



# DATA PRODUCT SPECIFICATION FOR FDCHP DATA PRODUCTS

Version 1-00  
Document Control Number 1341-00280  
2015-02-26

Consortium for Ocean Leadership  
1201 New York Ave NW, 4<sup>th</sup> Floor, Washington DC 20005  
[www.OceanLeadership.org](http://www.OceanLeadership.org)

in Cooperation with

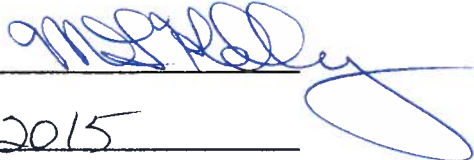
University of California, San Diego  
University of Washington  
Woods Hole Oceanographic Institution  
Oregon State University  
Scripps Institution of Oceanography  
Rutgers University

### Document Control Sheet

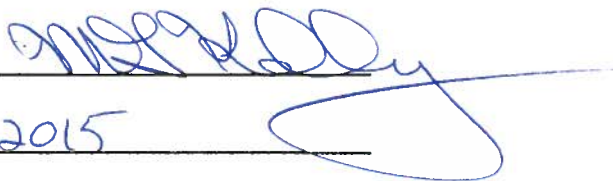
<b>Version</b>	<b>Date</b>	<b>Description</b>	<b>Author</b>
0-01	2014-05-08	Initial draft	J. Fredericks, J. Edson
0-02	2014-06-09	First Revision	J. Edson
0-03	2015-01-19	Second Revision	J. Edson
0-04	2015-01-28	Third Revision	J. Edson
1-00	2015-02-26	Initial Release per ECR 1300-00479	S. White

### Signature Page

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

Senior PM  
OOI ~~Senior Systems Engineer~~:   
Date: 3-2-2015

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: 3-2-2015

## Table of Contents

1	Abstract.....	1
2	Introduction .....	1
2.1	Author Contact Information .....	1
2.2	Metadata Information .....	1
2.3	Instruments .....	5
2.4	Literature and Reference Documents .....	5
2.5	Terminology .....	6
3	Theory.....	6
3.1	Description .....	6
3.2	Mathematical Theory.....	7
3.3	Known Theoretical Limitations .....	11
3.4	Revision History .....	11
4	Implementation .....	11
4.1	Overview .....	11
4.2	Inputs .....	12
4.3	Processing Flow .....	13
4.4	Outputs.....	14
4.5	Computational and Numerical Considerations.....	14
4.6	Code Verification and Test Data Set.....	14
Appendix A	Example C and Matlab processing code (UConn/WHOI).....	15
Appendix B	Output Accuracy .....	25
Appendix C	Sensor Calibration Effects .....	25

## 1 Abstract

The Flux Direct Covariance High Power (FDCHP) system is an instrument package that collects vertical and horizontal wind components, air temperature and platform motion. This data is used to directly compute air-sea fluxes of momentum and buoyancy. The air-sea flux of momentum is the vertical transfer of horizontal momentum from the air to the ocean and is often referred to as the wind stress. The air-sea buoyancy flux is the vertical transfer of buoyancy associated with moist air and represents a mixture of sensible and latent heat exchange. Computation of the fluxes is accomplished using the motion-corrected direct covariance (MCDC) approach. This approach requires the system to measure the motion of the platform that can be used to compute the significant wave height and its direction. The system also provides the associated means and additional statistical measures of atmospheric turbulences and platform motion.

This document describes the computation used to calculate the OOI Level 1 and Level 2 FDCHP products from data collected by the system. The L1 products provide the motion corrected time series of 3-axis winds and sonic temperature denoted by WINDTUR\_L1 and TMPATUR\_L1 in this document. The L2 products provide the air-sea fluxes of momentum and buoyancy computed from the L1 products. The along-wind and cross-wind components of the momentum flux vector are denoted by FLUXMOM-U\_L2 and FLUXMOM-V\_L2, respectively; while the buoyancy flux is denoted by FLUXHOT\_L2. This document is to be used by OOI programmers to construct appropriate processes to read the L1 and L2 products.

## 2 Introduction

### 2.1 Author Contact Information

Please contact Jim Edson ([james.edson@uconn.edu](mailto:james.edson@uconn.edu)) or the Data Product Specification lead ([DPS@lists.oceanobservatories.org](mailto: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 Names

The OOI L0 Core Data Product Names and Descriptive Names for the products are:

<b>Name</b>	<b>Descriptive Name</b>
MOTFLUX_L0	Platform Motion in buoy reference frame [decimal counts]
TMPATUR_L0	Speed of sound [counts]
WINDTUR_L0	Wind speed components in buoy reference frame [counts]

The coefficients to convert from counts to physical units are listed in Table 1.

The OOI L1 Core Data Product Names and Descriptive Names for the products are:

<b>Name</b>	<b>Descriptive Name</b>
WINDTUR-VLN_L1	Motion-corrected northward wind speed component [m/s]
WINDTUR-VLW_L1	Motion-corrected westward wind speed component [m/s]
WINDTUR-VLU_L1	Motion-corrected upward wind speed component [m/s]
TMPATUR_L1	Sonic temperature [°C]

The OOI L2 Core Data Product Names and Descriptive Names for the products are:

<b>Name</b>	<b>Descriptive Name</b>
-------------	-------------------------

<b>FLUXMOM-U_L2</b>	Along-wind component of momentum flux [m <sup>2</sup> /s <sup>2</sup> ]
<b>FLUXMOM-V_L2</b>	Cross-wind component of momentum flux [m <sup>2</sup> /s <sup>2</sup> ]
<b>FLUXHOT_L2</b>	Buoyancy Flux [m/s K]

## Data Product Abstract (for Metadata)

This document describes how to compute the L1 and L2 data products from the raw data upon instrument recovery.

A number of the OOI surface buoys are equipped with the Flux Direct Covariance High Power (FDCHP). The FDCHP collects and stores the raw motion (**MOTFLUX\_L0**), sonic anemometer (**WINDTUR\_L0**), and sonic temperature (**TMPATUR\_L0**) data. The document describes how the raw data is processed to compute the Level 1 motion-corrected wind speeds (**WINDTUR-L1**) and sonic temperature (**TMPATUR\_L1**) data. The Level 1 data is then used to compute the Level 2 momentum flux components (**FLUXMOM-U\_L2** and **FLUXMOM-V\_L2**) required to produce the surface stress vector, and the buoyancy flux (**FLUXHOT\_L2**).

The momentum flux is the vertical transfer of horizontal momentum from the air to the ocean and is called the wind stress. It is the transfer of energy from the wind physically pushing against the water. The wind stress is a vector quantity that can be defined as

$$\frac{\vec{\tau}}{\rho_a} = \hat{i} \overline{u'w'} + \hat{j} \overline{v'w'}$$

where  $\vec{\tau}$  is the stress vector,  $\rho_a$  is the density of air, and  $\overline{u'w'}$  and  $\overline{v'w'}$  are the along-wind and cross-wind components, respectively. The vector wind in this coordinate system is given by  $\vec{U} = \hat{i}\bar{U}$ , where  $\bar{U}$  is the mean wind speed.

**FLUXMOM-U:** The kinematic form (Stull, 1988) of the **along-wind component of momentum flux**,  $\overline{u'w'}$ , is computed using the motion-corrected direct covariance (MCDC) method where  $u'$  represents fluctuations in the along-wind (or streamwise) wind component and  $w'$  represents fluctuations the vertical velocity component. These fluctuations are computed after removal of the platform motion from the measured wind vector as described in section 3. The along-wind component generally carries most of the momentum flux, i.e., it is responsible for most of the surface stress.

**FLUXMOM-V:** The kinematic form of the **cross-wind component of momentum flux**,  $\overline{v'w'}$ , is computed using the MCDC method where  $v'$  represents fluctuations in the cross-wind (or lateral) wind component. The cross-wind component is generally smaller than the along-wind component, signifying that the wind and stress vectors are closely aligned. However, this component can become as large as or even larger than the along-wind component near the ocean surface in the presence of waves. It can also be large in light-wind conditions where the wind and stress vectors are poorly defined.

**FLUXHOT:** The kinematic form of the **buoyancy Flux**,  $\overline{w'T_v'}$ , is computed using the direct covariance method where  $T_v'$  represents fluctuations in the virtual temperature  $T_v = T(1 + 0.61q)$ , where  $T$  is air temperature and  $q$  is the specific humidity. The virtual temperature is defined as and incorporates the effect of both temperature and moisture on the buoyancy of an air parcel (i.e., its density compared to the density of the surrounding air). For example, a moist parcel of air is less dense than a dry parcel of air at the same temperature. Such a parcel would have positive buoyancy and would want to rise thereby transferring moisture (and latent heat) upwards. The DCFS approximate the virtual temperature using the sonic temperature given by  $T_v \approx T_s = T(1 + 0.51q)$ . The small difference between the buoyancy flux computed using sonic

temperature can be removed by users with estimates of the moisture (or latent heat) flux from the bulk fluxes.

Ancillary Data: The Level 0 and Level 1 data are also used to provide ancillary data that includes the mean, standard deviation, minimum value, and maximum value of the variables from each sensor. These are intended to provide diagnostic data to monitor system performance.

### 2.2.2 Computation Name

The Motion-Corrected Direct Covariance Method (MDCD)

### 2.2.3 Computation Abstract (for Metadata)

This DPS describes the motion-corrected direct covariance (MDCD) approach used by the FDCHP to produce the L2 data products. The FDCHP system provides a means to directly compute the fluxes; it makes rapid (e.g. 10 Hz) observations of turbulent three-dimensional wind velocity ( $\bar{U} + u', v', w'$ ) and sonic temperature ( $\bar{T}_s + T_s'$ ) from a 3-axis sonic anemometer, where sonic temperature is derived from the measured speed of sound. The velocity data is contaminated by platform motion, which is removed prior to calculation of the direct covariance fluxes. This is accomplished using a "strapped-down" system with 3-axis linear accelerometers, 3-axis angular rates sensors (gyros), and a pitch, roll and yaw magnetometer. These sensors are used to determine the rotation matrix and the 3-axis platform velocities as described in section 3.2. The fluxes are computed from the products of motion corrected velocities  $\overline{u'w'}$ ,  $\overline{v'w'}$  and  $\overline{w'T_s'}$  to provide estimates, respectively, of the two horizontal components of wind stress and the buoyancy flux.

### 2.2.4 Instrument-Specific Metadata

The height of each sensor above the nominal sea surface and the recording period (length of time over which an observation is taken) must be recorded and kept as part of the metadata. The latitude of the FDCHP system is required and kept as part of the metadata.

The sign convention for the instruments in the buoy reference frame is based on a right-handed (x, y, z) coordinate system with x positive towards the buoy vane, y positive to port of the buoy vane (i.e., to the left looking in the positive-x direction), z positive upward, roll positive for positive-y rolled up, pitch positive for positive-x pitched down, and yaw (heading) positive for positive-z yawed counter-clockwise. Note that the right-handed definition of yaw is opposite the typical left-handed definition used for a compass. Also note that the x-component of the relative wind speed is generally positive in this coordinate system as it usually blows towards the vane.

The FDCHP will provide a time stamp using its internal clock. The metadata must include the time the internal clock was set such that drift can be easily computed in post-processing. The FDCHP will also provide a version number of the software and a status integer.

#### Quality Control Variables/User Auxiliary Data

During the computations associated with the MDCD, auxiliary data products are computed, saved and telemetered to provide quality control of the FDCHP data (for more information, see the FDCHP Interface Document). Specifically these are:

#### **Sonic Anemometer/Thermometer (Gill Windmaster Pro Model 1561-PK-020)**

WindU: Average wind speed component along instrument x-axis [m/s]

WindV: Average wind speed component along instrument y-axis [m/s]

WindW: Average wind speed component along instrument z-axis [m/s]

Tsonic: Average sonic temperature [C]

StdU: Standard deviation of WindU [m/s]

StdV: Standard deviation of WindV [m/s]

StdW: Standard deviation of WindW [m/s]

StdTs: Standard deviation of Tsonic [C]  
MaxU: Maximum of WindU [m/s]  
MaxV: Maximum of WindV [m/s]  
MaxW: Maximum of WindW [m/s]  
MaxTs: Maximum of Tsonic [C]  
MinU: Minimum of WindU [m/s]  
MinV: Minimum of WindV [m/s]  
MinW: Minimum WindW [m/s]  
MinTs: Minimum of Tsonic [C]

**Linear Accelerometers (Microstrain Model 3DM-GX3-25)**

AX: Average observed acceleration along the instrument x axis [m/s<sup>2</sup>]  
AY: Average observed acceleration along the instrument y axis [m/s<sup>2</sup>]  
AZ: Average observed acceleration along the instrument z axis [m/s<sup>2</sup>]  
AXstd: Standard deviation of AX [m/s<sup>2</sup>]  
AYstd: Standard deviation of AY [m/s<sup>2</sup>]  
AZstd: Standard deviation of AZ [m/s<sup>2</sup>]  
AXmax: Maximum AX [m/s<sup>2</sup>]  
AYmax: Maximum AY [m/s<sup>2</sup>]  
AZmax: Maximum AZ [m/s<sup>2</sup>]  
AXmin: Minimum AX [m/s<sup>2</sup>]  
AYmin: Minimum AY [m/s<sup>2</sup>]  
AZmin: Minimum AZ [m/s<sup>2</sup>]

**Angular Rate Sensors (Microstrain Model 3DM-GX3-25)**

RX: Average observed angular rate about the instrument x-axis [radian/sec]  
RY: Average observed angular rate about the instrument y-axis [radian/sec]  
RZ: Average observed angular rate about the instrument z-axis [radian/sec]  
RXstd: Standard deviation of RateX [radian/sec]  
RYstd: Standard deviation of RateY [radian/sec]  
RZstd: Standard deviation of RateZ [radian/sec]  
RXmax: Maximum RateX [radian/sec]  
RYmax: Maximum RateY [radian/sec]  
RZmax : Maximum RateZ [radian/sec]  
RXmin : Minimum RateX [radian/sec]  
RYmin: Minimum RateY [radian/sec]  
RZmin: Minimum RateZ [radian/sec]

**Magnetometer (Microstrain Model 3DM-GX3-25)**

Heading: Average Heading [radians]  
Pitch: Average Pitch [radians]  
Roll: Average Roll [radians]  
stdH: Standard deviation of Heading [radians]  
stdP: Standard deviation of Pitch [radians]  
stdR: Standard deviation of Roll [radians]  
maxH: Maximum Heading [radians]  
maxP: Maximum Pitch [radians]  
maxR: Maximum Roll [radians]  
minH: Minimum Heading [radians]  
minP: Minimum Pitch [radians]  
minR: Minimum Roll [radians]

**Additional Motion-Corrected Data**

Ucorr: Motion-corrected Northerly wind speed component [m/s]  
Vcorr: Motion-corrected Westerly wind speed component [m/s]  
Wcorr: Motion-corrected vertical wind speed component [m/s]



StdUcorr: Standard deviation of along-wind component [m/s]  
StdVcorr: Standard deviation of cross-wind component [m/s]  
StdWcorr: Standard deviation of vertical component [m/s]  
WindSpeed: Motion-corrected wind speed relative to ground [m/s]  
UWcorr: Along-wind momentum flux (corresponds to **FLUXMOM-U\_L2**)  
VWcorr: Cross-wind momentum flux (corresponds to **FLUXMOM-V\_L2**)  
  
WTcorr: Sonic temperature flux (corresponds to **FLUXHOT\_L2**)  
SigH: Significant wave height  
SigCp: Significant wave period

### 2.2.5 Data Product Synonyms

- The momentum flux is also known as wind stress.
- The direct covariance method is also known as the eddy correlation method.
- The Flux Direct Covariance High Power (FDCHP) system was originally known as the Direct Covariance Flux System (DCFS) as described by Edson et al. (1998).

### 2.2.6 Similar Data Products

The L2 **BULKFLUX** core data products provide estimates of the latent heat, sensible heat, and momentum fluxes using the bulk formulae method. The latent and sensible heat fluxes can be combined to provide estimates of the buoyancy flux.

## 2.3 Instruments

For detailed information on the instruments from which the L2 FDCHP data product inputs are obtained, see the FCDHP Processing Flow document (1342-00280) and FDCHP Interface Document. Briefly, the system measures the horizontal and vertical wind components in the buoy reference frame using a 3-axis Gill Windmaster Pro (Model 1561-PK-020) sonic anemometer. The sonic anemometer also provides the speed of sound that is readily converted into sonic temperature, which closely approximates the virtual air temperature. The platform motion is characterized using a Microstrain (Model 3DM-GS5-25) Inertial Measurement Unit (IMU). This device integrates 3-axis linear accelerometers, 3-axis angular rate sensors (solid state gyros), and a 3-axis magnetometer. The data is data from the sonic anemometer and IMU are merged, time-stamped and stored by a AA3355 1 Ghz ARM Cortex-A8 processor. The processor collects data for 20 minutes out of the hour and processes it during the 40 minute interval to generate the ancillary data telemetered to monitor system performance.

## 2.4 Literature and Reference Documents

- Ancil, F., M. A. Donelan, W. M. Drennan, and H. C. Graber, 1994: Eddy-correlation measurements of air-sea fluxes from a discus buoy. *J. Atmos. Oceanic Technol.*, 11, 1144–1150.
- Axford, D. N., 1968: On the accuracy of wind measurements using an inertial platform in an aircraft and an example of a measurement of the vertical mesostructure of the atmosphere. *J. Appl. Meteor.*, 7, 645–666.
- Dugan, J. P., S. L. Panichas, and R. L. DiMarco, 1991: Decontamination of wind measurements from buoys subject to motions in a seaway. *J. Atmos. Oceanic Technol.*, 8, 85–95.
- Edson, J.B., A. A. Hinton, K. E. Prada, J.E. Hare, and C.W. Fairall, 1998: Direct covariance flux estimates from mobile platforms at sea, *J. Atmos. Oceanic Tech.*, 15, 547-562
- Edson, J. B., and C.W. Fairall, 1998: Similarity relationships in the marine atmospheric surface layer for terms in the TKE and scalar variance budgets. *J. Atmos. Sci.*, 55, 2311-2328.

- Fairall, C.W., A. B. White, J. B. Edson, and J. E. Hare, 1997: Integrated shipboard measurements of the marine boundary layer. *J. Atmos. Oceanic Technol.*, 14, 368–379.
- Fujitani, T., 1981: Direct measurement of turbulent fluxes over the sea during AMTEX. *Pap. Meteor. Geophys.*, 32, 119 –134.
- Fujitani, T., 1985: Method of turbulent flux measurement on a ship by using a stable platform system. *Pap. Meteor. Geophys.*, 36, 157–170.
- Goldstein, H., 1965: *Classical Mechanics*. Addison-Wesley, 398 pp.
- Hristov, T. S., S. D. Miller, and C. A. Friehe, 2003: Dynamical coupling of wind and ocean waves through wave-induced air flow, *Nature*, **422**, 55-58.
- Miller, S., C. Friehe, T. Hristov, and J. Edson, 2008: Platform motion effects on measurements of turbulence and air-sea exchange over the open ocean, *J. Atmos. Oceanic Tech.*, 25, 1683-1694.
- Oost, W. A., C. W. Fairall, J. B. Edson, S. D. Smith, R. J. Anderson, J. A. B. Wills, K. B. Katsaros, and J. DeCosmo, 1994: Flow distortion calculations and their application in HEXMAX. *J. Atmos. Oceanic Technol.*, 11, 366–386.
- Stull, R. B., 1988: *An Introduction to Boundary Layer Meteorology*. Kluwer Academic Publishers, 666 pp.
- Ware, J., 2014, Interface Document for the Flux Direct Covariance High Power System. (see DPS Artifacts >> DCHPFLX >> FDCHP\_Interface\_Document-{revNN}.pdf)

## 2.5 Terminology

### 2.5.1 Definitions

None.

### 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).

### 2.5.3 Variables and Symbols

Temperatures in degrees Kelvin are denoted by K and in Celsius by C, where  $T(K) = T(C) + 273.15$  K.

## 3 Theory

### 3.1 Description

The time-averaged flux determined using the direct covariance (or eddy correlation) technique is regarded as the most direct estimate of the ensemble average flux. In the field, a sonic anemometer is commonly used to provide the three velocity measurements required to compute the vector stress

$$\frac{\vec{\tau}}{\rho_a} = \hat{i} \overline{u'w'} + \hat{j} \overline{v'w'} \quad (1)$$

where  $\rho_a$  is the density of air; the overbar denote a time average; and  $u'$ ,  $v'$  and  $w'$  are the longitudinal, lateral, and vertical velocity fluctuations about their means, respectively. In (1),  $-\overline{u'w'}$  represents the longitudinal (along-wind) component of the stress, and  $-\overline{v'w'}$  is the lateral component.

The vertical velocity can also be correlated with scalar quantities to compute their vertical flux. For example, investigations of air-sea investigation often need estimates of the buoyancy flux to account for atmospheric stability,

$$Q_B = \overline{w'T_v'} \approx \overline{w'T_s'} \quad (2)$$

where  $T_v'$  denotes fluctuations in the virtual temperature. This temperature is closely approximately by the sonic temperature  $T_s$  provided by the speed of sound measurements from a sonic anemometer. The buoyancy flux is used to quantify the buoyant production or consumption of turbulent kinetic energy (TKE), and is used to define the convective velocity scale (Stull 1988). This flux can also be combined with estimates of the surface stress to compute atmospheric stability parameters used in Monin-Obukhov similarity theory (e.g., Edson and Fairall, 1998).

The obvious problem that arises when estimating these fluxes from a moving platform is that part of the fluctuating velocity is due to platform motion. This motion contamination must therefore be removed before we can compute the fluxes. The contamination arises from three sources: 1) instantaneous tilt of the anemometer due to the pitch, roll, and heading variations of the platform; 2) angular velocities at the anemometer due to rotation of the platform about its local coordinate system axes; and 3) translational velocities of the platform with respect to a fixed frame of reference (Dugan et al. 1991; Ancil et al. 1994; Edson et al. 1998; Miller et al. 2008).

### 3.2 Mathematical Theory

A variety of approaches have been used to correct wind sensors for platform motion. True inertial navigation systems (Axford 1968) are standard for research aircraft. These systems are expensive and subject to the so-called Schuler oscillation, so simpler techniques have been sought for ships where the platform mean vertical velocity is unambiguously zero. The basic approach that we are using follows that of Fujitani (1981), where the true wind vector (i.e., uncontaminated by motion) can be written as

$$\vec{V}_{true} = T\vec{V}_{obs} + \vec{\Omega} \times T\vec{M} + \vec{V}_{CM} \quad (3)$$

where  $\vec{V}_{true}$  is the desired wind velocity vector in the reference coordinate system,  $\vec{V}_{obs}$  is the measured wind velocity vector in the platform frame of reference,  $T$  is the coordinate transformation matrix for a rotation of the platform frame coordinate system to the reference coordinates,  $\vec{\Omega}$  is the angular velocity vector of the platform coordinate system,  $\vec{M}$  is the position vector of the wind sensor with respect to the center of gravity, and  $\vec{V}_{CM}$  is the translational velocity vector at the center of motion/mass of the platform with respect to a fixed coordinate system.

The motion measurement system is often separated from the center of motion of the platform. As a result, an additional correction term is required to account for the angular velocities that are sensed at that location as translational velocities by the accelerometers (Fujitani 1985). This term is incorporated in (3) as

$$\vec{V}_{true} = T\vec{V}_{obs} + \vec{\Omega} \times T(\vec{M} - \vec{S}) + \vec{V}_{mot} \quad (4)$$

where  $\vec{S}$  is the vector distance from the motion system to the center of motion of the platform and  $\vec{V}_{mot}$  now includes the additional translational velocities. Fortunately,  $\vec{M} - \vec{S}$  is just the position vector of the wind sensor with respect to the motion package. Therefore, one does not need to know the exact location of the center of motion, which is often difficult to identify, just the distance between the motion sensors and the sampling volume of the sonic anemometer.

### 3.2.1 Angles and angular rates

To use (4), we need three angular variables describing the platform's orientation in the fixed frame and the angular velocity vector describing the time rate of change of its orientation. Several different angular coordinate systems are available (Goldstein 1965), but roll  $\phi$ , pitch  $\theta$ , and yaw  $\psi$  are most often used because they are the variables output from doubly gimballed gyro-stabilized systems commonly used on research vessels (e.g., the gyro-compass often located on the bridge). Such gyro-stabilized systems provide the user with pitch, roll, and yaw angles that describe the ship's orientation in the fixed frame. These angles can be used directly in the total rotational coordinate transformation matrix that we define as

$$\begin{aligned}
 T(\phi, \theta, \psi) &= A(\psi)A(\theta)A(\phi) \\
 &= \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \\
 &= \begin{bmatrix} \cos(\psi)\cos(\theta) & -\sin(\psi)\cos(\phi) + \cos(\psi)\sin(\theta)\sin(\phi) & \sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi) \\ \sin(\psi)\cos(\theta) & \cos(\psi)\cos(\phi) + \sin(\psi)\sin(\theta)\sin(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) - \cos(\psi)\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}
 \end{aligned} \tag{5}$$

where the sign convention here is based on a righthanded ( $x, y, z$ ) coordinate system with  $x$  positive forward (to bow),  $y$  positive to port,  $z$  positive upward,  $\psi$  positive for the ship's bow yawed counter-clockwise from north,  $\phi$  positive for the port side rolled up, and  $\theta$  positive for the bow pitched down. Note that the right-handed definition of  $\psi$  is opposite the typical left-handed definition used for a compass. However, the incorporation of a compass in (5) simply requires multiplication of the compass heading by -1.

Equation (5) represents the coordinate system transform for a combination of the three separate rotations of the platform coordinate frame about the three axes of our frame of reference (i.e., the earth). Note that this total coordinate transformation matrix is dependent on the order of the three separate rotations. However, for small roll and pitch angles, such as those encountered on a large research vessel or discus buoy in the ocean environment (perhaps  $\pm 15^\circ$ ), the error due to the order of rotation is negligible. Additionally, the errors associated with the order of rotation are minimized by the 3, 2, 1 rotation used in (5).

### 3.2.2 The FDCHP Strapped-Down System

In contrast to a gyro-stabilized system, the FDCHP uses a "strapped-down" approach where the motion sensors are firmly attached to the buoy frame. The angular rate sensors used in this system directly measures the rate of angular rotation about the three axes in the buoy frame. In such systems, the angular rate vector is given by

$$\Omega_{obs} = \begin{pmatrix} \dot{\phi}_{obs} \\ \dot{\theta}_{obs} \\ \dot{\psi}_{obs} \end{pmatrix} \tag{6}$$

where the subscript *obs* denotes measurements made in the buoy frame of reference. We remind the reader that the yaw rate is defined positive for a left-handed rotation. This vector

can be related to the fixed frame angular rate through  $\vec{\Omega} = T\vec{\Omega}_{obs}$ . This relationship allows one to rewrite (4) using our direct measurements of angular rate in the buoy frame as

$$\vec{V}_{true} = T(\vec{V}_{obs} + \vec{\Omega}_{obs} \times \vec{R}) + \vec{V}_{mot} \quad (7)$$

where  $\vec{R}$  is the position vector of the wind sensor with respect to the motion package.

The difficulty then is to approximate the Euler angles ( $\phi$ ,  $\theta$ ,  $\psi$ ) from the strapped-down angular rate sensors. The general approach is to use  $\vec{\Omega} = T\vec{\Omega}_{obs}$  with (5) to obtain an expression for the time derivative of these angles in terms of the measured angular rates given by

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{pmatrix} \dot{\phi}_{obs} + [\dot{\psi}_{obs} \cos(\phi) + \dot{\theta}_{obs} \sin(\phi)] \tan(\theta) \\ \dot{\theta}_{obs} \cos(\phi) - \dot{\psi}_{obs} \sin(\phi) \\ [\dot{\psi}_{obs} \cos(\phi) + \dot{\theta}_{obs} \sin(\phi)] / \cos(\theta) \end{pmatrix} \quad (8)$$

The angles can then be approximated by integrating (8) and updating this matrix with successive approximation of  $\phi$ ,  $\theta$  and  $\psi$  (Fairall et al. 1997).

### 3.2.3 Complementary filtering

In practice, problems often arise with this approach due to the drift found in angular rate sensors. Therefore, in the approach used by the FDCHP, the angles are found by high-pass filtering the angles that are computed by integration of (8) and then adding these results to low-pass filtered reference angles using an approach known as complimentary filtering. These complimentary-filtered angles provide an estimate of  $\phi$  and  $\theta$ , which are used in the update matrix to provide better approximation of the Euler angles through subsequent iteration and integration.

The reference angles used in the FDCHP approach are found from the measured accelerations in the buoy frame of reference. In this frame of reference, the measured accelerometer output is a combination of the gravitational component due to the pitching and rolling of the buoy (i.e., due to tilting of the system) plus the accelerations arising from the motion of the buoy along the accelerometer axes

$$\begin{bmatrix} \ddot{x}_{obs} \\ \ddot{y}_{obs} \\ \ddot{z}_{obs} \end{bmatrix} = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} + \begin{pmatrix} -g \sin(\theta) \\ g \sin(\phi) \cos(\theta) \\ g \cos(\phi) \cos(\theta) \end{pmatrix} \quad (9)$$

where the double dots denote second derivatives of the position vector  $\vec{X} = \hat{i}x + \hat{j}y + \hat{k}z$ , and  $g$  is the gravitational acceleration. The second term on the right-hand side of (9) represents the tilt-induced acceleration. These tilt-induced accelerations will eventually have to be removed before we integrate our accelerometers to compute the buoy velocities as described in section 3. However, we can use these measured accelerations and angular rates to approximate the desired angles using complementary filtering.

The original DCFS system described in Edson et al. (1998) used true complementary filter with simply first-order Butterworth filters. This approach is easily illustrated using Laplace transform notation where, e.g., the roll is approximated by

$$\phi \approx \frac{1}{\tau s + 1} \frac{\ddot{y}_{obs}}{g} + \frac{\tau s}{\tau s + 1} \phi_{obs} \quad (10)$$

where  $s$  represents the differentiation operator such that  $\dot{\phi}_{obs} = s\phi_{obs}$ , and  $\tau$  is a time constant, and we have assumed that the tilts are small, i.e.,  $g \sin(\phi) \cos(\theta) \approx g\phi$ . The first term on the right-hand side is the low-frequency tilt reference from the accelerometers as follows from (9). The second term on the right-hand side represents a high-pass filter [i.e.,  $\tau s/(\tau s + 1)$ ] that integrates the angular rate sensors to provide the wave-induced angular motions. By filtering the signals in this way we do not introduce any time delays, that is, the process is an all-pass filter that removes the unwanted drift in the rate gyros while retaining the low-frequency tilt reference.

However, the first-order filter used in the original implementation allows significant leakage of the integrated angular rate component beyond the cutoff frequency. This generates noise due to integration of angular rate sensors that are often characterized by significant drift at low-frequencies. Therefore, the most recent method used to estimate the Euler angles uses a fourth-order Butterworth filter, which effectively removes the adverse effects of leakage into the lowest frequencies. The fourth-order filters are not integrating complimentary filters and will distort the phase. Instead, the angular rates are numerically integrated and the 4<sup>th</sup>-order high-pass filter is applied forward and then backward to the integrated time series to remove the phase shift (Miller et al. 2008). This time series is then added to the forward and backward low-pass filtered tilts from the accelerometers. This procedure can be summarized as:

$$\phi \approx \left[ \frac{\ddot{y}_{obs}}{g} - HP\left(\frac{\ddot{y}_{obs}}{g}\right) \right] + HP(\phi_{obs}) \quad (11)$$

where  $HP$  represents a high-pass filter operator applied in the forward and backward direction, and is numerically integrated prior to filtering using the trapezoidal formula. The bracketed term provides the low-pass filtered tilts from the accelerometers.

### 3.2.4 Platform Motion

The Euler angle estimates are then used to define the transformation matrix,  $T(\phi, \theta, \psi)$ , that is used to compute the platform velocities  $\vec{V}_{mot}$ . The platform velocity is defined as

$$\vec{V}_{mot} = \vec{V}_{lp} + \vec{V}_{hp} \quad (12)$$

where we have divided the velocities into low-pass ( $lp$ ) and high-pass ( $hp$ ) components. The high-pass platform velocities are computed by rotating the strapped down accelerations into the fixed frame using the transformation matrix, subtracting the gravity vector, integrating the remainder, and then high-pass filtering the resultant velocities,

$$\vec{V}_{hp} = HP\left[\int (T\vec{\ddot{x}}_{obs} + \vec{g})dt\right] \quad (13)$$

where  $\vec{g} = -\hat{k}g$  and  $g$  is the gravitational acceleration.

This high-pass component of the platform velocity provides an estimate of the true wind velocity relative to the buoy:

$$\vec{V}_{true}^{buoy} = T(\vec{V}_{obs} + \vec{\Omega}_{obs} \times \vec{R}) + \vec{V}_{hp} \quad (14)$$

The low-pass components are computed only for the horizontal velocities (i.e., we assume the buoy does not leave the ocean surface) using a GPS to measure the buoy speed relative to Earth or a current meter to measure the buoy speed relative to water. The combination of the high-pass and low-pass signals results in a value of  $\vec{V}_{mot}$  that describes the mean velocity relative to the frame of reference plus the fluctuating velocity components computed from our accelerometers. For example, the relative velocity components obtained from the current meters are rotated to

give the north and west components. When these are added from the north and west components of the true wind speed relative to the buoy, we obtain the wind velocity relative to the water:

$$\vec{V}_{true}^{water} = \vec{V}_{true}^{buoy} + \vec{V}_0 \quad (15)$$

where  $\vec{V}_0$  is the buoy velocity relative to water. Obviously, in the absence of a current meter, the velocities are measured relative to Earth.

### 3.3 Known Theoretical Limitations

The velocity measurements made on a surface buoy differ from those made on a fixed platform. The buoy measurements are essentially made in a wave-following coordinate system while the tower measurements are made relative to earth. This causes uncertainty on how to interpret fluxes made in either coordinate system. For example, measurements from fixed platforms (e.g., Hristov et al. 2001) show clear wave-induced fluctuations in the measured velocities. The correlation between these fluctuations is associated with a wave-induced component of the momentum flux at the height of measurement. However, the (quasi-potential) flow is expected to follow the long waves upon which a buoy rides. Therefore, one would expect to see less wave-induced fluctuations measured by an anemometer in a wave-following coordinate system. The same correlation associated with the wave-induced momentum flux is still expected because these are mainly a result of the non-potential (rotational) component of the flow. However, there remains some uncertainty as to how to remove the wave-induced platform motion in this coordinate system. For example, if the anemometer is generally in a coordinate system following the flow, then “removing” the low frequency platform velocities that are not actually part of the measured wind velocities may be adding noise. Research to date has shown that this is mainly a problem in light winds over swell, i.e., old seas.

One simple solution is to move the cutoff frequency to a higher value, which effectively removes the low-frequency platform velocities that are not actually present in the measured wind velocities. This requires a means to dynamically choose the value of the cutoff frequency based on, e.g., wave age or wave slope. This approach is an area of active research and we expect such a capability in future revisions. It should be noted that the initial version of the code sets the cutoff frequency to 1/(12 seconds), such that it is generally at a slightly lower frequency than the dominant waves.

### 3.4 Revision History

No revisions to date.

## 4 Implementation

### 4.1 Overview

The current implementation of the FDCHP collects all of the data required by (14) to compute the true wind speed relative to the buoy for 20 minutes out of every hour. The sonic anemometer and IMU data is merged, time-stamped using the processor’s internal clock and stored as the level 0 products **WINDTUR\_L0**, **TMPATUR\_L0** and **MOTFLUX\_L0**. The processor will compute ancillary data that will be telemetered to shore to monitor system performance. However, DPAs are not required for this data.

Once recovered, the time series are processed using DPAs that carry out the above steps to compute the motion-corrected time series of wind speed **WINDTUR\_L1** and **TMPATUR\_L1**. Thirty seconds of data from the beginning and end of the 20 minute time series are discarded to remove the edge effect of the filters using in the routines. The remaining 19 minutes of true wind velocities are rotated into the longitudinal (streamwise) wind. This rotation forces the mean cross-wind and vertical components of the wind to zero and has been shown to reduce the

effect of flow distortion on the fluxes (Oost et al. 1994). The rotated velocities and sonic temperature are then linearly detrended by removing a least-squares-fit to the time series, which provides fluctuations about removed trend. The kinematic components of the buoyancy, along-wind, and cross-wind momentum fluxes, i.e.,  $\overline{w'T_s'}$ ,  $\overline{u'w'}$  and  $\overline{v'w'}$ , respectively, are then computed to provide estimates of (1) and (2). The inclusion of the surface currents required to compute the fluxes relative to water using (15) as well as the small correction to account for the use of sonic temperature (i.e., rather than virtual temperature) would be carried out in post-processing.

## 4.2 Inputs

The inputs for the Data Processing Algorithms (DPAs) are the measured time series from the sonic anemometer/thermometer and IMU, which are sampled at 10 Hz. Specifically, these are:

- The Level 0 Data Product **WINDTUR\_L0**, which is the **wind speed components** [i.e.,  $U(t)$ ,  $V(t)$ , and  $W(t)$ ] measured in the buoy frame of reference by the sonic anemometer;
- The Level 0 Data Product **TMPATUR\_L0**, which is the **speed of sound** [i.e.,  $C_s(t)$ ] measured by the sonic anemometer;
- The Level 0 Data Product **MOTFLUX\_L0**, which is:
  - o the **horizontal and vertical linear accelerations** [i.e.,  $\ddot{x}(t)$ ,  $\ddot{y}(t)$  and  $\ddot{z}(t)$ ] measured in the buoy frame of reference by the IMU accelerometers;
  - o the **roll, pitch and yaw rates** [i.e.,  $\dot{\phi}(t)$ ,  $\dot{\theta}(t)$  and  $\dot{\psi}(t)$ ] measured in the buoy frame of reference measured by the IMU gyros; and
  - o the **roll, pitch and yaw** [i.e.,  $\phi(t)$ ,  $\theta(t)$  and  $\psi(t)$ ] measured in the buoy frame of reference measured by the IMU magnetometer.

The raw 10-Hz input data (i.e., **MOTFLUX\_L0**, **TMPATUR\_L0**, and **WINDTUR\_L0** data products) are stored on board the buoy in the processors non-volatile system memory, such that all of the data required to reprocess the data will be available on recovery. These variables are stored in the units outputted from the sonic anemometer/thermometer and IMU. These units and the factory calibration required to convert them to physical units is summarized in **Table 1**.

<b>Table 1.</b> Units of stored Level 0 data products and factory calibrations to convert to physical units.					
Device	Variable(s) <i>Symbol</i>	Stored Units	Factory Calibration		Physical Units
			Gain	Offset	
Sonic Anemometer	Velocity Components $U, V, W$	Counts	0.01	0	m/s
Sonic Anemometer	Speed of Sound $C_s$	Counts	0.01	0	m/s
Accelerometers	Linear Accelerations $\ddot{x}, \ddot{y}, \ddot{z}$	Decimal Counts	1	0	m/s <sup>2</sup>
Angular Rate Sensors	Angular Rates $\dot{\phi}, \dot{\theta}, \dot{\psi}$	Decimal Counts	1	0	radians/s
Magnetometer	Angles $\phi, \theta, \psi$	Decimal Counts	1	0	radians

The speed of sound is converted to sonic temperature using the following equation



$$T_s [^{\circ}C] = \frac{C_s^2}{403} - 273.15K \quad (16)$$

The factory calibrated velocity, sonic temperature and motion data are the inputs to the motion correction DPA shown in the FCDHP Processing Flow document (1342-00280).

### 4.3 Processing Flow

Data Product Algorithms (DPAs) will process the **Level 0** data upon recovery to compute the Level 1 data products **WINDTUR\_L1** and **TMPATUR\_L1**, which are time series of the motion-corrected wind velocity and sonic temperature, respectively. The Level 1 products are then used to produce the Level 2 data products **FLUXMOM-U\_L2**, **FLUXMOM-V\_L2**, and **FLUXHOT\_L2**, which are the along-wind momentum, cross-wind momentum and buoyancy fluxes, respectively.

The specific steps necessary to create the FCDHP Level 1 and 2 data products are shown in the FCDHP Processing Flow document (1342-00280) and can be summarized as:

1. FCDHP Dataset Agent Driver reads in **WINDTUR\_L0**, **TMPATUR\_L0**, and **MOTFLUX\_L0** from recovered data and passes them to the DPA (see Appendix A-1 for example code).
2. The factory calibrations and (16) are applied to convert data to physical units as summarized in Table 1.
3. Secondary post-deployment calibrations are read in and applied as necessary.
4. Despiking and additional automated quality control (QC) is applied to the time series.
5. The calibrated and automated QC data quality are passed to the motion correction DPA:
  - a. The Euler angles are approximated using (11).
  - b. These angles are used to update the angular rates using (8) with successive approximation of  $\phi$ ,  $\theta$  and  $\psi$  using (11). This is repeated 5 times.
  - c. The Euler angles are used to rotate the measured accelerations into the vertical using the transformation matrix given by (5).
  - d. The gravity vector is removed and the accelerations are integrated and filtered using (13) to compute the platform velocity,  $\vec{V}_{hp}$ .
  - e. The measured wind velocities,  $\vec{V}_{obs}$ , are added to the measured angular velocity,  $\vec{\Omega}_{obs} \times \vec{R}$ .
  - f. Their sum is transformed into the vertical and added to the platform velocity to produce the wind velocities relative to Earth,  $\vec{V}_{true}^{buoy}$ , as shown by (14).
6. The first and last 30 seconds of  $\vec{V}_{true}^{buoy}$  are removed to produce **WINDTUR\_L1**, which contains 3 element time series of the wind components relative to Earth. The right-handed orientation of the wind components are:
  - WINDTUR-VLN\_L1** is positive to the North,
  - WINDTUR-VLW\_L1** is positive to the West
  - WINDTUR-VLU\_L1** is positive upward
7. The velocity components in **WINDTUR\_L1** are rotated into the longitudinal (streamwise) wind.
8. The first and last 30 seconds of the sonic temperature are removed to match the velocity time series.
9. The rotated velocities and sonic temperature are linearly detrended to provide the velocity and temperature fluctuations  $u'$ ,  $v'$ ,  $w'$  and  $T_s'$ .

10. The vertical velocity fluctuations are correlated with the temperature and horizontal velocity fluctuations to compute the kinematic form of the buoyancy flux,  $\overline{w'T_s'}$ , along-wind momentum flux,  $\overline{u'w'}$ , and cross-wind momentum flux,  $\overline{v'w'}$ .
11. These provide the Level 2 fluxes computed over 19 minute averaging periods:
- FLUXHOT\_L2** =  $\overline{w'T_s'}$
  - FLUXMOM-U\_L2** =  $\overline{u'w'}$
  - FLUXMOM-V\_L2** =  $\overline{v'w'}$ .

**Note:** The wind speeds relative to earth and water are required to compute the fluxes relative to water. Bulk estimates of the latent heat flux are required to provide the small correction needed to convert  $\overline{w'T_s'}$  to  $\overline{w'T_v'}$ . Additionally, bulk estimates of the air density,  $\rho$ , and specific heat at constant pressure,  $c_p$ , are required to convert the kinematic values into the momentum and heat fluxes with units of  $N/m^2$  and  $W/m^2$ , respectively.

## 4.4 Outputs

4.4.1 The following L1 Products are output

**WINDTUR\_L1:** 3-element 19 minute time series of motion-corrected velocity vector relative to Earth where the orientation of the wind components are:

**WINDTUR-VLN** is positive to the North [m/s]

**WINDTUR-VLW** is positive to the West [m/s]

**WINDTUR-VLU** is positive upward [m/s]

**TMPATUR:** Time series of sonic temperature in °C.

4.4.2 The following L2 Products are output:

**FLUXMOM-U:**  $\overline{u'w'}$  Along-wind component of momentum flux [ $m^2/s^2$ ] relative to Earth

**FLUXMOM-V:**  $\overline{v'w'}$  Cross-wind component of momentum flux [ $m^2/s^2$ ] relative to Earth

**FLUXHOT:**  $\overline{w'T_s'}$  Buoyancy Flux [ $m/s \text{ } ^\circ K$ ]

## 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 example input and output data are accessible through the following path:

<https://alfresco.oceanobservatories.org/> and navigate to OOI >> REFERENCE >> Data Product Specification Artifacts >> 1341-00280\_FDCHP

## Appendix A Example C and Matlab processing code (UConn/WHOI)

The MATLAB® code used to process the input data has been converted to an executable file to run on the buoy processor using MATLAB's Coder®. This required conversion of the MATLAB® processing code and the intrinsic functions it calls to C++ files. Only a subset of intrinsic functions is available with Coder®. Those functions that were not available (e.g., the MATLAB® `filtfilt` and `medfilt1` functions) have been created as functions within the processing code.

The processing code and any intrinsic functions that had to be written are given by:

```
function [fluxes] = ProcessDCFS03(rawdata,lat);

Ts1 = 10/100;           % Sampling period for DCFS
fs = 1/Ts1;            % Sampling frequency for R2
dt=1/fs;
tc1=12;                %Define constants for filters
tc2=tc1;
fc1=1/tc1;
fc2=1/tc2;
fcwaves=1/40;
rad2deg=180.0/pi;
G=9.80665;

% JBE 06/29 JW 07/18
version_number=1.3;
status_val = uint32(1);

%JBE Redefine files for 10 Hz and tc1=12
ahi=[1,-3.869797539975553,5.617802044587563,-3.625896801659080,0.877898078061700];
bhi=[0.936962154017744,-3.747848616070974,5.621772924106461,-
3.747848616070974,0.936962154017744];

gv=grv(lat);
Rvec=zeros(1,3);      %Distance vector
Rvec(3)=0.753;        %Vertical separation
roffset=0;            %Maybe non-zero for post-calibration
poffset=0;

L=12000;              %Fix length
%L=length(rawdata);

dcfsdata=zeros(15,L);

dcfsdata(1,1:L) =
datenum(rawdata(1,1:L),rawdata(2,1:L),rawdata(3,1:L),rawdata(4,1:L),rawdata(5,1:L),rawdata(6,1:L));
%UNITS#3 Velocities are m/s
dcfsdata(2,1:L) = 0.01 * rawdata(8,1:L); %wind x
dcfsdata(3,1:L) = 0.01 * rawdata(9,1:L); %wind y
dcfsdata(4,1:L) = 0.01 * rawdata(10,1:L); %wind z
dcfsdata(5,1:L) = 0.01 * rawdata(11,1:L); %Speed of sound
% Convert Sonic Speed of Sound to temperature
dcfsdata(5,1:L) = dcfsdata(5,1:L).* dcfsdata(5,1:L)/403 - 273.15;
%UNITS#4 - Rates are in radians/s, accels are in G=9.80665 m/s^2, pitch,roll and yaw are in radians.
dcfsdata(6,1:L) = rawdata(20,1:L); %heading
dcfsdata(7,1:L) = rawdata(18,1:L); %roll
dcfsdata(8,1:L) = rawdata(19,1:L); %pitch
dcfsdata(9,1:L) = rawdata(12,1:L); %rate x
dcfsdata(10,1:L) = rawdata(13,1:L); %rate y
dcfsdata(11,1:L) = rawdata(14,1:L); %rate z
%JBE The temperature is no longer output from IMU - Use this as counter
```

```

dcfsdata(12,1:L) = 1:L';           % counter
dcfsdata(13,1:L) = rawdata(15,1:L); % accel x
dcfsdata(14,1:L) = rawdata(16,1:L); % accel y
dcfsdata(15,1:L) = rawdata(17,1:L); % accel z
disp(mean(rawdata(17,1:L)));
disp(datestr(dcfgdata(1,1)));

% Convert IMU from North East Down (right-handed z-down) coordinate system
% to North West Up (right-handed z-up) coordinate system to match Sonic

dcfsdata(8,1:L) = -1.0*dcfsdata(8,1:L); %y pitch
dcfsdata(6,1:L) = -1.0*dcfsdata(6,1:L); %z heading(yaw)
dcfsdata(10,1:L) = -1.0*dcfsdata(10,1:L); %rate y
dcfsdata(11,1:L) = -1.0*dcfsdata(11,1:L); %rate z
dcfsdata(14,1:L) = -1.0*dcfsdata(14,1:L); %accel y
dcfsdata(15,1:L) = -1.0*dcfsdata(15,1:L); %accelz

sonics=zeros(3,L);
Tv=zeros(1,L);
compass=zeros(1,L);
roll=zeros(1,L);
pitch=zeros(1,L);
platform=zeros(3,L);
deg_rate=zeros(3,L);

counter=dcfgdata(12,1:L);

%*****
% Sonic mean, max, min, std of velocities
%*****
sonics(1:3,1:L)=dcfgdata(2:4,1:L);
rdir=atan2(mean(sonics(2,1:L)),mean(sonics(1,1:L)));

%*****
% Sonic mean, max, min, std of temperature
%*****
Tv=dcfgdata(5,1:L);

%*****
% Prep Compasss
%*****
% Fill in bad points first units and signs
% UNITS#5-Roll, pitch and yaw are in radians
compass=dcfgdata(6,1:L);
roll=dcfgdata(7,1:L);
pitch=dcfgdata(8,1:L);
compass(1:10)=compass(11); %first few points are often bad
compass(L-9:L)=compass(L-10); %last few points are often bad

%*****
% UNITS#6 Calc compstd,compmin and compmax in radians
%*****

gx=cos(compass);
gy=sin(compass);
compcos = mean(gx);
compsin = mean(gy);
compavg = atan2(compsin,compcos);

%JBE Leave this in case we add or subtract when post-calibrating

if (compavg < 0)

```

```

    compavg=compavg + 2.0*pi;
elseif (compavg >= 2*pi)
    compavg=compavg -2.0*pi;
end

[gx] = despikesimple(gx);
[gy] = despikesimple(gy);
gsmooth=atan2(gy,gx);
i=find(gsmooth<0);
gsmooth(i)=gsmooth(i)+2*pi;
gyro=gsmooth;

gchk=unwrap(gyro);
stdhdg=std(gchk);
hdgrange=max(gchk)-min(gchk);
if (hdgrange>(120/180*pi) | stdhdg>(45/180*pi))
    goodcompass=0;
else
    goodcompass=1;
end
disp(goodcompass)
%*****
% Angular rate mean, max, min and std
% UNITS#9 Rates are in radian/sec
%*****
deg_rate = dcfsdata(9:11,1:L);
[deg_rate] = despikesimple(deg_rate);

%*****
% Accelerometer mean, max, min and std
% UNITS#8 convert platform accels to m/s^2
%*****
platform = dcfsdata(13:15,1:L)*G;
[platform] = despikesimple(platform);
gcomp=zeros(1,3);
gcomp(1)=mean(platform(1,1:L));
gcomp(2)=mean(platform(2,1:L));
gcomp(3)=mean(platform(3,1:L));

g=sqrt(sum(gcomp.*gcomp));
platform=platform*gv/g;
platform(1,:)=platform(1,1:L)+poffset;
platform(2,:)=platform(2,1:L)+roffset;

gcomp(1)=mean(platform(1,1:L));
gcomp(2)=mean(platform(2,1:L));
gcomp(3)=mean(platform(3,1:L));
g=sqrt(sum(gcomp.*gcomp));
platform=platform*gv/g;

%*****
% Compute Angles and Accelerations
%*****
its=5;
[euler,dr] = anglesclimodeyaw(ahi,bhi,fs,platform,deg_rate,gyro,its,goodcompass,L); % euler angles are
right-handed
[acc, uvwplat, nope] = accelsclimode(bhi,ahi,fs,platform,euler,L);
[uvw,uvwr,uvwrot] = sonic(sonics,dr,euler,uvwplat,Rvec,L);

edge = fix(1 * 30 * fs);
tot=length(uvw)-edge*2;
incr1=1+edge;

```

```
incr2=incr1+tot-1;
incr=incr1:incr2;
```

```
UVWraw=sonics(1:3,incr);
UVW=uvw(1:3,incr); %These is L1
Ts=Tv(incr); %This is L1
```

```
[u, alpha, beta] = alignwind(UVW);
```

```
%*****
% Fluxes are computed relative to Earth
%*****
wspd=mean(u(1,1:tot));
u=u';
uh=sqrt(u(1:tot,1).*u(1:tot,1)+u(1:tot,2).*u(1:tot,2));
u=detrend(u);
Ts=detrend(Ts);
fluxes=zeros(1,3);
uwavg=mean(u(1:tot,3).*u(1:tot,1));
vwavg=mean(u(1:tot,3).*u(1:tot,2));
wTavg=mean(u(1:tot,3).*Ts);
```

```
fluxes(1) =uwavg; %These are L2
fluxes(2)=vwavg;
fluxes(3)=wTavg;
```

```
end
```

```
% *****
% Function calls
% *****
```

```
function [euler, dr] = anglesclimodeyaw(ahi,bhi,sf,accm,ratem,gyro,its,goodcompass,L)
```

```
%#codegen
% Function from EDDYCORR toolbox
%
% Sept 2000 Replaced integrations with cumtrapz function
%
% May 16 1997 - modified to remove the first estimate of the euler
% angles in the nonlinear euler angle update matrix, F^-1 matrix
% is approximated by the identity matrix. still uses trapezoidal
% integration
%
% INPUT
%
% ahi,bhi - filter coefficients
% sf - sampling frequency
% accm - (3xN) array of recalibrated linear accelerations,accx,accy,accz
% ratem - (3xN) array of recalibrated angular rates, ratex, ratey, ratez
% gyro - (1xN) array of gyro signal
% its - number of iterations
%
% OUTPUT
```

```
%
% euler - (3xN) array of the euler angles (phi, theta, psi) in radians.
%
```

```
%%%%%%%%%%
%%%%%%%%%%
```

```
% THE ANGLES ARE ESTIMATED FROM
```

```
%
% angle = slow_angle (from accelerometers) + fast_angle (integrated rate sensors)
%
```

## % CALCULATE GRAVITY

```

gravxyz=zeros(1,3);
gravxyz(1) = mean(accm(1,1:L));
gravxyz(2) = mean(accm(2,1:L));
gravxyz(3) = mean(accm(3,1:L));
gravity = sqrt( sum(gravxyz.^2) );

```

```

% Unwrap compass
gyro = unwrap(gyro);

```

## % REMOVE MEAN FROM RATE SENSORS

```

ratem = detrend(ratem');

```

```

% LOW FREQUENCY ANGLES FROM ACCELEROMETERS AND GYRO
% SLOW ROLL FROM GRAVITY EFFECTS ON HORIZONTAL ACCELERATIONS. LOW PASS
% FILTER SINCE HIGH FREQUENCY HORIZONTAL ACCELERATIONS MAY BE 'REAL'
%

```

## % PITCH

```

accm1grav = -accm(1,1:L)./gravity; %Small angle approx
accm1grav = min(accm1grav,1);
accm1grav = max(accm1grav,-1);
theta = asin(accm1grav);

```

```

thetaslow = theta - filtfilt(bhi,ahi,theta);

```

## % ROLL

```

accm2grav = accm(2,1:L)./gravity; %Small angle approx
accm2grav = accm2grav./cos(thetaslow);
accm2grav = min(accm2grav,1);
accm2grav = max(accm2grav,-1);
phi=asin(accm2grav);

```

```

phislow = phi - filtfilt(bhi,ahi,phi);

```

## % YAW

```

% HERE, WE ESTIMATE THE SLOW HEADING. THE 'FAST HEADING' IS NOT NEEDED
% FOR THE EULER ANGLE UPDATE MATRIX. THE NEGATIVE SIGN PUTS THE GYRO
% SIGNAL INTO A RIGHT HANDED SYSTEM.
% IF THE COMPASS HAS ISSUES, WE EXTEND THE FILTER
% SO THE INTEGRATED YAW GOES OUT TO LOW FREQUENCIES

```

```

fc=1/240;
ahi2=[1,-3.993489157035384,5.980488658273062,-3.980509805074932,0.993510303875667];
bhi2=[0.996749870266190,-3.986999481064761,5.980499221597142,-
3.986999481064761,0.996749870266190];
if goodcompass
    psislow = -gyro - filtfilt(bhi2,ahi2,-gyro);
else
    psislow = -median(gyro)*ones(size(phi));
end

```

## % USE SLOW ANGLES AS FIRST GUESS

```

euler = [phislow; thetaslow; psislow];
rates = update(ratem,euler,L);

```

## % INTEGRATE AND FILTER ANGLE RATES, AND ADD TO SLOW ANGLES

```

for i = 1:its
    phi_int = 1/sf*cumtrapz(rates(1,1:L));
    phi_int = phislow + filtfilt(bhi,ahi,phi_int);
    theta_int = 1/sf*cumtrapz(rates(2,1:L));

```

```

theta = thetaslow + filtfiler(bhi,ahi,theta_int);
psi_int = 1/sf*cumtrapz(rates(3,1:L));

if goodcompass
    psi = psislow + filtfiler(bhi2,ahi2,psi_int);
else
    psi = psislow + psi_int;
end
euler = [phi; theta; psi];
rates = update(ratem,euler,L);
rates(1:2,1:L) = detrend(rates(1:2,1:L),'constant');
rates(3,1:L) = detrend(rates(3,1:L),'constant');
end

dr = ratem;
end

function [Y] = despikesimple(Y);
%Remove Outliers

[col, N]=size(Y);
t=1:N;
for iter=1:3 %Do it threetime
    for tot=1:col
        M=median(Y(tot,1:N));
        S=std(Y(tot,1:N));
        j=find(Y(tot,1:N)<M+4*S & Y(tot,1:N)>M-4*S);
        Y(tot,1:N)=interp1(t(j),Y(tot,j),t,'nearest');
    end
end

end

function [acc,uvwplat,xyzplat] = accelsclimode(bhi,ahi,sf,accm,euler,L)
%#codegen
% Function from EDDYCORR toolbox
%
% 2008 Allows filter to have different cutoff from angular filter
%
% Sept 2000 Replaced integrations with cumtrapz function
%
% Mar 3 1998 Redesigned the high pass filter (see below).
%
% Revised: June 10, 1997 - high pass filter with higher cutoff
% frequency than previous version
%
% Integrate linear accelerations to get platform velocity
% and displacement. After each integration, signals are
% high pass filtered to remove low frequency effects.
%
% INPUT
%
% bhigh,ahigh - high pass filter coefficients
% sf - sampling frequency
% accm - calibrated linear accelerations (output from recal.m)
% euler - (3xN) Euler angles phi,theta,psi
%
% OUTPUT:
%
% acc - (3XN) linear accelerations in FLIP/Earth reference
% uvwplat - (3XN) linear velocities at the point of motion measurement
% xyzplat - (3XN) platform displacements from mean position

```



```

% DEFINE ARRAY SIZES UP FRONT

gravxyz=zeros(1,3);
gravxyz(1) = mean(accm(1,1:L));
gravxyz(2) = mean(accm(2,1:L));
gravxyz(3) = mean(accm(3,1:L));
gravity = sqrt( sum(gravxyz.^2) );

acc = trans(accm,euler,0,L); % first rotate
acc(3,1:L) = acc(3,1:L) - gravity; % remove gravity

% INTEGRATE ACCELERATIONS TO GET PLATFORM VELOCITIES

uvwplat = zeros(size(accm));
for i=1:3
    uvwplat(i,1:L) = cumtrapz(acc(i,1:L))/sf;
    uvwplat(i,1:L) = filtfilt(bhi,ahi,uvwplat(i,1:L));
end

% INTEGRATE AGAIN TO GET DISPLACEMENTS

xyzplat = zeros(size(accm));
for i=1:3
    xyzplat(i,1:L) = cumtrapz(uvwplat(i,1:L))/sf;
    xyzplat(i,1:L) = filtfilt(bhi,ahi,xyzplat(i,1:L));
end
end

function [uvw,uvw,uvwrot] = sonic(sonics,omegam,euler,uvwplat,R,L)
%#codegen
% Function from EDDYCORR toolbox
%
% CORRECT SONIC ANEMOMETER COMPONENTS FOR PLATFORM MOTION AND ORIENTATION.
%
% INPUTS:
%
% Sonics - row of integers corre to sonic numbers which are to be
% corrected
% omegam - (3XN) measured angular rate 'vector' in platform frame
% euler - (3XN) array of euler angles (phi, theta, psi)
% uvwplat - (3XN) array of platform velocities (output from accels_.m)
%
% OUTPUTS:
%
% uvw - (MXN) array of corrected sonic anemometer components, in the
% fixed earth reference frame (North-West-up)
%
% CALCULATE WINDS IN EARTH BASED FRAME. THE ANGULAR VELOCITY IS CALCULATED AS
% THE CROSS PRODUCT BETWEEN THE ANGULAR RATE VECTOR AND POSITION VECTOR.THE
% MEASURED AND ANGULAR VELOCITIES ARE IN THE PLATFORM FRAME AND MUST BE
% ROTATED INTO THE EARTH FRAME. THE PLATFORM VELOCITY IS ALREADY IN THE EARTH
% FRAME (FROM ACCELS.M), SO HERE THEY CAN JUST BE ADDED.
%
% UVW = MEASURED VELOCITY + ANGULAR RATE INDUCED VELOCITIES +
% INTEGRATED ACCELEROMETERS

Rvec = [R(1); R(2); R(3)] * ones(1,L);
uvwrot = cross(omegam,Rvec);

uvw = trans(sonics + uvwrot,euler,0,L) + uvwplat;
uvw = trans(sonics + uvwrot,euler,0,L);

```

```

end

function OUT = trans(IN,ANGLES,IFLAG,L)

if nargin==2
    IFLAG=0;
    L=12000;
end

sinp = zeros(1,L);
cosp = zeros(1,L);
sint = zeros(1,L);
cost = zeros(1,L);
sinps = zeros(1,L);
cosps = zeros(1,L);

sinp = sin(ANGLES(1,1:L));
cosp = cos(ANGLES(1,1:L));
sint = sin(ANGLES(2,1:L));
cost = cos(ANGLES(2,1:L));
sinps = sin(ANGLES(3,1:L));
cosps = cos(ANGLES(3,1:L));

up = IN(1,1:L);
vp = IN(2,1:L);
wp = IN(3,1:L);

if IFLAG      % =1, from xyz to x'y'z'

    u = up.*cost.*cosps          + vp.*cost.*sinps          - wp.*sint;
    v = up.*(sinp.*sint.*cosps-cosp.*sinps) + vp.*(sinp.*sint.*sinps+cosp.*cosps) + wp.*(cost.*sinp);
    w = up.*(cosp.*sint.*cosps+sinp.*sinps) + vp.*(cosp.*sint.*sinps-sinp.*cosps) + wp.*(cost.*cosp);

else      % =0, from x'y'z' to xyz

    u = up.*cost.*cosps + vp.*(sinp.*sint.*cosps-cosp.*sinps) + wp.*(cosp.*sint.*cosps+sinp.*sinps);
    v = up.*cost.*sinps + vp.*(sinp.*sint.*sinps+cosp.*cosps) + wp.*(cosp.*sint.*sinps-sinp.*cosps);
    w = up.*(-sint)      + vp.*(cost.*sinp)          + wp.*(cost.*cosp);

end;

OUT = [u;v;w];
end

function OUT = update(IN,ANGLES,L)
% Function from EDDYCORR toolbox
%
% This function computes the angular update matrix
% as described in Edson et al. (1998) and Thwaites
% (1995) page 50.

p = ANGLES(1,1:L);
t = ANGLES(2,1:L);
ps = ANGLES(3,1:L);

up = IN(1,1:L);
vp = IN(2,1:L);
wp = IN(3,1:L);

u = up + vp.*sin(p).*tan(t) + wp.*cos(p).*tan(t);
v = 0 + vp.*cos(p)          - wp.*sin(p);
w = 0 + vp.*sin(p)./cos(t) + wp.*cos(p)./cos(t);

```

```
OUT = [u;v;w];
end

function [u, alpha, beta] = alignwind(U)
% Function from EDDYCORR toolbox
%
Ub = mean(U(1,:));
Vb = mean(U(2,:));
Wb = mean(U(3,:));
Sb = sqrt(Ub^2+Vb^2);
beta = atan2(Wb,Sb);
alpha = atan2(Vb,Ub);
Ur = U(1,:)*cos(alpha)*cos(beta) + U(2,:)*sin(alpha)*cos(beta) + U(3,:)*sin(beta);
Vr = -U(1,:)*sin(alpha) + U(2,:)*cos(alpha);
Wr = -U(1,:)*cos(alpha)*sin(beta) - U(2,:)*sin(alpha)*sin(beta) + U(3,:)*cos(beta);

% predefine u for coder
nU=length(U);
u=zeros(3,nU);

u(1,:) = Ur;
u(2,:) = Vr;
u(3,:) = Wr;

beta = beta*180/pi;
alpha = alpha*180/pi;
end

function g=grv(lat)
gamma=9.7803267715;
c1=0.0052790414;
c2=0.0000232718;
c3=0.0000001262;
c4=0.0000000007;

phi=lat*pi/180;
x=sin(phi);
g=gamma.*(1+c1*x.^2+c2*x.^4+c3*x.^6+c4*x.^8);
end

function y=medfilt(x,n)
% This version of medfilt can only hand one time series at a time
blksz = [];
DIM = [];

% Check if the input arguments are valid
if isempty(n)
    n = 3;
end

[x, nshifts] = shiftdim(x);

% Verify that the block size is valid.
siz = size(x);
blksz = siz(1); % siz(1) is the number of rows of x (default)

% Initialize y with the correct dimension
y = zeros(siz);

% Call medfilt1D (vector)
for i = 1:prod(siz(2:end)),
```

```

    y(:,i) = medfilt1D(x(:,i),n,blksz);
end

% Convert y to the original shape of x
y = shiftdim(y, -nshifts);
end

function y = medfilt1D(x,n,blksz)
%MEDFILT1D One dimensional median filter.
%
% Inputs:
% x - vector
% n - order of the filter
% blksz - block size

nx = length(x);
if rem(n,2)~=1 % n even
    m = n/2;
else
    m = (n-1)/2;
end
X = [zeros(m,1); x; zeros(m,1)];
y = zeros(nx,1);

% Work in chunks to save memory
indr = (0:n-1)';
indc = 1:nx;
blksz=1;
for i=1:blksz:nx
    ind = indc(ones(1,n),i:min(i+blksz-1,nx)) + ...
        indr(:,ones(1,min(i+blksz-1,nx)-i+1));
    xx = reshape(X(ind),n,min(i+blksz-1,nx)-i+1);
    y(i:min(i+blksz-1,nx)) = median(xx,1);
end
end

function yout = filtfilter(b,a,xin)
% x must be a column vector for this to work
%FILTFILT Zero-phase forward and reverse digital IIR filtering.
% Y = FILTFILT(B, A, X) filters the data in vector X with the filter
% described by vectors A and B to create the filtered data Y. The
% filter is described by the difference equation:
%
% 
$$a(1)y(n) = b(1)x(n) + b(2)x(n-1) + \dots + b(nb+1)x(n-nb) - a(2)y(n-1) - \dots - a(na+1)y(n-na)$$

%
% The length of the input X must be more than three times
% the filter order, defined as max(length(B)-1,length(A)-1).
%
% References:
% [1] Sanjit K. Mitra, Digital Signal Processing, 2nd ed.,
% McGraw-Hill, 2001
% [2] Fredrik Gustafsson, Determining the initial states in forward-
% backward filtering, IEEE Transactions on Signal Processing,
% pp. 988-992, April 1996, Volume 44, Issue 4

% Copyright 1988-2010 The MathWorks, Inc.
% $Revision: 1.7.4.8 $ $Date: 2011/05/13 18:07:25 $

% If input data is a row vector, convert it to a column
isRowVec = size(xin,1)==1;
if isRowVec

```

```
x = xin(:);
else
  x=xin;
end
[Npts,Nchans] = size(x);

%-----
% Parse coefficients vectors and determine initial conditions
% b and a are vectors that define the transfer function of the filter
%-----
[L,nfilt] = size(b);
% Check coefficients
b = b(:);
a = a(:);
nfact = 3*(nfilt-1); % length of edge transients

% The non-sparse solution to zi may be computed using:
zi=(eye(nfilt-1) - [-a(2:nfilt), [eye(nfilt-2); zeros(1,nfilt-2)]]);
zi=zi \ (b(2:nfilt) - b(1)*a(2:nfilt));

% Filter the data
y = [2*x(1)-x(nfact+1:-1:2); x; 2*x(end)-x(end-1:-1:end-nfact)];

% filter, reverse data, filter again, and reverse data again
y = filter(b,a,y,zi*y(1));
y = y(end:-1:1);
y = filter(b,a,y,zi*y(1));

% retain reversed central section of y
y = y(end-nfact:-1:nfact+1);
if isRowVec
  yout = y.'; % convert back to row if necessary
else
  yout = y;
end
end
```

## **Appendix B      Output Accuracy**

For an accuracy analysis of the flux measurements, see Bigorre, et al. (2013) and F. Bradley and C. Fairall (2006), section “B3. Estimate of Turbulent Flux Errors”.

## **Appendix C      Sensor Calibration Effects**

<N/A>