Optimally Calculating and Plotting Satellite Subpoints for AO-85 (Fox-1A) CPU Resets

Douglas D. Quagliana, KA2UPW/5
dquagliana@gmail.com

keywords: bash, scripts, predict, date, bc, HTML, Google maps, AO-85, Fox-1A, telemetry, CPU, resets, South Atlantic Anomaly

AO-85, Upsets and Resets; the Early Years[1]

Before it was even launched, during development of the Fox-1A satellite, Tony Monteiro, AA2TX (SK), predicted that the satellite could reset often, frequently over the South Atlantic Anomaly, and he encouraged the development of an onboard recovery mechanism to survive these resets. Based on our experiences with the ARISSat-1 satellite resetting on every orbit, we knew it would be every desirable to know when each reset occurred and how many resets had occurred. ARISSat-1 had no way of counting the resets or saving the time of the reset. After several days in orbit the ARISSat-1 battery would no longer take a charge, causing the satellite to power off each and every time it entered eclipse and power on/reset every time it exited eclipse.

No matter where you go, there you are[6]

For Fox-1a, we have a mechanism to determine the number of resets and the time of the reset. Thanks to the onboard software, the non-volatile MRAM, the telemetry downlink, a worldwide network of telemetry ground stations, the FoxTelem telemetry software, and the AMSAT volunteers, there is an up-to-date master list of all the resets and reset times available on the AMSAT website. Using the satellite reset time, which is the Unix time in milliseconds since 1/1/1970, and the NORAD TLEs for the AO-85 satellite, we can determine the satellite's latitude and longitude at the time it reset. It's just a set of calculations. The more accurately we know the time when the reset happened, and the younger the NORAD TLEs are, the more accurately we know the satellite's position.[2] Therefore, in order to create as accurate a map as possible, it is necessary to use those TLEs for the satellite that are closest to the desired prediction time. Using two year old TLEs will not give accurate results for the satellite subpoint location.[12]

The end result of all of the above work is that there is a complete list of all of the CPU resets and the epoch time of the reset online at

http://www.amsat.org/tlm/ops/FOX1T0.txt

and the format of each line is just the reset number, a comma, and the reset time. For example,

```
0,1444323370000
1,1444836255000
2,1445891544000
```

In order to determine where the satellite was at the time of the reset we need to find the set of TLEs closest to this date/time and then use "predict" to calculate the satellite subpoint. However the TLEs use a date format of YYDDD where YY is the year and DDD is the day of the year (up to 365, but sometimes up to 366).

Stand on giant's shoulders: Don't reinvent the wheel, or calendar calculations, or the date command

Assuming that we have a collection of TLEs, it is possible to convert the reset time to a date and then search through the collection of TLEs for the TLE that is closest to the reset time. The bash script "find_closest_tle.sh" does exactly this. This bash script accepts a timestamp (the reset time) and then searches through a list of TLEs to find the TLE with the time closest to the reset time. This gives the "freshest" TLEs *for this reset*. To determine the closest TLE, take the year, Julian day and fractional day for the TLE and convert this to a Unix timestamp (seconds since 1/1/1970). Performing date calculations can be non-trivial. Fortunately, we can use the Linux "date" and "bc" commands to perform most of the date calculations without all that tedious mucking about with leap years.[11] The conversion between Unix epoch and YYDDD format is done using the standard Linux "date" command with the "+%s" option. Since the date command has already been written and debugged, and since it already knows about things like leap years, we can leverage the efforts of its author and minimize our own efforts in coding and debugging which could introduce date computation errors. Once the dates is converted, we can compare the TLE timestamp to the desire timestamp and save the TLEs if they are the best seen so far.

Don't reinvent satellite orbital calculations either, use something that is known to work correctly

Since we really want to do this for each and every CPU reset, we can use another bash script (convert_T0_time_to_SSP.sh). This script gets the latest list of resets from the AMSAT website using the "wget" command and calls find_closest_tle.sh for each reset. The reset times and best TLEs are passed to the KD2BD predict program which calculates the satellite subpoint latitude and longitude for that time using those TLEs. After we perform the calculations for satellite subpoint for every reset and then we have a list of resets, latitudes, and longitudes. Now we can plot all of these points on a map using Google maps.

Map Making and the South Atlantic: Here Be Dragons (and Anomalies, and single event upsets, and plastic…lots of plastic)[8]

There's more than one way to make a webpage with a map using the Google APIs. This is just one of the ways, but it is pretty simple. There is some canned HTML "header" code that needs to be at the top of the webpage, some other HTML "trailer" that needs to be at the bottom, and the list of latitude and longitude points that are placed in between those two sections. When the three pieces are put together into an HTML file, it creates a webpage with a map. This makes it easy to generate a simple map from a bash script which will answer our question “Where are the CPU resets happening?” This method has the disadvantage that it can only be used to display about four hundred points on the map. Even though there have been over seven hundred resets, many of them are “duplicates” and do not need to be plotted.

This is the HTML “header” code:

```
<html><head>
  <script type="text/javascript" src="https://www.google.com/jsapi"></script>
  <script>
    google.load('visualization', '1', { 'packages': ['map'] });
    google.setOnLoadCallback(drawMap);
    function drawMap() {
      var data = google.visualization.arrayToDataTable([
        ['Lat', 'Long', 'Reset#'],
```

That’s all there is! Then we add our list of latitude/longitude points within square brackets and with a comment containing the reset number and latitude and longitude. The comment will be visible when the user hovers the mouse over the point on the map.

```
[ -56, -24, 'Reset#0 -56, -24' ],
[ 55, 158, 'Reset#1 55, 158' ],
[ 39, 51, 'Reset#2 39, 51' ],
```

… and so on until all of the points are listed. Lastly, this is the HTML “trailer” code:

```
]);
    var options = { showTip: true };
    var map = new
google.visualization.Map(document.getElementById('chart_div'));
    map.draw(data, options);
  };
  </script></head><body><div id="chart_div"></div></body></html>
```

These scripts were written just to “get the job done” with no attempts at optimization. After all, “premature optimization is the root of all evil (or at least most of it) in programming.”[10]

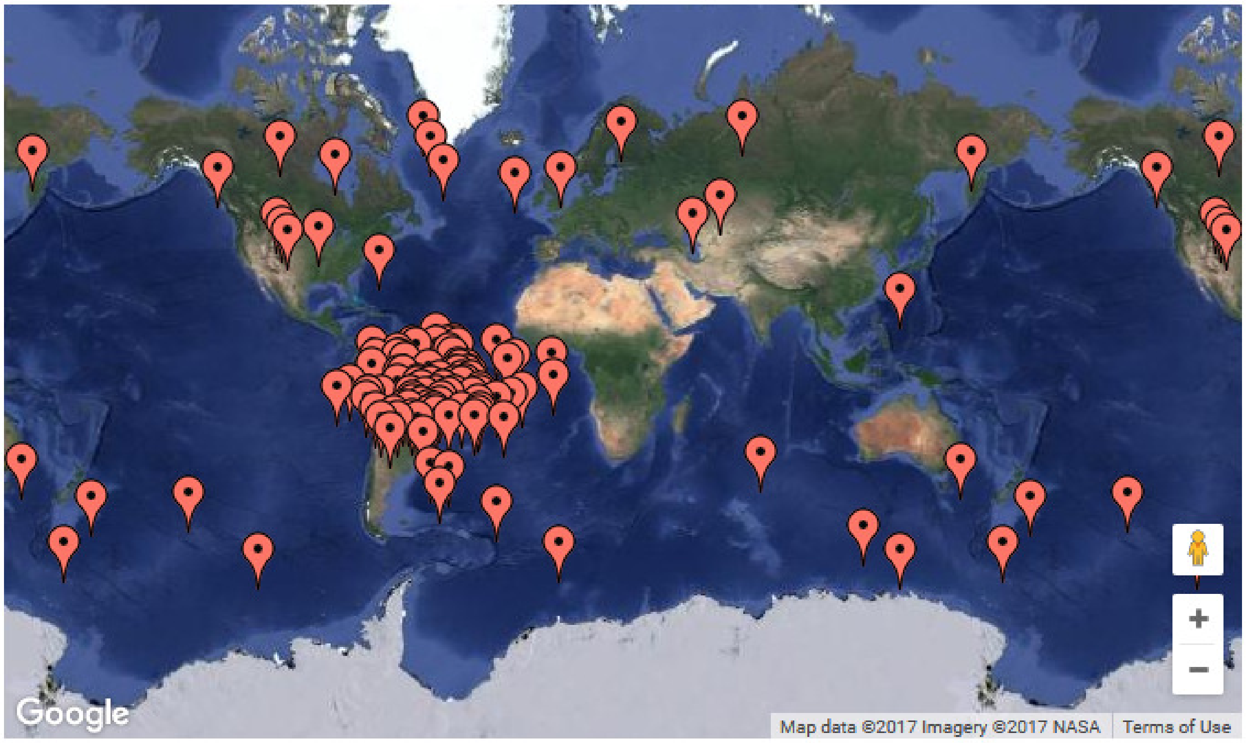
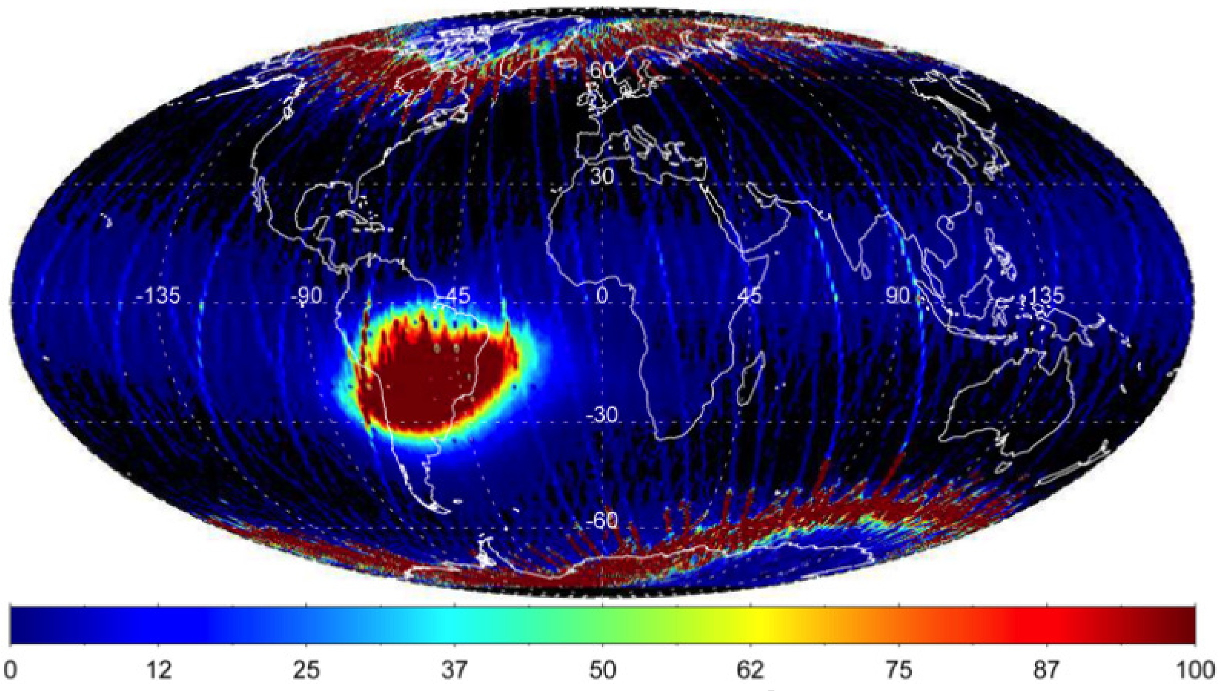


Figure 1 – The map of satellite subpoints for AO-85 CPU reset numbers 0 through 713. There is an interactive version of this map online at http://qsl.net/ka2upw/ao85_reset_map.html but eventually I would like to get the map and scripts onto the AMSAT website



The South Atlantic Anomaly[5]

Sticking pins in a map – well, what's the point?

A quick look at the map of the subpoints for CPU resets shows good agreement with the South Atlantic Anomaly (SAA).[4] It was expected that many of the resets, but not all, would be near or over the SAA and this is indeed what we find. A further analysis of the telemetry will let us look at the type of reset and match those up with the satellite subpoints. It will also be interesting to try and correlate the results of the radiation experiment, time of day, and solar activity with the satellite's resets. If you're interested in analyzing this, please contact the author.

Credits

"It is amazing what you can accomplish if you do not care who gets the credit."
--Harry S Truman

I would like to thank the following people who helped make this paper possible.

- Alan Biddle, WA4SCA for keeping track of CPU resets as they happen and sending emails to the FoxTLM mailing list
- Chris Thompson G0KLA/AC2CZ for FoxTelem and the AMSAT telemetry web pages
- Mark Hammond, N8MH for constructive feedback and support
- John A. Magliacane, KD2BD for writing predict and making it open source
- Philip A. Nelson for the "bc" command
- David MacKenzie, for the Linux "date" command and all its options
- Brian Fox and Chet Ramset for bash

And all of the ground stations that sent in telemetry, because without all of the collected telemetry none of this would be possible.

The author encourages and welcomes feedback and commentary, especially constructive criticisms and particularly welcomes deconstructive comments and puns, copies of your satellite recordings and telemetry, and perhaps any suggestions and remedies you might have for excessive "and" conjunctionitis.

Listing 1 - convert_t0_times_to_SSPs.sh

```
#!/bin/bash

# get the latest copy of all of the T0 reset times
# assumes that we have an internet connection
rm FOX1T0.txt 2> /dev/null
wget http://www.amsat.org/tlm/ops/FOX1T0.txt

previous_reset_time=0

# use "for p in $(<filename);" notation .
# If you use the "while read p;" version then the
# last line of FOX1T0.txt will not get processed.
#
for p in $(<FOX1T0.txt); do

   # file format is reset_number, reset_time -- for example
   # 0,1444323370000

   # get just the reset number
   reset_number=`echo "$p" | cut --delimiter=, -f1`

   # get just the reset time, dropping the trailing 000 (milliseconds)
   # leaving just the seconds since 1/1/1970 (Unix epoch time)
   reset_time=`echo "$p"   | cut --delimiter=, -f2 | cut --bytes=1-10`

   if [ "$previous_reset_time" != "$reset_time" ]; then
      previous_reset_time=$reset_time
      ./find_closest_tle.sh  $reset_time > reset_$reset_number.tle
      echo -n $reset_number, > reset_$reset_number.out
      predict -q /home/pi/predict/ka2upw.qth  -t reset_$reset_number.tle  -f AO-85
$reset_time $reset_time >> reset_$reset_number.out
   fi
   echo " Done"
done
```

Listing 2 - find_closest_tle.sh

```
#!/bin/bash
# find closest TLE (to a give date).
# Given a reference date from the user (on the command line), and a file of TLEs
# (all for the same satellite) search the file containing all the TLEs of this
# satellite and find the TLE with the date closest to the reference date.
# The TLE file is assumed to be triple lines of SATNAME, TLE1, TLE2 but not
# necessarily in date order. The TLE file is assumed to contain only TLEs for
# this satellite.
# Background: orbital prediction errors get larger when older TLEs are used.
# In order to get the most accurate predictions we need to use the TLEs that
# were generated closest to the time of the desired prediction.
# Since there could be gaps in the available TLE entries, we can't just take
# the most recent before or most recent after (consider the case where you
# want a  prediction for June 1 and you have TLEs for May 30 then a gap until
# December, or you want June 1 and have TLEs for January then nothing until
# June 4).  We want the closest date.
# Note: requires the "bc" package to do some of the calculations.
#

# Reset times from AO-85 are in milliseconds, so drop the
# trailing 000 to get seconds since 1/1/1970
time_to_match=$1
closest_matching_tle_delta=999999
smallest_delta_time=99999999

# N.B. date command's "j" is day of year 1..366
julian_date_to_match=`date -d @$1 +%j`

# read lines from AO-85.TLE - a single TLE is actually three lines long.
# The first line has the satellite name.  Lines 2 and 3 have the orbital elements.
# For example:
#
# AO-85
# 1 40967U 15058D   15344.32453720  .00001845  00000-0  20662-3 0 00659
# 2 40967 064.7773 101.4283 0217658 263.0133 094.6183 14.74493538009130
#
# 1234567891111111111222222222233333333334444444444555555555566666666666
#          0123456789012345678901234567890123456789012345678901234567890123456789
# See also:
# https://en.wikipedia.org/wiki/Two-line_element_set
#

# 86400 = 60 seconds per minute times 60 minutes per hour times 24 hours per day
seconds_per_day=86400

line_number_of_element_data=0
IFS=''
while read p; do
   line_number_of_element_data=${p:0:1} # 1 or 2

   if [ "$line_number_of_element_data" != "1" ]; then
      if [ "$line_number_of_element_data" != "2" ]; then
         tle1=$p
      fi
   fi
   # is this the first line of orbital elements?  (starts with a 1)
   if [ "$line_number_of_element_data" == "1" ]; then
      tle2=$p

      # satellite_number=`echo "$p" | cut --bytes=3-7`          #
      # classification=`echo "$p" | cut --bytes=8`
      # classification: a U here means unclassified.
      # Isn't it odd that all of the satellites are unclassified?

      epoch_year=${p:18:2}
      epoch_whole_days=${p:20:3}
      epoch_fractional_days=${p:23:9}
   fi

   # is this the second line of orbital elements?  (starts with a 2)
   if [ "$line_number_of_element_data" == "2" ]; then
      tle3=$p
      tle_epoch_as_unix_epoch=`date -d "20$epoch_year-01-01 +$epoch_whole_days days -1
day" "+%s"`

      delta_time=$(echo "($tle_epoch_as_unix_epoch + $epoch_fractional_days *
$seconds_per_day) - $time_to_match" | bc -l)

      # kludge to compute absolute value; sqrt(x*x) might be better?
      abs_delta_time=$(echo $delta_time | tr -d -)

      if (( $(echo "$smallest_delta_time > $abs_delta_time" |bc -l) )); then
         smallest_delta_time=$abs_delta_time
         smallest_tle1="$tle1"
         smallest_tle2="$tle2"
         smallest_tle3="$tle3"
      fi
   fi
done < AO-85.TLE
echo $smallest_tle1
echo $smallest_tle2
echo $smallest_tle3
```

Listing 3 - make_resets_map_from_out_files.sh

```
#!/bin/bash
# each of the *.out files is a single line output from "predict"
# prepended with the reset number. Each one looks like this line
# 619,1460541153 Wed 13Apr16 09:52:33  -28  158  245  -27   74   6941   1371 *
# reset,Unix epoch DOW Date Time(UTC)

# PREDICT produces an output consisting of the date/time in Unix format, the
# date and time in ASCII (UTC), the elevation of the satellite in degrees,
# the azimuth of the  satellite  in  degrees,  the orbital phase (modulo 256),
# the latitude (N) and longitude (W) of the satellite's sub-satellite point,
# the slant range to the satellite (in kilometers), the orbit number,
# and the spacecraft's  sunlight  visibility  information.

rm reset_positions.txt 2> /dev/null
# start building HTML table
echo "<table style="width:100%">"  >> reset_positions.txt
echo "<tr>"                        >> reset_positions.txt
echo "   <th>Reset #</th>"         >> reset_positions.txt
echo "   <th>Epoch</th>"           >> reset_positions.txt
echo "   <th>Date</th>"            >> reset_positions.txt
echo "</tr>"                       >> reset_positions.txt

cp map_header.html reset_map.html

for f in `ls *.out | sort -V`
do
   echo -n "Processing $f"
   SatResetNumber=`cat $f | tr -s ' ' | cut -d, -f 1`
   SatSSPLat=`cat $f | tr -s ' ' | cut -d ' ' -f 8`
   SatSSPWLon=`cat $f | tr -s ' ' | cut -d ' ' -f 9`  # west longitude

   # predict outputs west longitude values but google maps uses only
   # latitude values from -180..0..+180 so we need to convert
   SatSSPLon=`echo "define convert ( lon ) { if ( lon < 180.0 ) return ( lon * -1 )
else return ( 360 - lon ) } convert( $SatSSPWLon )" | bc -l`

   echo " Lat=$SatSSPLat WLon=$SatSSPWLon Lon=$SatSSPLon"
   echo "[ $SatSSPLat, $SatSSPLon, 'Reset#$SatResetNumber $SatSSPLat, $SatSSPLon' ],"
>> reset_map.html
   echo "<tr>"  >> reset_positions.txt
   echo "<td>"  >> reset_positions.txt
   cat $f       >> reset_positions.txt
   echo "</td>" >> reset_positions.txt
   echo "</tr>" >> reset_positions.txt
done
echo "</table>" >> reset_positions.txt

cat map_trailer.html >> reset_map.html
cat reset_positions.txt >> reset_map.html

# Append date this map was created so we can verify auto-updates
echo -n "Last update:" >> reset_map.html
date >> reset_map.html
```

Footnotes

1 This subtitle was suggested by Alan Biddle, WA4SCA.

2 Werner Heinsberg would have hated this.

3 Deleted

4 The South Atlantic Anomaly isn't actually centered over the southern portion of the Atlantic Ocean. In fact, the anomaly moves. For more details on the movement of the SAA and a discussion of some special comments on the literature of the SAA, see footnote 5.

5 Stassinopoulos, Epaminondas G., Xapsos, Michael A., Stauffer, Craig A. Forty-Year "Drift" and Change of the SAA. NASA Goddard Space Flight Center Technical Report NASA/TM-2015-217547, GSFC-E-DAA-TN28435. December 1, 2015. https://ntrs.nasa.gov/search.jsp?R=20160003393

6 Weller, Peter, actor. The Adventures of Buckaroo Banzai Across the 8th Dimension. Sherwood Productions. 1984.

7 I was motivated to not re-invent the wheel and use the Linux "date" command for all of the date calculation after reading some of James Miller's (G3RUH) papers including "30.6 Days Hath September" and "Sun's Up". For a collection of Miller's work see http://www.amsat.org/articles/g3ruh/bibliog.txt.

8 Actually, I'm told that there really aren't any dragons at all[9], and in fact, there are very few maps where the phrase "Here be dragons" actually appears despite there being many maps with pictures of dragons and lots of people being familiar with the phrase. I'm assured, however, that there certainly is a lot of plastic in the oceans.

9 Except in Komodo, where there really are dragons, but there are far enough away from the south Atlantic that we can ignore them for purposes of our calculations here. But I digress…

10 This quote is frequently attributed to Knuth, Donald. Computer Programming as an Art. 1974. However, it is also attributed by Knuth to Sir Tony Hoare. However, wikiquotes notes that the attribution to Hoare is doubtful. https://en.wikiquote.org/wiki/C._A._R._Hoare

11 Apologies to "other Doug".

12 Calculating precisely how inaccurate the two year old TLEs would be is left as an exercise to the reader. Hint: use KD2BD's predict program running under Linux.