Supplementary Material for: A Rough Guide to Pre-processing High-Frequency Animal Tracking Data

Pratik R. Gupte Christine E. Beardsworth Orr Spiegel
Emmanuel Lourie Allert Bijleveld

2020-11-16

Contents

1 Processing Egyptian fruit bat tracks

We show the pre-processing pipeline at work on the tracks of three Egyptian fruit bats (*Rousettus aegyptiacus*), and construct residence patches.

1.1 Prepare libraries

Install the required `R` libraries that are available on CRAN.

```
# load libs
library(data.table)
library(RSQLite)
library(ggplot2)
library(patchwork)

# prepare a palette
pal <- RColorBrewer::brewer.pal(4, "Set1")
```

1.2 Install `atlastools` from Github.

`atlastools` is available from Github and is archived on Zenodo (Gupte 2020). It can be installed using `remotes` or `devtools`. Here we use the `remotes` function `install_github`.

```
install.packages("remotes")
```

```
# installation using remotes
remotes::install_github("pratikunterwegs/atlastools")
```

1.3 Read bat data

Read the bat data from an `SQLite` database local file and convert to a plain text csv file. This data can be found in the "data" folder.

```
# prepare the connection
con <- dbConnect(drv = SQLite(),
                 dbname = "data/Three_example_bats.sql")

# list the tables
table_name <- dbListTables(con)

# prepare to query all tables
query <- sprintf('select * from \"%s\"', table_name)

# query the database
data <- dbGetQuery(conn = con, statement = query)

# disconnect from database
dbDisconnect(con)
```

Convert data to csv, and save a local copy in the folder "data".

```
# convert data to datatable
setDT(data)

# write data for QGIS
fwrite(data, file = "data/bat_data.csv")
```

1.4 A First Visual Inspection

Plot the bat data as a sanity check, and inspect it visually for errors (Figure 1). The plot code is hidden in the rendered copy (PDF) of this supplementary material, but is available in the `Rmarkdown` file "06_bat_data.Rmd". The saved plot is shown below as Figure 1.

1.5 Prepare data for filtering

Here we apply a series of simple filters. It is always safer to deal with one individual at a time, so we split the data.table into a list of data.tables to avoid mixups among individuals.

Prepare data per individual

```
# split bat data by tag
# first make a copy using the data.table function copy
# this prevents the orignal data from being modified by atlastools
# functions which DO MODIFY BY REFERENCE!
data_split <- copy(data)

# now split
data_split <- split(data_split, by = "TAG")
```

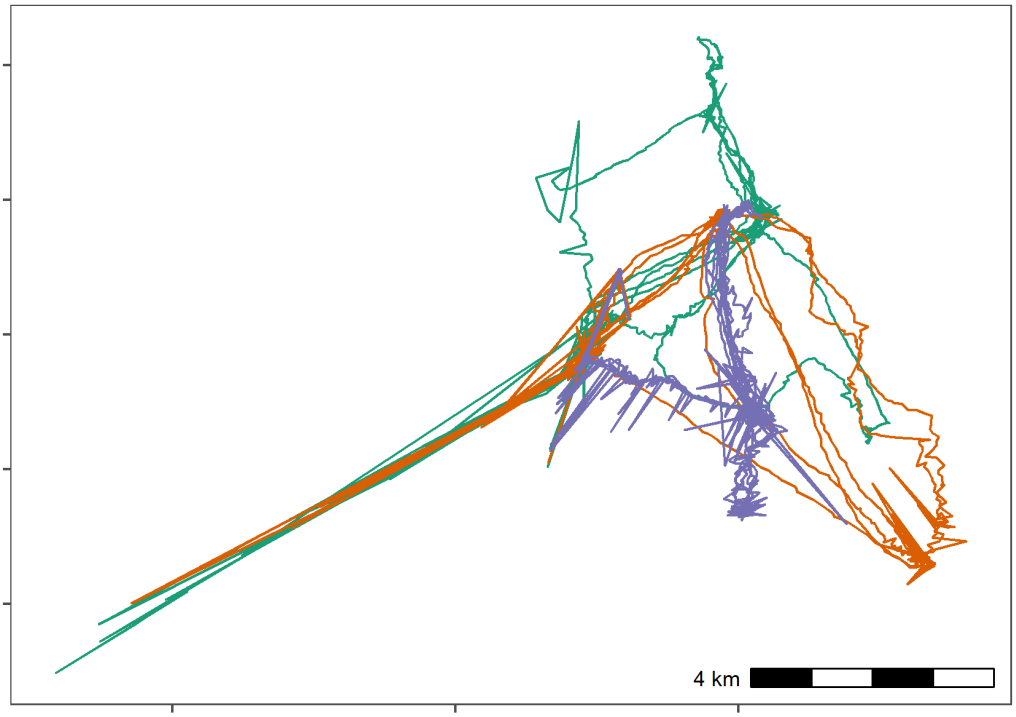

Figure 1: Movement data from three Egyptian fruit bats tracked using the ATLAS system (*Rousettus aegyptiacus*; (Toledo et al. 2020; Shohami and Nathan 2020)). The bats were tracked in the Hula Valley, Israel (33.1°N, 35.6°E), and we use three nights of tracking (5th, 6th, and 7th May, 2018), for our demonstration, with an average of 13,370 positions (SD = 2,173; range = 11,195 – 15,542; interval = 8 seconds) per individual. After first plotting the individual tracks, we notice severe distortions, making pre-processing necesary

1.6 Filter by covariates

No natural bounds suggest themselves, so instead we proceed to filter by covariates, since point outliers are obviously visible.

We use filter out positions with `SD > 20` and positions calculated using only 3 base stations, using the function `atl_filter_covariates`.

First we calculate the variable `SD`.

```
# get SD.
# since the data are data.tables, no assignment is necessary
invisible(
  lapply(data_split, function(dt) {
    dt[, SD := sqrt(VARX + VARY + (2 * COVXY))]
  })
)
```

Then we pass the filters to `atl_filter_covariates`. We apply the filter to each individual's data using an `lapply`.

```
# filter for SD <= 20
# here, reassignment is necessary as rows are being removed
# the atl_filter_covariates function could have been used here
data_split <- lapply(data_split, function(dt) {

  dt <- atl_filter_covariates(
    data = dt,
    filters = c("SD <= 20",
                "NBS > 3")

  )
})
```

Sanity check: Plot filtered data

We plot the data to check whether the filtering has improved the data (Figure 2). The plot code is once again hidden in this rendering, but is available in the source code file.

1.7 Filter by speed

Some point outliers remain (Figure 2), and could be removed using a speed filter.

First we calculate speeds, using `atl_get_speed`. We must assign the speed output to a new column in the data.table, which has a special syntax which modifies in place, and is shown below. This syntax is a feature of the `data.table` package, not strictly of `atlastools` (Dowle and Srinivasan 2020).

```
# get speeds as with SD, no reassignment required for columns
invisible(
  lapply(data_split, function(dt) {

    # first process time to seconds
    # assign to a new column
    dt[, time := floor(TIME / 1000)]

    dt[, `:=`(speed_in = atl_get_speed(dt,
                                       x = "X", y = "Y",
                                       time = "time",
```

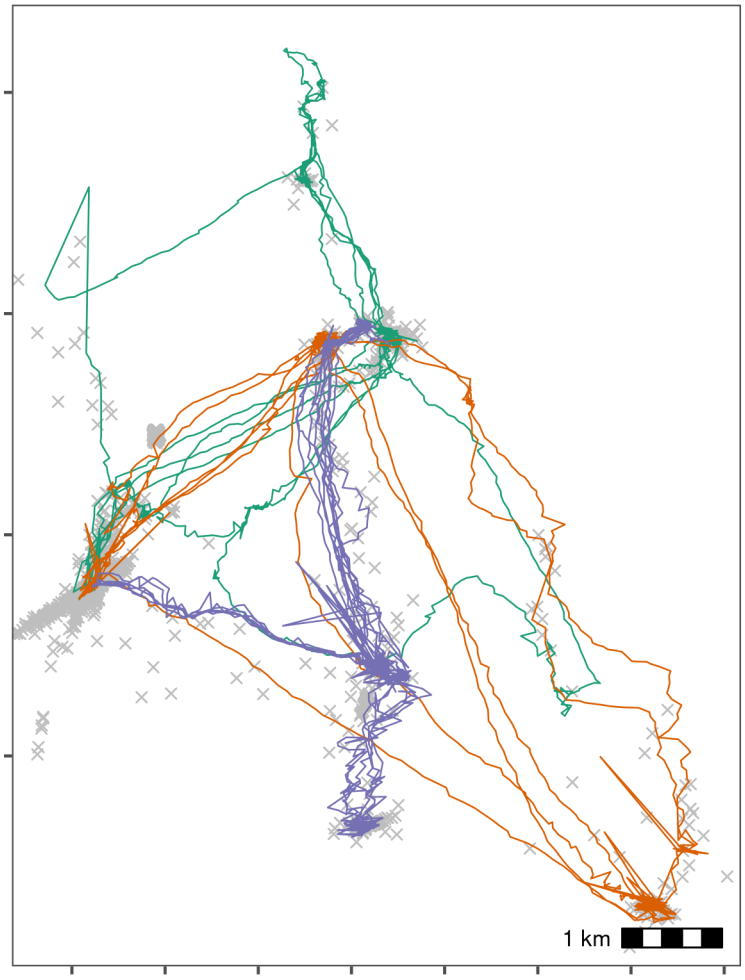

Figure 2: Bat data filtered for large location errors, removing observations with standard deviation > 20. Grey crosses show data that were removed. Since the number of base stations used in the location process is a good indicator of error (Weiser et al. 2016), we also removed observations calculated using fewer than four base stations. Both steps used the function `atl_filter_covariates`. This filtering reduced the data to an average of 10,447 positions per individual (78% of the raw data on average). However, some point outliers remain.

```
                                    type = "in"),
                speed_out = atl_get_speed(dt,
                                    x = "X", y = "Y",
                                    time = "time",
                                    type = "out"))]
  })
)
```

Now filter for speeds > 20 m/s (around 70 km/h), passing the predicate (a statement return TRUE or FALSE) to `atl_filter_covariates`. First, we remove positions which have `NA` for their `speed_in` (the first position) and their `speed_out` (last position).

```
# filter speeds
# reassignment is required here
data_split <- lapply(data_split, function(dt) {
  dt <- na.omit(dt, cols = c("speed_in", "speed_out"))

  dt <- atl_filter_covariates(data = dt,
                              filters = c("speed_in <= 20",
                                          "speed_out <= 20"))
})
```

Sanity check: Plot speed filtered data

The speed filtered data is now inspected for errors (Figure 3). The plot code is once again hidden.

1.8 Median smoothing

The quality of this data (Figure 3) is relatively high, and a median smooth is not strictly necessary. We demonstrate the application of a 5 point median smooth to the data nonetheless.

Since the median smoothing function `atl_median_smooth` modifies in place, we first make a copy of the data, using `data.table`'s `copy` function. No reassignment is required, in this case. The `lapply` function allows arguments to `atl_median_smooth` to be passed within `lapply` itself.

In this case, the same moving window K is applied to all individuals, but modifying this code to use the multivariate version `Map` allows different K to be used for different individuals. This is a programming matter, and is not covered here further.

```
# since the function modifies in place, we shall make a copy
data_smooth <- copy(data_split)

# split the data again
data_smooth <- split(data_smooth, by = "TAG")

# apply the median smooth to each list element
# no reassignment is required as THE FUNCTION MODIFIES IN PLACE!
invisible(

  # the function arguments to atl_median_smooth
  # can be passed directly in lapply

  lapply(X = data_smooth,
         FUN = atl_median_smooth,
         time = "time", moving_window = 5)
)
```

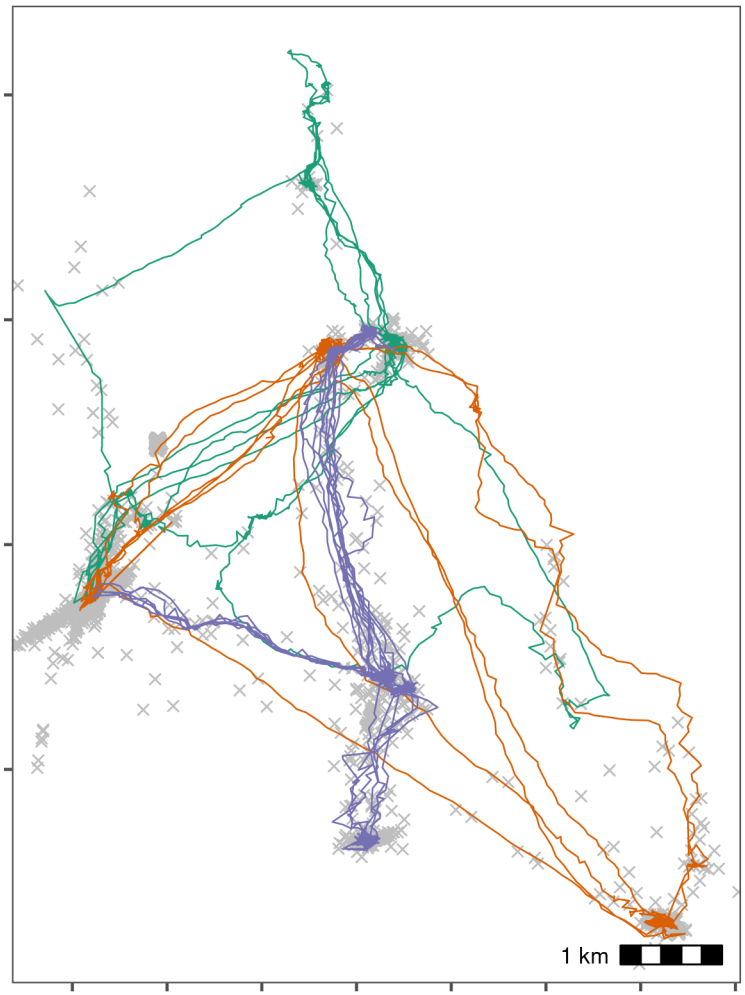

Figure 3: Bat data with unrealistic speeds removed. Grey crosses show data that were removed. We calculated the incoming and outgoing speed of each position using `atl_get_speed`, and filtered out positions with speeds > 20 m/s using `atl_filter_covariates`, leaving 10,337 positions per individual on average (98% from the previous step).

Sanity check: Plot smoothed data

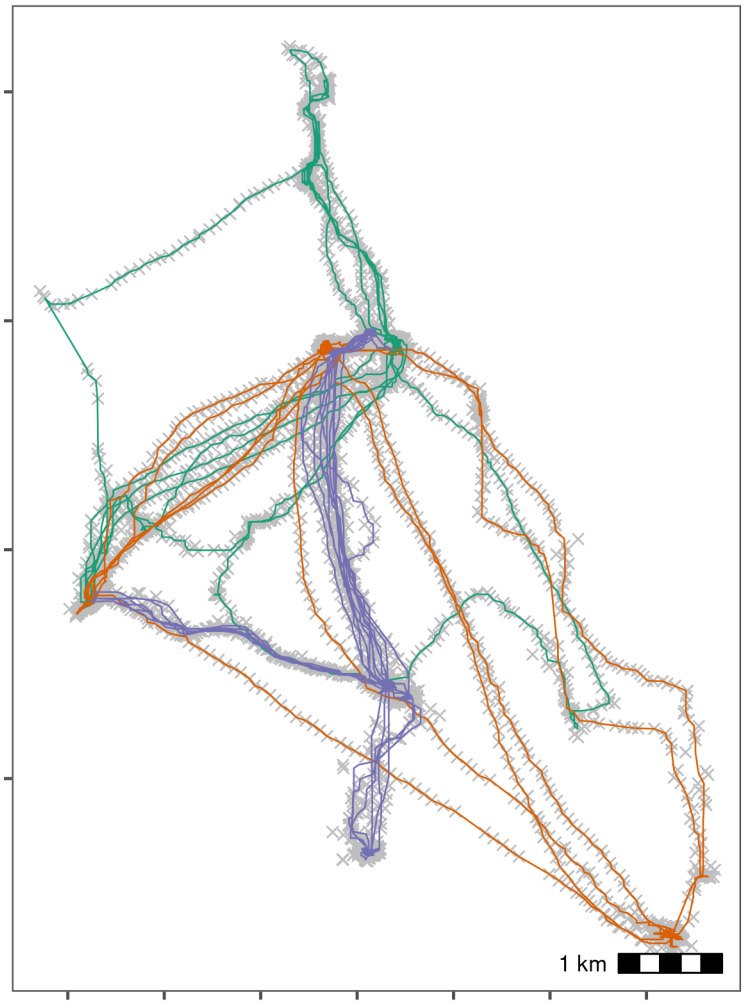

Figure 4: Bat data after applying a median smooth with a moving window $K = 5$. Grey crosses show data prior to smoothing. The smoothing step did not discard any data.

1.9 Making residence patches

Calculating residence time

First, the data is put through the `recurse` package to get residence time (Bracis, Bildstein, and Mueller 2018).

```
# load recurse
library(recurse)

# split the data
data_smooth <- split(data_smooth, data_smooth$TAG)
```

We calculated residence time, but since bats may revisit the same features, we want to prevent confusion between frequent revisits and prolonged residence.

For this, we stop summing residence times within Z metres of a location if the animal exited the area for one hour or more. The value of Z (radius, in `recurse` parameter terms) was chosen as 50m.

This step is relatively complicated and is only required for individuals which frequently return to the same location, or pass over the same areas repeatedly, and for which revisits (cumulative time spent) may be confused for residence time in a single visit.

While a simpler implementation using total residence time divided by the number of revisits is also possible, this does assume that each revisit had the same residence time.

```
# get residence times

data_residence <- lapply(data_smooth, function(dt) {
  # do basic recurse
  dt_recurse <- getRecursions(
    x = dt[, c("X", "Y", "time", "TAG")],
    radius = 50,
    timeunits = "mins"
  )

  # get revisit stats
  dt_recurse <- setDT(
    dt_recurse[["revisitStats"]]
  )

  # count long absences from the area
  dt_recurse[, timeSinceLastVisit :=
          ifelse(is.na(timeSinceLastVisit), -Inf, timeSinceLastVisit)]
  dt_recurse[, longAbsenceCounter := cumsum(timeSinceLastVisit > 60),
             by = .(coordIdx)
             ]
  # get data before the first long absence of 60 mins
  dt_recurse <- dt_recurse[longAbsenceCounter < 1, ]

  dt_recurse <- dt_recurse[, list(
    resTime = sum(timeInside),
    fpt = first(timeInside),
    revisits = max(visitIdx)
  ),
  by = .(coordIdx, x, y)
  ]

  # prepare and merge existing data with recursion data
  dt[, coordIdx := seq(nrow(dt))]

  dt <- merge(dt,
              dt_recurse[, c("coordIdx", "resTime")],
              by = c("coordIdx"))

  setorderv(dt, "time")
})
```

We bind the data together and assign a human readable timestamp column.

```
# bind the list
data_residence <- rbindlist(data_residence)

# get time as human readable
data_residence[, ts := as.POSIXct(time, origin = "1970-01-01")]
```

Constructing residence patches

Some preparation is required. First, the function requires columns `x`, `y`, `time`, and `id`, which we assign using the `data.table` syntax. Then we subset the data to only work with positions where the individual had a residence time of more than 5 minutes.

```
# add an id column
data_residence[, `:=`(id = TAG,
                      x = X, y = Y)]

# filter for residence time > 5 minutes
data_residence <- data_residence[resTime > 5, ]

# split the data
data_residence <- split(data_residence, data_residence$TAG)
```

We apply the residence patch method, using the default argument values (`lim_spat_indep = 100` (metres), `lim_time_indep = 30` (minutes), and `min_fixes = 3`). We change the `buffer_radius` to 25 metres (twice the buffer radius is used, so points must be separated by 50m to be independent bouts).

```
# segment into residence patches
data_patches <- lapply(data_residence, atl_res_patch,
                       buffer_radius = 25)
```

Getting residence patch data

We extract the residence patch data as spatial `sf-MULTIPOLYGON` objects. These are returned as a list and must be converted into a single `sf` object. These objects and the raw movement data are shown in Figure 5.

```
# get data spatials
data_spatials <- lapply(data_patches, atl_patch_summary,
                        which_data = "spatial",
                        buffer_radius = 25)

# bind list
data_spatials <- rbindlist(data_spatials)

# convert to sf
library(sf)
data_spatials <- st_sf(data_spatials, sf_column_name = "polygons")

# assign a crs
st_crs(data_spatials) <- st_crs(2039)
```

Write patch spatial representations

```
st_write(data_spatials,
         dsn = "data/data_bat_residence_patches.gpkg")
```

Write cleaned bat data.

```
data_clean <- fwrite(rbindlist(data_smooth),
                     file = "data/data_bat_smooth.csv")
```

Write patch summary.

```
# get summary
patch_summary <- lapply(data_patches, atl_patch_summary)

# bind summary
patch_summary <- rbindlist(patch_summary)

# write
fwrite(patch_summary,
       "data/data_bat_patch_summary.csv")
```

2 References

Bracis, Chloe, Keith L. Bildstein, and Thomas Mueller. 2018. "Revisitation Analysis Uncovers Spatio-Temporal Patterns in Animal Movement Data." *Ecography* 41 (11): 1801–11. https://doi.org/10.1111/ecog.03618.

Dowle, Matt, and Arun Srinivasan. 2020. *Data.Table: Extension of 'data.Frame'*. Manual.

Gupte, Pratik Rajan. 2020. "Atlastools: Pre-Processing Tools for High Frequency Tracking Data." Zenodo. https://doi.org/10.5281/ZENODO.4033154.

Shohami, David, and Ran Nathan. 2020. "Cognitive Map-Based Navigation in Wild Bats Revealed by a New High-Throughput Tracking System." Dryad. https://doi.org/10.5061/DRYAD.G4F4QRFN2.

Toledo, Sivan, David Shohami, Ingo Schiffner, Emmanuel Lourie, Yotam Orchan, Yoav Bartan, and Ran Nathan. 2020. "Cognitive MapBased Navigation in Wild Bats Revealed by a New High-Throughput Tracking System." *Science* 369 (6500): 188–93. https://doi.org/10.1126/science.aax6904.

Weiser, Adi Weller, Yotam Orchan, Ran Nathan, Motti Charter, Anthony J. Weiss, and Sivan Toledo. 2016. "Characterizing the Accuracy of a Self-Synchronized Reverse-GPS Wildlife Localization System." In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 1–12. https://doi.org/10.1109/IPSN.2016.7460662.

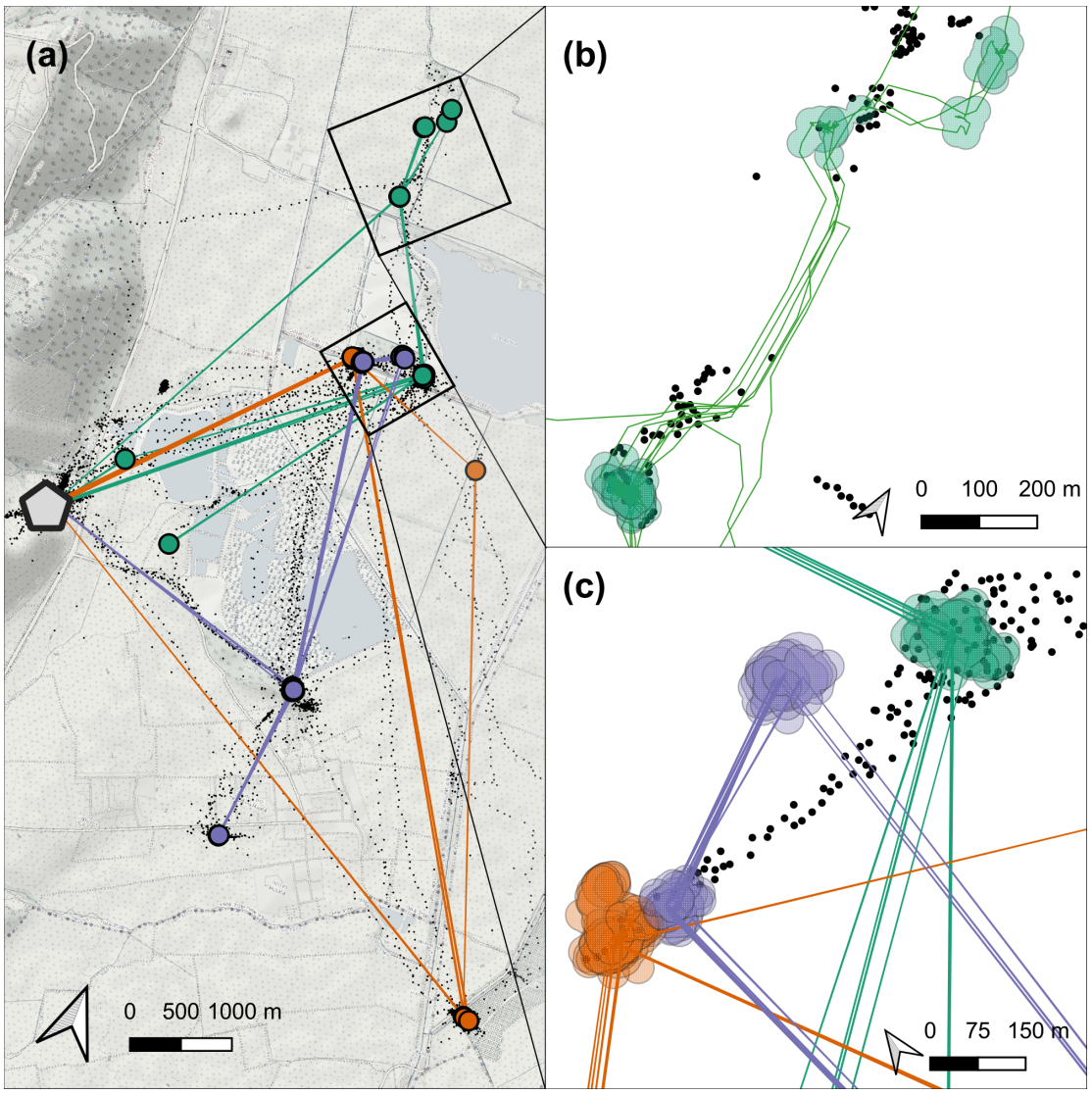

Figure 5: A visual examination of plots of the bats' residence patches and linear approximations of paths between them showed that though all three bats roosted at the same site, they used distinct areas of the study site over the three nights **(a)**. Bats tended to be resident near fruit trees, which are their main food source, travelling repeatedly between previously visited areas **(b, c)**. However, bats also appeared to spend some time at locations where no fruit trees were recorded, prompting questions about their use of other food sources **(b, c)**. When bats did occur close together, their residence patches barely overlapped, and their paths to and from the broad area of co-occurrence were not similar **(c)**. Constructing residence patches for multiple individuals over multiple activity periods suggests interesting dynamics of within- and between-individual overlap **(b, c)**.