NOAA Technical Memorandum OAR ARL-264

A FAST-RESPONSE DATA ACQUISITION SYSTEM FOR SONIC ANEMOMETERS

Christoph A. Vogel
J. Randy White
David L. Senn

Oak Ridge Associated Universities
Oak Ridge, Tennessee

Thomas S. Wood V

Wood International, Inc.
Oak Ridge, Tennessee

William R. Pendergrass

Atmospheric Turbulence and Diffusion Division
Oak Ridge, Tennessee

Air Resources Laboratory
Atmospheric Turbulence and Diffusion Division
Oak Ridge, Tennessee 37831
March, 2013

noaa NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION

Oceanic and Atmospheric Research Laboratories

Contents

List of Figures

A FAST-RESPONSE DATA AQUISTION SYSTEM FOR SONIC ANEMOMETERS

Christoph A. Vogel[1,2], J. Randy White[1,2], David L. Senn[1,2], Thomas S. Wood V[3], and William R. Pendergrass[1]

[1]NOAA Air Resources Laboratory, Atmospheric Turbulence and Diffusion Division, 456 S. Illinois Ave., Oak Ridge, TN 37830
[2]Oak Ridge Associated Universities, P.O. Box 117, Oak Ridge, TN 37831
[3]Wood International, Inc., Oak Ridge, TN 37830

Abstract

For many applications allowing atmospheric turbulent flow data to be acquired and processed by a datalogger in the field is sufficient. Turbulence statistics are downloaded to a computer and analyses are performed using those statistics. However, if there is a desire to perform more in-depth analyses on the flow, raw data at the maximum sampling rates should be logged. The following technical memorandum describes a data acquisition program, written in the Python programming language, that allows for the logging of raw data from an R. M. Young sonic anemometer. The program was written so that minimal modifications could be made to include other types of sonic anemometers in its ability to log raw data.

1 Introduction

In the course of using sonic anemometers to investigate atmospheric turbulent flow characteristics near the ground, there is frequently a desire to collect raw data as opposed to statistical variables from data processed by dataloggers or other computers. This allows for more in-depth analyses of the flow, and greater assessment of the quality of the data. Currently, in taking raw data, either a commercial product must be used as an interface to the instrument, or a computer program must be written.

Commercial products can be used to log raw sonic anemometer data, however, there are disadvantages. First, the cost of the software can be relatively high. Also, often the software is only available for a specific operating system, one that inherently has a large overhead of processes running, and hence is less than conducive to taking data. Finally, most proprietary software limit the amount of options available, and don't allow modifications to the underlying code. Thus, for a significantly more cost effective way to log high frequency data, and for the ability to develop fully controllable data acquisition code that can run in a variety of operating system environments, a self-written computer program is preferred.

The following describes a measurement system, including computer code, used to obtain raw data from a sonic anemometer. The model used in the testing and out in the field was the R. M. Young 81000. Since sonic anemometers are manufactured differently, significant modifications to the present system would potentially need to be implemented. However, what is described here is an example (and potential template) for successfully logging raw data from a variety of sonic anemometers. Concerning the data acquisition software, the code was written in the python programming language, and optimized to run in the linux operating system environment. Test and field operation was performed on an ASUS 1005HA-PU17 netbook computer running Ubuntu Linux 10.04 LTS.

2 Measurement System

The following describes the measurement system used to test and implement the high frequency data acquisition capability. Attempts were made so that only minor modifications to the code would be needed to adapt to new equipment. The system is described here to establish a baseline as to what operated successfully to aide in making future upgrades.

2.1 Sonic Anemometer

The sonic anemometers used in all testing and measurement campaigns performed at the time of this writing are type R. M. Young model 81000. These anemometers have a nominal maximum output rate of 32 Hz, although testing and analyses have shown that the maximum rates experienced with the particular units used was closer to 31.8 Hz. This sampling rate was determined through precise (time wise) opening and closing of files at particular times, and counting the number of scans (a fixed byte amount) present. This was performed repeatedly over long periods of time at different open/close file intervals to assess consistency so that the reduced output rates were not due to the rejection of bad data.

Nevertheless, the R. M. Young sonics were set to output at 32 Hz, at a 38400 baud rate, with Error Handling set to “Omit Invalid Data”. The RS232 serial output format of the sonic data was “U V W WS WD Ts”, where U is the east-west component of the wind, V the north-south component, and W the vertical component. WS and WD are the two dimensional wind speed and wind direction, while Ts is the temperature inferred from the speed of sound. A set of these six variables is termed a “scan” of data. The redundancy in the wind quantities arises from existing formats used in the field that would have been inconvenient to change due to the affect on processing by other data acquisition systems.

2.2 Serial Device

The interface between the sonic anemometers and the netbook computer was a SerialGear CM-41082 8S RS232 to USB device with optical isolation and surge protection. As indicated by

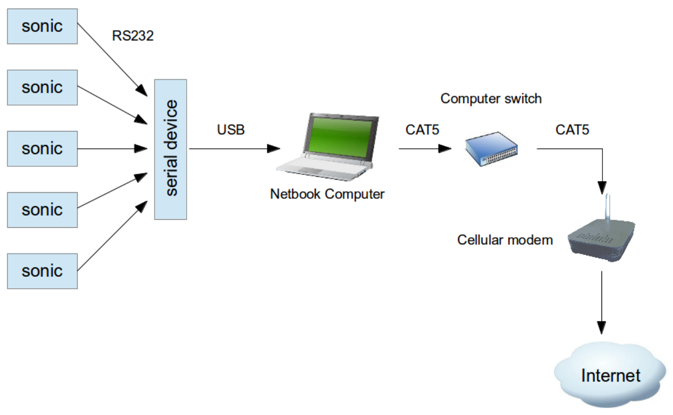

Figure 1: A diagram of the high frequency sonic anemometer data acquisition system

the model the device has 8 RS232 serial ports to which the sonic anemometer serial cables can be attached. The sonic signals are then routed through the device, and output through the USB protocol into the netbook. The 8 ports can easily be addressed in the Ubuntu Linux operating system.

2.3 Netbook Computer

As mentioned, the computer used to acquire the high-frequency data from the sonic anemometers is an ASUS 1005HA-PU17 Netbook running the Ubuntu Linux 10.04 LTS operating system. It has an ATOM N280 1.66 GHz central processing unit and 1 GB of Synchronous DIMM SDRAM. The USB controller is of the Intel N10/ICH7 family. The netbook has dimensions of 262mm(W) x 178mm(D) x 25.9mm(H), and weighs 1.36 kg. A diagram of the high-frequency data acquisition system is shown in Figure 1.

2.4 Ethernet Switch and Cellular Modem

In order to enable communications to multiple devices through the Internet, using one cellular modem as an access point, a Cisco/Linksys 5-port fast ethernet switch was used. Consequently, each computer had a unique local area network (LAN) address that was accessed through assigned ports of the cellular modem. These LAN addresses were set up through port forwarding rules on the cellular modem. The cellular modem model used was a Sierra Wireless AirLink Raven-X Intelligent 3G Gateway device.

3 Python Script

The following describes the python code used to acquire the R. M. Young sonic anemometer data. At this writing there is a first version (Appendix A) written to acquire data from the Summer of 2010 to the Summer of 2011. In the Summer of 2011 the code was rewritten (Appendix B) to improve efficiency, and improve the methods of opening and closing files. The description below highlights the basic structure of the code, and it is left to the reader to investigate the particulars of employing certain python modules. It must be noted that the author of the code is not a professional software developer, and that there most likely are more efficient ways to write the scripts. However, from our standpoint the primary purpose of the script is to successfully achieve its goal, and once that has been accomplished to not spend too much additional time optimizing.

3.1 Version 1

A customary method of writing Python scripts is to declare modules, subroutines, and functions prior to the primary portion of the program. In version 1 of the program, the primary part of the python code begins on line 48 where commands are called from the *optparse* module (PSF, 2012b). Lines 48–55 allow for the entering of command line arguments to specify a station identifier and the serial port through which the data will be acquired. Line 57 sets the Station ID from the *options* parser object, while lines 60–61 set the file name, through the *get_filename* function, and open the data file. Line 63 configures the serial connection, given 1) again the *options* parser object, which contains the serial port number, 2) the baud rate, and 3) the timeout period (in seconds) to wait for data. The serial object *ser0* is set through the *serial* module (Liechti, 2010). The serial input buffer is then flushed (line 66), and the first (possibly partial) line of data is received through *ser0.readline* call. A counter (*cntr*) is then initialized, and a *while* loop is begun to read the data (line 70).

Within the *while* loop, the subroutine *get_sample_scan* is called to read the sonic data and write to file. The subroutine operates initially through a call to the serial object *ser0* to obtain a scan of data (line 17). The data block is assigned to *son_scan* which is then split to the six different variables. A conditional statement checks for the proper variable count, and, if true, multiplies the variables by 100 and converts them to integers. If the conditional statement is false, then the value of *cntr* is incremented and, along with the length of the string and the time, is printed to screen.

Also within this subroutine, on line 18, the current universal time object *tn* is obtained through the *datetime* module (PSF, 2012a). This time is used to time tag the sonic scan in line 27 and contribute to the logic, on lines 28–38, of when to close the current file and open a new one. The issue of time tagging the data is a quality control check, and not intended to actually give the “true” time of the scan. The time written to the data file is a function of not only time between scans, but also of the netbook computer’s scheduler in handling processes.

Finally, once the variables are split from the sonic scan string, they are packed into a binary string using the Python *struct* module (PSF, 2012d). As shown on line 27, three time variables and the four data variables are packed according to the “little-endian” (’$<$’) byte order format: [*unsigned short integer, unsigned short integer, unsigned long integer, short integer, short integer, short integer, short integer*]. The binary string is then written to file.

3.2 Version 2

Appendix B gives a successive improved version (Version 2) of the python data acquisition program. To start, there is an additional data file period specification for the command line parameters (lines 56 – 58). The number of minutes to which a file is written can be set, as opposed to hard coding the value as in Version 1. The program proceeds similarly, with an additional flushing of the output as well as input buffers, to the *while* conditional statement starting the *get_sample_scan* loop (lines 77–78).

The subroutine *get_sample_scan*, however, was rewritten to utilize the python *os* module for its operating system interfaces (PSF, 2012c). As in Version 1, the serial string is obtained from the sonic anemometer, and the date and time is obtained through the *datetime* module. Since the code operates such that new files are generated according to the file periods set by the *ftim* command line argument, a check is performed through the *check_data_file* subroutine, to establish whether data will be written to an existing file or a new file. If a new file generation is warranted then, as illustrated in lines 18–22, a new file descriptor *fd* is generated, the previous file object is closed, and a new file object is created based on the new file descriptor. The sonic anemometer variables are then obtained as in Version 1 and written to file.

4 Discussion

The two versions of the python scripts described here work well for the measurement systems currently being used. However, it is important to recognize that the versions described will change as different instruments are utilized to obtain fast-response wind and temperature information, and as different communication strategies are implemented in getting data from instrumentation to researchers. Even as of this writing potential improvements are being evaluated. However, in part because the python scripting and the operating systems are virtually at no cost, function in the open-source environment, and allow for greater freedom in regulating the operation of the measurement system, the measurement suite can be viewed as a good model upon which other systems could evolve.

Appendix A Python Program for Logging R. M. Young Sonic Data: Version 1

```
from optparse import OptionParser
import datetime
import serial
import struct


def get_filename(stid):
    filename=datetime.datetime.strftime(datetime.datetime.utcnow(),stid+'UTC%Y%j_%H%M.raw')
    return filename

def get_sample_scan():
    global son_scan
    global stid
    global f0
    global f1
    global flg
    global cntr
    son_scan=ser0.readline(eol='\r')
    tn=datetime.datetime.utcnow()
    sondata=son_scan.split()
    if len(sondata) == 6:
        v0=int(float(sondata[0])*100.)
        v1=int(float(sondata[1])*100.)
        v2=int(float(sondata[2])*100.)
        v5=int(float(sondata[5])*100.)
        sondata=[]
        ttn=tn.minute+60
        ds=struct.pack('<HHLhhhh',tn.minute,tn.second,tn.microsecond,v0,v1,v2,v5)
        if ttn % 30 == 0 and tn.second == 0 and tn.microsecond < 50000 and flg==0:
            print tn
            f0.close()
            fname=get_filename(stid)
            f0=open(fname,'wb')
            f0.write(ds)
            flg=1
            cntr=0
        else:
            f0.write(ds)
            flg=0
    else:
        cntr=cntr+1
        print cntr,'len␣=␣',len(sondata),tn



##################################################################
# main start

parser=OptionParser()                                  # obtain command line values
parser.add_option("-p","--srdv",action="store",
                  type="string",dest="srdv",
                  help="serial␣device")
parser.add_option("-r","--stid",action="store",
                  type="string",dest="stid",
                  help="station␣identifier")
(options, args) = parser.parse_args()

stid=options.stid                                      # set station id from input
son_scan=""                                            # set up null string to concatenate

fname=get_filename(stid)                               # open file to which to write
f0=open(fname,'wb')

```

```
ser0 = serial.Serial(options.srdv,38400,timeout=.05)          # configure the serial connection
print ser0.portstr

ser0.flushInput()                                              # flush the sonic buffer
ser0.readline(eol='\r')

cntr=0
while 1:
    get_sample_scan()
```

Appendix B Python Program for Logging R. M. Young Sonic Data: Version 2

```python
# Task: Collects maximum data output (32 Hz) from an R. M. Young sonic anemometer
#       and outputs to file.
# Author: C. A. Vogel, NOAA/OAR/ARL/ATDD
# Last update: 30 July 2011

from optparse import OptionParser
import os,serial,struct
import datetime

def get_filename(stid):
    filename=datetime.datetime.strftime(datetime.datetime.utcnow(),stid+'UTC%Y%j_%H%M.raw')
    return filename

def check_data_file(to):
    global stid,ftim,f0,cntr
    ti=to.minute+60
    if ti % ftim == 0 and to.second == 0 and to.microsecond < 60000:
        try:
            fname=get_filename(stid)
            fd=os.open(fname,os.O_CREAT|os.O_EXCL|os.O_RDWR,0644)
            f0.close()
            print to
            f0=os.fdopen(fd,'wb')
            cntr=0
        except:
            pass

def get_sample_scan():
    global stid,ftim,f0,cntr
    son_scan=ser0.readline(eol='\r')
    tn=datetime.datetime.utcnow()
    check_data_file(tn)
    sondata=son_scan.split()
    if len(sondata) == 6:
        v0=int(float(sondata[0])*100.)
        v1=int(float(sondata[1])*100.)
        v2=int(float(sondata[2])*100.)
        v5=int(float(sondata[5])*100.)
        ds=struct.pack('<HHLhhhh',tn.minute,tn.second,tn.microsecond,v0,v1,v2,v5)
        f0.write(ds)
        sondata=[]
    else:
        cntr=cntr+1
        print cntr,'len␣=␣',len(sondata),tn


# -----------------------------------------------------------------------------
# main start

parser=OptionParser()                                  # define command line parameters
parser.add_option("-p","--srdv",action="store",
                  type="string",dest="srdv",
                  help="serial␣device")
parser.add_option("-r","--stid",action="store",
                  type="string",dest="stid",
                  help="station␣identifier")
parser.add_option("-k","--ftim",action="store",
                  type="string",dest="ftim",
                  help="data␣file␣period␣(min)")
(options, args) = parser.parse_args()
# -----------------------------------------------------------------------------

stid=options.stid                                      # set station id from input
```

```
ftim=int(options.ftim)                                  # set file period from input

fname=get_filename(stid)                                # open file to which to write
f0=open(fname,'wb')

son_scan=""                                             # set up null string to concatenate
cntr=0                                                  # initialize 'no data' counts

ser0 = serial.Serial(options.srdv,38400,timeout=.05)    # configure the serial connection
print ser0.portstr

ser0.flushInput()                                       # flush the sonic input buffer
ser0.flushOutput()                                      # flush the sonic output buffer

while 1:
    get_sample_scan()
```

References

Liechti, C., 2010: *pySerial*. http://pyserial.sourceforge.net/.

PSF, 2012a: *Python v2.6.8 documentation: datetime — Basic date and time types*. Python Software Foundation, http://docs.python.org/release/2.6.8/library/datetime.html.

—, 2012b: *Python v2.6.8 documentation: optparse — Parser for command line options*. Python Software Foundation, http://docs.python.org/release/2.6.8/library/optparse.html, v.1.5.3.

—, 2012c: *Python v2.6.8 documentation: os — Miscellaneous operating system interfaces*. Python Software Foundation, http://docs.python.org/release/2.6.8/library/os.html.

—, 2012d: *Python v2.6.8 documentation: struct — Interpret strings as packed binary data*. Python Software Foundation, http://docs.python.org/release/2.6.8/library/struct.html.