Author: Robert Giuliano, KB8RCO
Address:
Saline, MI., 48
e-mail: KB8RCO@yahoo.com

Title: A Novel Method to Track High-Altitude Balloons Using Dire Wolf

In recent years, student organizations at the University of Michigan have approached the W8UM Amateur Radio Club with general amateur radio-related questions. Questions have included interpretation of FCC Rules, transmitter power limits, VHF/UHF transmitting distances, antenna design and band plan interpretation.

Two often asked questions, however, were specific to APRS beaconing procedures and finding a balloon transmitter. We guessed this was simply to improve the process of tracking and finding "their" high-altitude balloons. This generated some discussions among club members. (Photo 1)

John (KB8OIS) came to me (KB8RCO) and Michael (KE8AQW) asking if; 1)is it possible to track a high-altitude balloon, during flight, using the balloons transmitted position reports via RF only? and 2) could this data be used to control the Alfa-Spid satellite rotor controller to aim the satellite antennas during its flight path?

My response was - almost certainly! It "should be" as simple as monitoring the frequency with a radio, a TNC to decode the APRS packets, calculate the bearing angle (Azimuth) and 2D distance to the balloon, then use the altitude information to calculate the elevation. Seems simple enough. All I need to do is:

1. Get the required APRS parameters into the Alpha-Spid program to move the antenna to the correct Azimuth and Elevation.

2. Ensure we could decode the "from Callsign" and "SSID" to track the balloon.

3. Use the decoded APRS balloon information (Lat, Lon, & Alt) to calculate the bearing, Elevation and distance, in miles, from the ground station to the target balloon.

4. Finally, put it all together and test with actual APRS data.

Because of my lack of coding skills, each step was coded as a separate application and then later combined. I wrote the code to parse the payload while looking for any characters that indicated the text type messages. Following that character would be either a timestamp (7 characters) or the desired Lat and Lon data.

I now had a C++ application that logs in to APRS-IS, collects the text data from APRS-IS and parses out Lat, Lon, and Alt parameters, as well as the Callsign and SSID of the transmitting station. First step successfully completed!

Next, there were plenty of examples for determining bearing and distance between 2 GPS points. After several practice runs, with a manually entered location, I was able to obtain a compass bearing and elevation angle. Code was written with the calculations based on "received" APRS information while ground station GPS coordinates and elevation were hard coded at this point.

Now the Problems Begin

How **to** Interface with the Alfa-Spid Controller? I had the manual and the SPID Rot1Prog and Rot2Prog protocol documents (Alpha Radio, LTD.) for the controller. The Rot2Prog protocol was needed as this is used by the controller

to position both Azimuth and Elevation. The manual provided the basic information showing it runs at 600 baud, 8 data bits, no parity, and 1 stop bit. I just have to copy the format and enter my values.

What's this about no negative numbers? Oh, the Alpha-Spid offsets everything by 360 degrees (as zero). To go east from due north, you add an extra 360 degrees, but what about going west of north? For 270 degrees, which would be -90 degrees from due north, I can just send 270. That works! Since we calculate values as zero to 360, anything ≤180 degrees, add 360 and anything between 180 and 360 degrees, use the value as calculated. Remember, both Azimuth and Elevation use 360 as zero. Elevation is more like 360 to 450 (straight up).

Most of the Alpha-Spid commands are made up of readable characters, but there are two characters that will be nonprintable. Guess I will use Tera Term Pro's macro capability to send the raw values and see how it works. I next wrote a simple macro to send the antenna to its 'home position' (270 deg Az and 0 deg El - HEY, IT WORKS! Now try a different position, say 50 deg Az and 15 deg El (410 and 375 respectively). That also worked! Now just write the code to follow the examples.

Wait, what are these PH (Azimuth) and PV values (Elevation)?
Can I get better resolution of the controller by changing these? Let's give that a try. OOPS, the controller doesn't seem to accept those values. The antenna isn't STOPPING! It's trying to spin around like a top. KILL the power! That was an unpleasant adventure! Okay, rewrite everything with PV and PH set to 1. Macro's work!

Receiving Simultaneous APRS Data
To enhance balloon tracking and recovery, each balloon would now carry 2 transmitters; one would beacon every 2 minutes on the APRS frequency (144.390 MHz) while the second would beacon every 20 seconds on 144.350 MHz. The students felt adding a second transmitter would expediate their fox-hunting and balloon recovery.

MircoTrack Devices Set to MIC-E protocol
The students had formatted their data using MIC-E. At this point, I had only accounted for text-based data. MIC-E is a compressed format and uses some fields very differently (To Field). If we are going to obtain APRS data directly, we need to first interface with a radio and decode the MIC-E. This was particularly tricky since I was now searching the entire string for 5 "start characters" (=, !, @, /, and `).

How to Get Data from the Radio to My Application?
I have been a huge fan of Dire Wolf both as a software TNC and with its APRS-IS capabilities. So, if I write my application to interface to Dire Wolf, I can get everything I need from one source. I'll go with KISS over TCP/IP as the protocol since it would be the same data format no matter if I use TCP/IP or a physical TNC. But that will mean a computer.

I have this Pi3 with the Nexus operating system, Fe-Pi stereo input sound card and Nexus DR-X patch board. This would allow for the 2 radios and the internet to all be fed to my application through Dire Wolf in a nice small package! (Photo 2) I coded everything into one large C++ program; pulled to APRS-IS interface from the C++ code, downloaded Dire Wolf 1.7 from GitHub and compiled everything onto the Pi3. I took the audio **outputs** from both the satellite radio (ICOM 910H; VHF side) and the ICOM 746 (set to 2 meters), everything is working from the radio sides, but no APRS-IS data coming through Dire Wolf to my application.

I posted a comment on the Dire Wolf groups.io board asking about APRS-IS data through the TCP/IP KISS port. I received a direct response from John (WB2OSZ), the author of Dire Wolf. John said that data can go from IS to RF ports but there was no connection to the attached KISS streams. He said to give him a couple of weeks and he'd get back to me. A few weeks later, John said he was having more difficulty than expected but the received (IS to client) seemed to be working and communicating on KISS STREAM 15. I mentioned that was all I was looking for. John directed me to his Dire Wolf Dev Branch 1.7C for the update along with suggested improvements to streamline my parsing of the data stream.

I downloaded the updated Dire Wolf 1.7C onto the Pi3 and recompiled. Everything seems to be working and now receiving data from the 3 sources: 144.390 MHz, 144.350 MHz and APRS activity via the internet. We were ready!

One Unexpected Problem
We received notice the students were planning to launch not 1 BUT 5 balloons! Yep, semester is ending, so everyone must complete their projects. Could we track 5 balloons simultaneously?

Launch Day Arrives!
Finally, the 1st balloon went up and was at about 80,000 feet when the 2nd balloon was launched. The remaining 3 balloons followed in rapid succession. We were tracking and everything was working perfectly with the beacons being heard on both radios and the 144.350 MHz data was appearing on APRS-IS. By the time the last balloon took off, the first balloon had been recovered and back on campus.

Most balloons had reached heights greater than 90,000 feet while 1 balloon failed to ascend beyond 80,000 feet. All balloons were recovered. It was a REMARKABLY successful day. (Photo 3)

About one-week later, we met with the launch team for a debrief. They were quite pleased and incredibly surprised to see their secondary radio (144.350 MHz) broadcast onto aprs.fi. This alone greatly aided the students and supported the rapid recovery of their balloons. (Photo 4)

Now it's time to prepare for next semesters balloon launch(es).

Photo Legends:

Photo 1
L-R; Display showing balloon telemetry data on computer monitor, ICOM 746 transceiver monitoring 144.390 MHz, Raspberry Pi3 (partially shown), ICOM 910H transceiver monitoring 144.350 MHz, Roof Cam showing the satellite antennas and APRS display monitor.

Photo 2
Raspberry Pi3; CH-0 represents data from 144.390 MHz, CH-2 represents data from 144.350 MHz. Orange ethernet cable to APRS.IS.

Photo 3
Balloon telemetry data displaying CH 15 (current APRS activity), CH-0 and CH-2 (balloon telemetry via RF).

Photo 4
APRS.IS tracking map showing the 5 balloons