Revision 0.2 September 14, 2020

GRADIOMETER USER MANUAL

Cole Bunch

TUCAN nEDM Magnetics

TRIUMF

SUMMARY

The TUCAN Magnetics gradiometer has existed in various forms and levels of functionality since 2017. It was built to support the nEDM experiment at TRIUMF. All reports on the gradiometer can be found on the UCN Plone, but specific documents will be referenced when relevant.

The gradiometer is a device for measuring the permanent magnetization of samples. It uses fluxgate magnetometers to measure magnetic flux, with a magnetic shield to reduce flux from external sources. Components that will be used in the extremely magnetically sensitive nEDM experiment can be tested in the gradiometer for residual magnetizations.

This manual describes all the components of the gradiometer and how to set them up. It is also a reference for the code that controls the gradiometer's functions. This manual can be updated as the gradiometer is improved and modified.

Many thanks to Beatrice Franke for supervising the gradiometer project, and to Matthew Palmer, Josh Smith, Elspeth Cudmore, Nadia Chigaroff, Matthew Wilson, and Fabian Piermaier for their work in designing and building the gradiometer.

Setup

This section will cover:

1. The components of the gradiometer, their function, and how to connect them.
2. How to connect the Raspberry Pi to TRIUMF Secure WiFi and operate it in the headless configuration.

COMPONENTS

Shield

The bulk of the gradiometer is an aluminum t-slot frame cradling an open ended cylindrical mu-metal shield. The shield consists of two concentric sheets of mu metal rolled into cylinders and welded closed, with diameters of 20cm and 22.6cm. Additional flexible mu-metal material was placed in the gap between these two cylinders. Information on the original design of the gradiometer by Mathew Palmer can be found here.

The shield is 74.5cm long, but because of magnetic flux ”leaking” in through the open end caps, only the middle 30cm should be considered well shielded from external fields. Information on the shielding factors along the length of the shield can be found in Figs 11,12,13, and 14.

Fluxgates

Measurements are made by two Bartington Mag-03MSL100 fluxgates. The fluxgates measure magnetic flux on three perpendicular axis in the range $\pm 100\mu\mathrm{T}$ and output a voltage of $\pm 10\mathrm{V}$. This means that voltages need to be multiplied by a factor of 10 to convert between V and $\mu\mathrm{T}$.

Note

The current measurement scripts measure V, not μT, so you will need to multiply your data by 10 to convert to μT in your analysis.

Another important feature of the fluxgates is that, unlike some other Bartington fluxgates, the three measurement axis are offset as seen in Fig 1. The z-axis is offset by 1.5cm from the y-axis and the x-axis is offset by 1.5cm from the z-axis. This will be important when we consider the position of each measurement in the gradiometer. For example, if we make a measurement where the position of the y measurement axis is $pos(y) = p$, then in our analysis we must remember that $pos(z) = p - 1.5$ and $pos(x) = p - 3$.

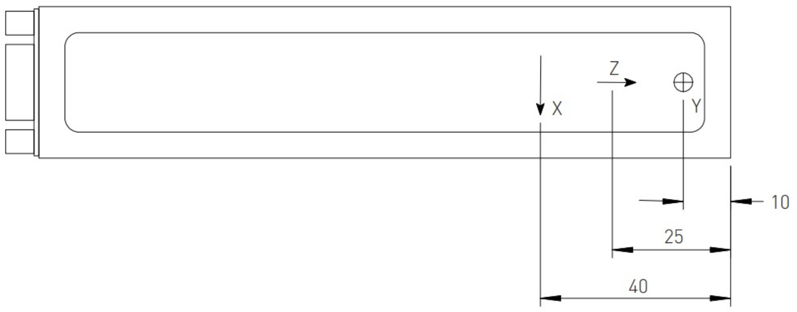


Figure 1: Drawing of Bartington Mag-03MSL100 fluxgate.

This is perhaps more evident in Fig 2 which shows a background measurement taken along the length of the gradiometer. Here we see that the zero reference position was taken to be the starting point of the y measurement axis, shown in orange. Note that the green and blue lines are offset from the orange line, representing how the starting position of the z-axis and x-axis were 1.5cm and 3cm behind the y-axis.

A full schematic of the fluxgates including a pin-out of the DB9 connector can be found in the appendix.

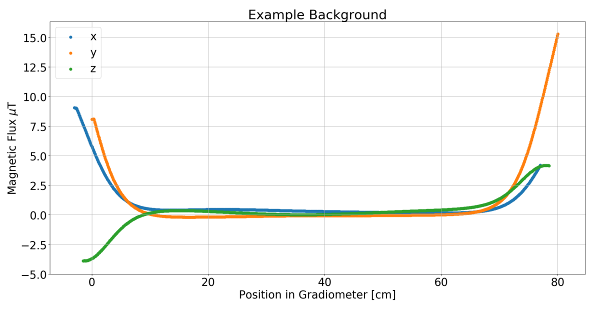


Figure 2: Sample gradiometer background measurement demonstrating measurement axis offset.

The specs of the fluxgate can be found here.

DAQ Unit

The data aquisition (DAQ) unit designed by Elspeth Cudmore provides ±12V power to the fluxgates as well as active filtering for the raw fluxgate signal before it proceeds to the ADC. The fluxgates are connected to the 10-pin Hirose sockets via special DB9 to 10-pin Hirose cables. The pin-out of the Hirose connectors are shown in Fig 6.

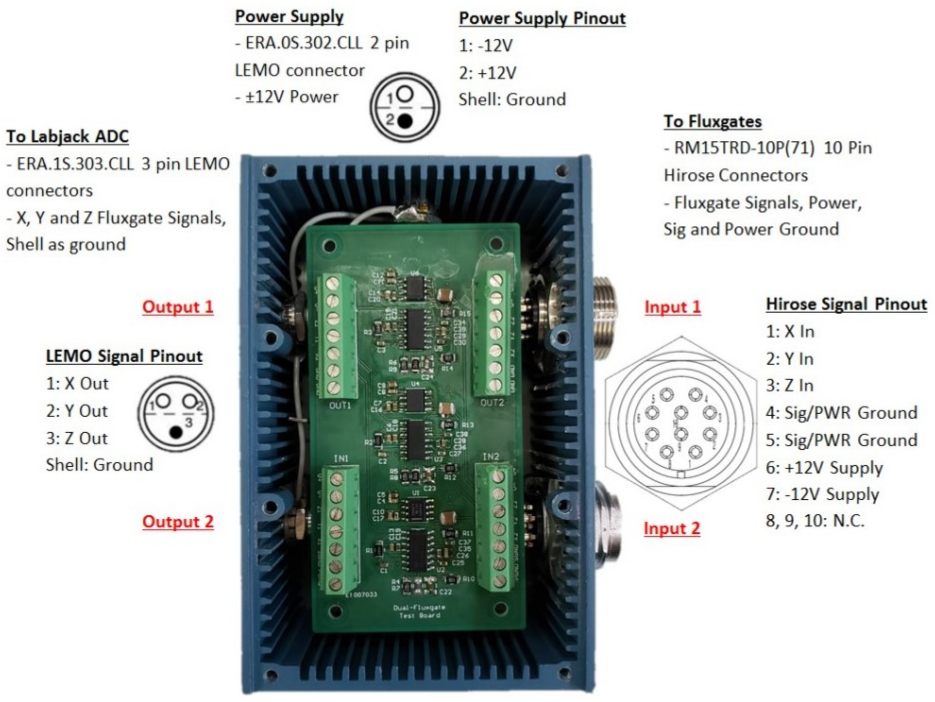


Figure 3: Pin-outs of PSU/Filtering inputs, outputs, and power.

talk about floating ground Power is provided by two 12V AC to DC adapters with the high of one adapter connected to the low of the other adapter resulting in the remaining two leads having ±12V with respect to the connected leads. The connected leads are the reference ground voltage for the fluxgates, integrate circuit, and the ADC inputs. The leads are connected to the DAQ via a 2 pin LEMO connector which is shown in Fig 6.

Note

If troubleshooting the PCB, note that Output 1 is mislabeled as Output 2, and vice versa.

Filtering is done using an integrated circuit that includes instrumentation amplification and a low pass filter for each of the six signals (2 fluxgates, 3 axis each). The low pass filter cut-off is at 10Hz, and has 5th order rolloff above the cut-off. Because the gradiometer is designed to measure objects with static, permanent magnetizations, this frequency cut-off is appropriate and greatly reduces signal noise.

If the DAQ does not seem to be functioning properly, connections should be checked using a multimeter and the diagrams of the amplification circuit and filtering circuit in the appendix. Further diagrams and more design information can be found in Elspeth's full report here.

Labjack ADC

To convert the analogue voltage from the fluxgates to a digital signal, a Labjack U6 Pro with an Expansion Screw Terminal Board is used. The Labjack is powered by the Raspberry Pi via a USB connection.

The 3-pin LEMO from the DAQ is connected to the screw terminal via lead wires. The black, green, and white leads correspond to the x, y, and z signals respectively, and the ground is a group of bare silver wires from the co-axial cord shielding. Fluxgate 1 should be connected with x in AIN0, y in AIN1, z in AIN2, and the ground in GND. Fluxgate 2 should be connected with x in AIN3, y in AIN4, z in AIN5, and the ground in GND.

The capabilities and uses of the Labjack as it pertains to the gradiometer are discussed in the Control Code section. The full datasheet of the Labjack U6 Pro can be found here.

Raspberry Pi

The gradiometer is controlled by a Raspberry Pi 3 running the Raspbian operating system. The RPi is powered with a 5V AC to DC adapter connected via the RPi's Micro USB port. The RPi is equipped with HDMI for a display, USB for a mouse and keyboard, MicroSD for storage, and WiFi/Ethernet for internet connectivity. A mouse, keyboard, and display should be available in the Magnetics Lab for use with the gradiometer.

Connected to the RPi's GPIO pins is a Adafruit DC & Stepper Motor HAT which is powered by an additional 12V AC to DC adapter.

Information on how to setup the RPi is in the RPi Setup section, and information on the gradiometer code on the RPi is in the Control Code section.

Stepper Motor and Drive Train

To facilitate quick and repeatable measurement of a sample, the gradiometer is outfitted with a system to automatically move the fluxgate magnetometer. This is achieved using the Adafruit Motor HAT mentioned above, a Trinamic Motion QSH4218-51-10-049 stepper motor, a series of timing belts and gears, and a fluxgate carriage. The advantage of using a stepper motor is that the motor only turns in discrete steps and error in rotation does not accumulate over many steps. This provides superior fluxgate positional accuracy.

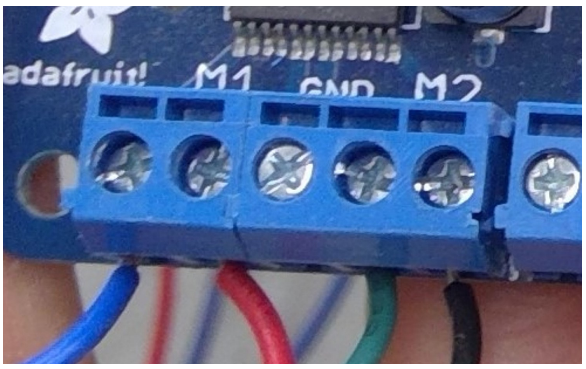


Figure 4: Leads of Trinamic stepper motor.

The motor is attached to a timing belt that turns a shaft, which turns a larger timing belt. The fluxgate carriage is attached to the larger timing belt with tape. The Trinamic stepper motor completes a full rotation after 200 steps, each step moving the fluxgate $\sim$0.8mm (see Calibration). The Adafruit Motor HAT and the Trinamic motor have the capability of "microstepping," dividing the rotation into even smaller steps, but this reduces motor torque. Because the fluxgate carriage merely slides over the gradiometer frame, the highest possible torque is required to overcome friction and prevent the motor from stalling. Should more granular positional resolution become necessary, the fluxgate carriage will have to be upgraded with wheels or some other mechanism with less friction before microstepping is used.

The Trinamic motor is connected to the Motor HAT via four coloured leads, two for each coil in the motor.

These pairs need to be connected to the M1 and M2 screw terminals on the Motor HAT, with blue/red connected to M1 and green/black to M2 as shown in Fig 4. This order should not be changed unless you are sure you know what the effects will be and have made the appropriate changes to the gradiometer code.

The full specifications of the Trinamic Motion QSH4218-51-10-049 stepper motor can be found here. Information from Adafruit about the operation of the Motor HAT with a stepper motor can be found here.

SETTING UP THE RASPBERRY PI

The simplest way to operate the gradiometer is like any computer: with mouse, keyboard, and monitor directly connected to the RPi. While the gradiometer could be entirely operated in this manner, there are benefits of connected it to WiFi and operating it remotely such as easy transfer of measurement data and code updates, or being able to move the table the gradiometer is on to reduce backgrounds.

Connecting to TRIUMF Secure

The TRIUMF Secure network uses the WPA2-Enterprise security protocol, which is more secure than a usual home WiFi connection. Unfortunately, the distribution of Linux on the RPi (Raspbian) cannot connect to WPA2-Enterprise networks with the user interface normally used for WiFi connections. The network has to be manually added to the RPi's list of known networks along with the user's credentials. If the network is not already configured, the mouse, keyboard, and display must be used for the initial setup.

The list of known networks is in a file called `wpa_supplicant.conf`, which can be edited using the following command in the Raspbian terminal:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

The contents of the file should contain the various parameters to configure the TRIUMF Secure connection.

```
network={
    ssid="TRIUMF SECURE"
    scan_ssid=1
    key_mgmt=WPA-EAP
    eap=PEAP
    identity="YOUR_USERNAME"
    password="YOUR_PASSWORD"
    phase1="peaplabel=0"
    phase2="auth=MSCHAPV2"
}
```

In the identity field, input your TRIUMF Trident username. To avoid storing your password in plain text on the RPi, your password should be hashed using the following terminal command:

```
echo -n "YOUR_PASSWORD" | iconv -t utf16le | openssl md4
```

This will output a string of alphanumeric characters which can be copy and pasted into the file with the prefix `hash:` as such:

```
password=hash:6602f435f01b9173889a8d3b9bdcfd0b
```

With your login information entered, use ctrl + x, Enter, Enter to save the file. The RPi should be connected to TRIUMF Secure.

SSH to the Raspberry Pi

With the RPi connected to TRIUMF Secure, it can be easily accessed from your own computer while you are also connected to TRIUMF Secure using the command line or an ssh client (like Mobaxterm for Windows or Terminus for Mac) with the command

```
ssh pi@gradiometer
```

with the password `gradiometer`. If the host `gradiometer` is not found, use an external display to find the RPi's IP address by hovering a mouse over the WiFi connection symbol at the top right of the screen. Then use:

```
ssh pi@142.90.68.101 ##replace with actual IP address
```

At the start of the ssh session, a graphic is displayed by Raspbian followed by the command line that allows you to control the gradiometer. Use the command `exit` to end the ssh connection.

GitHub Repository

The folder that contains all the gradiometer control code and collected data is `/home/gradiometer/` and is associated with a GitHub repository. Storing the code in this way makes it convenient to edit code on another machine using your preferred IDE, and then push your updates to the gradiometer, which has only basic text editors.

There is a log of the current repository URL in the Logs, as well as contact information for the owner of the repository if there is access problems.

Alternatively, you can create your own repository in GitHub and change the origin URL on the gradiometer using the command:

```
git remote set-url origin YOUR_URL_HERE
```

Then push all the files to your new repository using the commands:

```
git add -A
git push
```

If you have never used git before, the basic workflow is use

```
git pull
```

to get the latest files from the server on your local machine (the gradiometer or your computer), then edit files or produce data, then use the commands

```
git add -A
git commit -m "write a little message about the update here"
git push
```

to send updates or data to the server. When using `push` and `pull`, you will be asked to provide a passphrase, which is simply `gradiometer`.

Control Code

The library of code controlling the gradiometer has gone through many iterations expanding functionality. The current code was written by Cole Bunch and the GUI by Xander Naumenko, heavily borrowing from Nadia Chigmaroff and Matthew Wilson.

Nadia was responsible for transitioning the code to an object-oriented approach. As we will see later, using objects is convenient for issuing commands to the gradiometer on the fly, as well as scripting specific measurement sequences. It is also useful for creating live plots and (in the future) GUIs.

Nadia's code operating using multithreading, where one thread continuously moved the motor back and forth and the other read continuous measurements from the Labjack in a time series. While the position of the fluxgate can be inferred by multiplying the time stamp of each measurement by the average velocity of the fluxgate cart, this does not take advantage of the discrete steps of the stepper motor or allow for measuring in specific locations. For this reason, the multithreading was removed and the code now single steps the motor and makes a measurement in an alternating sequence on the same thread. Nadia and Matthew had implemented live plotting using pyqt, but that has since been replaced by the new GUI due to the different code bases.

Nadia and Matthew's unchanged code is still available on the RPi in `/home/gradiometer/` in the files `LiveGraph.py`, `LiveGraph2.py`, and `vel_test.py`. Nadia's full report on the design of the object-oriented, multithreaded code can be found here and Mathew's report with some updates and further work can be found here.

The following sections will cover:

1. The graphical user interface and how to use it
2. The basic structure of the code and what each function does.
3. Some important information on how to calibrate the gradiometer.
4. How to combine functions into a measurement script.

GRAPHICAL USER INTERFACE (GUI)

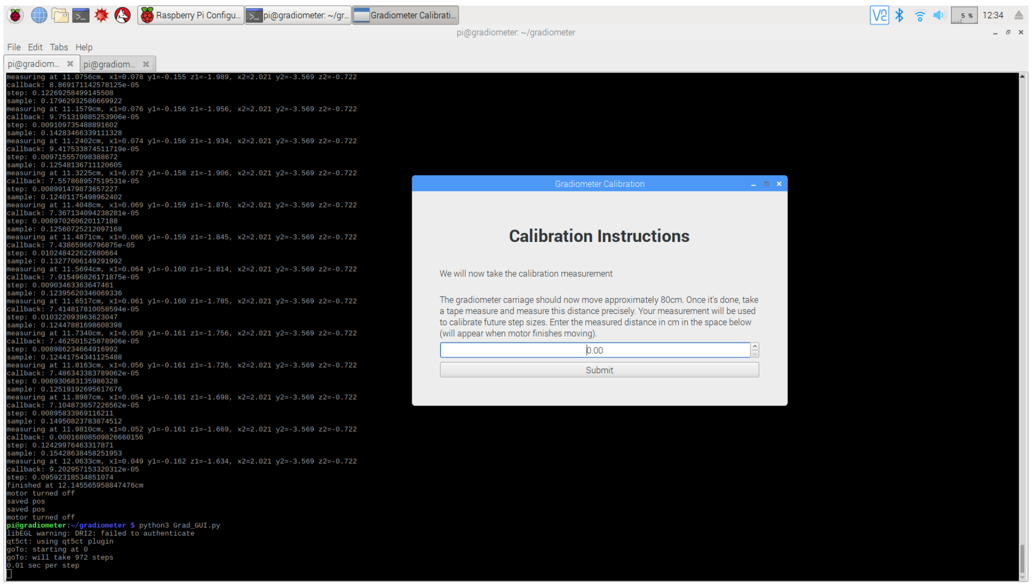
All of the code described below has been wrapped in a GUI that has (almost) identical functionality to using a python scripting approach. The benefit of having such a GUI is that it's easier to learn to use, as well as having live graphs while mapping to make sure nothing is going wrong as the user is taking data.

All of the code for the GUI is in `/home/pi/gradiometer/Grad_GUI.py`. To use it, go to the directory in a terminal and execute the python file by running the command `python3 Grad_GUI.py`. The GUI should pop up a prompt for the mode to run it in. There are three modes that the GUI can be used in: calibration, position run and time run. The first is to calibrate the stepper motor step size, while the latter two correspond to the two measurement methods described in detail below.

The calibration mode has instructions embedded in the GUI for what to do, so follow them and the calibration should be done automatically. For more information on what exactly it is doing, see the subsection below on calibration.

For the two measurement modes, the functionality of the GUI exactly mirrors that of the original python functions. For information about what all the parameters mean, see the `posRun` and `timeRun` documentation below in the Code Basics section. Once all the paramters have been set on the left side of the screen, you can click the begin run button to start live graphing the mapping on the screen. Note that you can take multiple measurements within the same session of the GUI by adjusting parameters and starting another run after the first, and both measurements will be overlayed on each other to give a better understanding of the underlying behaviour of the magnetic field.

In terms of the development of the GUI itself, the python module pyqt is used. For the graphing, Matplotlib was originally used, but due to performance constraints (it is running on a Rasberry Pi which is obviously not a powerful device by any stretch), the graphing was switched to pyqtgraph. This offers better performance for live graphing which suits the requirements.



```
measuring at 11.0756cm, x1=0.078 y1=-0.155 z1=-1.989, x2=2.021 y2=-3.569 z2=-0.722
callback: 8.869171142578125e-05
step: 0.12269258499145508
sample: 0.17962932586669922
measuring at 11.1579cm, x1=0.076 y1=-0.156 z1=-1.956, x2=2.021 y2=-3.569 z2=-0.722
callback: 9.751319885253906e-05
step: 0.009109735488891602
sample: 0.14283466339111328
measuring at 11.2402cm, x1=0.074 y1=-0.156 z1=-1.934, x2=2.021 y2=-3.569 z2=-0.722
callback: 9.417533874511719e-05
step: 0.009715557098388672
sample: 0.12548136711120605
measuring at 11.3225cm, x1=0.072 y1=-0.158 z1=-1.906, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.557868957519531e-05
step: 0.008991479873657227
sample: 0.12401175498962402
measuring at 11.4048cm, x1=0.069 y1=-0.159 z1=-1.876, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.367134094238281e-05
step: 0.008970260620117188
sample: 0.12560725212097168
measuring at 11.4871cm, x1=0.066 y1=-0.159 z1=-1.845, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.43865966796875e-05
step: 0.010248422622680664
sample: 0.13277006149291992
measuring at 11.5694cm, x1=0.064 y1=-0.160 z1=-1.814, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.915496826171875e-05
step: 0.00903463363647461
sample: 0.12395620346069336
measuring at 11.6517cm, x1=0.061 y1=-0.160 z1=-1.785, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.414817810058594e-05
step: 0.010322093963623047
sample: 0.12447881698608398
measuring at 11.7340cm, x1=0.058 y1=-0.161 z1=-1.756, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.462501525878906e-05
step: 0.008986234664916992
sample: 0.12441754341125488
measuring at 11.8163cm, x1=0.056 y1=-0.161 z1=-1.726, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.486343383789062e-05
step: 0.008930683135986328
sample: 0.12519192695617676
measuring at 11.8987cm, x1=0.054 y1=-0.161 z1=-1.698, x2=2.021 y2=-3.569 z2=-0.722
callback: 7.104873657226562e-05
step: 0.00895833969116211
sample: 0.14950823783874512
measuring at 11.9810cm, x1=0.052 y1=-0.161 z1=-1.669, x2=2.021 y2=-3.569 z2=-0.722
callback: 0.00016808509826660156
step: 0.12429976463317871
sample: 0.15428638458251953
measuring at 12.0633cm, x1=0.049 y1=-0.162 z1=-1.634, x2=2.021 y2=-3.569 z2=-0.722
callback: 9.202957153320312e-05
step: 0.09592318534851074
finished at 12.145565958847476cm
motor turned off
saved pos
saved pos
motor turned off
pi@gradiometer:~/gradiometer $ python3 Grad_GUI.py
libEGL warning: DRI2: failed to authenticate
qt5ct: using qt5ct plugin
goTo: starting at 0
goTo: will take 972 steps
0.01 sec per step
```

Gradiometer Calibration

Calibration Instructions

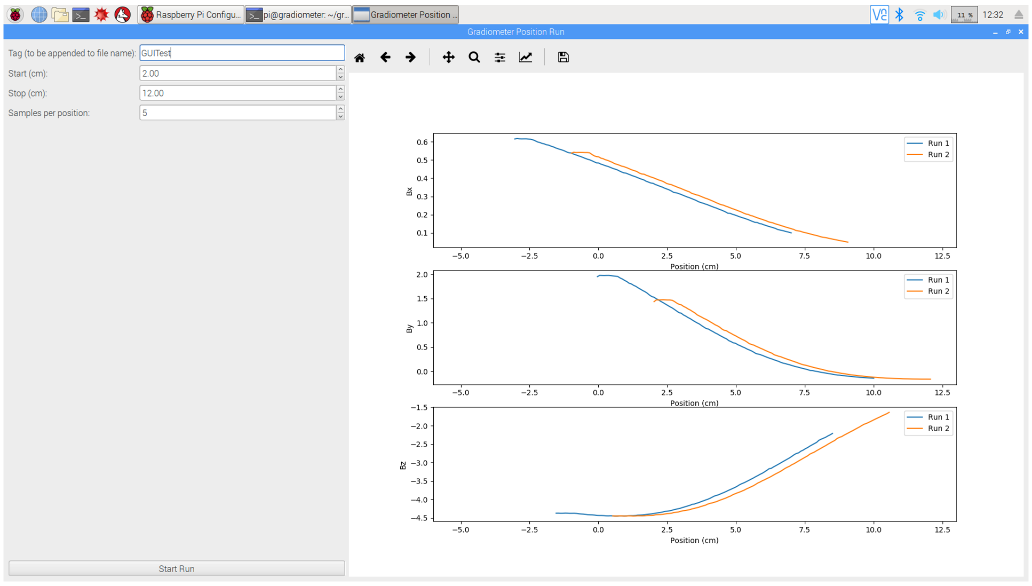
We will now take the calibration measurement

The gradiometer carriage should now move approximately 80cm. Once it's done, take a tape measure and measure this distance precisely. Your measurement will be used to calibrate future step sizes. Enter the measured distance in cm in the space below (will appear when motor finishes moving).

0.00

Submit

Figure 5: Calibration user interface.



Tag (to be appended to file name): GUITest
Start (cm): 2.00
Stop (cm): 12.00
Samples per position: 5

Figure 6: Position run interface. Note that time run interface is extremely similar.

CODE BASICS

The gradiometer is currently controlled by using three python classes stored in `/home/pi/gradiometer/`. Most of the functionality is in `Gradiometer.py`, while `Motor.py` and `Fluxgate.py` are used by `Gradiometer.py` for functions related to the stepper motor and fluxgates respectively.

Gradiometer.py

This class contains all of the top level functions and is responsible for most of the gradiometer operation. The class has one class variable called `CM_PER_STEP=0.08` which defines how many centimeters the fluxgate cart moves for each stepper motor step. See Calibration for how to set this variable.

The Gradiometer object contains two instances of Fluxgate (representing the two fluxgates), one instance of Motor (representing the motor), a Labjack U6 object, and a float representing fluxgate position. Only one instance of Gradiometer can exist at a time because the U6 object will fail to be opened if one already exists (unless you have a second Labjack U6).

Note

When position is changed, floating point error accumulates at a rate on the order of 2e-14cm per 10cm of movement.

The position of the fluxgate is saved and loaded by the python package `pickle`, which can write any object to a simple binary file and then reconstruct it later. This allows the gradiometer to remember where the fluxgate is between runs. The position is saved to `POSITION.pickle` in the `/home/gradiometer/` folder using the `savePos()` method in Gradiometer. The position is loaded during Gradiometer construction by the `loadPos()` method. Gradiometer also has the methods `getPos()` and `setPos(x)` for accessing or setting the position to x. There is also a convenient method `zero()` which sets the position to 0cm, equivalent to `setPos(0)`. If the fluxgate is moved by hand between runs, the saved position will no longer be accurate and will have to be recalibrated.

The method `oneStep(direction)` makes the stepper motor take one step in the specified direction, either `1` for forward (towards the motor side) or `2` for backwards (away from the motor side). In practice, these numbers are actually passed into the function as class variables from the class `Adafruit_MotorHAT` from `Adafruit_MotorHAt_Motors.py` where we see:

```
class Adafruit_MotorHAT:
    FORWARD = 1
    BACKWARD = 2
    BRAKE = 3
    RELEASE = 4

    SINGLE = 1
    DOUBLE = 2
    INTERLEAVE = 3
    MICROSTEP = 4
...
```

In the Gradiometer class, these are accessed like `self.motor.mh.FORWARD`. When this method tells the motor to step, it uses the `DOUBLE` step style which uses two coils to take the step, providing the highest available torque. You will likely never call `oneStep(direction)` by itself, but it is used by other methods.

The method `goTo(cm)` asks for a desired position in cm and uses `CM_PER_STEP` to calculate the number of steps needed to move the fluxgate from its current position to the desired position. Because the motor moves in discrete steps, it may not be possible for it to land exactly at the position you specify, but it will go to the nearest possible position. For example, if the fluxgate starts at 30cm and you call `goTo(40)` with `CM_PER_STEP=0.082268`, the function rounds to 122 steps (121.55 would have been the exact number of steps) and ends at `pos=40.036696`.

The two main measurement methods are `posRun` (short for "position run") and `timeRun`. In simple terms, `posRun` allows you to specify a range of positions over which the gradiometer will take measurements at every step, and `timeRun` allows you take continuous measurements at a single position for a certain amount of time. Both methods create .csv files recording time stamps, position stamps, and measurements with uncertainties for all three axes of both fluxgates.

The full function call of `posRun` is:

```
posRun(start,stop,tag,graph=False,samples_per_pos=5)
```

`start` and `stop` are the positions in cm where you want the run to begin and end. `posRun` first calls

Warning!

There is no coded limit on the position values you give the gradiometer. If you define pos=0 as the end of the shielding close to the motor, you should not need to go past pos=80. Giving too large a value will ram the fluxgate into the gears and could damage it.

`goTo(start)` and then does the same calculation as `goTo` to determine the number of steps to take to get to `stop`. The gradiometer will move in a similar way to `goTo`, but will take a measurement at each step and write it to a .csv file.

`tag` is a string to include in the name of the .csv file for the run so it can be distinguished from others. `posRun` saves files in the folder `Run_Data` and names them like `"date_time-tag.csv"`. For example you could measure parallel to the mounting direction of neutron spin polarizer foil 7, set `tag="parallel7"` and get the file `"/Run_Data/2020-03-25_13-44-40-parallel7.csv"`.

The boolean `graph` (default False) determines whether the `plotter` method will be called at the end of the run. `graph=True` shows a raw plot of the data from all six fluxgate axes (something like Fig 2), and is useful to do a quick visual check of the results before moving on to more measurements.

The integer `samples_per_pos` tells the Fluxgate object how many samples to average together for each measurement. This allows a measurement uncertainty to be calculated using the standard deviation of the set of samples, divided by the square root of `samples_per_pos`. The standard deviation of less than 3 points is meaningless, 5 is a good minimum that doesn't take too long. Based on tests, increasing `samples_per_pos` by ten samples adds a quarter of a second to each measurement, which corresponds to an extra 30 seconds for a 10cm measurement run. Note that increasing `samples_per_pos` will increase individual measurement precision, but runs should be repeated multiple times to get actual statistically relevant results.

As an example, a call to `posRun` where we're measuring a component in the center of the gradiometer (35-60cm) where we want to check the graph afterwards and have higher sampling for better uncertainties might look like: `posRun(35,60,"component37",True,10)`

Where `posRun` takes single measurements at many positions, `timeRun` takes many measurements at a single position using the Labjack's data stream functionality. `timeRun` first configures a data stream and moves the fluxgate into position, and then begins streaming packets of samples continuously until the time limit is reached. The full function call of `timeRun` is:

```
timeRun(sec,tag,scanFreq=1000,cm=None,graph=False)
```

`sec` is an integer number of seconds for which to run the data stream. Both the system time and the time since the start of the run will be logged in the .csv file.

`tag` is a string to distinguish the run in the .csv file name, which are named in the same fashion as the files in `posRun`, described above.

`cm` is a float that specifies what position in cm the run should be done at. The default is `None`, in which case the run will be done at the fluxgate's current position.

The boolean `graph` (default False) determines whether the `plotter` method will be called at the end of the run in the same way as for `posRun`, except with time on the x-axis rather than position.

`scanFreq` is a parameter used in configuring the data stream. It is the frequency with which the Labjack reads a measurement from **all** of the six AIN channels used by the two fluxgates. This is different from the sample frequency which is the frequency with which the Labjack reads **any** of the channels. For example, if you set `scanFreq=1000` (the default value), the Labjack will read **all** six channels 1000 times per second, which means the sample rate is 6000Hz. To further complicate things, samples are sent to the RPi in large groups called requests. Each request contains 48 packets which each contain 25 samples, so 1200 samples are sent to the RPi at a time, 200 for each channel. `timeRun` averages each of these 200 samples together to get a data point for a channel, and uses the standard deviation as the uncertainty. This means that for `scanFreq=1000`, we only actually get 5 data points for each channel per second. Test measurements show that maximum scanFreq is around 3900Hz. You can also lower the samples per packet using an optional parameter in the `streamConfig` call in `Gradiometer.timeRun` to further speed up measurement. The default is 25, but lowering this decreases data transfer efficiency and has not been tested.

Motor.py

The Motor object is a small helper class that is constructed as a part of the Gradiometer class to contain the functions related to the motor. At construction, Motor creates an instance of `Adafruit_MotorHAT` which talks to the MotorHAT and creates an `Adafruit_StepperMotor` object initialized with motor number 1 (corresponding to the screw terminals on the MotorHAT that the motor is connected to) and `speed=30`.

The single method of the Motor class is `turnOffMotors()`, which sends a signal to all of the motor terminals to turn off voltage to the motors. This is a method that should be called at the end of every measurement script. If the motor is used and this method is not called, the motor will be locked in place, even after the script finishes and the Gradiometer object is closed. If this happens, the motor can be turned off using a new instance of Gradiometer or Motor, or by turning off the RPi. A simple way of making sure this happens is discussed in the Data Acquisition section.

Fluxgate.py

The Fluxgate class is a class to contain the functions relating to the fluxgates. It is initiated with a number referring to one of the fluxgates (either a 1 or a 2) which determines which Labjack registers data is read from. Fluxgate 1 reads from registers 0, 2, and 4, while Fluxgate 2 reads from registers 6, 8, and 10. The class is also passed the U6 object that is created in the Gradiometer class construction.

Fluxgate has one method called `sample` which is the function that is called for each fluxgate every time `Gradiometer.posRun` makes a measurement. This function is a convenient place to define how `posRun` makes a measurement; currently it takes the parameter `samples_per_pos` and reads the registers that many times. It returns two length 3 arrays, one of the average reading on each axis, and one of the standard deviation of the average on each axis. While the return should always be in this format, it is easy to change how the measurement happens. For example, you could add a second parameter `interval` and add `sleep(interval)` to the loop to slow down the repeated measurements without having to change the structure of `posRun` (except for adding the parameter to the `sample` call).

CALIBRATION

There are a few factors that need to be considered while using the gradiometer to ensure proper results.

The first factor is using a reference position. In order to get consistent and repeatable results, the user should define a reference position that is reused for all measurements. For example, you could choose $pos = 0$ to be where the end face of the fluxgate is exactly aligned with one end of the gradiometer shielding. This spot could be marked on the gradiometer frame so that the fluxgate can always be returned to zero by hand (see Fig 7). In that case, one would call `Gradiometer.zero()` before running the next command.

The next factor is backlash, which is something that affects all timing belt systems. When the fluxgate is moving, only one side of the pulley teeth are in contact with one side of the timing belt teeth, with a small amount of clearance on the other side to allow the belt teeth to easily enter and exit the gear grooves. When the fluxgate switches directions, it takes 2-3 steps for the pulley to close this gap and engage the other side of the belt teeth, at which point the fluxgate actually starts to move in the other direction. To combat this effect, the direction of measurement should be determined before the run so that the proper side can be engaged. The fluxgate should be moved to the reference position in the opposite direction to the direction you want it to measure in. For example, if your reference position is at the motor end of the gradiometer and you want do a `posRun` into the shielding (the ”BACKWARDS” direction), the fluxgate should be moved by hand in the ”FORWARD” direction to the reference position. This way, the proper side of the teeth will be engaged and the fluxgate will start moving on the first step. In this case, the position will be accurate as long as the fluxgate is traveling in the ”BACKWARD” direction.

The last thing to check is the `CM_PER_STEP` parameter. For unknown reasons, this number can change slightly on different days. It’s a good practice to do a quick check before spending a day taking measurements. To do this calibration one would normally use the calibration GUI as described above. If this is not possible the steps to do this manually are as follows:

1. Move the fluxgate to the far motor end of the gradiometer and mark the location on the frame.
2. Use the `Gradiometer.zero()` method to set the position to zero.
3. Call `Gradiometer.goTo(80)` to move the fluxgate approximately 80cm. `goTo` will print the number of steps it is taking in the console as `'goTo: will take (number) steps'`.
4. Mark the end location on the frame and use a tape measure to find the distance the fluxgate moved.
5. Divide the distance traveled in cm by the number of steps taken and set `CM_PER_STEP` in `config.json` to this result.

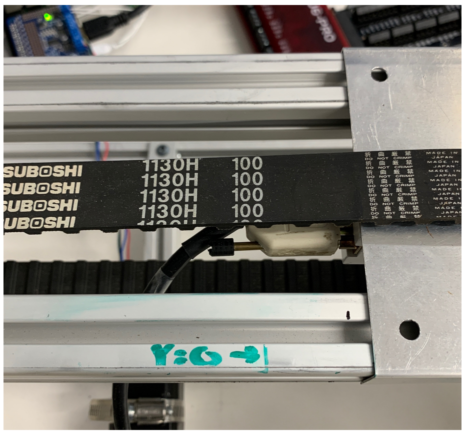


Figure 7: An example of a reference position. The fluxgate mount can be aligned with the mark and zeroed.

DATA ACQUISITION

The benefit of using objects for the control code is that it makes sending commands to the gradiometer easy and flexible. The methods described above can be used in various combinations to make more complicated measurement scripts, or even sent to the gradiometer one by one using an ipython session. When running a measurement script, make sure to use the command `python3 "name_of_script.py"`. Using just `python` will run the script using Python 2.7; the control code was written in Python 3.

Using `atexit`

Regardless of how you measure, there are three functions that should be called at the end of every script or session:

1. Use `Gradiometer.motor.turnOffMotors()` to turn off the motor so the fluxgate can be moved after measuring.
2. Use `Gradiometer.savePos()` to save the fluxgate position to `"POSITION.pickle"`.
3. Use `Gradiometer.labjack.close()` to end communication with the Labjack. If this is not done, you will have to turn the RPi off and then back on to use the Labjack again.

While you could end all your scripts and sessions with these commands, the safest way to ensure that they always get called is to use the `atexit` package. `atexit` allows you to register functions that will be called when a session ends, even if it ends due to a `Ctrl+c` interrupt or a dropped ssh connection. The beginning of your measurement script should look like:

```
from Gradiometer import Gradiometer
import atexit

g=Gradiometer()
atexit.register(g.motor.turnOffMotors)
atexit.register(g.savePos)
atexit.register(g.labjack.close)
... # measurement commands
```

`posRun` example

The following is an example of a script called `just_foil_routine.py` that could measure a neutron polarization foil.

```
from Gradiometer import Gradiometer
import atexit

g=Gradiometer()
atexit.register(g.motor.turnOffMotors)
atexit.register(g.savePos)
atexit.register(g.labjack.close)

g.zero()

foilNum = input('foil number? \n')
direction = input('par or perp? \n')
g.posRun(20, 58, 'foil{}-{}-short'.format(foilNum,direction))
g.goTo(0)
```

This script is designed to start at the reference zero, as shown by the `g.zero()` command. It then prompts the user for a foil number and foil orientation for the filename tag, and proceeds to measure from 20cm to 58cm inside the shielding where the foil is positioned.

`timeRun` example

This example, `axial_coil_routine.py`, is slightly more complicated and shows how multiple measurements can be fit into one script.

```
from Gradiometer import Gradiometer
import numpy as np
import atexit

g=Gradiometer()
atexit.register(g.motor.turnOffMotors)
atexit.register(g.savePos)
atexit.register(g.labjack.close)

g.zero()

positions = np.linspace(0,80,17)
for pos in positions:
    g.timeRun(10,'axial-IOswitch-{}'.format(pos),pos,False)
```

In this script, a loop is used to take 10 second recordings at 17 positions between 0 and 80cm.

Appendix

ADDITIONAL FIGURES

Fluxgate Schematic

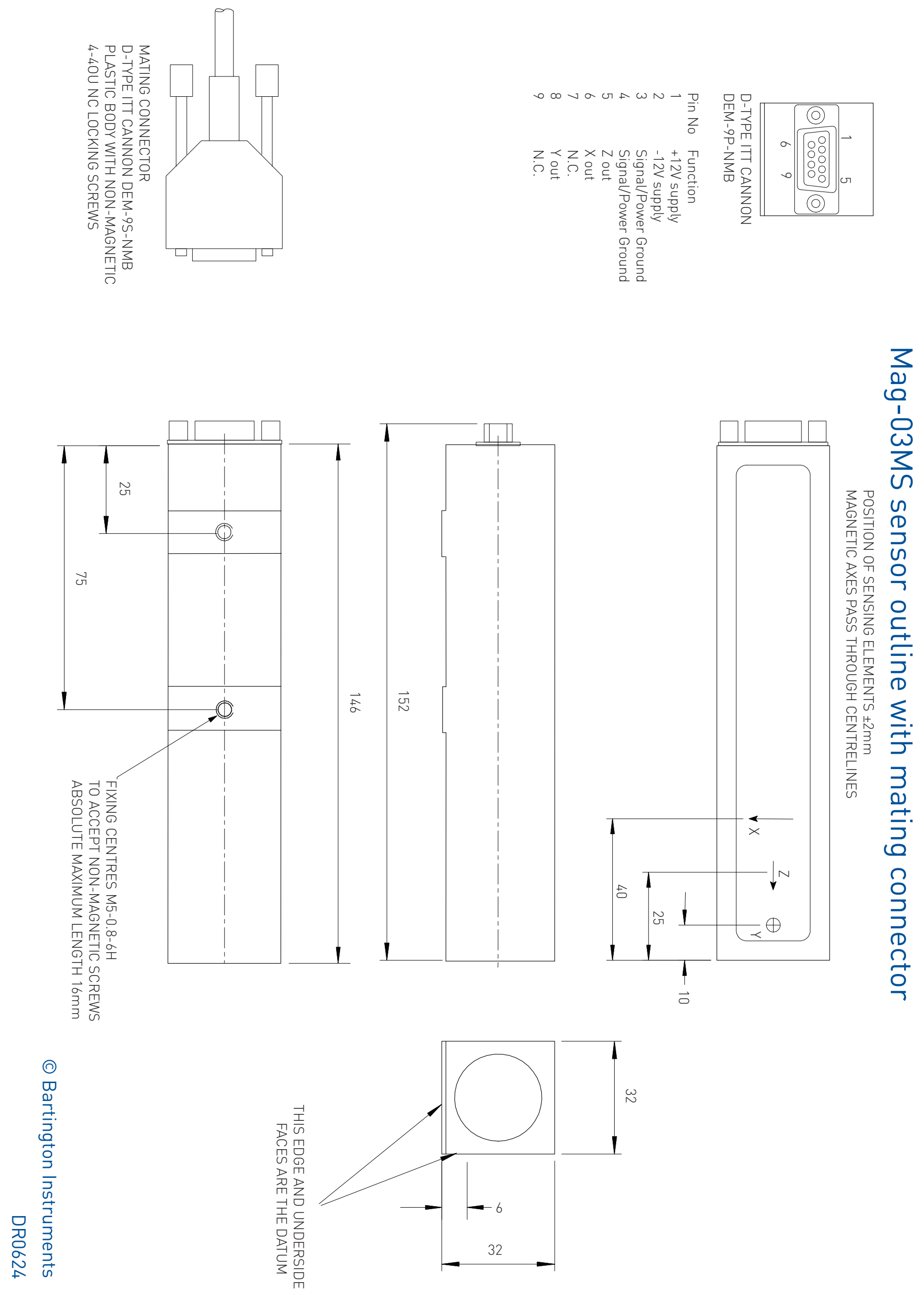


Figure 8:

PSU Amplifier Circuit Diagram

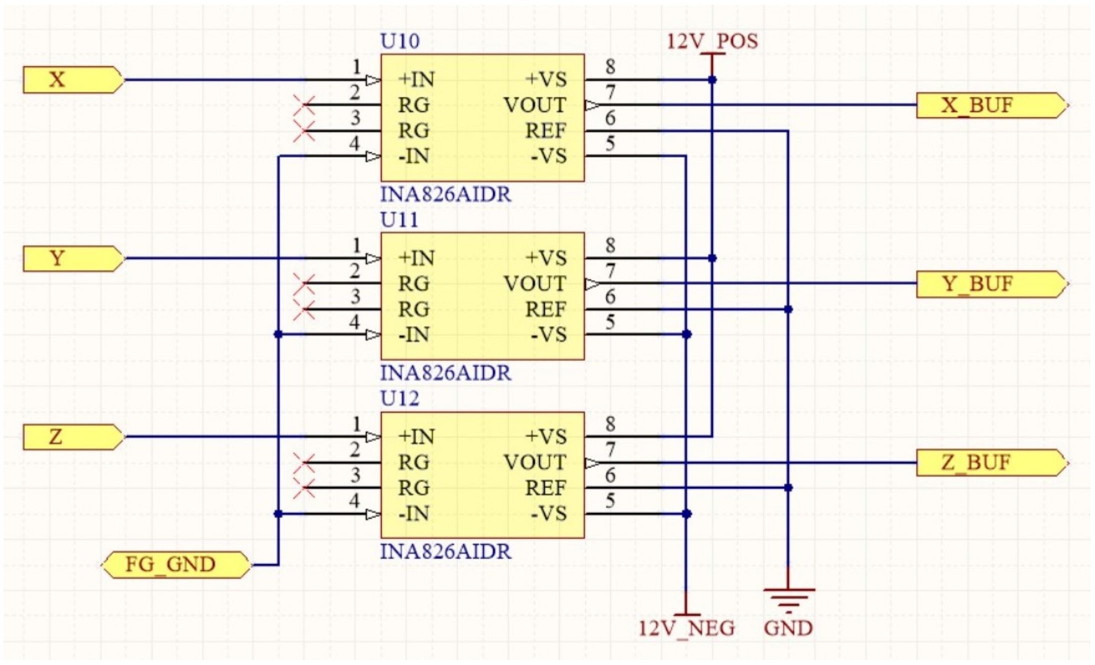


Figure 9:

PSU Filter Circuit Diagram

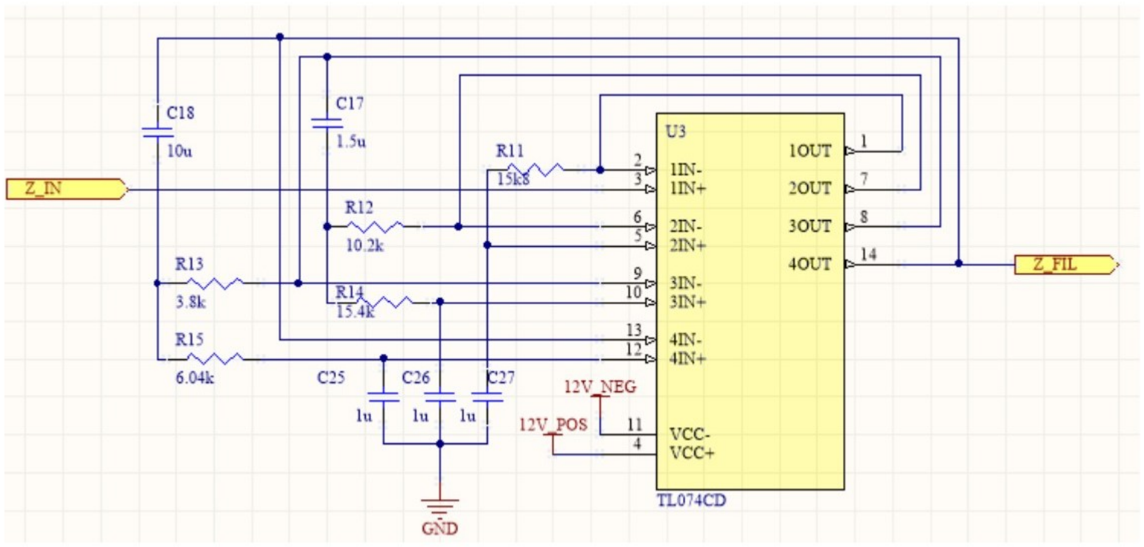


Figure 10:

Gradiometer Shielding

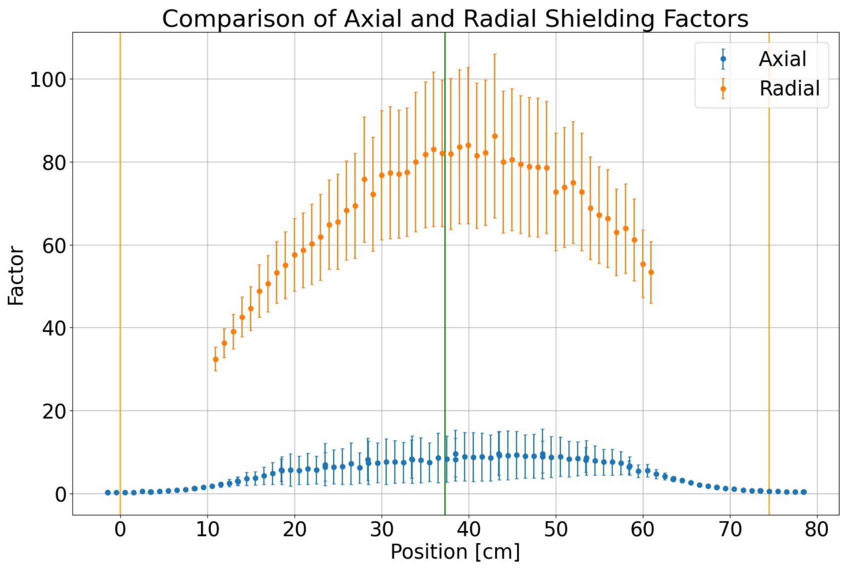


Figure 11: Shows the factor of reduction of magnetic flux within the gradiometer due to the magnetic shielding. The flux is not reduced isotropically; radial fields are better shielded than axial/longitudinal fields. The vertical yellow lines represent the ends of the mu-metal tube.

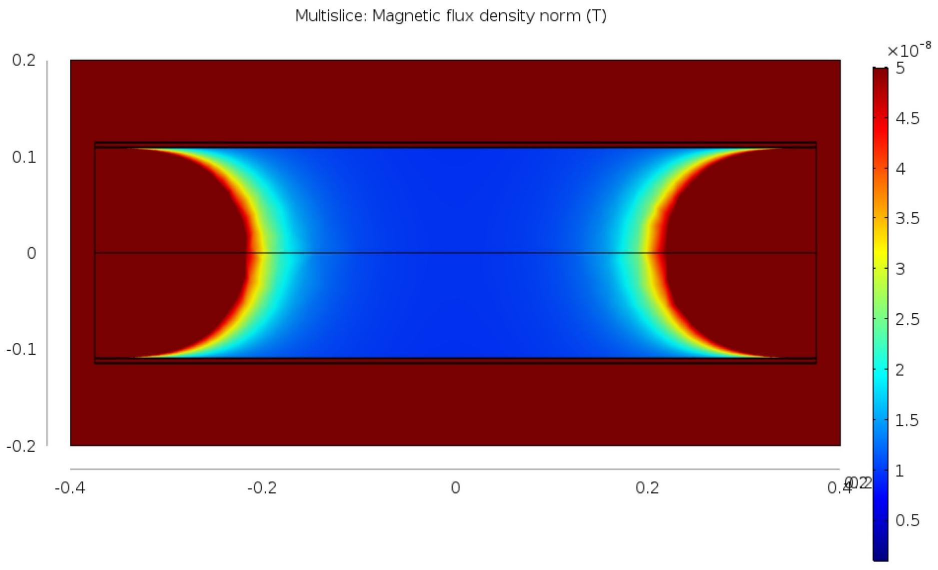


Figure 12: A COMSOL simulation of the shielding factor for a radial field.

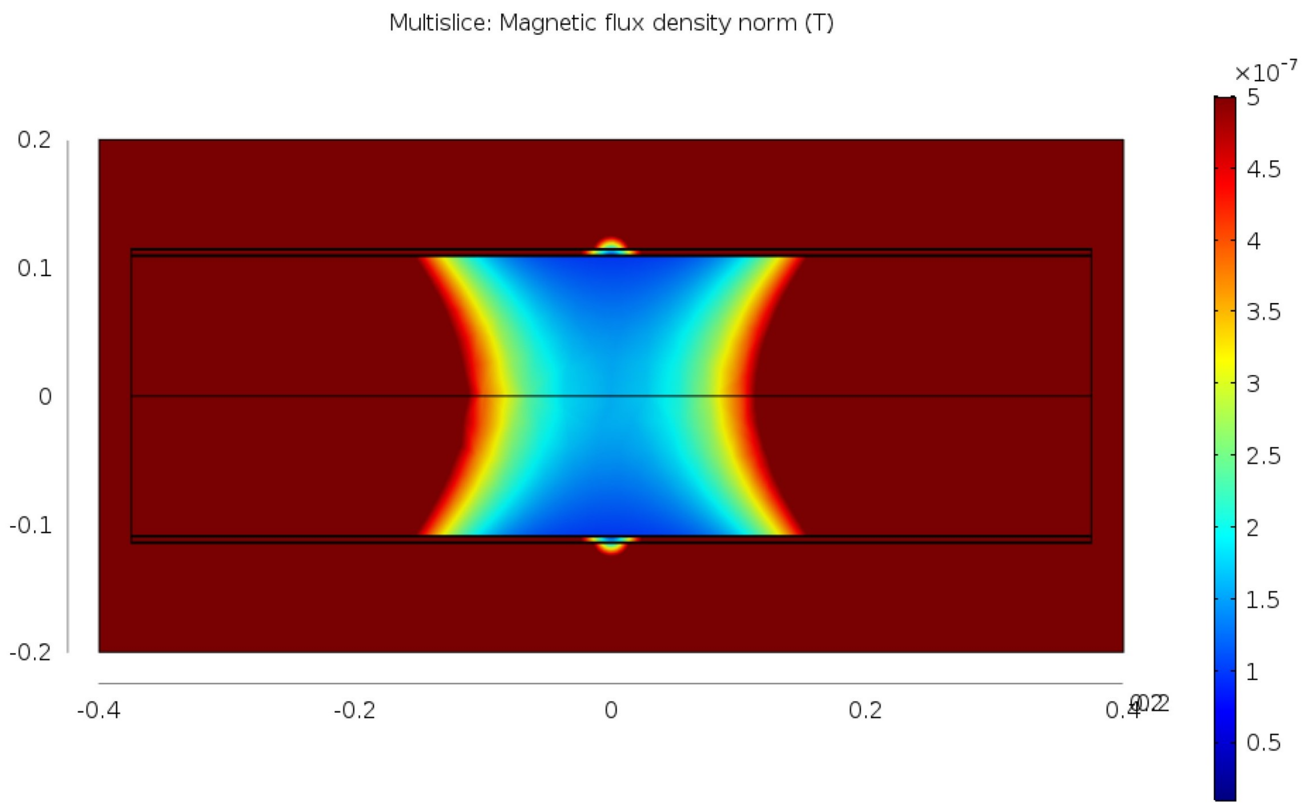


Figure 13: A COMSOL simulation of the shielding factor for an axial/longitudinal field.

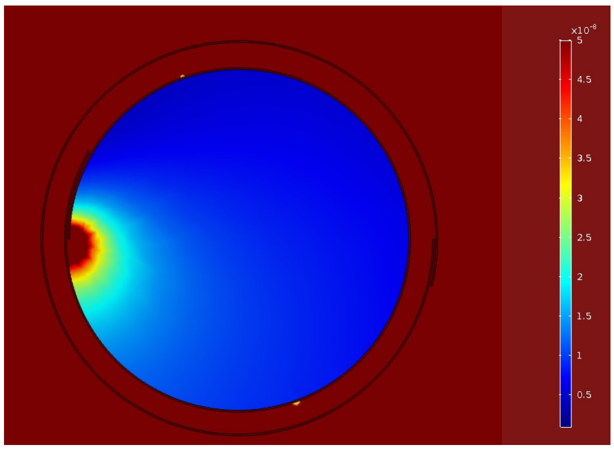


Figure 14: A COMSOL simulation of the shielding factor for a radial field from an end view. Shows how some flux ”leaks” through the welded joint.

Logs

USER MANUAL VERSION

Version	Date	Author	Notes/Changes
1.0	1.1.2021	Cole Bunch	
2.0	TBD	Xander Naumenko	Added GUI description and new repository

RPI WIFI CREDENTIALS

The credentials for TRIUMF Secure WiFi access located in `/etc/wpa_supplicant/wpa_supplicant.conf` on the RPi currently belong to:

Name	Start date	End date	Contact email
Cole Bunch	1.1.2021	31.5.2020	bunchcole@gmail.com
Beatrice Franke	9.9.2020	Current	bfranke@triumf.ca

GRADIOMETER GITHUB REPOSITORY

The current GitHub repository containing the gradiometer control code is:

URL	Start date	End date	Contact email
https://github.com/bunchcole/gradiometer	1.1.2020	1.21.2021	bunchcole@gmail.com
https://github.com/tucan-magnetics/gradiometer	1.21.2021	Current	xandernaumenko@gmail.com