Contents

[WHOI READ SENSOR DATA 2](#_Toc502133120)

[function X = d3readswv(recdir,prefix,df) 2](#_Toc502133121)

[UM READ SENSOR DATA 2](#_Toc502133122)

[function TagData = readSwv(TagData, recdir) 2](#_Toc502133123)

[WHOI CALIBRATION FUNCTIONS 3](#_Toc502133124)

[function [A,CAL,fs] = d3calacc(X,CAL,test,mindepth,ch) 3](#_Toc502133125)

[function [p,CAL,fs] = d3calpressure(X,CAL,test) 3](#_Toc502133126)

[function [M,CAL,fs,Mz] = d3calmag(X,CAL,test,mindepth) 4](#_Toc502133127)

[UM CALIBRATION FUNCTIONS 4](#_Toc502133128)

[function TagData = optCalib(TagData) 4](#_Toc502133129)

[function TagData = estOrient(TagData, orientTab) 6](#_Toc502133130)

[TagData = orderfields(TagData); 7](#_Toc502133131)

[function TagData = shiftDepth(TagData, method, ENABLE\_PLOT) 7](#_Toc502133132)

[Add Metadata to Structure 9](#_Toc502133133)

[TagData.localdeployinfo = DeployInfoLoc; 9](#_Toc502133134)

[Truncate Relevant Sensor Data to Animal-Only Portion 10](#_Toc502133135)

[Calculate Summary Statistics 10](#_Toc502133136)

[**Magnitude (Q): as a 4th dimension.** 11](#_Toc502133137)

[**Acceleration Statistics** 11](#_Toc502133138)

[**AccelStat Function:** 11](#_Toc502133139)

[%%%%%%%%%%%%%%%% To DO %%%%%%%%%%%%%%%%%%%%%% Error! Bookmark not defined.](#_Toc502133140)

## **WHOI READ SENSOR DATA**

### function X = d3readswv(recdir,prefix,df)

% X = d3readswv(recdir,prefix,df)

% Reads a sequence of D3 format SWV (sensor wav) sensor files

% and assembles a continuous sensor sequence in x.

% Calls d3parseswv to read in each file.

%

% Returns:

% X is a structure containing:

% x: a cell array of sensor vectors. There are as many

% cells in x as there are unique sensor channels in the

% recording. Each cell may have a different length vector

% according to the sampling rate of the sensor channel.

% fs: a vector of sampling rates. Each entry in fs is the

% sampling rate in Hz of the corresponding cell in x.

% cn: a vector of channel id numbers corresponding to

% the cells in x. Use d3channames to get the name and

% description of each channel.

%

% markjohnson@st-andrews.ac.uk

% Licensed as GPL, 2012

## **UM READ SENSOR DATA**

### function TagData = readSwv(TagData, recdir)

prefix = TagData.deployName;

RawVolt = d3readswv(recdir,prefix,TagData.desampFreq);

% [ch\_names,descr,ch\_nums,type] = d3channames(X.cn);

% Register the deployment:

[Calib,Deploy] = d3deployment(recdir, prefix, prefix) ;

depthOrig = d3calpressure(RawVolt, Calib);

accelTagOrig = d3calacc(RawVolt,Calib);

magTagOrig = d3calmag(RawVolt,Calib);

depthOrig(isnan(accelTagOrig)) = nan;

TagData.RawVolt = RawVolt;

TagData.CalibOrig = Calib;

TagData.accelTagOrig = accelTagOrig;

TagData.depthOrig = depthOrig;

TagData.magTagOrig = magTagOrig;

end

**Gives:**

*RawVolt: [1x1 struct]*

*CalibOrig: [1x1 struct]*

*accelTagOrig: [n x3 double]*

*depthOrig: [n x1 double]*

*magTagOrig: [n x3 double]*

## **WHOI CALIBRATION FUNCTIONS**

### function [A,CAL,fs] = d3calacc(X,CAL,test,mindepth,ch)

% [A,CAL,fs] = d3calacc(X,CAL,test,mindepth,ch)

% Automatically performs a calibration sequence on the

% accelerometer data in raw sensor structure X. If no test

% is specified, a calibration using the current settings is made.

% If test = 'full', all of the following tests are performed in

% sequence. Otherwise, specify one of the following in test:

% 'bias' remove bias

% 'p' compensate for pressure effects

% 't' compensate for temperature effects

% 'sens' adjust sensitivity of each axis

% 'cross' compensate for cross-axis coupling

% Optional argument mindepth is used to exclude data at depths<mindepth

% from automatic calibration. The output vector will still include this

% data but the calibration will be performed only on the deeper data.

% Optional argument ch is used to identify the calibration

% field of the correct accelerometer if there are multiple triaxial

% accelerometers in the sensor set. For example, to calibrate

% the gyro accelerometers, put ch='GACC'.

%

% mark johnson

% markjohnson@st-andrews.ac.uk

% last modified: July 2012

### function [p,CAL,fs] = d3calpressure(X,CAL,test)

% [p,CAL,fs] = d3calpressure(X,CAL,[test])

% Apply calibration constants to the raw pressure signal in

% sensor matrix s. CAL is a structure of calibration constants

% from a cal file (e.g., tag210.m or sw05\_199a.m).

% test can be 'full', 'lowbat', 'tempr' or 'none' (the default is none).

% A full test performs low battery compensation and characterizes

% the temperature effect and the 0-pressure offset.

% If a 'full extended' test is requested, a wider range of pressures

% are analyzed for temperature effects. This is useful for data sets

% with a wide range of temperature values.

% If a 'full notemp' test is requested, no change is made to the current

% temperature coefficient.

%

% Pressure result p is in m H20 (salt) per degree Celsius. Temperature

% result tempr is in degrees Celsius.

%

% Constants fields used are CAL.TCAL, CAL.PCAL, and CAL.PTC.

% If r is the raw pressure reading and t is the temperature,

% p = a\*r^2 + b\*r + c + tc\*t for a 2nd order calibration.

%

% mark johnson

% majohnson@whoi.edu

% last modified: 29 August 2008

### function [M,CAL,fs,Mz] = d3calmag(X,CAL,test,mindepth)

% [M,CAL,fsm,Mz] = d3calmag(X,CAL,test,mindepth)

% Automatically performs a calibration sequence on the raw

% magnetometer data in raw sensor structure X. If no test

% is specified, a calibration using the current settings is made.

% If test = 'full', all of the following tests are performed in

% sequence. Otherwise, specify one of the following in test:

% 'hard' remove bias

% 't' compensate for temperature effects

% 'p' compensate for pressure effects

% 'soft' compensate for cross-axis coupling

% 'sens' adjust sensitivity of each axis

% Optional argument mindepth is used to exclude data at depths<mindepth

% from automatic calibration. The output vector will still include this

% data but the calibration will be performed only on the deeper data.

%

% mark johnson

% markjohnson@st-andrews.ac.uk

% last modified: July 2012

## **UM CALIBRATION FUNCTIONS**

### function TagData = optCalib(TagData)

satisCalib = 0;

while ~satisCalib

TagData = reoptCalib(TagData);

correctKey = 0;

while ~correctKey

keyPress = input('Satis for the calib? y, n ', 's');

switch keyPress(1)

case 'y'

satisCalib = 1;

correctKey = 1;

case 'n'

correctKey = 1;

otherwise

fprintf('Please type y, n')

end

end

end

end

function TagData = reoptCalib(TagData)

RawVolt = TagData.RawVolt;

Calib = TagData.CalibOrig;

prefix = TagData.deployName;

%% Optimize the Calibration

% Optimize the pressure calibration:

% X = trPress(X);

[depth,Calib,sampleFreq] = d3calpressure(RawVolt,Calib,'full');

% Optimize the acceleration calibration:

% min\_depth = 10;

[accelTag,Calib,sampleFreq] = d3calacc(RawVolt,Calib,'full');

% Optimize the magnetometer calibration:

[magTag,Calib,sampleFreq] = d3calmag(RawVolt,Calib,'full');

% Save the calibration information so far to the deployment CAL file:

d3savecal(prefix,'CAL',Calib) % bug needs to be fixed

timeHour = (1:length(accelTag))'/sampleFreq/3600;

timeSec = timeHour\*3600;

depth(isnan(accelTag)) = nan;

TagData.accelTag = accelTag;

TagData.depth = depth;

TagData.magTag = magTag;

TagData.timeHour = timeHour;

TagData.timeSec = timeSec;

TagData.sampleFreq = sampleFreq;

TagData.Calib = Calib;

end

**Gives:**

(from read swv) :

*RawVolt: [1x1 struct]*

*CalibOrig: [1x1 struct]*

*accelTagOrig: [151278x3 double]*

*depthOrig: [151278x1 double]*

*magTagOrig: [151278x3 double]*

*dataLength: 151278*

(from optCalib):

*accelTag: [151278x3 double] >>> accel after calibration*

*depth: [151278x1 double] >>> depth after calibration*

*magTag: [151278x3 double] >>> mag after calibration*

*timeHour: [151278x1 double]*

*timeSec: [151278x1 double]*

*sampleFreq: 20*

*Calib: [1x1 struct]*

### function TagData = estOrient(TagData, orientTab)

%% If want to continue to save all the data with p,r,h estimation

% accelTag = TagData(nTagData).accelTag;

% recdir = 'M:\DQ\_All\_Day\_Data';

prefix = TagData.deployName;

df = TagData.desampFreq;

% % sampleFreq = TagData(nTagData).sampleFreq;

isDeg = find(orientTab(:, 3:5) > pi/2);

if isDeg

TagData.orientTabDeg = orientTab;

orientTab = deg2rad(orientTab);

else

TagData.orientTabRad = orientTab;

end

% [accelAnim, magAnim] = tag2whale(accelTag,magTag,OrientTab,sampleFreq);

% All\_Plot(p, Aw, Mw, fs)

d3savecal(prefix,'OTAB',orientTab);

% d3makeprhfile(recdir,prefix,deploy\_name,df)

TagData = d3makeprhfile\_simple(prefix,df, TagData);

% TagData = d3makeprhfile\_mod(recdir,prefix,prefix,df, TagData, nTagData);

end

**Gives:**

(from read swv) :

*RawVolt: [1x1 struct]*

*CalibOrig: [1x1 struct]*

*accelTagOrig: [151278x3 double]*

*depthOrig: [151278x1 double]*

*magTagOrig: [151278x3 double]*

*dataLength: 151278*

(from optCalib):

*accelTag: [151278x3 double] >>> accel after calibration*

*depth: [151278x1 double] >>> depth after calibration*

*magTag: [151278x3 double] >>> mag after calibration*

*timeHour: [151278x1 double]*

*timeSec: [151278x1 double]*

*sampleFreq: 20*

*Calib: [1x1 struct]*

(from estOrient):

*orientTabRad: [0 0 0 0 0]*

*pitchDeg: [151278x1 double] Pitch*

*rollDeg: [151278x1 double] Roll*

*headDeg: [151278x1 double] Heading*

*accelAnim: [151278x3 double] Accel on animal axes (will be same as accelTag unless rotated (post OTAB)*

*magAnim: [151278x3 double] Mag on animal axes (will be same as accelTag unless rotated (post OTAB)*

*sampleFreqtag: 20 >>> desampled Hz*

### TagData = orderfields(TagData);

(from orderfields):

*I think thus rearranges the final TagData structure ….*

*TagData =*

*Calib: [1x1 struct]*

*CalibOrig: [1x1 struct]*

*RawVolt: [1x1 struct]*

*accelAnim: [151278x3 double]*

*accelTag: [151278x3 double]*

*accelTagOrig: [151278x3 double]*

*dataLength: 151278*

*deployName: 'tt17082'*

*depth: [151278x1 double]*

*depthOrig: [151278x1 double]*

*desampFreq: 10*

*headDeg: [151278x1 double]*

*magAnim: [151278x3 double]*

*magTag: [151278x3 double]*

*magTagOrig: [151278x3 double]*

*orientTabRad: [0 0 0 0 0]*

*pitchDeg: [151278x1 double]*

*rollDeg: [151278x1 double]*

*sampleFreq: 20*

*sampleFreqtag: 20*

*timeHour: [151278x1 double]*

*timeSec: [151278x1 double]*

### function TagData = shiftDepth(TagData, method, ENABLE\_PLOT)

if ~exist('ENABLE\_PLOT', 'var') || isempty(ENABLE\_PLOT)

ENABLE\_PLOT = 0;

end

accelTag = TagData.accelTag;

depth = TagData.depth;

timeHour = TagData.timeHour;

depth(isnan(accelTag(:, 1))) = nan;

depth = -depth;

% DataPlot\_Feb1(t\_h, p\_trc, 'dep');

THLD\_RUB = 0.05;

[timeHourPeak, depthPeak, indPeak] = getPeaks(timeHour, depth, THLD\_RUB, 0, 0);

THLD\_LARGE\_DESC = -0.02;

THLD\_LARGE\_ASC = 0.02;

if strcmpi(method, 'interp')

for i = 1:3

largeDesc = diff([0; depthPeak]) < THLD\_LARGE\_DESC;

largeAsc = diff([depthPeak; 0]) > THLD\_LARGE\_ASC;

isPeak = ~(largeDesc | largeAsc);

depthPeak = depthPeak(isPeak);

indPeak = indPeak(isPeak);

timeHourPeak = timeHourPeak(isPeak);

end

shiftDepthMeter = interp1(timeHourPeak, depthPeak, timeHour, 'pchip','extrap');

elseif strcmpi(method, 'max')

shiftDepthMeter = nan(numel(depth), 1);

dep = [depthPeak(1); depthPeak; depthPeak(end)];

indPeak = [1; indPeak; length(timeHour)];

ARVG\_WIN = 10;

shiftDepthMeter(1:indPeak(1+floor(ARVG\_WIN/2))) = max(dep(1:1+floor(ARVG\_WIN/2)));

for i = 1+floor(ARVG\_WIN/2):length(indPeak)-floor(ARVG\_WIN/2)

shiftDepthMeter(indPeak(i):indPeak(i+1)) = max(dep(i-floor(ARVG\_WIN/2):i+floor(ARVG\_WIN/2)));

end

shiftDepthMeter(indPeak(i+1):end) = max(dep(length(indPeak)-floor(ARVG\_WIN/2)+1:end));

end

if ENABLE\_PLOT

figure; plot(timeHour, depth); hold on; plot(timeHourPeak, depthPeak, 'r.')

plot(timeHour, shiftDepthMeter, 'c.')% figure; plot(t\_h, p\_trc); hold on; plot(t\_h, depthShift, 'c.--')

end

depthShift = depth-shiftDepthMeter;

TagData.depthShift = depthShift;

end

*TagData =*

*Calib: [1x1 struct]*

*CalibOrig: [1x1 struct]*

*RawVolt: [1x1 struct]*

*accelAnim: [151278x3 double]*

*accelTag: [151278x3 double]*

*accelTagOrig: [151278x3 double]*

*dataLength: 151278*

*deployName: 'tt17082'*

*depth: [151278x1 double]*

*depthOrig: [151278x1 double]*

*desampFreq: 10*

*headDeg: [151278x1 double]*

*magAnim: [151278x3 double]*

*magTag: [151278x3 double]*

*magTagOrig: [151278x3 double]*

*orientTabRad: [0 0 0 0 0]*

*pitchDeg: [151278x1 double]*

*rollDeg: [151278x1 double]*

*sampleFreq: 20*

*sampleFreqtag: 20*

*timeHour: [151278x1 double]*

*timeSec: [151278x1 double]*

*TagData =*

*Calib: [1x1 struct]*

*CalibOrig: [1x1 struct]*

*RawVolt: [1x1 struct]*

*accelAnim: [151278x3 double] >> Animal-based accel*

*accelTag: [151278x3 double]*

*accelTagOrig: [151278x3 double]*

*dataLength: 151278*

*deployName: 'tt17082'*

*depth: [151278x1 double]*

*depthOrig: [151278x1 double]*

*desampFreq: 10*

*headDeg: [151278x1 double]*

*magAnim: [151278x3 double] >> Animal-based mag*

*magTag: [151278x3 double]*

*magTagOrig: [151278x3 double]*

*orientTabRad: [0 0 0 0 0]*

*pitchDeg: [151278x1 double] >> Animal pitch*

*rollDeg: [151278x1 double] >> Animal roll*

*sampleFreq: 20*

*sampleFreqtag: 20*

*timeHour: [151278x1 double]*

*timeSec: [151278x1 double]*

*depthShift: [151278x1 double] >> depth corrected to 0*

## **Add Metadata to Structure**

### TagData.localdeployinfo = DeployInfoLoc;

DeployInfoLoc.Deploy = datestr([2017,3,23,14,22,05]);

DeployInfoLoc.OnAnimal = datestr([2017,3,23,13,18,52]);

DeployInfoLoc.OffAnimal = datestr([2017,3,23,16,22,32]);

DeployInfoLoc.ShowStart = 'NA';%datestr([2016,3,22,10,11,31]);

DeployInfoLoc.ShowEnd = 'NA';%datestr([2016,3,22,10,11,31]);

DeployInfoLoc.VideoFile = '1'; %1;

DeployInfoLoc.Dolphin = 'Spree';

DeployInfoLoc.Group = 'SG';

DeployInfoLoc.Pool = 'SH+Med';

## **Truncate Relevant Sensor Data to Animal-Only Portion**

This isn’t a function – it’s just a portion of the script that truncates the track to the part of the deployment when the tag is on the animal only. See below:

TagData =

Calib: [1x1 struct]

CalibOrig: [1x1 struct]

RawVolt: [1x1 struct]

accelAnim: [151278x3 double]

accelTag: [151278x3 double]

accelTagOrig: [151278x3 double]

dataLength: 151278

deployName: 'tt17082'

depth: [151278x1 double]

depthOrig: [151278x1 double]

desampFreq: 10

headDeg: [151278x1 double]

magAnim: [151278x3 double]

magTag: [151278x3 double]

magTagOrig: [151278x3 double]

orientTabRad: [0 0 0 0 0]

pitchDeg: [151278x1 double]

rollDeg: [151278x1 double]

sampleFreq: 20

sampleFreqtag: 20

timeHour: [151278x1 double]

timeSec: [151278x1 double]

depthShift: [151278x1 double]

localdeployinfo: [1x1 struct]

depth2A: [92524x1 double] >> TRUNCATED

acc2A: [92524x3 double] >> TRUNCATED

head2A: [92524x1 double] >> TRUNCATED

mag2A: [92524x3 double] >> TRUNCATED

pitch2A: [92524x1 double] >> TRUNCATED

roll2A: [92524x1 double] >> TRUNCATED

Hour2A: [92524x1 double] >> TRUNCATED

Sec2An: [92524x1 double] >> TRUNCATED

## **Calculate Summary Statistics**

% NOTE: accelerometer units are already in "g" from D3 codes used

% NOTE: For placement on dolphins, Surge is Y axis (1st column); Sway is X

% axis (2nd column); Heave is Z axis (3rd column)

### **Magnitude (Q): as a 4th dimension.**

Q = sqrt(heave^2 + surge^2 + sway^2) >> see Nathan et al 2012......

(Q is the length of the diagonal of the x–y–z volume involved)

a = TagData.acc2A(:,3).^2;

b = TagData.acc2A(:,1).^2;

c = TagData.acc2A(:,2).^2;

staticQ = sqrt(a+b+c);

clear a b c

### **Acceleration Statistics**

#### **Static Acceleration:**

Used a 2 second running mean of raw acceleration to calculate static acceleration (gravitational acc)

This is calculated with dynamic acceleration and odba and the output can be found in the AccelStat structure.

#### **Dynamic Acceleration:**

Subtracted static acc from raw acc

This is calculated with satic acceleration and odba and the output can be found in the AccelStat structure.

#### **ODBA: Overall Dynamic Body Acceleration**

odbaWilson = abs(dynAccel(:,1)) + abs(dynAccel(:,2)) + abs(dynAccel(:,3));

ODBA\_METHOD = 'Wilson'; % Choose Which ODBA Method

TimeWindow = 2; % Choose time interval in seconds >> Need to justify and confirm this time interval (see Pagano et al 2017)

### **AccelStat Function:**

function AccelStat = getOdba2(TagData, ODBA\_METHOD, TimeWindow, Q) % UM code to calc accel stats (static, dynamic, odba)

if ~exist('ODBA\_METHOD', 'var') || isempty(ODBA\_METHOD)

ODBA\_METHOD = 'total';

if nargin < 1

help getSegNew

end

end

%%

if nargin < 1

help getOdba;

end

%%

AccelStat = [];

%%

accelTagOrig = TagData.accelTag;

sampleFreq = TagData.sampleFreq;

timeHour = TagData.timeHour;

totalAccel = accelTagOrig; % accel for averaging

AVRG\_WINDOW = sampleFreq\*TimeWindow; How does this make seconds?

**% get mean static/dynamic acc**

staticAccel = lowpassFilt(totalAccel, 'moveAvrg' , AVRG\_WINDOW);

dynAccel = totalAccel - staticAccel;

**% get max and std dynamic acc**

maxbaY = slidefun(@max, AVRG\_WINDOW, dynAccel(:,1)) ;

maxbaX = slidefun(@max, AVRG\_WINDOW, dynAccel(:,2)) ;

maxbaZ = slidefun(@max, AVRG\_WINDOW, dynAccel(:,3)) ;

maxQ = slidefun(@max, AVRG\_WINDOW, Q);

stdbaY = slidefun(@std, AVRG\_WINDOW, dynAccel(:,1));

stdbaX = slidefun(@std, AVRG\_WINDOW, dynAccel(:,2));

stdbaZ = slidefun(@std, AVRG\_WINDOW, dynAccel(:,3));

stdQ = slidefun(@std, AVRG\_WINDOW, Q);

**% get ODBA**

if strcmpi(ODBA\_METHOD, 'wilson')

odba = abs(dynAccel(:,1)) + abs(dynAccel(:,2)) + abs(dynAccel(:,3));

elseif strcmpi(ODBA\_METHOD, 'norm')

odba = sqrt(sum(dynAccel.^2, 2)); % 0.11

elseif strcmpi(ODBA\_METHOD, 'total')

odba = abs(sqrt(sum(totalAccel.^2, 2)) - 1); % 0.4

end

%%

AccelStat.totalAccel = totalAccel;

AccelStat.staticAccel = staticAccel;

AccelStat.dynAccel = dynAccel;

AccelStat.maxDBA = [maxbaY,maxbaX,maxbaZ];

AccelStat.stdDBA = [stdbaY, stdbaX, stdbaZ];

AccelStat.odba = odba;

AccelStat.maxQ = maxQ;

AccelStat.stdQ = stdQ;

end

AccelStat =

totalAccel: [151278x3 double]

staticAccel: [151278x3 double]

dynAccel: [151278x3 double]

maxDBA: [151278x3 double]

stdDBA: [151278x3 double]

odba: [151278x1 double]

maxQ: [92524x1 double]

stdQ: [92524x1 double]