**Main paper:** Gundog.Tracks - [10.1186/s40317-021-00245-z](https://animalbiotelemetry.biomedcentral.com/articles/10.1186/s40317-021-00245-z)

**Additional file 1**

**Text SI** – Device set up and capture protocol **[Excluded here]**

**Text S2** – The importance of having the correct coordinate system and axis alignment

It is essential that tri-axial accelerometer and magnetometer readings are aligned to a designated tag coordinate system for accurate delineation of 3-D rotation with respect to the fixed Earth’s surface. The equations and R source code detailing the tilt-compensated method (*cf.* Text S3) assumes the aerospace (x-North, y-East, z-Down) coordinate system, or ‘NED’ (*cf.* Text S2: Fig. S1). When lying flat and upright, the z-axis reads approximately 1 *g* (1 *g* = 9.81 m/s) and the x- and y-axes are near 0 *g,* inversing the device 180o negates the z-axis reading. Pointing the device 90o downward and then upward along the x- and y-axis results in readings near 1 *g* and –1 *g*, respectively. This is the reason why x-axis measurements are negated (essentially multiplied by -1) in the derivation of pitch (E14 – main text), because in the NED system, a downward inversion of the x-axis results in a positive increase in measured acceleration. Assuming the magnetometer and accelerometer channels are in alignment, the x-channel magnetometer reading is at a maximum and minimum when the (whole) device (positioned flat) is pointed North and South, respectively. The y-channel is at a maximum when the device is rotated to point West and reads a minimum when the device is pointed East, with the z-channel reading remaining relatively constant. Conversely, the y-axis will read a maximum when the device is pointed East following a 180o inversion, with the z-axis reading (though still constant) being lower than the upright condition (in the Northern hemisphere), because the channel is aligned against the downward geomagnetic field vector. Using this local frame, we assume that the x, y and z axes of both the accelerometer and magnetometer sensors are aligned to the anterior-posterior (surge), medio-lateral (sway) and dorsal-ventral (heave) axes of a given animal, respectively.

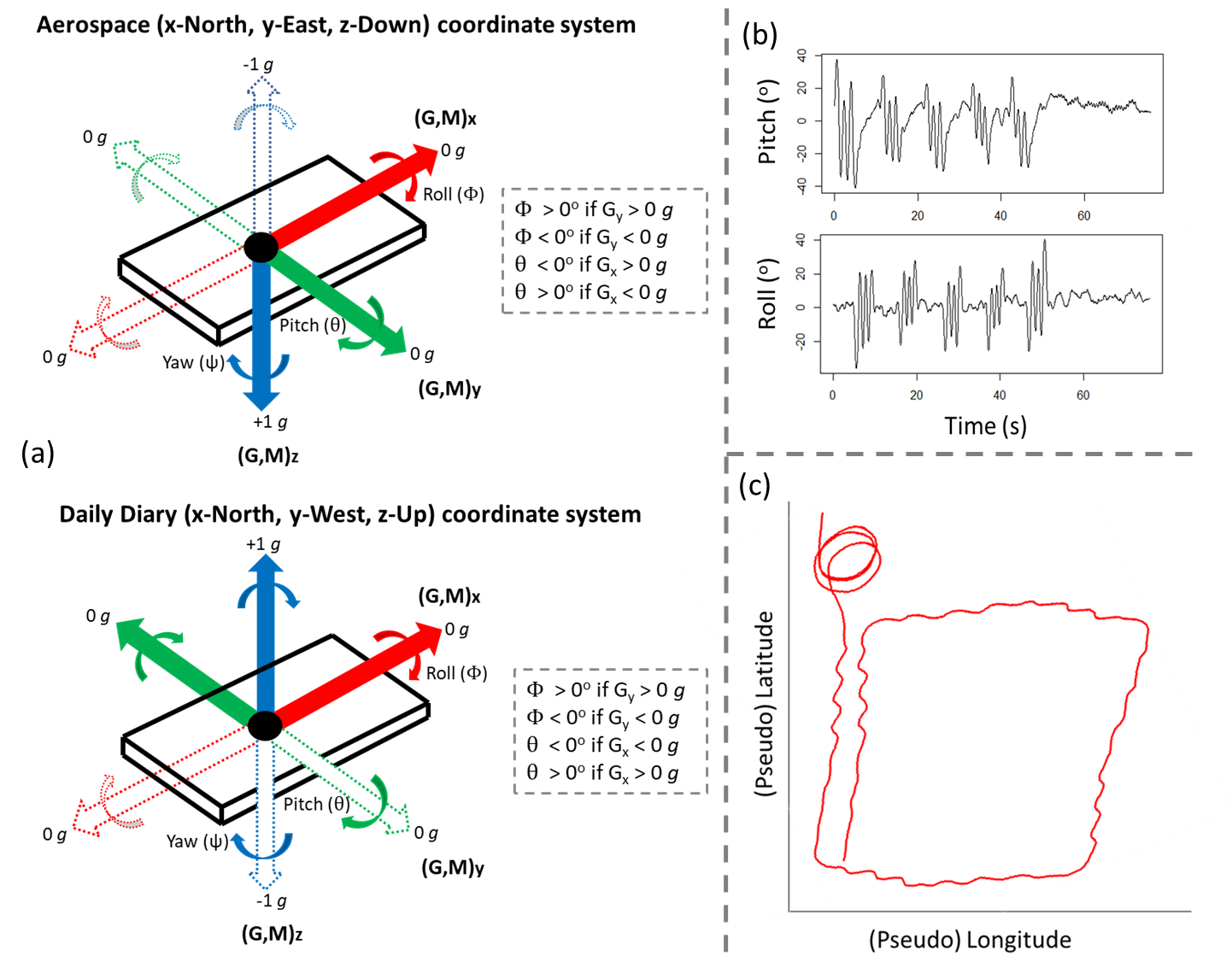
Not all devices have an orthogonal set up with respect to the NED local frame, whilst discrepancies between deployment position may also change the local frame with respect to the earth’s vector. Transforming between coordinate frames requires 3 x 3 rotation matrices, with the unique combination dependent on the coordinate frame of your device and the coordinate frame you are interested in [cf. 1]. This can become tricky however, since the direction of measured acceleration (with respect to the Earth's downward gravitational field) can differ between devices with similar orthogonal layout. For example, the Daily Diary (DD) units [cf. 2] deployed on animals used in this study measure +1 *g* when lying flat and upright, similar to the NED system (Text S2: Fig. S1), despite an upward direction of measurement (NED being downward). Consequently, there is no ‘set’ rotation matrix for a given local coordinate frame transformation, whilst depending on board layout, magnetometry channels may require a variant matrix to the one used for acceleration readings. Put simply though, due to the layout of perpendicular (orthogonal) channels, any required correction just involves swapping and/or negating the measurements from your accelerometer and/or magnetometer channels until correctly aligned with the NED coordinate system [cf. 3].

As a case in point using the above tilt-compensated method for DD’s (cf. Text S2: Fig. S1), the y- and z- axis of the magnetometer are required to be negated, since they are measuring magnetism in the opposite direction to that governed by the NED system. As for acceleration, despite the y-axis being opposite in direction from that of NED local frame, raising the left side of the device results in a positive increase of measured acceleration, which transcends to a positive roll (cf.E15 – main text). This is the same as the NED coordinate system, because of the point alluded to earlier whereby *g* is positive towards the downward component of earth vector for NED but negative for the DD axes configuration. If we picture a seesaw effect, as the DD y-axis tilts up, the corresponding NED y-axis tilts down, but both are measuring increasing *g* and thus depicted as rolling right*.* As such the y-axis does not need to be negated and in a similar manner, the z-axis also remains the same, but the x-axis does require negation, since a downward tilt (pitch) measures negative *g* for the DD configuration and positive *g* for the NED system. If a device was positioned on an animal perpendicular to ‘normal’ then as well as deciding channel negation, certain channels will be required to be swapped. Again, in the case of the DD, z-axis values are required to be exchanged with x-axis values (of both the accelerometer and magnetometer) following a clockwise 90o inversion about the y-axis (e.g., x-axis now points upwards and thus represents the ‘heave’ plane (originally reflected by z-axis values)). In this instance, the z-axis (now representing the (inverse) surge plane and swapped with x-axis values) of the magnetometer requires negating, due to a 180o inversion, though the equivalent accelerometer channel does not (because a negative increase in measured *g* now corresponds to a positive pitch - same as the NED system). Lastly, the x-axis (now containing z-axis values) and y-axis values of the accelerometer and magnetometer follow the same protocol as the ‘original orientation’ described above.

Judicious choice of channel configuration is required for correct output of pitch, roll and yaw, though once this is obtained, the user can save the channel configuration for all future uses at this given orientation. Note, however, animals that travel with postures greater than perpendicular from one another (e.g., a penguin walking through a colony *vs* swimming in the ocean or a pine martin bounding across the forest floor *vs* climbing a tree) would require variant axis alignments according to mode of movement (when using Euler angles). Assuming a tag can be deployed at any perpendicular orientation, there are 24 possible channel local coordinate frame configurations.

To aid the above process, we recommend alongside the magnetic calibration, the user undertakes a configuration calibration, in that the device is slowly pitched up and down and rolled left and right three times at each cardinal direction, starting and ending at North, before spinning the device slowly in a circle clockwise three times, again, starting and ending at North. Note, during this procedure, the ‘default’ device orientation should be equivalent to that when deployed on the animal (during ‘normal’ traversing posture). The user can inspect results obtained during this calibration period to ensure the sequence of pitch and roll angles follow the intended direction of rotation (seeSI2. Fig. 1b). Further, the shape of this calibration period when dead-reckoned (using a constant progression (speed) value) should produce an approximate square shape followed by three circles, finishing at North, (seeSI2. Fig. 1c). Should these diagnostic plots not show what is expected, then the accelerometer and/or magnetometer alignment is wrong with respect to the coordinate system assumed within the tilt compensated compass method outlined in SI3.

*Figure S1. 1. The aerospace north-east-down (NED) coordinate system (top) and the Daily diary north-west-up (NWU) coordinate system (bottom) (a). Both systems assume the x (red), y (green) and z (blue) channels of the magnetometer (M) and accelerometer (G = gravitational component) are in alignment. Dotted arrows reflect a 180o inversion of each channel. Note the difference in the direction of measured acceleration (-1 g/ +1 g) between the two coordinate systems when pointing downward in relation to the earth’s gravitational vector. A clockwise rotation about the y-axis results in a positive inversion of the x-axis (pitch ()) and a clockwise rotation about the x-axis results in a positive inversion of the y-axis (roll ()), though note, for the NED coordinate system, x-axis values are required to be negated within the computation of so that a positive increase in measured acceleration results in a decreasing . Evaluation of whether a coordinate system is correctly aligned to that required in the tilt-compensated compass method can be shown by plotting computed pitch (b), roll (b) and yaw (the latter being assessed as a dead-reckoned track (c)), from data collected during a configuration calibration (outlined in the text above).*



**Text S3**– Magnetometer calibration, rotation correction and deriving yaw (heading)

Device calibration should be performed so that magnetic distortions to the data can be corrected post-hoc. It is recommended that the device is rotated slowly, ideally in an open space, away from potential sources of magnetic disturbance, relative to the Earth’s magnetic field. Each orientation of roll, pitch and yaw should be incorporated in the device rotations (simply put, imagine a pen is attached to the end of the device being rotated and the aim is to ‘colour in’ all parts of a sphere). This section of data can then be used as a reference for the vectorial sum of magnetometry data across all three spatial dimensions, from which ‘hard’ and ‘soft iron’ errors which can occur in magnetometry data can be corrected. Note, only data during the calibration procedure should be used to deduce soft and hard iron offsets for each magnetometer channel, which is subsequently applied to all magnetometer readings. Hard iron deposits (e.g., ferrous materials or magnets) introduce a constant additive bias to the earth’s magnetic field and thus magnetometer readings, whereas soft iron distortions (non-magnetic materials that alter the magnetic field, e.g., nickel) are caused by variations in the surrounding magnetic field. A tri-axial magnetic field intensity scatterplot of the calibration period shows such distortion (cf. SI3. Fig. 1), with hard iron deposits causing the sphere to shift from origin and soft iron deposits, stretching the sphere into an elliptical shape. This function corrects magnetometry data for soft and hard iron distortion and performs rotation correction of the magnetic and gravity vectors (assuming they are in alignment) if the user inputs Euler angle offsets (of the tag relative to the animal’s body frame), prior to computing heading.

Gundog.compass = **function**(mag.x, mag.y, mag.z, acc.x, acc.y, acc.z, ME, pitch.offset = 0, roll.offset = 0, yaw.offset = 0, method = 3, plot=TRUE)

The function inputs given below follow in the order:

* Raw tri-axial magnetometer data (mag.x,y,z)
* Tri-axial **static** acceleration data (acc.x,y,z) (for computation of pitch and roll)
* Marked events data (ME) specifying the period of the magnetic calibration period (as denoted by ‘M’ – any other input signifies data acquisition not part of calibration procedure).
* Pitch, roll and/or yaw device offsets relative to the animal’s body frame (default values are zero)
* method
  + method = 1, is the computationally cheaper method, using scale biases with simple orthogonal rescaling (avoiding matrices altogether), outlined by Winer [4], but may prove effective when data is less affected by noise – e.g., nearly spherical. Within this method, hard iron offsets are resolved by obtaining the average recorded magnetism bias from each channel, specifically by dividing the summed maximum and minimum values of raw magnetism by two. Soft iron scale factors are used to rescale and thus equalise the magnetometer data along the three measurement axes. Here, the average maximum chord length is calculated from the ratio of the average maximum – minimum values of each axis by the summed equivalent of all axes. This is then multiplied to all data, subsequent to hard iron distortions being removed.
  + method = 2 and method = 3 essentially compute gains and cross-axis gains (from derived eigen- values and vectors, respectively), by performing a sphere (ellipsoid) fitting. This involves centring a sphere (or ellipsoid) to (0,0,0) origin, de-rotating by multiplying by the inverse of the rotation matrix and scaling appropriately *via* multiplying by the inverse of the gains. For rotated ellipsoids use method = 3 and for near spherical data and/or influenced by heavy noise, use method = 2. For, non-rotated ellipsoids use method = 4, when x and y channel radii the same, use method = 5, when x and z are the same, use method = 6, when y and z are the same, use method = 7 and for spherical data use method = 1, 2 or 8. Essentially, there up to eight variant methods included (default is method = 3). For a detailed review of the underlying mathematics of these methods, see [9]. Inspection of summary plots can aid in demonstrating the ‘best’ method to use.
* Plot = TRUE (default) compares 2-D magnetic field intensity scatterplots (mag.x - mag.y, mag.x - mag.z and mag.z - mag.y) of the calibration period both pre- and post-calibration from one of the user-defined methods dictated above (see SI3. Fig. 1).

Note, within the function itself, Bx,y,z refers to (uncorrected) magnetism data during the calibration period (which should be coded for by ‘M’ in ME) and Mx,y,z refers to all (uncorrected) magnetism data (incl. calibration data). NMbx,y,z are the normalised channels expressed in the animal’s body frame after calibration and NMbfx,y,z are the calibrated, normalised data expressed in the animal’s body frame, after tilt-correction. NGbx,y,z are the normalised static acceleration data expressed in the animal’s body frame after calibration. Note the above notations stand when no rotation correction is performed (e.g., no Euler angle offsets supplied), since the magnetic and gravity vectors are assumed to aligned with the animal’s body-carried NED frame.

The function returns a data frame containing columns in the order:

* Normalised tri-axial **static** acceleration data expressed in the animal’s body frame (NGbx,y,z).
* Calibrated tri-axial magnetometry data (Mx,y,z)
* Calibrated, normalised tri-axial magnetometry data expressed in the animal’s body frame (NMbx,y,z)
* Calibrated, normalised tri-axial magnetometry data expressed in the animal’s body frame, after tilt-correction (NMbfx,y,z)
* Marked events (ME)
* Pitch
* Roll
* Yaw

Chart, bubble chart

Description automatically generatedSee Text S2 for relevance pertaining to the order of variable input. This function assumes the NED coordinate frame. Changing the order of variable input/channel negation may be required. To avoid confusion, within the function, assume the x, y and z input fields for both magnetometry and accelerometery data represents the surge, sway heave dimensions of orientation, respectively, which the user allocates the appropriate channel and/or required negation to (dependent on the local coordinate frame of the device used).

**See *Gundog.Compass.R* file for annotated function.**

Figure S2. 1. Output summary plots from Gundog.Compass() (when plot = TRUE), showing 2-D dimensions of the xy, xz and zy magnetometry components during the calibration period (ME = ‘M’), both pre- (red) and post- (blue) correction. The 0,0 origin (centroid) is denoted with a black circle.

**Text S4** – Step counts as a distance estimate

Within the main text, a step count was used as a speed proxy within *Gundog.Tracks* for twelve penguin’s walking trajectories (cf. Fig. 7). There are multiple ways in which this could be performed. In R, if we use a Reference Vector (*RV*) and assume the duration of each stride cycle is coded for with a one (1) and anything else, a zero (0) (from some previous behavioural identifier), then the length (duration) of each uninterrupted RV (matching numerical) sequence can be computed (R1). For *RV* values of 1, this is the time period per step (*TPS*). The unique incremental change of each *RV* number (each time it ‘reappears’ - changes from 1 to 0 or *vice versa*) can be computed (R2:7). For *RV* values of 1, this is the step count (*SC*). Because of the nature of how *SC* is calculated, the rolling difference of *SC* provides an index of step frequency (*SF*) (with the value of 1 indexing ‘another step’; R8). One could then calculate the mean number of steps over a given window length and standardise to number of steps per second by dividing by the time difference (s) between values (*TD*; R9 - e.g., for 40 Hz data, *TD* will be approx. 0.025 s). Multiplying this step frequency by a ‘distance per step’ (*d*) progression value (0.16 m used for penguins; R10) gives an index of speed (m/s), which can be converted to *q* *via* dividing by the earth’s radius (R11).

TPS = **sequence**(**rle**(RV)**$**lengths) (R1)   
x1 = **rle**(RV)**$**values (R2)  
x2 = **rle**(RV)**$**lengths (R3)  
z = **c**() (R4)  
**for**(i **in** **unique**(x1)){ (R5)   
 z[x1**==**i] = 1**:sum**(x1 **==**i)} (R6)  
SC = **rep**(z,x2) (R7)  
SF = **c**(0, **diff**(SC)) (R8)   
SF = **rollapply**(SF, width=w, FUN=mean, fill=0, align = 'right') / **TD** (R9)

d = 0.16 (R10)  
q = (SF **\*** d) **/** 6378137 (R11)

Here*, w* should be replaced with the rolling window of choice (e.g., assuming 40 Hz data and a five second window length desired, replace with 200).

We provide a peak finder function; *Gundog.Peaks()* that locates peaks based on local signal maxima, using a given rolling window, with each candidate peak filtered according to whether it surpassed a threshold height. The function returns a data frame containing the (original) row index position of peaks, their amplitude and periodicity between peaks. To calculate the above outlined speed proxy, the row index position of each peak (this is contained within the column termed ‘*Index’*, from the function output) is required to be inserted within a reference vector (*RV*) the length of all other variables used within the dead-reckoning process (e.g., heading ()) to match step peaks in time with motion sensor data. This is because the function output does not include inter-peak data and so output is sub-sampled from the original data input. This process can be achieved by creating a sequentially increasing numerical vector (*RN*; e.g., analogous to the row number index of a data frame, or element number index of a vector), the length of, in this case, *h* (R12) and pasting the index location, ‘*Index*’ within a vector termed *RV*, based on the matching *Index* and *RN* values (R13).

RN = **rep**(1**:length**(h)) (R12)  
RV = **as.integer**(**sub**("\\.$", "", RN) **%in%** Index) (R13)

Note, for column operations, replace ‘length’ with ‘nrow’ (R78).

*Gundog.Peaks* was modelled from the *find\_peaks()* function within *ggpmisc* package [cf. 5], with inputs, function processes and outputs explained below. Note, this function requires the *zoo* package installed and required. Further, this function is only applicable to searching peak maxima, however if peak minima are of interest, we suggest that the user multiplies all values within the spectrum by minus one, subsequent to changing negative values to zero. The user can then use the function in the same way (since minima become maxima, and values originally above zero become zero). The vector sum of acceleration across all three spatial dimensions [cf. 6] is the variable we recommend using.

Gundog.Peaks = **function**(TS, x, thresh = 0, LoM = 5, constant ="med”, ME = 1, plot=TRUE, outlier = FALSE)

The function inputs given below follow in the order:

* TS = timestamp data (POSIX class)
* x = the variable to locate peaks on
* thresh = the minimum height of the candidate peaks (%): size threshold relative to the tallest peak considered (though seeoutlier = TRUE) with constant baseline subtracted
* LoM = span (rolling window length) for local maximum, default is 5, ensure this value is odd (otherwise the function will add 1)
* constant = the constant baseline (y-axis value to be surpassed according to the height threshold) can be user defined (own constant baseline value). The default setting is “med” for median constant. For the mean, “mn” can be input; both are calculated from all values ≥ 0 that have an ME value of one (1)
* ME = marked events (default = 1) specifies which values to locate peaks on (e.g., periods of moving already demarked with a one (1), as opposed to periods to ignore, marked with a zero (0)
* plot = TRUE (default) shows the peak spectrum over time with the identified peaks, height threshold, constant baseline and marked events demarked (cf*.* SI4 Fig. 1)
* outlier = FALSE (default). If changed, then the max value is not used when scaling height threshold (%) but rather, the quantile value input instead of FALSE (do not input TRUE). For example, rather than scaling a 35 % threshold height from the specified constant and the maximum value in the spectrum (with ME of 1), scale it in relation to outlier = 0.99 (0.99 quantile of data (with ME of 1)).

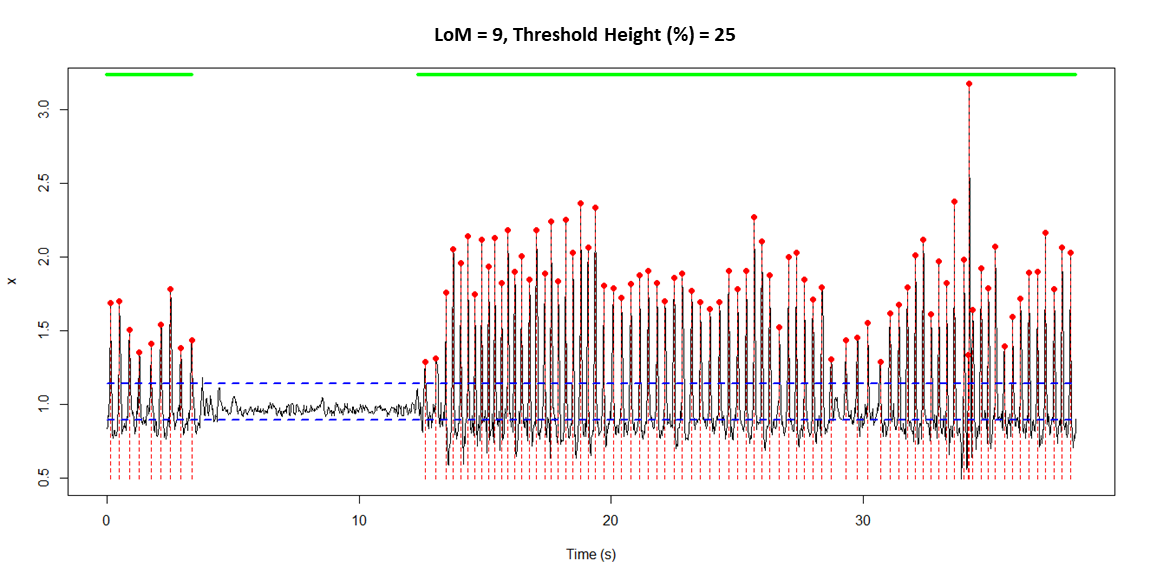


Figure S3. 1. Output plot from Gundog.Peaks() (when plot = TRUE) on a snippet of data (x; vector sum of acceleration from a penguin walking). The green band represents ME values > 0 and each peak is signified with a red circle and dotted vertical line dissecting the peak. The user defined constant and height threshold (%) values are denoted with horizontal dashed blue lines. Here thresh = 25, LoM = 9 (data = approx. 40 Hz) and constant = "med").

The function returns a data frame containing columns in the order:

* Timestamp
* Index (the original row/element position of input data)
* Peak.Amplitude (value of x at peak location)
* Peak.Period (duration between peaks (s))
* Marked.events

**See *Gundog.Peaks.R* file for annotated function.**

**Text S5** – Time Data in R (POSIXct)

In R, to determine accurate time period lengths between values, it is best to save date and time variables together as *POSIX* class (date-time format), so R recognises the character string to be manipulated. If date (*D*) and time (*T*) are within different vectors, simply paste together and convert to a *POSIXct* timestamp object (*TS*) (and specify the time zone (R1).

TS = **as.POSIXct**(**strptime**(**paste**(D, T), format = "%d/%m/%Y %H:%M:%S", (R1)

tz = “GMT”))

The order and type of code used in the format string composed may differ according to the original date and time format registered by the device (e.g. above, we assume date is in the format; ‘06/01/1995’ and time; ‘14:15:30’) [cf. 7].

To convert to a different format, it is best to make the *POSIXct* variable using the format dictated by device output (R3; 1st format string) before converting to a character string using the format you desire (R3; 2nd format string) and then converting this character string back to a *POSIXct* variable (R4). In this example, we assume *D* was in the format, ‘1995/01/06’ and *T*, ’14-15-30’ and we desired the original format stated in (R1);

TS = **paste**(D, T) (R2)   
TS = **format**(**as.POSIXct**(TS, format = "%Y/%m/%d %H-%M-%S"), (R3) "%d/%m/%Y %H:%M:%S")   
TS = **as.POSIXct**(TS, format= "%d/%m/%Y %H:%M:%S") (R4)

For data recorded at infra second frequency, it is vital to include sub-seconds within the *POSIX* object. Below we provide R code (R5:9) using an example where date (*D*), time (*T*) and decimal seconds (*DS*) are stored in separate vectors, with date and time in the format alluded to above and decimal seconds in the format “0.555” (a zero before the decimal present).

DS = **sub**("^(-?)0.", "\\1.", **sprintf**("%.3f", DS)) (R5) T = **paste0**(T, DS) (R6)   
TS = **paste**(D, T) (R7)   
**options**(digits.secs=3) (R8)  
TS = **as.POSIXct**(**strptime**(TS, format = "%d/%m/%Y %H:%M:%OS", tz = “GMT”) (R9)

Here, we remove the zero before the decimal point of *DS*, ensuring three decimal places follow ("%.3f") (R5). *DS* is then appended alongside *T* (no separation) (R6) before *D* and *T* are pasted together (space-separated) (R7). We then specify the number of decimal places of the fractional seconds to express (R8) subsequent to creating the *POSIXct* object ‘*TS*’ (R9) (note, ‘%OS’ refers to decimal seconds). The resultant *TS* object has the format, ‘06/01/1995 14:15:30.555’.

Creating timestamp objects with *POSIX* class enables greater control and manipulation of time data, useful when preparing data to be dead-reckoned. Some useful processes include; correctly time ordering rows of a data frame (*df*) (R10), deleting rows with duplicated timestamp readings (R11), merging two data frames together based on matching time stamps (e.g., VP data and motion sensor data) (R12) and importantly for the dead-reckoning calculation, computing the rolling time difference (*TD*) between data points (R13).

df = df[**order**(df**$**TS), ] (R10)   
df = df[**!duplicated**(df**$**TS), ] (R11)   
new.df = **merge**(df1, df2, by.x = "TS" , by.y = "TS") (R12)

Here user should replace the “TS” input after ‘by.x =’ and ‘by.y =’ with the column names of each of the two data frame’s (df1 & df2) time stamp variables (assuming they have matching time format).

TD = **c**(0, **difftime**(TS, **lag**(TS), units = "secs")[**-**1]) (R13)

In conjunction with the ‘*lubridate*’ package [8] (R14), it becomes easy to add or subtract time (useful if VPs are pre-programmed to record in a different time zone; R15) and to subset a data frame (*df*) between two time periods (TS1 & TS2; R16:19)

Install.packages("lubridate") ; library(lubridate) (R14)   
TS = TS + hours(3) ; TS = TS - minutes(20) ; TS = TS - seconds(55) (R15)   
TS1 = as.POSIXct("2020-08-13 11:15:31.049") (R16)   
TS2 = as.POSIXct("2020-08-19 17:48:11.705") (R17)   
int = interval(TS1, TS2) (R18)   
df = df[df$timestamp %within% int,] (R19)

Lastly, one can easily create a ‘pseudo’ time variable (of a given length, advanced according to a specified time interval) to act as the *TS* variable within *Gundog.Tracks()*. Below (R20), a single datetime *POSIXct* variable is created (though format and time zone may differ according to user choice). This is then advanced by a user-defined time interval (default assumes seconds) – In this example, this is a 0.025 second interval between values (assuming 40 Hz). This incremental time column is advanced to a specified bounded length (here we use a data frame column ‘VeDBA’ to act as the (row-wise) dimension limit.

df**$**TS = **seq.POSIXt**(from = **as.POSIXct**("1995-01-06 05:07:28.05", (R20)

format = "%Y-%m-%d %H:%M:%OS", tz = "GMT"), length.out = **length**(df**$**VeDBA), by = 0.025)

**Text S6** – VPC dead-reckoning – *Gundog.Tracks*

***Gundog.Tracks(TS, h, v, elv = 0******, cs = NULL, ch = NULL, m = 1, c = 0, ME = 1, lo = 0, la = 0, VP.lon = NULL, VP.lat = NULL, VP.ME = FALSE, method = NULL, thresh = 1, dist.step = 1, bound = TRUE, Outgoing = TRUE, plot = FALSE)***

*Gunodg.Tracks()* input and output variables are explained within the main text (Table. 2 & Table. 3). The function outputs written messages to the console to update the user of the progressions of each stage. Important assumptions for variable input include:

* Heading (*h*) (with magnetic calibration and any required declination angle already applied); scale 0o to 360o (both corresponding to true North - or 0o to 35o)
* Either speed (m/s) or a DBA proxy input (*v*). If inputting a DBA proxy, *m* and *c* values should be modulated
* Depending on whether *outgoing* = TRUE or *outgoing* = FALSE (reverse dead-reckoning), user supplies either starting or finishing longitude (*lo*) and latitude (*la*) coordinates, respectively. VP data input as decimal format (e.g., 26.31989, -06.11995)
* **Required packages installed: 'dplyr' & 'zoo'** (these will be checked as dependencies and installed within the function)
* If carrying out VPC, VP data should be synced up alongside motion sensor data within the same length columns/elements. Missing VP data should be replaced with NA or 0.
* Timestamp (*TS*) must be in specific format (*POSIXct* object) (cf*.* SI5). Function calculates time differences between values in seconds, thus infra-second units must be included in *TS* object if data frequency is > 1 Hz. No duplicates are allowed.
* Only supply pitch (*p*) if user wants radial distance modulated according to pitch (cf. E27 - main text).
* No missing data (NA’s) for; *TS, h, v, elv, p, TS, ME, m, c* variables. The variables *ch* and *cs* may have NA’s but observations are carried forward and thus at least the first row/element must contain values (though this can also be supplied as a single value input – analogous to *m* and *c* values)
* User can select a different *dist.step* value (default = 1) to change the stepping interval when computing distance between VPs (within the ‘*method = distance*’ method of VP under-sampling and VP distance moved summary outputs)
* Data from only one animal should be dead-reckoned at a time

The 3-D distance and speed proxies are output using variants of the *disty(method = Hav.SLD)* function which is posited and fully annotated within the subsection of *Gundog.Tracks()* termed; ‘#2) Prepare '*disty*' (distance) and '*beary*' (bearing) functions for VPC and/or summary variables’.

**See *Gundog.Tracks.R* file for annotated function.** Note more information detailing input and output is given at beginning and end of the .R file. See SI6 Fig. 1 for an indication of processing times (on a MSI GP72 7RD Leopard laptop with intel core i7 processor). Notably, system time seems appropriate to dead-reckon weeks-months long datasets at reasonable speeds (e.g., a few mins), though only when dead-reckoning at lower frequencies (e.g., 1 Hz).

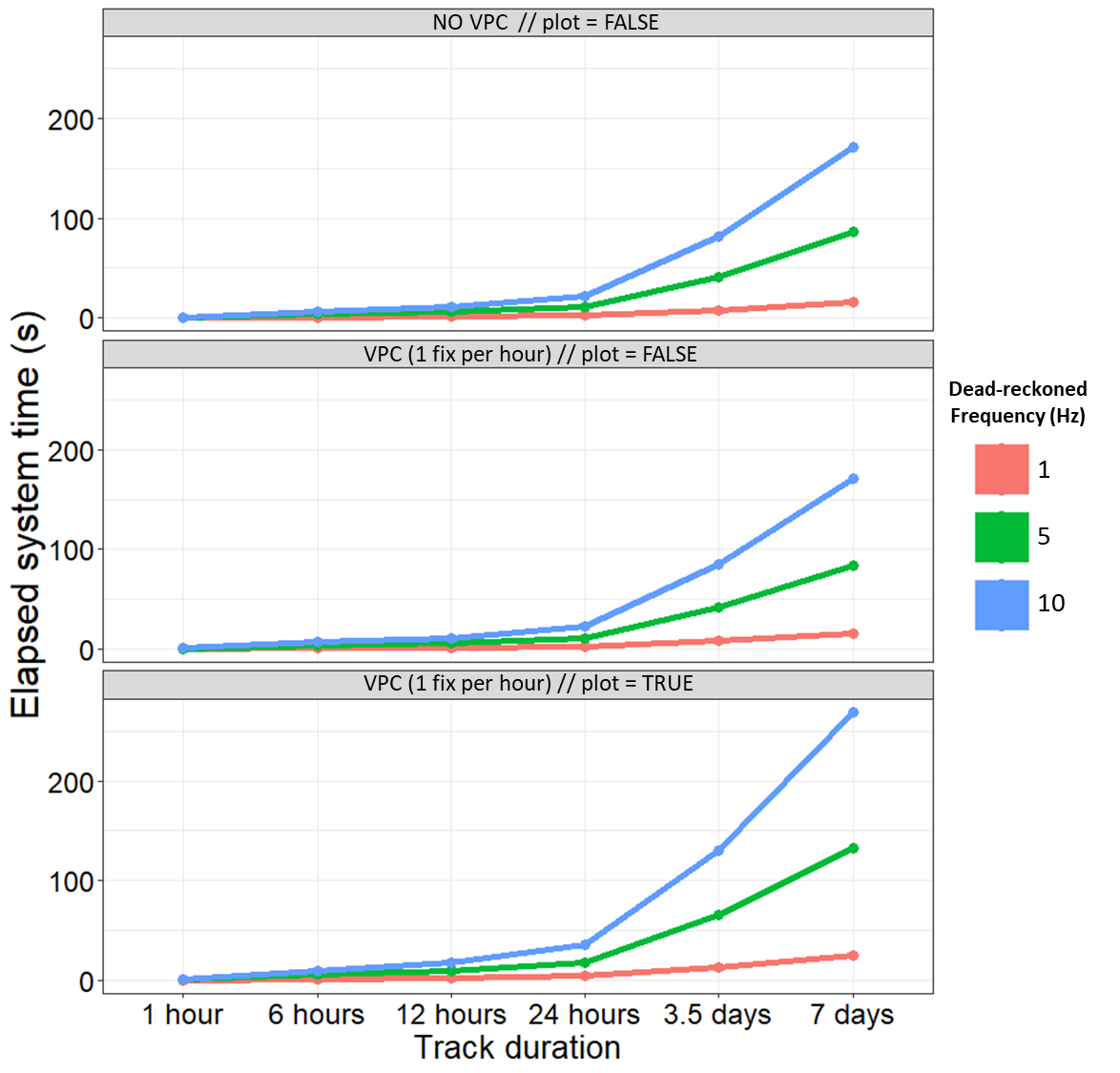


Figure S4. 1. Elapsed system time (s) to execute Gundog.Tracks according to the duration of data acquisition, frequency (Hz) of motion-sensor derived speed and heading estimates and user defined inputs (plot = TRUE vs FALSE and whether VPC occurred). Note that the biggest increase in computation time is the frequency of initial data input. The biggest source of system time is the for-loops used when dead-reckoning, plotting the tracks and computing the summary metrics (e.g., distance, speed etc.) at the end. This is why there is not much difference in computation speed when carrying out the method and degree of VPC as opposed to when not (VPC rate was chosen as 1 fix per hour here). This analysis was carried out on a MSI GP72 7RD Leopard laptop with intel core i7 processor. Whilst various changes in other user-defined inputs may modulate processing speeds somewhat, DR frequency and executing the summary plots are the main modulators.

**References**

1. Pedley M: **Tilt sensing using a three-axis accelerometer**. *Freescale semiconductor Application Note 2461* 2013, **1**(Rev. 6):1–22.

2. Wilson RP, Shepard E, Liebsch N: **Prying into the intimate details of animal lives: use of a daily diary on animals**. *Endangered Species Research* 2008, **4**(1-2):123-137.

3. Cheng G: **How to Convert 3-Axis Directions and Swap XY Axis of Accelerometer Data within Android™ Driver**. In: *FreeScale, Document Number AN4317.* 2011.

4. **Simple and Effective Magnetometer Calibration** [<https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration>]

5. Aphalo PJ: **Ggpmisc: Miscellaneous Extensions to’Ggplot2’**. *R package version 03* 2018, **3**.

6. Wilson RP, Rose KA, Gunner R, Holton M, Marks N, Bennett NC, Bell S, Twining J, Hesketh J, Duarte CM: **Forces experienced by instrumented animals depend on lifestyle**. *bioRxiv* 2020.

7. Fortin D, Beyer HL, Boyce MS, Smith DW, Duchesne T, Mao JS: **Wolves influence elk movements: behavior shapes a trophic cascade in Yellowstone National Park**. *Ecology* 2005, **86**(5):1320-1330.

8. Grolemund G, Wickham H: **Dates and times made easy with lubridate**. *Journal of statistical software* 2011, **40**(i03):1-25.