XLF: Website, Monitor Plots, Beam Calibration

Levi Patterson

September 14, 2011

1 Introduction

The XLF generates local monitoring data that can be used to determine the beam energy and monitor the system operations. masterplot.sh runs a series of programs that creates plots under the address astroserve.mines.edu/xlf. The data is stored on the astroserve server under /srv/xlf. This document is divided into three sections; data used, website and plotting scripts, and energy calibration.
Data used outlines the vtmon, cldmon, autolog, and cal_const data files that the plotting programs read from. The website section explains the directory structure and the plots created. The third section is the plotting scripts section which goes into detail about program operations. Lastly, the energy calibration section explains the cal_const files produced and the plots that are created by the monthxlfcalibpoints.py. These files and numbers are useful for better understanding how much energy is being released into the sky.

2 Data

There are three sets of data files that the xlf creates called autolog, vtmon, and cldmon. These files are located under /srv/xlf/ in their respective directory. These files may appear binary so it is necessary to use the '–binary-files=text' command when grepping the files. An example of this is 'grep –binary-files=text STATSF'.
The only data used from the autologs files are located at /srv/xlf/autologs and can be seen by grepping the STATSF line from the files. A legend for these lines is:

```
 DATA:STATSF format
unused      %d
set_id      %d number of set
n_req       %d number of shots requested
n_good      %d  number of "good" shots (gps time, pick-off energy measurment>0)
e_mon_avg   %e  average monitor energy in Joules
e_mon_rms   %f  rms of e_mon_avg in percent
e_cal_avg   %e  average of calibration probe energy in Joules
e_cal_rms   %f  rms of e_cal_avg in percent
rat_avg     %e  average of e_cal/e_mon  (e_cal/e_mon) is calculated for each
                shot, so rat_avg is the average of n_good e_cal/e_mon ratios
```

```
rat_avg_rms %d  rms of rat_avg in percent
dpw     %d diode pulse width in microseconds of laser.. to set energy
                140 is max  90 is min
q     %d unused
azi     %f azimuth angle of beam (degrees CCW from North)
elv     %f elevation angle of beam  (90 is vertical)
temp_head   %d  temperature of laser head in deg C * 10  509 means 50.9
temp_dump   %d  temperature of laser dump resistor in deg C * 10
temp_plate  %d  temperature of laser interface plate in deg C * 10
volts_batt  %f  Voltage of battery bank 1
temp_equip  %f  Temperature on optical table
unused      %f  unused
year     %d Time that the shot was fired.  The time is that captured
month     %d     by the gpsy2 board on the xlf2 computer.  The actual pulse
day     %d     time-stamped is the Q-Switch output pulse from the laser
hour     %d     This pulse is generated at the time the light left the
minute      %d     laser to an accuracy of better than 10 ns.  The time that the
second     %d     light passes the elevation of the GPS antenna that is defined
nanosec.    %d     as the location of the XLF is ?? (10-20) ns later than the
                   nanosec. recorded in the log.
gps_sec.    %d second since Jan 6 1980, birth of gps
                Auger FD and SD record the gps_second of event
stage1     %f  positions of stages that hold beam calibration equipment
stage2     %f  (see menu_cal.msg for more information)
stage3     %f
stage4     %f
```

The data for the cldmon file is located under /srv/xlf/cldmon and can be seen by 'grep 00003 filename'. They are formatted as follows:

Table 1: Cldmon Format

Yr	MM	Dy	hr	mn	sc	code	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5	ADC6	ADC7
2011	06	15	21	01	31	00001	354.0	640.0	1483.0	1198.0	0.0	132.0	238.0	700.0
2011	06	15	21	01	31	00003	-35.550	-23.875	10.536	2.445	0.000	1.612	41.845	9.490

Table 2: Channel Descriptions

Channel	Name	Description
ADC0	CLF_uncp	uncompensated sky temperature (C)
ADC1	CLD_comp	compensated sky temp
ADC2	CLD_therm	temp of case of monitor
ADC3	CLF_vref	reference voltage (should be 2.5V)
ADC4	Rain	Rain Sensor
ADC5	Wind_Speed	Wind Speed (m/s?)
ADC6	Wind Dir	Wind direction
ADC7	Temp_Out	Outside Temp (C)

The data for the vtmon file is located under /srv/xlf/vtmon and can be seen by 'grep 00003 filename'. They are formatted as follows:

Table 3: vtmon Format

Yr	MM	Dy	hr	mn	sc	code	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5	ADC6	ADC7
2011	07	20	23	52	26	00003	12.856	0.000	12.843	0.000	22.895	21.066	17.965	19.972
2011	07	20	23	57	29	00003	12.877	0.000	12.859	0.000	24.016	21.170	17.418	20.359

Table 4: Channel Descriptions

Channel	Name	Description
ADC0	Batt1_V	voltage on bank 1 of batteries
ADC1	Solar1_V	voltage on solar panel bank 1
ADC2	Batt2_V	second bank of batteries
ADC3	Solar2_V	second bank of panels
ADC4	Ctrl_T	control room temperature
ADC5	L_Room_T	Laser room temp
ADC6	Water_T	temperature of probe submerged in water tank
ADC7	L_Box T	temperature of probe in middle of optical table

The data for the cal_const files are located under astroserve.mines.edu/xlf/cal_const/logs or /srv/www/html/xlf/cal_const/logs. The files follow the following format:

Table 5: cal_const Format

YY	MM	DD	Calibration Constant	Flag
2011	1	10	12.322	0110000
2011	1	11	12.132	0000000

This table shows what each digit means for the flag. More than one digit can be flagged per point. There are three calibration measurements done at the beginning of the night and three done at the end of the night. Which probe and at what energy the laser is firing at is described in Table 6.

Table 6: Channel Descriptions

Channel	Name	Description
0000001	Missing E2100	Early 100 dpw measurement from the 2nd calibration probe
0000010	Missing E3100	Early 100 dpw measurement from the 3rd calibration probe
0000100	Missing E3140	Early 140 dpw measurement from the 3rd calibration probe
0001000	Missing L2100	Late 100 dpw measurement from the 2nd calibration probe
0010000	Missing L3100	Late 100 dpw measurement from the 3rd calibration probe
0100000	Missing L3140	Late 140 dpw measurement from the 3rd calibration probe
1000000	Only Calibration	The stages were in calibration position for all shots

3 Website and Plotting Scripts

The website is located at astroserve.mines.edu. On this site each plot is created by its own independent plotting package. There is a master plotting script that is run every night through cron at '/srv/xlf/plots/scripts/masterplot.sh'. masterplot.sh creates the proper website structure, runs plotting programs, converts .eps files to .jpg files, concatenates files together, and moves files around. The programs that masterplot.sh runs are listed below.

1. xlfautolog.py
2. xlfcalib.py
3. xlfvtmon.py
4. xlfcldmon.py
5. xlfcalibpoints.py
6. dayxlfcalibfile.py

Autolog, calib, vtmon, cldmon, all have a different python script creating year, month, and day plots. That totals 12 separate scripts that create plots for year/month/day for the autolog, calibration, vtmon, and cldmon files. The xlfcalibpoints.py script uses files created from the dayxlfcalibfile.py to create two sets of plots. The xlf cal_const yearly plot and the calibpoints monthly plot. After all these programs have been run, the plots are converted to .jpg, then moved to their proper directories, the masterplot.sh then uses the mkgallery script to create viewable galleries form the web page.

These programs were written with matplotlib, so in order to run these programs; python, matplotlib, and pylab all must be installed.

3.1 Directory Tree

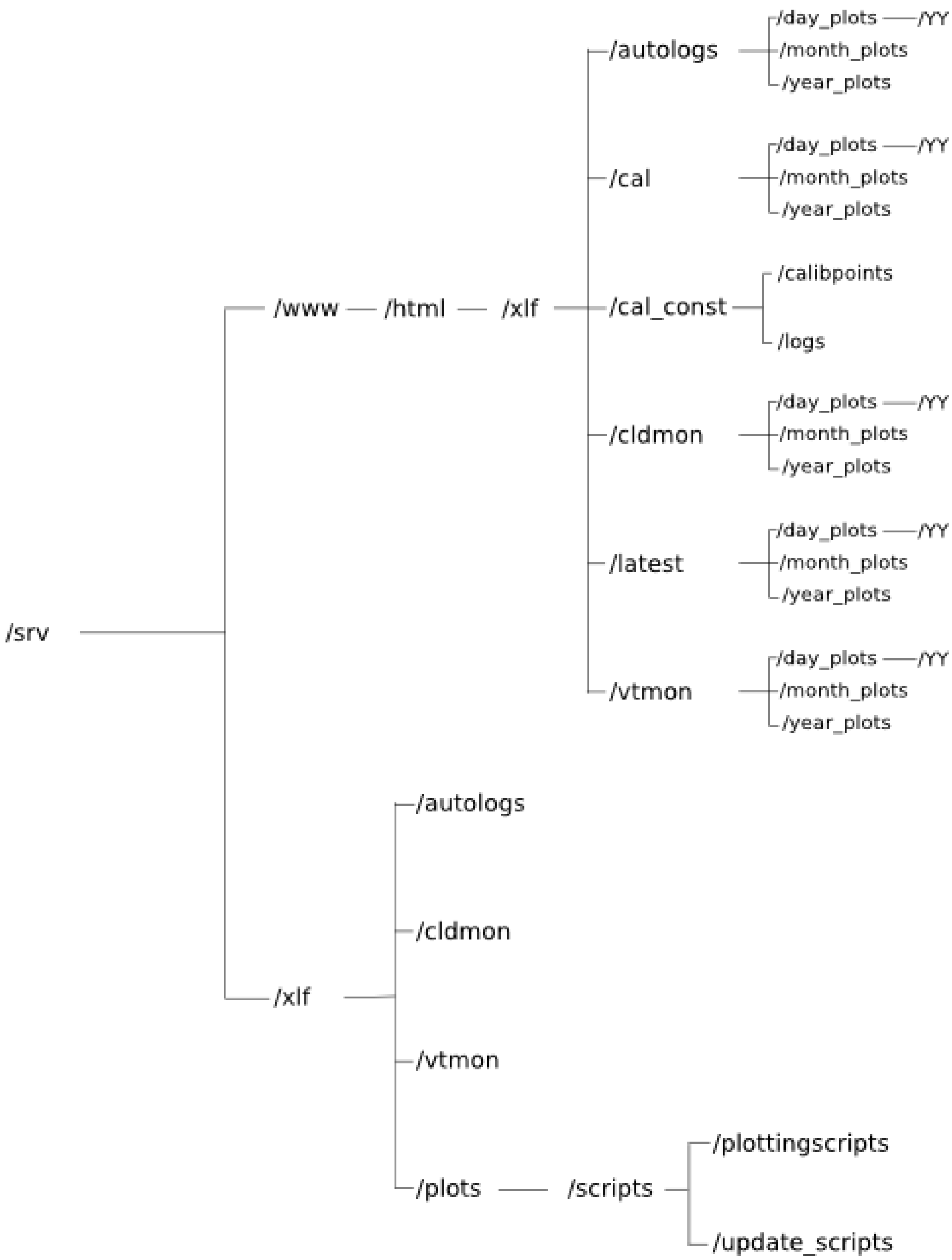

Figure 1: This shows the directory structure that is used for all of the plotting and data.

3.2 masterplot.sh and the Nightly Programs

The masterplot.sh is located at /srv/xlf/plots/scripts/masterplots.sh and does the following processes:

1. Runs all the nightly programs
2. Creates folders in the directory tree
3. Converts .eps images to .jpg
4. Moves these new images to where they need to be in the directory tree
5. Updates plots from previous nights
6. Runs the mkgallery scripts

Masterplot.sh does not need any input and it outputs the directory tree and images. This script is run nightly by the crontab under user levi.

3.2.1 xlfautolog

When using this program, run it in command line with the following command string:
"dayxlfautolog.py date [output location]"
Date should be in the format "yXXXXmXXdXX". This will open all the files of a particular day and plot them as a .eps file like Figure 2.

There are two more variations of this program that run the similarly:
"monthxlfautolog.py date [output location]"
"yearxlfautolog.py date [output location]"
These programs output an image under the name 'date.autolog.xlf.eps'.

This program has six plots. It plots the monitor energy, moniter rms energy, dpw, temperatures of the head, dump, and plate, and the nanosecond measurements that are all given directly from the autolog files in /srv/xlf/autolog.

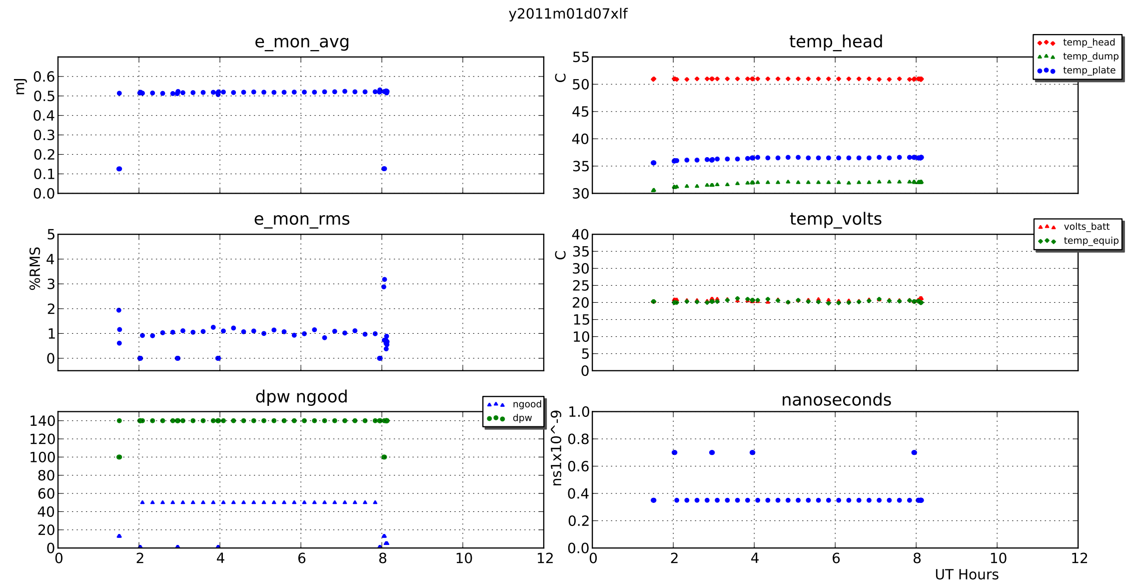

Figure 2: This plot is created by running dayxlfautolog.py y2011m01d07 /srv/www/html/xlf/autologs/day_plots/2011.

3.2.2 dayxlfautolog info

All the plotting python programs were written by changing the autolog program. A detailed description of how this base program works follows:
The first block of text imports packages needed while line "matplotlib.use('Agg') changes the environment so that this program can run on computers without a GUI.

```
import matplotlib
import os
matplotlib.use('Agg') #this is a fix so you can run the program on a OS without a GUI
import pylab
from pylab import *
```

These next lines make it possible to specify the date from the command line. The date is saved as the first option after the program name. This is then appended to the extension for opening the correct file.

```
date = sys.argv
date = sys.argv[1]
filename = date + '*.xlf2v1.autolog'#adds the extension to the filename
```

The first step here is to grep the lines desired from the file specified by the date command and then saved in a temporary file. This temp file is then opened with read privileges and separated into lines, the readlines() command is necessary for that. The data is then split off from the first column for each element in the newly created list. The next line floats the data so we can plot the data. Lastly the data is put into an array.

```
os.system('grep --binary-files=text STATSF ' + filename + ' >temp')#uses terminal to grep

data=open('temp','r').readlines()#reads the file as individual lines

data=[x.split()[1:] for x in data]#removes the STATSF column of the lines
data=[map(float,x) for x in data]#floats each column of data
dataarray=array(data)#puts data in an array
```

If you wanted to change which data is plotted this is were you would look up the variable that you need. There are some conversions here that should be noted. If a data set is not specified here but desired add it by following the format "dataarray[:,"whatever column it is in"].

```
e_mon_avg   = dataarray[:,4]*1000#this is a seperate column of data for each variable from
e_cal_avg   = dataarray[:,6]*1000
temp_head   = dataarray[:,14]/10
temp_dump   = dataarray[:,15]/10
temp_plate  = dataarray[:,16]/10
volts_batt  = dataarray[:,17]
temp_equip  = dataarray[:,18]
unused      = dataarray[:,19]
e_mon_rms   = dataarray[:,5]
nanosecond  = dataarray[:,26]/1000000000
dpw         = dataarray[:,10]
ngood       = dataarray[:,3]
time        = dataarray[:,23]+dataarray[:,24]/60+dataarray[:,25]/3600#conversion to UT tim
```

This step imports a couple more packages and adjusts the preset parameters that matplotlib uses for its plots. These steps fix any clipping effects. For isntance, if the label on the y axis on the right set of plots is overlapping the left set of plots, simply adjust the figure.subplot.right. You would need to make the value closer to one for that example.

```
import matplotlib.pyplot as plt#important package for plotting data
rcParams['legend.fontsize'] =8#this changes the built in parameters of the pyplot package,
rcParams['figure.subplot.left'] =.05#adjusts the how far the left subplots start from the
rcParams['figure.subplot.right'] =.95#adjusts where the right subplots start
#now when legend is used it will have different default fontsize
```

These steps create a figure and adjust the white space between the plots. If you wanted to change the figure size or move the plots closer together or farther apart, that would be done here. Also, the xdomain and pointsize can easily be adjusted for all the plots right here.

```
plots=plt.figure(figsize=(13.5,7.3))#makes a figure
plots.subplots_adjust(wspace=.10)#changes the spacing width for subplots
plots.subplots_adjust(hspace=.3)#changes the spacing height for subplots
xdomain=(0,12)
pointsize= 8
```

This is an example of the plots actually being configured. The first line specifies how many subplots and where the specific plot is located. In this example the subplot is 3 rows by two columns and the fourth plot. The next couple of lines specify which data is to be plotted in the format of scatter('x axis data', 'y axis data', 'marker type', 'marker color', 'edgecolor' , 'point size'). The next line of data is commented out but can easily be added again by removing the '#'. The next line makes a legend which is necessary for multiple data sets on one chart. To add more data sets to the legend just add('volts_batt','temp_equip','new data') in the order that they are added to the axScatter. bbox_to_anchor specifies the distance from the bottom left of the plot. (1,1) means top right so (1.1,1) means a little to the right of the top right. The next two lines specify the xbounds and the y bounds. The xbounds are already set, but if you wanted them to be different you could change them here. The title of the plot is set here and y labels as well.

```
axScatter = plots.add_subplot(324)
axScatter.scatter(time, volts_batt ,marker = '^', color='red', edgecolor= 'red', s=pointsi
axScatter.scatter(time, temp_equip ,marker = 'd', color = 'green', edgecolor= 'green', s=p
#axScatter.scatter(time, unused, color = 'blue', edgecolor='blue', s=8)
plt.legend(('volts_batt', 'temp_equip'),
           'right', bbox_to_anchor=(1.1, 1), shadow=True)
plt.ylim(0,40)
plt.xlim(xdomain)
title('temp_volts')
ylabel('                      C')
plt.grid(True)
axScatter.set_xticklabels([])
```

This block sets the title of the plot to the date and then saves the file as a .eps. Simply change the extension to change the file type. The last step removes the temporary file that was created.

```
plt.suptitle( date + 'xlf')#adds a title to the figure
plt.savefig(date + '.eps')#saves the figure and makes it an eps file
```

```
os.system('rm temp')#removes the temp file from the directory
```

3.2.3 xlfcalib

When using this program run it in command line with the following command string
"dayxlfcalib.py date [output location]"
The date should be in the format "yXXXXmXXdXX". This will open all the files of a particular day and plot them as a .eps file like the Figure 3.

There are two more variations of this program that run the similarly:
"monthxlfcalib.py date [output location]"
"yearxlfcalib.py date [output location]"
These programs output an image under the name 'date.calib.xlf.eps'.

Figure 3 has six plots. It plots the e_mon_avg, temperatures of the head, dump, plate, equipment, and battery. Those are all given directly off of the autolog files. e1100ratio is the measurement of the second probe divided by the emoniter energy at that particular measurement. This measurement should happen twice per day. e2140 ratio is the measurement of the third probe divided by the emoniter energy at that particular measurement. This measurement should happen twice per day. ecal2140 is taken directly from the autologs and should also happen twice per day. e_cal_1/e_cal_2 is the ratio of the second and third probes. These measurement are not taken simultaneously so the plots use GPS time to select the measurements that are in close time proximity and then divides them. There should be two per day. These measurements use the autolog files which are located in /srv/xlf/autolog.

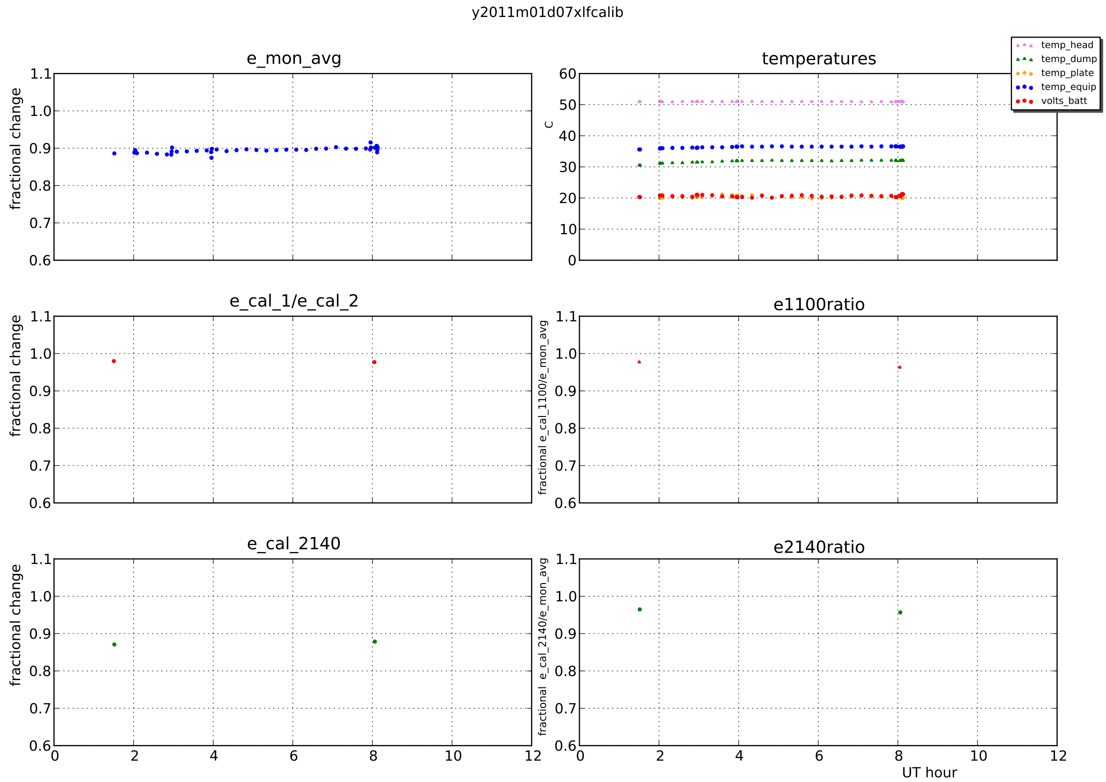

Figure 3: This plot is created by running dayxlfcalib.py y2011m01d07 /srv/www/html/xlf/cal/day_plots/2011.

3.2.4 xlfcldmon

When using this program run it in command line with the following command string
"dayxlfcldmon.py date [output location]"
The date should be in the format "yXXXXmXXdXX". This will open all the files of a particular day and plot them as a .eps file like the Figure 4.

There are two more variations of this program that run the similarly:
"monthxlfcldmon.py date [output location]"
"yearxlfcldmon.py date [output location]"
These programs output an image under the name 'date.cldmon.xlf.eps'.

Figure 4 has six plots which are all taken directly off of the cldmon files in /srv/xlf/cldmon.

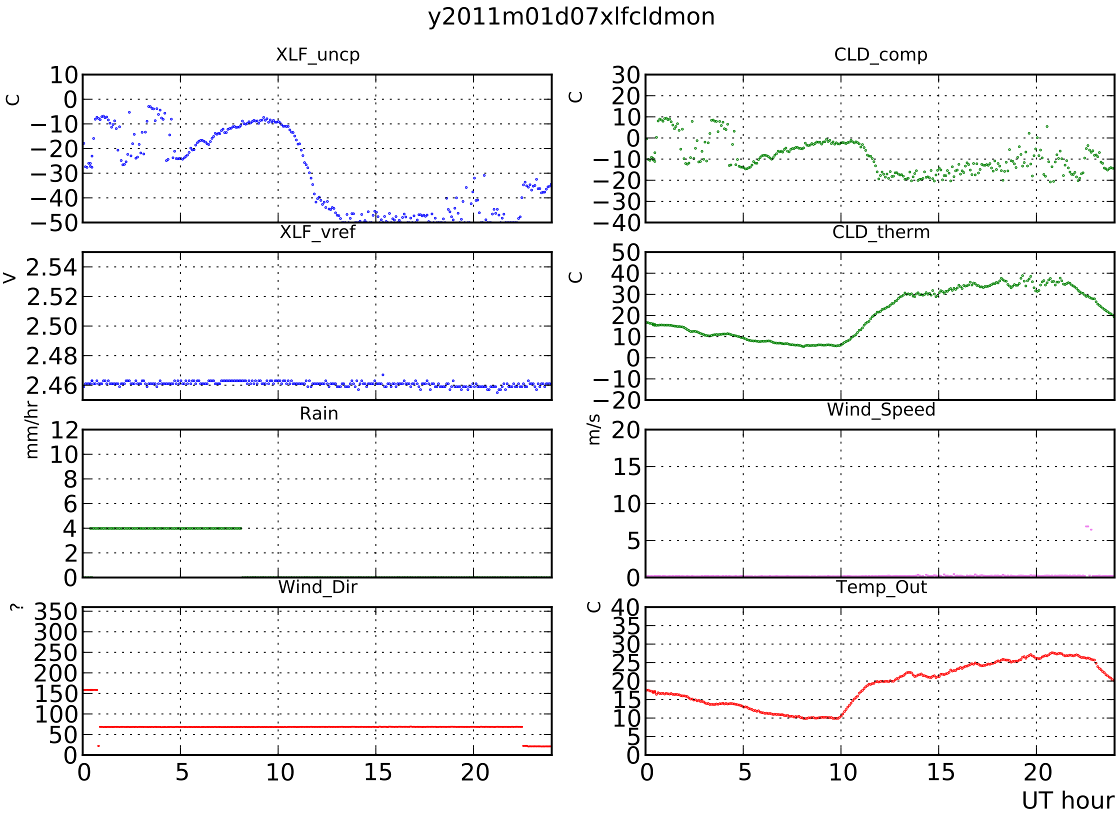

Figure 4: This plot is created by running dayxlfcldmon.py y2011m01d07 /srv/www/html/xlf/cldmon/day_plots/2011.

3.2.5 xlfvtmon

When using this program run it in command line with the following command string
"dayxlfvtmon.py date [output location]" date should be in the format "yXXXXmXXdXX". This will open all the files of a particular day and plot them as a .eps file like the following image.

There are two more variations of this program that run the similarly:
"monthxlfvtmon.py date [output location]"
"yearxlfvtmon.py date [output location]"
These programs output an image under the name 'date.vtmon.xlf.eps'.

Figure 5 has eight plots that are all taken directly off of the vtmon files in /srv/xlf/vtmon.

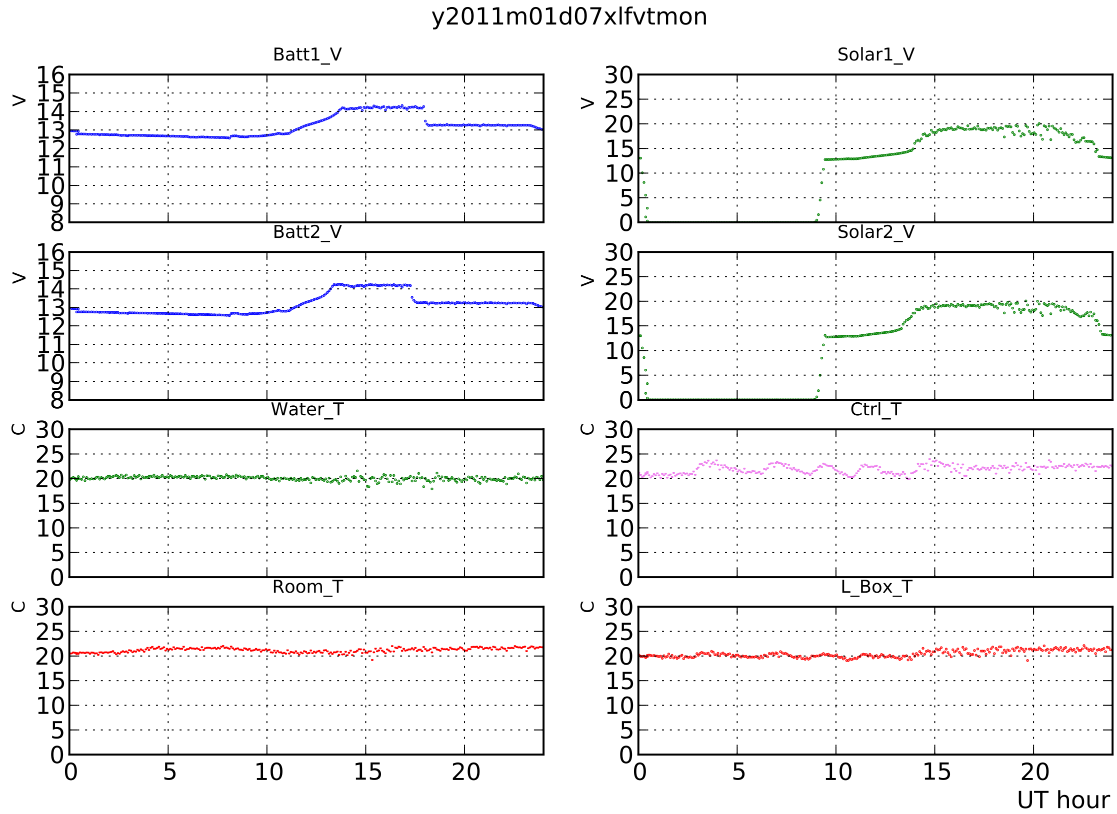

Figure 5: This plot is created by running dayxlfvtmon.py y2011m01d07 /srv/www/html/xlf/vtmon/day_plots/2011.

3.2.6 monthxlfcalibpoints

When using this program run it in command line with the following command string
"monthxlfcalibpoints.py date [output location] [data path]"
date should be in the format "yXXXXmXXdXX". This will open all the files of a particular day and plot them as a .eps file like the following image.

This program outputs two images under the name 'date.calibpoints.xlf.eps' and 'year.cal_const.xlf.eps'.

Figure 8 has one plot which shows exactly what calibration measurements points we have for a given day. If zero is the value for that day it means we have all six calibration points for that day. If more than one point is plotted for the day then there is more than on calibration measurement missing. If 1 is lit up it means we are missing the first measurement which is E2100. E stands for early and L stands for late. The other measurements are, 2 for E3100, 3 for E3140, 4 for L2100, 5 for L3100 and 6 for L3140.

Figure 7 has one yearly plot that shows the calibration constant over the course of the year. The calibration constant is more deeply explained in the next section. The different colors show which data points were missing for the individual day which hints at the statistical error. The legend shows which points are missing based on the representative color. If the legend says 000000 that means we have all data points. If the legend says 111111 we are missing all the data points. It is important to note that I am ignoring the farthest left digit point, as we are more interested in the cal_const. Therefore, if the legend says 000100 we are missing data point 3 which is E3140. If the legend says 0001xx, then it means that we are missing one or more of the x digits but it doesn't matter which digit because we won't be able to use it in our calculation anyway. Which data points we can use if we are missing others is further explained in the next section.

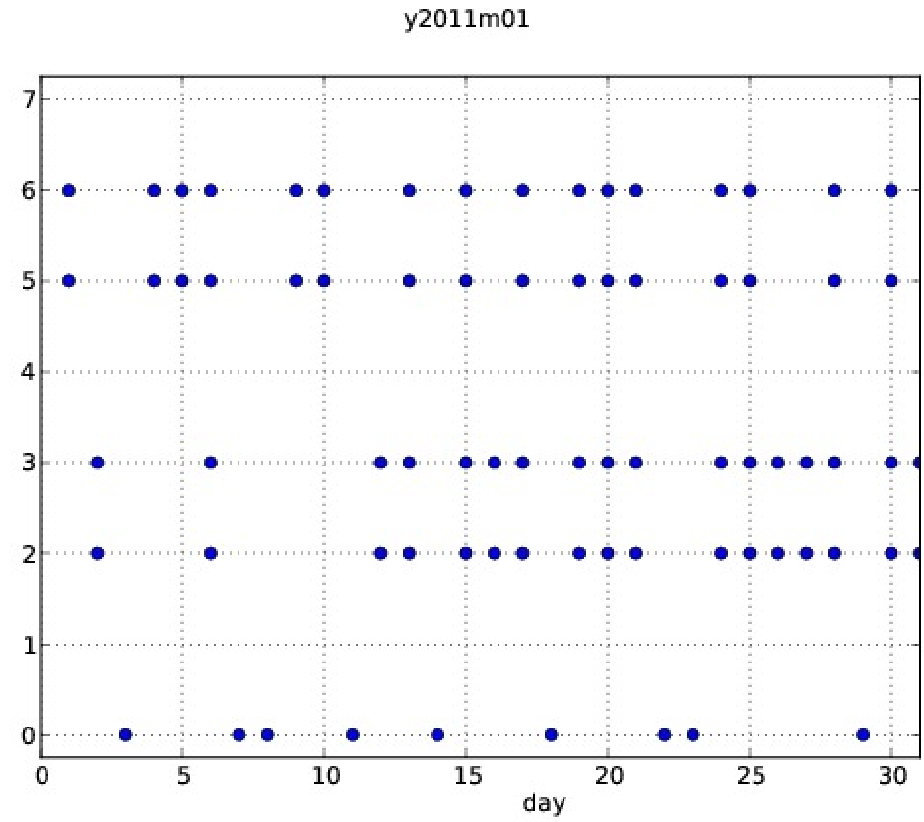

Figure 6: This plot is created by running monthxlfcalibpoints.py y2011m01d07 /srv/www/html/xlf/cal_const /srv/www/html/xlf/cal_const/logs

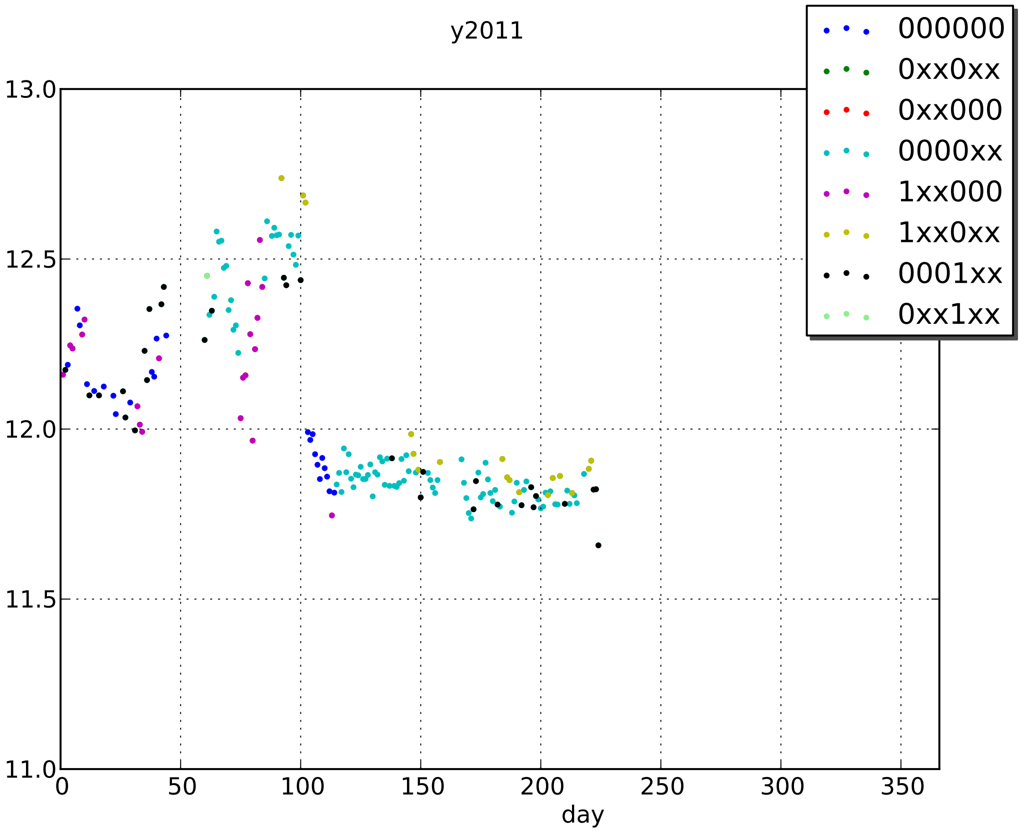

Figure 7: This plot is created by running monthxlfcalibpoints.py y2011m01d07 /srv/www/html/xlf/cal_const /srv/www/html/xlf/cal_const/logs

3.2.7 Update Scripts

The masterplot.sh also runs an update for the last thirty files created. But if a larger update is needed, this is where that would be done. The scripts that update all plots are as follows.

1. dates.sh
2. compile.sh
3. compileautolog.sh
4. compilecldmon.sh
5. compilevtmon.sh
6. compilecalib.sh
7. compile.sh

dates.sh is a program that uses bash commands such as 'ls' and 'sed' to create a list of dates directly from the data files then outputs all the dates into a series of text files. These text files are named similar to the program that uses them, autologdates.txt, cldmondates.txt, vtmondates.txt. Note that calib programs use the autologdates.txt. compile.sh takes those dates and then runs all the plotting programs. compileautolog.sh runs all the autolog dates and runs the autolog programs. These programs need to be updated.

4 Energy Calibration of the XLF

The year.cal_const.xlf.eps plot shows the calibration constant as a function of time. The calibration constant is produced by the dayxlfcalibfile.py program which outputs one constant per day. When the system is running correctly, the xlf records three data points in the beginning of the night and three at the end of the night. The first data point used is recorded by probe 2 and is shot at 100 dpw. The second is 140 dpw measured by probe 3. Lastly the third point is probe 3 measuring the calibration energy at 100 dpw. **??** is a schematic of the xlf laser and calibration system.

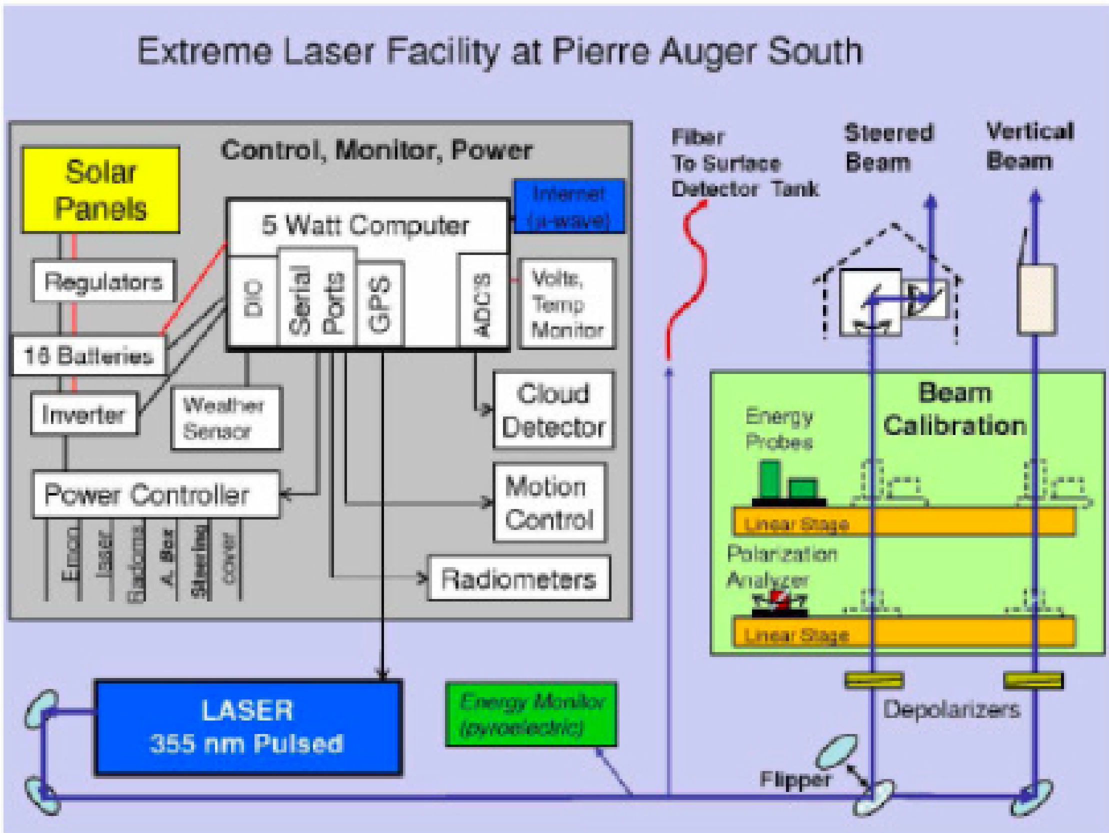

Figure 8: This shows the schematic for the eXtreme Laser Facility.

4.1 The cal_const File and Calibration Measurements

The dayxlfcalibfile.py script provides a file in the following format:

The flag has seven digits and the farthest left digit shows whether or not non-calibration shots were fired that night. If only calibration shots are fired that day then the most left digit will be 1. The other digits correspond to the calibration probes and are further explained.

With calibration measurements at the beginning and end of the night we can use an average of four calibration constants to get a calibration constant ,'C', for the whole day. We will call these measurements by their time of day, dpw, and probe number. So E2100 means the early energy calibration of probe 2 at 100 dpw. Emon is the monitor energy at the second data point.

$$C = \frac{1}{4}(\frac{E2140}{Emon} + \frac{E3140}{Emon} + \frac{L2140}{Emon} + \frac{L3140}{Emon}) \tag{1}$$

We do not actually have a data point for E2140 or L2140 so we will have to use the 2100 and the 3100 measurements to approximately scale the 2100 to 2140.

$$2140 = 2100\frac{3140}{3100} \tag{2}$$

$$C = \frac{1}{4}(\frac{E2100\frac{E3140}{E3100}}{Emon} + \frac{E3140}{Emon} + \frac{L2100\frac{L3140}{L3100}}{Emon} + \frac{L3140}{Emon}) \tag{3}$$

$$C = \frac{1}{4}[\frac{E3140}{Emon}(\frac{E2100}{E3100} + 1) + \frac{L3140}{Emon}(\frac{L2100}{L3100} + 1)] \tag{4}$$

This is the final calibration constant that we apply to our date each night. But in many cases we are missing data so we have to use the following equations in order to compensate for this. The days that we are missing points and which points we are missing are outlined in dayxlfcalibpoints.py outputs. The system the program uses to recognize which data points are missing is by digits. If the program inputs 000000 it means we have all the calibration measurements and we will use the given calibration equation. If the program inputs 111111 then we have no calibration measurements and 0 is the calibration constant for that day with a flag of 111111. Also, if we are missing any of the 140 dpw measurements then we can not apply a calibration equation and the calibration constant for the day appears as 0.

For '011011'

$$C = \frac{1}{2}[\frac{E3140}{Emon} + \frac{L3140}{Emon}] \tag{5}$$

For '011000

$$C = \frac{1}{3}[\frac{E3140}{Emon}(\frac{E2100}{E3100} + 1) + \frac{L3140}{Emon}] \tag{6}$$

For '000011'

$$C = \frac{1}{3}[\frac{E3140}{Emon} + \frac{L3140}{Emon}(\frac{L2100}{L3100} + 1)] \tag{7}$$

For '1xx000'

$$C = \frac{1}{2}[\frac{E3140}{Emon}(\frac{E2100}{E3100} + 1)] \tag{8}$$

For '1xx0xx'

$$\frac{E3140}{Emon} \tag{9}$$

For '0001xx'

$$C = \frac{1}{2}[\frac{L3140}{Emon}(\frac{L2100}{L3100} + 1)] \tag{10}$$

For '0xx1xx'

$$\frac{L3140}{Emon} \tag{11}$$

These equations are lacking a method for error calculation. The application for the cal_const is to multiply th cal_const for the day by the emonitor energy. This will give you a more accurate depiction of how much energy is being released into the sky.

4.2 Plots and Analysis

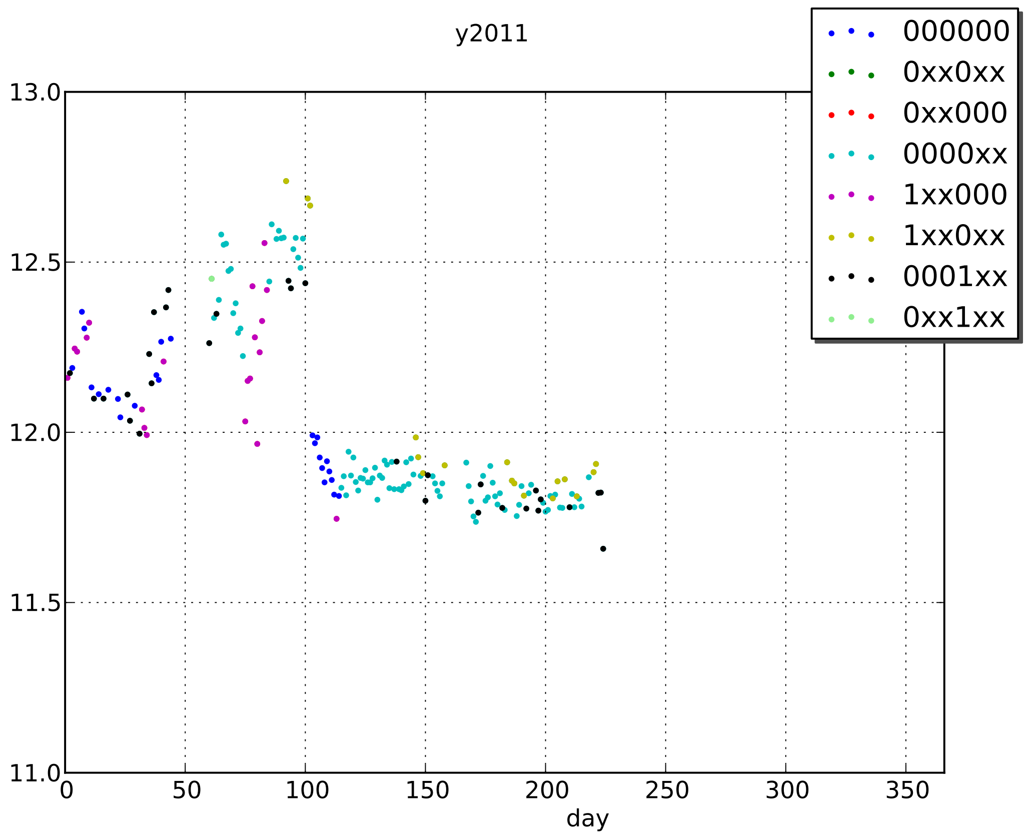

Figure 9: This plot is created by running monthxlfcalibpoints.py y2011m01d07 /srv/www/html/xlf/cal_const /srv/www/html/xlf/cal_const/logs

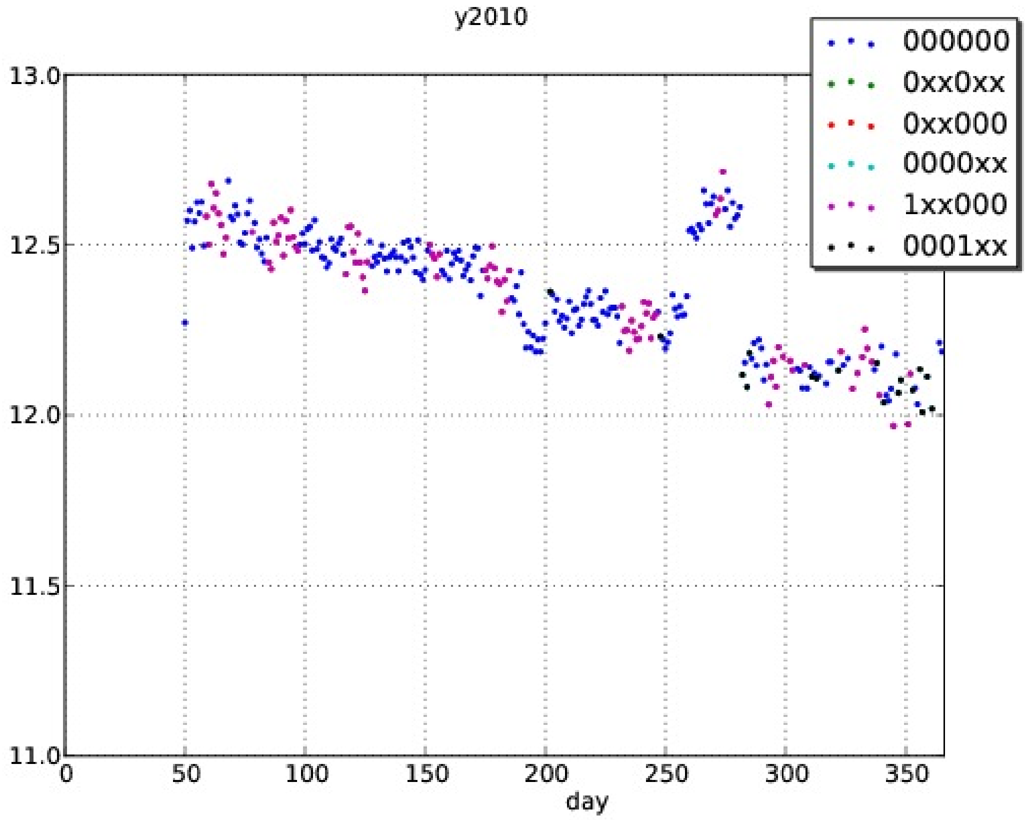

Figure 10: This plot is created by running monthxlfcalibpoints.py y2010m01d07 /srv/www/html/xlf/cal_const /srv/www/html/xlf/cal_const/logs

The calibration system was started in February so there is no data before the 50th day of the year. The data for 2010 shows a band of points which show a statistical error of about 0.25. The decline shows that the probes may be aging or something is causing one of the probes to be declining in recorded energy. The problem seems to be with the 3rd probe and the calib plots support this claim. The 2011 data seems less consistent, but this may be because we have less calibration measurements per day. Most days in 2011 are missing data points.