NOVEL ANALYSIS FRAMEWORK USING QUANTUM OPTOMECHANICAL READOUTS FOR DIRECT DETECTION OF DARK MATTER

by

Ashwin Nagarajan

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Aeronautics and Astronautics

School of Aeronautics and Astronautics

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dr. Rafael Lang, Chair

School of Physics and Astronomy

Dr. Kathleen Howell

School of Aeronautics and Astronautics

Dr. James Longuski

School of Aeronautics and Astronautics

Approved by:

Dr. Gregory Blaisdell

Dedicated to all

ACKNOWLEDGMENTS

My sincere thanks to Dr. Rafael Lang for agreeing to mentor and supervise this project, and to Dr. Kathleen Howell and Dr. James Longuski for agreeing to be part of the advisory committee. Furthermore, my heartfelt thanks to the entire Windchime collaboration, which is the basis of this thesis, right from the software inputs and framework suggestions of Bahaa Elshimy, Juehang Qin, and Vinay Samuel, to the hardware work along with Juan Pablo, Shengchao Li, Mayukh Bagchi, Qingyang Li., and Rebecca Hackett. My thanks also to the other members of the team for their inputs along the way, including Abigail Kopec, Amanda Depoian, and Michael Clark, and the remaining members of the Windchime collaboration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the increase in speculation about the nature of our universe, there has been a growing need to find the truth about Dark Matter. Recent research shows that the Planck-Mass range could be a well-motivated space to probe for the detection of Dark Matter through gravitational coupling. This thesis dives into the possibility of doing the same in two parts. The first part lays out the analysis framework that would sense such an interaction, while the second part outlines a prototype experiment that when scaled up using quantum optomechanical sensors would serve as the skeleton to perform the analysis with.

1. INTRODUCTION

Ever since Lord Kelvin gave an estimate [1] on the number of dark bodies in the Milky Way after careful observation of the velocity dispersion of the stars in orbit around the center of the galaxy, the search for Dark Matter has been a holy grail in modern physics. This chapter outlines some of the history followed by the modern methods of detection, before highlighting the core principle of this thesis.

1.1 History

Through the 20th century, there was growing interest in this hypothesis, but without much actual evidence. This changed in 1922, when Dutch astronomer Jacobus Kapteyn suggested the existence of Dark Matter [2] after studying stellar velocities which was soon corroborated within a decade by fellow Dutch astronomer Jan Oort who studied stellar motions [3] in the local galactic neighborhood.

Fast-forwarding to the 1960s and 1970s, Vera Rubin, Kent Ford, and Ken Freeman strengthened the argument [4] for the existence of Dark Matter using galaxy rotation curves. Rubin and Ford later went on to publish another influential paper [5] in the 1980s that brought to light the fact that Dark Matter was a major unsolved problem in modern astronomy.

1.2 Modern Approaches

Primary motivations for proving the existence of Dark Matter come from indirect observations. While there has a plethora of theories proposed and many more experiments conducted to validate the same, one of the key insights to the existence of Dark Matter was obtained from the application of the Virial Theorem [6] to mass distribution in bound systems. Fritz Zwicky was one of the first [7] to use the Virial Theorem to deduce the presence of what he referred to as “unseen matter” at that time. Plotting the velocity dispersion estimates in elliptical galaxies, it was observed that there was a mismatch between the expected and actual rotation curves, as shown below.

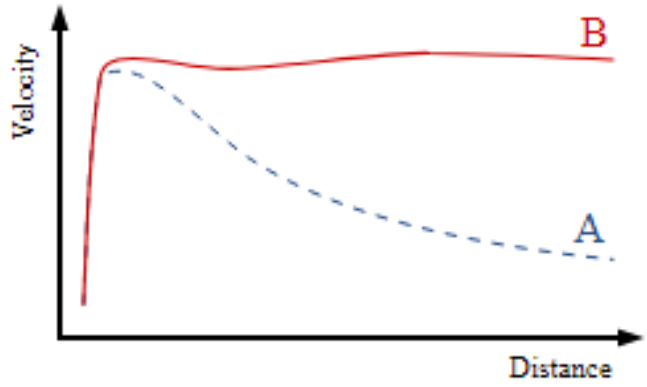

Figure 1.1. The figure above depicts the general relation between velocity and distance. Curve A is the predicted relation between the two, while Curve B is the observed dependence. The above plot confirms the analysis by Oort and Zwicky.

Another important case for the existence of Dark Matter comes from a region hosting the collision of two clusters of galaxies, known as the Bullet Cluster. After a time when Modified Newtonian Dynamics [8] was proposed and gaining traction to explain the observed properties of galaxies, the Bullet Cluster proved to be a nail in the coffin against it.

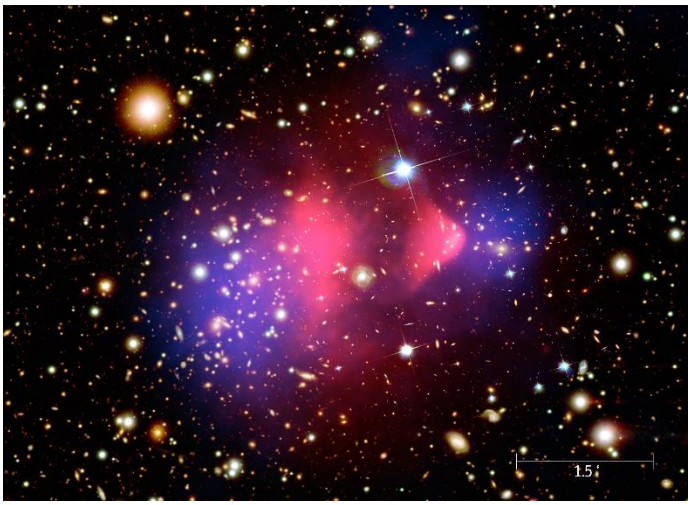

Figure 1.2. A composite image of the Bullet Cluster, with the X-Ray portion in pink against backdrop of visible light data; blue indicates the matter distribution estimated from gravitational lensing

With a statistical significance of 8σ, this showed that the spatial offset of the center of total mass from the center of the baryonic mass of this region could not successfully by explained just with a modified theory of gravity alone.

The case for Dark Matter was further strengthened when standard models could easily explain the observations [9] and were model-independent. Following these theoretical proposals, the Planck Mission launched by the European Space Agency and the Wilkinson Microwave Anisotropy Probe gave insights [10] on the possible composition of the universe, which can be tabulated as follows:

Table 1.1. Percentage composition of the universe in terms of its mass-energy density.

Type	**Percent**
Dark Energy	70%
Dark Matter	25%
Baryonic Matter	5%

This leads to the question of how one goes about with the detection of the above quantities. Here, the focus is on Dark Matter.

1.3 Detection Methods

There are many suggested ways of detecting Dark Matter. These can broadly be divided into three main categories: Collider Searches, Indirect Detection, and Direct Detection.

1.3.1 Collider Searches

In this method, Dark Matter is attempted to be produced within the confines of a laboratory. Similar to how the Large Hadron Collider [11] probes the space of particle physics, a setup could detect Dark Matter particles in terms of missing energy and momentum from the collisions that do not register on the detectors of the experiment. A similar experiment, called the Large Electron-Positron Collider, also set limits [12] by probing the interaction of Dark Matter particles with electrons instead of quarks.

1.3.2 Indirect Detection

In this approach, secondary effects of Dark Matter are the main goal. This could include self-annihilation or decay of Dark Matter particles. In regions such as the center of galaxies, where the Dark Matter density is supposed to be high, two Dark Matter particles could annihilate, producing particle-antiparticle pairs, or one unstable particle could decay into Standard Model particles. These processes would upset the balance of gamma rays, antiprotons, positrons, or other particles in the region, revealing [13] the presence of Dark Matter.

A few Dark Matter particles passing through the Earth may lose energy by scattering off atoms. Dark Matter could accumulate at the center of such particles, thereby increasing the probability of collision or annihilation. This could lead to the formation of a signal in the form of high-energy neutrinos. Such an event would strengthen the possibility of a Weakly Interacting Massive Particle (WIMP), which is what high-energy neutrino telescopes [14] such as the Antarctic Muon and Neutrino Detector Array (AMANDA), IceCube, and Astronomy with a Neutrino Telescope and Abyss environmental Research project (ANTARES), are looking for. Recently, IceCube discovered [15] the presence of high-energy astrophysical neutrino flux which could be attributed to Dark Matter decay.

1.3.3 Direct Detection

This tactic is the most important in the context of this thesis. Direct detection involves observing the recoils of a chosen nuclei induced by the Dark Matter candidate. In order to keep the background interactions to a minimum, such experiments are usually conducted deep under the surface of Earth. Some examples of such experiments are Stawell Mine, the Gran Sasso National Laboratory, the Boulby Underground Laboratory, and the Deep Underground Science and Engineering Laboratory. In order to keep the reactivity down, these detectors use cryogenic or noble liquid inside their detectors. For example, the Gran Sasso National Laboratory in Italy uses Xenon to distinguish the background particles that scatter off electrons, from the Dark Matter particles [16] that scatter off nuclei.

A new and crucial approach is the direct detection by leveraging [17] the gravitational interaction of Dark Matter. The crux of this thesis was presented [18] in the last couple of years, coupling gravitational direct detection with quantum optomechanics.

Earlier this decade, Laura Baudis enlisted methods [19] to achieve direct detection using apparatus on Earth. Baudis gives a range of 10^{-22} eV to 10^{15} GeV for the possible masses. These 46 orders of magnitude could be expanded to 60 if one approaches the subject in terms of interaction strength with Standard Model particles. Building on the detection of WIMPs, this paper states that direct detection experiments placed deep underground aim to observe a weak scattering signal between a WIMP and a chosen atomic nucleus where the background noise from natural radioactivity is kept to a minimum.

The focus is brought back to the potential detection with quantum optomechanics, starting with building the analysis framework before ideating a small-scale prototype for experimental verification.

1.4 Gravitational Direct Detection

The globally accepted notion of the presence of Dark Matter has not yet meant the confirmed existence of the same, and while other methods may prove fruitful, probing the gravitational interaction of Dark Matter may bring rewards sooner than later.

The quantum angle of the detection is to have an array of close to a billion sensors in the micro-to-milligram scale, within a lattice of spacing in the millimeter-to-centimeter range. These mechanical resonators are sensitive to optical light, and use quantum noise-evading measurement protocol, to give readouts of the mechanical position of these sensors.

Laser interferometers [20] could be used to infer the presence of Dark Matter by scanning for macroscopic objects that interact gravitationally with the setup. If the size of the Dark Matter candidate is limited to be smaller than the physical dimensions of a given detector, a correlation between a range of given masses and the corresponding gravitational effect can be produced. This is the key idea behind this thesis, and steps to achieve the eventual goal are enlisted at a smaller scale.

2. METHODS

This section details the various steps required to build this analysis framework. They are broadly divided into a section on the initial calculations to establish mass limits, followed by one on setting up the virtual array of sensors, before finally outlining the a prototype in the works to eventually realize the experiment in its physical form.

2.1 Mass Range Calculations

Despite the methods suggested above for several years, there has so far been no confirmed detection [21] of Dark Matter in the neighborhood of Earth. While most of the attempts have been using the interaction of Dark Matter with non-gravitational forces which puts the mass range in the relatively light spectrum, gravitational direct detection hopes to probe the mass range on the heavier side, as shown below.

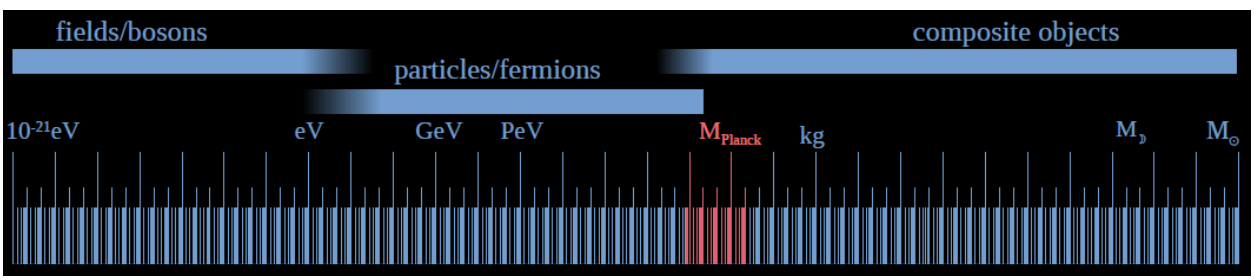

Figure 2.1. This figure shows the total range of masses available to probe in blue, while the red region indicating M_{Planck} is the slice pertaining to gravitational direct detection

Next, given the mass range to be looked at, some insight must be gained to predict the probability of such an interaction with the detector occurring. In order to accomplish that, consider the following values [22] which are taken to be roughly constant:

$$\rho_{DM} = 0.3\ GeV/cm^3 \tag{1}$$

$$m_{PL} = 1.22 \times 10^{19}\, GeV/c^2 \tag{2}$$

$$v = 220\, km/s = 0.75 \times 10^{-3} c \tag{3}$$

$$v_{esc} = 544\ km/s \tag{4}$$

$$r = 5.2m = 520cm \tag{5}$$

This gives:

$$\sigma = \pi r^2 = 84.95\, m^2 = 8.49 \times 10^5 cm^2 \tag{6}$$

$$n = \frac{\rho_{DM}}{m} = \frac{0.3}{m} cm^{-3} \tag{7}$$

$$\phi = n \times v = \frac{0.3}{m} cm^{-3} \times 22 \times 10^6\, cm/s = \frac{66}{m} \times 10^5 cm^{-2} s^{-1} \tag{8}$$

$$F = \phi \times \sigma = \frac{66}{m} \times 10^5 cm^{-2} s^{-1} \times 8.49 \times 10^5 cm^2 = \frac{5.6066}{m} \times 10^{12} s^{-1} \tag{9}$$

Where:

$$\rho_{DM} \equiv Dark\ Matter\ Density$$

$$m_{PL} \equiv Mass\ of\ Dark\ Matter\ Candidate$$

$$v \equiv Velocity\ Function\ f(v)\ and\ velocity\ relative\ to\ target$$

$$v_{esc} \equiv Escape\ Velocity$$

$$r \equiv Radius\ of\ Detector$$

$$\sigma \equiv Cross\ Sectional\ Area$$

$$n \equiv Number\ Density$$

$$\phi \equiv Mass\ Flux$$

$$F \equiv Particle\ Flux$$

This can also be visualized in the following log-log plot:

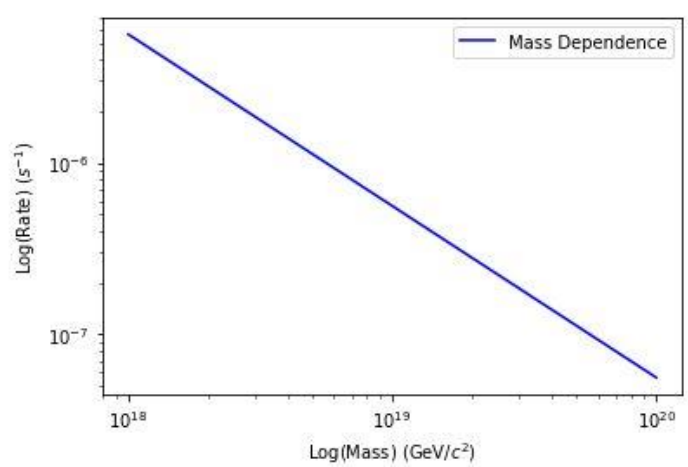

Figure 2.2. Relation between mass of the Dark Matter candidate and transit rate across a given detector geometry and orientation

The above calculations show that, for the given size of detector, assuming it is a bounding sphere through which a track of Dark Matter particle would pass through, and that the chosen candidate is of the Planck Mass scale of 10^{19} GeV/c^2, there would be roughly 1~3 events per year.

2.2 Array of Detectors

The setup includes an array of force sensors, which is sensitive to the various background forces it would be subject to. A Dark Matter particle passing through this arrangement would exert a correlated, albeit small, force on the sensors of the array that are nearest to its trajectory. This correlated force signal can then be selected out of the remaining noise, revealing the presence of the Dark Matter particle. The advantage with this approach is that, since the data measures the track of the incoming Dark Matter particle, directional information is preserved.

The quantum optomechanics part of the detection provides limits on the sensitivity [23] of these detectors, which is in turn limited by the noise from the thermal motion of the detectors, and limits inherently present in making such measurements. The array of detectors would roughly look like the diagram below:

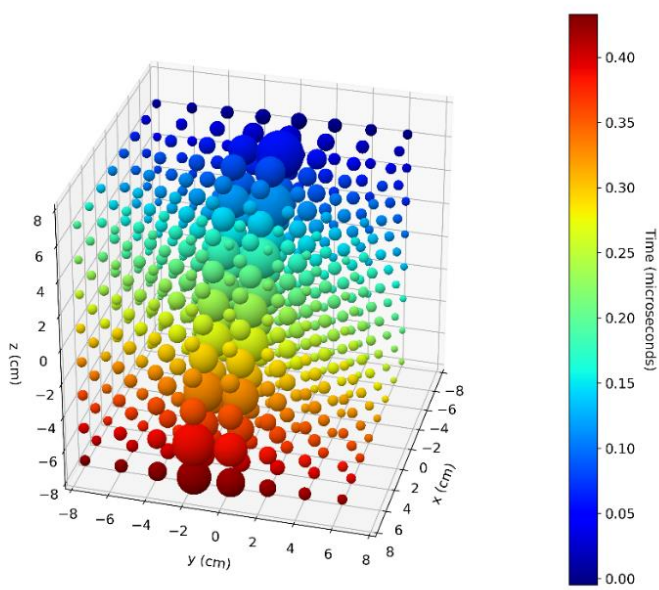

Figure 2.3. The above figure shows a simulated array of force detectors with the bigger spheres indicating the passing of track; the colors aid in visualizing the passage of time and thereby give directional information of the orientation and approach of the track

The next step involves creating an analysis framework that can perform the task of detecting the fluctuations in the force sensors produced by the incoming track. The results of this simulation can be found in the next chapter.

2.3 Building a Prototype

While the actual experiment would be large-scale with close to a billion sensors for precise measurement, for the purpose of this thesis, discussion will be about setting up a small-scale proof-of-concept.

The basic idea is to have an array of accelerometers on their evaluation boards arranged on a mechanical board, which is in turn setup of a laboratory table. These accelerometers are wired up with coaxial cables, and the signal cables are connected to an Analog to Digital Converter (ADC) to process an incoming input and convert it to a digital signal. The setup is encased in a grounded housing for purposes of shielding. Additionally, a manifold distributes power from the main power supper to the array of accelerometers and the ADC, with as efficient a setup as possible to keep the noise to a minimum. LabVIEW is then used to build the interface that would work with this setup to obtain a signal, read, and process it, and write the desired output in a file of suitable format.

For the purposes of a basic test, measuring the speed of sound was picked, as it is relatively simpler than other measurements, with a form of plastic as the material, as the speed of sound in plastics is ideal for measurement as compared to metals, without the setup needing to be large in size.

In the next chapter, results from the simulation and progress on the prototype are discussed.

3. RESULTS

In this chapter, plots of the simulation and images of the prototype setup are shown, which will then be discussed in the subsequent chapter.

3.1 Analysis

Using code (Appendix A) in Python, an 4x4x4 array of values was created which would serve as the placeholder for the detectors in this virtual space. A velocity that is randomly generated based on the standard halo model of Dark Matter in the neighborhood of Earth is used. A custom track 7.5 meters in length was generated and run through this array. The simulation then randomizes the number of samples taken before the track intersects the bounding sphere, so as to make the entry time random. The true values of entry and exit time are then recorded, along with the truth values of the entry and exit angles for spatial information, and velocity for plotting the relations between these parameters.

With these initial conditions of space, time, and velocity, an integral transform was performed to obtain the Signal-to-Noise Ratio (SNR) and the associated four-vectors of 3 spatial coordinates and time for each of the templates in this analysis. For the purposes of this thesis, the analysis is performed without taking noise into consideration, so the plots work with the S value or signal strength, instead of SNR value.

Selecting the pertinent data from all the output (Appendix A), here are the entry and exit values:

Table 3.1. The following table lists the entry and exit truth values and calculated values of one instance of this simulation, with a sample track running through the array of sensors

Quantity	Truth Value	Calculated Value
Entry Time	384.5 μs	383 μs
Exit Time	396.98 μs	399.06 μs
Entry Phi	1.44 rad	1.44 rad
Exit Phi	0.17 rad	0.11 rad
Entry Theta	1.49 rad	1.49 rad
Exit Theta	1.71 rad	1.79 rad

While this shows that the track is interacting with the detector and the signal can be recovered accurately, there is no visual representation of what is going on. In order to do that, plotting these values against each other is necessary. However, single-variable plots are not enlightening, as they only have a peak around the truth value, and the relation between the parameters is not obvious. For this purpose, a corner plot was produced as shown below, which covers the complete phase space of all the parameters involved in the analysis.

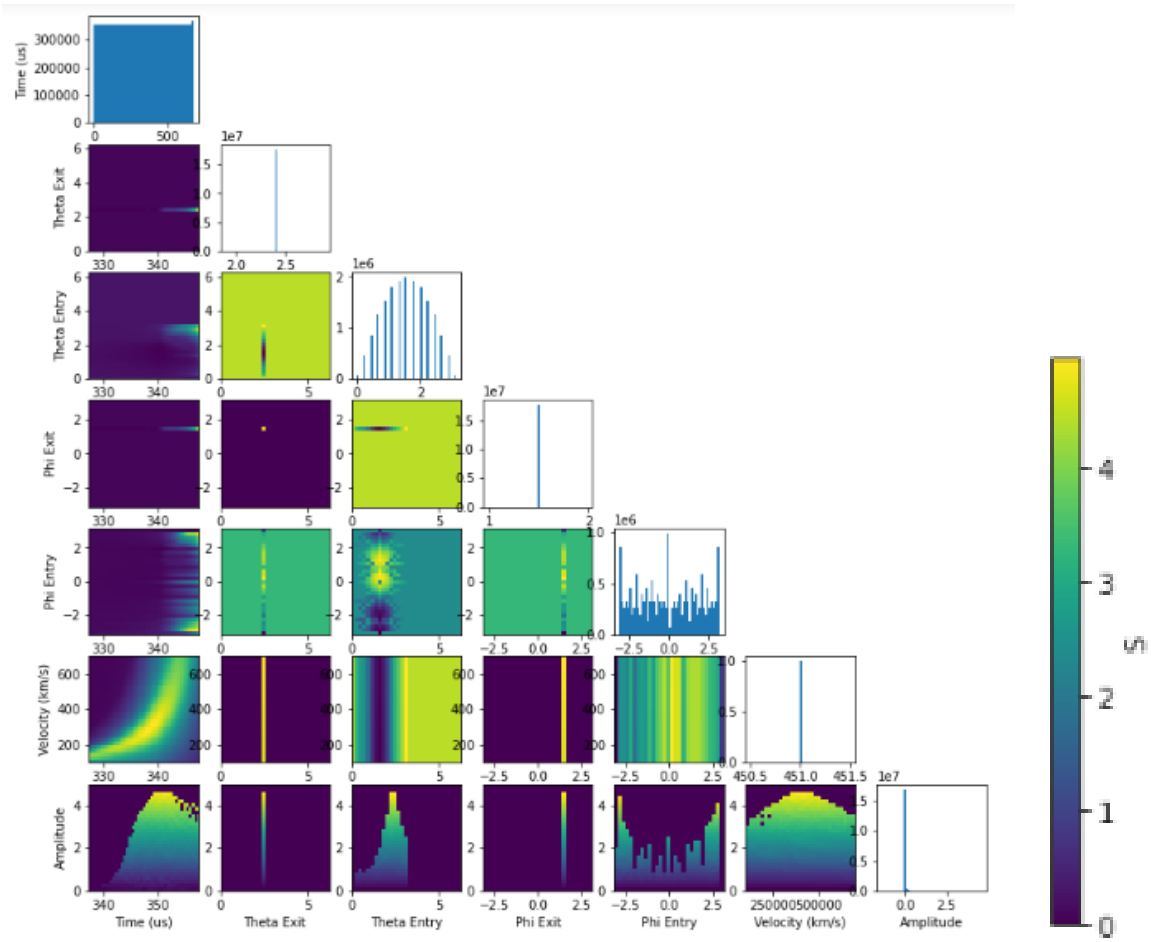

Figure 3.1. The plot above shows the the corner plots between the parameters involved in this analysis, with the color bar indicating the signal strength

While the plot is in line with what is to be expected, it is to be noted that currently, the framework can only analyze the phase space at either the exit time alone, or entry time alone. This is evident in the above plot having one-dimensional results for the exit subplots, since this was obtained using the velocity, time, and spatial entry values, without the spatial exit values.

3.2 Prototype

On the hardware side of things, a simple setup is in the process of being built in order to detect the speed of sound. The following accelerometer was purchased from Analog Instruments for this purpose.

Figure 3.2. Image of ADXL1005 (black) in the evaluation board (green) used in the setup of an array of accelerometers

This accelerometer has a resonant frequency of 42 kHz and provides an analog output proportional to a mechanical vibration between 100 Hz and 23 kHz. The bottom of the accelerometer shows the direction in which it is sensitive to movement, which is nothing but the axis along which the vibration is detected and measured. This accelerometer has a sensitivity of 20 mV/g, which means that 20 mV will be measured for every 9.8 m/s^2 that the accelerometer feels in its proper rest frame, as shown below.

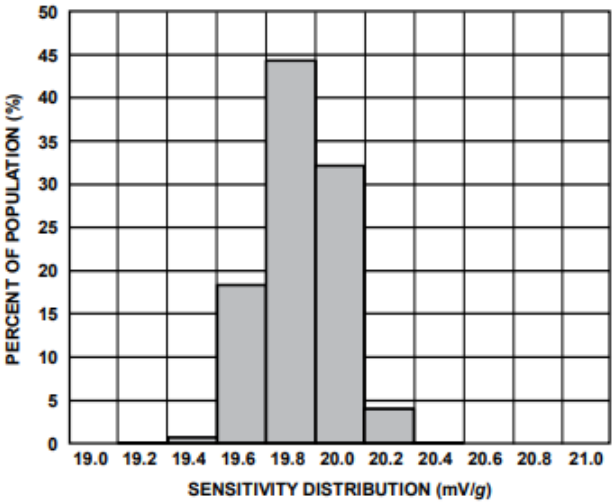

Figure 3.3. Sensitivity behavior of the accelerometer, obtained from its data sheet

Fifteen of these accelerometers are then wired to a mechanical board for the eventual measurement of the speed of sound. Before doing that, the ability of the software to detect this accelerometer needs to be tested. For this purpose, the accelerometer was wired to an oscilloscope, which was the Keysight InfiniiVision DSOX3012A Oscilloscope in this case. The accelerometer was kept in place using a L-shaped piece of plexiglass. These components were wired up with a BNC to BNC cable and a BNC adaptor. To test that the accelerometer is sensitive to a range of frequencies, a transducer, namely the COM-10975 Surface Transducer, and a pulse generator were used. The setup can be seen below.

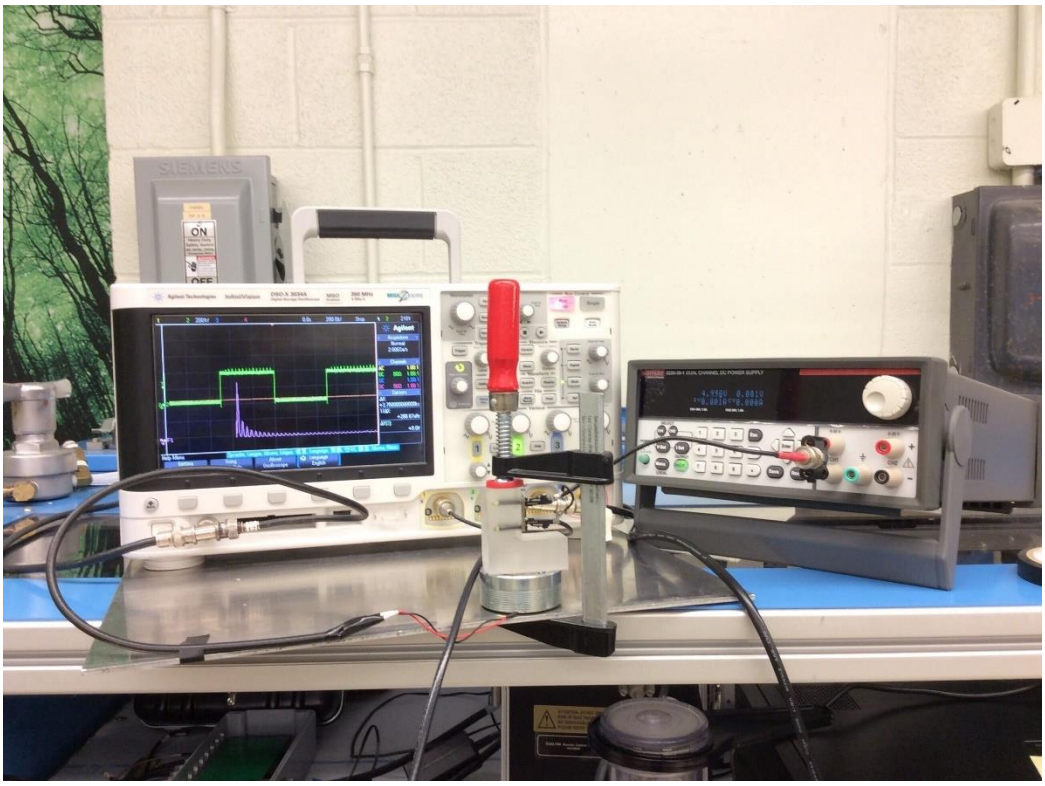

Figure 3.4. This figure shows the initial setup of one accelerometer wired to the oscilloscope and power supply, along with the transducer

As seen in the picture, the system can recover the input signal at a given frequency, verified by comparing with the Fast Fourier Transform (FFT) which peaks at the input frequency. In the above picture, the green line is a square wave, while the purple is the FFT corresponding to that.

4. DISCUSSION

In this chapter, some of the results are discussed. In the analysis section, selected subplots from the corner plot are explained, whereas in the prototype part, discussion moves to how the system can be used to achieve the goal of interest.

4.1 Analysis

Looking at the corner plots, it is evident that the parameters in the phase space have a trend consistent with expectation. In particular, the velocity-time plot shown below is one that shows that the signal can indeed be recovered using this integral transform analysis.

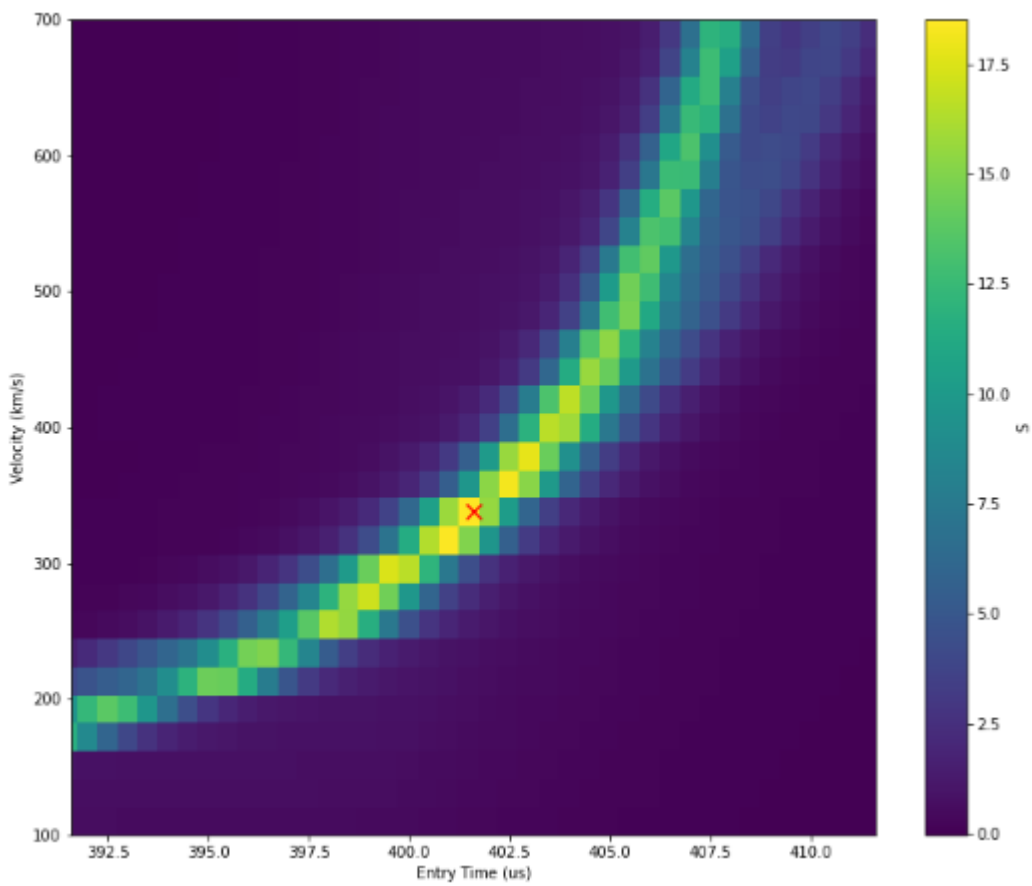

Figure 4.1. The above figure shows a Velocity-Time with the red cross indicating the signal that can be recovered from this integral transform; the plot was for an entry time of 401.5 µs and exit time of 424.35 µs

Moreover, in line with expectation, the plots of parameters against themselves are histograms, with a peak near the truth value. These are placed along the diagonal in the corner plot.

As mentioned before, since the algorithm is only capable of analyzing either the entry or the exit at a given time, and since this particular corner plot is with the entry, the exit plots are one-dimensional.

One other point to be noted is that the final row of the corner plot uses the signal strength as the amplitude, which is why it has a histogram-like appearance. In an improved model, this amplitude should be the interaction strength, which would be the range of input masses. Numerically, this could be the Universal Gravitational Constant G multiplied by the mass of the Dark Matter candidate.

4.2 Prototype

On the hardware side, what the setup currently shows is that given an input vibration, which in this case is provided by the transducer, the accelerometer can detect the signal and display an output to the screen of the oscilloscope. Using LabVIEW, a Field Programmable Gate Array (FPGA) target can be created, where one can add a Virtual Instrument (VI) and code a particular set of commands for the program to execute.

Data is written in two streams, a fast-data set, and a slow-data set. The former would store the values of the frequencies sampling at MHz, while the latter is mainly for measurements of the setting, such as temperature and humidity.

5. CONCLUSION

As seen in previous chapters, this thesis lays out a basic framework in eventually achieving an experimental setup that could measure the presence of a Dark Matter particle.

The analysis portion discusses a virtual array of sensors that uses an integral transform to recover a signal passing through. Some developments to be made include making the code work with both entry and exit parameters at the same time to obtain a full-blown two-dimensional correlation. Moreover, running the analysis over typical mass ranges would make the corner plot more complete, while providing insight of the potential Dark Matter candidate.

The prototype section suggests a basic setup of accelerometers connected to a system of ADC and a transducer, to measure the speed of sound through a material, using LabVIEW to read the input signals and write them in a binary format onto a file. Going forward, the present setup using one accelerometer needs to be expanded to about 15 on a mechanical board. This would serve as a small mock-up of an eventual large scale detector with billions of sensors that use quantum optomechanics to sense the extra perturbation produced by a Dark Matter particle passing through.

APPENDIX A. CODE

CODE

The following is the code that was used to obtain the results above. It makes use of an integral transform to recover a signal. This can be thought of as a template-matching scheme that tries to match different signals distributed in a higher-dimensional space. It can also be visualized as a modified four-dimensional x-ray transform that takes an inverse-square distribution into account.

This code demonstrates the analysis framework based on the integral transform with simulated data. Using a 4x4x4 grid of sensors, the framework recovers a signal that is not trivially visible.

```
from planckmc.track_generation import halo_model
from planckmc.track_generation import halo_model
from planckmc.track_generation import make_tracks
from planckmc.detector_characteristics import
DETECTOR_CHARACTERISTICS
from planckmc.response import sensor_response, RESPONSE_DICT
from planckmc.config import CONFIG
import planckanalysis.separated_integral_transform as pint

import numpy as np
from numba import njit, jit
from tqdm import tqdm
import numericalunits as nu
from scipy.signal import peak_widths
from scipy.signal import peak_prominences
from random import randrange, uniform
import json

%matplotlib inline
```

```
import matplotlib # plotting libraries

import matplotlib.pyplot as plt
import matplotlib.colors as clr
from matplotlib.colors import LogNorm
from mpl_toolkits.mplot3d import Axes3D

import os.path
import time
from multiprocessing import Pool, Manager

#import seaborn as sn
#%pdbfrom planckmc.track_generation import halo_model
from planckmc.track_generation import halo_model
from planckmc.track_generation import make_tracks
from planckmc.detector_characteristics import
DETECTOR_CHARACTERISTICS
from planckmc.response import sensor_response, RESPONSE_DICT
from planckmc.config import CONFIG
import planckanalysis.separated_integral_transform as pint

import numpy as np
from numba import njit, jit
from tqdm import tqdm
import numericalunits as nu
from scipy.signal import peak_widths
from scipy.signal import peak_prominences
from random import randrange, uniform
import json

%matplotlib inline
import matplotlib # plotting libraries
```

```
import matplotlib.pyplot as plt
import matplotlib.colors as clr
from matplotlib.colors import LogNorm
from mpl_toolkits.mplot3d import Axes3D

import os.path
import time
from multiprocessing import Pool, Manager

#import seaborn as sn
#%pdb
scl_fct = 1e4 # Scale Factor for Optimized Time variables (entry
time and time difference)
hnds_un = 1e-2 # Units to Hundreds -> Because Some values are
already multiplied by one hundred (i.e. every unit is one
hundred of the value)
tnsMS_ms = 1e-1 # Tens of microseconds to microseconds
s_ns = 1e9 # Seconds to nanoseconds
s_ms = 1e6 # Seconds to microseconds
```

Simulation Accelerations and Definition of Truth Values

First, a velocity is randomly generated based on the standard halo model. Then, a track that intersects the detector is generated. The simulation code is asked to start recording a random number of samples (between 50 and 200) before the track enters the bounding sphere of the detector, so as to make the time at which the event starts random

```
#Custom Track: 7.5 m in length
#vel, entry_vecs, exit_vecs, t_entry, t_exit =
np.array([257500.]), np.array([[2.10893786], [3.19662096],
[3.51766905]]), np.array([[-3.94093443], [ 3.26567685], [-
0.91890723]]), np.array([0.]), np.array([2.91362915e-05])
```

```
#Randomized Track:
vel = halo_model.generate_vel_array(n_vels=1) # The number
infront of n_vels is how many tracks are produced, Keep at one
for analysis
entry_vecs, exit_vecs, t_entry, t_exit =
make_tracks.generate_tracks(vel, np.zeros(vel.shape))

#t_entry, t_exit are in seconds, the space variables are in
meters

track=0 # Identifies that we are looking at the first track,
number of tracks is equal to n_vels
radius = float(CONFIG['Track
Generation']['BoundingSphereRadius'])
track_len = np.linalg.norm(exit_vecs - entry_vecs)
adc_timestep_size = int(CONFIG['Track
Generation']['Timestep'])/s_ns

sensors = tuple(DETECTOR_CHARACTERISTICS.keys()) # Loads the
sensor information

# Make sure the linear response in RESPONSE_DICT (simulation) is
the same as the one used in the analysis (Run Response_update)
# Eventually, the linear response and the FIR filter used for
the template generation, which is currently this same one, will
be the same
# as that of the sensors used in the physical experiment. By
matching the two linear responses and FIR filters together, the
matching can work

lin_resp = RESPONSE_DICT[sensors[0]]['linear_response']
```

```
lin_resp_len = len(lin_resp)

# The floor value is important because of two reasons: 1) it
provides a way to analyze for entry time, 2) it makes the length
of the data longer than
# the response length (which is CRUCIAL)
floor_val = ((lin_resp_len - (((t_exit - t_entry) * hnds_un *
s_ns) + 2)) // 2) + 1

# Padding is in 10s of microseconds (nanoseconds/100)
if floor_val > 1:
    n_pad_strt = randrange(floor_val, floor_val + 5000)
    n_pad_end = randrange(floor_val, floor_val + 5000)
else:
    floor_val = 1
    n_pad_strt = randrange(floor_val, floor_val + 5000)
    n_pad_end = randrange(floor_val, floor_val + 5000)

# Acceleration without noise:
out = make_tracks.generate_acceleration_dict(entry_vecs,
exit_vecs, t_entry, t_exit, {'M':1e8, 'G':6.67e-11},
strt_padding=n_pad_strt, end_padding=n_pad_end)

accels = []
sensors_pos = []
cnt = 0

# Below we initialize for the acceleration array and the sensor
positions array
for key in out[track]:
    if key not in ['time', 'particle_location']:
        accels.append(out[track][sensors[cnt]])
```

```
sensors_pos.append(DETECTOR_CHARACTERISTICS[key]['position'])
        cnt += 1

## To make a 'zero track', uncomment the line below:
#accels = np.zeros(np.array(accels).shape)

# Below we generate the corresponding truth angle spatial
information of the track from the cartesian truth data:
alpha_theta = np.array([0, 0, 1]) # Z-axis
cos_theta_entry = pint.py_ang(entry_vecs.T[0], alpha_theta.T)
if cos_theta_entry < -1:
    cos_theta_entry = -1

if cos_theta_entry > 1:
    cos_theta_entry = 1
theta_entry_truth = np.arccos(cos_theta_entry) # Theta entry
truth

cos_theta_exit = pint.py_ang(exit_vecs.T[0], alpha_theta.T)
if cos_theta_exit < -1:
    cos_theta_exit = -1

if cos_theta_exit > 1:
    cos_theta_exit = 1
theta_exit_truth = np.arccos(cos_theta_exit) # Theta exit truth

alpha_phi = np.array([1, 0, 0]) # X-axis
phi_entry_truth = np.arctan2(entry_vecs[1], entry_vecs[0])[0] #
Phi entry truth
phi_exit_truth = np.arctan2(exit_vecs[1], exit_vecs[0])[0] # Phi
exit truth
```

```
# Note: Theta goes from 0 to 180 deg, and phi goes from -180 to
180 deg

t_0 = t_entry + ((n_pad_strt - 1) / (hnds_un * s_ns)) # Truth
start time of track; This variable is in Seconds
d_t = t_exit - t_entry # Period of track; This variable is in
Seconds



# The conversion below of t_entry and n_pad_strt is what matches
the padding and the time to be in nanoseconds ->
(t_entry*s_ns)+(n_pad_strt/hnds_un)
sensor_response(sensors[0], out[0][sensors[0]]), track_len,
n_pad_strt, n_pad_end, len(out[track]['time']),
(t_entry*s_ns)+(n_pad_strt/hnds_un)
```

Analysis Alpha Generation

This cell takes the type of analysis requested and defines the analysis alphas. Then, the signal is integrated along these possible tracks to complete the integral transform. Typically, possible tracks would be evenly distributed in the parameter space. For the purposes of this code snippet, the possible tracks are manually generated.

```
time_track = out[track]['time']-out[track]['time'][0]

# Input the particular names of the analyses requested into the
analysis_parameter below:
    # Analysis Types: 'Time', 'Velocity', 'Spatial',
'Spatial_entry', 'Spatial_exit'

analysis_parameter = ['Velocity', 'Spatial_entry', 'Time']
```

```
## Note: vel, entry_vecs, exit_vecs, n_pad_strt, n_pad_end,
radius, and time_track are all truth values from the simulation

### 'Time' Only Variables: vel, entry_vecs, exit_vecs,
time_track

### 'Velocity' Only Variables: num_bins, entry_vecs, exit_vecs,
n_pad_strt, n_pad_end, time_track

### 'Time' & 'Velocity' Variables: num_bins, entry_vecs,
exit_vecs, time_track

### 'Spatial' Only Variables: vel, n_pad_strt, n_pad_end,
radius, N_thetas, N_phis_at_eq, epsilon
### 'Spatial_entry' Only Variables: vel, n_pad_strt, n_pad_end,
radius, N_thetas, N_phis_at_eq, epsilon, entry_Anl=True,
exit_vals=[theta_exit_truth, phi_exit_truth]
### 'Spatial_exit' Only Variables: vel, n_pad_strt, n_pad_end,
radius, N_thetas, N_phis_at_eq, epsilon, exit_Anl=True,
entry_vals=[theta_entry_truth, phi_entry_truth]

### 'Spatial' & 'Velocity' Variables: num_bins, n_pad_strt,
n_pad_end, radius, N_thetas, N_phis_at_eq, epsilon
### 'Spatial_entry' & 'Velocity' Variables: num_bins,
n_pad_strt, n_pad_end, radius, N_thetas, N_phis_at_eq, epsilon,
entry_Anl=True, exit_vals=[theta_exit_truth, phi_exit_truth]
### 'Spatial_exit' & 'Velocity' Variables:  num_bins,
n_pad_strt, n_pad_end, radius, N_thetas, N_phis_at_eq, epsilon,
exit_Anl=True, entry_vals=[theta_entry_truth, phi_entry_truth]

### 'Spatial' & 'Time' Variables: vel, radius, N_thetas,
```

```
N_phis_at_eq, epsilon
### 'Spatial_entry' & 'Time' Variables: vel, radius, N_thetas,
N_phis_at_eq, epsilon, exit_vals=[theta_exit_truth,
phi_exit_truth]
### 'Spatial_exit' & 'Time' Variables:  vel, radius, N_thetas,
N_phis_at_eq, epsilon, entry_vals=[theta_entry_truth,
phi_entry_truth]

### 'Spatial' & 'Velocity' & 'Time' Variables: num_bins, radius,
N_thetas, N_phis_at_eq, epsilon
### 'Spatial_entry' & 'Velocity' & 'Time' Variables: num_bins,
radius, N_thetas, N_phis_at_eq, epsilon,
exit_vals=[theta_exit_truth, phi_exit_truth]
### 'Spatial_exit' & 'Velocity' & 'Time' Variables:  num_bins,
radius, N_thetas, N_phis_at_eq, epsilon,
entry_vals=[theta_entry_truth, phi_entry_truth]

num_thetas = 15
num_phis_at_eq  = 2*num_thetas

if 'Velocity' in analysis_parameter:
    num_bin = 50
    velocity_bins = np.linspace(1e5, 8e5, num_bin)
    vel_array = velocity_bins[:-1] + np.diff(velocity_bins) / 2
else:
    vel_array = [vel]

if 'Time' in analysis_parameter:
    # Acceleration Settings: Analyzing the whole time parameter
space
    tm_steps = 5
    tmstep_strt = 0
```

```
    tmstep_end = len(time_track) - 1
else:
    # Acceleration Settings: Removing the padding and only
considering the true entry time
    tm_steps = len(time_track) - n_pad_strt - n_pad_end
    tmstep_strt = n_pad_strt - 1
    tmstep_end = len(time_track) - n_pad_end - 1

length_of_run = int((tmstep_end-tmstep_strt) / tm_steps)
print("Number of start times:", length_of_run)

if 'Spatial' in analysis_parameter:
    alphas, angles = pint.Spatial_Analysis_alphas(vel=vel_array,
radius=radius, N_thetas=num_thetas, N_phis_at_eq=num_phis_at_eq)
elif 'Spatial_entry' in analysis_parameter:
    alphas, angles = pint.Spatial_Analysis_alphas(vel=vel_array,
radius=radius, entry_Anl=True,

exit_vals=[theta_exit_truth, phi_exit_truth],
N_thetas=num_thetas, N_phis_at_eq=num_phis_at_eq)
elif 'Spatial_exit' in analysis_parameter:
    alphas, angles = pint.Spatial_Analysis_alphas(vel=vel_array,
radius=radius, exit_Anl=True,

entry_vals=[theta_entry_truth, phi_entry_truth],
N_thetas=num_thetas, N_phis_at_eq=num_phis_at_eq)
else:
    alphas = pint.Non_Spatial_Analysis_alphas(vel_array,
entry_vecs, exit_vecs)
```

Template Generation

This block runs the transform and gives back templates composed of each sensor's accelerations in the three dimensions for each timestep in the generated track.

```
def transform_temp(input_list):
    alpha_index, alpha_pair, sensors_pos, lin_resp,
adc_timestep_size = input_list
    response_length = len(lin_resp)

    signal_array = []

    dir_vector = np.array([
        alpha_pair[4] - alpha_pair[0],
        alpha_pair[5] - alpha_pair[1],
        alpha_pair[6] - alpha_pair[2],
    ])
    initial_pos = np.array([alpha_pair[0], alpha_pair[1],
alpha_pair[2]])

    dir_vector_step = dir_vector / (alpha_pair[7] -
alpha_pair[3]) * adc_timestep_size
    n_steps = int(np.ceil((alpha_pair[7] - alpha_pair[3]) /
adc_timestep_size))

    particle_pos_arr = np.array([initial_pos + j *
dir_vector_step for j in range(n_steps)])

    for sens_num, sensor_pos in enumerate(sensors_pos):
        vector_delta = np.zeros((n_steps, 4))
        for j in range(n_steps):
            vector_delta[j, 0] = (particle_pos_arr[j][0] -
sensor_pos[0])
```

```
            vector_delta[j, 1] = (particle_pos_arr[j][1] -
sensor_pos[1])
            vector_delta[j, 2] = (particle_pos_arr[j][2] -
sensor_pos[2])



signal_array.append(np.array(pint.signal_function(vector_delta,
lin_resp, adc_timestep_size)))

    return alpha_index, signal_array

input_list = []
for indx in range(len(alphas)):
    input_list.append((indx, alphas[indx, :], sensors_pos,
lin_resp, adc_timestep_size))

Library = {"Alpha_num": [], "Signal": []}

with Pool(50) as p:
    starttime = time.time()
    #for thing in p.imap(f, input_list):#, chunksize=1):
    for i, result in enumerate(p.imap(transform_temp,
input_list, chunksize=1)):
        Library['Alpha_num'].append(result[0])
        Library['Signal'].append(result[1])

        print('Time taken = {} seconds'.format(time.time() -
starttime))
```

Sensor Response and Analysis

This cell block runs the transform given the times and start time analysis. The transform gives back the signal strength and the two associated four-vectors for each of the templates considered in the analysis.

```
def transform_function(time_track, tm_steps, tmstep_strt,
tmstep_end, accels, alphas, sensors_pos, Library):
    start_time_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
    start_times = time_track[start_time_indices]

    transformed_data = pint.transform_calc(accels,
np.array(alphas), sensors_pos, time_track/1e9, start_times/1e9,
start_time_indices, Library)
    return transformed_data

if tm_steps == len(time_track) and (tmstep_strt != n_pad_strt -
1 or tmstep_end != len(time_track) - n_pad_end):
    raise ValueError("You are only analyzing 1 time step, but
you also are considering the padding. Please input correct
values")
else:
    transformed_data = transform_function(time_track, tm_steps,
tmstep_strt, tmstep_end, accels, alphas, sensors_pos, Library)
```

Plotting

The following cells define some variables that aid in plotting, which is followed by the call to plot.

```
# Format: variable = np.array(transformed_data['variable_name'])
```

```
SNR_data_plt = np.array(transformed_data['SNR'])

# Calculate variable from the standard varaibles of transform if
necessary (e.g. theta from the cartesian coordinates)

tm_strt_plt = np.array(transformed_data['alpha0_t'])
tm_end_plt = np.array(transformed_data['alpha1_t'])
vel_plt = np.array(np.sqrt((transformed_data['alpha1_x'] -
transformed_data['alpha0_x'])**2 +
                   (transformed_data['alpha1_y'] -
transformed_data['alpha0_y'])**2 +
                   (transformed_data['alpha1_z'] -
transformed_data['alpha0_z'])**2) /
(transformed_data['alpha1_t'] - transformed_data['alpha0_t']))

theta_entry_plt = np.arccos(transformed_data['alpha0_z'] /
np.sqrt(transformed_data['alpha0_x'] ** 2 +
transformed_data['alpha0_y'] ** 2 + transformed_data['alpha0_z']
** 2))
theta_exit_plt = np.arccos(transformed_data['alpha1_z'] /
np.sqrt(transformed_data['alpha1_x'] ** 2 +
transformed_data['alpha1_y'] ** 2 + transformed_data['alpha1_z']
** 2))
phi_entry_plt = np.arctan2(transformed_data['alpha0_y'],
transformed_data['alpha0_x'])
phi_exit_plt = np.arctan2(transformed_data['alpha1_y'],
transformed_data['alpha1_x'])

fig = plt.figure(figsize=(13, 13))

ax = fig.add_subplot(7,7,1)
plt.hist(tm_strt_plt*(10**6), bins=50)
```

```
plt.ylabel('Time (us)')

ax = fig.add_subplot(7,7,8)
timestep_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
timesteps = time_track[timestep_indices]
theta_exit_bin_edges = np.linspace(0, 6.28, 30)
if len(timesteps) > 1:
    timestep_edges = list(timesteps-(timesteps[1]-
timesteps[0])/2)
    timestep_edges.append(timesteps[-1] + (timesteps[1]-
timesteps[0])/2)
    timestep_edges = np.array(timestep_edges)*1e-3
else:
    timestep_edges = np.array([timesteps[0] * 1e-3])
X, Y = np.meshgrid(timestep_edges, theta_exit_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(timestep_edges, row['alpha1_t']*1e6) - 1
    i = np.searchsorted(theta_exit_bin_edges,
theta_exit_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter((len(time_track) - n_pad_end)/10, theta_exit_truth,
c='red', marker='x', s=10)
ax.set_xlim([n_pad_strt*0.1 - 10, n_pad_strt*0.1 + 10])
plt.ylabel('Theta Exit')

ax = fig.add_subplot(7,7,9)
plt.hist(theta_exit_plt, bins=50)
```

```
ax = fig.add_subplot(7,7,15)
timestep_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
timesteps = time_track[timestep_indices]
theta_entry_bin_edges = np.linspace(0, 6.28, 30)
if len(timesteps) > 1:
    timestep_edges = list(timesteps-(timesteps[1]-
timesteps[0])/2)
    timestep_edges.append(timesteps[-1] + (timesteps[1]-
timesteps[0])/2)
    timestep_edges = np.array(timestep_edges)*1e-3
else:
    timestep_edges = np.array([timesteps[0] * 1e-3])
X, Y = np.meshgrid(timestep_edges, theta_entry_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(timestep_edges, row['alpha1_t']*1e6) - 1
    i = np.searchsorted(theta_entry_bin_edges,
theta_entry_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter((len(time_track) - n_pad_end)/10, theta_entry_truth,
c='red', marker='x', s=10)
ax.set_xlim([n_pad_strt*0.1 - 10, n_pad_strt*0.1 + 10])
plt.ylabel('Theta Entry')

ax = fig.add_subplot(7,7,16)
theta_exit_bin_edges = np.linspace(0, 6.28, 30)
theta_entry_bin_edges = np.linspace(0, 6.28, 30)
X, Y = np.meshgrid(theta_exit_bin_edges, theta_entry_bin_edges)
C = np.zeros(X.shape)
```

```
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_exit_bin_edges,
theta_exit_plt[i_row]) - 1
    i = np.searchsorted(theta_entry_bin_edges,
theta_entry_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_exit_truth, theta_entry_truth, c='red',
marker='x', s=10)

ax = fig.add_subplot(7,7,17)
plt.hist(theta_entry_plt, bins=50)

ax = fig.add_subplot(7,7,22)
timestep_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
timesteps = time_track[timestep_indices]
phi_exit_bin_edges = np.linspace(-3.14, 3.14, 30)
if len(timesteps) > 1:
    timestep_edges = list(timesteps-(timesteps[1]-
timesteps[0])/2)
    timestep_edges.append(timesteps[-1] + (timesteps[1]-
timesteps[0])/2)
    timestep_edges = np.array(timestep_edges)*1e-3
else:
    timestep_edges = np.array([timesteps[0] * 1e-3])
X, Y = np.meshgrid(timestep_edges, phi_exit_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(timestep_edges, row['alpha1_t']*1e6) - 1
```

```
    i = np.searchsorted(phi_exit_bin_edges, phi_exit_plt[i_row])
- 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter((len(time_track) - n_pad_end)/10, phi_exit_truth,
c='red', marker='x', s=10)
ax.set_xlim([n_pad_strt*0.1 - 10, n_pad_strt*0.1 + 10])
plt.ylabel('Phi Exit')

ax = fig.add_subplot(7,7,23)
theta_exit_bin_edges = np.linspace(0, 6.28, 30)
phi_exit_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(theta_exit_bin_edges, phi_exit_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_exit_bin_edges,
theta_exit_plt[i_row]) - 1
    i = np.searchsorted(phi_exit_bin_edges, phi_exit_plt[i_row])
- 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_exit_truth, phi_exit_truth, c='red',
marker='x', s=10)

ax = fig.add_subplot(7,7,24)
theta_entry_bin_edges = np.linspace(0, 6.28, 30)
phi_exit_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(theta_entry_bin_edges, phi_exit_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
```

```
    j = np.searchsorted(theta_entry_bin_edges,
theta_entry_plt[i_row]) - 1
    i = np.searchsorted(phi_exit_bin_edges, phi_exit_plt[i_row])
- 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_entry_truth, phi_exit_truth, c='red',
marker='x', s=10)

ax = fig.add_subplot(7,7,25)
plt.hist(phi_exit_plt, bins=50)

ax = fig.add_subplot(7,7,29)
timestep_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
timesteps = time_track[timestep_indices]
phi_entry_bin_edges = np.linspace(-3.14, 3.14, 30)
if len(timesteps) > 1:
    timestep_edges = list(timesteps-(timesteps[1]-
timesteps[0])/2)
    timestep_edges.append(timesteps[-1] + (timesteps[1]-
timesteps[0])/2)
    timestep_edges = np.array(timestep_edges)*1e-3
else:
    timestep_edges = np.array([timesteps[0] * 1e-3])
X, Y = np.meshgrid(timestep_edges, phi_entry_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(timestep_edges, row['alpha1_t']*1e6) - 1
    i = np.searchsorted(phi_entry_bin_edges,
phi_entry_plt[i_row]) - 1
```

```
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter((len(time_track) - n_pad_end)/10, phi_entry_truth,
c='red', marker='x', s=10)
ax.set_xlim([n_pad_strt*0.1 - 10, n_pad_strt*0.1 + 10])
plt.ylabel('Phi Entry')

ax = fig.add_subplot(7,7,30)
theta_exit_bin_edges = np.linspace(0, 6.28, 30)
phi_entry_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(theta_exit_bin_edges, phi_entry_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_exit_bin_edges,
theta_exit_plt[i_row]) - 1
    i = np.searchsorted(phi_entry_bin_edges,
phi_entry_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_exit_truth, phi_entry_truth, c='red',
marker='x', s=10)

ax = fig.add_subplot(7,7,31)
theta_entry_bin_edges = np.linspace(0, 6.28, 30)
phi_entry_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(theta_entry_bin_edges, phi_entry_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_entry_bin_edges,
theta_entry_plt[i_row]) - 1
```

```
    i = np.searchsorted(phi_entry_bin_edges,
phi_entry_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_entry_truth, phi_entry_truth, c='red',
marker='x', s=10)

ax = fig.add_subplot(7,7,32)
phi_exit_bin_edges = np.linspace(-3.14, 3.14, 30)
phi_entry_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(phi_exit_bin_edges, phi_entry_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(phi_exit_bin_edges, phi_exit_plt[i_row])
- 1
    i = np.searchsorted(phi_entry_bin_edges,
phi_entry_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(phi_exit_truth, phi_entry_truth, c='red',
marker='x', s=10)

ax = fig.add_subplot(7,7,33)
plt.hist(phi_entry_plt, bins=50)

ax = fig.add_subplot(7,7,36)
timestep_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
timesteps = time_track[timestep_indices]
if 'Velocity' in analysis_parameter:
    v_bin_edges = np.linspace(1e5, 7e5, 30)
```

```
else:
    v_bin_edges = vel
if len(timesteps) > 1:
    timestep_edges = list(timesteps-(timesteps[1]-
timesteps[0])/2)
    timestep_edges.append(timesteps[-1] + (timesteps[1]-
timesteps[0])/2)
    timestep_edges = np.array(timestep_edges)*1e-3
else:
    timestep_edges = np.array([timesteps[0] * 1e-3])
X, Y = np.meshgrid(timestep_edges, v_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(timestep_edges, row['alpha0_t']*1e6) - 1
    i = np.searchsorted(v_bin_edges, vel_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y/1e3,C)
#ax.scatter(n_pad_strt*0.1, vel[0]/1e3, c='red', marker='x',
s=10)
ax.set_xlim([n_pad_strt*0.1 - 10, n_pad_strt*0.1 + 10])
plt.ylabel('Velocity (km/s)')

ax = fig.add_subplot(7,7,37)
if 'Velocity' in analysis_parameter:
    v_bin_edges = np.linspace(1e5, 7e5, 30)
else:
    v_bin_edges = vel
theta_exit_bin_edges = np.linspace(0, 6.28, 30)
X, Y = np.meshgrid(theta_exit_bin_edges, v_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
```

```
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_exit_bin_edges,
theta_exit_plt[i_row]) - 1
    i = np.searchsorted(v_bin_edges, vel_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y/1e3,C)
#ax.scatter(theta_exit_truth, vel[0]/1e3, c='red', marker='x',
s=10)

ax = fig.add_subplot(7,7,38)
if 'Velocity' in analysis_parameter:
    v_bin_edges = np.linspace(1e5, 7e5, 30)
else:
    v_bin_edges = vel
theta_entry_bin_edges = np.linspace(0, 6.28, 30)
X, Y = np.meshgrid(theta_entry_bin_edges, v_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_entry_bin_edges,
theta_entry_plt[i_row]) - 1
    i = np.searchsorted(v_bin_edges, vel_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y/1e3,C)
#ax.scatter(theta_entry_truth, vel[0]/1e3, c='red', marker='x',
s=10)

ax = fig.add_subplot(7,7,39)
if 'Velocity' in analysis_parameter:
    v_bin_edges = np.linspace(1e5, 7e5, 30)
else:
    v_bin_edges = vel
```

```
phi_exit_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(phi_exit_bin_edges, v_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(phi_exit_bin_edges, phi_exit_plt[i_row])
- 1
    i = np.searchsorted(v_bin_edges, vel_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y/1e3,C)
#ax.scatter(phi_exit_truth, vel[0]/1e3, c='red', marker='x',
s=10)

ax = fig.add_subplot(7,7,40)
if 'Velocity' in analysis_parameter:
    v_bin_edges = np.linspace(1e5, 7e5, 30)
else:
    v_bin_edges = vel
phi_entry_bin_edges = np.linspace(-3.14, 3.14, 30)
X, Y = np.meshgrid(phi_entry_bin_edges, v_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(phi_entry_bin_edges,
phi_entry_plt[i_row]) - 1
    i = np.searchsorted(v_bin_edges, vel_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y/1e3,C)
#ax.scatter(phi_entry_truth, vel[0]/1e3, c='red', marker='x',
s=10)

ax = fig.add_subplot(7,7,41)
```

```
plt.hist(vel/1000, bins=50)

ax = fig.add_subplot(7,7,43)
timestep_indices = np.array(range(tmstep_strt, tmstep_end,
tm_steps))
timesteps = time_track[timestep_indices]
amplitude_bin_edges = np.linspace(0, 5, 30)
if len(timesteps) > 1:
    timestep_edges = list(timesteps-(timesteps[1]-
timesteps[0])/2)
    timestep_edges.append(timesteps[-1] + (timesteps[1]-
timesteps[0])/2)
    timestep_edges = np.array(timestep_edges)*1e-3
else:
    timestep_edges = np.array([timesteps[0] * 1e-3])
X, Y = np.meshgrid(timestep_edges, amplitude_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(timestep_edges, row['alpha1_t']*1e6) - 1
    i = np.searchsorted(amplitude_bin_edges,
SNR_data_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter((len(time_track) - n_pad_end)/10, theta_exit_truth,
c='red', marker='x', s=10)
ax.set_xlim([n_pad_strt*0.1, n_pad_strt*0.1 + 20])
plt.ylabel('Amplitude')
plt.xlabel('Time (us)')

ax = fig.add_subplot(7,7,44)
theta_exit_bin_edges = np.linspace(0, 6.28, 30)
```

```
amplitude_bin_edges = np.linspace(0, 5, 30)
X, Y = np.meshgrid(theta_exit_bin_edges, amplitude_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_exit_bin_edges,
theta_exit_plt[i_row]) - 1
    i = np.searchsorted(amplitude_bin_edges,
SNR_data_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_exit_truth, vel[0]/1e3, c='red', marker='x',
s=10)
plt.xlabel('Theta Exit')

ax = fig.add_subplot(7,7,45)
theta_entry_bin_edges = np.linspace(0, 6.28, 30)
amplitude_bin_edges = np.linspace(0, 5, 30)
X, Y = np.meshgrid(theta_entry_bin_edges, amplitude_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(theta_entry_bin_edges,
theta_entry_plt[i_row]) - 1
    i = np.searchsorted(amplitude_bin_edges,
SNR_data_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(theta_entry_truth, vel[0]/1e3, c='red', marker='x',
s=10)
plt.xlabel('Theta Entry')
```

```
ax = fig.add_subplot(7,7,46)
phi_exit_bin_edges = np.linspace(-3.14, 3.14, 30)
amplitude_bin_edges = np.linspace(0, 5, 30)
X, Y = np.meshgrid(phi_exit_bin_edges, amplitude_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(phi_exit_bin_edges, phi_exit_plt[i_row])
- 1
    i = np.searchsorted(amplitude_bin_edges,
SNR_data_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(phi_exit_truth, vel[0]/1e3, c='red', marker='x',
s=10)
plt.xlabel('Phi Exit')

ax = fig.add_subplot(7,7,47)
phi_entry_bin_edges = np.linspace(-3.14, 3.14, 30)
amplitude_bin_edges = np.linspace(0, 5, 30)
X, Y = np.meshgrid(phi_entry_bin_edges, amplitude_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(phi_entry_bin_edges,
phi_entry_plt[i_row]) - 1
    i = np.searchsorted(amplitude_bin_edges,
SNR_data_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X,Y,C)
#ax.scatter(phi_entry_truth, vel[0]/1e3, c='red', marker='x',
s=10)
```

```
plt.xlabel('Phi Entry')

ax = fig.add_subplot(7,7,48)
if 'Velocity' in analysis_parameter:
    v_bin_edges = np.linspace(1e5, 7e5, 30)
else:
    v_bin_edges = vel
amplitude_bin_edges = np.linspace(0, 5, 30)
X, Y = np.meshgrid(v_bin_edges, amplitude_bin_edges)
C = np.zeros(X.shape)
analysis_min_steps = 10
for i_row,row in enumerate(transformed_data):
    j = np.searchsorted(v_bin_edges, vel_plt[i_row]) - 1
    i = np.searchsorted(amplitude_bin_edges,
SNR_data_plt[i_row]) - 1
    C[i,j] = row['SNR']
pcm = ax.pcolormesh(X/1e3,Y,C)
#ax.scatter(vel[0]/1e3, vel[0]/1e3, c='red', marker='x', s=10)
plt.xlabel('Velocity (km/s)')

ax = fig.add_subplot(7,7,49)
plt.hist(transformed_data['SNR'], bins=50)
plt.xlabel('Amplitude')

plt.show()
```

REFERENCES

[1] Bertone, G., & Hooper, D. (2016, May 24). A History of Dark Matter. *Review of Modern Physics, Vol. 90, Issue 4*, pp. 1-88. https://doi.org/10.1103/RevModPhys.90.045002

[2] Kapteyn, J., C. (1922, May 18). First Attempt at a Theory of the Arrangement and Motion of the Siderial System. *Astrophysical Journal, Vol. 55, Issue 80*, pp. 302-328. https://doi.org/10.1086/142670

[3] Oort, J., H. (1932, August 17). The Force Exerted by the Stellar System in the Direction Perpendicular to the Galactic Plane and Some Related Problems. *Astronomical Institutes of The Netherlands, Vol. 6, Issue 238*, pp. 249-287. https://doi.org/10.1073/pnas.10.6.253

[4] Freeman, K., C. (1970, June 8). On the Disks of Spiral and S0 Galaxies. *The Astrophysical Journal, Vol. 160, Issue 53*, pp. 811-830. https://doi.org/10.1086/150474

[5] Rubin, V., C., Ford. W., K., & Thonnard., N. (1980, June 1). Rotational Properties of 21 Sc Galaxies with a Large Range of Luminosities and Radii, from NGC 4605 (R = 4 kpc) to UGC 2885 (R = 122 kpc). *The Astrophysical Journal, Vol. 238, Issue 25*, pp. 471-487. https://doi.org/10.1086/158003

[6] Clausius, R., J., E. (1870, May 13). On a Mechanical Theorem Applicable to Heat. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, Vol. 40, Issue 265*, pp. 122-127. https://doi.org/10.1080/14786447008640370

[7] Zwicky, F. (1933, June 29). The Redshift of Extragalactic Nebulae. *Helvetica Physica Acta, Vol. 6, Issue 15*, pp. 110-127. https://doi.org/10.1007/s10714-008-0706-5

[8] Milgrom, M. (1983, July 15). A Modification of the Newtonian Dynamics as a Possible Alternative to the Hidden Mass Hypothesis. *The Astrophysical Journal, Vol. 270, Issue 71*, pp. 365-370. https://doi.org/10.1086/161130

[9] Robertson, A., Massey, R., & Eke, V. (2016, May 13). What Does the Bullet Cluster Tell Us about Self-Interacting Dark Matter? *Monthly Notices of the Royal Astronomical Society, Vol. 465, Issue 267*, pp. 569-587. https://doi.org/10.1093/mnras/stw2670

[10] Ostriker, J., P., & Steinhardt, P. (2003, June 20). New Light on Dark Matter. *Science, Vol. 300, Issue 5627*, pp. 1909-1913. https://doi.org/10.1126/science.1085976

[11] Kane, G., & Watson, S. (2008, July 18). Dark Matter and LHC: What is the Connection? *Modern Physics Letters A, Vol. 23, Issue 26*, pp. 2103-2123. https://doi.org/10.1142/S0217732308028314

[12] Fox, P., J., Harnik, R., Kopp, J., & Tsai, Y. (2011, July 22). LEP Shines Light on Dark Matter. *Physics Review D, Vol. 84, Issue 11*, pp. 11-29. https://doi.org/10.1103/PhysRevD.84.014028

[13] Ellis, J., Flores, R., A., Freese, K., Ritz, S., Seckel, D., & Silk, J. (1988, November 24). Cosmic Ray Constraints on the Annihilations of Relic Particles in the Galactic Halo. *Physics Letters B, Vol. 214, Issue 3*, pp. 403-412. https://doi.org/10.1016/0370-2693(88)91385-8

[14] Bertone, G., Hooper, D., & Silk, J. (2005, January 17). Particle Dark Matter: Evidence, Candidates, and Constraints. *Physics Reports, Vol. 405, Issue 5*, pp. 279-390. https://doi.org/10.1016/j.physrep.2004.08.031

[15] Esmaili, A., Kang, S., K., & Serpico, P., D. (2014, December 23). IceCube Events and Decaying Dark Matter: Hints and Constraints. *Journal of Cosmology and Astroparticle Physics, Vol. 2014, Issue 54*, pp. 12-38. https://doi.org/10.1088/1475-7516/2014/12/054

[16] Drukier, A., K., Freese, K., & Spergel, D., N. (1986, June 15). Detecting Cold Dark Matter Candidates. *Physics Review D, Vol. 33, Issue 12*, pp. 3495-3508. https://doi.org/10.1103/PhysRevD.33.3495

[17] Lee, S., K., Lisanti, M., Peter, A., H., G., & Safdi, B., R. (2014, January 3). Effect of Gravitational Focusing on Annual Modulation in Dark-Matter Direct-Detection Experiments. *Physics Review Letters, Vol. 112, Issue 10*, pp. 11-16. https://doi.org/10.1103/PhysRevLett.112.011301

[18] Carney, D., Ghosh, S., Krnjaic, G., & Taylor, J., M. (2019, March 5). Gravitational Direct Detection of Dark Matter. *Physics Review D, Vol. 102, Issue 7*, pp. 7-12. https://doi.org/10.1103/PhysRevD.102.072003

[19] Baudis, L. (2017, December 26). The Search for Dark Matter. *European Review, Vol. 26, Issue 1*, pp. 70-81. https://doi.org/10.1017/S1062798717000783

[20] Hall, E., D., Adhikari, R., X., Frolov, V., V., Muller, H & Pospelov, M. (2018, October 23). Laser Interferometers as Dark Matter Detectors. *Physics Review D, Vol. 98, Issue 8*, pp. 3-21. https://doi.org/10.1103/PhysRevD.98.083019

[21] Tanabashi, M., Particle Data Group. (2018, August 17). Review of Particle Physics. *Physics Review D, Vol. 98, Issue 3*, pp. 3-1901. https://doi.org/10.1103/PhysRevD.98.030001

[22] Read, J., I. (2014, June 13). The Local Dark Matter Density. *Journal of Physics G: Nuclear and Particle Physics, Vol. 41, Issue 6*, pp. 6-67. https://doi.org/10.1088/0954-3899/41/6/063101

[23] Aasi, J., The LIGO Scientific Collaboration. (2015, March 3). Advanced LIGO. *Classical and Quantum Gravity, Vol. 32, Issue 7*, pp. 7-48. https://doi.org/10.1088/0264-9381/32/7/074001