Internship report SRON

Testing the X-IFU Focal Plane Assembly: PID Temperature Control & Applying Machine Learning to Detector Classification

Author:
Arnold Dongelmans

Examiners:
Damian Audley
Gert de Lange
Willem-Jan Vreeling

July 17, 2020

Abstract

This internship report describes two pieces of work done to support ground testing of the X-IFU [1] focal plane assembly (FPA). X-IFU stands for X-ray Integral Field Unit, a cryogenic spectrometer, based on a large array of Transition Edge Sensors [2] (TES) which is to be built into the Advanced Telescope for High Energy Astrophysics (ATHENA) [3] chosen for ESA's Cosmic Vision programme. In the first part of the report, efforts to regulate the 50mK, 300mK and 2K stages of the thermal ground support equipment (TGSE) are described. The aim is for the TGSE to provide a stable thermal environment for testing the X-IFU FPA. The second part of the report is dedicated to a machine-learning approach to classifying the characteristic curves of TES detectors in order to automate the characterization and calibration of X-IFU's kilopixel detector array.

Contents

1 Introduction **2**

2 Temperature control **2**
2.1 TGSE 2
2.2 PID controller 2
2.3 Technical jargon 3
2.4 Physics related to the TGSE 3
2.5 Results of measurements 4

3 Machine learning **8**
3.1 Introduction 8
3.2 How we determine the error 9
3.3 Neural network or support vector machine? 9
3.4 Preprocessing 10
3.5 The neural network implementation 11

4 Conclusion **11**

A Neural network scripts **12**
A.1 read_qdp.py 12
A.2 trainingdataobtainer.py 14
A.3 NNtrainer.py 15
A.4 tester.py 16

1 Introduction

X-IFU is an instrument proposed for the Athena X-ray Observatory; the second large mission of the European Space Agency science program, to be launched in the early 2030s. The Athena mission is designed to implement the Hot and Energetic Universe science theme. The project in this report is about experimentally verifying if the temperature of X-IFU's detector can be held stable, that is by measuring the fluctuations around the temperature equilibrium level (as there are no high energy particles entering the detector in our lab, any fluctuations we find must be the result of solely the TGSE).

2 Temperature control

2.1 TGSE

The thermal ground support equipment consists of several stages which are all connected to each other, see figure 1. That is why the temperature control of one stage may interfere with the temperature control of another stage. This results in a non-linear control system. The goal is to control all three stages, the 50mK, 300mK and 2K stages simultaneously using three different PID controllers. The thermal straps linking the stages are made of copper and the purity of this copper, which plays a role in heat conduction, can be determined by calculating the heat conduction from experimental results. The obtained RRR-value is a good quantifier of this purity, as will be explained in section 2.4. The goal is to make sure the obtained control of the temperatures and more importantly its standard deviation, are classified. Once the detector goes into space, it will need a stable thermal environment to maximize its sensitivity.

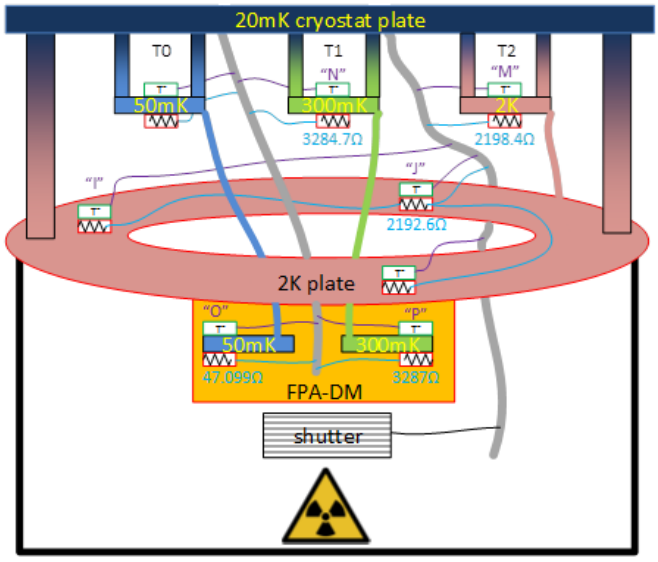


Figure 1: The thermal ground support equipment showcasing the different stages, the "sensors labels", and the different straps. This is the configuration on 17-01-2020.

2.2 PID controller

To control the 2K stage, a Lakeshore 218 temperature controller was used in conjunction with a power supply to regulate the heating. The 300mK and 50mK stages were to be controlled by AVS47B and AVS47 resistance bridges, respectively. All controllers measure the resistance of a

NTC[1] Lakeshore RX-102B-CB sensor which is done by letting an excitation current flow through the sensor. The higher the excitation current, the more accurate the reading is made. However, self-heating will start to affect your reading as the current dissipates via Joule-heating. The reading itself is a 4-wire measurement, to increase the accuracy of the measured resistance. The 4-wire method's advantage over the 3- or 2-wire method is that one avoids lead resistances due to high-ohmic wiring [4].

As the temperature controllers need to know what resistance corresponds to what temperature, they need calibration curves. Every different *type* of sensor needs a different type of calibration curve. An RTD needs a log(R) vs. temperature calibration curve, whereas a diode makes use of a voltage vs. temperature plot.

A Labview program was used to log the temperature readings and communicate with the temperature controller. This was done using GPIB cables which one can "daisy chain" so multiple devices can send information to the same GPIB slot of the computer. The PID values, the power bias and the heater power range were optimized to reach the desired temperatures swiftly and with as few fluctuations as possible. There exist premade methods to optimize these values such as the Ziegler-Nichols method [5] but this turned out not to work. Instead, we approached the correct values using trial and error.

There are two questions to be answered:

1. How much power is needed to reach the desired temperature?

2. What is the smallest $\frac{\Delta T}{dQ}$ we can obtain near the desired temperature? I.e. what is our sensitivity for measuring power flow from temperature differences. Here Q is the power.

2.3 Technical jargon

-*Shunt resistance*: In case the output of a power supply is too high, a shunt resistance may be added to a circuit to make sure a low excitation current flows through the sensor. This is in order to avoid any high Joule heating of the sensor.
-*Chopped source*: only a signal of a certain frequency is let through and a lock-in amplifier can amplify that signal to increase the signal-to-noise ratio. However, it also amplifies harmonics of that frequency.
-*ADC*, an analogue to digital converter, is needed to communicate with the computer. A n-bit ADC has a resolution of $\frac{1}{2^n}$.
-*BOBs*: The 300mK and 50mK stage were controlled by the Lakeshore 218 and AVS47 respectively using breakout boxes (BOBs). These BOBs can be used to read out pins of the same connection. Unfortunately the Lakeshore's excitation current is too high and raises the 50mK stage its temperature to 87mK and the 300mK stage to 370mK without any heater turned on. Instead we use the AVS for both stages, but not simultaneously. Just before the corona-quarantine, an extra AVS was obtained so all 3 stages could be controlled simultaneously. The output of this AVS was too low due to the internal resistances having been modified. This can be fixed by soldering the original resistances back.

2.4 Physics related to the TGSE

An NTC RTD works on the principle of how electrons move in semiconductors. At a temperature high enough for the electron to cross the band-gap into the conduction band, the resistance of the device drops and this continues for higher temperatures. PTC RTDs, work on the principle of a dominant rise of resistance due to an increase of the collisions of electrons with atoms, impurities or other electrons.

[1]Negative Temperature Coefficient, some controllers are not compatible with NTC RTDs, a problem we met but solved by changing a Lakeshore 330 to 230

The RRR value is the ratio of the resistivity at 300K to the resistivity at a designated low temperature, e.g. 4K. Blackbody radiation can be neglected if every emitter is cold enough.

How a dilution fridge works: ^{3}He is 25% lighter than ^{4}He. There is a van der Waals force between ^{3}He and ^{4}He which is stronger than between two ^{3}He atoms due to the ^{3}He being closer to ^{4}He as it is lighter and thus contains a larger volume. In a mixture of both ^{3}He and ^{4}He, the ^{3}He flows on top because it is lighter. Some ^{3}He may spontaneously *dilute* into the bottom ^{4}He layer due to an attractive force. At this physical location of dilution, anything can be hooked up to get cooled. The equilibrium concentration of ^{3}He into ^{4}He is 6.4% at absolute zero, at this % the chemical potential equals the latent heat of evaporation of pure ^{3}He. The fundamental reason is that ^{3}He atoms have to obey the Pauli Exclusion principle and therefore occupy increasingly higher energy states. The Fermi energy $k_B T_F$ will increase with the He concentration x, as shown in figure 2 [6]. The mixing chamber with the mixture is connected to a distiller ("still", dis- means apart and stillare comes from to drop in Latin) which evaporates and operates on the principle of different vapor pressures of ^{3}He and ^{4}He, much as how in a hot cup of water the top layer evaporates first.

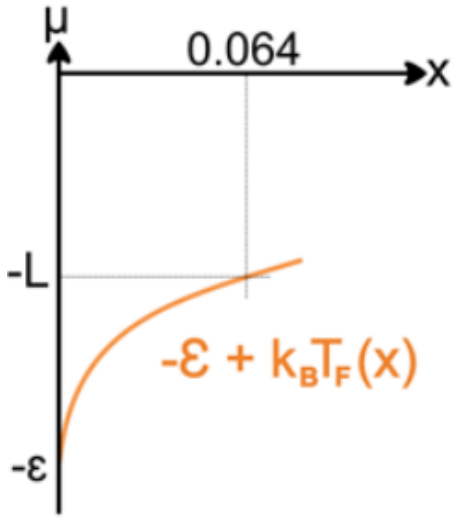


Figure 2: Chemical potential energy μ vs concentration x of ^{3}He. The first ^{3}He will occupy ε with antiparallel spins after which the levels fill up toward L, the latent energy.

2.5 Results of measurements

The conductance can be found from plotting ΔT versus the amount of power supplied. Its slope is the conductance. The conductivity is then obtained by entering the values for the length and surface area of the thermal straps as in table 1: $G = \sigma \frac{A}{l}$ where G is the conductance and σ the conductivity.

	l (mm)	A (mm)
2K	103.041	$\pi(1.5)^2$
300mK	150.436	$\pi(1.5)^2$
50mK	120.797	$\pi(2.5)^2$

Table 1: The length and surface area of the various straps connected to the three different stages.

The following plots were obtained:

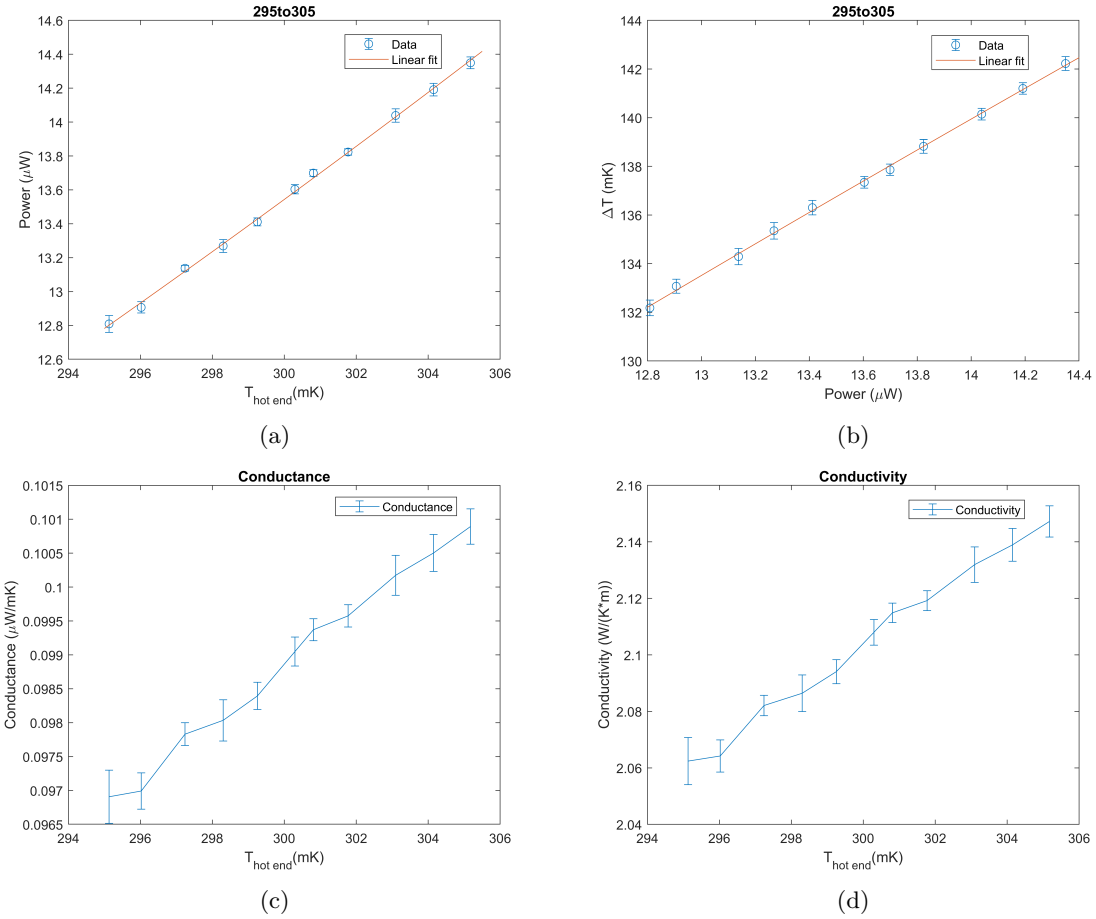


Figure 3: 300mK strap measurements. (a) Power needed to heat up the hot end of the strap to the desired temperature (b) The difference in temperature between the hot and cold end of the strap (c) The conductance (d) The conductivity. An excitation voltage of $300\mu V$ was used. Each data point (and error bar) consists out of 30 measurements.

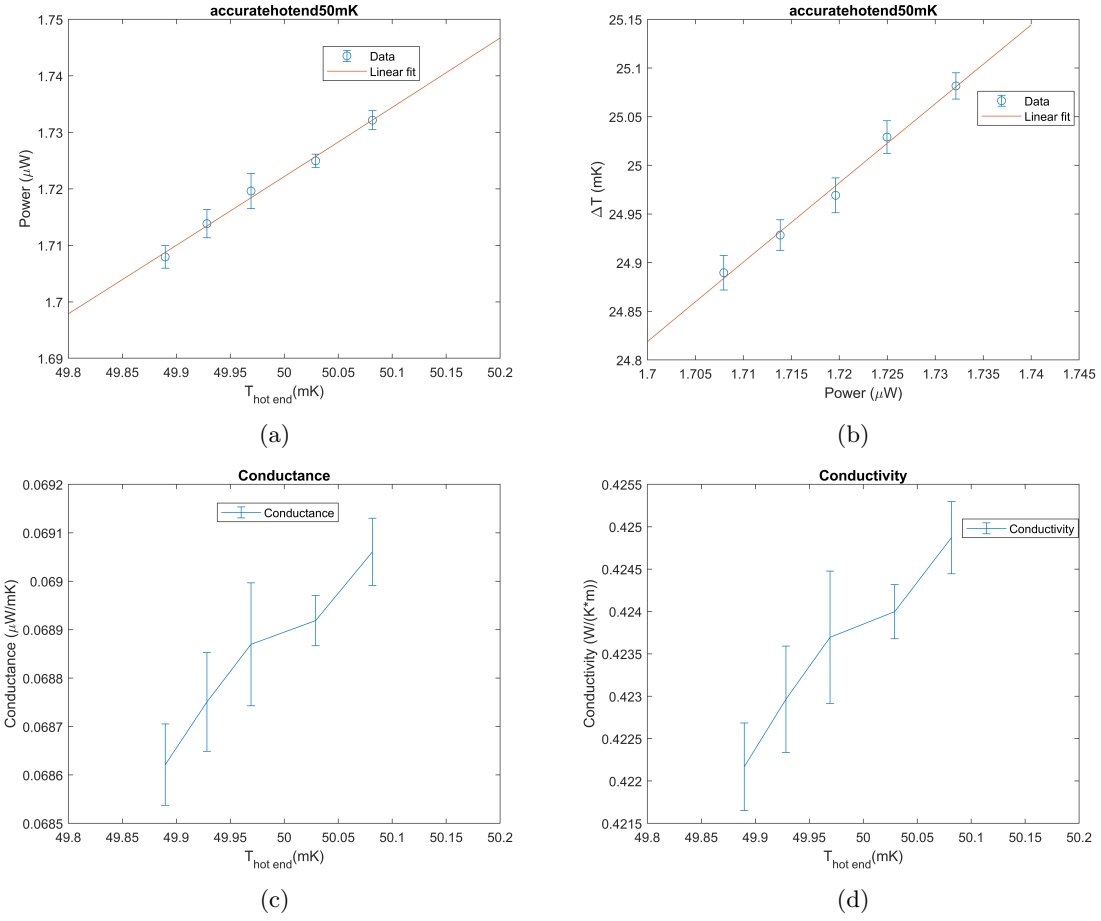


Figure 4: 50mK strap measurements. (a) Power needed to heat up the hot end of the strap to the desired temperature (b) The difference in temperature between the hot and cold end of the strap (c) The conductance (d) The conductivity. An excitation voltage of $300\mu V$ was used. Each data point (and error bar) consists out of 30 measurements.

For the 50mK plot, the stepsize as read by the Labview software was ≈0.02mK so reaching even a margin around the equilibrium of 0.05mK takes a lot of patience as the PID values should not converge to the new setpoint too swiftly in order to avoid overshoot.

The RRR value can be obtained by comparing the conductivity at 4.2K versus 300K. This was done by extrapolating the 300mK conductivity (figure 3) results using a linear equation. One obtains a value of $\approx \frac{370W}{K*m}$ at 4.2K. The room temperature conductivity for pure copper is $\approx \frac{400W}{K*m}$ [7]. This leaves a very low RRR value of $\frac{400}{370} = 1.08$. Reasons for this could be because of 1) the extrapolation error for the 300mK plot as it is far outside the datarange 2) because the room temperature conductivity differs from the theoretical value 3) due to a high contact resistance. Comparing our thermal conductivity at crygogenic temperatures with already known values [8] brings us closer to an RRR value of 50. This only further highlights that the estimate of the RRR-value itself is ill-defined.

The 2K stage was also regulated, but no conductivity tests were performed. The data gave a reading of $2 \pm 3.68 \cdot 10^{-4} K$ for a sample size of 2100 (1 per second). If one wants to measure the 300mK stage with the LS218, you will run into trouble: the excitation current of the LS218 is too high and raises the 50mK stage to 87mK and the 300mK stage to 370mK *without* any heating.

Error analysis

The error bars are calculated from the standard deviation of the averaged measurements. They give a measure of how stable the temperature could be held. This could be due Johnson-Nyquist (white) noise [9]: a fluctuation in the density of electrons which because of thermal agitation, leads to a fluctuation in voltage across the resistor which in turn makes the reading fluctuate. The equation for this noise is $P = 4k_BT\Delta f$ where T is the temperature and f is the frequency bandwidth, in our case 13.7Hz for the excitation frequency of the AVS-47B [10]. A quick look at figure 3 & 4 indeed shows that the errors increase with temperature. For a temperature of about 500mK, this noise is about $3.696 \cdot 10^{-22}W$ which is much smaller than the error bars. However, there could be a Johnson-noise contribution from the room-temperature readout electronics.

A self-heating current would not show up as an error bar, but merely give a false reading so this is also not the case.

Therefore, the only remaining source of fluctuation is that the PID has trouble regulating at these temperatures. The RMS resolution for the $300mK$ reading is higher ($\frac{STD(T)}{T} = \frac{0.6K}{132K} = 0.005$) than for $50mK$ ($\frac{0.05K}{25K} = 0.002$) when looking at the ΔT plots (figure 3 & 4). Looking into the manual of the AVSes, we indeed see the resolution is higher for higher resistances (higher resolution in this case is unwanted) as in figure 5 [10]. The 300mK resistor has a higher resistor (3287Ω) versus the 50mK resistor (47.099Ω).

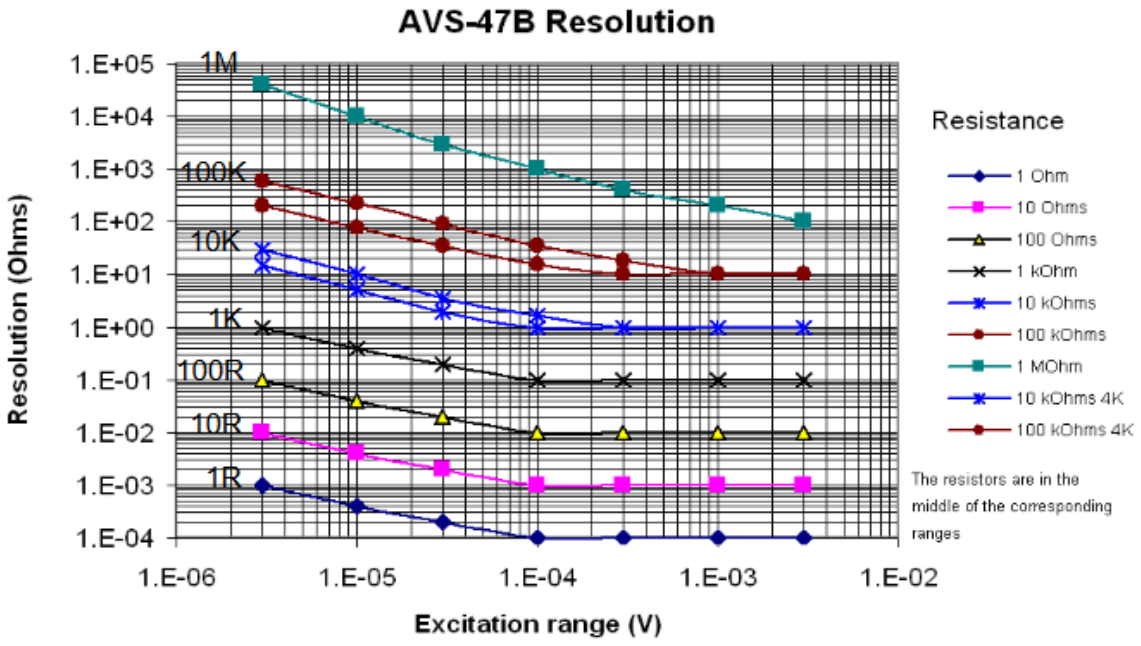


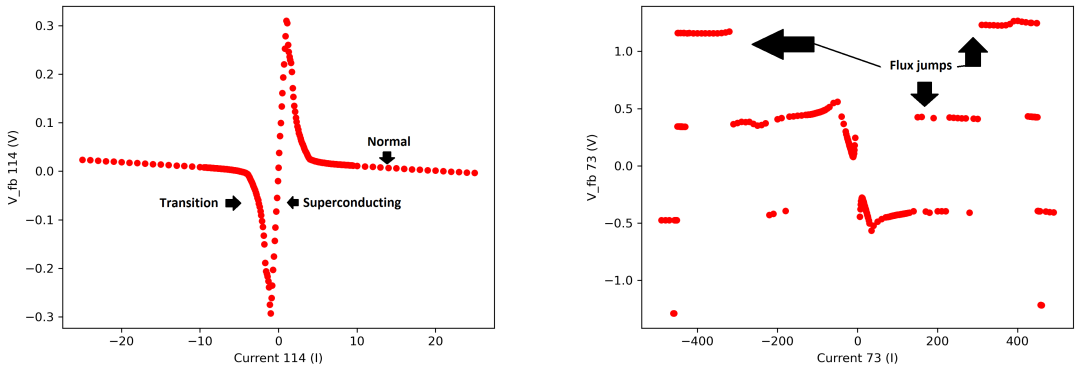
Figure 5: The resolution of the AVS47-B

3 Machine learning

3.1 Introduction

Machine learning is the discipline of teaching a machine how to learn in either of two ways: 1) supervised learning 2) unsupervised learning. 1) is about feeding a program a labeled set of data which the machine can use to classify, cluster or use linear regression. In classification, one can use the so called training dataset to classify new datasets in e.g. either 1 or 0 or more classes. These classes are determined by the features of that class. An example is a dataset that contains features of tumors, whether it be size, lump thickness, radiation penetration, redness etc. and then classifies it to be malignant or benign depending on these features. Clustering groups datapoint together of datasets with similar features, this in order to *make* classes. Consider it a prelude to classification. Linear regression is a method to fit the line through the dataset with the ultimate goal to predict future possible outcomes. Unfortunately, the more you fit according to your training dataset, the less room for any possible divergence from your plotted linear fit is possible, a case called overfitting (whereas underfitting is applying linear regression but not being as strict w.r.t. the error in the training dataset). 2) is about feeding a program data without labels. This is a lot more prone to errors and requires big datasets but saves you the trouble of having to find classes yourself or could be used to find structure/correlation in datasets that at first sight seem to contain random datapoints.

The assignment is to use classification, with predetermined classes, to classify IV curves into different (transition) behaviours. IV curves are plots which have current on the Y-axis and voltages on the X-axis and give information on how much resistance the that circuit path has. In regions of superconductivity for example, this resistance should drop to zero. These IV plots can help us characterize and calibrate the X-ray microcalorimeters. Since there are thousands of detectors in the array, identifying anomalies in the IV curves automatically would speed up the calibration. Manual analysis can be a real waste of manhours and bring in human error. The different classifications could be ohmic (linear), showing a normal superconducting transition, showing a flux jump, only showing one side of the transition etc. This automatic classification is useful for X-IFU. An example of two IV curves is shown in figure 6.



(a) (b)

Figure 6: On the left a "normal" IV curve which should only have the feature of having a normal superconducting transition. On the right a much more difficult plot to be classified. It exhibits a positive match for a lot of features (e.g. quantum flux and double transition) and can be classified into many classes.

A machine learning course was followed on `https://www.coursera.org/learn/machine-learning`

offered by Stanford University during this internship to get familiar with ML's concepts.

3.2 How we determine the error

The error is the most important concept in machine learning and defined as the difference between the predicted value (hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$ and the actual value, y, once an initial linear fit is made. It is formally incorporated into the so-called cost function $J(\theta)$ where the letters J is used as a convention, probably named after Jacobi. The cost function is defined as:

$$J(\theta_0, \theta_1) = \frac{1}{2n} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2) \tag{1}$$

The question is how we determine θ. We do this by minimizing the cost function (simply taking the derivative and equating to 0) to find the global minimum. This can be done iteratively using, for example, the gradient descent method, where one stats with some θ_0, θ_1 and changes these values to reduce J: $\theta_j := \theta_j - a\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ where is the learning rate a.k.a. the size of the the step down the gradient contour hill and := is an assignment operator. This repeats until convergence is reached.

In classification problems, we have to set a threshold for $h_\theta(x)$ e.g. for certain classes so that if it is in a certain range, we give it a certain value that corresponds to that class. If we have 4 classes, we could give it value n where $n \in \{1, 2, 3, 4\}$. An example of such a threshold function is the logistic/sigmoid function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

where $z = \theta^T x$. The function always returns a value between 0 and 1, so the output can easily be classified. We would have to train a h_θ^i for each class i. This is called the one-vs-all method. An example is shown below in figure 7. The axes show the features, and the line in the middle the hyperplane which seperates both classes. The margin length is defined as the distance perpendicular to the hyperplane to the nearest datapoint. This margin should be as big as possible to avoid classification errors.

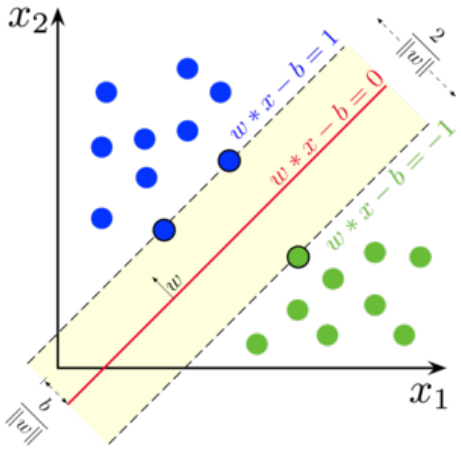


Figure 7: A hyperplane example where the 2 classes are colored differently and the axes show features x_1, x_2

3.3 Neural network or support vector machine?

In order to analyze our data, a Python script was written which extracts certain features such as mean, standard deviation, skewness, kurtosis and the gradient variants from the raw .qdp files. By

assigning each plot to a class manually (supervised) we can train a model. Initially, the assignment is to classify the plots obtained into 4 different classes:

1. If there is a flux jump
2. If there is a superconducting region
3. If there is a double transition (as opposed to a single transition)
4. If there is a linear region

In section 3.2, the workings of a support vector machine (SVM) were explained, that is the one-vs-all method which, which makes a linear distinction between part of class A or class B. Another option is to work with a neural network. A neural network works with nodes, similarly to how a brain works with neurons and the information passed between their synapses. It consists out of an input layer x_i, hidden layer(s) a_i and an output layer $h_\theta(x)$. The θ is again introduced, now as a matrix of weights controlling the mapping from layer j to layer $j+1$. The weights determine if the signal is put through to the next layer. We can model this using a linear regression formula: $h_\theta(x) = g(-30 + 20x_1 + 20x_2)$ where the weights are -30, 20, 20. If $x_1 \& x_2 = 1$, using the sigmoid function g(x) from equation 2, $h_\theta(x) \approx 1$ and the signal gets transmitted. An example is shown in figure 8 [11].

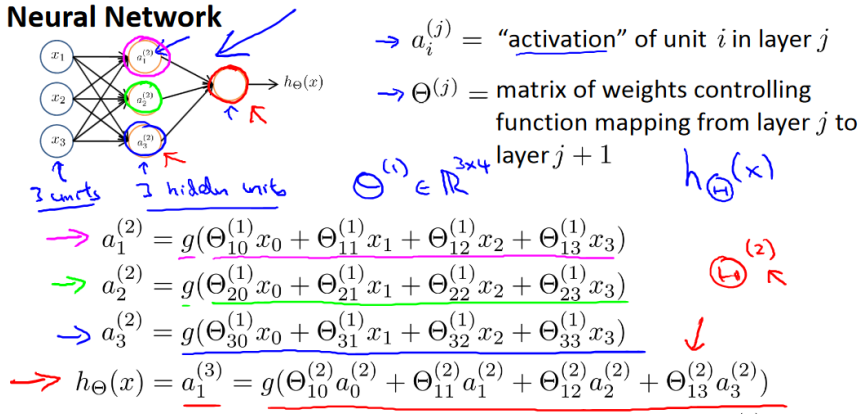


Figure 8: A neural network representation with comments from Andrew Ng

The advantage of a neural network over an SVM is that the neural network can work with a multi-class problem and calculate probabilities for each class while we would need 4 separate classifiers in the SVM method. Despite this, we still make 4 different neural networks as a multiclass classification divides the probability (=100%) over the number of classes you have instead of giving a probability that it belongs to that particular class.

3.4 Preprocessing

The hardest part of machine learning as an operation is getting your data in the correct shape. This is called preprocessing and involves feature-scaling, binning invalid datasets and most importantly labeling. Feature-scaling: it is wise to make sure all features are at least on the same order of magnitude. The convention in machine learning is that they should all be $0 \leq x_i \leq 1$ which can be achieved by e.g. normalizing: $x_i = \frac{x-\mu}{\sigma}$ for every feature x. The reason is that we want each feature to contribute equally to the solution of classification, not just features with a wide range of values. Invalid datasets could be presents due to a low number of datapoints or too scattered

datapoints. Labeling to train the dataset is the most time-intensive job to do after writing the neural network.

3.5 The neural network implementation

As my predecessor, Callum Blair, tried the SVM method (albeit with fewer features) we want to try and see if a neural network can work also (and do better). He got a validation rate of 85% meaning that the training set correctly classifies 85% of the plots.

A single hidden layer suffices for classes that can be neatly separated using a straight line. "Specifically, the universal approximation theorem states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units." [12] But this says nothing about the number of nodes present or the weights at that layer. In practice, people in the ML-discipline figure out how many nodes and layers are adequate by trial and error.

Instead of the sigmoid activation function, we use the Relu activation function as an adequate start (computationally cheap) [13] and finally output using the sigmoid function with a range between 0 and 1 (to classify positive or negative).

In training the network, the optimizer sweeps through different sets of θ_i^j (the weights, where j is the node number and i the layer) to find the ones that maximize the margin and minimize the derivative of equation 1.

For this we need a loss function and an optimizer. A loss function is for a single training example. It is also sometimes called an error function. A cost function, on the other hand, is the average loss over the entire training dataset. For binary problems, the loss function cross-entropy is recommended [14]. The optimizer is the algorithm used to travel down toward the global minimum such as via the gradient descent method. The norm is to use the "Adam" method, based on the gradient descent but with an added adaptive learning (step) rate α [13].

Next, the training process runs for a certain number of iterations called epochs, Greek for "stoppage" in time. The batch size argument is needed to let the model know after how many iterations the weights θ need to be updated.

Finally, we can verify the accuracy of this model by predicting what class a sample belongs to and comparing that to its label.

4 Conclusion

The attempt to experimentally test how a non-linear system of 3 PIDs operates has not yet been completed due to corona. However, it can still be done as all the equipment has been set up during this project and is ready for the test. A friend and fellow intern, Jip van Ham, is currently modelling this theoretically and one can use his results to see if theory matches up with practice.

Regarding the neural network; the end result is that there is no longer a need to search manually for specific traits of plots which makes analysing kilopixel detector arrays much easier. The user can add additional classification columns if he is interested in more traits of those plots such as "outliers" or "asymmetry". The accuracy for at least the superconducting transition and for the flux jump detection is 97.92% and 99.47% respectively for a testset of 0.2 * the total set and a trainingset of 0.8 * totalset, much higher than 85% for the SVM implementation of my predecessor. It has to be noted that these numbers can change due to the initial weights being initialized randomly and the order of experiments being shuffled for each training model (like shuffling a deck of card to avoid a static seed). The accuracy for the double transition is low (88.75%), due to the training data being rather ambiguous whether it shows a double transition or not. A second opinion on labeling this would help raise this accuracy. The linear region testing still has to be implemented, that is trying to classify with a "normal" region as in figure 6.

References

1. Barret, D. *et al.* The Hot and Energetic Universe: The X-ray Integral Field Unit (X-IFU) for Athena+. *arXiv:1308.6784 [astro-ph]*. arXiv: 1308.6784 (Aug. 2013).
2. *Cryogenic particle detection* (ed Enss, C.) *Topics in applied physics* **v. 99.** OCLC: ocm60800561 (Springer, Berlin ; New York, 2005).
3. Constortium, X.-I. *http://x-ifu.irap.omp.eu/* en-GB. Library Catalog: x-ifu.irap.omp.eu.
4. Janesch, J. *EDN - Two-wire vs. four-wire resistance measurements* en-US. Apr. 2013.
5. Tomas, B. *Ziegler-Nichols Method* Feb. 2004.
6. Batey, G. & Teleberg, G. *Principles of dilution refrigeration* 2015.
7. Kasap, S., Málek, J. & Svoboda, R. en. in *Springer Handbook of Electronic and Photonic Materials* (eds Kasap, S. & Capper, P.) 1–1 (Springer International Publishing, Cham, 2017).
8. Thompson, C., Manganaro, W. & Fickett, F. *Cryogenic Properties of Copper* July 1990.
9. Nyquist, H. Thermal Agitation of Electric Charge in Conductors. *Physical Review* **32.** Publisher: American Physical Society, 110–113 (July 1928).
10. Picowatt, R.-E. O. *AVS-47B AC Resistance Bridge* Feb. 2020.
11. Ng, A. *Week 4: Models and representation* Stanford, May 2020.
12. Goodfellow, I., Bengio, Y. & Courville, A. *Deep learning* (The MIT Press, Cambridge, Massachusetts, 2016).
13. Bushaev, V. *Adam — latest trends in deep learning optimization.* en. Library Catalog: towardsdatascience.com. Oct. 2018.
14. Brownlee, J. *How to Choose Loss Functions When Training Deep Learning Neural Networks* en-US. Library Catalog: machinelearningmastery.com. Jan. 2019.

A Neural network scripts

A.1 read_qdp.py

The read_qdp.py function was made more easy to understand and code was added to unskew plots, extract features such as gradient and kurtosis. This code shown below:

```
import pandas as pd
import re
from scipy.stats import describe
import numpy as np


def read_qdp(filelocation, iandv, features, expnr, filelocationlist,
↪   make_excel=False):
    """
    Accepts a raw sting literal for the filename and path.
    x and y are strings which name the first 2 columns, default to bias
    and V fb (V)
    If make_excel is set to True then excel file is created in the name
    of the input file.
    Return the a pandas dataframe with the data.
    """

    # name first two cols
    x = 'Current ' + str(expnr + 1) + ' (I)'
    y = 'V_fb ' + str(expnr + 1) + ' (V)'
    cols = {0: x, 1: y}
    header = 0

    # skip amount of rows until a number is found (which is valid data)
    with open(filelocation) as file:
        for line in file:
            if line[0].isdigit():
```

```
            break
        header += 1
df = pd.read_csv(filelocation, sep='\\s+', header=None, skiprows=header,
↪  index_col=False, comment='!')

# make list of col names for df and number ones without labels
col_names = []
for i in range(len(df.columns)):
    try:
        col_names.append(cols[i])
    except:  # way too broad exception... again ask Callum
        col_names.append(str(i))
    i += 1

# set column names and sort by x vals
df.columns = col_names
df.sort_values(by=[x], inplace=True)

# no idea what the purpose of this is, ask Callum
if df.iloc[:, 0][1] * df.iloc[:, 1][1] < 0:
    df.iloc[:, 1] *= -1

# make excel file
if make_excel:
    # find name for excel sheet from end of path
    name_match = re.search(r'(\\?((?:.(?!\\))+)$)', filelocation)  # also no
    ↪  clue what this does, ask Callum
    name = re.sub(r'(\.qdp|\\)', r'', name_match.group(1))
    # write to file and save
    writer = pd.ExcelWriter(str(name) + '.xlsx')
    df.to_excel(writer, 'Sheet1', index=False)
    writer.save()

# obtain IV columns and select features
firsttwocol = df.iloc[:, [0, 1]].dropna()

# check if any of the two columns are empty and check for enough data points,
↪  if not obtain data
if not firsttwocol.empty and len(firsttwocol.index) > 100:
    # make sure the plots aren't skewed, take mean of first and last n points
    ↪  to draw slope
    n = 10
    a = (np.mean(firsttwocol.iloc[(2 * -n):-n, 1]) -
    ↪  np.mean(firsttwocol.iloc[n:(2 * n), 1])) \
        / (np.mean(firsttwocol.iloc[(2 * -n):-n, 0]) -
        ↪  np.mean(firsttwocol.iloc[n:(2 * n), 0]))
    b = firsttwocol.iloc[:, 1] - a * firsttwocol.iloc[:, 0]
    firsttwocol.iloc[:, 1] = firsttwocol.iloc[:, 1] - a * firsttwocol.iloc[:,
    ↪  0] + b

    # obtain data
    iandv = pd.concat([iandv, firsttwocol], axis=1)
```

```
        stats = describe(firsttwocol.iloc[:, 1])
        gradient = np.gradient(firsttwocol.iloc[:, 1])
        statsgrad = describe(gradient)
        features = features.append([(stats.mean, stats.variance, stats.skewness,
        ↪  stats.kurtosis, statsgrad.mean,
                                     statsgrad.variance, statsgrad.skewness,
                                     ↪  statsgrad.kurtosis)])

        # up the counter
        expnr += 1

        # valid filelocations
        filelocationlist.append(filelocation)

    return df, iandv, features, expnr, filelocationlist
```

A.2 trainingdataobtainer.py

The trainingdata.py code was edited and code was added to foresee for any potential I/O errors, to obtain a filelocation list, feature-scale and plot all IV curves. This code is shown below:

```
import os
import pandas as pd
from read_qdp import read_qdp
import matplotlib.pyplot as plt
from pathlib import Path

# initialise variables
rootdir = r'C:\Users\arnol\Desktop\newproject\SAFARI TES data\Selection'
Path(rootdir + r"\Plots").mkdir(parents=True, exist_ok=True)
save_plots_to = rootdir + r'\Plots'
filelocationlist = []

extensions = '.qdp'
iandv = pd.DataFrame()
features = pd.DataFrame()
expnr = 1
errorcount = 0

# extract df, iandv, features, expnr from .qdp files
for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        if file[0] == 'I':  # if starting with I to make sure other files are
        ↪  excluded
            ext = os.path.splitext(file)[-1].lower()
            if ext == '.qdp':
                filelocation = os.path.join(subdir, file)
                # if file cannot be read, display error
                try:
```

```
                df, iandv, features, expnr, filelocationlist =
                ↪  read_qdp(filelocation, iandv, features, expnr,
                ↪  filelocationlist, make_excel=False)
            except (pd.errors.ParserError, pd.errors.EmptyDataError):
                errorcount += 1
                print(filelocation)

print(
    "The above " + str(errorcount) + " file(s) were invalid and skipped, please
    ↪  check formats manually. Continuing...")

# output filelocationlist.txt
filelocationlistfile = open('filelocationlist.txt', 'w')
for filelocation in filelocationlist:
    filelocationlistfile.write(filelocation)
    filelocationlistfile.write('\n')
filelocationlistfile.close()

# convert to .xlsx files
iandv.to_excel('iandv.xlsx', index=False)
features.columns = ['mean', 'std', 'skewness', 'kurtosis', 'grad mean', 'grad
↪  std', 'grad skewness', 'grad kurtosis']
features = (features - features.mean()) / features.std()
features.to_excel('features.xlsx', index=False)

# plot where the i + 1 means you can more easily compare rows in .xlsx files with
↪  plots as headers are in row 1
for i in range(1, expnr):
    iandv.plot(kind='scatter', x='Current ' + str(i + 1) + ' (I)', y='V_fb ' +
    ↪  str(i + 1) + ' (V)', color='red')
    plt.savefig(save_plots_to + r'\row ' + str(i + 1) + '.png', dpi=300)
    plt.close('all')
```

A.3 NNtrainer.py

A neural network NNtrainer was written from scratch to make models for each column after the features in featureclassification.xlsx. A confusion matrix, verification based on 80% training data and 20% test data and the models are its outputs. The code:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import os

rootfolder = r'C:\Users\arnol\Desktop\newproject'
Y, Y_train, Y_test = {}, {}, {}
begin_column_feature = 0            # remember python starts counting at 0
end_column_feature = 8
```

```
begin_column_label = 8
end_column_label = 11                # change this line to add more classes

# load the dataset
dataset = pd.read_excel(os.path.join(rootfolder, r'featureclassification.xlsx'))

# split into input (X) and output (Y) variables
X = dataset.iloc[:, :end_column_feature]  # Up to and not including column 8
for i in range(begin_column_label, end_column_label):  # label columns
    Y[i] = dataset.iloc[:, i]

    # split in training and testset
    X_train, X_test, Y_train[i], Y_test[i] = train_test_split(X, Y[i],
    ↪  test_size=0.2)

    # define the keras model
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # compile the keras model
    model.compile(loss='binary_crossentropy', optimizer='adam',
    ↪  metrics=['accuracy'])

    # fit the keras model on the training dataset
    model.fit(X_train, Y_train[i], epochs=150, batch_size=10)

    # make class predictions with the model for data X_test
    predictions = model.predict_classes(X_test)

    # give accuracy for all data input cases
    for j in range((len(X_test))):
        print('%s => %d (expected %d)' % (X_test.iloc[j].tolist(),
        ↪  predictions[j], Y_test[i].iloc[j]))

    # print confusion matrix
    print("Confusion matrix:\n", confusion_matrix(Y_test[i], predictions,
    ↪  labels=None, sample_weight=None,
                                                    normalize=None))
    # save models
    model.save('model ' + dataset.columns[i])

print("Saved models to disk")
```

A.4 tester.py

Finally, tester.py was written from scratch to test the models on new data, find the positive hits (which ones belong to that class) and copy them to folders so these files can be altered later. The code:

```
# load and evaluate a saved model
import os

from keras.models import load_model
import pandas as pd
import shutil

rootfolder = r'C:\Users\arnol\Desktop\newproject'

begin_column_feature = 0  # remember python starts counting at 0
end_column_feature = 8

begin_column_label = 8
end_column_label = 11  # change this line to add more classes

positivenrlist = []
qspname = open('filelocationlist.txt', 'r')
qsp_data = qspname.readlines()

for i in range(begin_column_label, end_column_label):

    # load dataset
    dataset_uncut = pd.read_excel(os.path.join(rootfolder,
    ↪  r'featureclassification.xlsx'))
    dataset = dataset_uncut.iloc[:, :end_column_feature]

    # load model
    model = load_model('model ' + dataset_uncut.columns[i])

    # evaluate the model
    predictions = model.predict_classes(dataset)

    for j in range((len(dataset))):
        if predictions[j] == 1:
            positivenrlist.append(str(j + 2))  # + 2 so 0 becomes 2 to correspond
            ↪  to features.xlsx file rows
    print("The following rows were tested positive for " +
    ↪  dataset_uncut.columns[i]
          + ': ' + ', '.join(positivenrlist))
    print("The corresponding .qsp filenames are stored in Positive " +
    ↪  dataset_uncut.columns[i] +
          ' and copied to the Positives folder.')

    positivenamelist = open('Positive ' + dataset_uncut.columns[i] + '.txt',
    ↪  'w+')
    for number in positivenrlist:
        positivenamelist.write(qsp_data[int(number) - 2])  # - 2 to account for
        ↪  previous + 2
    positivenamelist.close()

    # copy all the .qsp files that were tested positive to their designated
    ↪  folders
```

```
    target = os.path.join(rootfolder, r'SAFARI TES data\Selection\positives',
    ↪  dataset_uncut.columns[i])
    if not os.path.exists(target):
        os.mkdir(target)

    positivenamelist = open('Positive ' + dataset_uncut.columns[i] + '.txt',
    ↪  'r+')
    for line in positivenamelist.readlines():
        shutil.copy(line.strip(), target)
    positivenamelist.close()

    positivenrlist.clear()

qspname.close()
```