Applied Science Research
Dann
Christopher Rebert
5/10/08

Final Project Paper: Muon lifetimes

I. Abstract

In this project, the number of atmospheric muons was measured at day and night using provided scintillators and a QuarkNet DAQ board. This entailed raw count calibration, counter plateauing, and the creation of a Python program to more easily interact with the QuarkNet board. Experimental results support the idea of solar muon emission, with coincidence rates increasing until a noon peak of 62.64 Hz (when the Sun is approximately overhead and closest to the Earth's surface) and then decreasing all the way down to 32.46 Hz at 2AM the following morning with the Sun shining on the opposite side of the Earth.

II. Motivation and History

Originally, I had intended to do an experiment proving relativity by showing a discrepancy in the lifetimes of cosmic ray muons, which sounded particularly intriguing to me.[1] It would also involve writing a program to communicate with the DAQ unit, and being a programmer, this piqued my interest. However, once Dr. Dann had located the exact paper outlining the experiment, we both concluded that it would not be feasible for a high school student considering the limited time remaining in the course. However, having already made significant progress on my computer program and finished initial calibration, I decided to proceed with an alternate muon-related experiment. After consulting further with Dr. Dann, it was decided that I would do a day-night comparison experiment, wherein I would count the number of muons per hour during day and night and see if there was an appreciable difference between the night and day numbers.

There have been a few fairly recent studies about atmospheric muon detection. In [2], the experimenters used an underwater neutrino telescope and evaluated its performance as well as that of their DAQ system. They also measured muon flux as a function of zenith angle and found their data to be in line with previous measurements and predictions. [3] was a similar experiment, measuring muon flux at the South Pole at 5 different zenith angles using a unique scintillator compound. [4] examines muon flux at several altitudes and uses the data to calculate the error in neutrino flux generated by a certain neutrino interaction model, which they then go on to refine themselves. [5] uses cosmic ray energy spectrum data from a balloon experiment to calculate cosmic ray muon flux at ground level, and finds the results to be in keeping with those of similar experiments.

III. Theory of operation

To begin with, according to the Standard Model of particle physics, the muon (symbol: μ) is an elementary particle with electric charge -1e and spin ½. It was discovered by Carl D. Anderson, winner of the 1936 Nobel Prize in Physics, in 1936 at Caltech.[6]

When cosmic rays, high-energy particles from space, hit atoms in the atmosphere, they react producing, among other things, pions, non-elementary particles that quickly decay into muons and neutrinos via the following reaction:

$\pi^- \rightarrow \mu^- + \nu_\mu$

These muons then continue traveling along almost exactly the same path as their progenitor pions. It is also theorized that the Sun could be producing muons as a result of its nuclear reactions.[7]

When a muon hits a scintillator, the scintillator then absorbs the radiation and in response fluoresces.[8] The light produced is then directed into a Photomultiplier Tube (PMT). In the PMT, after being focused by a lens, the photons are converted into electrons by a photocathode. These electrons go through a MicroChannel Plate (MCP), a plastic disc with metal electrodes on each side and millions of microscopic holes in it. High-voltage bursts that are constantly sent through the electrode pair accelerate the electrons through the microchannels in the plate, causing cascaded secondary emission, drastically increasing the number of electrons.[9]

These electrical impulses are then conveyed via Lemo signal cables to ports on the QuarkNet DAQ board for processing . After being amplified by 10x, the signal is sent through discriminators. The discriminators check whether the signal meets a specified threshold level set by potentiometer. If it does, then this constitutes an incident and the incident counter for that scintillator is incremented. And if at least a specified number of incidents occur within a specified time interval, this constitutes a coincidence and the coincidence counter is incremented. There are also 2 parameters, d and w, The incidence and coincidence counts can then be read by computer over a serial cable connection.[10]

The interactive console-based Python[11] script communicates using the simple ASCII-based commands defined in the user manual[10] while presenting a friendlier menu-driven interface to users and not requiring them to know anything about the DAQ board's command language. Note that the script should work on both Windows and unix (including Mac) systems, but only Windows was used in this project due to the difficulty of locating a Mac with a serial port or a serial port adapter. For further information on how the script operates, see the heavily commented source in Appendix A.

IV. Design

Due to uncertainty as to their effect, d and w were left at their default values (d = 6 = 144ns, w = 10 = 240ns)[10] throughout the experiment. -0.3V was used as the threshold voltage value throughout this experiment on the advice of Dr. Dann. All experiments took place in the first-floor ASR classroom at Menlo School, approximately at ground level.

Before performing the day-night experiment, it is necessary to “plateau” the scintillators. This is done in 2 steps: raw-count calibration and multiple counter plateauing. Plateauing is needed to compensate for differing PMT gain and aging/degredation of the scintillator material in order to get accurate data. Plateauing adjusts the PMT voltage so that the data acquired contains a minimum of electronic noise and a maximum of real muon interactions. If the voltage is set too high, then electronic noise will be heavily amplified and give false positives. If the voltage is set too low, then the amount of data acquired will be significantly decreased only high-energy muons will be detected while low or medium energy muons will be ignored, resulting in the loss of legitimate data. In the multi-counter plateauing step, the “flat” part of the graph is chosen as this indicates that one is detecting

most of the muons present with only minimum electronic noise and increasing the voltage slightly is finding no additional muons. The edges of the graph go up and down exponentially due to the cascading effect used by the PMTs.

For the raw count calibration, PMT voltage was varied and incidence counts were taken at each voltage setting for ~45s. From this the incidence rate was calculated. The scintillators were laid flat on a countertop with nothing on top of them so as to avoid any extra muon shielding and expose maximum surface area to skyward muon detection. As a muon incidence rate of 40Hz at ground level had been established as the accepted value[12], the PMT voltage was set to the voltage setting which gave a reading closest to 40Hz until after counter plateauing was done.

For the plateauing experiment and the day-night experiment, the scintillators were placed one on top of another in a "sandwich" configuration, with some wood blocks used to space and stabilize them. Scintillator "D" was not used due to time constraints and only 3 scintillators being required for plateauing.

For ease of explanation in the plateauing experiment, let:

- 1 be the top scintillator
- 2 be the middle scintillator
- 3 be the bottom scintillator

In the plateauing experiment, 2-fold coincidences between 1 & 2 were counted for ~120s and then 2-fold coincidences between 1 & 3 were counter for ~120s. The ratio of the frequency of the former over the frequency of the latter term is then calculated. This process is then repeated for several PMT voltage values, and the frequency ratio vs. PMT voltage is then plotted. The voltage where the graph is flattest is then the optimal value. The scintillators are then repositioned appropriately and the entire process conducted again to plateau the next scintillator until all of them have been plateaued.

In the day-night experiment, 2-fold coincidences between the scintillators were counted and polled periodically throughout consecutive ~1-hour periods with the starting and ending times of each period noted. Simple division yields average muon rates for each 1-hour period, which can then be plotted.

V. Results

The results of the initial scintillator calibration may be found in Tables 2-5 in Appendix B and Figures 1-4 which follow.

Figure 1: Graph of scintillator "A" raw count data. Optimal voltage setting: 0.917V, which gave reading of 40.18Hz

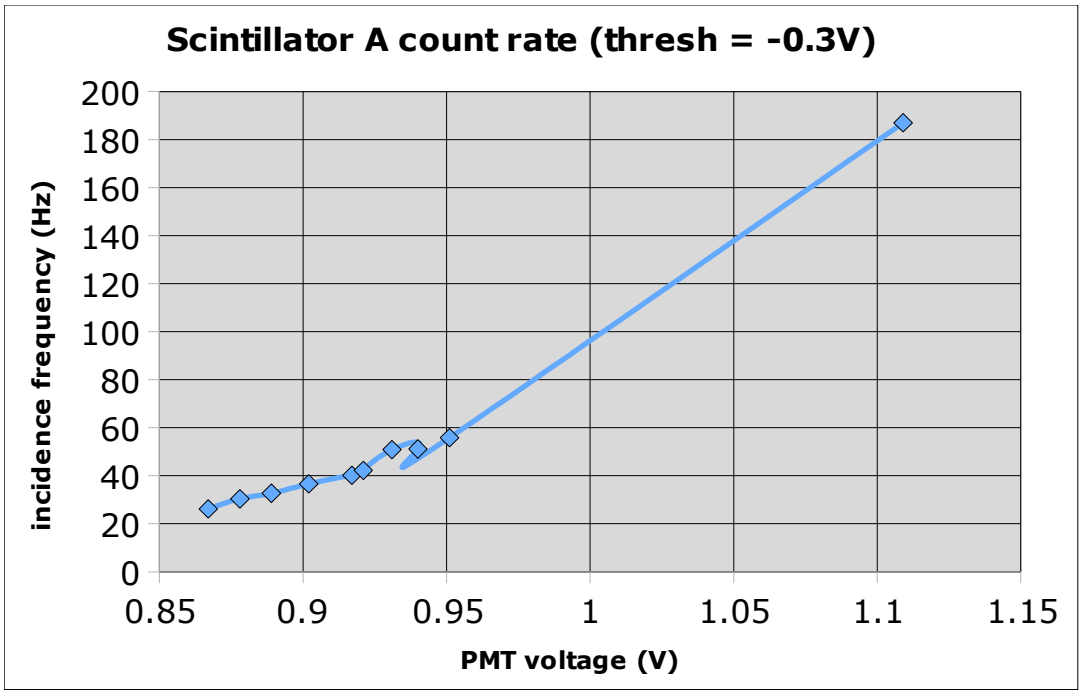


Figure 2: Graph of scintillator "B" raw count data. Optimal voltage setting: 0.983V, which gave reading of 39.83Hz

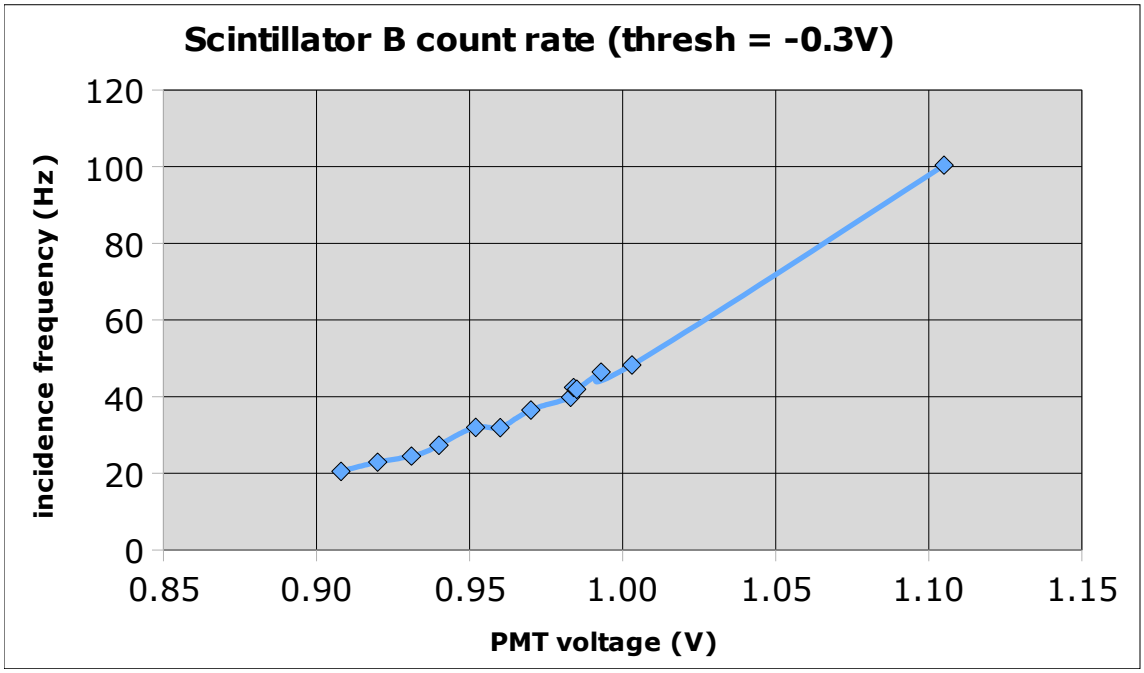


Figure 3: Graph of scintillator "C" raw count data. Optimal voltage setting: 1.019V, which gave a reading of 39.62Hz

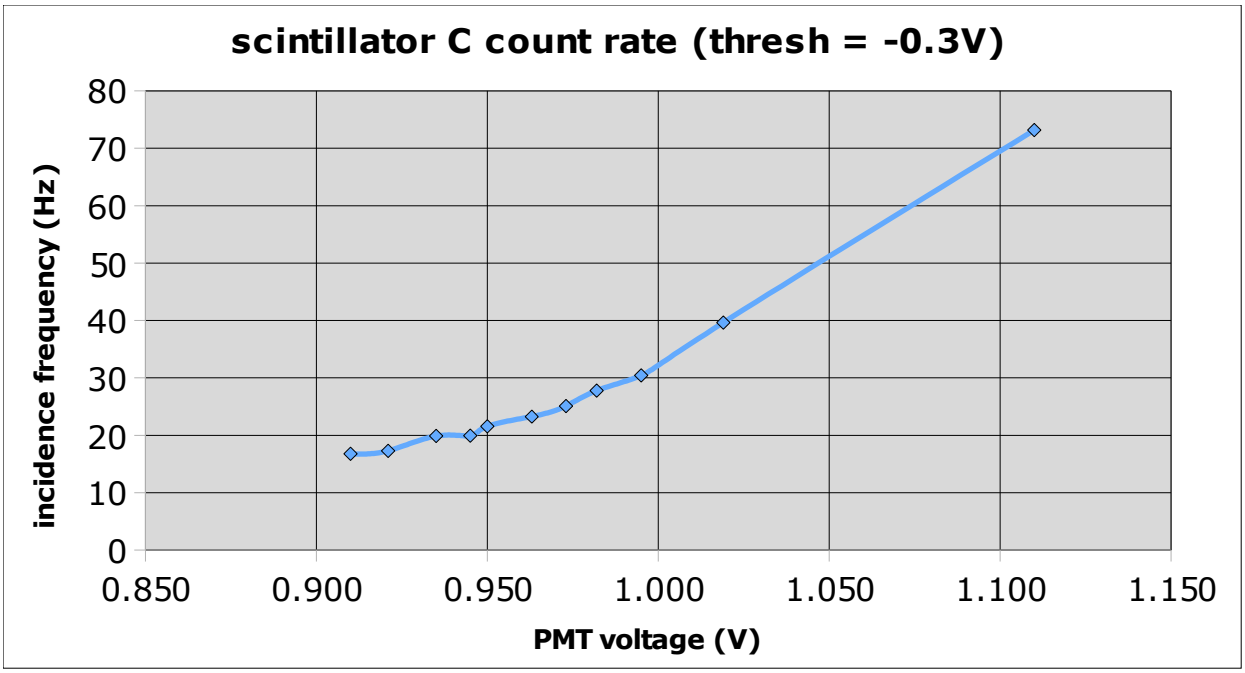
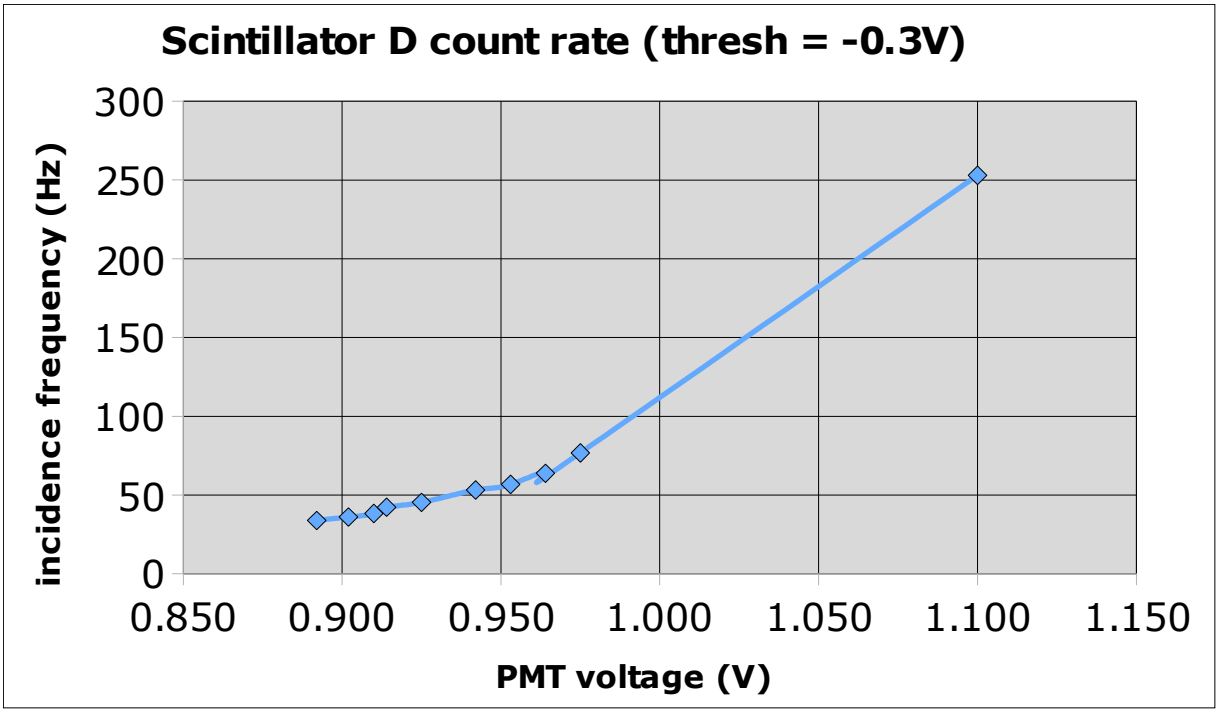


Figure 4: Graph of scintillator "D" raw count data. Optimal voltage: 0.910V, which gave a reading of 38.31Hz



Data Table 1: Plateauing results for scintillator "A"

Voltage (V)	Frequency ratio
0.729	0.00314
0.835	0.0192
0.947	0.860
0.951	0.928
0.957	1.02
0.970	1.13
1.106	1.42
1.120	987
1.140	7260

Data Table 2: Plateauing results for scintillator “B”

Voltage (V)	Frequency ratio
0.737	0.00528
0.886	0.0344
0.976	0.941
0.982	0.973
1.003	0.986
1.062	1.108
1.119	1.242
1.360	769
1.429	8416

Data Table 3: Plateauing results for scintillator “C”

Voltage (V)	Frequency ratio
0.754	0.00259
0.861	0.0385
0.972	0.969
0.989	0.981
1.089	1.09
1.154	1.26
1.189	1.48
1.359	747
1.412	4392

Unfortunately, I was unable to figure out how to create an Excel graph with a logarithmic scale as is required for graphs of the plateauing data to be intelligible, but exponential trends are clearly visible in the data, and based on [10], ratios closest to 1.0 indicate optimal voltage. These voltages are summarized in Table 4.

Data Table 4: Optimum voltages for scintillators based on plateauing

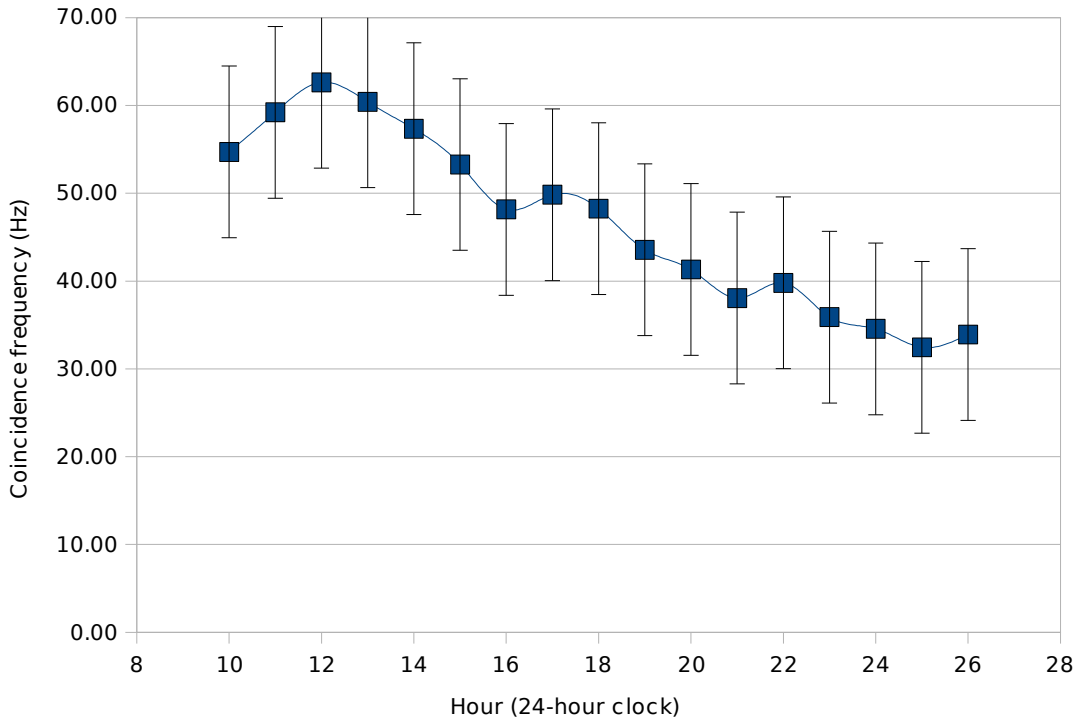
Scintillator	Optimum voltage (V)
A	0.957
B	1.003
C	1.089

The following experimental results (Table 5 & Figure 5) from the day-night muon detection experiment support the idea of solar muon emission because the muon coincidence rates increased to a noon peak of 62.64 Hz (when the Sun is approximately overhead and closest to the Earth's surface) and then decreased all the way down to 32.46 Hz at 2AM the following morning, with the Sun shining all the way on the opposite side of the Earth. This correlation of Sun exposure and muon coincidences suggest that the Sun is outputting muons, which are then detected when the Sun is visible thus increasing the observed muon counts. However, as discussed in the Conclusion's error section, these data do not prove definitively that the Sun is emitting muons.

Data Table 5: Data from day-night muon flux experiment

Time period	Coincidence frequency (Hz)	Time elapsed (s)	Coincidences
10:02:12AM – 11:00:07AM	54.71	3475	190128
11:00:15AM – 12:00:24PM	59.22	3609	213731
12:00:33PM – 01:00:38PM	62.64	3605	225815
01:00:47PM – 02:00:03PM	60.42	3556	214867
02:00:18PM – 03:00:23PM	57.36	3605	206781
03:00:40PM – 04:00:01PM	53.28	3561	189733
04:00:52PM – 05:00:29PM	48.16	3577	172257
05:00:37PM – 06:00:08PM	49.83	3571	177949
06:00:42PM – 07:00:31PM	48.25	3589	173156
07:00:56PM – 08:00:11PM	43.56	3555	154839
08:00:38PM – 09:00:21PM	41.32	3583	148041
09:00:32PM – 10:00:03PM	38.07	3571	135934
10:00:06PM – 11:00:55PM	39.79	3649	145194
11:01:09PM – 12:00:48AM	35.89	3579	128468
12:00:53AM – 01:00:40AM	34.56	3587	123982
01:01:13AM – 02:00:10AM	32.46	3537	114813
02:00:34AM – 03:00:36AM	33.91	3602	122142

Figure 5: Graph of day-night experiment data with error bars. Error calculated using standard deviation. Hour values greater than 24 indicate times in the following day.



VI. Conclusion

Overall, the project was a success though its solar muon results are not definitive. The experimental results (Table 1 & Figure 5) support the idea of solar muon emission due to the correlation between the distance between the Sun and the Earth's surface and the observed muon coincidence rates, with a noon peak of 62.64 Hz and a minimum value of only 32.46 Hz at 2AM the following morning, with the Sun facing roughly the other side of the planet.

Error for the day-night experiment was calculated using standard deviation of the coincidence frequencies, as shown in Figure X's error bars. Due to the magnitude of error involved, approximately +/-10 on average, our conclusions regarding noticeable solar muon activity are only probable, not certain, but the data does certainly seem to suggest that the solar muon effect does exist. Error in this experiment could have been caused by irregularities/aging of the scintillator material or PMTs, QuarkNet hardware malfunction, programming error in the Python script, unexpected muon shower or other non-solar fluctuation in natural muon emission, unknown shielding effects of the experiment site, or unknown unnatural sources of muon emission rates. These could have caused the numbers to be biased or entirely incorrect, though their coherency makes this less likely.

Over the course of the project, much was learned about interfacing with scientific hardware as well as the particle physics involved in the experiments. Unfortunately, much time was taken up by learning and trying to understand the somewhat arcane operation of QuarkNet board, which left less time in which to perform the actual experiments.

Future work could include re-doing of the experiments for longer time periods to further minimize error, calibration and plateauing of scintillator “D”, and enhancement of the program produced including proper handling of the integer overflow of the QuarkNet incidence and coincidence counters, and parsing of the live “data words” stream the board

makes available but was not utilized by this project. Unfinished code for the data words processing is included in the *quarknet.py* source. Hopefully, future classes or projects can build on the experience and tools gleaned from this experiment to do more interesting or complex experiments.

VII. Bibliography

[1] Nichols A. Romero & Mukund T. Vengalattore, Speed and Decay of Cosmic Ray Muons, Junior Physics Laboratory, Massachusetts Institute of Technology, 1998
[2] The NESTOR Collaboration (Aggouraset. al), A measurement of the cosmic-ray muon flux with a module of the NESTOR neutrino telescope, Astroparticle Physics 23, p377–392, 2005
[3] Bai et. al, Muon flux at the geographical South Pole, Astroparticle Physics, Volume 25, Issue 6, p361-367, 7/2006
[4] Sanuki et. al, Atmospheric neutrino and muon fluxes, 29th International Cosmic Ray Conference Pune 00, p101–104, 2005
[5] Ochanov, Calculation of the atmospheric muon flux motivated by the ATIC-2 experiment, 30th International Cosmic Ray Conference, 6/2007
[6] Wikipedia – The Free Encyclopedia, Muon, Wikimedia Foundation Inc., http://en.wikipedia.org/wiki/Muon, last modified 5/08/2008 3:14PM UTC
[7] Wikipedia – The Free Encyclopedia, Pions, Wikimedia Foundation Inc., http://en.wikipedia.org/wiki/Pions, last modified 3/23/2008 1:06PM UTC
[8] Wikipedia - The Free Encyclopedia, Scintillator, Wikimedia Foundation Inc., http://en.wikipedia.org/wiki/Scintillator, last modified 3/22/2008 10:13AM UTC
[9] Jeff Tyson, How Night Vision Works, Howstuffworks, http://electronics.howstuffworks.com/nightvision.htm
[10] Rylander et. al, QuarkNet/Walta/CROP Cosmic Ray Detectors User's Manual, 08/2004
[11] Python programming language, version 2.5, Guido van Rossum et al., Python Software Foundation, http://python.org, 4/2008
[12] Dr. James Dann, Applied Science Research class, Menlo School, Atherton, CA, 2007-2008

Acknowledgments

The author/experimenter would like to take this opportunity to thank:

- Dr. James Dann for coming up with the idea for this project and helping explain some of the physics and electronics behind the experiment to me
- Jeff Rylander, Tom Jordan, R.J. Wilkes, Hans-Gerd Berns, Richard Gran, Fermilab, and the University of Washington for writing the user's manual for QuarkNet
- the myriad of people who developed the QuarkNet data acquisition unit
- the Menlo School Technology Department for finding and letting me use their serial cable and gender changer
- Chris Liechti, developer of the PySerial library which made writing my program infinitely easier
- the Applied Science Research class of 2008 for providing me encouragement and entertainment while I agonizingly waited for my program to gather data

Appendix A. - Python scripts

Please note that line-wrapping by the word processor may have slightly altered the sources' whitespace.

quarknet.py – user-friendly console interface to the QuarkNet DAQ board

```
#!/usr/bin/env python
#Program to facilitate interaction with QuarkBoard
#Copyright 2008 Christopher V. Rebert
from __future__ import division
import serial
from time import sleep, time, strftime
from menulib import *
from sys import exit
from itertools import izip, chain, repeat
from time import time, sleep
from csv import writer
from os.path import exists

NEWLINE = '\r'#Quarkboard's newline char
PORT = None# serial connection to QuarkBoard
TRIGGER_PERIOD = -1#'d' in the manual
TMC_DELAY = -1#'w' in the manual
NS_PER_TICK = 24#tick = 24 nanoseconds
DATA = None #log file
DATA_DIR = "C:/Documents and Settings/MenloTech/Desktop/"#None #folder
to store logs in
DATA_EXT = ".csv" #extension of log files
OUTPUT = True

def writeln(s):
    '''Write line of text to board with proper newline char'''
    PORT.write(s + NEWLINE)

if False:##__debug__:
    old_write = writeln
    def new_write(s):
        print
        print "@@@", s
        old_write(s)
    writeln = new_write

def readline():
    '''Read a line of output from the board'''
    line = PORT.readline().strip()
    if __debug__: pass ##print "GOT:", line
    return line

BOARD_DELAY = 1#in seconds
def eatLine(n=1):
    '''Retrieve and discard next output line from the board'''
    for i in xrange(n):
        sleep(BOARD_DELAY)#wait for board to catch up
        readline()
```

```
def hexSans0x(n):
    '''Returns string representing integer "n" in base 16 without
leading "0x"'''
    return hex(n)[2:].upper()

##def _int2bin(n, bits=8):
##    '''Returns int "n" represented in binary as a string, using
"count" number of digits'''
##    return "".join([str((n >> y) & 1) for y in range(count-1, -1,
-1)])
##
##def hex2binDigits(hexNum):
##    #convert from hex to string of binary digits
##    #then reverse string so digits[n] is nth bit (n starts @ 0)
##    digits = ''.join(reversed(_int2bin(int(hexNum, 16))))
##    return digits

def binary2int(bin):
    return int(''.join(reversed(bin)), 2)

#utility function from itertools recipes
def groupIn(n, iterable, padvalue=None):
    "grouper(3, 'abcdefg', 'x') --> ('a','b','c'), ('d','e','f'),
('g','x','x')"
    return list(izip(*[chain(iterable, repeat(padvalue, n-1))]*n))

def connect():
    '''Connect to serial port QuarkBoard is connected to and output
connection details.
    Returns opened PySerial serial.Serial object, or exits on
failure.'''
    global PORT
    print "Opening serial port...",
    try:
        PORT = serial.Serial(port=0, baudrate=19200, xonxoff=True)
    except serial.SerialException, e:
        print "failed"
        print "Error while connecting:", e.message
        print "Exiting on error!"
        exit(1)
    else: print "done"
    print 'Connected on serial port #%s' %(PORT.port,)
##    print 'Baud rate:', PORT.baudrate
##    print 'Xon/Xoff enabled?:', PORT.xonxoff
##    print 'Timeout:', PORT.timeout,
##    if PORT.timeout is not None: print "seconds"
##    else: print
    stopCounting()

VERIFICATION_GREETING = "Quarknet Scintillator Card"
EXTRA_GREETING_LINES = 3
RESET_DELAY = 1
def resetBoard():
    '''Reset board to defaults. Also confirms that board is connected'''
    if OUTPUT: print "Resetting board configuration to defaults..."
    writeln("RE")
    sleep(RESET_DELAY)
```

```
    if VERIFICATION_GREETING not in readline():
        if OUTPUT: print "failed"
        raise IOError, "Card not connected (properly)"
    #QUESTION: add 'SA 1' or 'SA 2' here???? try both ways
    eatLine(EXTRA_GREETING_LINES)
    if OUTPUT: print "done"

##def disableCounters():
##    '''Disable coincidence counters'''
##    print "Disabling coincidence counters...",
##    writeln("CD")
##    eatLine()
##    print "done"

def enableCounters():
    '''Re-enable coincidence counters and start displaying data lines'''
    if OUTPUT: print "Enabling coincidence counters...",
    writeln("CE")
    eatLine(2)
    if OUTPUT: print "done"

def setupChannels(fold, channels):#bit-twiddling has been checked to be
theoretically correct
    '''Sets coincidence level and enables desired channels.
    fold - int indicating coincidence level required
    channels - list of bools specifying which channels on/off'''
    if len(channels) != 4: raise ValueError, "On/off state not
specified for all channels"
    if fold < 1 or fold > 4: raise ValueError, "Invalid # of channels
specified"
    if OUTPUT: print "Enabling channels & setting coincidence level...",
    leftChar = hexSans0x(fold - 1)
    channels.reverse()
    #convert channels from bools to ints (0 or 1)
    #then combine into binary string, and convert that to hex
    rightChar = hexSans0x(int(''.join([str(int(chan)) for chan in
channels]), 2))
    writeln("WC 00 "+leftChar+rightChar)
    eatLine(2)
    if OUTPUT: print "done"

def stopCounting():
    '''Stop incidence and coincidence counters.'''
    global OUTPUT
    if OUTPUT: print 'Stopping counters...',
    oldOutput = OUTPUT
    OUTPUT = False
    setupChannels(4,[False]*4)
    OUTPUT = oldOutput
    if OUTPUT: print 'done'

def setTriggerPeriod(ticks):
    '''Set (in ticks) how close time pulses must be to cause a
trigger'''
    global TRIGGER_PERIOD
    #period is difference between memory cells 01 and 02, so set cell
01 to 0
```

```
    TRIGGER_PERIOD = ticks
    if OUTPUT: print 'Setting trigger period ["d"]...',
    writeln("WT 01 00")
    eatLine(2)
    writeln("WT 02 %s" % (hexSans0x(ticks).zfill(2),))
    eatLine(2)
    if OUTPUT: print "done"

def setGateWidth(ticks):
    '''Set "gate width" ("w" in the board manual) to given value in
ticks'''
    if OUTPUT: print 'Setting gate width ["w"]...',
    TMC_DELAY = ticks
    bits = hexSans0x(ticks).zfill(4)
    #4 hex digit number split across 2 memory cells
    writeln("WC 02 %s" % (bits[2:],))
    eatLine(2)
    writeln("WC 03 %s" % (bits[:2],))
    eatLine(2)
    if OUTPUT: print 'done'

class ScalerData(object):
    '''Represents output of "DS" command'''
    def __init__(self, scalers, coins):#, timeOverThresh):
        self.scalers = scalers
        self.coins = coins#coincidence count
        #self.timeOver = timeOverThresh

def readScalers():
    '''Return ScalerData obj representing output from "DS" cmd'''
    if OUTPUT: print "Reading scaler values...",
    writeln("DS")
    eatLine()#eat echoing of cmd
    pairs = readline().strip().split(' ')[1:]#split @ spaces, ignore
leading "DS" field
    hexVals = [pair.split('=')[1] for pair in pairs]#grab hex number
from 'S1=846738'-like fields
    scalers = [int(val, 16) for val in hexVals]#convert hex to integer
    scalers.pop()#discard S5, the 1PPS signal, GPS-related
    if OUTPUT: print 'done'
    print 'Scaler values:', scalers
    return ScalerData(scalers, scalers[4])

def prompt4chans():
    '''Prompt user for coincidence and channel settings'''
    channels = [ask4bool("Enable scintillator %s?" % i) for i in
xrange(4)]
    nFold = ask4num("Detection fold to be considered a coincidence", 1,
4)
    setupChannels(nFold, channels)

def prompt4period():
    '''Prompt user for coincidence trigger period'''
    setTriggerPeriod(ask4num('Period ["d"] for triggers to be
considered a coincidence [in %s ns ticks]' % NS_PER_TICK, 2))

def prompt4width():
```

```
    '''Prompt user for gate width'''
    setGateWidth(ask4num('Gate width ["w"]', TRIGGER_PERIOD - 1))

def terminal():
    '''Gives user terminal-like access to board'''
    connect()
    print 'You now have terminal access to the Quarkboard.'
    print 'Enter "end" to terminate the session.'
    while True:
        input = raw_input("QuarkNet> ")#get cmd from user
        if input.lower() == 'end':
            print 'Exiting...'
            exit(0)
        writeln(input)
        sleep(1)
        while PORT.inWaiting() > 0:#display output
            print '========= ', readline()

def filename2logPath(filename, ext):
    '''Generates full path to log file based on given filename'''
    return DATA_DIR + filename + ext

def makeValidator(ext):
    def _validator(name):
        '''Checks whether a filename has already been used in the log
directory'''
        path = filename2logPath(name, ext)
        return not exists(path)
    return _validator

def ask4aLog(prompt, ext):
    '''Prompt user for log file name and return full path of desired
log file'''
    return filename2logPath(ask4file(prompt, makeValidator(ext)), ext)

def prompt4log():
    '''Prompt user for file to log to and open the file.'''
    createLog(ask4aLog("Filename to log data to?", DATA_EXT))

def createLog(filepath):
    '''Create log file or die on error'''
    global DATA
    if OUTPUT: print "Creating log file '%s'..." % filepath,
    try:
        DATA = file(filepath, 'w')
    except EnvironmentError, err:
        print
        print "Error opening log file:", str(e)
        print "Exiting on error..."
        raise SystemExit
    else:
        if OUTPUT:
            print "done"
            print
            print

##def log(line):
```

```
##    '''Write line to log file.'''
##    LOG.write(line+"\n")

##def parseEventLine(line):
##    fields = line.split(' ')[:9]#chop off GPS fields
##    log(','.join(fields))
##    #triggerCount = int(fields[0], 16)
##    #riseFalls = [(parseRise(rise), parseFall(fall)) for rise, fall
in groupIn(2, fields[1:])]
##
##def parseRise(hexVal):
##    bits = hex2binDigits(hexVal)
##    ticks = binary2int(bits[:5])
##    valid = bool(int(bits[5]))
##    newTrigger = bool(int(bits[7]))
##
##def parseFall(hexVal):
##    bits = hex2binDigits(hexVal)
##    ticks = binary2int(bits[:5])
##    valid = bool(int(bits[5]))

DEFAULT_D = 6
DEFAULT_W = 10
def calibrationExpmt():
##    f = file(ask4aLog("Filename to log incidents to?", ".csv"), 'wb')
    w = writer(DATA)#f)
    print 'Connect scintillator to QuarkBoard port #0.'
    wait4user()
    w.writerow(["scintillator", ask4str("Scintillator's designation")])
    w.writerow(["trigger period [d] (ticks)", DEFAULT_D])
    w.writerow(["gate window [w] (ticks)", DEFAULT_W])
    fieldNames = ["voltage", "time (s)", "scaler", "scal freq (Hz)"]
    w.writerow(fieldNames)
    DURATION  = ask4num("Duration of each run (sec)", kind=int)
    print "Connect scintillator to be calibrated to port #0."
    wait4user()
    try:
        while True:
            connect()
            v = ask4num("Scintillator voltage (V)?", kind=float)
            setTriggerPeriod(DEFAULT_D)
            setGateWidth(DEFAULT_W)
            init = _ask4boardValue()
            print "***Gathering data. This will take ~"+str(DURATION),
"seconds...***"
            prevOutput = OUTPUT
            OUTPUT = False
            start = time()
            setupChannels(1, [True] + 3*[False])
            sleep(DURATION)
            s = readScalers()
            end = time()
            print "***Data gathered!***"
            OUTPUT = True
            scal = s.scalers[0] - init
##            c = s.coins
            duration = end-start
```

```
            freq = scal/duration
            row = [v, duration, scal, freq]
            print "Incidence frequency:", freq, "Hz"
            w.writerow(row)
            PORT.close()
            if ask4bool("Done with this scintillator?"): break
            print "Power-cycle the QuarkBoard."
            print "Also, change the voltage setting on the scintillator
being calibrated."
            wait4user()
            DATA.flush()
    finally:
        DATA.close()
    exit(0)

def _ask4counter(prompt):
    '''Asks the user for a number corresponding to a port #.'''
    return ask4num(prompt, mini=0,maxi=3,kind=int)

def coincidenceCount():
    return readScalers().coins

def _runCoinExpmt(chans, PLATEAU_DURATION):
    global OUTPUT
    connect()
    setTriggerPeriod(6)
    setGateWidth(10)
    offset = coincidenceCount()
    print "***Gathering datum. Please wait", '~'+str(PLATEAU_DURATION),
"seconds...***"
    oldOutput = OUTPUT
    OUTPUT = False
    start = time()
    setupChannels(2, chans)
    sleep(PLATEAU_DURATION)
    coins = readScalers().coins - offset
    PORT.close()
    end = time()
    OUTPUT = oldOutput
    print "***Done gathering datum***"
    elapsed = end - start
    freq = coins / elapsed
    return freq

ALL_OFF = [False]*4
def _1plateauDatum(top, mid, bot, PLATEAU_DURATION):
    print "*****Gathering datapoint...*****"
    first = ALL_OFF[:]
    first[top] = first[mid] = True
    second = ALL_OFF[:]
    second[top] = second[bot] = True
    numer = _runCoinExpmt(first, PLATEAU_DURATION)
    print "Power-cycle the QuarkBoard."
    wait4user()
    denom = _runCoinExpmt(second, PLATEAU_DURATION)
    ratio = numer / denom
    print "*****Datapoint calculated*****"
```

```
    print "Coincidence frequency ratio:", ratio
    return ratio

def plateauExpmt():
    PLATEAU_DURATION = ask4num("Duration of one plateauing experiment
run (sec)", mini=0, kind=float)
    plateaued = 1
    top = 0
    bot = 2
    print 'Connect 3 scintillators in a "sandwich" configuration as
follows:'
    print 'Top: scintillator connected to port #%s' % top
    print 'Middle: scintillator to be plateaued, connected to port #%s'
% plateaued
    print 'Bottom: scintillator connected to port #%s' % bot
    wait4user()
##    f = file(ask4aLog("Log file to output plateauing data to?",
'.csv'), 'w')
    log = writer(DATA)
    log.writerow(["Scintillator voltage (V)", "Coincidence frequency
ratio"])
    while True:
        v = ask4num("Scintillator #%s voltage (V)?" % plateaued,
kind=float)
        freq = _1plateauDatum(top, plateaued, bot, PLATEAU_DURATION)
        log.writerow([v, freq])
        DATA.flush()
        if ask4bool("Done plateauing?"): break
    DATA.close()


def dayNight():
    setupParameters()
    nPeriods = ask4num("Number of periods?")
    duration = ask4num("Duration of periods (min)?", mini=2)*60
    delay = ask4num("Time to wait before beginning experiment (sec)?")
    w = writer(DATA)
    w.writerow(["start", "end", "elapsed (s)", "coincidences",
"frequency (Hz)"])
    DATA.flush()
    base = time()
    doneInit = time()
    delay -= doneInit - base
    sleep(delay)
    for i in xrange(nPeriods):
        togo = duration
        oldcount = 0
        total = 0
        start = time()
        while togo > 0:
            count = coincidenceCount()
            if count < oldcount:
                total += oldcount
                oldcount = count
            sleep(60)
            togo -= 60
        end = time()
```

```
        total += count
        elapsed = end-start
        w.writerow([strftime(start), strftime(end), elapsed, total,
total/elapsed])
        DATA.flush()
    DATA.close()


def prompt4DataDir():
    global DATA_DIR
    DATA_DIR = ask4dir("Log & data directory")

def setupParameters():
    connect()
    prompt4log()
    prompt4period()
    prompt4width()
    prompt4chans()

ACTIVITIES = (("Calibrate scintillator",calibrationExpmt),
              ("Terminal interface to Quarkboard",terminal),
              ("Plateau scintillator",plateauExpmt),
              ("Perform day-night solar muon experiment",dayNight))
try:
    #prompt4logDir()
    prompt4log()
    ask4choice("Choose activity", ACTIVITIES)()

finally:
    print "Goodbye!"
    print "Closing port...",
    try: PORT.close()
    except: pass
    print "done"
    print "Closing log file...",
    try: LOG.close()
    except: pass
    print "done"
```

menulib.py – simple console-based menu-driven user interface library

```
from os.path import isdir
from string import ascii_letters as _LETTERS, digits as _DIGITS
_ACCEPTABLE = set(_LETTERS + _DIGITS + '_- ')
del _LETTERS, _DIGITS

def ask4bool(question):
    '''Asks the user a yes/no question. Returns a bool indicating their
response.'''
    question += ": "
    while True:
        input = raw_input(question).lower()
        if input in ('yes', 'y'): return True
        elif input in ('no', 'n'): return False
        print "Invalid input. Must be either (y)es or (n)o."
```

```
def ask4num(question, mini=None, maxi=None, kind=int):
    '''Asks the user for an integer within the given range.'''
    rng = ''.join(('(', str(mini) if mini is not None else '', '-',
str(maxi) if maxi is not None else '', ')'))
    mid = " " + rng
    if mini is None and maxi is None:
        mid = ""
    prompt = question + mid + ": "
    while True:
        try:
            s = raw_input(prompt)
            if s != '0': s = s.lstrip('0')
            val = kind(s)
        except ValueError: print "Invalid input. Must be a valid
decimal number."
        if (maxi is not None and val > maxi) or (mini is not None and
val < mini):
            print "Invalid input. Not within valid range %s." % rng
        else: return val

def ask4str(question, validator=None, errMsg="Try again."):
    '''Asks the user for a string for which "validator" returns True.'''
    question += ": "
    error = "Invalid input. "+errMsg
    while True:
        input = raw_input(question)
        if validator and not validator(input): print error
        else: return input

def wait4user():
    '''Does not return until user acknowledges the program'''
    raw_input("Press Enter to continue")
    print
    print


def _validateFilename(name):
    '''Check that proposed filename contains only valid characters.
    Allowed chars are letters, digits, underscores, dashes, and
spaces.'''
    for char in name:
        if char not in _ACCEPTABLE:
            return False
    return True

_BAD_FILENAME_MSG = "Filename must contain only letters, digits,
underscores, dashes, or spaces, and must not already be in use."
def ask4file(prompt, validator):
    '''Asks the user for a filename that passes _validateFilename and
the given "validator"'''
    def validater(s): return _validateFilename(s) and validator(s)
    return ask4str(prompt, validater, _BAD_FILENAME_MSG)

def ask4choice(prompt, titleValPairs):
    '''Asks the user to choose an option from a list.'''
    prompt += ":"
    while True:
```

```
        print prompt
        print '='*(len(prompt)+2)
        for i, titleVal in enumerate(titleValPairs):
            title = titleVal[0]
            print "[%s]"%(i+1), title
        print
        try:
            choice = int(raw_input("Enter the number of your choice:
")) - 1
            if choice > i or choice < 0: raise ValueError
        except:
            print "Invalid input. Try again"
            wait4user()
            continue
        else: break
    print
    print
    return titleValPairs[choice][1]

def ask4dir(prompt):
    '''Asks the user for an existing directory.'''
    prompt += ': '
    while True:
        path = raw_input(prompt)
        if isdir(path): return path
        print "Directory does not exist. Try again."
        wait4user()
```

Appendix B. - Raw count data

Data Table 2: Raw count calibration data for scintillator “A”

Voltage (V)	incidence frequency (Hz)	incidence count	time (s)
0.867	26.19	1258	48.03
0.878	30.40	1460	48.03
0.889	32.69	1570	48.03
0.902	36.62	1759	48.03
0.917	40.18	1930	48.03
0.921	42.24	2029	48.03
0.931	50.86	2443	48.03
0.940	51.13	2456	48.03
0.951	55.86	2683	48.03
1.109	187.02	8983	48.03

Data Table 3: Raw count calibration data for scintillator “B”

Voltage (V)	incidence frequency (Hz)	incidence count	time (s)
0.908	20.51	985	48.03
0.920	22.90	1100	48.03
0.931	24.48	1176	48.03
0.940	27.32	1312	48.03
0.952	31.98	1536	48.03
0.960	31.92	1533	48.03
0.970	36.50	1753	48.03
0.983	39.83	1913	48.03
0.984	42.39	2036	48.03
0.985	41.94	2015	48.05
0.993	46.43	2230	48.03
1.003	48.28	2319	48.03
1.105	100.39	4822	48.03

Data Table 4: Raw count calibration data for scintillator “C”

Voltage (V)	incidence frequency (Hz)	incidence count	time (s)
0.910	16.76	805	48.03
0.921	17.30	831	48.03
0.935	19.86	954	48.03
0.945	19.92	957	48.05
0.950	21.57	1036	48.03
0.963	23.26	1117	48.03
0.973	25.09	1205	48.03
0.982	27.79	1335	48.03
0.995	30.42	1461	48.03
1.019	39.62	1903	48.03
1.110	73.18	3515	48.03

Data Table 5: Raw count calibration data for scintillator “D”

Voltage (V)	incidence frequency (Hz)	incidence count	time (s)
0.892	33.85	1627	48.06
0.902	36.02	1734	48.14
0.910	38.31	1840	48.03
0.914	42.22	2030	48.08
0.925	45.37	2179	48.03
0.942	53.19	2559	48.11
0.953	56.78	2730	48.08
0.964	63.79	3071	48.14
0.975	76.75	3691	48.09
1.100	252.96	12150	48.03