ColonyTrack analysis

Introduction

The ColonyTrack package provides a convenient workflow for the analysis of timestamped RFID[1] data from the ColonyRack system. The raw output of the ColonyRack (a collection of CSV files) is read in, together with several metadata files (see the following section), to create a data object containing processed data in a format ready for analysis. In a second step, a range of features/metrics[2] are calculated. These metrics can be used directly for visualisation, machine learning or for downstream analysis. The ColonyTrack package also includes several helpful functions for visualisation of the data—useful for quality control of the data and for creating figures for publication.

Collecting the necessary data files

The first step in processing an experiment is to collect the files required. As well as the raw data files generated by the ColonyRack system, several metadata files are required to properly define the experimental conditions of your project.

- **Raw CSV files** The raw data consist of the files in CSV format that are output by the ColonyRack system. Typically, there will be one file for each day of recording. These files will be passed to the `read_data` function as a list of file paths. If the files have been stored in single data directory (which we recommend for simplicity), this list can be easily generated using the R command `list.files(data_directory, full.names = TRUE)`, where 'data_directory' is the path of the data storage directory containing the CSV files.
- **Subject description** A tab-delimited file describing the subjects and their RFID tags, as well as any optional metadata about the subjects.
- **Cage layout** A tab-delimited file describing the cage network used in the experiment.
- **Events** A tab-delimited file specifying the light cycle used in the experiment.
- **Cage quality** This is optional and currently not used, so can be ignored in the current software version.

The format and contents of these files are described in detail in the document Input files.

The data object

Preparation of a `colonytrack_data` object

Once all of the data and metadata files have been collated, the next step is to read the files and perform pre-processing to bring the data into a usable format. This is the job of the function `read_data`. The files collected in the previous step are read in together with a list of the raw CSV files that were generated by the ColonyRack.

[1] Radio-frequency identification. Subjects carry chips/tags which are detected when the subject moves past an antenna. In the ColonyRack system, antennae are placed around the tunnels joining cages, so the transition from one cage to another can be detected.

[2] We prefer the generic term 'metrics' to describe the calculated variables, although 'features' (a term common in the machine learning field) is used interchangeably. Whereas the raw data consists of just timestamped RFID antenna contacts, the 'number of contacts per hour' would be an example of a metric. See the ColonyTrack metrics description for examples of the metrics calculated.

```
require(ColonyTrack)
dataFiles = list("example_data/RawData.csv")
subjectFile = "example_data/SubjectInfo.tsv"
networkFile = "example_data/CageNetwork.tsv"
eventsFile = "example_data/Events.tsv"

data = read_data(dataFiles, subjectFile, networkFile, eventsFile)
#>   Reading tracking data...
#>   Resolving subject ids...
#>   Checking timestamp chronology...
#>   Calculating trajectories...
```

```
class(data)
#> [1] "colonytrack_data"
```

The resulting object is of the class `colonytrack_data` and may be quite large—depending on the number of subjects[3] in the experiment and the length of tracking.

This small example of one day of data is only about 8 Mb.

```
format(object.size(data), units = "auto")
#> [1] "7 Mb"
```

Printing the data object shows a brief summary. Because the recording times do not perfectly align with the start and end of the light cycle, there will usually be an extra day at the start and finish of the experiment, which do not contain a full day's worth of tracking data. These 'padding' days can be trimmed before analysis.

```
print(data)
#>   A 'colonytrack_data' object containing data for 10 subjects over 3 days.
```

You will probably want to save the processed data object to disc. This is easily done with the `save()` command, which creates an `RData` archive. This archive can be quickly read back into your R environment later with the command `load()`.

```
save(data, file = "ExampleData.RData")
```

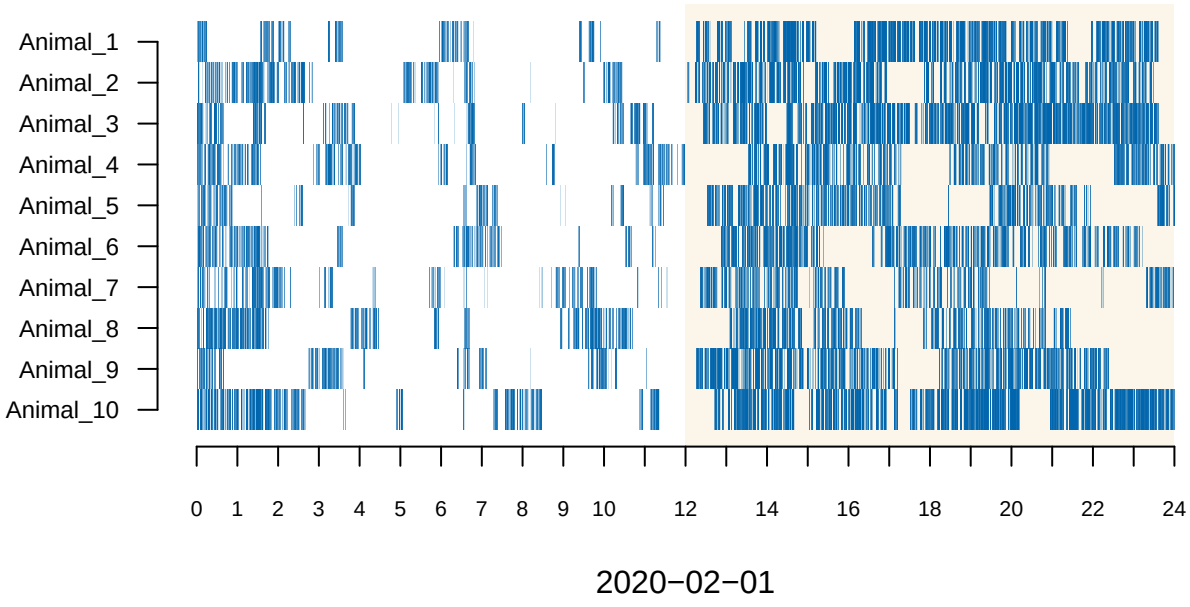
Checking the data

Before proceeding with analysis, the data should be inspected for any potential problems.

An actogram can be plotted for a quick visual overview of the activity data. In these plots, each cage transition (when an animal moves from one cage to another) is drawn as a faint blue line. On the x-axis is 'Zeitgeber time' (ZT) in hours. ZT = 0 is when the lights come on and ZT = 12 when the lights go off. The dark period of the light/dark cycle (when nocturnal animals like mice are most active) from ZT12–ZT24 is shown shaded in pale yellow. Here, just one of the days (day 2) was plotted to the screen.

```
plot(data, days = 2)
```

[3]The terms 'subject' and 'animal' are used interchangeably throughout this document. The package prefers the more generic term 'subject'.



We can also plot actograms for every day and write these to a multi-page PDF document.

```
plot(data, days = "all", file = "Actogram.pdf")
```

Inspecting metadata

The `colonytrack_data` object also contains a range of additional information. We will just look at a few of the embedded metadata elements here.

The `info` element has information about the animals and the times[4] for which they were tracked.

```
data$info$subjects
#>  [1] "Animal_1"  "Animal_2"  "Animal_3"  "Animal_4"  "Animal_5"
#>  [6] "Animal_6"  "Animal_7"  "Animal_8"  "Animal_9"  "Animal_10"
data$info$nights
#>                    id      start        end
#> 2020-01-31 2020-01-31 1580497200 1580540400
#> 2020-02-01 2020-02-01 1580583600 1580626800
#> 2020-02-02 2020-02-02 1580670000 1580713200
```

The `info` element also has information about when the data were processed and the version of ColonyTrack used.

```
data$info$processed
#> [1] "2023-02-21 14:45:38 CET"
data$info$version
#> [1] "ColonyTrack 1.0.4"
```

The remaining information in the data object will be presented in a more advanced tutorial. A detailed technical description of the `colonytrack_data` object can be found in the ColonyTrack data description document.

[4] All timestamps are in Unix time.

The metrics object

Calculating metrics

Once the data have been pre-processed, they can be passed to the `calculate_metrics()` function to generate higher-level metrics.

```
days = "all"
drop.days = c("2020-01-31", "2020-02-02")
subjects = "all"
drop.subjects = NULL

metrics = calculate_metrics(data, days = days, drop.days = drop.days, subjects = subjects,
    drop.subjects = drop.subjects)
#>   Preparing data...
#>   Calculating metrics...
#>   Collating results...
#>     Total time taken: 3.23 secs
```

Notice that we can drop certain days from the metrics analysis—and this has been used to trim the partial days at the beginning and end of the data.

The resulting object is of the class `colonytrack_metrics` and, like for the data object, the print method shows a brief summary.

```
class(metrics)
#> [1] "colonytrack_metrics"

print(metrics)
#>   A 'colonytrack_metrics' object containing metrics data for 10 subjects over 1 day.
#>
#>     Animal_1: 2020-02-01 ... 2020-02-01
#>     Animal_2: 2020-02-01 ... 2020-02-01
#>     Animal_3: 2020-02-01 ... 2020-02-01
#>     Animal_4: 2020-02-01 ... 2020-02-01
#>     Animal_5: 2020-02-01 ... 2020-02-01
#>     Animal_6: 2020-02-01 ... 2020-02-01
#>     Animal_7: 2020-02-01 ... 2020-02-01
#>     Animal_8: 2020-02-01 ... 2020-02-01
#>     Animal_9: 2020-02-01 ... 2020-02-01
#>     Animal_10: 2020-02-01 ... 2020-02-01
```

Exploring the metrics object

The parts of the metrics object can be roughly grouped into three categories;

- Metadata (`info` and `development`[5]).
- Individual metrics (`features`, `individual`, `cage.use` and `ethogram`), which contain results for each of the animals on their own.
- Interaction metrics (`clustering`, `dominance` and `follow.events`), which provide information on the interactions between animals.

```
names(metrics)
#> [1] "info"          "features"      "individual"    "cage.use"
```

[5]The 'development' element just contains information for the package developers and will not be discussed further.

```
#> [5] "ethogram"      "clustering"    "dominance"     "follow.events"
#> [9] "development"
```

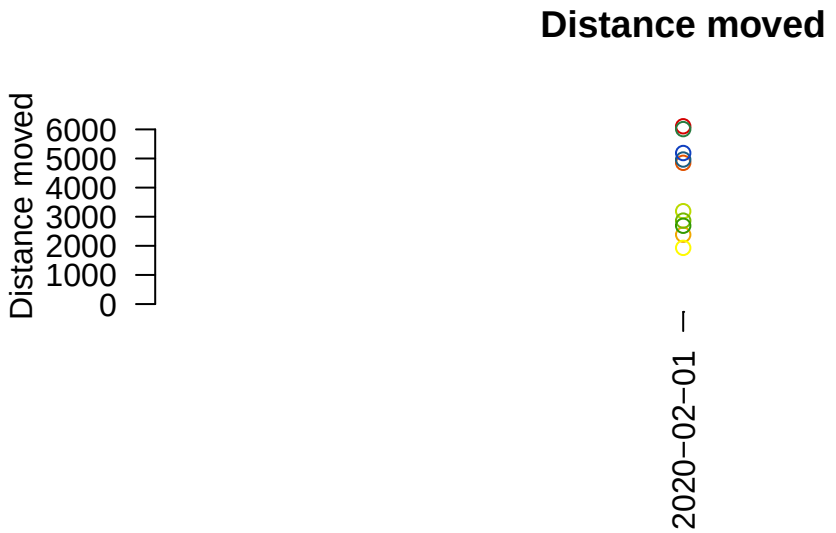
The core results are the `individual` metrics (the `features` element is a superset of this that is intended only for machine learning applications, so can be ignored for a basic analysis). These are grouped by day and then by animal/subject.

The different metrics available can be discovered via the `metrics$info$var.names` element.

```
metrics$info$var.names
#>  [1] "distance.moved"             "time.per.cage"
#>  [3] "high.activity"              "sustained.activity"
#>  [5] "cage.variability"           "cage.time.entropy"
#>  [7] "adjusted.cage.time.entropy" "cage.location.entropy"
#>  [9] "revisit.time"               "revisit.length"
#> [11] "peak.inactive"              "peak.active"
#> [13] "activity.blocks"            "cage.sharing"
#> [15] "time.alone"                 "social.interaction"
#> [17] "social.distance"            "social.gradient"
#> [19] "social.influence"           "follow.events"
#> [21] "follow.dominance"
```

The individual metrics can be plotted using a built-in convenience function. This is a good way to quickly assess the variance in your experiment and see the change over time. As our minimal example dataset only has data for one full day, the plot is shown as points. For a longer experiment, this function will produce a line plot showing the trend over time for each subject.

```
plot_metric("distance.moved", metrics)
```

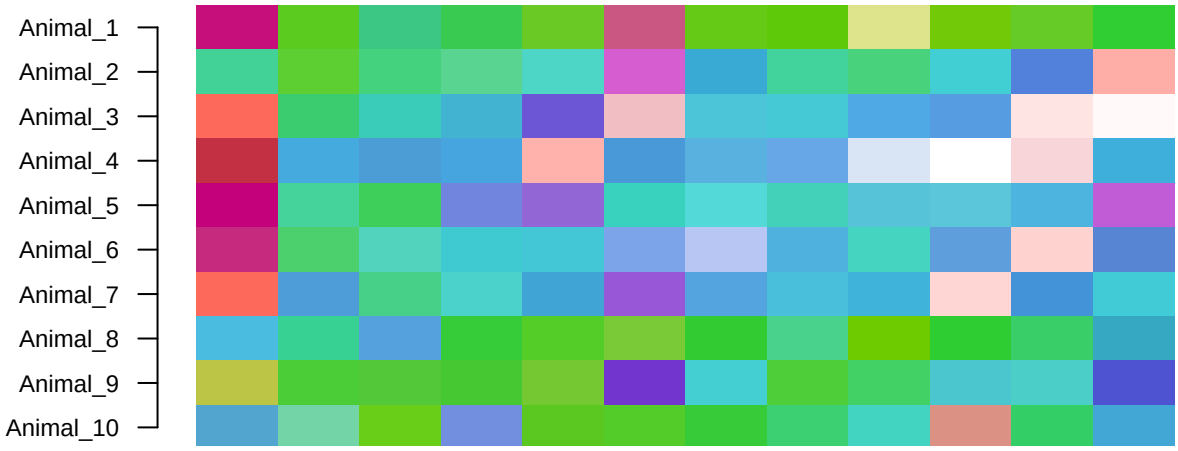


The plot function, while not intended for more complex custom analyses, actually has some flexibility built in. A separate tutorial will cover these features.

Automated ethology

Sometimes you just want to get a global view of an experiment and see what the animals are up to during the day. The ColonyTrack software performs a simple 'ethology' for the animals by selecting a representative metric from each of the three major behaviour classes; activity (how much the animals move), exploration (how the animals interact with their environment) and sociality (how the animals interact with each other).

```
plot_ethogram(metrics)
```



The colours (yellow for activity, blue for exploration and red for sociality) are shown using a subtractive colour model[6] The more advanced features of the metrics object will be presented in other tutorials. A detailed technical description of the `colonytrack_metrics` object can be found in the ColonyTrack metrics description document.

Notes

[6] A 'subtractive' colour model is what you will be used to when mixing paint—the three primary colours are yellow, blue, red and a mixture of all three is a muddy brown. This is in contrast to the 'additive' model when mixing light, or colours on a computer. This model uses the primary colours red, green, blue and a mixture of all three yields white. We felt that the subtractive model was more intuitive; especially where white (the background page colour) represents absence of all three summary metrics—as is the case when a subject is missing.