12-bit ADC:

Angle_new = Angle_old +

((IDG_gyro_reading_in_ADU – gyro_DC_offset_in_ADU) * ADU_to_mV conversion ÷ gyro_sensitivity_in_mV/d/s * sample_rate_in_Hz)

So, for my example let's say that we are integrating the Invensense IDG-1215 gyro at a sample rate of 100 Hz, that the existing integrated angle is currently at 10 degrees, and we take a gyro sample that reads 1230 ADU (which is equal to a rotation rate of +33.5 degrees per second). The formula would be:

Angle_new = Angle_existing + Angle_incremental

Angle_new = 10 degrees + ((1230 ADU – 820 ADU) * 0.00122 V/ADU ÷ 0.015 V/d/s * 100 Hz)

= 10 degrees + (410 ADU * 0.00122 V/ADU ÷ 0.015 V/d/s * 0.01 seconds)

= 10 degrees + 0.333 degrees

Angle_new = 10.333 degrees

So, after much time and study, I had finally determined what I needed to do to build a tilt meter. I had my gyros, I had a microcontroller, I had a way to input the gyro data into the microcontroller, and I had learned enough programming along the way in Basic that I figured I could actually do this! I figured I could develop a gyro-based tilt meter similar to the accelerometer-based one I had first assembled months ago.

Implementation

Trying to actually implement the above integration formula on the integer-only math of the Parallax BASIC Stamp was a bit onerous. I was using fractional conversion factors and performing division that resulted in remainders. I had some difficulty obtaining the gyro rate integration using this approach and the relatively simple, inter-only-math Parallax BS2pe. There are integer-only conversions you can make to do so, but the accuracy is limited and the processing power to do so is lacking with that

microcontroller. I decided I really needed to move to floating point math capability. I would learn much later that a lot of my difficulty at this point was not due to these constraints, but rather another problem I did not recognize at the time.

In addition to the integer-only difficulty, there is no convenient way to accurately set the sample rate of the BASIC Stamp since there is no access to timer control or interrupts on the BS2pe. I had to essentially set up my code, then insert some programming feedback indicator loops (e.g., turn on an LED via my code, run a hundred samples, then turn off the LED and use a stopwatch to see how long it took to run the 100 samples. Then I would divide the stopwatch time by the 100 samples in order to know what the sample rate was. Then I would go into the code and change the sample rate factor in my calculations – but I was always playing with the sample rate factor because any time I modified the code, the execution time changes and therefore the time to take 100 samples changes - tedious.

Even when I got the sample rate routine sort of figured out, I was still having little success in generating angles. The angles I got were very erratic and always greatly less than the actual observed orientation of my setup. If I changed one of the formula factors by an order of magnitude I could almost get there, but it was very inconsistent in nature. It was very frustrating. Though I thought I was doing things correctly, my calculated angles always seemed to be about a third of the actual observed angle. I went over things in my setup and code a hundred times and did more research to confirm that my integration formula was right. I concluded that the "looseness" of the integer-only, non-interrupt processing of the BS2pe was the culprit. So, I searched the Parallax site for an alternative.

Floating Point Unit - FPU

I came across the microMega Floating Point Unit, or uMFPU. Cam Thompson has produced a great chip and created a great web site at microMega. At Cam's site you can find support for using his uMFPU with a variety of microcontrollers. With the uMFPU, I felt would be home free since I would be able to implement floating point math, as well as precise sample rates.

The uMFPU contains, in addition to its built in floating point support, a couple of channels of 12-bit ADC, so it was a relatively simple process to attach my IDG gyro to one of the ADC inputs on the FPU, and then connect the FPU to the BS2pe. Among others, one of the built-in functions of the FPU was the ability to program/select specific sample rates for the ADC reads.

Serial Interfaces

The are two fairly common methods to interface peripheral chips with a microcontroller, I^2C and SPI. Implementing one or the other is not too difficult, but doing so with the uMFPU was facilitated by BASIC Stamp-specific example code and demo programs provided by Cam on the web site. Since I had some SPI experience interfacing my earlier 12-bit ADC into the BS2pe, along with Cam's excellent documentation, I had the IDG1215 talking to the FPU and the FPU talking to the BS2pe in short order. I figured I was getting much closer and good angles would be had in quick order!

Programming the FPU however was another learning curve and took a bit of work and study – just another in the series of bumps I was going through. In order to configure or program devices such as a microcontroller or FPU, the manufacturer will often create what is called an IDE – Interactive Development Environment. The IDE provides support for your programming by including access to various libraries of functions, utilities, compilers and the like to help keep you out of trouble and assist in developing your application. I had used an IDE from Parallax when I was programming the BS2pe, and now I had another one to learn for the microMega FPU.

The IDE for the uMFPU is very complete in this regard, even going so far as to allow you to describe what you want to do in simple written terms and then have it compile your basic statements into the machine language that the FPU requires to run. In addition, Cam was always ready to lend a hand to help guide me through the process. I doubt he has the time to spend with very many people at the level he was assisting me, but he was intrigued by my application with the rocket tilt meter and went out of his way to be helpful. The FPU is a beautiful device and fun to work with, though a bit daunting for a neophyte such as me.

microMega Floating Point Unit

However, with some hard work and Cam's help, I was able to tackle the FPU and get my code running. However, I still was not able to get good angles! Oh, the frustration. I kept checking my formula to be sure I was not off by a decimal point or two in some of my factors. I changed out the sensors. I changed the microcontroller. I had Cam check all my code. Everything seemed to be OK, but I could not get any reliable data – again, the angles always seemed to indicate 30-40% of the actual observed tilt.

William Premerlani

As I was exploring all the various web sites during my research, I came across so many resources, microcontrollers, sensors, IMUs, autopilots and the like. My favorite sites were SparkFun and DIY Drones – SparkFun seemed to have all the goodies I might need and DIY Drones had a community of open source programmers and RC pilots that provided a wealth of information related to what I was exploring.

In going through some of the blogs on the DIY Drones site, I kept seeing references to the UAV Development Board (UDB) and a guy named William Premerlani. I found out that much of what DIY

Drones had developed for their autopilot was modeled after Bill's open-source UAV Development Board (UDB). Apparently, Bill was challenged by his son one day a few years ago to build an autopilot after they saw a UAV make a Transatlantic crossing. Though not expecting to cross the Atlantic, Bill took on the challenge and began to develop his own autopilot board and software.

William Premerlani's UAV Development Board

I contacted Bill one day and told him of my project – he showed immediate interest and offered to help. After a few discussions with Bill to sort through what I had been doing, we realized that one of the pins on the gyro I was using needed to be grounded. Once corrected, I fired up my project and instantly had a working, gyro-based tilt meter giving me proper angles (January 2010)!

The UAV Development Board - UDB

Now that I fumbled my way and learned enough to actually do rotational rate integration and derive angles, I spent time discussing the more rocket-specific details of my project with Bill. I came to realize after really studying his web site and the related blogs, that with where I ultimately wanted to go, Bill's UDB already incorporated so much of what I needed to get there.

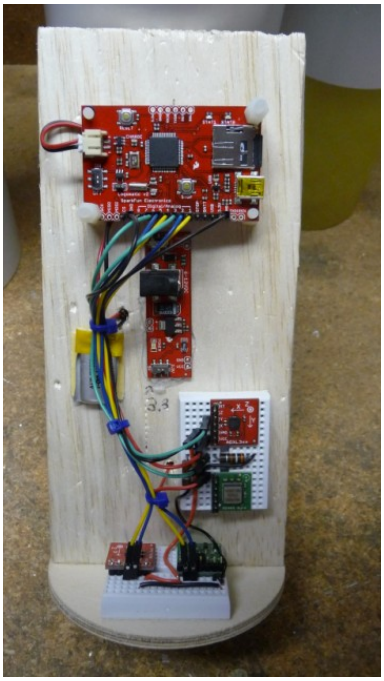
So, after discussing it with him for awhile, I ordered a UDB and started to study the code on the UDB web site in earnest, ordering a couple of books on C along the way.

I did not expect to actually use the UDB for my tiltmeter because I wanted to use different gyros that it used. I figured I would experiment with the UDB, and then adapt its code and my gyros to a similar PIC-based development board. Bill had used a PIC microcontroller as the heart of the UDB.

Initial Gyro Flight Testing

While waiting for my UDB to show up, I wanted to determine if the Invensense IDG1215 dual-axis gyro sensor was actually the right gyro to use. I needed to know if it was capable of performing while under

the stress of the g load experienced during a high-powered rocket flight. The datasheet for the sensor did not specify what effect linear acceleration had on its performance. My alternative, the single axis, more expensive Analog Devices (AD) ADXRS614, included such a specification that indicated it should not be subject to a significant error due to linear acceleration. Additionally, the AD gyro was what is called a Y-axis or orthogonal package meaning it needed to stand vertically on the board to sense X and Y, whereas in addition to including dual sensors in one package, the IDG1215 was designed to lay flat on the motherboard, possibly reducing other effects of high g forces, vibration, etc.



Gyro Test Fixture w/Logomatic Data Logger

I designed a little fixture that I could fly in a “mule” rocket that would simultaneously record the output of the two different gyros during a test flight. I used another device from SparkFun, a Logomatic SD card-based data logger. With the Logomatic, I could connect both of the gyros simultaneously through two of its eight ADC channels, then download the data after the flight and compare the outputs.

I flew the gyros a couple of times (March 2010). The results indicated to me and Bill that there was not a substantial difference between the performance of the two designs, so I thought at that time I could probably get away with using the less expensive, flat dual-axes Invensense gyros.

I did note one interesting thing about the gyro output that I had not considered earlier in my tilt meter project. It set me back quite a bit once I realized what I was looking at. I was about to finally get a true hint about the magnitude of the project I had taken on. Another learning curve here we come!

Frame of Reference Transformation

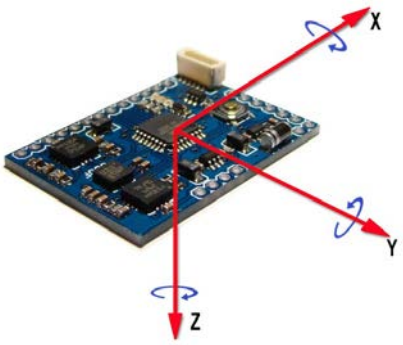
I took the data of the Logomatic and inserted it into an Excel spreadsheet. The data I had was simply the raw output of the gyro sensors. One nice thing about the Logomatic is that I was able to precisely define the sample rate that the Logomatic used while collecting the data. This made it easy to apply my integration formulae in Excel. I was easily able to apply the formula for the specific sensor and calculate the angles the rocket flew during the test.

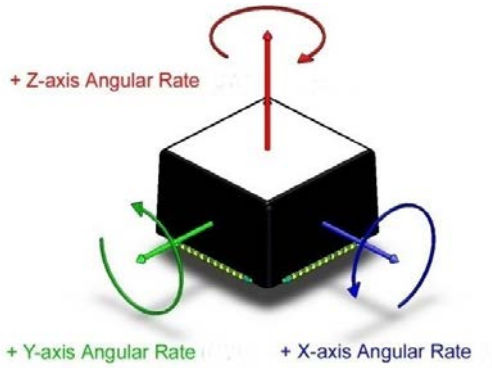
The motor I used in the test flights was a bit underpowered so the rocket began to arc over almost immediately after launch – I worried about that at first but then thought maybe it was better since I would possibly get more relevant data that if it flew perfectly straight up to apogee. Since I was able to calculate the rate integration and determine the flight path angle (or so I thought at the time), I was quite proud of myself and very happy to have actually made what I thought was significant progress after such a long journey to that point.

I calculated the angles of the two different gyros and they looked very similar as I scanned down the column in Excel, convincing me that the performance of the IDG gyro was nearly the same as the AD gyro. I then plotted the data out into a graph for better comparison. The plots looked very similar, further convincing me of their like performance.

However, something I had not considered became apparent as I studied the curves. Though I knew that the rocket flight path had a continuous arc from my observation, the angles I had plotted grew as expected, but then diminished further along the flight up to apogee. Hmmm, that was odd. It was also not consistent with my observation that the rocket continued to arc over at a fairly steady state all the way up. What did that mean? Then it dawned on me. The rocket had rolled during it ascent to apogee and this threw the angle calculation off.

I slowly realized that the rocket, as it lifted off and began its ascent, began to arc over to the right, which in this case due to how I had it mounted caused the output voltage of the X-axis gyro to decrease which, when integrated, would indicate an increasing negative angle. So far, so good. However, as it got higher in its flight, the rocket rolled around the Z-axis such that the X-axis gyro now began to output an increasing voltage even though the rocket's flight path continued to arc over to the right. If the rocket had not rolled, the indicated negative angle would have continued to increase, but since it rolled, the positive output when integrated began to cancel out the accumulated negative angle! The net effect was that the indicate angle was less than the actual incurred angle.





3-Axes Gyro/IMU

I thought about this some and figured I could simply incorporate a third gyro axis to track the Z-axis and include the rocket's roll about the Z-axis into my angle integration formula. That seemed easy enough to do. I had hoped I was at the end of most of the hard technical aspects of the problem. But, I was slowly realizing how subtle all this was and that I just was not able to predict where things might lead. As my physicist friend shortly thereafter advised me and I found to be true, "You've just made the first baby steps towards the transformation of two coordinate systems, the earth's and the rocket's…"

At the time though, thinking about it some more, I started feeling that maybe it would not be so bad after all. If the rocket took off and arced over to the right 10 degrees, then instantaneously rolled about the Z-axis by 180 and continued to arc over another 10 degrees the actual angle in the earth's frame of reference would be 20 degrees. The integrated angle of the gyro's output would be 0 however since the roll would cancel out to cumulative angle. But, if I knew that the rocket had rolled 180 degrees, I could subtract the second part of the integration rather than add it and the result would be consistent with the actual tilt. Hmm, that did not seem too bad. If it rolled only 90 degrees, but I knew that, I could figure out how much to add and how much to subtract, or I could in that case use the output from the Y-axis gyro that would effectively have replaced the X-axis gyro with respect to the earths orientation or frame of reference. It seemed a bit complicated, but not insurmountable.

Linear Problem?

So, after realizing the need to most likely have to incorporate the Z-axis in any attitude calculations, I did more research. My thoughts going in at the time were that my solution was a straightforward, linear calculation. This was supported, or so I thought, in some of the additional research I carried out. Some of what I read seemed to indicate that if there is enough processing power, that it is a linear problem. However, from a practical point, transformations such as I needed to do cannot be done with microcontrollers and linear math. It requires using estimates or approximations of position rather than actual coordinates - this pushes it for all intents and purposes into a short cut, non-linear math solution.

That is, it is ultimately easier to throw some fancy math at the problem rather than grind through all the calculations needed. I learned that most such attitude coordinate transformation solutions rely on some

form of a Kalman filter. Such a filter is a general mathematical construct that evaluates information from various sources and predicts and updates the evaluation in order to calculate an estimate of the actual solution.

I was anxious to throw this line of modified integration calculations thinking over to Bill and see what he thought. I wanted to see how tough this new direction I found myself headed in was going to be. I had seen references to Kalman filters on the DIY Drones web site and figured there was some code out there that I would be able to adapt, so, I was still not completely discouraged. I also figured that since my tilt project was really simpler than what I was reading about, I still thought that maybe my situation was a linear, less complicated problem.

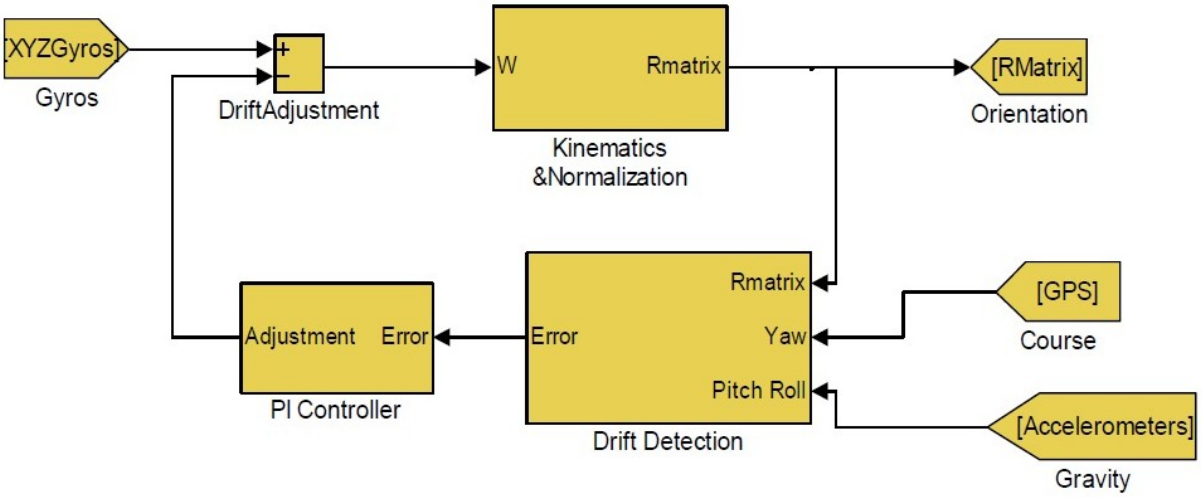
I sent Bill an email, describing my view and asked if it was indeed a relatively straightforward linear problem as I perceived, or if it were subtler than that. Disappointingly to me because of the implications to my project, Bill answered that it was indeed subtler. I should have known right?

Fortunately though, Bill also said that he had something he thought would help. Thinking back now, he was probably grinning a bit as he wrote that, having been through these type issues ages ago.

Direction Cosine Matrix - DCM

The Kalman filter is probably the most common approach to use for coordinate transformations. Another approach to the problem is the rotation matrix, or direction cosine matrix (DCM). The heart of the UDB's attitude orientation system is the DCM, which Bill developed and adapted a few years ago with help from Paul Bizard (interestingly, a similar approach was being independently developed by Robert Mahoney about the same time – see reference section).

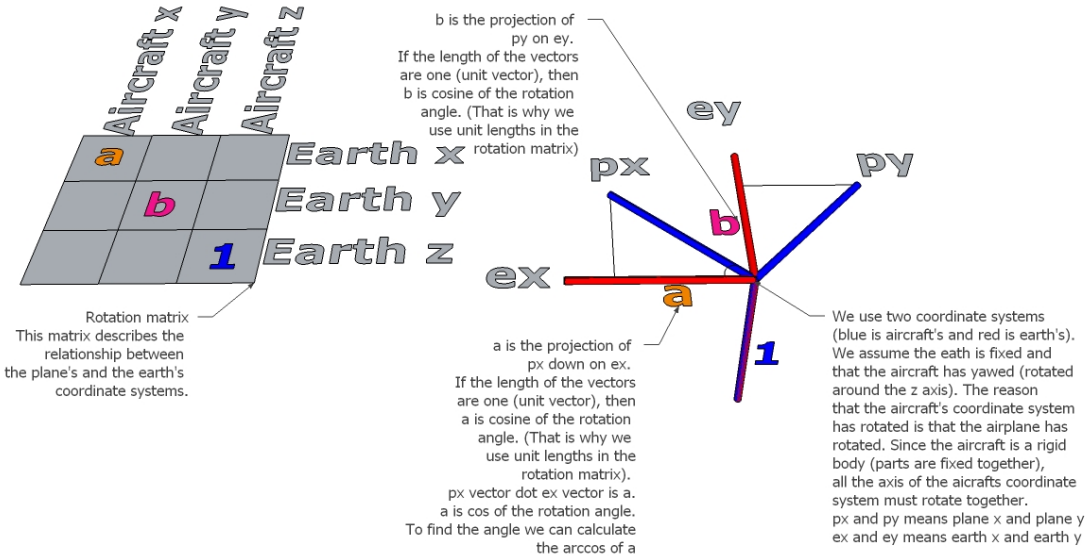
The DCM is another mathematical construct to keep track of attitude and orientation, but rather than being a general approach like the Kalman filter, it is designed specifically for tracking rotation. Because it is specific to the task, it is very effective and efficient. Therefore, the computations are fewer and easier for the microcontroller to process. There is extensive background material included in the reference section at the end of the article on Kalman filters and the DCM. Another advantage is that the DCM is not subject to some "singularity" solutions that cause some implementations of the Kalman filter to be somewhat troublesome.

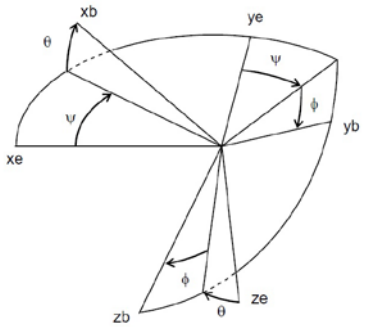


DCM Block Diagram

In his reply to me, Bill offered that obtaining the X and Y angles I wanted from the existing calculations being done by the DCM-based UDB firmware was straightforward and that he could help me easily derive them. I worked with Bill and revised the UDB firmware accordingly. Soon I had a working gyro-based tilt meter!! In April of 2010, after nine months of work, I had the solution.

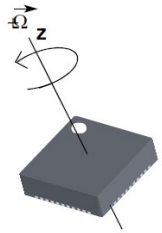
After working with me to adapt the firmware, Bill offered another insight. He made me realize that by monitoring X and Y for plus and minus tilt (i.e., right-left, front-back), I would actually be creating a "boxy" envelope rather than a true uniform tilt meter. What I needed was something that monitored the tilt orientation off the Z-axis, rather than monitoring X and Y. If I could do that, I would indeed have a true tilt meter with a cone-shaped Z-axis envelope instead of the boxy-shaped X and Y envelope. A few more emails, a bit more programing, and I had a working, gyro-based, true cone-shaped tilt monitor – the RockeTiltometer™!





<u>The Coast Optimization System - COS</u>

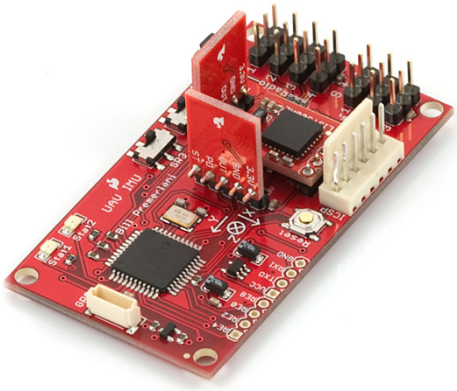
So, now that I had a sensor to monitor tilt, how to I put it into action?



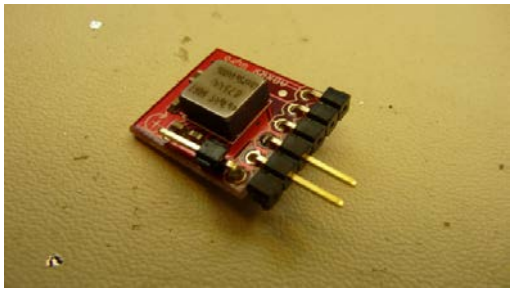
Single Axis Yaw Gyro

I had been using a standard UDB to develop my variation of the firmware. I had decided finally to go with the Analog Devices gyros for my project. They were more expensive that the Invensense ones, but

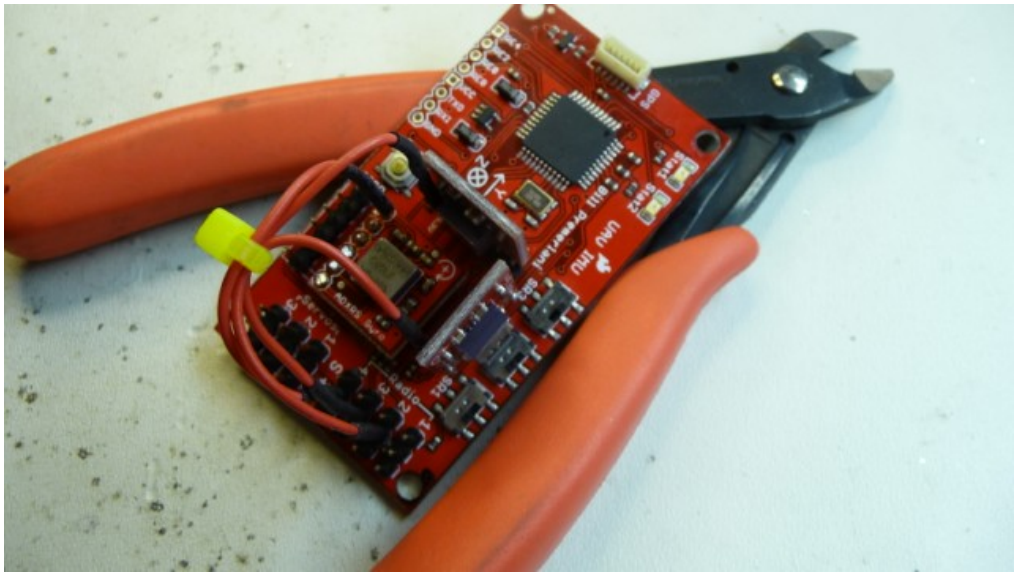
Bill and I both thought they would ultimately make a better choice, so, I planned to replace the standard gyro breakout boards on the main UDB board with the AD gyros. Fortunately, SparkFun makes a breakout board for the ADXRS614 gyro. The pinout is a bit different than that of the ST Electronic gyros that are standard for the UDB, but since they are both single-axis yaw rate gyros, they can be made to fit and work just fine. I was planning on unsoldering the three standard gyro breakout boards and adapting three ADXRS614 breakout boards to replace them.



Standard UDB

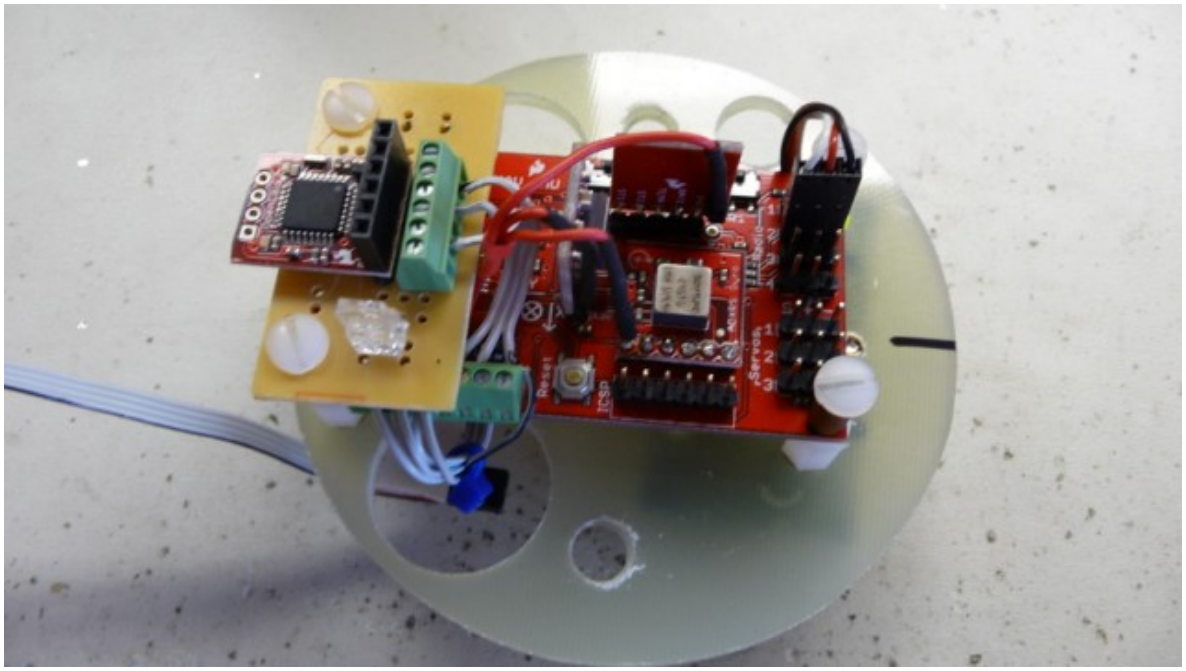


ADXRS614 Gyro w/ Modified Header



Modified UDB w/ 3 each ADXRS614 Gyros

As it turned out, SparkFun also makes the UDB boards for Bill Premerlani. It just so happened when I was working with Bill on my project, that he was in the middle of doing some research with some other new gyros for the next iteration of the UDB, so SparkFun had sent him a batch of boards without the standard gyros mounted. Bill sent me one of the gyro-less boards. With a bit of guidance from Bill, I was able to modify the breakout boards and the UDB in order to supply them with a 5 volt feed rather than the designed 3.3 volt feed. With a modified header, they fit just fine and I was shortly in business with an ADXRS614 sensored UDB! I added a uLog data logger from SparkFun and mounted the components onto a fiberglass coupler bulkhead plate so the system could fit into my sustainer avionics bay.



Analog Devices-gyro-based UDB with uLog™ Data Logger

What I did then was to insert the modified UDB in between two separate flight computers, a Featherweight Altimeters' Raven and a G-Wiz Partners' HCX. The Raven was already acting as the secondary dual-deployment flight computer in the sustainer, with the HCX as the primary. I then used the Featherweight Interface Program (FIP) to configure the Raven's remaining two pyro channels to send signals to the UDB gyro-computer - one of the Raven channels signals launch-detect and the other signals its verticality-check.

Coast Optimization System in Fabrication

- Raven and HCX Flight Computers with Gyro Computer –

I thought I would have to do some voltage conversion on the output of the Raven pyro channels to properly interface with the spare digital input/output channels on the UDB. However, I quickly realized that when the Raven pyro channel fires, its FET is taken to ground, so it was a simple matter to common the grounds on the Raven and the UDB and configure the UDB inputs as active-low. Then, whenever the Raven channel fired, it simply brought the UDB input low and signaled the Raven event to the UDB.

The UDB receives the Raven signals and uses them in the following ways:

1) Launch-detect: when the UDB is powered up, it actually utilizes 6 degrees of freedom – X, Y and Z accelerometers and the X, Y, and Z gyros…the accelerometers are used to measure the earth's frame of reference and correct for any gyro drift…while powered

up on the pad awaiting launch, the accelerometers continually keep the UDB's orientation properly aligned and give the gyros their frame of reference.

For reasons covered earlier, at launch the accelerometers lose their value as orientation references since the earth's gravity at 1 g is so overwhelmed by the rocket's 5 – 30 g acceleration during motor burn.

The Raven detects launch by invoking an algorithm that signals whenever it senses acceleration greater than 3 g that integrates into a velocity that exceeds 3 MPH (this allows the rocket to bounce around a bit on the launch pad from gusts of wind.

When the UDB receives the Ravens launch signal, it effectively turns off the influence of the accelerometers in the DCM calculations and the gyros become the sole source of orientation reference. The gyros will begin to drift, but the drift during the relatively short period of interest (while the rocket moves from the launch pad to a point of motor ignition, say 10 to 30 seconds) is not significant enough to materially affect the DCM calculations.

2) Verticality-check: the Raven's verticality-check trigger is constantly monitored by the UDB through another input channel and compared to the UDB's ongoing angle calculation – if the critical angle off the Z-axis (the value is pre-programmed), e.g., 15 degrees, has not been exceeded when the UDB sees the verticality check signal from the Raven, the UDB will signal the HCX via its user-programmable input channel and fire the sustainer motor ignition.

   If the Raven verticality signal is received and the angle has been exceeded, ignition is inhibited, i.e., sustainer motor ignition is aborted.

The logic contained in these steps allows that if the rocket's angle has gotten too far past vertical, we would rather not ignite the sustainer motor and have the rocket just return for another attempt. The purpose of the system is to maximize the coast period in an effort to maximize altitude. Therefore, rather than waste an expensive motor that is simply going to go "sideways", we would rather save it, have the rocket deploy back to earth and make another attempt after whatever corrections are needed to the rocket configuration.

As a tremendous side-benefit not initially on my radar at the start of my journey, is that in the event of an adverse attitude flight where the tilt angle is significantly off-axis and poses a safety threat, ignition will be inhibited by the system!

Test Flights

I flew three test flights of the RockeTiltometer^{TM}/COS (May/June 2010) including at LDRS 2010 at Lucerne Valley, and the results were all that could be expected. All of the circuitry worked well and the firmware/software performed properly, including the ability to easily change the critical tilt envelope degree window.

There is no specification for the ADXRS gyro in regards to the maximum g load under which it will continue to operate. The specification includes a linear acceleration error and a maximum-g shock number, but nothing that indicates a maximum operating limit. In anticipation of utilizing the COS on a flight at BALLS 19 at Black Rock, Nevada, in the fall of 2010 where the number of g that the rocket would see exceeds 22, I wanted to be sure it would continue to operate well under those conditions. One of the test flights was flown as a single stage with a J1520 VMax motor to give it a pretty good ride. The gyros saw a g factor during that flight of over 21 g and all the data readings indicated that it performed perfectly.

Real Flights!

I flew that planned flight at BALLS 19 in September of 2010. The rocket I flew was named Hermes 8, Mark III, and it housed the prototype RockeTiltometer and the Coast Optimization System (COS). The booster contained a Cesaroni N5800 C-Star and the sustainer a Cesaroni M2020 I-Max. My RockSim simulation predicted a 50K'+ flight.

Liftoff was great, but shortly afterwards the rocket started arcing over, finally breaking up at about 2,000'.

Though the flight was an obvious disappointment, the COS wound up saving my sustainer motor to fly another day. Since one of the parameters I had programmed into the Raven flight computer as a condition of ignition never came “true”, ignition was aborted. I had included a test that the rocket was above 16K’. Since it never achieved that altitude, the Raven never sent an ignition signal to the RockeTiltometer. In addition to the expense of the motor reload kit (~$370 retail) that was saved, the end result was much safer than it could have been if the motor had been tied to a straight timer for ignition. With the adverse event happening so close to the ground and the wild nature of the airframe sections at that point, motor ignition could have been a serious threat to the spectators.

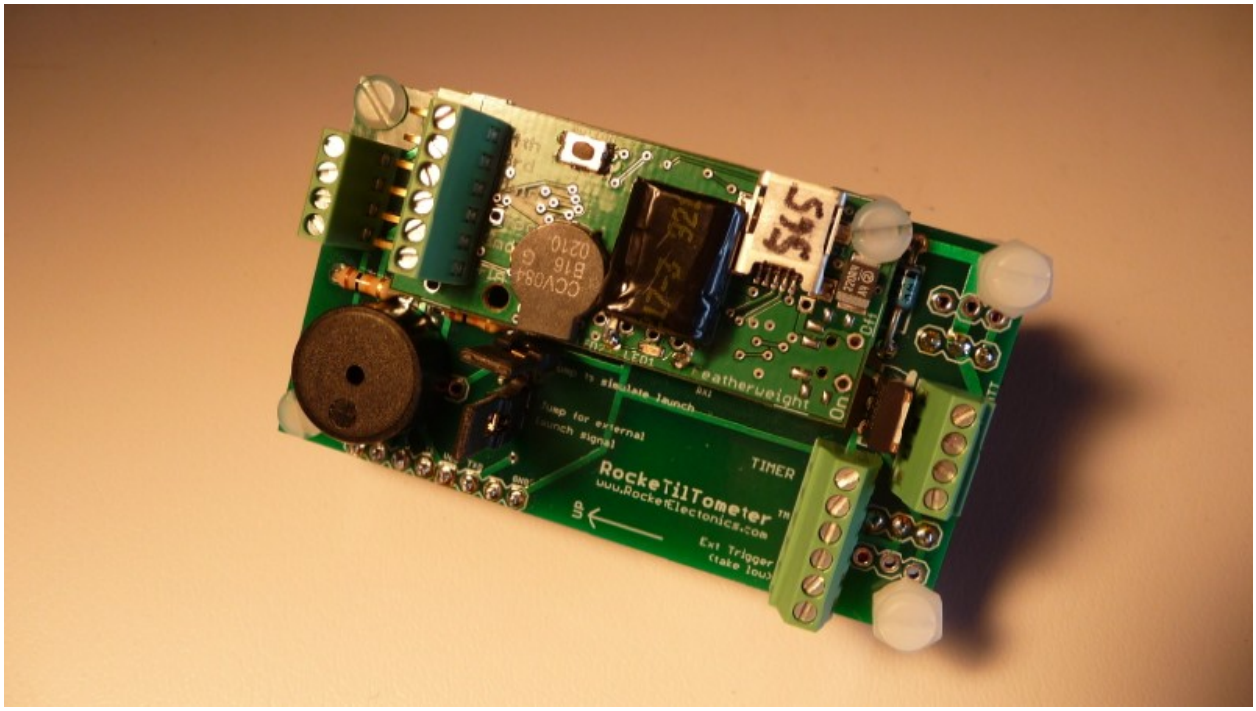
On a better note, I altered the motor configuration to allow for the lesser waiver height allowed at Plaster City, CA for the annual Plaster Blaster event held in November of 2010.With an upper limit of 25K’, I used the saved BALLS M2020 sustainer motor in the booster and put an L995 in the sustainer, with an expectation of exceeding 20K’ if all went according to plan. Well, it did!

Hermes 8, Mark II, reached an altitude of 22,299' with a coast time between booster burn out and sustainer ignition of about 11 seconds. When the Raven saw all the other pre-programmed conditions "true", as it decelerated through 400 fps, it sent an ignition signal to the RockeTiltometer. Since the tilt was well within (9 degrees) the 20 degree envelope for which I had configured, when it received the ignition signal from the Raven, it in turn sent an ignition trigger signal to the G-Wiz HCX which then fired the ignition pyro channel and the L995 ignited. A great flight.

So, two different experiences, but in both cases the COS worked to perfection - very satisfying results and a nice payoff for the months of effort.

Further Development

I have developed the next generation of the RockeTiltometer and the COS by fabricating a new printed circuit board which sits atop the UDB. The board has an interface for the Raven flight computer and the data logger, as well as a separate igniter FET circuit. Interestingly, the latest UDB has moved over to the Invensense gyros and as expected after our earlier tests, they work very well for our application.



RockeTiltometer™ and Raven™ comprising one version of an integrated Coast optimization System (COS)

Summary

So there you have it – a many-month long journey of exploration, learning, mistakes, frustration and ultimate joy. It was a great journey that had surprises at almost every turn.

Future Applications

Abort or Ignite? Along the way, I considered other information that might be helpful coming from the UDB. For a long time after deciding to pursue a tilt meter, I debated whether it should be used as a trigger mechanism for sustainer motor ignition, or as an abort mechanism, or possibly both. It seemed to me that the rate at which the angle was accumulating could be used to determine if I should trigger ignition or abort ignition. The faster the rate of increasing angle change as the rocket approached the edge of the angle envelope I had programmed into the UDB, the more likely it was that I should use it to abort the ignition. If the increase in angle was slight, and the Raven verticality check had not yet triggered, I thought that I could use the edge of the angle envelope as an ignition trigger. I discussed this with Bill and he was able to show me how the DCM could be easily used to monitor the rate of angle change. Once I have more experience with the basic tilt-ignition process, I likely will incorporate such monitoring.

Vertical Steering System I also early on recognized the desirability of using the UDB for vertical steering. I envision further adapting the UDB's stabilization mode (it incorporates stabilization as well as navigation for its UAV function) to control servos that will be attached to what I call *canarderons*.

Elevons on an aircraft combine both the function of elevators and ailerons. In a similar fashion I imagine two forward control surfaces that appear to be canards, but are able to articulate like elevons do.

[I noticed a while back a reference to a research paper by young Brian Guzek entitled *Active Stabilization Flight Computer*, wherein he proposes a vertical steering system, but suggests using articulated nozzles for the steering mechanism.]

Stabilization Roll and Aerobatics The UDB already has servo outputs to work the control surfaces of an unattended remote-control airplane. The firmware of the UDB needs to be modified only slightly to allow the elevon logic to apply to the canarderons. The trick will be to have a very high resolution of control (fine, precise mechanical advantage) over the canarderon surfaces due to the relatively extreme speed of the rocket compared to an airplane. A further extension of this principle would be to have the canarderons slowly roll the rocket around the Z-axis in order to help stabilize its flight path. Rocket aerobatics are even possible, such as increasing and decreasing barrel rolls as the rocket climbs into the sky!

Recovery Steering Another application that suits an adaptation of the UDB is as a recovery guidance system. You have access to servo controllers. There is also an input for a GPS receiver and firmware available for the UDB GPS reads (normally used for navigation assist). How about taking a GPS reading at the launch pad; then when the rocket deploys for recovery, deploying a parafoil type chute that has a servo/pulley steering system coupled into the GPS signals. The pulleys tug on the steering lines of the parafoil and bring the payload back to the launch pad GPS coordinates – all the basic GPS and navigation software already exists in the UDB firmware – all you really would need to do is figure out the servo/pulleys mechanics!

References

"A Guide To Using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications", Starlino, http://starlino.com

J. Geen, D. Krakauer, "New iMEMs® Angular-Rate-Sensing Gyroscope", ADI Micromachined Products Division, Analog Devices www.analog.com

W. Premerlani, P. Bizard, "Direction Cosine Matrix IMU: Theory", May, 2009

R. Mahoney, T. Hamel, J. Pflimlin "Nonlinear Complementary Filters on the Special Orthogonal Group", IEEE Transactions On Automatic Control, Vol. 53, No. 5, June 2008

Web Sites

http://www.gwiz-partners.com

http://www.featherweightaltimeters.com/

http://diydrones.com/

http://diydrones.com/page/uav-devboard

http://www.sparkfun.com/commerce/categories.php

http://www.modernhpr.com/

http://www.apogeerockets.com/

http://stores.whatsuphobby.com/StoreFront.bok

http://www.blackmagicmissileworks.com/

http://www.riekerinc.com/E-Inclinometers/SlopeAlert.htm

http://www.parallax.com/

http://www.rocstock.org/

http://www.tripolisandiego.org/

http://www.transolve.com/Transolve/index.html

http://www.arduino.cc/

http://www.vectornav.com/

http://www.analog.com/en/index.html

http://micromegacorp.com/

http://www.invensense.com/