DATA PRODUCT SPECIFICATION FOR TURBULENT POINT WATER VELOCITY

FROM NORTEK VECTOR

ON ENDURANCE MFN AND ENDURANCE BEP

Version 1-01
Document Control Number 1341-00780
2012-08-15

Consortium for Ocean Leadership
1201 New York Ave NW, 4th Floor, Washington DC 20005
www.OceanLeadership.org

in Cooperation with

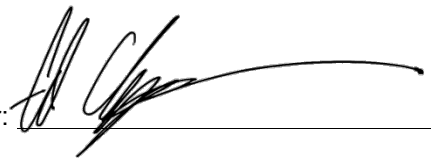
University of California, San Diego
University of Washington
Woods Hole Oceanographic Institution
Oregon State University
Scripps Institution of Oceanography
Rutgers University

Document Control Sheet

Version	Date	Description	Author
0-01	2012-04-20	Initial Draft	M. Vardaro
0-02	2012-06-15	Incorporated comments from focused review and added magnetic correction	M. Vardaro
0-03	2012-06-29	Incorporated comments from formal review and moved Nobska to 1341-00781	M. Vardaro
0-04	2012-07-09	Clarified where the L0 VELPTTU product is captured.	S. Webster
0-05	2012-07-09	Added parameter names and descriptions.	S. Webster
1-00	2012-07-17	Initial Release	E. Chapman
1-01	2012-08-15	Corrected units in 2.2.2.	S. Webster

Signature Page

This document has been reviewed and approved for release to Configuration Management.

OOI Senior Systems Engineer: 

Date:2012-07-17

This document has been reviewed and meets the needs of the OOI Cyberinfrastructure for the purpose of coding and implementation.

OOI CI Signing Authority: 

Date: 2012-07-17

Table of Contents

1 Abstract

This document describes the computation used to calculate the OOI Level 1 Turbulent Point Water Velocity VELPTTU data product, which is calculated using the data from the Nortek Vector instrument for Series C and D of the VEL3D instrument. This document is intended to be used by OOI programmers to construct appropriate processes to create the L1 VELPTTU product.

2 Introduction

2.1 Author Contact Information

Please contact Michael Vardaro (mvardaro@coas.oregonstate.edu) or the Data Product Specification lead (DPS@lists.oceanobservatories.org) for more information concerning the computation and other items in this document.

2.2 Metadata Information

2.2.1 Data Product Name

The OOI Core Data Product Name for this product is

- VELPTTU

The OOI Core Data Product Descriptive Name for this product is

- Turbulent Point Water Velocity

This Data Product is comprised of 3 parameters:

Identifier	Name	Description	L0 Units	L1 Units
VELPTTU-VLE	eastward sea water velocity	East component in earth coord.	m/s	m/s
VELPTTU-VLN	northward sea water velocity	North component in earth coord.	m/s	m/s
VELPTTU-VLU	upward sea water velocity	Up component in earth coord.	m/s	m/s

2.2.2 Data Product Abstract (for Metadata)

The OOI Level 1 Turbulent Point Water Velocity core data product measures current velocity, which can be used to derive turbulence data. The Nortek Vector measures current velocity within a 1 cm^3 box by transmitting a sound pulse from the center transducer. The instrument measures the Doppler shift introduced by the reflections from particles suspended in the water, which is picked up by the 3 receivers that surround the center transducer.

2.2.3 Computation Name

Not required for data products.

2.2.4 Computation Abstract (for Metadata)

This computation computes the OOI L1 Turbulent Point Water Velocity core data product, which is calculated using the raw binary data from the VEL3D family of instruments. The raw binary data is converted into ASCII text, magnetically corrected, and reported with associated metadata.

2.2.5 Instrument-Specific Metadata

See Section 4.4 for instrument-specific metadata fields that must be part of the output data.

2.2.6 Data Product Synonyms

n/a

2.2.7 Similar Data Products

A similar product that this data product may be confused with is the Mean Point Water Velocity, VELPTMN from the VELPT family of instruments, or VELPROF, from the ADCP family of instruments.

2.3 Instruments

For information on the instruments from which the L1 Turbulent Point Water Velocity core data product inputs are obtained, see the VELPTTU Processing Flow document (DCN 1342-00780). This document contains information on instrument classes and make/models; it also describes the flow of data from the VEL3D instruments through all of the relevant QC, calibration, and data product computations and procedures.

Please see the Instrument Application in the SAF for specifics of instrument locations and platforms.

2.4 Literature and Reference Documents

Vector Current Meter User Manual N300-100, Rev H (2005) Nortek AS
(see Alfresco: *DPS Artifacts >> 1341-00780_VELPTTU >> Vector_Manual_RevH.pdf)*
System Integrator Manual (2011) Nortek AS
(see Alfresco: *DPS Artifacts >> 1341-00780_VELPTTU >> system-integrator-manual-september-2011.pdf)*

2.5 Terminology

2.5.1 Definitions

N/A

2.5.2 Acronyms, Abbreviations and Notations

General OOI acronyms, abbreviations and notations are contained in the Level 2 Reference Module in the OOI requirements database (DOORS). The following acronyms and abbreviations are defined here for use throughout this document.

2.5.3 Variables and Symbols

FS = Fractional seconds (indicates that time is being measured in hundredths of a second)
VA = Raw velocity (axis A), HEX format
VB = Raw velocity (axis B), HEX format
VC = Raw velocity (axis C), HEX format
VD = Raw velocity (axis D), HEX format
E = xx = Corrected East velocity component in earth coordinates, in cm/s
N = yy = Corrected North velocity component in earth coordinates, in cm/s
W = Corrected Up velocity component in earth coordinates, in cm/s
T = Temperature, in °C
P = Pressure, in dbar
MX = X-axis direction cosine of the magnetic heading sensor
MY = Y-axis direction cosine of the magnetic heading sensor
Pitch = Angle of pitch, in °
Roll = Angle of roll, in °

3 Theory

3.1 Description

The Nortek Vector VEL3D instrument measures current velocity within a 1 cm^3 box by transmitting a sound pulse from the center transducer. The instrument measures the Doppler shift introduced by the reflections from particles suspended in the water, which is picked up by the 3 receivers that surround the center transducer. That data is combined with position information from an internal compass to create a raw binary data file (VELPTTU L0 data product) that is converted into ASCII text and reported as the VELPTTU L1 data product and associated metadata.

3.2 Mathematical Theory

N/A

3.3 Known Theoretical Limitations

Too few or too many scattering particles can limit the performance of the Nortek Vector.

3.4 Revision History

No revisions to date.

4 Implementation

4.1 Overview

This is a simple process of converting the Nortek binary files and parsing out the relevant outputs, namely the three velocity components (East, North, and Up), and all associated metadata. The instrument shall be configured to always generate data in earth coordinates, which are converted from beam coordinates using the manufacturer's onboard software. The flow direction is reported in "oceanographic convention" for direction – i.e. "direction to", as in "North" is flow towards the north. The compass heading is affected by the earth's magnetic field. As part of the generation of the L1a core product, a rotation on the vector will be applied. For input formats and relevant data, see below.

4.2 Inputs

Inputs are:

- L0 VELPTTU velocity data product output from the Nortek Vector driver
- Magnetic Variation ($magvar$). See section 4.3 Step 3 below.

Input Data Formats:
The Vector software creates the following binary files:

• The *.hdr file (Header Data) is a self-documented table that contains the detailed data format of all the other ASCII files.

• The *.dat files (Data File) contain velocity and pressure data at the full sample rate. The variables that make up the L0 VELPTTU data product are shown in bold in the table below.

• The *.vhd files (Burst Header Data) contain a time stamp and noise information from the beginning and end of each sampling burst

• The *.sen files (System Data) contain system data such as the time/date, compass, tilt, temperature, battery voltage, etc. These data are sampled once per second.

The content of each file is described in the tables below. These files can be converted to ASCII format files using the sample code in Appendix A. Once translated, the relevant information will

be corrected for the effect of the earth's magnetic field and reported as East, North and Up components. Relevant metadata from these files will also be stored (see section 4.4).

Sensors data (.sen)

Column	Type	Unit
1	Month	
2	Day	
3	Year	
4	Hour	
5	Minute	
6	Second	
7	Error code	
8	Status code	
9	Battery	Volts
10	Sound speed	m/s
11	Heading	°
12	Pitch	°
13	Roll	°
14	Temperature	°C
15	Analog input	
16	Checksum	1= failed, 0 = OK

Burst header data (.vhd)

Column	Type	Unit
1	Month	
2	Day	
3	Year	
4	Hour	
5	Minute	
6	Second	
7	No. of velocity samples	
8	Noise amplitudes (beam 1)	counts
9	Noise amplitudes (beam 2)	counts
10	Noise amplitudes (beam 3)	counts
11	Noise correlation (beam 1)	%
12	Noise correlation (beam 2)	%
13	Noise correlation (beam 3)	%

Velocity data (.dat)

Column	Type	Unit
1	Burst counter	
2	Ensemble counter (0–65535)	
3	**Velocity (beam 1, x or east)**	**m/s**
4	**Velocity (beam 1, y or north)**	**m/s**
5	**Velocity (beam 1, z or up)**	**m/s**
6	Signal strength (beam 1)	counts
7	Signal strength (beam 2)	counts
8	Signal strength (beam 3)	counts
9	Signal to Noise Ratio (beam 1)	dB
10	Signal to Noise Ratio (beam 2)	dB
11	Signal to Noise Ratio (beam 3)	dB
12	Correlation (beam 1)	%
13	Correlation (beam 2)	%
14	Correlation (beam 3)	%
15	Pressure	dbar
16	Analog input 1	
17	Analog input 2	
18	Checksum	1= failed, 0 = OK

4.3 Processing Flow

The specific steps necessary to create all calibrated and quality controlled data products for each OOI core instrument are described in the instrument-specific Processing Flow documents (DCN 1342-00780). These processing flow documents contain flow diagrams detailing all of the specific procedures (data product and QC) necessary to compute all levels of data products from the instrument and the order in which these procedures are performed.

The processing flow for the VEL3D computation is as follows (in Matlab syntax):

Step 1: Download binary file from Nortek instrument.

Step 2: Convert into ASCII text using provided sample code. The three Velocity variables shown in bold in the Velocity data file description above should be saved as the L0 VELPTTU data product.

Step 3: The magnetic variation correction should be applied to the velocity components using Equation 1. In Equation 1, the magnetic variation estimate for each deployment ($magvar$) can be calculated from the NOAA national geophysical data center website (http://www.ngdc.noaa.gov/geomagmodels/Declination.jsp)
Positive magnetic variation means that magnetic north is east of true north and negative one means that magnetic north is west of true north. Inputs are horizontal velocity profiles in earth coordinates (East and North, or "xx,yy").

$$\begin{bmatrix} X \\ \mathrm{Y} \end{bmatrix} = \begin{bmatrix} \cos(magvar) & \sin(magvar) \\ -\sin(magvar) & \cos(magvar) \end{bmatrix} * \begin{bmatrix} xx \\ \mathrm{yy} \end{bmatrix} \qquad \text{(Equation 1)}$$

Matlab example code is provided in Appendix D.

Step 4: Report the rotated, magnetic variation corrected velocity data components as the L1 VELPTTU data product along with the associated metadata

Step 5: Perform the necessary QC steps as outlined in the processing flow document

4.4 Outputs

The outputs of the Turbulent Point Water Velocity computation are

- L1 VELPTTU East, North, and Up components of velocity, corrected for magnetic variation, in m/s, as 4 character floating point numbers, %.3f

The metadata that must be included with the output are

- All other variables included in the file or associated header files, including:
  - Time stamp in MM:DD:YYYY HH:MM:SS.SS
  - Temperature in °C, as a 4 character floating point number, %.2f
  - Pressure in dbar, as a 3 character floating point number, %.2f
  - Pitch and roll in °, as a 2 character floating point number, %.1f
  - Heading in °, as a 4 character floating point number, %.1f
  - Signal strength in counts, as a 3 character floating point number, %.0f
  - Signal to Noise Ratio (SNR) in dB, as a 4 character floating point number, %.2f
  - Correlation in %, as a 3 character floating point number, %.0f
  - Sound speed in m/s, as a 5 character floating point number, %.1f

See Appendix B for a discussion of the accuracy of the output.

4.5 Computational and Numerical Considerations

4.5.1 Numerical Programming Considerations

There are no numerical programming considerations for this computation. No special numerical methods are used.

4.5.2 Computational Requirements

N/A

4.6 Code Verification and Test Data Set

The code will be verified using the test data set provided, which contains inputs and their associated correct outputs. CI will verify that the code is correct by checking that the output, generated using the test data inputs, is identical to the test data output. See Alfresco (*DPS Artifacts* >> *1341-00780_VELPTTU)* for Nortek test data sets, which are too large to include here. Magnetic correction was not done on this test data, but the code can be validated using the VELPTMN DPS (DCN 1341-00790).

Appendix A Nortek Vector Example Code

A.1 Read_vctr Code

(Code courtesy of Chris Wingard, Oregon State University)

```
function vctr = read_vctr(vctr_file,SampleRate,StrtTime)

% open the file for reading and mark the location(s) of the sync/id
% bytes in the file.
fid = fopen(vctr_file,'rb');
bytes = fread(fid,'uchar','l'); % read in the bytes to find the sync
characters
% Vector velocity data
vLoc = find(bytes(1:end-1) == hex2dec('a5') & bytes(2:end) ==
hex2dec('10'));
% Vector system data
sLoc = find(bytes(1:end-1) == hex2dec('a5') & bytes(2:end) ==
hex2dec('11'));
clear bytes

% test for end of file versus record length issues
nLast = ftell(fid);
% final number of bytes in file (end of file marker)
if nLast - sLoc(end) < 28;   % test if last record is correct length
    sLoc(end) = [];         % for the system data
end %if
sRec = length(sLoc); % number of sync bytes found equals estimated # of
records
if nLast - vLoc(end) < 24;   % test if last record is correct length
    vLoc(end) = [];         % for the velocity data
end %if
vRec = length(vLoc); % number of sync bytes found equals estimated # of
records
clear nLast

% Set up an array combining the locations of the sync/id pairs of the
% different records. The system data is sampled at 1 Hz, while the
% velocity data is sampled at a user specified rate (SampleRate). The
% number of bytes per velocity packet is 24, and the number of bytes
% per system packet is 28 bytes. Thus, if you do the math and have read
% the manual and Nortek's support website, there are 24 * SampleRate +
% 28 bytes between each system packet. So we can get at incomplete
% packets using the three different sets of numbers. This will be
% useful later on in determining the timing of each velocity data
% packet with respect to the system packets.
SyncLoc = zeros(sRec+vRec,3);
SyncLoc(1:sRec,1) = sLoc;
SyncLoc(sRec+1:end,1:2) = [vLoc ones(vRec,1)];
SyncLoc = sortrows(SyncLoc,1); % SyncLoc is an array indicating byte
count
% (or location) for sync/id pairs and a flag for packet source (0 =
% system, 1 = velocity)
```

```
% determine number of bytes between packets
nBytes = diff(SyncLoc);% number of bytes between sync/id pairs
m = (nBytes(:,1) ~= 24 & nBytes(:,2) <= 0) | ...
     (nBytes(:,1) ~= 28 & nBytes(:,2) == 1); % find & flag incomplete
records
SyncLoc(m==1,:) = []; clear nBytes m

% reset the sync/id locations and number of records to good packets
only
sLoc = find(SyncLoc(:,2) == 0); sRec = length(sLoc);
vLoc = find(SyncLoc(:,2) == 1); vRec = length(vLoc);

switch nargin
     case 3
      % determine the offset between the vector clock and GPS time.
      % This won't be 100% accurate, but it will work well enough for
      % our purposes (99%).
      tm2first = (sLoc(1) - vLoc(1)) / SampleRate;
      StrtTime = StrtTime + (tm2first / 60 / 60 / 24);
      clear tm2first
     case 2
      StrtTime = 0;
     otherwise
      error('you must give at least 2 inputs; in_file and sample
rate');
end

% preallocate arrays based on estimated number of records
% Velocity data
vSync = num2str(ones(vRec,1)*10);  % sync = 0xA5
vId = num2str(ones(vRec,1)*10);     % identification (0x10)
vAna2LSB = ones(vRec,1);            % analog input 2 LSB
vCount = ones(vRec,1);              % ensemble counter
vPrMSB = ones(vRec,1);              % pressure MSB
vAna2MSB = ones(vRec,1);            % analog input 2 MSB
vPrLSW = ones(vRec,1);              % pressure LSW
vSpare = ones(vRec,1);              % analog input 1 (fast)
vVel = ones(vRec,3);                % velocity (0.001 m/s)
vAmp = ones(vRec,3);                % amplitude (counts)
vCorr = ones(vRec,3);               % correlation (0-100%)
vChecksum = zeros(vRec,4);          % checksum

% System data
sSync = num2str(ones(sRec,1)*10);  % sync = 0xA5
sId = num2str(ones(sRec,1)*10);     % identification (0x11)
sSize = ones(sRec,1);               % size of structure (1 word = 2
bytes)
sClock = ones(sRec,6);              % date and time
sBattery = ones(sRec,1);           % battery voltage (0.1 V)
sSoundSpeed = ones(sRec,1);        % speed of sound (0.1 m/s)
sHeading = ones(sRec,1);           % compass heading (0.1 deg)
sPitch = ones(sRec,1);             % compass pitch (0.1 deg)
sRoll = ones(sRec,1);                    % compass roll (0.1 deg)
sTemperature = ones(sRec,1);       % temperature (0.01 degC)
sError = ones(sRec,1);             % error code
```

```
sStatus = ones(sRec,1);             % status
sSpare = ones(sRec,1);              % analog input 1 (slow)
sChecksum = zeros(sRec,4);          % checksum

% read in the velocity data
for i = 1:vRec
     fseek(fid,SyncLoc(vLoc(i),1)-1,'bof');
     vSync(i,:) = dec2hex(fread(fid,1,'uchar','l'));
     vId(i,:) = dec2hex(fread(fid,1,'uchar','l'));
     vAna2LSB(i) = fread(fid,1,'uchar','l');
     vCount(i) = fread(fid,1,'uchar','l');
     vPrMSB(i) = fread(fid,1,'uchar','l');
     vAna2MSB(i) = fread(fid,1,'uchar','l');
     vPrLSW(i) = fread(fid,1,'uint16','l');
     vSpare(i) = fread(fid,1,'uint16','l');
     vVel(i,:) = fread(fid,3,'int16','l') * 0.001;
     vAmp(i,:) = fread(fid,3,'uchar','l');
     vCorr(i,:) = fread(fid,3,'uchar','l');
     vChecksum(i,1) = fread(fid,1,'uint16','l');
     % test the checksum
     fseek(fid,SyncLoc(vLoc(i),1)-1,'bof'); vSize = 24 / 2;
     cbuf = fread(fid,vSize*2,'uchar','l');
     ck = cChecksum(cbuf,vSize-1);
     vSum = cbuf(2*vSize-1) + cbuf(2*vSize) * 256;
     if vSum > 65536, vSum = vSum - 65536; end %if
     vChecksum(i,2:3) = [ck vSum];
     if vChecksum(i,1) == ck && ck == vSum
      % checksum match, yeah!
      vChecksum(i,4) = 1;
      SyncLoc(vLoc(i),3) = 1;
     end %if
end %for
clear vLoc vRec i vSync vId vAna2LSB vAna2MSB vSpare vSize vSize cbuf
ck vSum

% bring all the velocity data together
mGood = find(vChecksum(:,4) == 1);

% pressure and cell depth calculations (from Nortek)
db = (65536 * vPrMSB(mGood) + vPrLSW(mGood)) * 0.001;
clear vPrMSB vPrLSW

% combine the velocity data
Velocity = [db vCount(mGood,:) vVel(mGood,:) vAmp(mGood,:)
vCorr(mGood,:)];
clear mGood nRec db vCount vVel vAmp vCorr vChecksum

% read in the system data
for i = 1:sRec
     fseek(fid,SyncLoc(sLoc(i),1)-1,'bof');
     sSync(i,:) = dec2hex(fread(fid,1,'uchar','l'));
     sId(i,:) = dec2hex(fread(fid,1,'uchar','l'));
     sSize(i) = fread(fid,1,'uint16','l');
     sClock(i,:) = str2num(dec2hex(fread(fid,6,'uchar','l')))';
     sBattery(i) = fread(fid,1,'uint16','l') * 0.1;
```

```
    sSoundSpeed(i) = fread(fid,1,'uint16','l') * 0.1;
    sHeading(i) = fread(fid,1,'int16','l') * 0.1;
    sPitch(i) = fread(fid,1,'int16','l') * 0.1;
    sRoll(i) = fread(fid,1,'int16','l') * 0.1;
    sTemperature(i) = fread(fid,1,'int16','l') * 0.01;
    sError(i) = fread(fid,1,'char','l');
    sStatus(i) = fread(fid,1,'char','l');
    sSpare(i) = fread(fid,1,'uint16','l');
    sChecksum(i,1) = fread(fid,1,'uint16','l');
    % test the checksum
    fseek(fid,SyncLoc(sLoc(i),1)-1,'bof');
    cbuf = fread(fid,sSize(i)*2,'uchar','l');
    ck = cChecksum(cbuf,sSize(i)-1);
    sSum = cbuf(2*sSize(i)-1) + cbuf(2*sSize(i)) * 256;
    if sSum > 65536, sSum = sSum - 65536; end %if
    sChecksum(i,2:3) = [ck sSum];
    if sChecksum(i,1) == ck && ck == sSum
     % checksum match, yeah!
     sChecksum(i,4) = 1;
     SyncLoc(sLoc(i),3) = 1;
    end %if
end %for
clear sLoc sRec i sSync sId sSize sSpare sSize cbuf ck sSum
fclose(fid); clear fid ans

% bring all the system data together
mGood = find(sChecksum(:,4) == 1);
sec = mGood(1) - 1;
clear sChecksum

% decimal day of year calculation
sdn = datenum(sClock(mGood,5)+2000,sClock(mGood,6),sClock(mGood,3),...
    sClock(mGood,4),sClock(mGood,1),sClock(mGood,2));
offset = sdn(1) - (StrtTime + (sec/60/60/24));
sdn = sdn - offset;
dVec = datevec(sdn(1));
yd = sdn - datenum(dVec(1)-1,12,31);
clear StrtTime sClock sdn offset

% Combine the system data
System = [yd sError(mGood) sStatus(mGood) sBattery(mGood)
sSoundSpeed(mGood) ...
    sTemperature(mGood) sHeading(mGood) sPitch(mGood) sRoll(mGood)];
clear yd sError sStatus sBattery sSoundSpeed sTemperature sHeading
sPitch sRoll
clear mGood

%%%% now glue it all together %%%%

% set the final sync/id array to good packets only
m = SyncLoc(:,3) == 0; SyncLoc(m==1,:) = []; clear m
sLoc = find(SyncLoc(:,2) == 0); sRec = length(sLoc);
vLoc = find(SyncLoc(:,2) == 1); vRec = length(vLoc);

% create a cumulative packet counter (based on the count variable in
```

```
the
% velocity data) and adjust it for any dropped packets
cnt = 0:vRec-1; dCnt = diff(Velocity(:,2));
m = find(dCnt ~= 1 & dCnt ~= -255);
for i = 1:length(m)
     if dCnt(m(i)) < 0
      dp = 255 - dCnt(m(i));  % number of dropped packets
     else
      dp = dCnt(m(i));
     end %if
     cnt(m(i)+1:end) = cnt(m(i)+1:end) + (dp - 1);
end %while
clear dCnt m i dp

% assign the 1 Hz time record to the appropriate velocity packets
tm = zeros(vRec,4) * NaN;
for i = 1:length(sLoc)
     m = find(SyncLoc(vLoc,1) == SyncLoc(sLoc(i)+1,1));
     if isempty(m) == 0
      nPac = (SyncLoc(vLoc(m),1) - SyncLoc(sLoc(i),1)) - 28;
      if nPac ~= 0, nPac = nPac / 24; end %if
      tm(m,1) = System(i,1) + ((nPac / SampleRate) ...
            / 60 / 60 / 24);
      tm(m,2) = i; % system packet number
     end %if
end %for
clear i m nPac

% extrapolate the 1 Hz time record to the SampleRate Hz velocity data
based on
% the cumulative counter created above.
m = find(isnan(tm(:,1)) == 0);
tic = (cnt(1:m(1)-1) - cnt(m(1))) / SampleRate;
tm(1:m(1)-1,1) = tm(m(1),1) + tic' / 60 / 60 / 24;
for i = 1:length(m)-1
     tic = (cnt(m(i)+1:m(i+1)-1) - cnt(m(i))) / 64;
     tm(m(i)+1:m(i+1)-1,1) = tm(m(i),1) + tic' / 60 / 60 / 24;
end %for
tic = (cnt(m(end)+1:end) - cnt(m(end))) / SampleRate;
tm(m(end)+1:end,1) = tm(m(end),1) + tic' / 60 / 60 / 24;
clear cnt m tic i

% extrapolate the rest of the 1 Hz data onto the SampleRate Hz velocity
% data.
% heading, pitch, roll, sound speed, battery, temperature
[Hx,Hy] = pol2cart(rad(System(:,7)),1);
extrp = interp1(System(:,1),[System(:,4:6) Hx Hy
System(:,8:9)],tm(:,1));
th = deg(cart2pol(extrp(:,4),extrp(:,5)));
m = find(th < 0); th(m) = th(m) + 360; clear m
clear Hx Hy

% error and status codes
for i = 1:sRec
     m = find(tm(:,2) == i);
```

```
    n = find(tm(:,2) - 1 == i);
    if isempty(n) == 1 || i == sRec
     n = vRec;
    end %if
    tm(m:n,3:4) = repmat(System(i,2:3),1 + n - m,1);
end %for
clear i m n

% and finally .... really ...
% smooth the pressure record
db = running(Velocity(:,1),'median',9,1);
db = running(db,'median',9,1);
db = running(db,'mean',13,1);

vctr = [tm(15:end-14,1) db tm(15:end-14,3:4) Velocity(15:end-14,2:11)
...
    th(15:end-14) extrp(15:end-14,[6:7 1:3])];
m = any(isnan(vctr)') == 1; vctr(m==1,:) = [];
clear tm extrp th db m

% clean up
clear SyncLoc sLoc sRec vLoc vRec System Velocity

% print the data to a file
dStr = datestr(vctr(:,1) + datenum(dVec(1)-
1,12,31),'yyyymmddHHMMSS.FFF');
fn = strrep(vctr_file,'/vec/','/vac/');
fn = strrep(fn,'.vec','.vac');

fid = fopen(fn,'wt');
fprintf(fid,'Yday(gmt)\tyyyymmddHHMMSS.SSS\tDepth(db)\tError\tStatus\tC
ount\t');
fprintf(fid,'East(m/s)\tNorth(m/s)\tUp(m/s)\t');
fprintf(fid,'Amp1(cnts)\tAmp2(cnts)\tAmp3(cnts)\t');
fprintf(fid,'Cor1(%%)\tCor2(%%)\tCor3(%%)\t');
fprintf(fid,'Heading(deg)\tPitch(deg)\tRoll(deg)\t');
fprintf(fid,'Battery(V)\tSoundSpeed(m/s)\tTemp(degC)\n');
for i = 1:size(vctr,1)
    fprintf(fid,'%11.7f\t%18s\t%6.3f\t%3d\t%4d\t%4d\t',vctr(i,1),dStr(i
,:),vctr(i,2:5));
    fprintf(fid,'%7.4f\t%7.4f\t%7.4f\t',vctr(i,6:8));
    fprintf(fid,'%4.0f\t%4.0f\t%4.0f\t',vctr(i,9:11));
    fprintf(fid,'%4.0f\t%4.0f\t%4.0f\t',vctr(i,12:14));
    fprintf(fid,'%5.1f\t%5.1f\t%5.1f\t',vctr(i,15:17));
    fprintf(fid,'%4.1f\t%6.1f\t%4.1f\n',vctr(i,18:20));
end %for
fclose(fid);
clear fid dStr fn i
```

A.2 pol2cart Code

pol2cart = A MATLAB function that converts polar coordinates to Cartesian: http://www.mathworks.com/help/techdoc/ref/pol2cart.html

A.3 cart2pol Code

cart2pol = A MATLAB function that converts Cartesian coordinates to polar: http://www.mathworks.com/help/techdoc/ref/cart2pol.html

Appendix B Output Accuracy

Nortek Vector: Current speed accuracy = ±0.001 cm s^{-1}
Direction accuracy = ±2 degrees

Nobska MAVS-4: Current speed accuracy = ±0.3 cm s^{-1}
Direction accuracy = ±2 degrees

VELPTTU Accuracy Requirements are stated as:

L4-CG-IP-RQ-260 Three axis point water velocity instruments shall measure three axes of current speed with an accuracy of ±1 cm s^{-1}.

L4-CG-IP-RQ-266 Three axis point water velocity instruments shall have a direction accuracy of ±2 degrees.

Appendix C Sensor Calibration Effects

Calibration for VEL3D instruments is extremely important, especially compass and pitch/roll calibration. Details can be found in the Nortek Vector and Nobska MAVS-4 User Manuals, archived on Alfresco (See Alfresco (*DPS Artifacts* >> *1341-00780_VELPTTU*).

Appendix D Magnetic Variance Example Code

Matlab example code for transformation to correct for magnetic variance:
(Code courtesy of Janet Fredericks, Woods Hole Oceanographic Institution)

```
% Apply magnetic variation 'Degrees' (see Section 4.3 Step 3)
% Note: use of atan2 changes heading range from 0-360 to +/-180.
% east/north are velocities relative to magnetic north
magvar = Degrees * pi/180;  % convert to radians
i = sqrt(-1);
unit_vec = sin(hd*pi/180) + i* cos(hd*pi/180);
x = real(unit_vec)*cos(magvar) + imag(unit_vec)*sin(magvar);
y = imag(unit_vec)*cos(magvar) - real(unit_vec)*sin(magvar);
hd = atan2( x, y ) * 180/pi;
x =  east*cos(magvar) + north*sin(magvar);
y = north*cos(magvar) -  east*sin(magvar);
east_rotated = x;  north_rotated = y;
magvar = magvar * 180/pi;  % save value in degrees
```