2. Reading in an sftrack

`sftrack` objects can be read in via raw data or from `sf` or `ltraj` objects.

Loading in raw data

To create `sftrack` objects data we use the `as_sftrack()` or `as_sftraj()` function, depending on your desired output. Both have the same arguments and output but differ in the way the geometry field is calculated.

`as_sftrack()` accepts 2 kinds of raw data for each of the 4 required parts. Either a vector/list representing the data where length = nrow(data), or it accepts the column name where the data exists. For any `sftrack` component you can input either vector data or the column name for any variable.

Global options

These are options that are required regardless of which input type you use.

data - is a data.frame containing your data. At present we are reserving 'burst' as a column name, so data will be overwritten if this column name exists.

crs - the coordinate references system/projection of the data, as implemented by rgdal. see CRS-class for more information. If none is supplied crs is set as NA and can be set later using the `sf` function CRS.

active_burst - This is a vector containing what bursts are 'active'. Meaning calculations and graphing will be grouped by these bursts. Can change active_burst whenever. If no value is supplied it defaults to all bursts.

Variable inputs

Vector inputs to `as_sftrack` in general involve feeding as_sftrack the data itself where length(vector) == nrow(data). Or a list where each component adheres to this rule. And a column name must be found in the data.frame supplied by `data`. If using entirely vector inputs, `data` is not required.

burst - a list with named vectors to group the sftrack where each list item is length(vector) = nrow(data). One item must be named `id`, but otherwise can be infinite number of grouping variables. Or a vector naming the column names for burst categories.

cords - data.frame of x,y,z coordinates where column with order : `c(x, y, z)`, z is optional. NAs are allowed, alhough NAs must exist through the entire row otherwise an error is thrown. Or a vector naming the column names for coordinates in x,y,z order. **time** - a vector containing the time information, must be either POSIX or an integer where length(time) == nrow(data). Using this argument will name the time column as 'reloc_time'. Or the column name for the time column. The output object will be sorted by the time column.

error - a vector containing the error information where length(error) == nrow(data). Using this argument will name the error information as 'track_error'. Input can be singular NA, inwhich the column is filled with NAs. Or the column name for the error column

Examples (Vector)

```r
raccoon_data <- read.csv(system.file('extdata/raccoon_data.csv', package='sftrack'))

#data
data = raccoon_data
#xyz
```

```
coords = data[,c('longitude','latitude')]
crs = '+init=epsg:4326'
#bursts
burst = list(id = raccoon_data$sensor_code,month = as.POSIXlt(raccoon_data$utc_date)$mon+1)
active_burst = c('id','month')
#time
time = as.POSIXct(raccoon_data$acquisition_time, tz='EST')
#error
error = data$fix
my_sftrack <- as_sftrack(data = data, coords = coords, burst = burst,
                         active_burst = active_burst, time = time,
                         crs = crs, error = error)

head(my_sftrack)
```

```
## Sftrack with 6 features and 14 fields (3 empty geometries)
## Geometry : "geometry" (XY, crs: +init=epsg:4326)
## Timestamp : "reloc_time" (POSIXct in EST)
## Burst : "burst" (*id*, *month*)
## -------------------------------
##   sensor_code   utc_date utc_time latitude longitude height hdop vdop fix
## 1        CJ11 2019-01-19 00:02:30       NA        NA     NA  0.0  0.0  NO
## 2        CJ11 2019-01-19 01:02:30 26.06945 -80.27906      7  6.2  3.2  2D
## 3        CJ11 2019-01-19 02:02:30       NA        NA     NA  0.0  0.0  NO
## 4        CJ11 2019-01-19 03:02:30       NA        NA     NA  0.0  0.0  NO
## 5        CJ11 2019-01-19 04:02:30 26.06769 -80.27431    858  5.1  3.2  2D
## 6        CJ11 2019-01-19 05:02:30 26.06867 -80.27930    350  1.9  3.2  3D
##      acquisition_time          reloc_time sftrack_error
## 1 2019-01-19 00:02:30 2019-01-19 00:02:30            NO
## 2 2019-01-19 01:02:30 2019-01-19 01:02:30            2D
## 3 2019-01-19 02:02:30 2019-01-19 02:02:30            NO
## 4 2019-01-19 03:02:30 2019-01-19 03:02:30            NO
## 5 2019-01-19 04:02:30 2019-01-19 04:02:30            2D
## 6 2019-01-19 05:02:30 2019-01-19 05:02:30            3D
##                  burst                   geometry
## 1 (id: CJ11, month: 1)                POINT EMPTY
## 2 (id: CJ11, month: 1) POINT (-80.27906 26.06945)
## 3 (id: CJ11, month: 1)                POINT EMPTY
## 4 (id: CJ11, month: 1)                POINT EMPTY
## 5 (id: CJ11, month: 1) POINT (-80.27431 26.06769)
## 6 (id: CJ11, month: 1)  POINT (-80.2793 26.06867)
```

As you can see in this case the data is not overwritten, but extra columns added with the correct data.

---

data.frame inputs

Data.frame inputs generally describe the columns in the data that represent each field. In the case of using data.frame inputs the columns are cbinded to `data`. Therefore you may experience duplicate columns if you did not subset appropriately.

Examples (data.frame inputs)

```
data$time <- as.POSIXct(data$acquisition_time, tz='EST')
data$month <- as.POSIXlt(data$acquisition_time)$mon+1

coords = c('longitude','latitude')
burst = c(id = 'sensor_code', month = 'month')
time = 'time'
error = 'fix'

my_sftraj <- as_sftraj(data = data, coords = coords, burst = burst, time = time, error = error)

head(my_sftraj)
```

```
## Sftraj with 6 features and 14 fields (3 empty geometries)
## Geometry : "geometry" (XY, crs: NA)
## Timestamp : "time" (POSIXct in EST)
## Burst : "burst" (*id*, *month*)
## -------------------------------
##   sensor_code   utc_date utc_time latitude longitude height hdop vdop fix
## 1        CJ11 2019-01-19 00:02:30       NA        NA     NA  0.0  0.0  NO
## 2        CJ11 2019-01-19 01:02:30 26.06945 -80.27906      7  6.2  3.2  2D
## 3        CJ11 2019-01-19 02:02:30       NA        NA     NA  0.0  0.0  NO
## 4        CJ11 2019-01-19 03:02:30       NA        NA     NA  0.0  0.0  NO
## 5        CJ11 2019-01-19 04:02:30 26.06769 -80.27431    858  5.1  3.2  2D
## 6        CJ11 2019-01-19 05:02:30 26.06867 -80.27930    350  1.9  3.2  3D
##      acquisition_time                time month             burst
## 1 2019-01-19 00:02:30 2019-01-19 00:02:30     1 (id: CJ11, month: 1)
## 2 2019-01-19 01:02:30 2019-01-19 01:02:30     1 (id: CJ11, month: 1)
## 3 2019-01-19 02:02:30 2019-01-19 02:02:30     1 (id: CJ11, month: 1)
## 4 2019-01-19 03:02:30 2019-01-19 03:02:30     1 (id: CJ11, month: 1)
## 5 2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11, month: 1)
## 6 2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11, month: 1)
##                         geometry
## 1                    POINT EMPTY
## 2     POINT (-80.27906 26.06945)
## 3                    POINT EMPTY
## 4                    POINT EMPTY
## 5 LINESTRING (-80.27431 26.06...
## 6 LINESTRING (-80.2793 26.068...
```

Conversion mode

`as_sftrack()` and `as_sftraj()` also accept other data types, and the arguments differ depending on the class. It currenly accepts, `sf`, `ltraj`, and eventually `tibbles`.

Import from ltraj

For an ltraj all you need is the ltraj object, all relevant information is taken from the object. The burst as defined in an ltraj is slightly different than in an sftrack, so it assumes the ltraj ‘burst’ is the id field of the sftrack object.

```
library(adehabitatLT)

ltraj_df <- as.ltraj(xy=raccoon_data[,c('longitude','latitude')], date = as.POSIXct(raccoon_data$acquisi
 id = raccoon_data$sensor_code, typeII = TRUE,
 infolocs = raccoon_data[,1:6] )

my_sf <- as_sftrack(ltraj_df)
head(my_sf)
```

```
## Sftrack with 6 features and 11 fields (3 empty geometries)
## Geometry : "geometry" (XY, crs: NA)
## Timestamp : "reloc_time" (POSIXct in no timezone)
## Burst : "burst" (*id*)
## -------------------------------
##           x        y      burst          reloc_time sensor_code   utc_date
## 1        NA       NA (id: CJ11) 2019-01-19 00:02:30        CJ11 2019-01-19
## 2 -80.27906 26.06945 (id: CJ11) 2019-01-19 01:02:30        CJ11 2019-01-19
## 3        NA       NA (id: CJ11) 2019-01-19 02:02:30        CJ11 2019-01-19
## 4        NA       NA (id: CJ11) 2019-01-19 03:02:30        CJ11 2019-01-19
## 5 -80.27431 26.06769 (id: CJ11) 2019-01-19 04:02:30        CJ11 2019-01-19
## 6 -80.27930 26.06867 (id: CJ11) 2019-01-19 05:02:30        CJ11 2019-01-19
##   utc_time latitude longitude height                   geometry
## 1 00:02:30       NA        NA     NA                POINT EMPTY
## 2 01:02:30 26.06945 -80.27906      7 POINT (-80.27906 26.06945)
## 3 02:02:30       NA        NA     NA                POINT EMPTY
## 4 03:02:30       NA        NA     NA                POINT EMPTY
## 5 04:02:30 26.06769 -80.27431    858 POINT (-80.27431 26.06769)
## 6 05:02:30 26.06867 -80.27930    350  POINT (-80.2793 26.06867)
```

sf objects

sf objects are handled similiarly to the standard raw data, except you do not need to input any information about the coordinates or projection.

```
library(sf)
```

```
## Linking to GEOS 3.7.0, GDAL 2.4.0, PROJ 5.2.0
```

```
df1 <- data[!is.na(raccoon_data$latitude),]
sf_df <- st_as_sf(df1, coords=c('longitude','latitude'), crs = crs)
burst = c(id = 'sensor_code')
time_col = 'time'

new_sftraj <- as_sftraj(sf_df,burst = burst, time = time_col)
head(new_sftraj)
```

```
## Sftraj with 6 features and 12 fields (0 empty geometries)
## Geometry : "geometry" (XY, crs: +init=epsg:4326)
## Timestamp : "time" (POSIXct in EST)
## Burst : "burst" (*id*)
## -------------------------------
##    sensor_code   utc_date utc_time height hdop vdop fix
## 2         CJ11 2019-01-19 01:02:30      7  6.2  3.2  2D
## 5         CJ11 2019-01-19 04:02:30    858  5.1  3.2  2D
## 6         CJ11 2019-01-19 05:02:30    350  1.9  3.2  3D
## 7         CJ11 2019-01-19 06:02:30     11  2.3  4.5  3D
## 8         CJ11 2019-01-19 07:02:04      9  2.7  3.9  3D
## 10        CJ11 2019-01-19 17:02:30     NA  2.0  3.3  3D
##       acquisition_time                time month      burst
## 2  2019-01-19 01:02:30 2019-01-19 01:02:30     1 (id: CJ11)
## 5  2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11)
## 6  2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11)
## 7  2019-01-19 06:02:30 2019-01-19 06:02:30     1 (id: CJ11)
## 8  2019-01-19 07:02:04 2019-01-19 07:02:04     1 (id: CJ11)
## 10 2019-01-19 17:02:30 2019-01-19 17:02:30     1 (id: CJ11)
##                           geometry
## 2  LINESTRING (-80.27906 26.06...
## 5  LINESTRING (-80.27431 26.06...
## 6  LINESTRING (-80.2793 26.068...
## 7  LINESTRING (-80.27908 26.06...
## 8  LINESTRING (-80.27902 26.06...
## 10 LINESTRING (-80.279 26.0698...
```

```r
new_sftrack <- as_sftrack(sf_df,burst = burst, time= time_col)
head(new_sftrack)
```

```
## Sftrack with 6 features and 12 fields (0 empty geometries)
## Geometry : "geometry" (XY, crs: +init=epsg:4326)
## Timestamp : "time" (POSIXct in EST)
## Burst : "burst" (*id*)
## -------------------------------
##    sensor_code   utc_date utc_time height hdop vdop fix
## 2         CJ11 2019-01-19 01:02:30      7  6.2  3.2  2D
## 5         CJ11 2019-01-19 04:02:30    858  5.1  3.2  2D
## 6         CJ11 2019-01-19 05:02:30    350  1.9  3.2  3D
## 7         CJ11 2019-01-19 06:02:30     11  2.3  4.5  3D
## 8         CJ11 2019-01-19 07:02:04      9  2.7  3.9  3D
## 10        CJ11 2019-01-19 17:02:30     NA  2.0  3.3  3D
##       acquisition_time                time month      burst
## 2  2019-01-19 01:02:30 2019-01-19 01:02:30     1 (id: CJ11)
## 5  2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11)
## 6  2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11)
## 7  2019-01-19 06:02:30 2019-01-19 06:02:30     1 (id: CJ11)
## 8  2019-01-19 07:02:04 2019-01-19 07:02:04     1 (id: CJ11)
## 10 2019-01-19 17:02:30 2019-01-19 17:02:30     1 (id: CJ11)
##                       geometry
## 2  POINT (-80.27906 26.06945)
## 5  POINT (-80.27431 26.06769)
## 6   POINT (-80.2793 26.06867)
## 7  POINT (-80.27908 26.06962)
## 8  POINT (-80.27902 26.06963)
## 10   POINT (-80.279 26.06982)
```

Inter-class conversion

Additionally `as_sftrack` and `as_sftraj` can convert back and forth between each other with no loss in information.

```r
# Make tracks from raw data
coords = c('longitude','latitude')
burst = c(id = 'sensor_code', month = 'month')
time = 'time'
error = 'fix'

my_sftraj <- as_sftraj(data = data, coords = coords, burst = burst, time = time, error = error)
my_sftrack <- as_sftrack(data = data, coords = coords, burst = burst, time = time, error = error)

# Convert between types
new_sftrack <- as_sftrack(my_sftraj)
#head(new_sftrack)
new_sftraj <- as_sftraj(my_sftrack)
#head(new_sftraj)

identical(my_sftraj,new_sftraj)
```

```
## [1] TRUE
```

```r
identical(my_sftrack,new_sftrack)
```

```
## [1] TRUE
```

A common issue with movement data is when duplicated gps time stamps are logged. When sftrack comes into contact with these records, it returns an error:

```r
data$time[1] <- data$time[2]
 try(as_sftrack(data = data, coords = coords, burst = burst, time = time, error = error))
```

```
## Error in dup_timestamp(time = data[[time_col]], x = burst) :
##   bursts: CJ11_1 have duplicated time stamps
```

It can be hard to identify which records are duplicated. You can use the which_duplicated function to check your inputs:

```r
which_duplicated(data = data , burst = burst, time = time)
```

```
##                      burst_time row
## 1 CJ11_1 | 2019-01-19 01:02:30   1
## 2 CJ11_1 | 2019-01-19 01:02:30   2
```