# Package ‘VTrack’

July 4, 2015

**Type** Package

**Title** A Collection of Tools for the Analysis of Remote Acoustic Telemetry Data

**Version** 1.11

**Date** 2015-07-03

**Author** Ross G. Dwyer, Mathew E. Watts, Hamish A. Campbell & Craig E. Franklin

**Maintainer** Ross Dwyer <ross.dwyer@uq.edu.au>

**Description** Designed to facilitate the assimilation, analysis and synthesis of animal location and movement data collected by the VEMCO suite of acoustic transmitters and receivers. As well as database and geographic information capabilities the principal feature of VTrack is the qualification and identification of ecologically relevant events from the acoustic detection and sensor data. This procedure condenses the acoustic detection database by orders of magnitude, greatly enhancing the synthesis of acoustic detection data.

**Depends** R (>= 3.1.0), foreach(>= 1.2.0), parallel, doParallel

**Imports** plotKML, sp, spacetime

**License** GPL (>= 2)

**URL** http://www.uq.edu.au/eco-lab/v-track

**Encoding** latin1

**Suggests** testthat

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2015-07-04 09:06:13

## R topics documented:

---

`VTrack-package` *VTrack: A Collection of Tools for the Analysis of Remote Acoustic Telemetry Data*

---

## Description

The package `VTrack` was designed by researchers at the University of Queensland to allow the analysis and visualisation of data generated from the VEMCO suite of passive and active acoustic receivers.

## Details

| | |
|---|---|
| Package: | VTrack |
| Type: | Package |
| Version: | 1.11 |
| Date: | 2015-07-03 |
| License: | GPL(>=2) |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell & Craig Franklin E.C.O.-lab and the E.D.G., School of Biological Sciences, University of Queensland, StLucia, Queensland, Australia.

Maintainer: Ross Dwyer <ross.dwyer@uq.edu.au>

### References

Campbell, H.A., Watts, M.E., Dwyer, R.G., Franklin, C.E. 2012. V-Track: software for analysing and visualising animal movement from acoustic telemetry detections. Marine and Freshwater Research, 63:815-820.

---

## `AATAMS1` *Passive Acoustic Monitoring of one animal in the AATAMS format*

---

### Description

This AATAMS dataset contains the relocations of one animal monitored between 08 January 2010 and 15 April 2011. Data supplied by Andrew Boomer from AATAMS-IMOS.

### Usage

```
data(AATAMS1)
```

### Format

A data frame with 2735 observations on the following 10 variables.

`timestamp` a vector of type POSIXct in Co-ordinated Universal Time (UTC) Greenwich Mean Time

`station.name` a character vector specifying the user-defined location for a particular deployment. This is usually assigned and recorded in the receivers memory in VUE before receiver deployment. Multiple receivers may be associated with the same `station.name`

`latitude` a numeric vector containing the location's latitude (decimal degrees)

`longitude` a numeric vector containing the location's longitude (decimal degrees)

`receiver.ID` a character vector specifying the unique identity of each receiver according to their model and serial number (i.e. `VR2W-101731`)

`tag.ID` a character vector containing either a combination of the code space and factory assigned transmitter ID number (i.e. `346`)

`species` the species being studied. `NA` suggests that no species name was supplied

`uploader` a character vector giving the identity of person who uploaded the data

`transmitter.ID` a numeric vector containing the factory assigned transmitter serial number, `A69-1303-7796`

`organisation` a character vector giving the organisation to which the data belongs. Data belongs to `AATAMS-IMOS`

### Details

The coordinates are given in decimal degrees (WGS 84), time is GMT+10hrs.

## Source

http://imos.org.au/home.html

## Examples

```
# Load the data and print the first few rows of the data frame
data(AATAMS1)
head(AATAMS1)
```

---

## `ComputeAzimuth` *Compute the Azimuth Between Two Coordinates*

---

### Description

This function computes the Azimuth from two geographical coordinates. These locations must be in decimal degrees.

### Usage

```
ComputeAzimuth(lat1, lat2, lon1, lon2)
```

### Arguments

| | |
|---|---|
| `lat1` | the latitude of the first coordinate |
| `lat2` | the latitude of the second coordinate |
| `lon1` | the longitude of the first coordinate |
| `lon2` | the longitude of the second coordinate |

### Details

Coordinates are given in decimal degrees (WGS 84)

---

## `ComputeDistance` *Compute the Distance Between Two Coordinates*

---

### Description

This function computes the distance between two geographical coordinates. These locations must be in decimal degrees.

### Usage

```
ComputeDistance(Lat1, Lat2, Lon1, Lon2)
```

## Arguments

| | |
|---|---|
| `Lat1` | the latitude of the first coordinate |
| `Lat2` | the latitude of the second coordinate |
| `Lon1` | the longitude of the first coordinate |
| `Lon2` | the longitude of the second coordinate |

## Details

Coordinates are given in decimal degrees (WGS 84)

## Examples

```
# Calculate the distance between two coordinates
ComputeDistance(-12.19506,-12.19477,141.8946,141.8980)
```

---

`crocs` *Passive Acoustic Monitoring of Saltwater Crocodiles*

---

## Description

This VEMCO dataset contains the relocations of 3 saltwater crocodiles monitored between 09 September 2008 to 31 December 2008 on the Wenlock River, Cape York, Queensland, Australia. Data supplied H. Campbell from the School of Biological Sciences, The University of Queensland, Queensland, Australia.

## Usage

```
data(crocs)
```

## Format

A data frame with 11229 observations on the following 14 variables.

`Date.Time` a vector of type POSIXct in Co-ordinated Universal Time (UTC)/ Greenwich Mean Time

`Code.Space` a character vector containing the type of coding scheme used for the particular tag type. This unique identifier encompasses all the information required for a receiver to detect and decode that particular transmitter (e.g. A69-0001 is an acoustic transmitter operating at a frequency of 69Hz and has 0001 as a unique number identifier).

`ID` a numeric vector giving the identity of each transmitter, `94,99,139,138`

`Sensor.1` a numeric vector giving the value of the environmental sensor such as temperature or depth at the time of detection

`Units.1` a factor with levels `m` and degrees C

`Sensor.2` as this study included coded tags only, no environmental sensor data are present in this vector

`Units.2` as this study included coded tags only, no sensor units are present in this vector

`Transmitter.Name` a character vector containing user defined animal names

`Transmitter.S.N` a numeric vector containing the factory assigned transmitter serial number

`Receiver.Name` a factor specifying the unique identity of each receiver according to their model and serial number. i.e. `VR2W-103548`

`Receiver.S.N` a numeric vector containing the factory assigned receiver serial number i.e. `1035481`

`Station.Name` an optional character vector specifying the user-defined location for a particular deployment. This is usually assigned and recorded in the receivers memory in VUE before receiver deployment. Multiple receivers may be associated with the same station name.

`Station.Latitude` a numeric vector containing the location's latitude in decimal degrees

`Station.Longitude` a numeric vector containing the location's longitude in decimal degrees

## Details

The coordinates are given in decimal degrees WGS 84, time is GMT +10 hrs

## Source

`www.uq.edu.au/eco-lab/V-Track`

## Examples

```
#load the data and print the first few rows of the data frame
data(crocs)
head(crocs)
```

---

`ExtractData` *Filter a Subset of Data from a VTrack File*

---

## Description

ExtractData enables the user to extract/remove a subset of data (i.e. transmitters, receivers, stations and time period) from the file. For dual sensor data, this function also allows the user to extract sensor only data (i.e. temperature or depth data) from the file.

## Usage

```
ExtractData(sInputFile, sQuerySTARTTIME = NULL, sQueryENDTIME = NULL,
  sQueryTransmitterList = NULL, sQueryReceiverList = NULL,
  sQueryStationList = NULL, sQueryDataType = NULL)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing VTrack-transformed acoustic tracking data |
| `sQuerySTARTTIME` | an optional POSIXct string specifying the date/time start point from which data will be extracted from the original file. Date and time must be in the format `'yyyy-mm-dd HH:MM:SS'`. Default is `NULL` |
| `sQueryENDTIME` | an optional POSIXct string specifying the date/time end point from which data will not be extracted from the original file. Date and time must be in the format `'yyyy-mm-dd HH:MM:SS'`. Default is `NULL` |
| `sQueryTransmitterList` | an optional character string specifying the individual transmitters to be extracted from the original file. Default is `NULL` |
| `sQueryReceiverList` | an optional character string specifying the receivers to be extracted from the original file. Default is `NULL` |
| `sQueryStationList` | an optional character string specifying the stations to be extracted from the original file. Default is `NULL` |
| `sQueryDataType` | an optional character string specifying the sensor data type (e.g. depth `m`) to be extracted from the original file. Default is `NULL` |

## Value

Subsets the original a data frame returning the following components:

| | |
|---|---|
| `DATETIME` | a vector of class POSIXct of the time of location fix of type `'yyyy-mm-dd HH:MM:SS'` |
| `TRANSMITTERID` | a numeric vector giving the identity of each transmitter (= `ID`) |
| `SENSOR1` | a numeric vector containing the value of the environmental sensor (i.e. temperature or depth) at the time of detection |
| `UNITS1` | a character vector containing the units of each sensor value (e.g. `m`) |
| `TRANSMITTERID` | a character vector containing the identity of each transmitter (= `ID or tag ID`) |
| `RECEIVERID` | a character vector containing the factory assigned receiver serial number (= `Receiver S/N or receiver ID`) |
| `STATIONNAME` | a character vector containing the user defined station name (= `Station.Name or station name`) |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## See Also

`ExtractUniqueValues`

## Examples

```
# Load the crocodile data set
data(crocs)

# Convert data into the VTrack archive format
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Extract list of transmitters from test archive 1
(TransmitterList <- ExtractUniqueValues(Vcrocs,2))


# Extract data from an archive only for the receivers listed.
R103555 <- ExtractData(Vcrocs,
                  sQueryReceiverList = 10355)

# Extract depth only data for a certain time period.
Vcrocs_Depth <- ExtractData(Vcrocs,
sQueryDataType = "m",
sQuerySTARTTIME = "2008-08-01 21:00:00",
sQueryENDTIME = "2009-10-31 23:03:00")

# Plot the detections against time for each TRANSMITTERID
par(mfrow=c(1,1),las=1,bty="l")
plot(as.Date(Vcrocs$DATETIME), as.numeric(as.factor(as.numeric(as.character(
Vcrocs$TRANSMITTERID)))),
     ylab="TRANSMITTERID",xlab="DATETIME",
     yaxt="n",pch=16,cex=0.7)
axis(side=2, at=seq(1,length(TransmitterList),1),
     labels = TransmitterList[order(as.numeric(
TransmitterList))])

# For TRANSMITTERID 139 plot the detections against time for each RECEIVERID
par(mfrow=c(1,1),las=1,bty="l")
T139 <- ExtractData(Vcrocs,sQueryTransmitterList = c("139"))
T139_R <- ExtractUniqueValues(T139,5)
plot(as.Date(T139$DATETIME),
as.numeric(as.factor(
as.numeric(as.character(T139$RECEIVERID)))),
ylab="RECEIVERID",xlab="DATETIME",
yaxt="n",pch=16,cex.axis=0.9,cex=0.7,
main=unique(T139[,2]))
axis(side=2,las=1, at=seq(1,length(T139_R),1),cex.axis=0.7,
     labels = T139_R[order(as.numeric(T139_R))])

# Extract data from TRANSMITTERID 139 and plot raw sensor data
par(mfrow=c(1,1),las=1,bty="l")
plot(T139$SENSOR1~
T139$DATETIME,xlab="Date",
ylab="Depth (m)",main=unique(T139[,2]),
pch=16,cex=0.7)
```

---

`ExtractRecSummary` *Extended Function to Extract Summary Data for each Receiver in the File*

---

## Description

This function extracts summary data for each receiver in the file

## Usage

```
ExtractRecSummary(sInputFile)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing VTrack archive data, this archive is created using the ReadInputData function |

## Details

duration given in days.

## Value

| | |
|---|---|
| `RECEIVERID` | a character vector containing the factory assigned receiver serial number |
| `STATIONNAME` | a character vector containing the user defined station name |
| `FIRSTDETECT` | a vector of class POSIXct of the time of the first location fix |
| `NODETECTS` | a numeric vector giving the number of transmitter location fixes |
| `LASTDETECT` | a vector of class POSIXct of the time of the last location fix |
| `NOTRANSMITTER` | a numeric vector giving the number of unique transmitter ids detected |

## Author(s)

Ross Dwyer

## Examples

```
data(crocs)
# Load the crocodile data in the VTrack archive format
#  adding 10 hours to convert from UTC
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                          dateformat = NULL,
                          sVemcoFormat='1.0')

# Extract summary table of receivers in the file
ExtractRecSummary(Vcrocs)
```

---

`ExtractTagSummary` *Function to Extract Summary Data for each Transmitter in the File*

---

## Description

This function extracts summary data for each transmitter in the File

## Usage

```
ExtractTagSummary(sInputFile)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing VTrack archive data, this archive is created using the ReadInputData function |

## Details

distances are given in meters

## Value

| | |
|---|---|
| `TRANSMITTERID` | a character vector containing the factory assigned transmitter tag id |
| `FIRSTDETECT` | a vector of class POSIXct of the time of the first location fix |
| `LASTDETECT` | a vector of class POSIXct of the time of the last location fix |
| `NODETECTS` | a numeric vector giving the number of transmitter location fixes |
| `STARTREC` | a character vector containing the first receiver at which the transmitter was detected |
| `ENDREC` | a character vector containing the last receiver at which the transmitter was detected |
| `NORECS` | a numeric vector giving the number of unique receivers at which the transmitter was detected |

## Author(s)

Ross Dwyer

## Examples

```
# Load the crocodile data in the VTrack 1.0 archive format
#  adding 10 hours to convert from UTC
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Extract summary table of transmitters in the file
ExtractTagSummary(Vcrocs)
```

---

`ExtractUniqueValues` *Extract Transmitters Found, or Receivers and Stations Used*

---

## Description

Extract a list of the transmitters located, receivers used or stations used during the study.

## Usage

```
ExtractUniqueValues(sInputFile,iFieldToExtract)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing `VTrack`-transformed tracking data |
| `iFieldToExtract` | numeric. Column number of `sInputFile` relating to field of interest (`TRANSMITTERID = 2`; `RECEIVERID = 5`; `STATIONNAME = 6`) |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## Examples

```
data(crocs)
# Load the crocodile data in the VTrack archive format
#  adding 10 hours to convert from UTC
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Extract list of transmitters from the crocs dataset
```

```
ExtractUniqueValues(Vcrocs,2)

# Extract list of receivers from the crocs dataset
ExtractUniqueValues(Vcrocs,5)
```

---

`GenerateAnimationKMLFile` *Create Animation of Transmitter Residences and Movements to View in Google Earth*

---

## Description

This function creates a Keyhole Markup Language (KML) animation of horizontal movements that can be displayed in Google Earth. The animation shows when a transmitter was within the detection field of a receiver and when it moved between receivers or stations. Users can adjust the time slider to visualise individual time periods for display.

## Usage

```
GenerateAnimationKMLFile(sInputResidenceFile, sInputNonResidenceFile,
    sInputPointsFile, sOutputFile, sReceiverColour)
```

## Arguments

| | |
|---|---|
| `sInputResidenceFile` | the location of a `residences` event file (.csv) containing the `residences` table created using the `RunResidenceExtraction` function |
| `sInputNonResidenceFile` | the location of a `nonresidences` event file (.csv) containing the `nonresidences` table created using the `RunResidenceExtraction` function |
| `sInputPointsFile` | the location of a points file (.csv) containing the latitude and longitude of all the `RECEIVERID` or `STATIONNAME` locations within the array. Location data should be uploaded in decimal degrees in the WGS 84 datum. In many arrays, animals may not be capable of moving in a direct line between receivers (e.g. in river systems). VTrack offers users the flexablity to include other points (with their corresponding geographical locations) to link receivers along a circuitous path |
| `sOutputFile` | a string detailing the location and name of the output file to be created |
| `sReceiverColour` | colour of the receivers in the output .kml |

## Details

the output is a .kml that can be viewed as an animation in Google Earth

## Author(s)

Ross Dwyer, Matthew Watts, Hamish Campbell

## See Also

GenerateDirectDistance, GenerateCircuitousDistance, RunResidenceExtraction

## Examples

```
## Not run:
###GenerateAnimationKMLFile example

# Note, users must download Google Earth in order to visualise the kml.
# Extract residence and nonresidence events from the archived crocodile data

# Load crocodile datset into VTrack archive
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Load Wenlock points file and generate circuitous distance matrix
data(PointsCircuitous_crocs)
CircuitousDM <- GenerateCircuitousDistance(PointsCircuitous_crocs)

# Extract transmitter #139 data from crocs dataset
T139 <- ExtractData(Vcrocs,sQueryTransmitterList = c("139"))

# Extract residence and nonresidence events from the archived crocodile data
#   Events occur when >1 detections occurs at a receiver and
#   detections are less than 43200 seconds (12hrs) apart
#   The circuitous distance matrix is used for distance calculations
T139Res<- RunResidenceExtraction(T139,
                                 "RECEIVERID",
                                 2,
                                 43200,
                                 sDistanceMatrix=CircuitousDM)

# The residences event file
T139resid <- T139Res$residences
# The nonresidences event file
T139nonresid <- T139Res$nonresidences

# Set working directory (in this case a temporary directory)
setwd(tempdir())

# Write the files to the temporary directory
write.csv(T139resid,"T139_resid.csv",row.names=FALSE)
write.csv(T139nonresid,"T139_nonresid.csv",row.names=FALSE)
write.csv(PointsCircuitous_crocs,"PointsCircuitous_crocs.csv",row.names=FALSE)

# Now generate the .kml animation and save to the temporary directory
```

```
GenerateAnimationKMLFile("T139_resid.csv","T139_nonresid.csv","PointsCircuitous_crocs.csv",
                   "T139.KML","ff0000ff")

# This file can be found within the tempdir() directory on your computer.
# Double-click on the .kml file to open in Google Earth

## End(Not run)
```

## GenerateAnimationKMLFile_Multitag *Create Animation of Multiple Transmitters to View in Google Earth*

### Description

This function creates a Keyhole Markup Language (KML) animation of transmitter detections at receivers that can be displayed in Google Earth. The animation shows the number of transmitters detected within the detection field of a receiver on a given day. Users can adjust the time slider to visualise individual time periods for display.

### Usage

```
GenerateAnimationKMLFile_Multitag(sInputFile,sPointsFile,sOutputFile)
```

### Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing VTrack archive data, this archive is created using the ReadInputData function |
| `sPointsFile` | a data frame containing the RECEIVERID, the coordinates and the detection RADIUS in meters. This should be in the format LOCATION, LATITUDE, LONGITUDE, RADIUS |
| `sOutputFile` | a string detailing the location and name of the output kml file to be created |

### Details

the output is a .kml that can be viewed as an animation in Google Earth

### Author(s)

Ross Dwyer, Matthew Watts, Hamish Campbell

### See Also

`ReadInputData`, `GenerateAnimationKMLFile_Track`

## Examples

```
  ## Not run:
    ###GenerateAnimationKMLFile_Multitag example

    # Note, users must download Google Earth in order to visualise the kml.

    # Load crocodile datset into VTrack archive
    data(crocs)
    data(PointsDirect_crocs)

    Vcrocs <- ReadInputData(infile=crocs,
                            iHoursToAdd=10,
                            fAATAMS=FALSE,
                            fVemcoDualSensor=FALSE,
                            dateformat = NULL,
                            sVemcoFormat='1.0')

    # Set working directory (in this case a temporary directory)
    setwd(tempdir())
    # or alternatively to your Desktop on Mac OS
    # setwd("~/Desktop")

    # Run the function to generate the KML
    GenerateAnimationKMLFile_Multitag(Vcrocs,
                                      PointsDirect_crocs,
                                      "Croc Multi.kml")

    # This file can be found within the tempdir() directory on your computer.
    # Double-click on the .kml file to open in Google Earth

## End(Not run)
```

---

`GenerateAnimationKMLFile_Track`
*Create Animation of Transmitter Track to View in Google Earth*

---

## Description

This function creates a Keyhole Markup Language (KML) animation of horizontal movements that can be displayed in Google Earth. The animation shows when a transmitter was within the detection field of a receiver and when it moved between receivers or stations. Users can adjust the time slider to visualise individual time periods for display.

## Usage

```
GenerateAnimationKMLFile_Track(sInputFile, sid, sPointsFile,
                               sOutputFile, sTrackColour)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing VTrack archive data, this archive is created using the ReadInputData function |
| `sid` | a string variable containing a single TRANSMITTERID |
| `sPointsFile` | a data frame containing the RECEIVERID, the coordinates and the detection RADIUS in meters. This should be in the format LOCATION, LATITUDE, LONGITUDE, RADIUS |
| `sOutputFile` | a string detailing the location and name of the output kml file to be created |
| `sTrackColour` | colour of the tracks in the output .kml |

## Details

the output is a .kml that can be viewed as an animation in Google Earth

## Author(s)

Ross Dwyer, Matthew Watts, Hamish Campbell

## See Also

`ReadInputData`, `RunResidenceExtraction`, `GenerateAnimationKMLFile`

## Examples

```
## Not run:
###GenerateAnimationKMLFile_Track example

# Note, users must download Google Earth in order to visualise the kml

# Load crocodile datset into VTrack archive
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Load Wenlock points file and generate circuitous distance matrix
data(PointsDirect_crocs)

# Set working directory (in this case a temporary directory)
setwd(tempdir())
# or alternatively to your Desktop on Mac OS
# setwd("~/Desktop")

(TransmitterList <- ExtractUniqueValues(Vcrocs,2)) # Extract the transmitter names
TransmitterList[1] # Let's create the track for this tag
```

```
# Run the function to generate the KML for a single transmitter
GenerateAnimationKMLFile_Track(Vcrocs, # VTrack archive file
                               TransmitterList[1], # Transmitter id
                               PointsDirect_crocs, # points file
                               "Track1.kml", # file name
                               "cc69deb3") # colour of the track

# This file can be found within the tempdir() directory on your computer.
# Double-click on the .kml file to open in Google Earth

## End(Not run)
```

---

## GenerateCircuitousDistance

*Converts a Points File into a Distance Matrix Using the Circuitous Distances Along a Series of Receivers or Stations*

---

### Description

This function calculates the straight line distance beween a set of geographical coordinates and generates a matrix containing the distances between each of the locations (i.e. receivers/stations) minus the detection radius. This function works in series through a set of locations and may contain waypoints to create indirect paths.

### Usage

```
GenerateCircuitousDistance(sPointsFile)
```

### Arguments

| | |
|---|---|
| `sPointsFile` | a dataframe containing the `LOCATION` (i.e. the STATIONNAME or the RECEIVERID), the coordinates and the detection `RADIUS` in meters. This should be in the format `LOCATION`, `LATITUDE`, `LONGITUDE`, `RADIUS`. Waypoints connecting receivers/stations in series should be located between the relevent locations and have a `LOCATION = 0` |

### Value

| | |
|---|---|
| `DM` | a 2x2 matrix containing the pairwise circuitous `DISTANCE` between each `LOCATION` minus the detection `RADIUS`. Distances are returned in kilometers |

### Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

### See Also

`GenerateDirectDistance`

## Examples

```
# Load the points file
data(PointsCircuitous_crocs)

# Generate the Circuitous Distance Matrix
CircuitousDM <- GenerateCircuitousDistance(PointsCircuitous_crocs)

# Now plot example of how circuitous distances between receivers were generated
# In this example an individual must follow the course of the river in order to
#   move between receivers
par(mfrow=c(1,1),las=1,bty="l")
plot(PointsCircuitous_crocs$LONGITUDE,PointsCircuitous_crocs$LATITUDE,
    pch=1,cex=0.5,col="grey",xlab="Longitude",ylab="Latitude")
lines(PointsCircuitous_crocs$LONGITUDE,PointsCircuitous_crocs$LATITUDE,
      col="grey",lwd=0.3,lty=3)

Receiversonly <- na.omit(PointsCircuitous_crocs)
points(Receiversonly$LONGITUDE,Receiversonly$LATITUDE,pch=10,cex=1)
```

---

`GenerateDirectDistance` *Converts a Points File into a Distance Matrix Using Direct Distances Between Receivers or Stations*

---

## Description

This function calculates the straight line distance between a set of geographical coordinates and generates a matrix containing the distances between each of the ponts (i.e. receivers) minus the detection radius.

## Usage

```
GenerateDirectDistance(sPointsFile)
```

## Arguments

| | |
|---|---|
| `sPointsFile` | a dataframe containing the `LOCATION` (i.e. the STATIONNAME or the RECEIVERID), the coordinates and the detection `RADIUS` in meters. This should be in the format `LOCATION`, `LATITUDE`, `LONGITUDE`, `RADIUS` |

## Value

a 2x2 matrix containing the pairwise direct `DISTANCE` between each `LOCATION` minus the detection `RADIUS`. Distances are returned in kilometers

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## See Also

`GenerateCircuitousDistance`

## Examples

```
# Load the points file
data(PointsDirect_crocs)
# Generate the direct distance matrix
DirectDM <- GenerateDirectDistance(PointsDirect_crocs)

# Now plot example of how direct distances between receivers were generated
# In this example there are no structural boundary preventing an individual from
#   moving between receivers
par(mfrow=c(1,1),las=1,bty="l")
plot(PointsDirect_crocs$LONGITUDE,PointsDirect_crocs$LATITUDE,pch=10,cex=1,
     xlab="Longitude",ylab="Latitude")
for(i in 1:length(PointsDirect_crocs$LONGITUDE))
  {
   lines(PointsDirect_crocs$LONGITUDE[c(1,i)],PointsDirect_crocs$LATITUDE[c(1,i)],
         lwd=0.3,col="grey",lty=3)
   }
points(PointsDirect_crocs$LONGITUDE,PointsDirect_crocs$LATITUDE,pch=10,cex=1)
```

---

## `NonResidenceExtractId` *Extract the Non-residence Events, the Corresponding Distance Moved and the Rate of Movement*

---

### Description

This function creates a `nonresidences` data frame from a `residences` event data frame and a optional distance matrix (`sDistanceMatrix`). This function is not mandatory as it is carried out automatically if the user provides a distance matrix in the `sDistanceMatrix` field when running the `RunResidenceExtraction` function.

### Usage

```
NonResidenceExtractId(sResidenceEventFile, sDistanceMatrix = NULL)
```

### Arguments

| | |
|---|---|
| `sResidenceEventFile` | a residence event table |
| `sDistanceMatrix` | an optional two dimentional array (matrix) containing the pairwise distances between a series of receivers |

## Value

| | |
|---|---|
| STARTTIME | a POSIXct vector object containing the date and time a transmitter left a receiver/station after a residence event |
| ENDTIME | a POSIXct vector object containing the date and time a transmitter arrived at a receiver/station and a new residence event was logged |
| NONRESIDENCEEVENT | a numeric vector indexing each nonresidence event |
| TRANSMITTERID | a numeric or character vector indexing the transmitter from which nonresidence events were determined |
| RECEIVERID1 | a numeric or character vector indexing the receiver which the transmitter initially moved from. If `STATIONNAME` is specified in the function, `STATIONNAME1` is returned |
| RECEIVERID2 | a numeric or character vector indexing the receiver which the transmitter moved to. If `STATIONNAME` is specified in the function, `STATIONNAME2` is returned |
| DURATION | a numeric vector containing the total time in seconds taken for the transmitter to move between two receivers or stations |
| DISTANCE | a numeric vector containing the minimum distance travelled in meters between two receivers/stations according to the distance matrix. If a distance matrix was not attached (=`NULL`), distance is returned as `0` |
| ROM | a numeric vector containing the rate of movement (`ROM`) in m/s. This is calculated from the distance travelled (i.e. `DISTANCE`) divided by the time taken to move between the receivers (i.e. `DURATION`) |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## See Also

`RunResidenceExtraction`

## Examples

```
# This function runs within the RunResidenceExtraction function when
# a distance matrix is provided by the user.

# Extract residence events at RECEIVERS from the VTrack-transformed
# saltwater crocodile archive

# Load the crocodile data into the VTrack archive
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')
```

```
# Extract data for only the transmitter #138
T138 <- ExtractData(Vcrocs,
                    sQueryTransmitterList = 138)

# Extract residence and non residence events using receiver data
#  Minimum number of detections to register as a residence
#  event = 2
#  Min time period between detections before residence event
#  recorded = 43200 secs (12 hours)
T139Res <- RunResidenceExtraction(sInputFile=T138,
                                 sLocation="RECEIVERID",
                                 iResidenceThreshold=2,
                                 iTimeThreshold=43200,
                                 sDistanceMatrix=NULL)

# The residences event table
T139resid <- T139Res$residences

# Generate the circuitous distance matrix
data(PointsCircuitous_crocs)
CircuitousDM <- GenerateCircuitousDistance(PointsCircuitous_crocs)

# Ensure there is only 1 Transmitter in dataset (if not, run the function within a loop)
length(unique(T139resid$TRANSMITTERID))

# Run the non-residence function
NonResidenceExtractId(sResidenceEventFile=T139resid,
                      sDistanceMatrix=CircuitousDM)
```

## PointsCircuitous_crocs *Points File Containing VR2 Locations on the Wenlock River in 2008 with Waypoints Connecting Receivers*

### Description

This points file contains the locations of 20 VR2 receivers plus their corresponding detection radiuses for monitoring saltwater crocodiles on the Wenlock River in 2008. When receivers have an obstructed line of view to landscape features (i.e. an island or a bend in the river) waypoints were added to facilitate the course of the shortest path. This points file corresponds with `crocs`.

### Usage

```
data(PointsCircuitous_crocs)
```

### Format

A data frame with 149 observations on the following 4 variables.

| | |
|---|---|
| `LOCATION` | a numeric vector containing the receiver serial number (i.e. `RECEIVERID`) |
| `LATITUDE` | a numeric vector containing the location's latitude in decimal degrees |
| `LONGITUDE` | a numeric vector containing the location's longitude in decimal degrees |
| `RADIUS` | a numeric vector containing the detection radius for the receiver in meters |

### Details

The coordinates are given in decimal degrees WGS 84, detection radiuses are in meters.

### Source

`www.uq.edu.au/eco-lab/V-Track`

### Examples

```
# Load the points file for the Wenlock River
data(PointsCircuitous_crocs)
head(PointsCircuitous_crocs)
receiversonly <- na.omit(PointsCircuitous_crocs)

# Plot the locations of the receivers plus the waypoints
par(mfrow=c(1,1),las=1,bty="l")
plot(PointsCircuitous_crocs$LONGITUDE, PointsCircuitous_crocs$LATITUDE,
     pch=1,cex=0.5,col="grey",xlab="Longitude",ylab="Latitude")
points(receiversonly$LONGITUDE,receiversonly$LATITUDE,cex=1,pch=10)
```

## `PointsDirect_AATAMS1` *Points File Containing VR2 Locations For AATAMS1*

### Description

This points file contains the locations of two acoustic stations plus their corresponding detection radiuses for monitoring x on y in 2009. This points file corresponds with `AATAMS1`

### Usage

```
data(PointsDirect_AATAMS1)
```

### Format

A data frame with 2 observations on the following 4 variables.

| | |
|---|---|
| `LOCATION` | a factor containing the station name |
| `LATITUDE` | a numeric vector containing the location's latitude (decimal degrees) |
| `LONGITUDE` | a numeric vector containing the location's longitude (decimal degrees) |
| `RADIUS` | a numeric vector the detection radius for the location in meters |

## Details

The coordinates are given in decimal degrees (WGS 84), detection radiuses are in meters.

## Source

urlhttp://imos.org.au/home.html

## Examples

```
# Load the points file for the AATAMS1 dataset
data(PointsDirect_AATAMS1)
head(PointsDirect_AATAMS1)
```

---

## PointsDirect_crocs *Points File Containing VR2 Locations on the Wenlock River in 2008*

---

## Description

This points file contains the locations of 20 VR2 receivers plus their corresponding detection radiuses for monitoring saltwater crocodiles on the Wenlock River in 2008. This points file corresponds with the `crocs` dataset

## Usage

```
data(PointsDirect_crocs)
```

## Format

A data frame with 20 observations on the following 4 variables.

`LOCATION` a numeric vector containing the factory assigned receiver serial number (`Receiver S/N`)

`LATITUDE` a numeric vector containing the location's latitude (decimal degrees)

`LONGITUDE` a numeric vector containing the location's longitude (decimal degrees)

`RADIUS` a numeric vector containing the detection radius for the location in meters

## Details

The coordinates are given in decimal degrees (WGS 84), detection radiuses are in meters.

## Source

`www.uq.edu.au/eco-lab/V-Track`

## Examples

```
# Load the points file for the Wenlock River
data(PointsDirect_crocs)
head(PointsDirect_crocs)

# Plot the locations of the receivers
par(mfrow=c(1,1),las=1,bty="l")
plot(PointsDirect_crocs$LONGITUDE,PointsDirect_crocs$LATITUDE,
     pch=10,cex=1,xlab="Longitude",ylab="Latitude")
```

---

`ReadInputData` *Read in a Raw VEMCO or AATAMS Data File into a VTrack Archive*

---

## Description

ReadInputData extracts single or dual sensor data from a raw VEMCO or AATAMS file to a VTrack structured data frame.

## Usage

```
ReadInputData(infile, iHoursToAdd=0, fAATAMS=FALSE, fVemcoDualSensor=FALSE,
    dateformat = NULL, sVemcoFormat='Default')
```

## Arguments

| | |
|---|---|
| `infile` | a data frame containing VEMCO/AATAMS tracking data |
| `dateformat` | an optional string containing the format of the `Date.Time` field |
| `iHoursToAdd` | the number of hours to add/subtract to convert the time-zone from Greenwich Mean Time (GMT) |
| `fAATAMS` | logical. If data frame is in AATAMS format (`fAATAMS = TRUE`), `fVemcoDualSensor` and `sVemcoFormat` are ignored |
| `fVemcoDualSensor` | logical. If VEMCO file contains single sensor data (`FALSE`), if dual sensor data (`TRUE`) |
| `sVemcoFormat` | an optional string containing the format of the VEMCO file. The infile was exported from VUE in either old Version (`'1.0'`) format, or in the new (`'Default'`) format |

## Value

| | |
|---|---|
| `DATETIME` | a vector of class POSIXct of the time of location fix of type |
| `TRANSMITTERID` | a numeric vector giving the identity of each transmitter (`ID`) |
| `SENSOR1` | a numeric vector containing the value of the environmental sensor (i.e. temperature or depth) at the time of detection |
| `UNITS1` | a character vector containing the units of each sensor value (e.g. `m` |

| | |
|---|---|
| `TRANSMITTERID` | a character vector containing the factory assigned transmitter tag id |
| `RECEIVERID` | a character vector containing the factory assigned receiver serial number (`Receiver S/N`) |
| `STATIONNAME` | a character vector containing the user defined station name |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## Examples

```
# Load the crocodile dataset
data(crocs)

# Convert data into the VTrack archive format
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat=NULL,
                        sVemcoFormat='1.0')

# Plot a frequency histogram of total detection per transmitter
NoDetect_ID <- tapply(rep(1,nrow(Vcrocs)),
                      Vcrocs$TRANSMITTERID,sum)
par(mfrow=c(1,1),las=1,bty="l")
bp <- barplot(height=NoDetect_ID,
              ylab="Number of detections",xlab="Transmitter ID",
              axes=FALSE,axisnames=FALSE)
labels <- names(NoDetect_ID)
text(bp, par("usr")[3],labels=labels,
     srt=45,adj=c(1.1,1.1),xpd=TRUE,cex=0.8)
axis(2)

# Plot a frequency histogram of total detection per receiver
NoDetect_REC <- tapply(rep(1,nrow(Vcrocs)),Vcrocs$RECEIVERID,sum)
bp <- barplot(height=NoDetect_REC,
              ylab="Number of detections",xlab="Receiver ID",
              axes=FALSE, axisnames=FALSE)
labels <- names(NoDetect_REC)
text(bp, par("usr")[3], labels=labels,
     srt = 45, adj=c(1.1,1.1),xpd=TRUE,cex=0.8)
axis(2)

# Plot a frequency histogram of total detections over time
NoDetect_DAY <- tapply(rep(1,nrow(Vcrocs)),
                       as.Date(Vcrocs$DATETIME),sum)
barplot(height=NoDetect_DAY,
        names.arg=names(NoDetect_DAY),
        ylab="Number of detections",
        xlab="Date")
```

## ReturnVR2Distance — *Extract the Distances Moved Between VR2 Receiver Units Within the Acoustic Detection Database*

### Description

This function uses combines the non-residence event table with a distance matrix to extract the minimum distance moved between two receivers. This function returns a numeric vector containing the minimum distance moved between receivers (extracted from `sDistanceMatrix`). This function is not mandatory as it is carried out automatically if the user provides a distance matrix in the `sDistanceMatrix` field when running the `RunResidenceExtraction` function.

### Usage

```
ReturnVR2Distance(NonResidenceFile, sDistanceMatrix)
```

### Arguments

| | |
|---|---|
| `NonResidenceFile` | a data frame containing the `nonresidences` event table |
| `sDistanceMatrix` | a two dimentional array (matrix) containing the pairwise distances between an array of VR2 receivers |

### Value

a numeric vector of minimum distance travelled (in kilometers) corresponding to the values listed in the distance matrix

### Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

### See Also

`RunResidenceExtraction`, `NonResidenceExtractId`

### Examples

```
## Not run:
# Extract residence events at RECEIVERS from the VTrack transformed
#   crocodile dataset

# Load the crocodile dataset into the VTrack archive format
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                            dateformat = NULL,
                            sVemcoFormat='1.0')

# Extract data for only the transmitter #138
T138 <- ExtractData(Vcrocs,
                    sQueryTransmitterList = 138)

# Extract residence and non residence events
#   Minimum number of detections to register as a residence
#   event = 2
#   Min time period between detections before residence event
#   recorded = 43200 secs (12 hours)
T139Res <- RunResidenceExtraction(sInputFile=T138,
                                  sLocation="RECEIVERID",
                                  iResidenceThreshold=2,
                                  iTimeThreshold=43200,
                                  sDistanceMatrix=NULL)

# The nonresidences event table
T139nonresid <- T139Res$nonresidences

# Generate the Direct Distance Matrix
data(PointsDirect_crocs)
DirectDM <- GenerateDirectDistance(PointsDirect_crocs)

# Run the VR2 distances function
(My_distances <- ReturnVR2Distance(NonResidenceFile = T139nonresid,
                                   sDistanceMatrix = DirectDM))

## End(Not run)
```

---

## `RunResidenceExtraction` *Extract Residence and Nonresidence Events Within the Acoustic Detection Database*

---

### Description

Events when the transmitter remained within the detection field of a given receiver. The event is triggered when a transmitter is detected by a receiver and terminated when the transmitter is detected at another receiver, or if the transmitter is not detected by the same receiver within a user defined timeout window. `nonresidences` (i.e. when a transmitter moves between the detection fields of two receivers) are generated from the `residences` event table. The function returns a list object containing a `residenceslog`, a `residences` event table and a `nonresidences` event table.

### Usage

```
RunResidenceExtraction(sInputFile, sLocation, iResidenceThreshold, iTimeThreshold,
    sDistanceMatrix = NULL)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a data frame containing VTrack archive data, this archive is created using the `ReadInputData` function |
| `sLocation` | the location at which we wish to analyse our residence and non-residence events (i.e. `RECEIVERID` or `STATIONNAME`) |
| `iResidenceThreshold` | the minimum number of successive transmitter pings detected at a receiver before a residence event is recorded |
| `iTimeThreshold` | the minimum time period in seconds between pings before a residence event is recorded |
| `sDistanceMatrix` | an optional two dimensional array containing the distances between a set of points. This can be the distances between each receiver or between each station. The first column in this matrix must contain the names of the receivers/stations. The diagonal of the distance matrix should be printed as `0` if appropriate and the upper triange of the distance matrix should also be calculated. Row 1 must contain the names of the corresponding receivers/stations and column 1 should be named `DM`. If a distance matrix is not available, Distances moved and the rate of movement (`ROM`) are returned as `0` in the `nonresidences` event table |

## Value

A `list` object containing 3 tables. In the `residenceslog` table:

| | |
|---|---|
| `DATETIME` | a POSIXct vector object containing the date and time that the information was logged at the receiver |
| `RESIDENCEEVENT` | a numeric vector indexing all the individual detections which make up each particular residence event listed in the residence event table |
| `RECORD` | a numeric vector indexing each detection within the event |
| `TRANSMITTERID` | a numeric or character vector indexing the transmitter from which residence events were determined |
| `RECEIVERID` | a numeric or character vector indexing the receiver where the event occurred. If `STATIONNAME` is specified in the function, `STATIONNAME` is returned |
| `ELAPSED` | a numeric vector containing the total time in seconds of the event |

In the `residences` event table:

| | |
|---|---|
| `STARTTIME` | a POSIXct vector object containing the date and time a residence event was initiated |
| `ENDTIME` | a POSIXct vector object containing the date and time a residence event ended |
| `RESIDENCEEVENT` | a numeric vector indexing each particular event back to the residenceslog table where all the individual detections making up the event can be viewed |
| `TRANSMITTERID` | a numeric or character vector indexing the transmitter from which residence events were determined |

| | |
|---|---|
| `RECEIVERID` | a numeric or character vector indexing the receiver where the event occurred. If `STATIONNAME` is specified in the function, `STATIONNAME` is returned |
| `DURATION` | a numeric vector containing the time in seconds from the first to last detection within the event |
| `ENDREASON` | a character vector containing the reason why the residence event ended. This may be due to the transmitter appearing at another receiver (`receiver`) or if the last detection had passed the user defined timeout threshold (`timeout`). `signal lost` indicates the last recording of each transmitter. |
| `NUMRECS` | a numeric vector containing the number of records detected within each event |

In the `nonresidences` event table:

| | |
|---|---|
| `STARTTIME` | a POSIXct vector object containing the date and time a transmitter left a receiver or station |
| `ENDTIME` | a POSIXct vector object containing the date and time a transmitter arrived at a different receiver orstation |
| `NONRESIDENCEEVENT` | a numeric vector indexing each nonresidence event |
| `TRANSMITTERID` | a numeric or character vector indexing the transmitter from which nonresidence events were determined |
| `RECEIVERID1` | a numeric or character vector indexing the receiver which the transmitter initially moved from. If `STATIONNAME` is specified in the function, `STATIONNAME1` is returned |
| `RECEIVERID2` | a numeric or character vector indexing the receiver which the transmitter moved to. If `STATIONNAME` is specified in the function, `STATIONNAME2` is returned |
| `DURATION` | a numeric vector containing the total ime in seconds taken for the transmitter to move between the two receivers |
| `DISTANCE` | a numeric vector containing the minimum distance travelled (m) between two receivers or stations according to the distance matrix. If a distance matrix was not attached (`NULL`), distance is returned as `0` |
| `ROM` | a numeric vector containing the rate of movement (ROM) in m/s. This is calculated from the distance travelled (`DISTANCE`) divided by the time taken to move between the receivers (`DURATION`) |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## See Also

`ReadInputData`, `RunSensorEventExtraction`, `RunTimeProfile`

## Examples

```
## Not run:
```

```
# Extract residence events from the archived crocodile data

# Load the crocodile dataset into the VTrack archive format
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Load and generate the direct distance matrix
data(PointsDirect_crocs)
DirectDM <- GenerateDirectDistance(PointsDirect_crocs)

# Extract data for only transmitter #139
T139 <- ExtractData(Vcrocs,sQueryTransmitterList = c("139"))
T139_R <- ExtractUniqueValues(T139,5)

# Extract residences and nonresidences events.
#   Events occur when >1 detection occurs at a receiver and detections
#   are less than 43200 seconds apart
#   The direct distance matrix is used for distance calculations
T139Res<- RunResidenceExtraction(T139,
                                 "RECEIVERID",
                                 2,
                                 43200,
                                 sDistanceMatrix=DirectDM)

# The residenceslog table
T139log <- T139Res$residenceslog
# The residences event file
T139resid <- T139Res$residences
# The nonresidences event file
T139nonresid <- T139Res$nonresidences

# The RESIDENCEEVENT number in the residences event table corresponds
# to the RESIDENCEEVENT number in the residenceslog table
subset(T139log,T139log$RESIDENCEEVENT==2)
subset(T139resid, T139resid$RESIDENCEEVENT==2)

subset(T139log,T139log$RESIDENCEEVENT==8)
subset(T139resid, T139resid$RESIDENCEEVENT==8)

# Scale duration spent at receivers into 4 bins: <1min, <1hr, <1day, >1day
pchDURATION <- ifelse(T139resid$DURATION<60,0.1,
                      ifelse(T139resid$DURATION<(60*60),0.5,
                             ifelse(T139resid$DURATION<(60*60*24),1,3)))

# For TRANSMITTERID 139 plot the detections against time for each RECEIVERID
par(mfrow=c(1,1),las=1,bty="l")
plot(as.Date(T139resid$STARTTIME),
     as.numeric(as.factor(
       as.numeric(as.character(T139resid$RECEIVERID)))),
     ylab="RECEIVERID",xlab="DATETIME",
     yaxt="n",pch=1,cex.axis=0.9,cex=pchDURATION,
     main=unique(T139resid$TRANSMITTER))
axis(side=2,las=1, at=seq(1,length(T139_R),1),cex.axis=0.7,
     labels = T139_R[order(as.numeric(T139_R))])

# Now plot the residence time at a receiver spatially and with
# each point representing the duration spent at each receiver
myresid1 <- subset(T139resid, T139resid$ENDREASON=="receiver")
totalDur <- tapply(myresid1$DURATION,myresid1$RECEIVERID,sum)
totalDurT <- data.frame(LOCATION=names(totalDur), DURATION=as.vector(totalDur))
XYDuration <- merge(PointsDirect_crocs,totalDurT)

plot(PointsDirect_crocs$LONGITUDE,PointsDirect_crocs$LATITUDE,
     pch=1,cex=0.5,col="grey40",
    xlim=c((min(PointsDirect_crocs$LONGITUDE)-0.01),(max(PointsDirect_crocs$LONGITUDE)+0.01)),
    ylim=c((min(PointsDirect_crocs$LATITUDE)-0.01),(max(PointsDirect_crocs$LATITUDE)+0.01)),
     xlab="Longitude",ylab="Latitude",
     main=unique(T139resid$TRANSMITTER))
points(XYDuration$LONGITUDE,XYDuration$LATITUDE,
       cex=XYDuration$DURATION/500000, pch=16)

## End(Not run)
```

---

`RunSensorEventExtraction`

*Extract Sensor Events within an Acoustic Detection Database*

---

## Description

This function identifies, qualifies and quantifies increasing or decreasing sensor events within the acoustic detection database. Events are defined by the user and are based on sensor threshold and time-out parameters between detections. These are established from changes in sensor values between detections, over a user-defined period of time. The location of the event is determined by either the station or the receiver or location

## Usage

```
RunSensorEventExtraction(sInputFile, iEventType, sLocation, iSensor,
   rTriggerThreshold, iTimeThresholdStart, iTimeThreshold, rTerminationThreshold)
```

## Arguments

| | |
|---|---|
| `sInputFile` | a dataframe containing VTrack-transformed acoustic tracking data |
| `iEventType` | the type of event the user wants to extract. This can be either an event whereby the sensor values increase within a certain time period (= `"INCREASE"`) or an event whereby the sensor values decrease within a certain time period (= `"DECREASE"`) |

| | |
|---|---|
| `sLocation` | the location at which we wish to analyse our sensor events (i.e. `RECEIVERID` or `STATIONNAME`) |
| `iSensor` | the sensor data type to be extracted from the original file. This corresponds to the sensor units (`UNITS1`) contained within the `sInputFile` data frame (e.g. Depth = `m`) |
| `rTriggerThreshold` | the minimum change in sensor units for an event to commence |
| `iTimeThresholdStart` | the maximum time period (seconds) in which the `rTriggerThreshold` is reached before a sensor event commences |
| `iTimeThreshold` | the maximum time period (seconds) between detections before the sensor event is completed and the counter is reset |
| `rTerminationThreshold` | how close the sensor must be to the starting value before a sensor event is completed and the counter is reset |

## Value

A `list` object 2 tables. In the sensor event `logtable`:

| | |
|---|---|
| `DATETIME` | a vector of type POSIXct in Co-ordinated Universal Time (UTC)/ Greenwich Mean Time. The date and time that the location and sensor data was logged at the receiver |
| `SENSOREVENT` | a numeric vector indexing all the individual detections which make up each particular sensor event listed in the `event` table |
| `RECORD` | a numeric vector indexing each detection within the event |
| `TRANSMITTERID` | a numeric or character vector indexing the transmitter from which sensor events were determined |
| `RECEIVERID` | a numeric or character vector indexing the location where the event occurred. If `STATIONNAME` is specified in the function, the `STATIONNAME` where the event occurred is returned here |
| `SENSOR1` | a numeric vector containing the duration of the event in seconds |
| `ELAPSED` | a numeric vector containing the total time in seconds of the event |

In the sensor `event` table:

| | |
|---|---|
| `STARTTIME` | a POSIXct vector object containing the date and time a sensor event was initiated |
| `ENDTIME` | a POSIXct vector object containing the date and time a sensor event ended |
| `SENSOREVENT` | a numeric vector indexing each particular event back to the `logtable`, where all the individual detections making up the event can be viewed |
| `TRANSMITTERID` | a numeric vector indexing the transmitter from which sonsor events were determined |
| `RECEIVERID` | a numeric vector indexing the location where the event occurred. If `STATIONNAME` is specified in the function, the `STATIONNAME` where the event occurred is returned here |

| | |
|---|---|
| DURATION | a numeric vector containing the duration of the event in seconds |
| STARTSENSOR | a numeric vector containing the sensor value when the event was initialised |
| ENDSENSOR | a numeric vector containing the sensor value when the event was either completed or terminated |
| MAXSENSOR | a numeric vector containing the maximum sensor value attained during the event |
| ENDREASON | a character vector providing information on why the event was terminated. If the sensor returned to a value within the termination threshold from the STARTSENSOR value and within the time threshold (= `return`) or exceeded the timeout threshold between successive detections (= `timeout`) |
| NUMRECS | a number vector containing number of detections that compose the event |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## See Also

RunResidenceExtraction, RunTimeProfile

## Examples

```
## Not run:

## Example 1

# Extract depth events from transmitters attached
#   to crocodiles and plot a single diving event

# Load crocodile data
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Extract depth data for only the transmitter #139
T139 <- ExtractData(Vcrocs,
                    sQueryTransmitterList = 139)

# Extract increasing depth sensor events
#   Start depth event when there is an depth increase of 0.5m within 1 hr
#   Max interval between detections = 1 hr
#   Complete event when sensor returns within 0.5 of the starting value
T139dives <- RunSensorEventExtraction(T139,
                                    "INCREASE",
                                    "RECEIVERID",
                                    "m",
                                    0.5,
                                           (1*60*60),
                                           (60*60),
                                           0.5)

# The sensor logfile
T139divelog <- T139dives$logtable
# The sensor event file
T139diveevent <- T139dives$event

# Return list of event numbers where sensor events were complete
T139diveevent[which(T139diveevent$ENDREASON=="return"),"SENSOREVENT"]

# Now extract and plot a single sensor event (we have swapped the axes round
#   to show the diving behaviour)
mylog <- subset(T139divelog,T139divelog$SENSOREVENT==19)
par(mfrow=c(1,1),las=1,bty="l")
plot(mylog$DATETIME,(mylog$SENSOR1),
     xlab="Event duration (mins)",ylab="Depth (m)",type="b",
     yaxs = "i", xaxs = "i", ylim = rev(c(0,max(mylog$SENSOR1+0.5))),
     xlim = (range(mylog$DATETIME)+(c(-60,30))),
     pch=as.character(mylog$RECORD))
title(main=paste("Id=",mylog[1,4],", event=",mylog[1,2], sep=" "))

#######################################################


## End(Not run)
```

## RunTimeProfile — *Extract a Time Profile for Depth, Temperature, Residence or Non-residence Events*

### Description

This function groups sensor, residence or non-residence events into time profiles classified by time. By specifying the time profile as hour, day, week, or month, the respective time profile is extracted for that particular event. Users can also extract a circadian profile for each event where events are filtered for each hour in a diel cycle (24 hr) and summed across days.

### Usage

```
RunTimeProfile(sInputFile, sDATETIMEField, sProfilePeriod)
```

### Arguments

| | |
|---|---|
| `sInputFile` | an event data frame containing either the residence, movement, diving or temperature events |
| `sDATETIMEField` | a character string identifying the `DATE` field (a POSIXct) used to create the time profile from the event data frame (= `STARTTIME`, `ENDTIME`) |

|  |  |
|---|---|
| `sProfilePeriod` | a character string relating to which profile should be extracted (= `hour`, `day`, `week`, `month`, `circadian`) |

## Value

|  |  |
|---|---|
| `DATE` | a POSIXct vector object containing the date and/or time an event was initiated |
| `FREQ` | a numeric vector containing the number of events for that hour/day/month |
| `SENSORMAX` | a numeric vector containing the maximum sensor reading for the time-grouped events |
| `SENSORAV` | a numeric vector containing the mean sensor reading for the time-grouped events |
| `SENSORSTDEV` | a numeric vector containing the standard deviation for the sensor readings for the time-grouped events |
| `TIMESUM` | a numeric vector containing the total duration of the time-grouped events (seconds) |
| `TIMEMAX` | a numeric vector containing the maximum duration reading for the time-grouped events (seconds) |
| `TIMEAV` | a numeric vector containing the mean duration reading for the time-grouped events (seconds) |
| `TIMESTDEV` | a numeric vector containing the standard deviation for the duration readings for the time-grouped events (seconds) |
| `DETECTIONS` | a numeric vector containing the number of detections which form all the events recorded for that time profile |
| `DISTANCE` | a numeric vector containing the sum of the minimum distance travelled which form all the events recorded for that time profile |

## Author(s)

Ross Dwyer, Mathew Watts, Hamish Campbell

## See Also

`RunResidenceExtraction`, `RunSensorEventExtraction`

## Examples

```
## Not run:
# RunTimeProfile example using residences, nonresidences and sensor events

# Load crocodile data and convert to a VTrack archive format
data(crocs)
Vcrocs <- ReadInputData(infile=crocs,
                        iHoursToAdd=10,
                        fAATAMS=FALSE,
                        fVemcoDualSensor=FALSE,
                        dateformat = NULL,
                        sVemcoFormat='1.0')

# Load receiver data and generate the circuitous distance matrix
data(PointsCircuitous_crocs)
CircuitousDM <- GenerateCircuitousDistance(PointsCircuitous_crocs)

# Extract depth data for transmitter #139
T139 <- ExtractData(Vcrocs,sQueryTransmitterList = c("139"))
T139_R <- ExtractUniqueValues(T139,5)

# Extract residence and non residence events
T139Res<- RunResidenceExtraction(T139,
                                 "RECEIVERID",
                                 2,
                                 43200,
                                 sDistanceMatrix=CircuitousDM)
# The residences event table
T139resid <- T139Res$residences
# The nonresidences event table
T139nonresid <- T139Res$nonresidences

# Generate plot dimentions
par(mfrow=c(2,2),las=1,bty="l")

## Plot a
# RESIDENCES: duration/day
Vres_D <- RunTimeProfile(T139resid,"STARTTIME","day")
day_res <- tapply(Vres_D$TIMEMAX,Vres_D$DATETIME,sum)[1:25]/(60*60)
numnames <- as.Date(as.character(names(day_res)))
plot(as.vector(day_res)~numnames,pch=16,
     xlab="Day",ylab="Duration (h)",main="",ylim=c(0,23))

## Plot b
# MOVEMENTS: distance/month

Vmove_M <- RunTimeProfile(T139nonresid,"STARTTIME","month")
mon_mov <- tapply(Vmove_M$DISTANCE,Vmove_M$DATETIME,mean)/1000
numnames <- as.numeric(as.character(names(mon_mov)))
movdata <- rep(0,12)
movdata[numnames] <- as.vector(mon_mov)
names(movdata)<-as.character(1:12)
plot(as.vector(movdata)[9:12]~ names(movdata)[9:12],pch=16,xaxt="n",
     xlab="Month",ylab="Min distance (km)",main="")
axis(side=1,las=1, at=seq(9,12),labels=month.name[9:12])

## Plot c
# DEPTH EVENTS: frequency/diel cycle

# Extract increasing depth sensor events for transmitter 139
T139dives <- RunSensorEventExtraction(T139,
                                      "INCREASE",
                                      "RECEIVERID",
                                      "m",
                                      0.5,
                                      (1*60*60),
                                      (60*60),
                                      0.5)
# The sensor logtable
T139divelog <- T139dives$logtable
# The sensor event file
T139diveevent <- T139dives$event
# Remove timeout events
T139diveevent<-subset(T139diveevent,T139diveevent$ENDREASON=="return")

Vdiv_C <- RunTimeProfile(T139diveevent,"STARTTIME","circadian")
cir_div <- tapply(Vdiv_C$FREQ,Vdiv_C$DATETIME,mean)
numnames <- as.numeric(as.character(names(cir_div)))
divdata <- rep(0,24)
divdata[numnames+1] <- as.vector(cir_div)
names(divdata)<-as.character(0:23)
plot(as.vector(divdata)~ names(divdata),pch=16,
     xlab="24 hr cycle",ylab="Number of depth events",main="")


## End(Not run)
```

# Index