Week 1 - Week commencing 7/2/2022

Week Outline

In-person meeting to visit the camera, take simple measurements and rediscuss the aims and objectives of the project and how the project will move forward. This week is mostly a refresher into the project as this semester will focus more on analyzing experimental detector data than understanding any additional theory. The tasks for this week is to understand what the aims and objectives of the project moving forward and read in I and Q values, and attempt to convert to dF0 using the "Magic Formula". In addition, some photos and simple measurements were taken for future use.

Complete Python Code Attached at the End of Diary

Tasks Outline

- Obtain and read-in detector data
- Convert I & Q values to dF0 using "Magic Formula"
- Plot dF0 vs Time

Simple Measurements

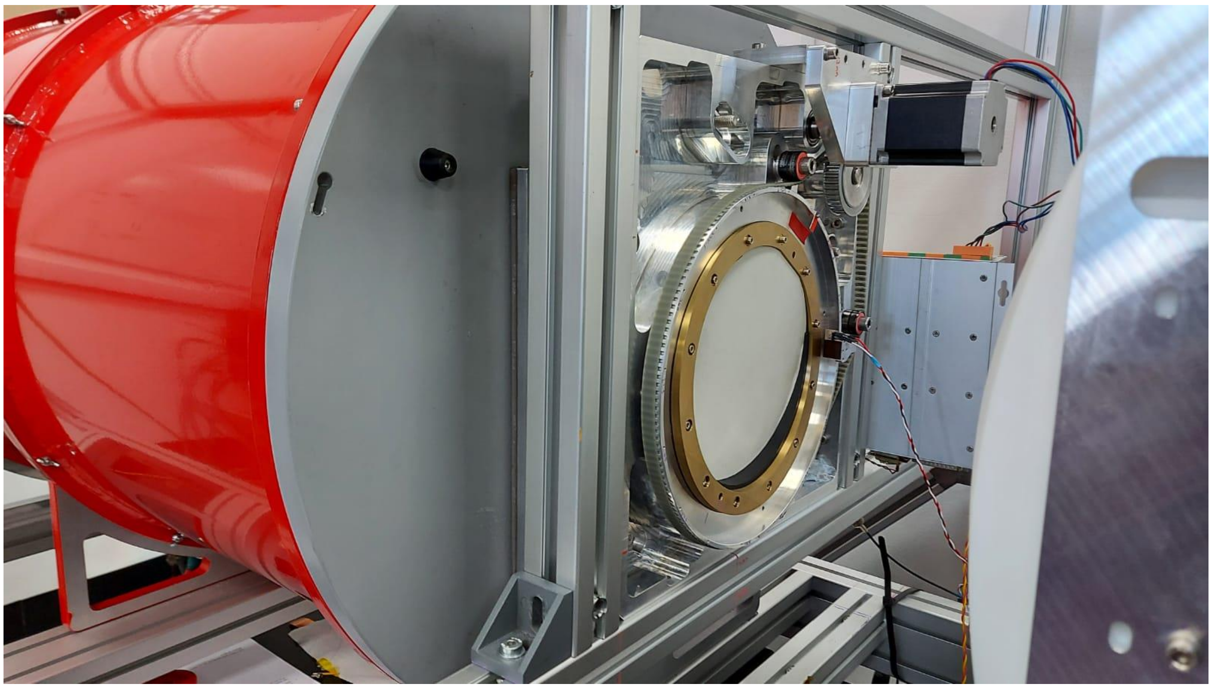
$$T_{hotbar} = 41.3\ °C$$

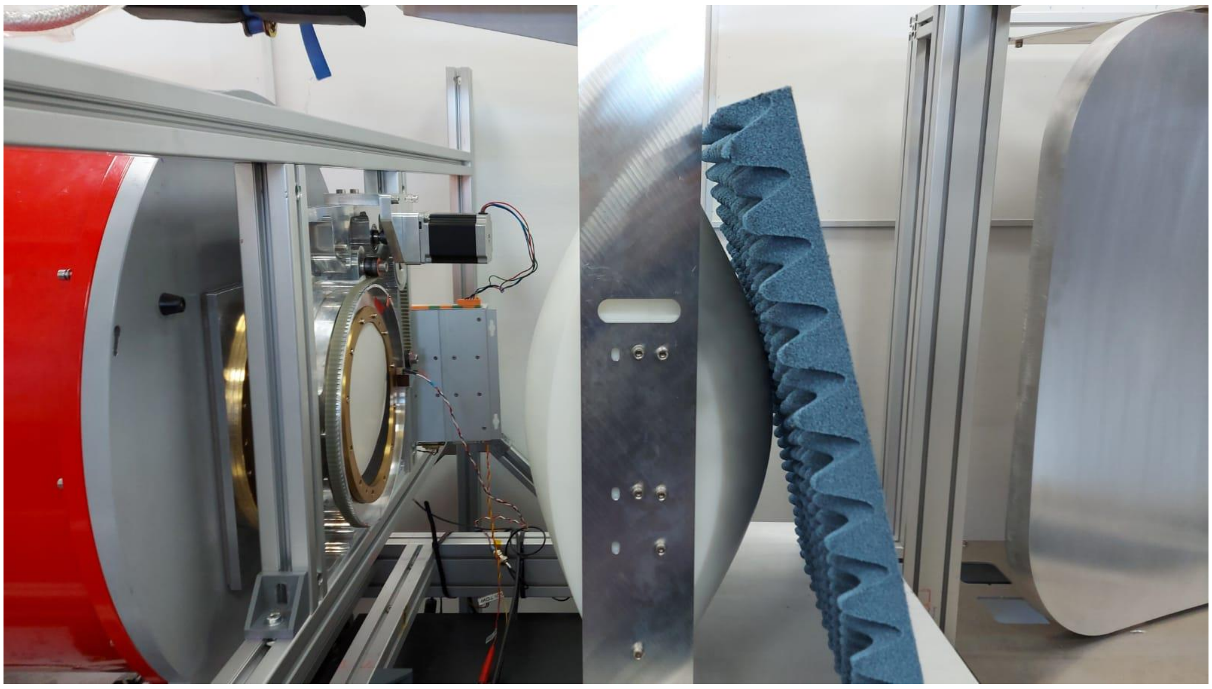
$$T_{surrounding} = 22.7\ °C$$

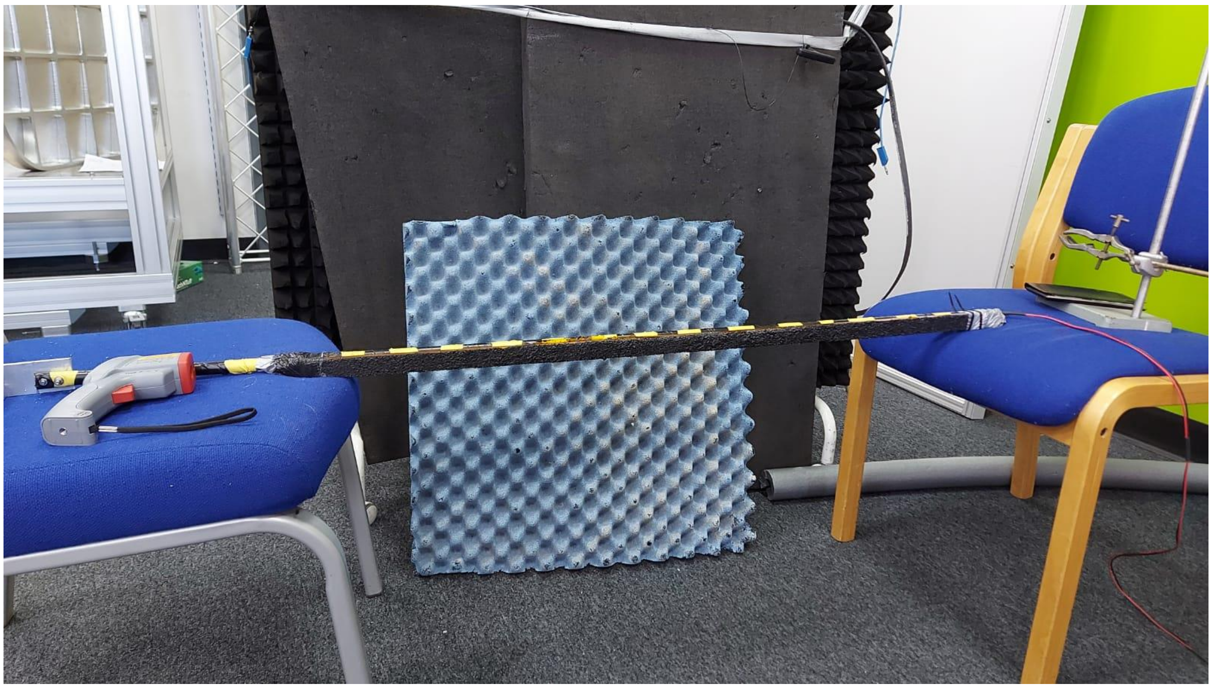
$$length\ of\ hotbar,\ L_{hotbar} = 100\ mm$$

$$width\ of\ hotbar,\ W_{hotbar} = 20\ mm$$

Photos of Camera Setup


Camera. The red drum contains the detector inside held at ~300mK.


Detector with foam screen.

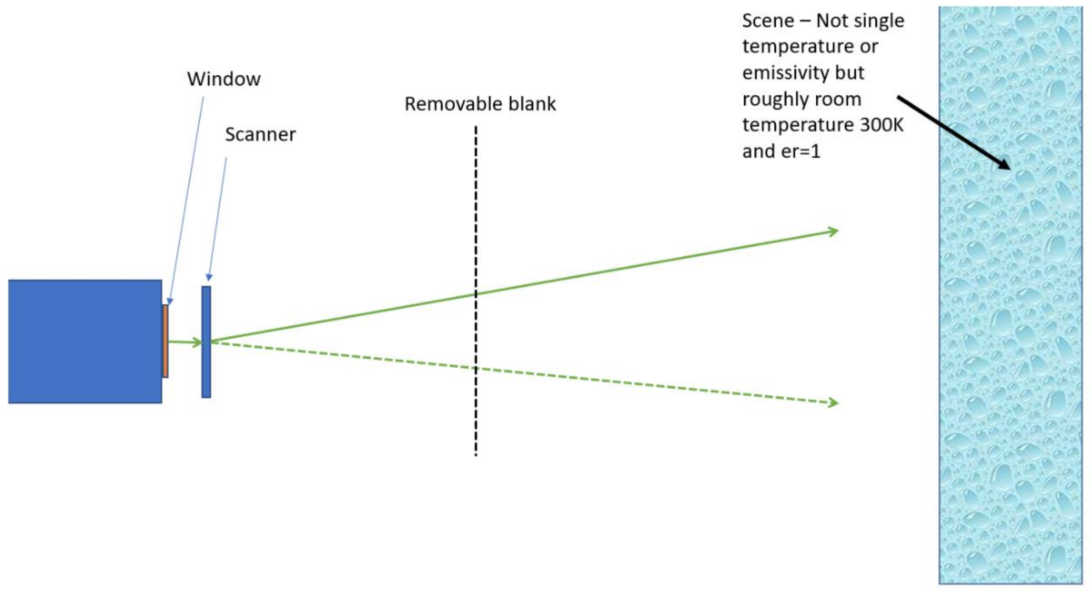


Setup of hot bar, heated electrically.

Measured Data

For the detector data, there are 2 sets of detector data that will be used throughout the project for analysis:

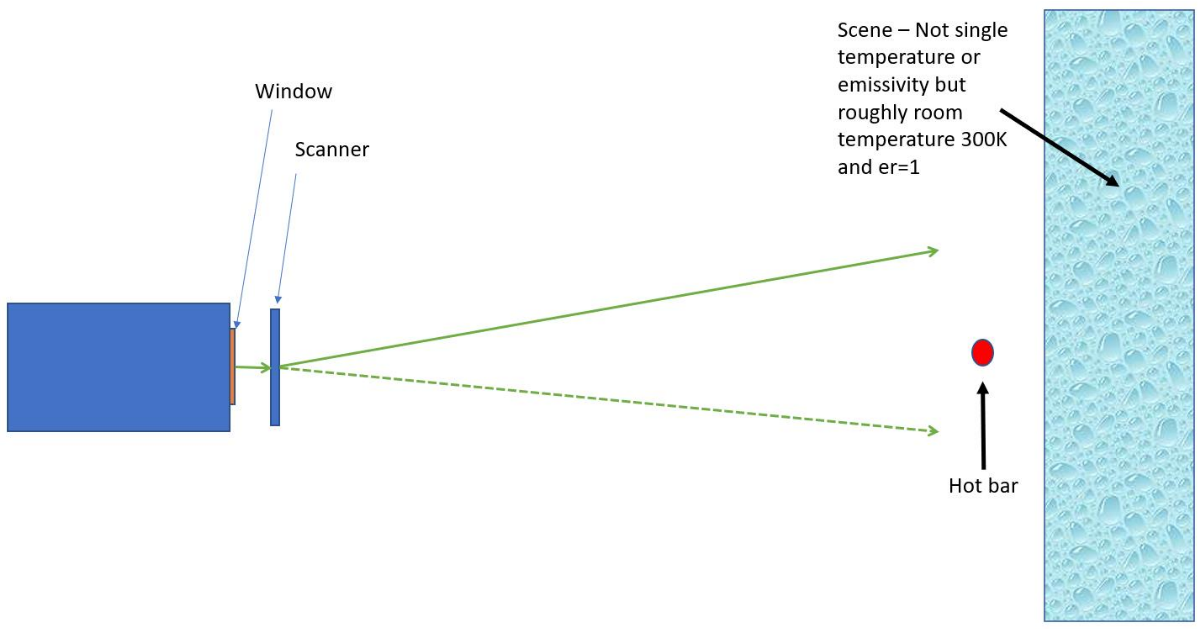
1. Measurement of a blank screen.



Window
Scanner
Removable blank
Scene – Not single temperature or emissivity but roughly room temperature 300K and er=1

This data allows us to measure the background for noise analysis.

2. Measurement with a hot bar


Window
Scanner
Scene – Not single temperature or emissivity but roughly room temperature 300K and er=1
Hot bar

This data allows us to characterize the response of the detector, with known temperature of the hotbar.

Week 2 - Week commencing 14/2/2022

Week Outline

Online meeting to discuss the next step of the data analysis and to clear up misconceptions about obtaining dF0 and responsivity of the KID. As explained, the next step is to read-in I and Q values and converting them to a dF0. Mostly as a continuation of last week's data and camera visit and project overview. By doing so, we will be able to use these dF0 values in finding a response by the detector.

Complete Python Code Attached at the End of Diary

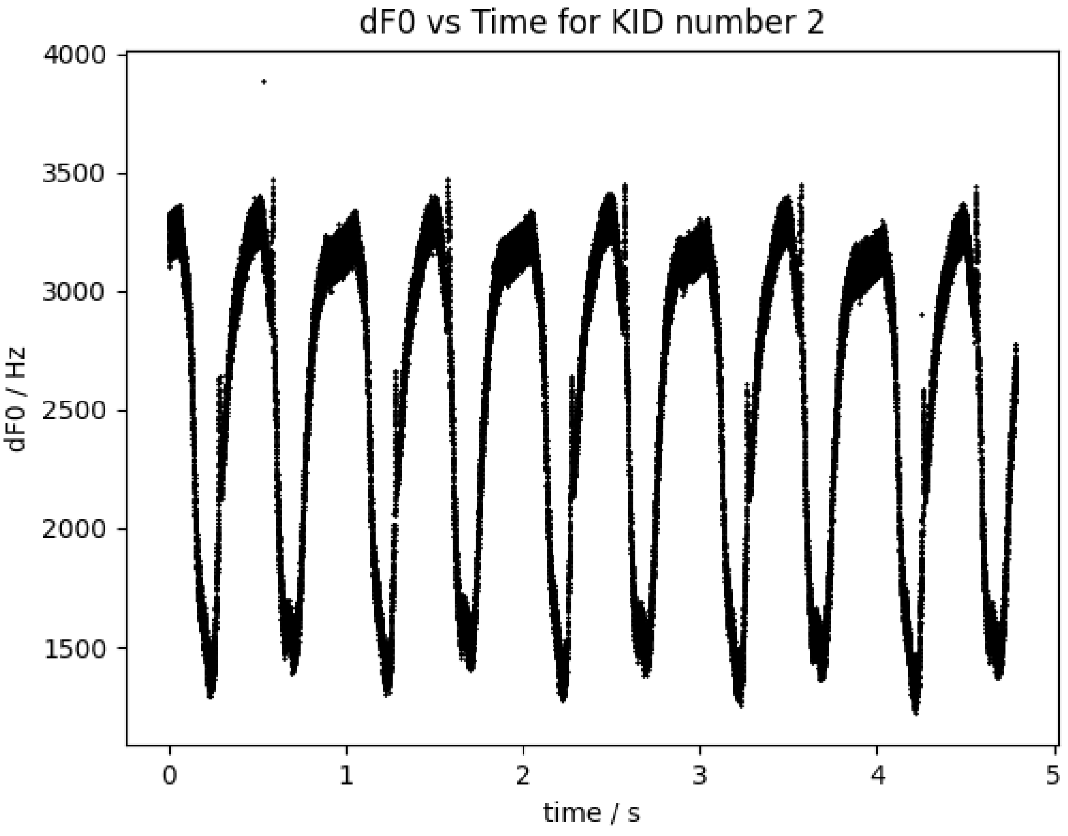
Tasks Outline

- Convert I and Q values to dF0 using the "Magic Formula" for hot bar data:

$$dF_0 = \frac{dI(t)\frac{dI}{dF} + dQ(t)\frac{dQ}{dF}}{\frac{dI}{dF}^2 + \frac{dQ}{dF}^2}$$

- Identify hot bar within data.

Graph of dF0 vs Time



This is **detector data for the hot bar for KID Number 2**. here are 2 distinct features visible, the large oscillating wave is the reflection of the detector screen as it oscillates. The small peaks present on the large waves are the hot bar. As the detector scans up and down, the hot bar falls in the camera's field of view momentarily. The hot bar will have a Gaussian curve with a peak height as the detector takes an integration time to "warm up" to the detection of the hot bar and it "cools down" again as it moves away.

One would expect the oscillating features to be symmetrical as it scans up and down or at a different time, however this is not the case. For example, the reflection and hot bar peaks and curves are not identical to itself in a different time. One of the main contributing reasons is of course the background as it is measured in a "un-ideal" scene with many furniture and objects, at fluctuating "room temperature", which may annoyingly "add extra features" to the curves.

The other reason is that the detector temperature fluctuates. One can imagine due to the internal electronics and the mechanism to regulate the heat itself may produce heat, and other factors such as temperature fluctuations may cause the detector temperature to not remain constant. The detector temperature was advised to only remain somewhat constant for around ~30 seconds before it deviates significantly. This may lead to some additional features to the curves as well.

Week 3 - Week commencing 21/2/2022

Outline

Online meeting to discuss the next step of the project: calculating the response and Noise – equivalent temperature (NET). Following from last week, the dF0 can now be used to find a response of the detector. Using the measurements from week 1, the hot bar is a known temperature T. We can obtain a responsivity based on dF0/dT.

The dF0 can be calculated by identifying the peak of hot bar that takes the shape of a Gaussian curve, where the location and where it is is explained in the previous week. The peak can have a Gaussian curve fitted on it and the peak height can be found, this is the dF0.

Next, the dT can be found by determining the change in temperature of the hotbar and the room. This is just the difference between the temperatures measured in week 1. The response can simply be calculated from these values and the process can be repeated for the other KIDs.

Finally, the NET is related to the response by:

$$NET = \frac{\sqrt{e_n}}{Response}$$

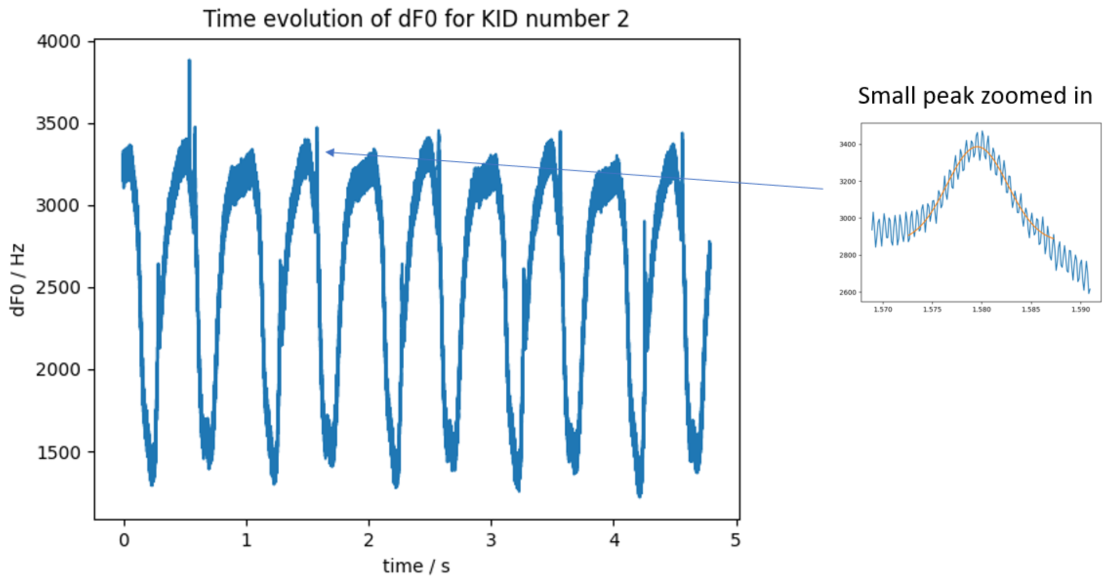
Where e_n is the spectral density. This value can be found by taking the Fourier Transform of the dF0 values and finding the intensities of each frequency bin, then use this value to calculate NET.

Complete Python Code Attached at the End of Diary

Tasks Outline

- Find the peak height of the hot bar feature.
- Calculate a response using the peak height and hotbar temperature
- Take the Fourier Transform of the dF0 data to find the spectral densities
- Find NET using spectral densities and response

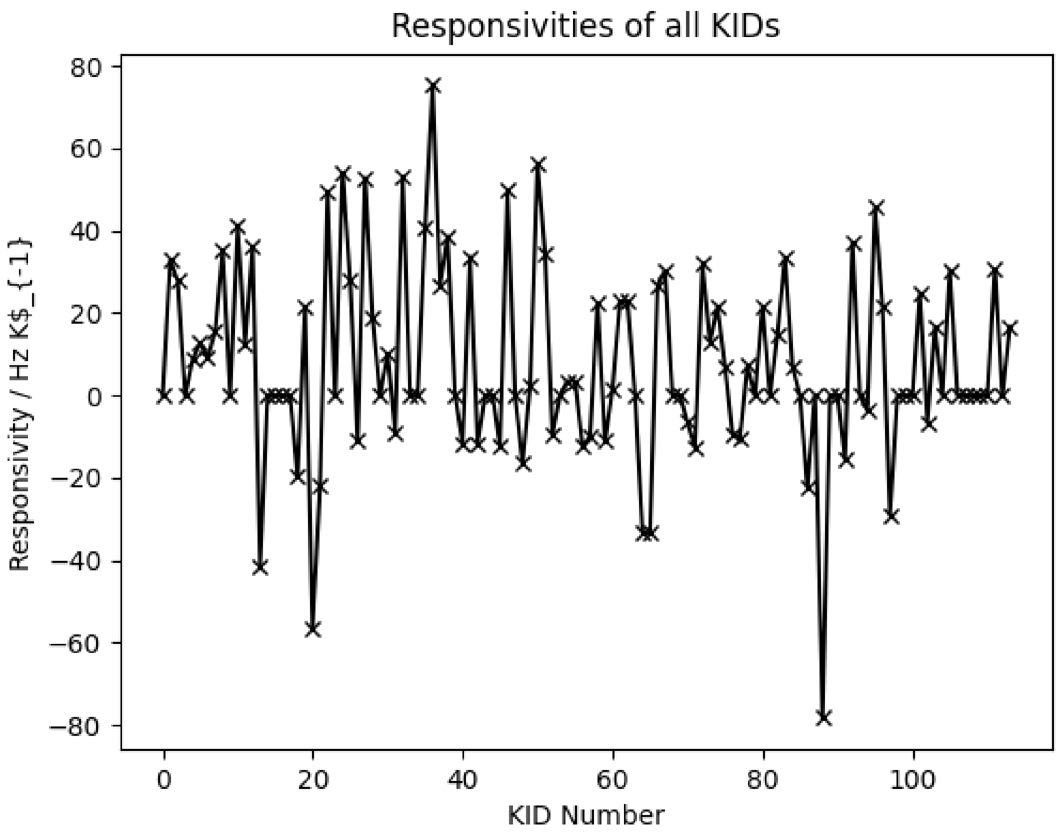
Curve Fit of Hot Bar Curve



A Gaussian was fitted to the hot bar feature of KID 2 and using the Python’s curve fit function, the parameters for the height of the curve can be found, which is the peak height.

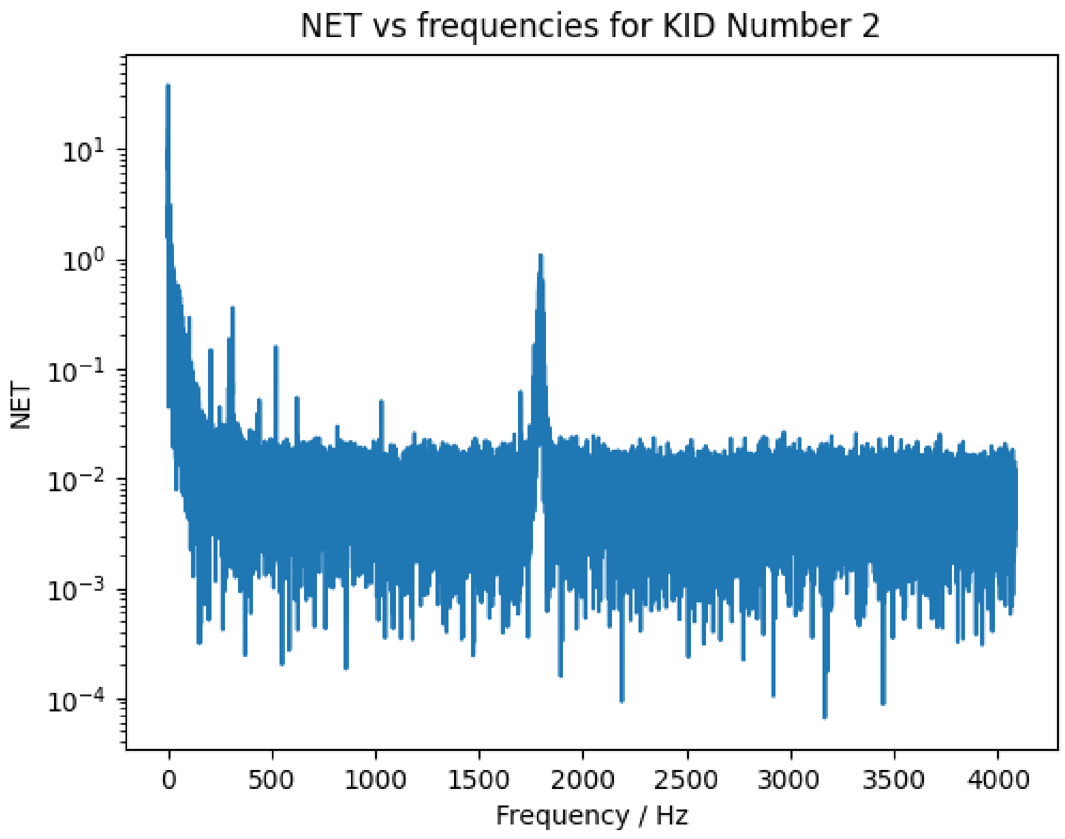
The response for KID 2 was found to be 27.5 Hz/K

Responsivities of all KIDs Graph



The process was looped in Python for the other KIDs. The values exceeding a magnitude of 100 was set to 0. This was probably due to a change in the shape of the curve causing distortions. This can be fixed in a future week.

Graph of NET vs Frequency for KID 2



The Fourier Transform of the dF0 data for KID 2 was taken and the spectral densities found. Then, using the densities, found NET using the mentioned formula. NOTE: NET =/= NEP. To find NEP, calculate power of bar. 1/f noise and hotbar feature evident. The white noise level is also deducible.

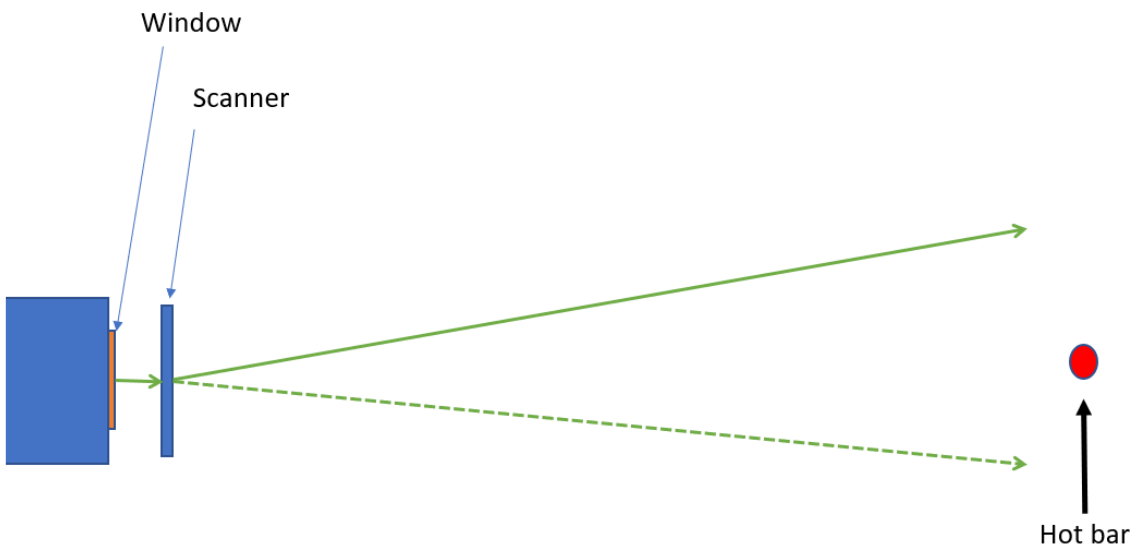
Week 4 - Week commencing 28/2/2022

Week Outline

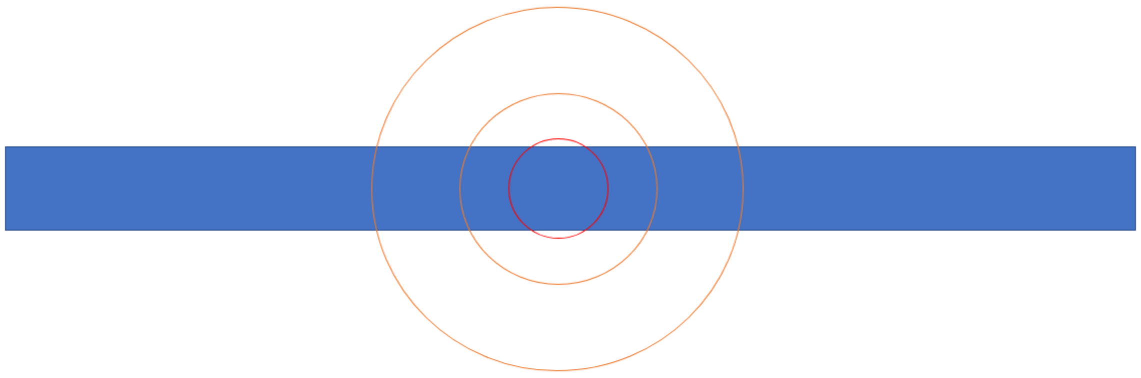
Online meeting discussing the implications and further analysis of the beam filling factor on the measured data from the detector.

The detector has a large solid angle, and thus the full beam of the detector is not entirely comprised of the radiance of the hot bar. An example diagram is given below:

Side view

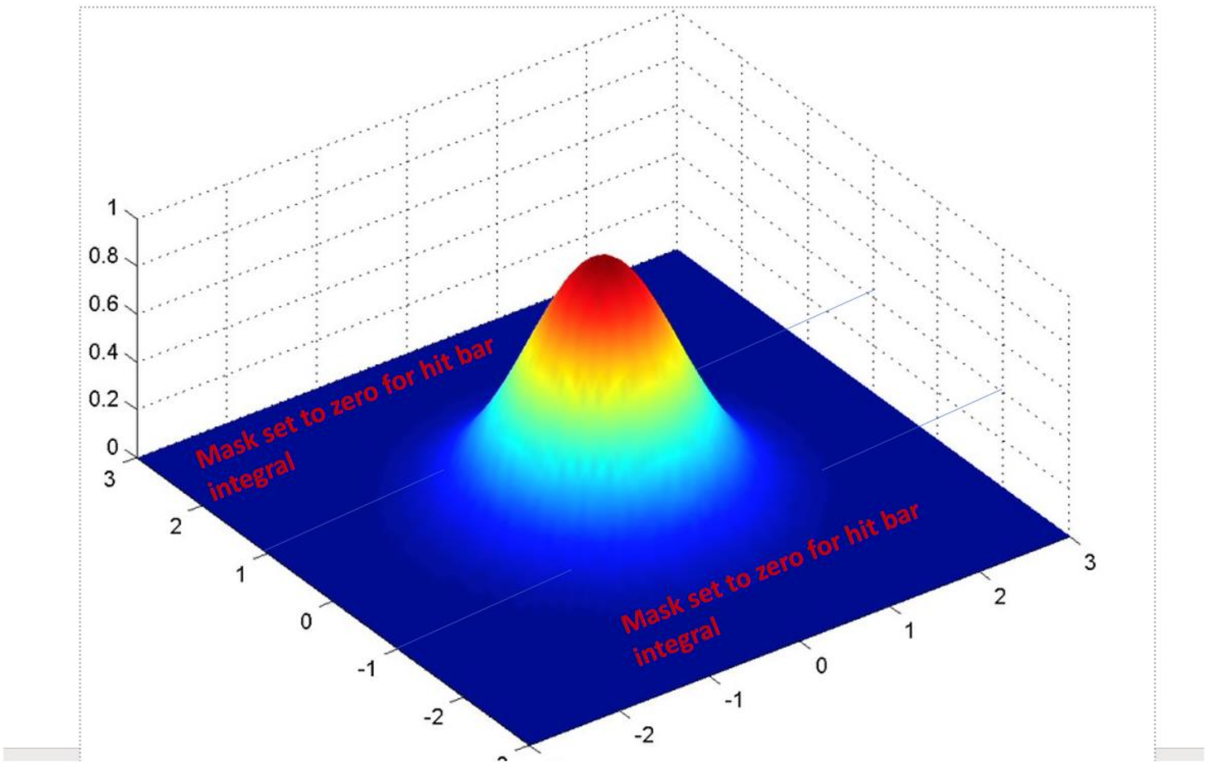


Front View



where the red circle is the beam. As observed, the beam covers parts of the detection that goes beyond the hotbar, and you can imagine that the 2-D Gaussian dF0 peak that forms from the detection for a given detector data at a given time comprises of "addition of power" from background sources and that not 100% of the peak of the curve used for calculating the response corresponds to the hot bar. This leads to the idea of a beam filling factor.

The 2D Gaussian corresponding to the detection of the hot bar can be modelled in such a way that if we use and "cut" a 2D Gaussian of length^{2}, to only the dimensions of the hot bar and set the outside of the curve to equal to 0. This is shown below:



We can then take the sum of the "cut" Gaussian curve and the "uncut" Gaussian curve and take the ratio of them, this gives a "Beam Filling Factor":

$$BFF = \frac{\sum hot\ bar\ Gaussian}{\sum 2D\ Gaussian}$$

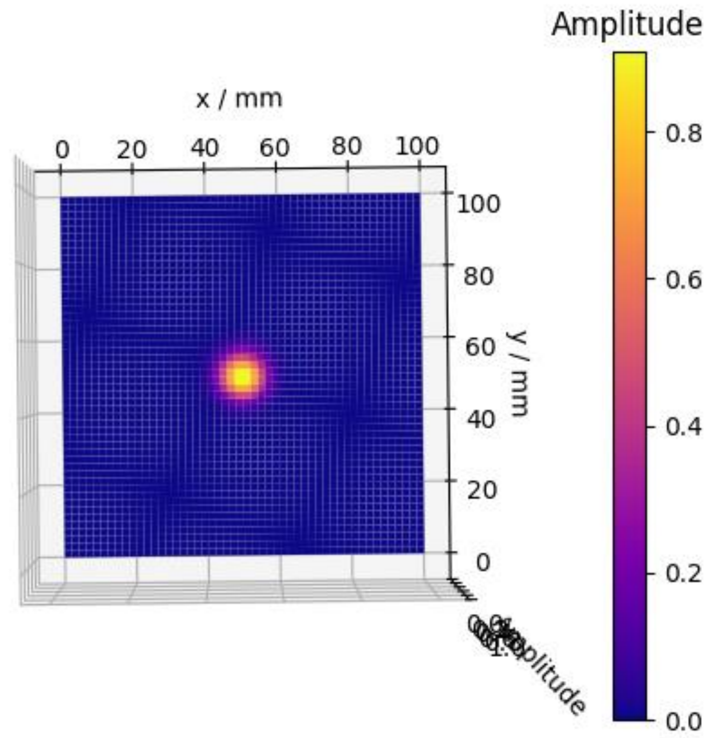
This beam filling factor essentially gives a ratio of the beam that is filled by the hot bar. This will be essential in calculating the power of the hot bar in the following weeks to find the NEP.

<u>Complete Python Code Attached at the End of Diary</u>

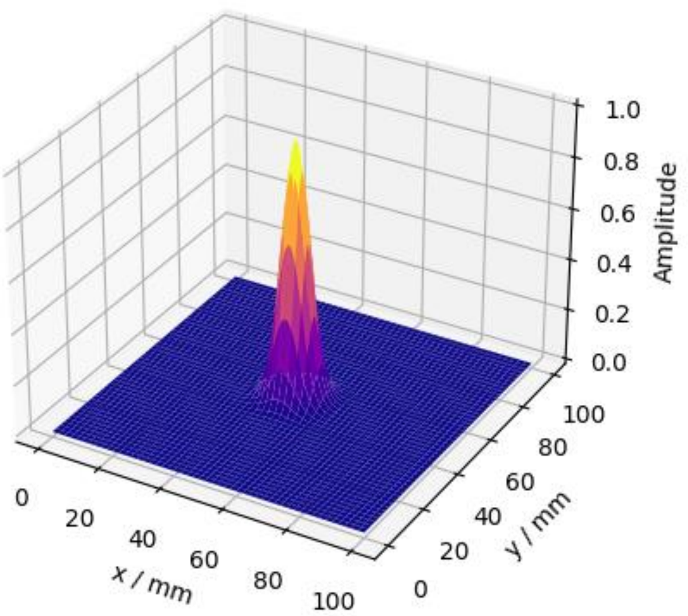
<u>Tasks Outline</u>

- Create a 2D Gaussian Curve with amplitude 1, fwhm = width of hot bar and plot this for the length of the hot bar squared.
- Take the sum of this Gaussian as Gaussian_Sum
- Set the points outside the hot bar dimensions as equal to 0
- Take the sum of this new "cut" Gaussian.
- Take the ratio of Gaussian_Sum to "cut" Gaussian sum as the Beam Filling Factor.

Top Down View of the cut Gaussian



Side View of the Cut Gaussian

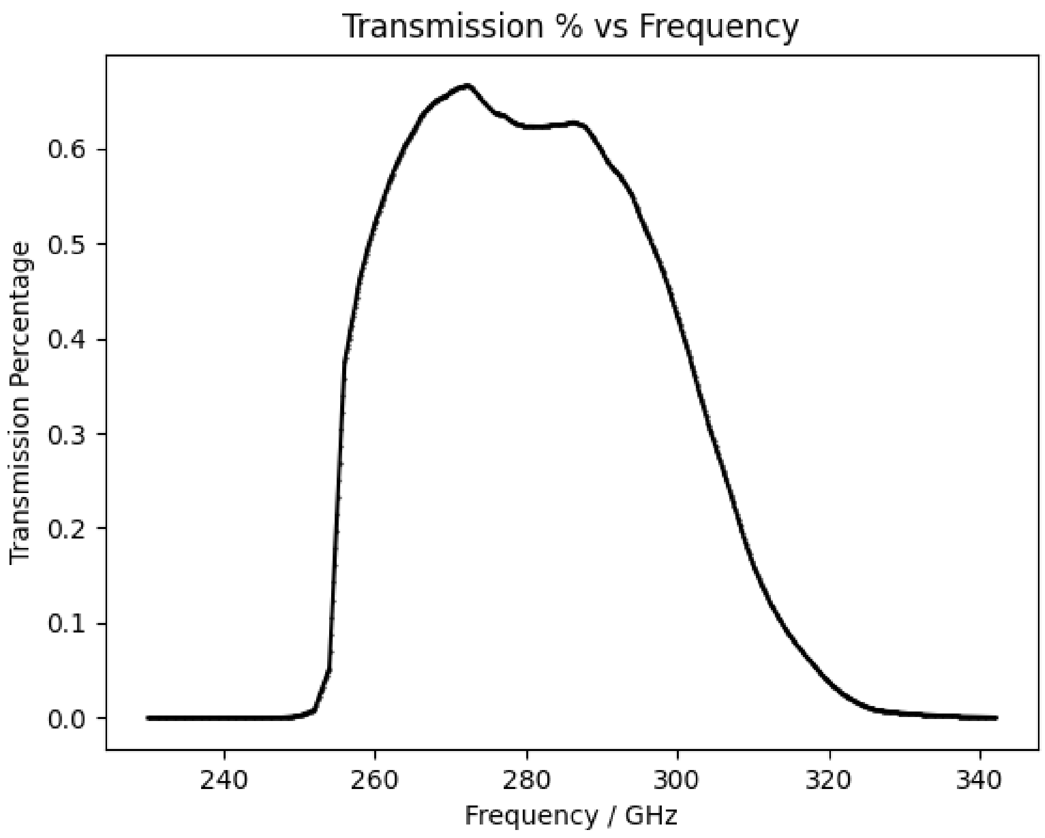


Beam Filling Factor was found to be: 0.9847

Week 5 - Week commencing 7/3/2022

Week Outline

Online meeting discussing the next steps of the project, calculating a power emitted by the background as a blackbody. Starting from the filter profiles of the detector, the filter profiles essentially acts as a "transmission" factor for a range of frequency ranges. These filter profiles were combined and given as data, plotted as shown:



This gives the overall power that is filterered away and can be multiplied with the power to give the received power by the detector.

The next step is to calculate the integral for the blackbody function over the filter bandwidth. We can model the background as a blackbody at temperature T measured in week 1 and by integrating over the bandwidth, gives us the power/solid angle.

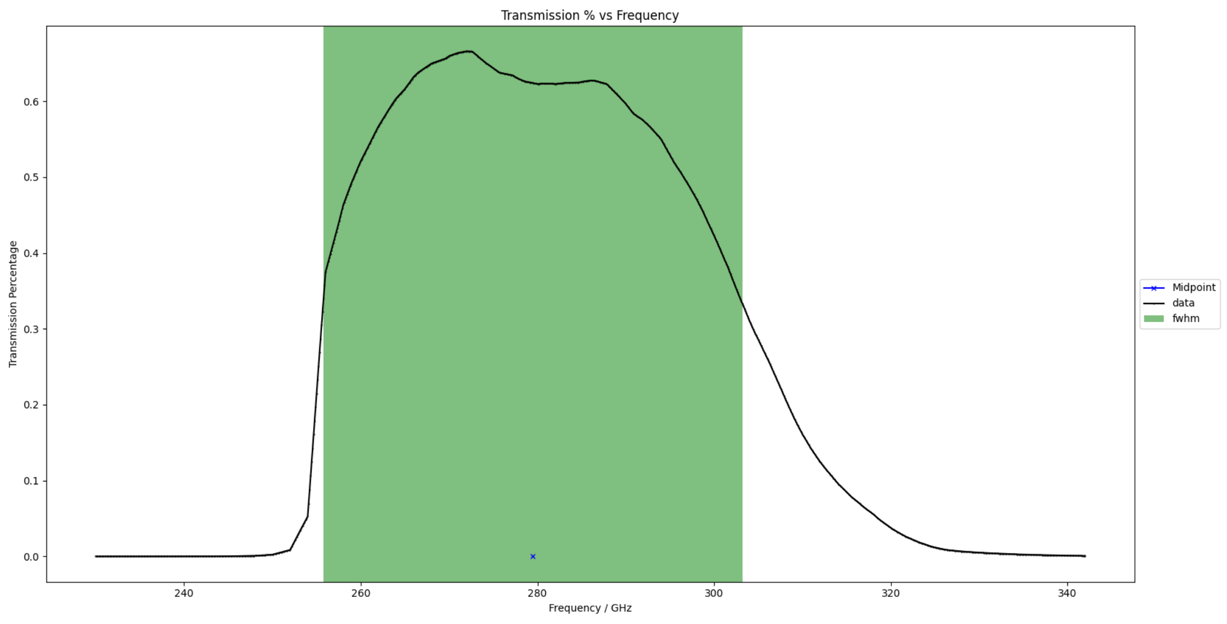
Finally, we can multiply this power/solid angle with λ^2 as the antenna throughput and obtain the power received by the detector from the room.

Complete Python Code Attached at the End of Diary

Tasks Outline

- Workout the Full-width-half-maximum of the filter profile and take the midpoint of it as the frequency v
- Workout dv as the final frequency point – first frequency point of the filter profile
- Create a Planck Function, use the temperature T of the room and frequency range as the inputs and integrate over the Planck Function over the first and last value of the frequency of the filter profile.
- Use this value and multiply by the transmission profile to get the filtered power/solid angle
- Finally, use this value and multiply by the throughput of the "antenna" λ^2 which is the midpoint frequency v converted to wavelength to get power received by the detector.

Transmission Profile with FWHM and Midpoint



Green bar = FWHM

Blue cross = midpoint

Results

Power received by the detector emitted by room at 293K over the filter profile bandwidth:

$$P = 2.430 \times 10^{-10}\ W$$

Or

$$P = 243\ pW$$

Distinction between the NEP and NET

Following from previously, we have calculated a NET based on detector data and the response. The NET gives a good description for the system as a whole, but the NEP gives a better description for th detector as NET does not consider the optical bandwidth but the NEP does.

A good way to think about it is to imagine a thermal camera. A thermal camera typically has a large bandwidth. Thus, this allows it to have a higher sensitivity to temperature gradients, e.g. low NET. However, having such a large bandwidth essentially “masks” the detector, as larger bandwidths means allowing for more noise to saturate the data, thus increasing the NEP. The NEP gives a better description of the fundamental limit of the detector when compared to the photon noise.

Week 6 - Week commencing 14/3/2022

Week Outline

Online meeting to cover error fixing in the Python code and to discuss the next step of the project. After obtaining the power received by the detector, we can now calculate the photon noise as a quadrature sum of the wave noise and shot noise. The formulas for calculating them are as follows:

$$shot\ noise = \sqrt{2P_{opt}hv}$$

Where v is the frequency, found from the midpoint of the bandwidth and P_{opt} is the optical power calculated from last week.

$$wave\ noise = \frac{P_{opt}}{\sqrt{2dv}}$$

Where dv is the bandwidth.

$$Total\ photon\ noise = \sqrt{WaveNoise^2 + ShotNoise^2}$$

Following this, we can also calculate the power emitted by the hot bar by using the same procedure as for the room power with the addition of the Beam Filling factor:

$$P_{hotbar} = (BeamFillingFactor)\lambda^2 \int B(v)\, Transmission\ dv$$

Then, the response of the hot bar in terms of power:

$$Response = \frac{dF0}{P_{hotbar}}$$

Then, finally the NEP can be found:

$$NEP = \frac{\sqrt{e_n}}{Response}$$

Tasks Outline

- Calculate the shot noise and wave noise using the optical power P_{opt} obtained last week
- Use the shot and wave noise to calculate the total photon noise
- Find the power of the hot bar using the equation given previously
- Use this power to find the response of the detector in terms of the power
- Calculate the NEP

Results

$$Total\ Photon\ Noise = 8.4367 \times 10^{-16}\ W/Hz^{0.5}$$

Incomplete tasks for week 6 as meeting was held on 17th of March, 1 day before Week 6 Diary Due on 18th

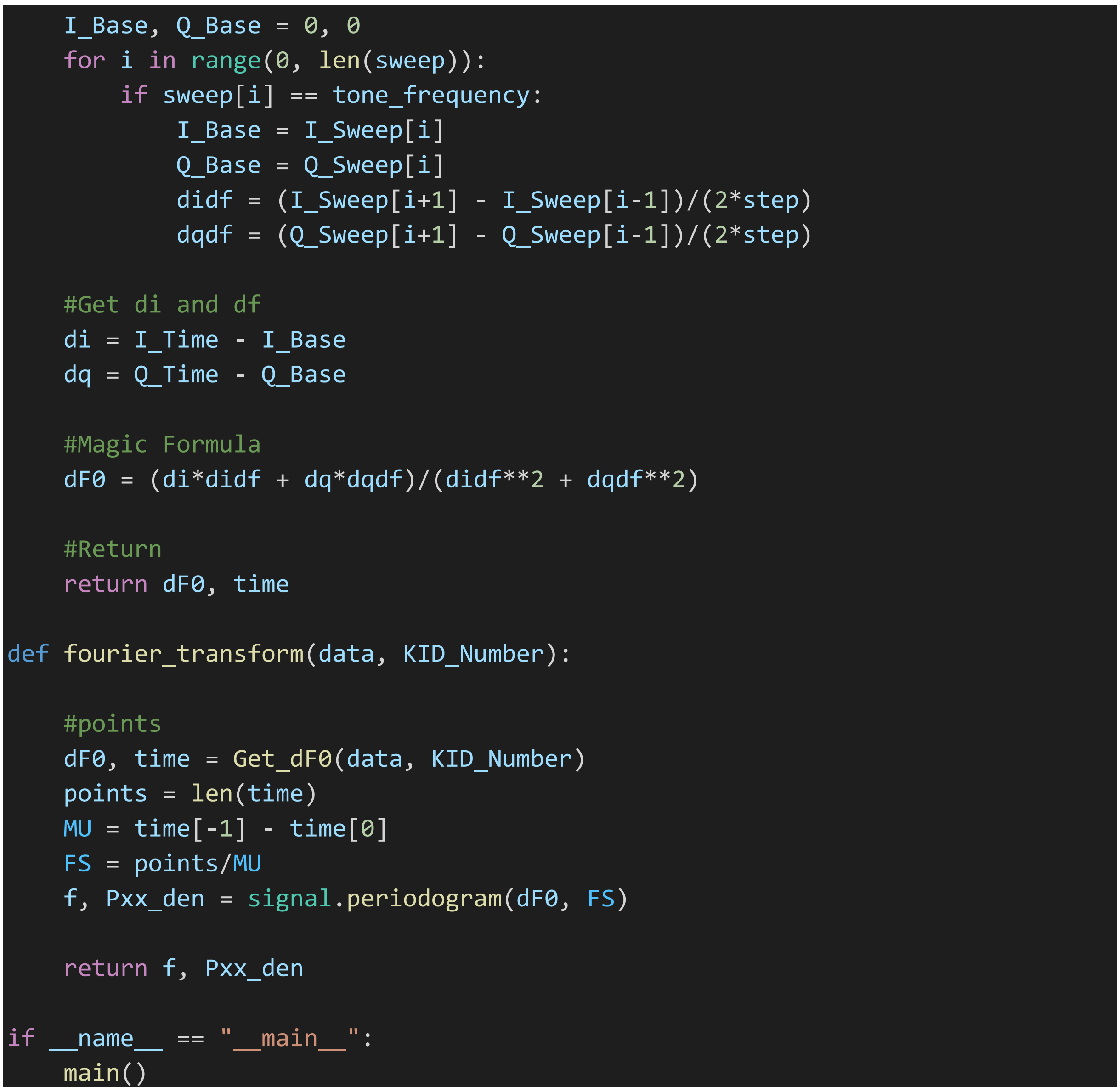
Python Code for the 6 Weeks of Data Analysis

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.special import iv as I0
from scipy.special import kv as K0
import fsab_dirfile_raw as fsab
import os
from scipy import signal
from scipy import constants as const
from scipy.interpolate import UnivariateSpline

#Constant Experiment variables
T_Hotbar = 41.3
T_Surrounding = 22.7
dT = T_Hotbar - T_Surrounding

#Hard Code User Inputs
data_folder_name = "ch1"
KID_Number = 2
lower_range = 1.5725
upper_range = 1.5875

def main():
    global KID_Number
    global lower_range
    global upper_range

    #Reading data
    local_file = os.getcwd()
    data = fsab.fsab_dirfile(local_file + "\\"  + data_folder_name)
    #Plotting Sweep
    user_input = input("Default analysis (y/n)? (KID = 2, 1.5725 < hotbar < 
1.5875)\n")

    #Get vals
    if user_input.lower() == "n":
        KID_Number, upper_range, lower_range = get_user_input(data)

    dF0, time = Get_dF0(data, KID_Number)

    plt.plot(time, dF0, 'x', color='black', markersize=1)
    plt.title("dF0 vs Time for KID number " + str(KID_Number))
    plt.ylabel("dF0 / Hz")
    plt.xlabel("time / s")
    plt.ticklabel_format(useOffset=False)
    plt.show()
    plt.figure()

    #Get range of time for peak height
    peak_range_array = np.where(np.logical_and(time>=lower_range, 
time<=upper_range))

    #Get index of upper and lower range for time
    lower_index = peak_range_array[0][0]
    upper_index = peak_range_array[0][-1]

    #Plot this range
    plt.plot(time[lower_index-30:upper_index+30], dF0[lower_index-
30:upper_index+30])

    #Curve fit
    #P_Guess
    height = 400
    mu = time[upper_index] - time[lower_index]
    sigma = 1
    c = 3000
    p_guess = [height, mu, sigma, c]

    #Gaussian curve
    popt_main, _ = curve_fit(Gaussian, time[lower_index:upper_index], 
dF0[lower_index:upper_index], maxfev=20000, p0=p_guess)
    response_main = popt_main[0]/dT


    #Plot curve_fit
    plt.plot(time[lower_index:upper_index], 
Gaussian(time[lower_index:upper_index], *popt_main))
    plt.show()
```

```python
    responsivity = list()

    #Loop for all KIDs
    number_of_KID = data.numkids

    #P_Guess
    height = 400
    mu = time[upper_index] - time[lower_index]
    sigma = 1
    c = 3000
    p_guess = [height, mu, sigma, c]
    popt=[]

    for KID_Num in range(0, number_of_KID):
        dF0, time = Get_dF0(data, KID_Num)

        #Get index of upper and lower range for time
        lower_index = peak_range_array[0][0]
        upper_index = peak_range_array[0][-1]

        #Curve Fit
        try:
            popt, _ = curve_fit(Gaussian, time[lower_index:upper_index],
dF0[lower_index:upper_index], maxfev=10000, p0=p_guess)
            #Get curve heigh
            dF0_hotbar = popt[0]

            #Get response
            response = dF0_hotbar/dT

            if abs(response) <= 100:
                responsivity.append(response)
            else:
                responsivity.append(0)

        except RuntimeError:
            responsivity.append(0)

    plt.plot(responsivity, marker='x', color='black')
    plt.title("Responsivities of all KIDs")
    plt.ylabel("Responsivity / Hz K^-1")
    plt.xlabel("KID Number")
    plt.ticklabel_format(useOffset=False)
    plt.show()
    plt.figure()

    #Spectral_densities
    frequencies, spectral_densities = fourier_transform(data, KID_Number)
    plt.plot(frequencies, spectral_densities)
    plt.title("Noise Spectral Densities vs Frequencies")
    plt.ylabel("Spectral Densities")
    plt.xlabel("Frequency / Hz")
    plt.show()
    plt.figure()

    #NET
    Noise_Eq_T = np.sqrt(spectral_densities)/response_main
    plt.plot(frequencies[1:], (Noise_Eq_T[1:]))
    plt.semilogy()
    plt.title("NET vs frequencies for KID Number " + str(KID_Number))
    plt.ylabel("NET")
    plt.xlabel("Frequency / Hz")
    plt.show()
    plt.figure()

    #Beam filling factor
    Beam_Filling_Factor = calculate_Beam_Filling_Factor(length=100, width=20,
FWHM=20, points=5000)

    #Get Transmission Factor
    transmission_frequency, transmission = np.loadtxt('MUSCAT_band.txt',
unpack=True)

    #Get dnu using fwhm
    dnu, midpoint_frequency, lower_bandwidth, upper_bandwidth =
transmission_fwhm(transmission_frequency, transmission)
    wavelength = const.c/midpoint_frequency

    #Blackbody Intensities

    blackbody_intensity_room = planck(transmission_frequency,transmission,
wavelength, T=293)
    #average power *
    #room_power =
integrate.simps(blackbody_intensity_room[lower_bandwidth_index:upper_bandwidth
_index])
    #sums = sum(blackbody_intensity_room)
    points = len(blackbody_intensity_room)
    room_power = ((sum(blackbody_intensity_room))/points)*(upper_bandwidth-
lower_bandwidth)
    room_power = room_power*wavelength**2
    print("Room optical power = " + str(room_power) + " W")

    total_photon_noise = np.sqrt(shot_noise(room_power, midpoint_frequency)**2
+ wave_noise(room_power, dnu)**2)
    print("Total Photon Noise = " + str(total_photon_noise) + " W/Hz^0.5")
```

```python
# define normalized 2D gaussian
def gaus2d(x, y, mean_x, mean_y, sigma_x, sigma_y, amplitude):
    first_frac = ((x-mean_x)**2)/(2*sigma_x*2)
    second_frac = ((y-mean_y)**2)/(2*sigma_y*2)
    return amplitude*np.exp(-(first_frac+second_frac))

def calculate_Beam_Filling_Factor(length=100, width=20, FWHM=20, points=5000):
    #Define x and y
    x = np.linspace(0, length, points)
    y = np.linspace(0, length, points)
    x, y = np.meshgrid(x, y) # get 2D variables instead of 1D

    #mean: square box of (longest length)/2
    mean = [length/2, length/2]

    #sigma = FWHM/(sqrt(8log2))
    divisor = np.sqrt(8*np.log(2))
    sigma = [FWHM/divisor, FWHM/divisor]
    z = gaus2d(x, y, mean[0], mean[1], sigma[0], sigma[1], 1)

    #Integral = sum over all points
    Gauss_integral = np.sum(z)

    #Bar dimensions centered at length/2
    sub = width/2
    first_cuttoff = int(((mean[0]-sub)/length)*points)
    second_cuttoff = int(((mean[0]+sub)/length)*points)

    #Set outside hotbar dimensions = 0
    z[0:first_cuttoff,:] = 0
    z[second_cuttoff:,:] = 0

    #Integrate over all points
    hotbar_integral = np.sum(z)

    #Calculate BFF
    Beam_Filling_Factor = hotbar_integral/Gauss_integral

    return Beam_Filling_Factor


def shot_noise(power, nu):
    shot = np.sqrt(2*power*const.h*nu)
    return shot

def wave_noise(power, dnu):
    wave = power/(np.sqrt(2*dnu))
    return wave
```

```python
def planck(frequency,transmission, wavelength, T=293):
    a = (2*const.h*frequency**3)/const.c**2
    b = 1/(np.exp((const.h*frequency)/(const.Boltzmann*T))-1)
    #intensity = a*b*transmission*wavelength**2
    intensity = a*b*transmission
    return intensity

def transmission_fwhm(x, y):
    # create a spline of x and freq-np.max(blue)/2 to find fwhm
    spline = UnivariateSpline(x, y-np.max(y)/2, s=0)
    r1, r2 = spline.roots() # find the roots
    fwhm = abs(r2 - r1)
    midpoint = fwhm/2 + r1

    plt.plot(midpoint/1e9, 0, label="Midpoint", marker = "x", color = "blue",
markersize = 4)
    plt.plot(x/1e9, y, marker = ".", color = "black", label = "data",
markersize = 1)
    plt.axvspan(r1/1e9, r2/1e9, facecolor='g', alpha=0.5, label="fwhm")
    plt.title("Transmission % vs Frequency")
    plt.xlabel("Frequency / GHz")
    plt.ylabel("Transmission Percentage")
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fancybox=True)
    plt.show()

    r1 = x[0]
    r2 = x[-1]
    return fwhm, midpoint, r1, r2

#navigate to which file. Default analysis is ch1 for hot bar data
def get_user_input(data):
    #Get Freq
    data_folder_name = input("Data file: ")
    local_file = os.getcwd()
    #Uncomment for use:
    #data_folder_name = input("Data file: ")
    data = fsab.fsab_dirfile(local_file + "\\"  + data_folder_name)

    #Plotting Sweep
    while True:
        Input = input("Know the range? (y/n):")
        if Input.lower() == "y":
            break
        KID_Number = int(input("KID Number?: "))

        #Plot dF0 v Time
        dF0, time = Get_dF0(data, KID_Number)
        plt.plot(time, dF0)
        plt.rcParams["figure.dpi"] = 400
        plt.ticklabel_format(useOffset=False)
        plt.xlabel("time / s")
        plt.ylabel("dF0 / Hz")
        plt.title("Time evolution of dF0 for KID number " + str(KID_Number))
        plt.show()
        plt.figure()

        Input = input("Again? (y/n):")
        if Input.lower() == "n":
            break

    #Get dF0 and Time
    KID_Number = int(input("KID Number?: "))

    lower_range = float(input("Lower bound of time peak range?:"))
    upper_range = float(input("Upper bound of time peak range?:"))

    return KID_Number, upper_range, lower_range

def Gaussian(x, height, mu, sigma, c):
  f = c + height*np.exp((-(x-mu)**2)/(2*(sigma)**2))
  return f

def Get_IQ_Sweep(data, KID_Number):
    IQ_data = data.sweep[KID_Number]["z"]
    sweep = data.sweep[KID_Number]["f"]
    tone_freq = data.sweep[KID_Number]["tone_freq"]
    I = IQ_data.real
    Q = IQ_data.imag
    return I, Q, sweep, tone_freq

def Get_IQ_Time(data, KID_Number):
    t = (data.start_time - data.stop_time)
    IQ_data = data.get_iq_data(KID_Number)
    I = IQ_data.real
    Q = IQ_data.imag
    time = np.linspace(0,t, len(I))
    return I, Q, time

def Get_dF0(data, KID_Number):
    #Initialize data
    I_Sweep, Q_Sweep, sweep, tone_frequency = Get_IQ_Sweep(data, KID_Number)
    I_Time, Q_Time, time = Get_IQ_Time(data, KID_Number)

    #Get didf and dqdf
    step = sweep[1] - sweep[0]
```



```python
    I_Base, Q_Base = 0, 0
    for i in range(0, len(sweep)):
        if sweep[i] == tone_frequency:
            I_Base = I_Sweep[i]
            Q_Base = Q_Sweep[i]
            didf = (I_Sweep[i+1] - I_Sweep[i-1])/(2*step)
            dqdf = (Q_Sweep[i+1] - Q_Sweep[i-1])/(2*step)

    #Get di and df
    di = I_Time - I_Base
    dq = Q_Time - Q_Base

    #Magic Formula
    dF0 = (di*didf + dq*dqdf)/(didf**2 + dqdf**2)

    #Return
    return dF0, time

def fourier_transform(data, KID_Number):

    #points
    dF0, time = Get_dF0(data, KID_Number)
    points = len(time)
    MU = time[-1] - time[0]
    FS = points/MU
    f, Pxx_den = signal.periodogram(dF0, FS)

    return f, Pxx_den

if __name__ == "__main__":
    main()
```