# Introduction to PITR

Joel Harding, Douglas Braun, Nicholas Burnett, Annika Putt

January 1, 2018

Passive integrated transponder (PIT) telemetry has become a universal tool to study, monitor and manage fish populations. One major challenge with PIT telemetry is data management. PIT telemetry produces large and complex datasets that can be challenging to collate and clean from raw formats without specialized expertise in data manipulation and management. To address this, we aimed to develop an efficient, standardized and user-friendly tool to manage PIT telemetry data. PITR is an open source R package that is designed to automate and standardize the process of collating, managing, summarizing and querying PIT telemetry data collected from half-duplex Oregon RFID readers. We hope that this analysis tool will provide significant time and cost savings to PIT users worldwide and standardize methods to encourage collaboration and integration of datasets from different projects and regions. Generally there are three main families of functions in PITR:

1. Data collation: these functions combine raw data from multiple readers and clean data to filter out test tags and include tags specific to a study.
2. Data management: this function allows you to restructure the configuration of readers and antennas into single or multiple arrays prior to summarizing or further query.
3. Data summary: these functions provide information on detection efficiency of antennas or arrays, direction and characteristics of movement, and plot the voltage of each reader to better inform power management. You can choose to summarize data by year, month, week, day or hour, and subset specific periods of time.

## Load the package

Load the package from GitHub:

```
install.packages("devtools")
library(devtools)
install_github("InStreamFisheries/PITR")
library(PITR)
```

## Data collation

Manually collating data can be time consuming for studies in complex river systems with large arrays and sample sizes. Consequently, the collation of data is often subject to significant human error. To avoid this, PITR allows you to collate any number of individual raw downloads (i.e., text files) located in a working directory.

Data collected from Oregon RFID readers may have different formats depending on the firmware version of the unit. Use of the `old_pit` and `new_pit` functions corresponds to data collected from earlier firmware versions (pre-April 2014) or more recent ones (post-April 2014), respectively.

If you have "old" and "new" PIT data from a study that needs to be collated, first you must define the working directory where the "old" files are located:

```
old <- "./Old PIT Data/"
```

Data files must be named in the following format 'pit_reader_mm_dd_yyyy.txt'. Note that pit_reader is the unique name of the individual reader (e.g., site name or river distance) - this populates the "reader" column in the final dataset.

Next you can run the `old_pit` function to collate the data:

```
old_dat <- old_pit(data = old, test_tags = NULL,
                   print_to_file = FALSE, time_zone = "America/Vancouver")
```

You must then define the working directory where the "new" files are located. If you only have "new" data, you would start the data collation process at this step:

```
new <- "./New PIT Data/"
```

You also have the option of removing test tags from the dataset. Test tags are often used to confirm antenna functionality and read range throughout its operation. If, for example, you had three test tags in a particular study, you can create a concatenated list of test tags to be removed from the final dataset:

```
tt <- c("900_230000010075", "900_230000010079", "900_230000010080")
```

Note that these tag codes are complete - i.e., 900_230000010075 and not 10075.

You can then run the `new_pit` function and include the `old_format_data` argument to also combine "old" data (object "old_dat" from the example above):

```
new_dat <- new_pit(data = new, test_tags = tt, print_to_file = FALSE,
                   time_zone = "America/Vancouver", old_format_data = old_dat)
```

If the `print_to_file` argument is set to TRUE, the function will export meta-data to the working directory to help confirm that data collation was performed correctly. You can also include a data frame called 'study_tags' in .csv or .txt

format (i.e., study_tags.csv or study_tags.txt) in the working directory that contains the full list of tags you want to subset out of the collated dataset. Note that the list of study tags needs to be in one column called 'study_tags'. If no data frame of study tags exists in the working directory, it is assumed that no study-specific tags need to be subsetted out of the dataset.

You can also choose to specify the time zone where data were collected in the event that it is different than the time zone in which the analysis is taking place. Note that Wikipedia has a list of time zones in the tz database.

## Data management

Depending on the initial setup of the antennas and arrays, you may have to restructure the configuration of antennas, readers and arrays after the data has been collected. You can use the `array_config` function to:

1. Combine unique readers into an array;
2. Split readers with multiple antennas into single readers; and/or
3. Rename up to four antennas on one reader or one array.

Use of this function allows you to configure antenna and array hierarchy and order in preparation for summary or further analyses. A summary of the current array, reader and antenna configuration is printed to screen after each use of the `array_config` function to inform you of the changes that have occurred.

One might have to restructure the configuration of antennas, readers and arrays to ensure that data summaries are correct and representative. An example restructuring would be a large scale project in a mainstem river and its tributaries with many arrays and tagged populations and/or species. You can isolate specific tag codes during data collation (if desired), use the `array_config` function to group antennas into arrays and then determine the detection efficiency of arrays that are encountered by a subset of tags. Using the `det_eff` function (see below), you can identify the sequence of arrays that fish encounter from downstream to upstream - i.e., array_mainstem_one, array_mainstem_two, array_mainstem_three, array_trib_one, array_trib_two, array_trib_three. Detection efficiency will be determined for these six arrays based on detections from fish swimming upstream in the mainstem river and into tributary one, tributary two and tributary three.

Let's look at a simple example of using the `array_config` function with a small dataset. First load the test dataset containing detections from a multi reader with two antennas:

```
oregon_rfid <- new_pit(data = "oregon_rfid", test_tags = NULL,
                       print_to_file = FALSE, time_zone = "America/Vancouver")

head(oregon_rfid)
```

```
##   reader antenna det_type       date    time           date_time
## 1    dam       2        D 2015-12-03 39:53.3 2015-12-03 15:39:00
## 2    dam       1        D 2015-12-03 40:35.3 2015-12-03 15:40:00
## 3    dam       1        D 2015-12-03 40:44.4 2015-12-03 15:40:00
## 4    dam       1        D 2015-12-03 40:47.6 2015-12-03 15:40:00
## 5    dam       1        D 2015-12-05 22:25.5 2015-12-05 02:22:00
## 6    dam       1        D 2015-12-05 26:25.0 2015-12-05 02:26:00
##           time_zone     dur tag_type              tag_code consec_det
## 1 America/Vancouver 00:00.2       HR 0000_0000000183783293          2
## 2 America/Vancouver 00:00.6       HR 0000_0000000183783293          4
## 3 America/Vancouver 00:00.6       HR 0000_0000000183783293          4
## 4 America/Vancouver 00:00.4       HR 0000_0000000183783293          3
## 5 America/Vancouver 00:01.6       HA      900_230000018152          9
## 6 America/Vancouver 00:03.2       HA      900_230000018152         17
##   no_empt_scan_prior
## 1                314
## 2               7694
## 3                 41
## 4                 12
## 5                  .
## 6               1180
```

You can use the `configuration` argument to `split` the data collected from the multi reader into two single readers:

```
split <- array_config(data = oregon_rfid, configuration = "split",
                      reader_name = "dam", new_reader_1_antennas = "1")
```

```
## [1] "Summary of current array, reader, and antenna configuration:"
##   array reader antenna
## 1 dam_2  dam_2       2
## 2 dam_1  dam_1       1
```

You can then recombine these two single readers into an array called "fishway" using the `configuration` argument. Note that the `array_config` function automatically assigns array names to the data frame for use in further analysis. But in this case, you want these two readers to be recombined into one array called "fishway".

```
combine <- array_config(data = split, configuration = "combine",
                        array_name = "fishway", r1 = "dam_1", r2 = "dam_2")
```

```
## [1] "Summary of current array, reader, and antenna configuration:"
##     array reader antenna
## 1 fishway  dam_2       2
## 2 fishway  dam_1       1
```

Next you can rename the antennas using the `rename_antennas` argument. One might do this in the event that antennas were not numbered sequentially from

downstream to upstream or an antenna was not wired correctly on the terminal strip of a reader. Renaming antennas allows the PITR data summary functions (e.g., direction of movement, detection efficiency) to correctly summarize the data. Let's rename A1 to A3:

```
rename_one <- array_config(data = combine, configuration = "rename_antennas",
                           array_name = "fishway", ao1 = "1", an1 = "3")
```

```
## [1] "Summary of current array, reader, and antenna configuration:"
##     array reader antenna
## 1 fishway  dam_2       2
## 2 fishway  dam_1       3
```

Then rename A2 to A4:

```
rename_two <- array_config(data = rename_one, configuration = "rename_antennas",
                           array_name = "fishway", ao1 = "2", an1 = "4")
```

```
## [1] "Summary of current array, reader, and antenna configuration:"
##     array reader antenna
## 1 fishway  dam_2       4
## 2 fishway  dam_1       3
```

Depending on the setup of the study, you may have to run the `array_config` function several times. Note that use of arguments `combine`, `split` and `rename_antennas` must be iterative. You can use the `array_config` function to restructure the dataset prior to using the summary functions listed below.

## Data summary

The PITR data summary functions are designed to help guide you through the initial exploratory phase of data analysis and provide tools for in-season management and upkeep of arrays. Data outputs from these functions can be expanded on and further analyzed to suit the needs of the study. You can use the `resolution` argument to summarize data by year, month, week, day, hour, or across the entire time period present in the dataset (if the `resolution` argument is left blank). Specific periods of time can be isolated through the use of the `start_date` and `end_date` arguments.

### Detection efficiency

You can use the `det_eff` function to compute the detection efficiency of antennas or arrays based on the array configuration and your assumption of the direction of fish movement (i.e., upstream, downstream or resident species). One application of this function might be to identify periods of time in which detection efficiency has decreased during the course of a study to help inform in-season antenna maintenance and tuning.

You can compute detection efficiency by month for fish assumed to be moving upstream:

```
month <- det_eff(data = oregon_rfid, resolution = "month",
                 by_array = FALSE, direction = "up")
```

Data output:

```
head(month)
```

```
##   array year month       date antenna detection_efficiency
## 1   dam 2015    12 2015-12-01       1                 1.00
## 2   dam 2015    12 2015-12-01       2                  NaN
## 3   dam 2016     4 2016-04-01       1                 0.67
## 4   dam 2016     4 2016-04-01       2                  NaN
## 5   dam 2016     5 2016-05-01       1                  NaN
## 6   dam 2016     6 2016-06-01       1                 1.00
##   shared_detections detections_on_array detections_not_on_array
## 1                 2                   4                       2
## 2                 0                   2                       0
## 3                 2                   2                       3
## 4                 0                   3                       0
## 5                 0                   2                       0
## 6                 2                   2                       2
##   missed_detections
## 1                 0
## 2                 0
## 3                 1
## 4                 0
## 5                 0
## 6                 0
```

Detection efficiency has been computed for each unique month for A1 based on shared and missed detections with A2. You would need an antenna upstream of A2 to compute the detection efficiency of A2. Note that in this data output detection efficiency for A1 on 2016-05-01 was not computed because there were no detections on A2.

Now compute detection efficiency by week for fish assumed to be moving upstream with a start date of 2016-10-11 08:45:00:

```
week <- det_eff(data = oregon_rfid, resolution = "week", by_array = FALSE,
                direction = "up", start_date = "2016-10-11 08:45:00")
```

Data output:

```
head(week)
```

```
##   array year month week       date antenna detection_efficiency
## 1   dam 2016    11   45 2016-10-30       1                   1
## 2   dam 2016    11   45 2016-10-30       2                 NaN
```

```
## 3   dam 2016    11   46 2016-11-06       1                   1
## 4   dam 2016    11   46 2016-11-06       2                 NaN
##   shared_detections detections_on_array detections_not_on_array
## 1                 2                   2                       2
## 2                 0                   2                       0
## 3                 1                   1                       1
## 4                 0                   1                       0
##   missed_detections
## 1                 0
## 2                 0
## 3                 0
## 4                 0
```

Here's a hypothetical example to further illustrate the `det_eff` function. Let's say you are studying an upstream migration in a river that contains two, two-antenna arrays. You must ensure that the downstream array (called "array_one") contains two antennas numbered A1 and A2, whereby A1 is the most downstream antenna in that array. Next, the upstream array (called "array_two") must contain two antennas numbered A3 and A4, whereby A3 is the most downstream antenna in that array. By doing this, the antennas from the two arrays are numbered from A1 (downstream) to A4 (upstream) and the detection efficiency for upstream migrating fish can be correctly computed by antenna or for each unique array.

Here's how you would summarize detection efficiency by array and by month for fish assumed to be moving upstream. You must list the two arrays in order from downstream to upstream in the `array_sequence` argument:

```
det_eff(data = multi_array, resolution = "month", by_array = TRUE,
        array_sequence = c("array_one", "array_two"), direction = "up")
```

### Direction of movement

You can use the `direction` function to determine the direction of movement of individual tags if there were two or more antennas deployed in a study. Data outputs from this function can highlight the stage of a migration (i.e., early or late, timing of kelting), problematic areas in fishways, locations of tagged fish relative to array(s) for mark-recapture modelling, and other applications.

Example:

```
dir <- direction(data = oregon_rfid)
```

Data output:

```
head(dir)
```

```
##     array reader antenna det_type       date    time           date_time
## 4     dam    dam       2        D 2016-04-13 25:36.2 2016-04-13 14:25:00
```

```
## 336   dam    dam       1        D 2015-12-03 40:35.3 2015-12-03 15:40:00
## 339   dam    dam       2        D 2016-04-22 42:11.3 2016-04-22 08:42:00
## 344   dam    dam       1        D 2016-05-27 07:43.9 2016-05-27 08:07:00
## 346   dam    dam       2        D 2016-06-29 34:27.6 2016-06-29 10:34:00
## 350   dam    dam       1        D 2016-06-29 22:19.1 2016-06-29 11:22:00
##         dur tag_type              tag_code consec_det no_empt_scan_prior
## 4   00:00.4       HR 0000_0000000183225426          3                  .
## 336 00:00.6       HR 0000_0000000183783293          4               7694
## 339 00:00.6       HR 0000_0000000183783293          4                109
## 344 00:00.4       HR 0000_0000000183783293          5                  .
## 346 00:01.0       HR 0000_0000000183783293         11                456
## 350 00:01.4       HR 0000_0000000183783293         15               4950
##     direction no_ant
## 4          up      1
## 336      down      1
## 339        up      1
## 344      down      1
## 346        up      1
## 350      down      1
```

### Summarize movements

You can use the `direction_total` function to summarize the direction of the first and last movements on each array over a user-defined period of time.

You can summarize movements by year:

```
dir_total <- direction_total(data = oregon_rfid, resolution = "year")
```

Data output:

```
head(dir_total)
```

```
##   array              tag_code year       date           first_det
## 1   dam 0000_0000000183225426 2016 2016-01-01 2016-04-13 14:25:00
## 2   dam 0000_0000000183783293 2015 2015-01-01 2015-12-03 15:40:00
## 3   dam 0000_0000000183783293 2016 2016-01-01 2016-05-27 08:07:00
## 4   dam      900_226000585626 2016 2016-01-01 2016-11-11 01:14:00
## 5   dam      900_226000586505 2016 2016-01-01 2016-09-20 12:38:00
## 6   dam      900_226000650315 2016 2016-01-01 2016-06-30 05:20:00
##   first_dir           last_det last_dir time_diff_days time_diff_mins
## 1        up 2016-04-13 14:25:00       up           0.00              0
## 2      down 2015-12-03 15:40:00     down           0.00              0
## 3      down 2016-08-10 11:36:00     down          75.15         108209
## 4        up 2016-11-11 01:14:00       up           0.00              0
## 5        up 2016-09-20 12:38:00       up           0.00              0
## 6        up 2016-07-11 12:29:00     down          11.30          16269
```

### First and last detections

You can use the `first_last` function to summarize the first and last detections and the difference in time (in minutes and days) between the first and last detections on each unique antenna. Such a function can assist in determining residence time between unique antennas or arrays.

For example, you can summarize data by week with a start date of 2015-10-15 08:00:00:

```
fi_la <- first_last(data = oregon_rfid, resolution = "week",
                    start_date = "2015-10-15 08:00:00")
```

Data output:

```
head(fi_la)
```

```
##   array antenna              tag_code year month week       date
## 1   dam       1 0000_0000000183225426 2016     4   15 2016-04-03
## 2   dam       1 0000_0000000183783293 2015    12   49 2015-11-29
## 3   dam       1 0000_0000000183783293 2016     5   22 2016-05-22
## 4   dam       1 0000_0000000183783293 2016     6   26 2016-06-19
## 5   dam       1 0000_0000000183783293 2016     8   32 2016-07-31
## 6   dam       1      900_226000584926 2016     9   39 2016-09-18
##             first_det           last_det time_diff_days time_diff_mins
## 1 2016-04-13 11:55:00 2016-04-13 13:47:00           0.08            112
## 2 2015-12-03 15:40:00 2015-12-03 15:40:00           0.00              0
## 3 2016-05-27 08:07:00 2016-05-27 08:07:00           0.00              0
## 4 2016-06-29 11:22:00 2016-06-29 11:33:00           0.01             11
## 5 2016-08-10 11:36:00 2016-08-10 11:36:00           0.00              0
## 6 2016-09-26 06:39:00 2016-09-26 06:47:00           0.01              8
```

### Voltage of readers

You can use the `volt_plot` function to plot the voltage of each reader over the study duration and save it to the working directory. Such information will help identify power outages or patterns of low voltage from solar- and battery-powered installations. Note that the voltage dataset (object called "volt_dat") is created using the `new_pit` function and can be managed separately by experienced users.

```
volt_plot(new_dat$volt_dat)
```

## Further resources

PITR's help document provides more details on the functions and their associated arguments. Bugs and requests can be submitted to the PITR issues page on

GitHub. We envision that PITR will evolve over time based on feedback from users, and as such we welcome any comments or suggestions.