loadcell

`loadcell` package provides tools for cleaning and parsing haul and endline data from load cell vertical line data collected for the Maine Department of Marine Resources.

Installing loadcell

`loadcell` is a non-CRAN package, and requires `devtools` for installation.

First, install `devtools` with the following command:

```
install.packages('devtools')
```

Then, install the load cell package directly from Bill's GitHub account with the following command:

```
## Install LoadCell from Bill's GitHub
devtools::install_github(repo = "bdevoe/loadcell",
                         auth_token = "de3bbc4b1df6ffeb03127b023edf68cd8b9e1755")
```

Basic function use

Loading CSVs

A sample CSV of actual load cell data has been included in the package directory and will be used in this example. To load load cell CSV files from a directory, the `load_csvs` function can be used. The function will scan the current working directory, or optionally a directory can be passed with the `dir` argument. Directories are scanned recursively (CSVs in subdirectories will be included). In this example, my working directory is set to `example_data` in the package directory, so the single example CSV `17348465-2018.11.15-09.30.24-20.csv` will be loaded. The CSV is loaded to an object `csv_data` - using the `str` function will show the structure of this object. For more details, see the function documentation.

```
  # Set wd to example_data folder
setwd("X:\\Bio Monitor\\LoadCells\\R\\loadcell\\example_data")
# Load loadcell
library(loadcell)
# Load CSV data
csv_data <- load_csvs()
# Examine
str(csv_data)
#> List of 1
#>  $ 17348481-2018.11.30-09.58.27-15.csv:List of 9
#>   ..$ data    :'data.frame': 17228 obs. of  4 variables:
#>   .. ..$ TimeStamp: chr [1:17228] "2018-11-30-09:58:27.538" "2018-11-30-
09:58:27.877" "2018-11-30-09:58:28.220" "2018-11-30-09:58:28.554" ...
#>   .. ..$ Seconds  : num [1:17228] 0.35 0.69 1.03 1.37 1.72 ...
#>   .. ..$ Load     : int [1:17228] 40 40 36 35 34 34 35 39 39 38 ...
#>   .. ..$ Raw      : num [1:17228] 0.0216 0.0215 0.0202 0.0199 0.0197 ...
#>   ..$ sn      : chr "17348481"
#>   ..$ traps   : num 15
#>   ..$ start_dt: POSIXlt[1:1], format: "2018-11-30 09:58:27"
#>   ..$ end_dt  : POSIXlt[1:1], format: "2018-11-30 11:37:37"
#>   ..$ seconds : num 5950
#>   ..$ max_load: int 1764
#>   ..$ min_load: int 28
#>   ..$ kfactor : num 0
#>  - attr(*, "class")= chr "LoadCellData"
```

Adjusting load values

Load values can be adjusted for the angle of the line through the block by applying a load multiplier.

```
  # Adjust load values
csv_data_adj <- adjust_load(data = csv_data, kfactor = 0.6)
```

Parsing Hauls

After CSVs are loaded, hauls can be parsed. The `parse_hauls` function handles this and has several options for how hauls should be delimited from load cell data. More details on these parameters is in the function help. In this example, `split = T` is being used to indicate the CSVs need to be split into separate hauls. If a fisherman created a separate CSV file for each haul, `split = F` would be used to simply number the hauls.

```
  # Parse hauls from LoadCellData class object
haul_data <- parse_hauls(csv_data_adj, split = T, # Split hauls within CSV
                         min_load = 40, # Remove data < 40LBF
                         min_time = 30, # Minimum 30 seconds haul length
                         min_gap = 30, # Minimum 30 seconds between hauls
                         pass = 30) # If a CSV is less than 30 seconds long,
                                    # assume it is a single haul and do not split
# Examine the first haul structure
str(haul_data[[1]])
#> List of 10
#>  $ haul    : num 1
#>  $ data    :'data.frame':    1460 obs. of  5 variables:
#>   ..$ TimeStamp: chr [1:1460] "2018-11-30-10:04:37.809" "2018-11-30-
10:04:38.154" "2018-11-30-10:04:38.501" "2018-11-30-10:04:38.838" ...
#>   ..$ Seconds  : num [1:1460] 371 371 371 372 372 ...
#>   ..$ Load     : num [1:1460] 44.4 54.6 69.6 74.4 74.4 ...
#>   ..$ Raw      : num [1:1460] 0.0318 0.0367 0.0443 0.0466 0.0468 ...
#>   ..$ Index    : int [1:1460] 1 2 3 4 5 6 7 8 9 10 ...
#>  $ sn      : chr "17348481"
#>  $ traps   : num 15
#>  $ start_dt: POSIXlt[1:1], format: "2018-11-30 10:04:37"
#>  $ end_dt  : POSIXlt[1:1], format: "2018-11-30 10:13:06"
#>  $ seconds : num 509
#>  $ max_load: num 1058
#>  $ min_load: num 42
#>  $ kfactor : num 0.6
```

Looking at the structure of `haul_data`, note that the single CSV has been split into 15 separate hauls based on the defined arguments.

Peak Analysis

Peaks analysis of haul data can be done with the `parse_peaks` function. This function will attempt to conduct peak analysis on each haul in an object of class `LoadCellHauls` created by the `parse_hauls` function. For more details on the arguments that can be input into this function, see the function help.

The goal of peak analysis is to identify the first major peak in the data that would typically occur when the first trap in a trawl reached the hauler. This first peak represents that maximum load on the vertical line that occured for the haul.

```
  # Parse peaks from LoadCellHauls class object
peaks <- parse_peaks(data = haul_data,
                     span=.05, # Percent smoothing to apply
                     peakdist=10, # Minimum 10 samples between peaks
                     peakheight=200) # Minimum peak height of 200 LBF
# Examine third haul peak analysis
str(peaks[[3]])
#> List of 18
#>  $ haul            : num 3
#>  $ data            :'data.frame':    1364 obs. of  5 variables:
#>   ..$ TimeStamp: chr [1:1364] "2018-11-30-10:36:06.837" "2018-11-30-
10:36:07.186" "2018-11-30-10:36:07.537" "2018-11-30-10:36:07.866" ...
#>   ..$ Seconds  : num [1:1364] 2260 2260 2260 2261 2261 ...
#>   ..$ Load     : num [1:1364] 43.8 51 68.4 81 97.2 ...
#>   ..$ Raw      : num [1:1364] 0.0314 0.035 0.0437 0.0499 0.058 ...
#>   ..$ Index    : int [1:1364] 1 2 3 4 5 6 7 8 9 10 ...
#>  $ sn              : chr "17348481"
#>  $ traps           : num 15
#>  $ start_dt        : POSIXlt[1:1], format: "2018-11-30 10:36:06"
#>  $ end_dt          : POSIXlt[1:1], format: "2018-11-30 10:43:59"
#>  $ seconds         : num 473
#>  $ max_load        : num 996
#>  $ min_load        : num 41.4
#>  $ kfactor         : num 0.6
#>  $ span            : num 0.05
#>  $ peakdist        : num 10
#>  $ peakheight      : num 200
#>  $ peak_analysis   : logi TRUE
#>  $ smoothed_peaks  :'data.frame':    15 obs. of  2 variables:
#>   ..$ Index: num [1:15] 213 276 505 336 672 ...
#>   ..$ Load : num [1:15] 791 713 657 579 531 ...
#>  $ smoothed_valleys:'data.frame':    24 obs. of  2 variables:
#>   ..$ Index: num [1:24] 1314 1238 467 547 1258 ...
#>   ..$ Load : num [1:24] 81.2 130.3 150.1 163.1 171.4 ...
#>  $ smoothed        :'data.frame':    1364 obs. of  2 variables:
#>   ..$ Index: int [1:1364] 1 2 3 4 5 6 7 8 9 10 ...
#>   ..$ Load : num [1:1364] 44.6 56.4 68 79.2 90.1 ...
#>  $ actual_peaks    :'data.frame':    15 obs. of  2 variables:
#>   ..$ Load : num [1:15] 996 865 752 715 778 ...
#>   ..$ Index: int [1:15] 221 288 515 326 677 910 761 1145 829 168 ...
```

Plotting data

Peaks can now be plotted with the `plot_hauls` function. Plots can be output to either the console or a PDF file. In this example, the hauls will be output to a PDF in the package folder called `TEST_PLOTS.pdf`

```
  # Set wd to example_output folder
setwd("X:\\Bio Monitor\\LoadCells\\R\\loadcell\\example_output")
# Build plots for hauls
plots <- plot_hauls(data = peaks, fileout = "TEST_PLOTS")
```

Exporting parsed data to CSVs

The `export_loadcell` function will export an object of class `LoadCellHauls` or `LoadCellPeaks` to a series of CSVs. A more in depth description of output is included in the function help. In this example, the data has been exported to a series of CSVs in the `example_output` folder in the package directory.

```
  # Set wd to example_output folder
setwd("X:\\Bio Monitor\\LoadCells\\R\\loadcell\\example_output")
# Export
export_loadcell(peaks, prefix = "TEST")
#> [1] TRUE
```