1 D-TACQ Bolometer Calibration Comparison

1.1 Introduction:

This document is a recording of the process of taking and analysing data from a bolometer using the D-TACQ BOLO8BLF. The document looks to find if the data acquired from the bolometer over one channel is accurate over several shots. The data is taken twice for two use cases: prolonged data capture with single calibration before 100 shots of data capture, and data capture with device calibration performed before each shot is taken. In both cases each shot contains 10^5 data points, and each shot takes approximately 22 seconds to acquire. One calibration takes 10 seconds to acquire. To take 100 shots takes approximately 37 minutes with one calibration at the start, and with one calibration for every capture the acquisition time is 55 minutes.

1.2 Analysis Techniques:

By taking the means of each shot the variation in the data can be inspected. By plotting the means it is possible to ascertain whether or not the bolometer and the apparatus used to control it is accurate over time with both calibration practices. The comparison between these two methods of data acquisition should illustrate the ideal way to ensure bolometer data stability.

1.3 Analysis of the mean value from each data shot:

The mean values across both data sets provided in Fig. 1 show that the values output by the bolometer are far more accurate when a calibration is not performed for every data capture.

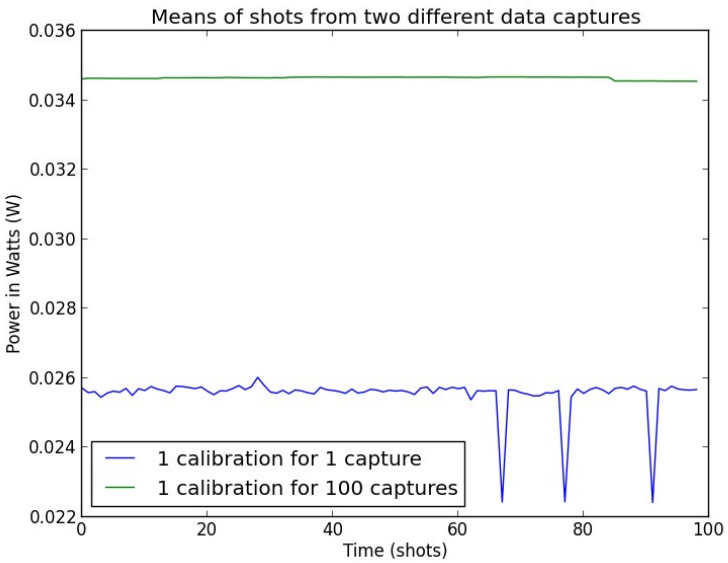


Fig. 1: A comparison between two different methods of calibration before data capture.

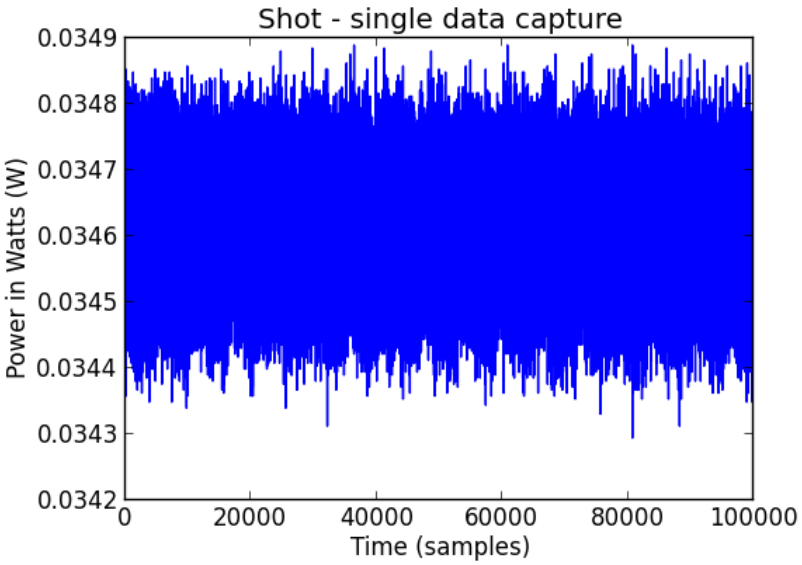


Fig. 2: A view of a single data shot from the single calibration per 100 shots series.

From Fig. 1 it can be seen that the mean value of each shot is far less variable when one calibration is performed before data capture. The mean values acquired when a calibration is performed for every shot are slightly more variable. Performing calibrations before each shot also means that the data is more susceptible to variation in the calibration, as can be observed from the three distinctive sharp troughs. Fig. 1 would indicate that the optimum method for taking data using a bolometer with this apparatus would be to perform a single calibration before taking the required amount of data. Fig. 2 reiterates this indication, as it can be seen that the data is forming a horizontal line.

1.4 Conclusion:

The conclusions from the data acquired suggest that the optimum method for data acquisition using a bolometer with the BOLO8BLF set-up is to take a single calibration for a series of shots. Not only does the data not “droop” over time, there is also some risk with taking calibrations for every data shot required. This means that a calibration will hold its validity for at least 37 minutes.

2 User Guide:

This section of the document will serve as a user guide for the BOLO8BLF set-up. It will take a new user through the initial procedures of setting up the BOLO8BLF module on the acq2106_089. The procedure includes installation of MDSplus software, setting up and using the python bolometer HAPI scripts, changing the channels the python scripts are to use, and viewing the data that has been taken using the MDSplus python module. Also included are some FPGA version and Firmware release information, which was taken just prior to shipping.

2.1 MDSplus:

MDSplus needs to be installed on the host computer. A new tree should be created to take data. Once this is done the MDSplus python module can be used to read data from the MDSplus tree and then it can be analysed. Once MDSplus is installed, a new tree can be created by using the make_acqtree.py script then using the make_bolo_tree.py python script. Then the new_shot script should be used- this will create a new tree in the specified MDSplus tree directory, and take a shot, so that the automatic calibration and capture python utility can be used. To create the tree an environment variable must be established. There is a specific syntax that the MDSplus python module takes, and it is as follows.

Environment variable is established in terminal (syntax below is for linux and may vary on windows):

```
export acq2106_089_path=/<path_to_tree>
```

More information on this can be found on General Notation: The Data Hierarchy - Trees, Nodes, and Models.

2.2 Bolometer data acquisition method:

The data from the bolometer was taken using a python script (bolo8_cal_cap_loop.py). This python script (and all of its related scripts) can be found on https://github.com/petermilne/acq400_hapi_tests and should be installed. It runs from the command line and can take several optional parameters in order to customise the way in which data is acquired. The bolo8_cal_cap_loop.py script can calibrate the device and can also capture data. Calibrations are stored on an MDSplus tree under odd numbered shots and the data shots are stored under even numbered shots.

The (most important) parameters taken by the script are as follows:

```
Parameters:
--cal <boolean> - If the script should run a calibration.
--cap <boolean> - If the script should run a data shot.
--post <number> - Number of data points per capture.
--shots <number> - Number of times calibration and/or capture should run.
```

In order to use the script with MDSplus there must also be an environment variable established and passed to the script as a parameter. This tells the script where to look for the MDSplus tree. There is a specific syntax that the MDSplus python module takes, and it is as follows.

Environment variable is established in terminal (syntax below is for linux and may vary on windows):

```
export acq2106_089_path=<server_where_data_is_stored>:://<path_to_tree>
```

Note that this sets up the environment variable on the host machine, for accessing the tree on a server where the data is stored. If the data is being stored on the host machine then the syntax is the same as when the tree was initialised:

```
export acq2106_089_path=/<path_to_tree>
```

Once the environment variable has been established the python script can be run:

```
python  bolo8_cal_cap_loop.py --cal 1 --cap 1 --shots=1 acq2106_089
```

Note that when running the python script that you only pass acq2106_089 (not the full environment variable acq2106_089_path). For more information on this please consult Remote Data Access in MDSplus. When the script is run it will write the data specified by the user (calibration and/or capture data) to the MDSplus tree by using the postshot.sh script found on the acq2106_089. This data can later be accessed using python (or CS Studio, MDSscope or another data visualisation tool).

2.3 Changing the channels to be calibrated:

There is a single file that is used to change which channels are to be calibrated by the acq2106_089. This must be changed on the acq2106_089. Navigate to /mnt/local/sysconfig/ and edit the bolo.sh shell script.

Its contents should read as such (or similar):

```
BOLO_ACTIVE_CHAN="1 2"
BOLO_VERBOSE=1
set.site 14 DIODE_DROP_V 0.5
set.site 14 THEAT 1.0
set.site 14 TCOOL 1.0
set.site 14 VBIAS 1.0
```

The BOLO_ACTIVE_CHAN variable can be changed to the desired channel(s). In the example above it is set to calibrate channels 1 and 2. The python script bolo8_cal_cap_loop.py will calibrate (if told to) the channels in the order they appear in this script. The other values set in this script set control knob values.

2.4 Viewing data using the MDSplus python module:

Data can be viewed and analysed using the MDSplus python module. Doing this also requires the environment variable described above, so if a separate terminal is being used on linux then the environment variable initialisation process will need to be repeated. The following commands are extremely useful for viewing data from an MDSplus tree:

```
tree = Tree("acq2106_089", shot) # opens the tree
node = tree.getNode("BOLO1.PWR_" + str(channel)) # opens the node
unfiltered_data = node.getData() # reads unfiltered data
filtered_data = unfiltered_data.data() # makes the data readable to python
```

These steps can be automated and the data can be analysed and plotted for any amount of shots.

2.5 MDSplus plotting utilities and relevant data:

DWscope is an extremely useful data visualisation tool for looking at data quickly. It allows the user to visually verify whether the last calibration was effective or not. By looking at the PHI, MAG, PWR (phase,

magnitude and power) the channel can be verified to be calibrated. This however won't tell you much about the recency or relevance of the calibration.

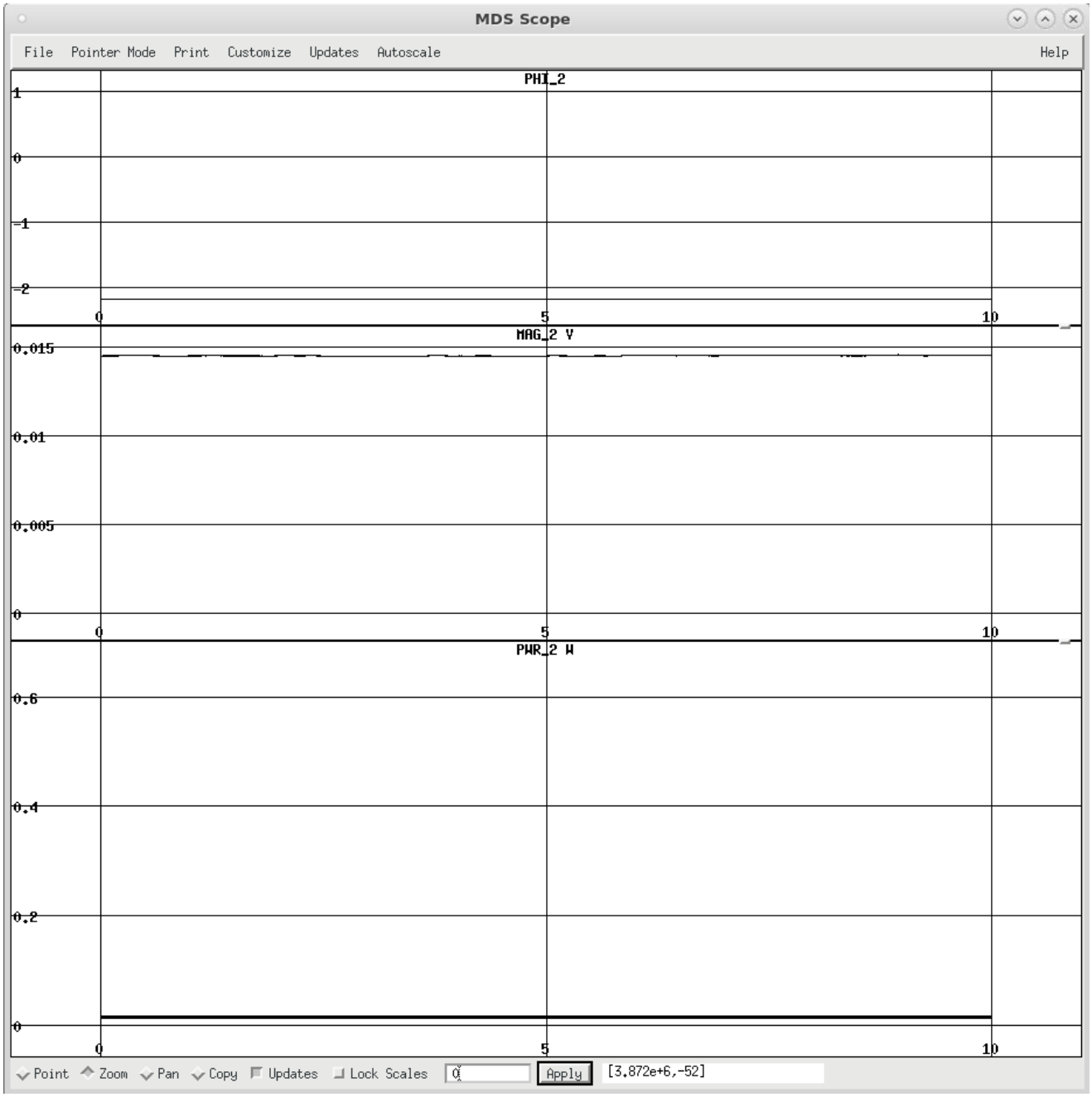


Fig. 3: A plot of calibrated phase, magnitude and power.

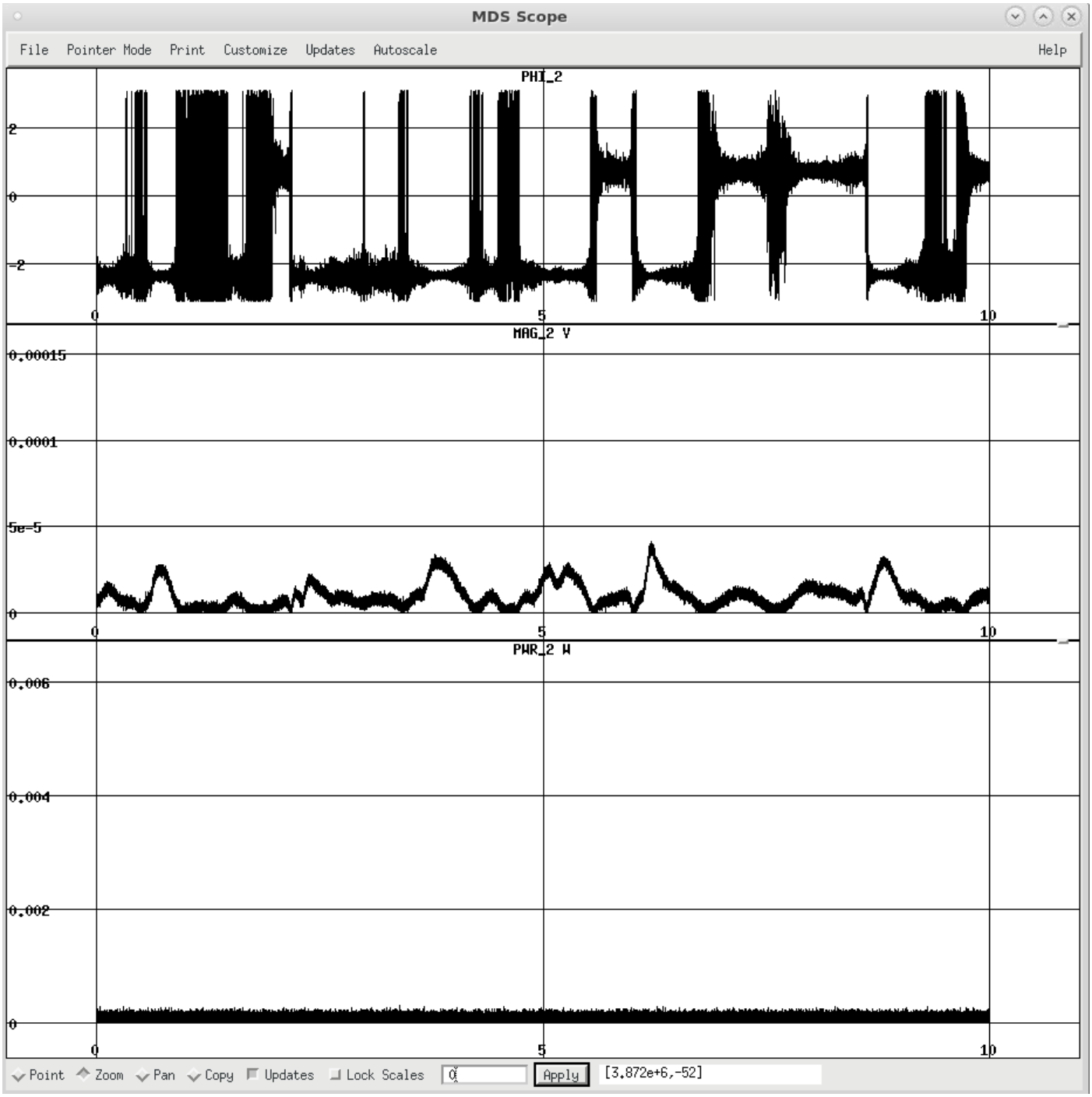


Fig. 4: A plot of uncalibrated phase, magnitude and power.

From figures 3 and 4 it can be determined whether the device has been calibrated. Furthermore, if DWscope is used to inspect each channel then it will provide more information about the data captured by the bolometer.

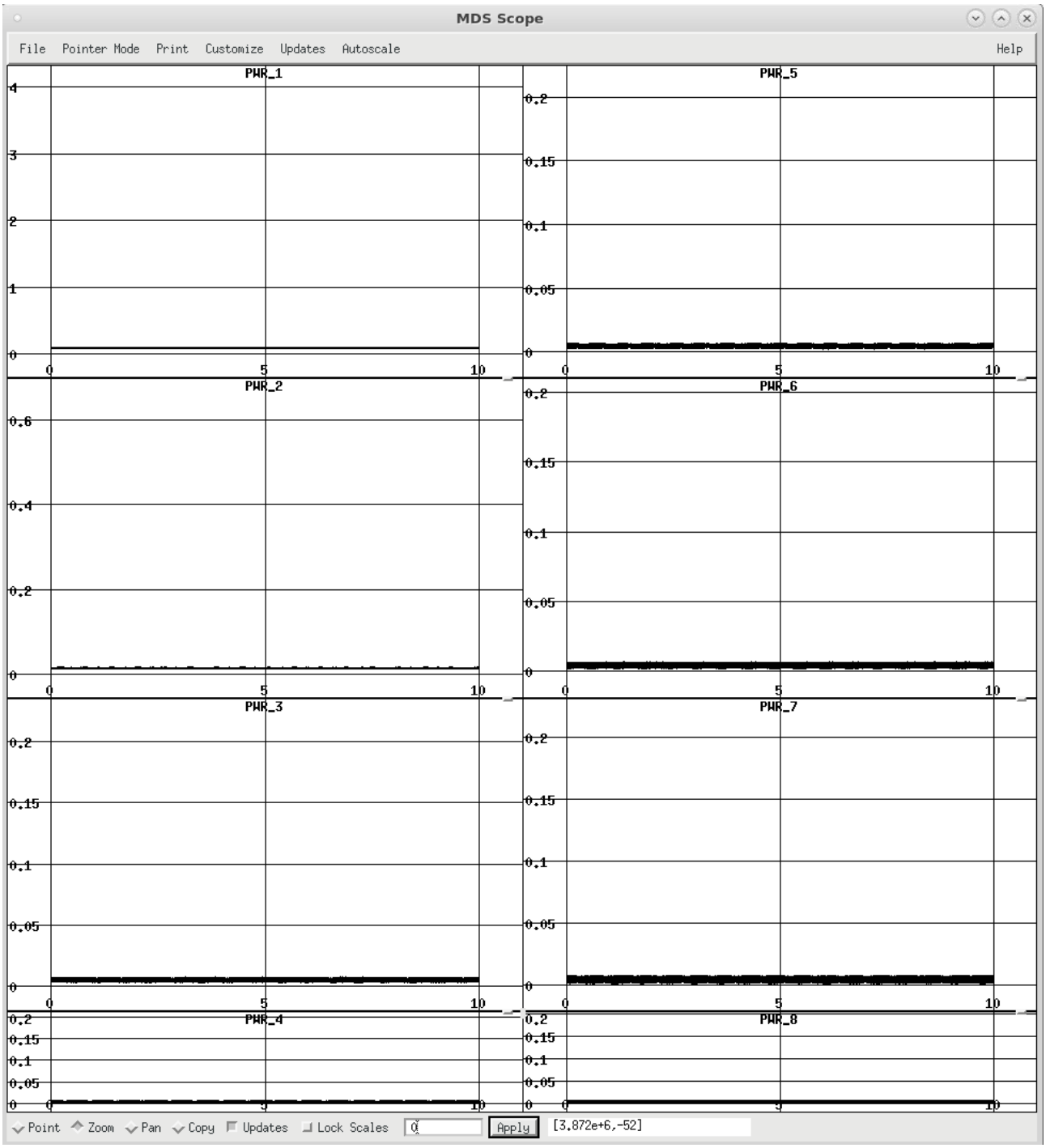


Fig. 5: Uncalibrated power channels 1 through 8.

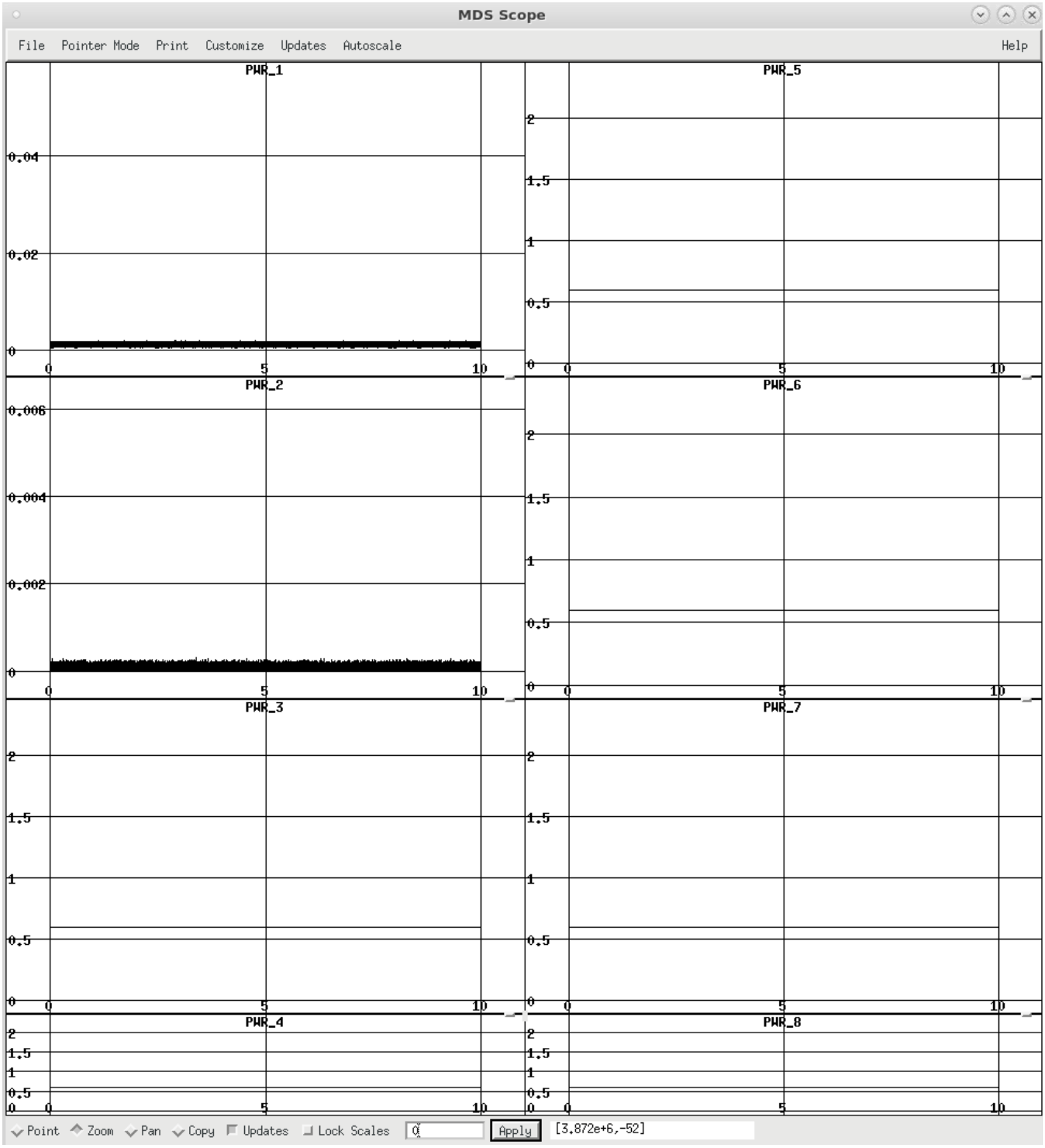


Fig. 6: Calibrated power channels 1 through 8.

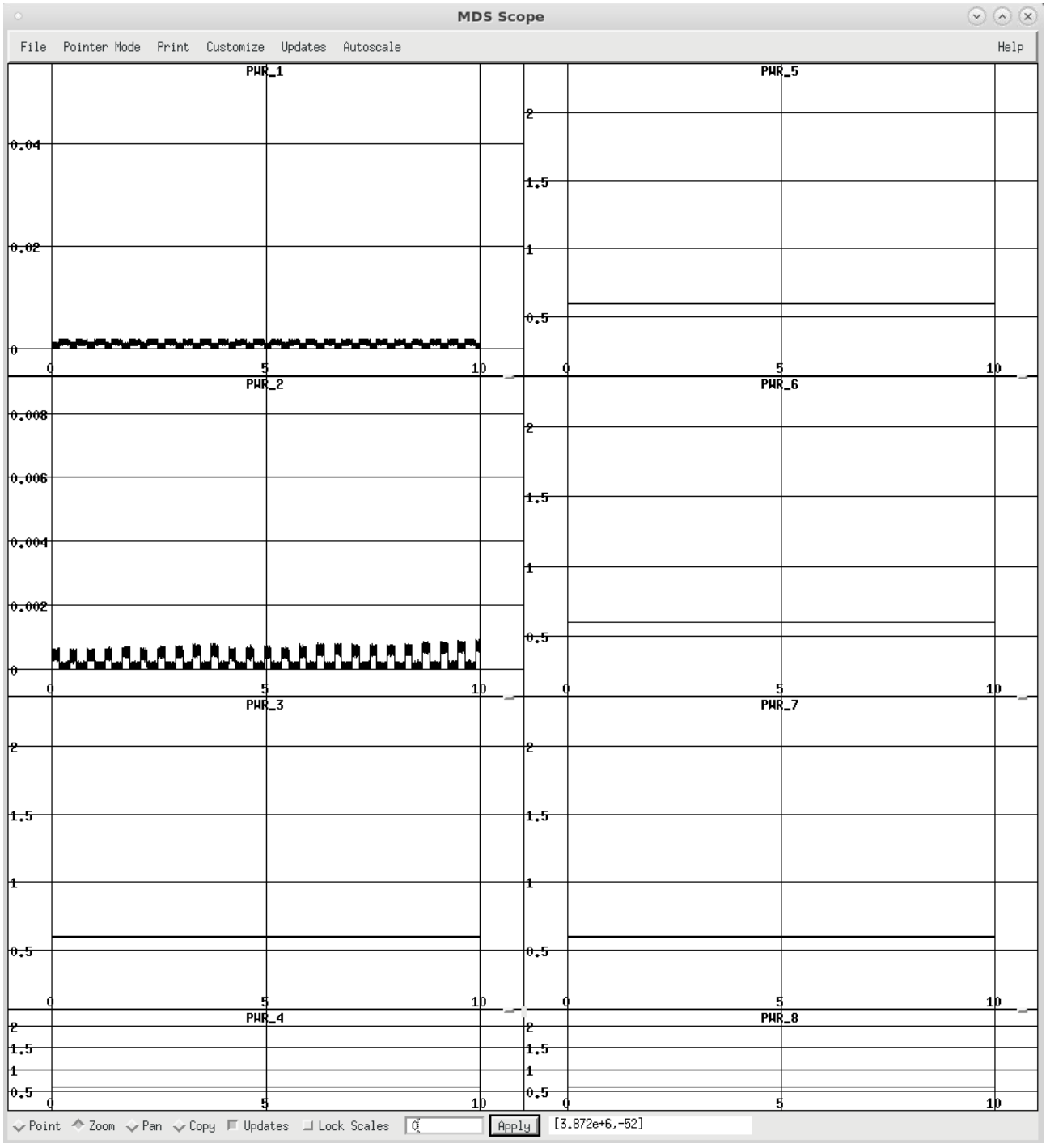


Fig. 7: Calibrated power channels 1 through 8 with a flashing bicycle lamp.

Figures 5, 6 and 7 show the ways in which DWScope can offer useful data about the data just obtained through MDSplus. Figure 5 is a completely uncalibrated system showing a lack of useful signal on any channel. Figure 6 depicts the background radiation in the test environment, and figure 7 was taken with a

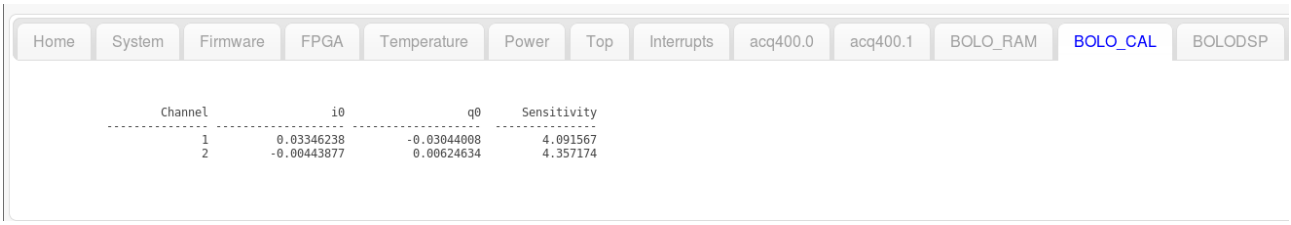
flashing bicycle lamp held over the bolometer. These graphs are useful when used in a test environment and MDSscope can even be used for some analysis of timings and values. For anything more complex it is useful to have Python and MatPlotLib.

2.6 Calibration and Voltage offsets readout:

The calibration data and voltage offsets can be viewed using the acq400 embedded web pages. These can be viewed by pointing a web browser to the device. The calibration and offset data can be viewed under the BOLO_CAL and BOLO_RAM tabs respectively.

Channel	Voltage Offsets 1	Voltage Offsets 2	Power Offsets 1	Power Offsets 2
1	808424	0	3087	0
1	-735407	0	-2808	0
2	-107237	0	-384	0
2	150906	0	541	0
3	0	0	0	0
3	0	0	0	0
4	0	0	0	0
4	0	0	0	0
5	0	0	0	0
5	0	0	0	0
6	0	0	0	0
6	0	0	0	0
7	0	0	0	0
7	0	0	0	0
8	0	0	0	0
8	0	0	0	0
9	0	0	0	0
9	0	0	0	0
10	0	0	0	0
10	0	0	0	0
11	0	0	0	0
11	0	0	0	0
12	0	0	0	0
12	0	0	0	0

Fig. 8: A screenshot of the BOLO_RAM embedded web page.



Channel	i0	q0	Sensitivity
1	0.03346238	-0.03044008	4.091567
2	-0.00443877	0.00624634	4.357174

Fig. 9: A screenshot of the BOLO_CAL embedded web page.

Fig. 10: A screenshot of the BOLODSP embedded web page.

These pages contain information about the calibration and the offset values that are loaded into RAM.

2.7 Firmware release and FPGA version:

```
RELEASE acq4xx-595-20171201154219
RELEASE : /tmp/release.md5
CURRENT : /tmp/current.md5
--- /tmp/release.md5
+++ /tmp/current.md5
+7e9ef13b3c0e0925e7b7b39f986c9b52  ./packages/35-bolodsp-v17.tgz
+2ed6d46d9ff4ba6e247efb2d84f51739  ./packages/70-mdsshell-1702031945.tgz
-2ed6d46d9ff4ba6e247efb2d84f51739  ./packages.opt/70-mdsshell-1702031945.tgz
Warning, patching detected
```

Fig 11. Firmware release

```
load.fpga loaded /mnt/ACQ2106_TOP_64_64_64_64_64_64_BOLODSP.bit.gz
xiloader r1.01 (c) D-TACQ Solutions
eoh_location set 0
Xilinx Bitstream header.
built with tool version   : 3f
generated from filename   : ACQ2106_TOP_64_64_64_64_64_64
part                      : 7z030fbg676
date                      : 2017/02/06
time                      : 10:20:51
bitstream data starts at  : 125
bitstream data size       : 5979916
```

Fig 12. FPGA version

Figures 11 and 12 show information regarding the Firmware release and FPGA version at the time of shipping.