Animtaed Home Ranges and Movement How-to V.0.5

Benjamin Michael Marshall

20/03/2018

This document includes the instructions and code to animated animal movement data using R. It is by no means comprehensive nor all that sophisticated. It does, however, cover the basics and provide you with a starting point for animating animal movement data. Included is also a dummy dataset link that will help run through the code for the first time.

Getting Started

While the R code for producing animated graphics isn't particularly complex the initial installation of the required packages is less than user-friendly. We require two main bits of software; ImageMagick and ffmpeg.

The procurement and installation of *ImageMagick*

ImageMagick can be downloaded from this page: link

Scroll down to and look at the instructions appropriate version for your operating system. For windows this is a simple process. Simply download the installer and it will take care of the entire process. During the installation make sure that the 'ffmpeg' option in ticked as that saves windows users from having to install that seperately

For Mac OSX you will need to follow the link to download MacPorts (HomeBrew that they suggest as an alternative no longer works on more modern versions of Mac OSX): link

Take care to install the correct version for the operating system you are running. If you are unsure you can check by clicking the apple top left in the menu bar and "About this Mac" will have the answers. Basically, all MacPorts does is expands the range of commands that Terminal will understand.

Next open terminal, usually found within the 'Utilities' or 'Other' folder in 'Applications'.

Enter

```
sudo port install ImageMagick
```

Terminal should start working away unpacking and installing *ImageMagick* from the relevant URL. You do not need to download the ImageMagick package the port command takes care of that for you.

The procurement and installation of ffmpeg

This step is only required for Mac OSX users as the *ImageMagick* package for windows should have done this itself.

First head to and download the appropriate version of the ffmpeg software: link

Thankfully the website auto-detects your system so you shouldn't have to worry about whether you have a 32 or 64 bit system.

Another interesting quirk of installing system level software like this on more modern Mac OSX systems is that Apple, in their infinite wisdom, decided to hide the system folders from everyday users. The easiest way to get around this is using 'Go To Folder…' under 'Go' in the menu bar when the Finder is selected. Then in that box enter

```
/usr/local/bin
```

This will open the folder you need to copy and paste all the file found in the bin folder within the ffmpeg folder you downloaded. You don't need to put them in an ffmpeg folder like the link suggests. There is further documentation found here that may help in the event of problems: link

Job done. That is the installations you need to take care of outside of R.

R code

Intial set-up

First step is to install all the necessary packages. This instal step will only need to be completed once and not at the beginning of every new session.

```r
install.packages(c("beepr", "animation", "magick", "reshape2", "ggplot2", "ggspatial",
                   "dplyr", "sp", "ggmap", "adehabitatHR", "scales", "cowplot"))
```

We then to tell R to load those packages into the current session.

```r
# beepr has functions that make a noise
library(beepr)

# These two are the packaging holding all the functions required for turning
# a series of images or figures into a proper animated .gif or .mp4 file.
library(animation)
library(magick)

# We are using reshape2 because it has the melt function that is useful
# when plotting with ggplot
library(reshape2)

# ggplot is the rather powerful and versatile plotting package for R
library(ggplot2)

# ggmap is the package that can pull maps from online sources like google or
# openmaps. Works well with ggplot2
library(ggmap)

# ggplot doesn't handle polygons with holes very well, so geom_polygon isn't
# good enough for our purposes. ggspatial has a gemo_spatial funtion that works
# better.
library(ggspatial)

# dplyr has a selection of data manipulation functions
library(dplyr)

# sp contains functions pertaining to all sorts of spatial data manipulations
# and projection
library(sp)

# adehabitatHR has all the functions that we need for home range estimations
library(adehabitatHR)

# scales expands the functionality when setting scales in ggplot,
# espeically when using dates
library(scales)

# cowplot is what we will use to plot multiple graphs in a single frame.
# It is not the most pleasant way of mapping multiple graphs but it does
# provide precsie control. I have found it very useful when trying to get
# mutliple graphs into a set resolution.
library(cowplot)
```

Now we have the required packages ready. Next is to prepare our data.

Data Preparation

For this how-to we required data that has an animal ID, the easting and northing of the animal's location, and the date and time at that location. Do not worry about what the column names are so long as they are free from special characters. We will be changing them later anyway.

Something like this:

```
##   X   Snake_ID Tracking_date Tracking_time Easting Northing
## 1 1 FakeSnake1    2017-12-01      10:00:00  817226  1606706
## 2 2 FakeSnake1    2017-12-02      11:00:00  816836  1606673
## 3 3 FakeSnake1    2017-12-03      12:00:00  816463  1606754
## 4 4 FakeSnake1    2017-12-04      13:00:00  816138  1606884
## 5 5 FakeSnake1    2017-12-05      14:00:00  815700  1606722
## 6 6 FakeSnake1    2017-12-06      15:00:00  815311  1606333
```

Make sure that information is saved as a .csv file in a location you are happy for R to work from and save to. Go ahead and tell R to use that location as your working directory. You can place the location in

```r
setwd("Your working directory here")
```

or alternatively you can use shift + command + h (on mac OSX systems) and shift + control + h (on windows systems). It can also be found under 'session' in R studio's menu bar.

So let's load in that animal data.

```r
animaldata <- read.csv("Your Data File Here.csv", header = TRUE, sep = ",")

# Make sure the data has been read in correctly by viewing the first few lines.
head(animaldata)
```

Note: the separator, denoted by sep = "," may need to be changed to ; if you are using a European computer.

Next we'll do some straight forward data cleaning. Top line will remove any line that has NA in either the easting or northing columns. The bottom two remove any line that has 0.

That's the basic data import complete, but we are planning on plotting graphs that show movement data. We shall do the calculations for that now, ready for plotting later.

We need to work out the distances between each of the animal's relocations. Cue some easy Pythagoras.

```r
attach(animaldata)  # We will attach the data at this point to simplify things here

## We create a function that gets the difference between sequential eastings (a).
diffE <- function(animaldata) {Easting[1:(length(Easting)-1)]-
    Easting[2:(length(Easting))]}
# Then we repeat that for every row.
diffE_list <- sapply(animaldata, diffE)[,1]
```

```r
# So now we have a list of the differences between each pair of eastings called
# diffE_list

## We can then repeat this for the northings (b).
diffN <- function(animaldata) {Northing[1:(length(Northing)-1)]-
    Northing[2:(length(Northing))]}
diffN_list <- sapply(animaldata, diffN)[,1]

## So we have the a and b values ready. Now to find c.
# sqrt(a^2+b^2), finding c for all
UTMdist <- sqrt((diffE_list^2) + (diffN_list^2))
# UTMdist now holds all the values between each location.

## We need to create a new column to sort these results, called Moves. But as
# we do, we need to add an additional 0 to the front of the UTMdist vector.
# This makes the vector that same length as the original locations and allow them
# to be joined. And of course we'd like the distance moved to start from 0.
animaldata$Moves <- c(0, UTMdist)

## For actual plotting we want cumulative distance not just each distance. This
# is easy calculated using the line below.
animaldata$CsumMoves <- cumsum(animaldata$Moves)
# Note we have gone right ahead and stored the cumulative moves in a new column
# called CsumMoves.
```

In terms of initial calculations we are done. However, the column names should be changed so the rest of the code can work seamlessly.

```r
# Rename your animal ID column to ID
animaldata$ID <-animaldata$your_animal_ID_here
animaldata$Tracking_time  <-  animaldata$your_time_column_name_here

## We also need to do the same with the dates.
# While we do this we can also tell R it is dealing with dates
# Beware - make sure you change the format to match your chosen date system.
animaldata$Tracking_date <- as.Date(animaldata$your_date_column_name_here,
                                    format = "%Y-%m-%d")
```

Now we have finished the instail data preparation we can move onto some spatial analysis. But before we move on detach animaldata so we don't want to accidental reference something within animaldata

```r
detach(animaldata)
```

Baseline Spatial Preparation

Before we animate anything we need define some boundaries for our maps and graphs. Some of the exact details of the next steps will differ depending on how you have collected you spatial data and what coordinate system you have used. For the purposes of this how-to we will be working from UTM coordinates and converting that into decimal degrees for ggmap. This is done via EPSG numbers, you will need to know the EPSG number for your coordinates. They can be found at http://www.epsg-registry.org link

```r
# Here we are storing the correct epsg number as an object called utm
utm <- CRS('+init=epsg:YOUR_EPSG_NUMBER_HERE')

# And this stores the number for WGS 84, the sort of go to standard for google
# maps and others.
latlong <- CRS("+init=epsg:4326")
```

Next we'll run the spatial analysis using the whole dataset to provides us with maximum values to be used in define our graph limits. If we didn't do this, ggplot would recalculate them for each frame of the animation. It is also good to test that all the packages are working.

First step is to pull out the two columns that contain your utm coordinates so we can convert them object to a spatial points object. A spatial points object can be plotted on a map and used in spatial calculations.

```
coords <- cbind(animaldata$Easting, animaldata$Northing)

## This makes the spatial points object, defining the coordinate system as utm
# as we stored earlier.
SP <- SpatialPoints(coords,proj4string = utm)
```

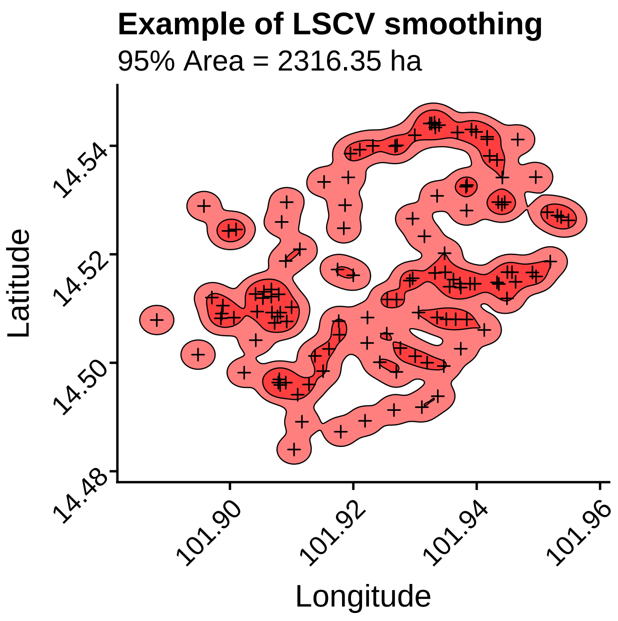
Now we have a spatial points object we can use that to calculate a Minimum Convex Polygon (MCP). The line below does this and provides an area output in hectares. You can change both the input and output units to fit your needs.

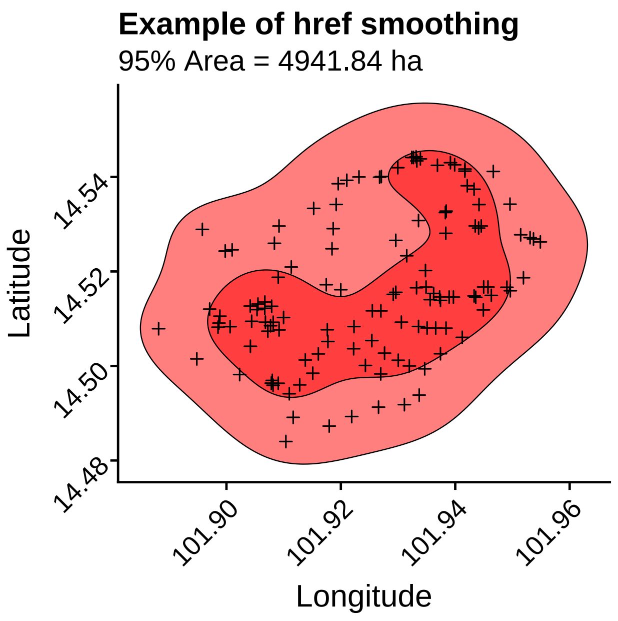
```
MCP <- mcp(SP, percent = 95, unin = "m", unout = "ha")

## We then retrieve the area of the MCP using the line below and store that for
# a little later
MCParea <- MCP$area
```

Note we are also generating a 95% MCP not a full 100%. This can be changed according to preference. As the norm is 95% we will continue with that for now.

The next few lines calculate a fixed kernel density utilisation estimation There are several aspect of the kernel estimation that can be changed here. One is more important than the others. The h value or smoothing factor can have a dramatic influence over the size and shape of the home range estimation. A higher h value will produce a larger, more generous range that is smoothed to a greater degree. In a manner of speaking it adds greater weight to each location. You have several options for h. Two are calculated from the data; href and LSCV. href tends to be generous and perhaps overestimate home range. LSCV tends to be the opposite and create a more conservative home range. The downside to LSCV is that it can easily produce ranges that are discontinuous So there is a good argument for selecting a h value by eye; trying to pick one that seems biologically feasible and relevant. However, this may need to be done per individual. And having individual h values may undermine the meaningfulness of comparisons. There has been quite a lot written about h values and their influence but there has not been one catch all solution that works for all species or scenarios.

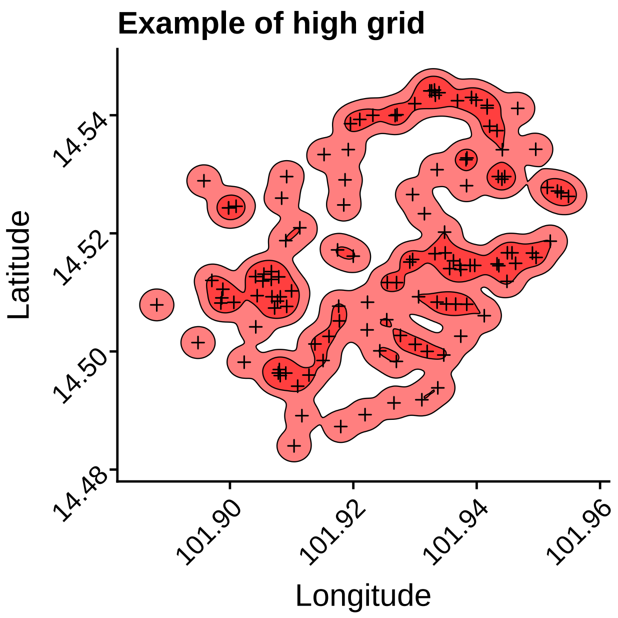
Here is an example of LSCV and href kernels based on the example dataset.

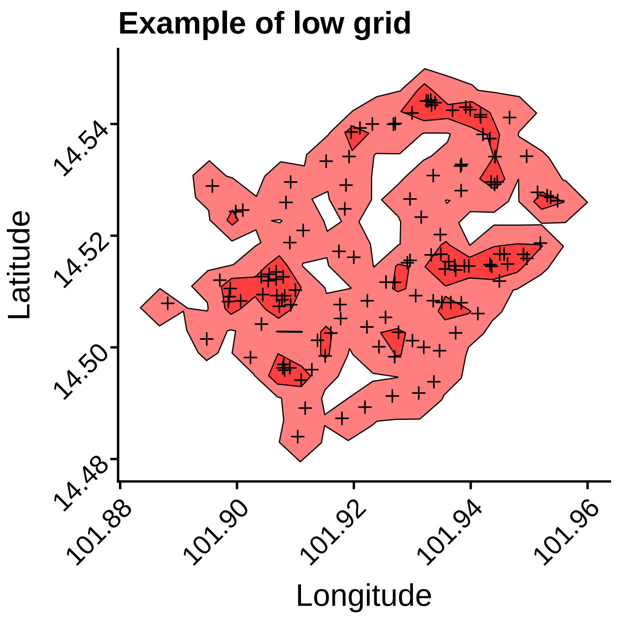




For the purposes for this example run through we will stick with using href. It will provide a good visual for the scale of the example data.

The other two aspects of the kernel estimations are grid and extent. The kernelUD function generates a raster that displays the utilisation distribution of the animal in question. Grid sets the resolution of that raster. Extent changes the 'zoom' of that grid, essentially how it is fitted to the map. There can be issues at this stage where the extent and/or grid are not sufficient to generate a raster for some datasets. This will need to be troubleshot per dataset, but beware of high grid values as they will increase the time required to generate the raster. Values very low will produce over simplified shapes. See the example below.





We can also pull the area from the kernels and MCPs. We will need to do this for the area graph in the animation. We should define the maximum y value for the graphs axis so ggplot does not recalculate it for every frame. Depending on the dataset you may just be able to use the largest area value returned from either kernel or MCP methods on the entire dataset. Other datasets may see a spike in area with only a few data points that then decreases as more are integrated into the area calculations. This is something that can only really be worked out once we see the final output. So for now we can use the more generous href to give us a starting y axis limit.

```r
# We can easily pull the area generated from the MCP here.
MCParea <- MCP$area

# Recalculate the kernel area using href
K <- kernelUD(SP, h = 'href', grid = 800, extent = 1)
# We will generate a 95% contour here for test purposes in a moment
K_href_95 <- getverticeshr(K, 95)

# then pull the area for the 95% range
kernelarea <- kernel.area(K, percent = 95, unin = "m", unout = "ha")

# Then we can store the largest area for use later on.
ylimforHRgraph <- max(cbind(MCParea, kernelarea))
ylimforHRgraph
```

```
## [1] 4941.844
```

```r
# Alternatively if we later find the href area to be insufficient we could pull
# the actual value of href, increase it and return it to the kernelUD function
K@h$h
h_forYLimit <- K@h$h + 100
K <- kernelUD(SP, h = h_forYLimit, grid = 800, extent = 10)
kernelarea <- kernel.area(K, percent = 95, unin = "m", unout = "ha")
# This would provide us with a greatly elevated y limit

# Or you can overwrite all this and manually define a y limit
# ylimforgraph <-
```

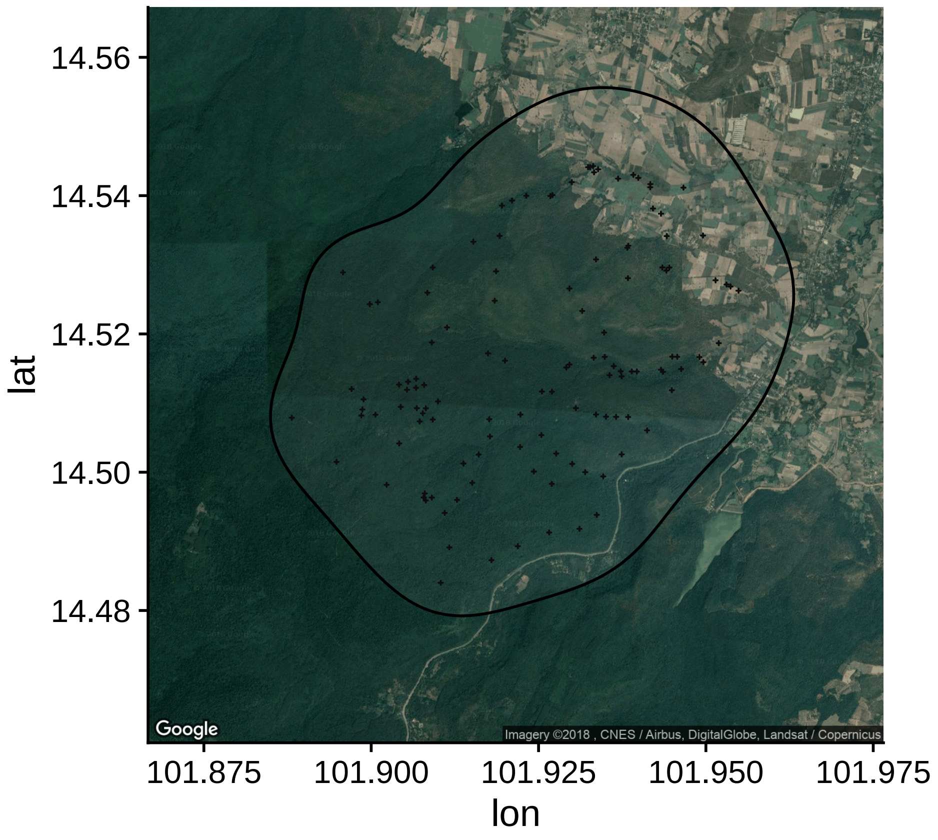
We will also need to set-up the base map. ggmap works best with latitude and longitude defined objects, so we'll covert our objects now. We previously stored the correct EPSG numbers for this earlier.

```r
SPlatlong <- spTransform(SP, latlong)
K_href_95_latlong <- spTransform(K_href_95, latlong)
```

To get a map from ggmap's get_map function we need to pass it a location in decimal degrees. We can get this by calculating the centre of the dataset's range of latitude and longitude values.

```r
# Because SP is a spatial object, we need to call the coords within the object,
# hence the @
location <- c(mean(range(SPlatlong@coords[,1])),
              mean(range(SPlatlong@coords[,2])))

# Now we can create a background layer to print the shapefiles
# upon using google maps
# use ?get_map to see all the options available
map <- get_map(location = location, crop = F,
               maptype = "satellite",
               source = "google",
               zoom = 13)
# Critical to check that the zoom works well for the scale of your data.
```

```r
# preview the map area and make sure it looks to cover all the locations in
# the dataset being used
# zoom levels may need to be changed. Again run ?get_map to get the details
# on what each zoom level is
ggmap(map) + geom_point(data = as.data.frame(SPlatlong),
                        aes(x = coords.x1, y = coords.x2),
                        colour = "black", pch = 3, size = 0.2)+
  geom_spatial(data = K_href_95_latlong, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 1, size = 0.5, colour = "black")
```



We can see from this test that all our points are visible and the polygon does not overlap the edge of the map. If this happens you can get some strange mapping issues later on, so go back now and make sure that the zoom and location are giving you the correct and workable basemap.

The animation and output

Before we get the animation running we need to set-up some setting for it. These are all found within ani.options. We can set things like delay between frames when replaying, resolution and aspect ratio. More options can be found using ?ani.options(). For now we will render the output in a standard 1080 HD format. But also, and most importantly, we need to tell it where the ffmpeg compression software is located. Make sure this is correct before running the animation loop. You will only be warned of its failure once you have waited for the entire loop to complete.

```r
# We can located the ffmpeg software using
Sys.which('ffmpeg')
```

```
##                   ffmpeg
## "/usr/local/bin/ffmpeg"
```

```r
# Then enter that into the appropriate part of ani.options()
ani.options(interval = .3, ani.width = 1920, ani.height = 1080, autoplay = FALSE,
            ffmpeg = "/usr/local/bin/ffmpeg")
# Do not take this directory location as correctly formatted if you are
# working on windows.
```

```r
# Before we begin we need to create a location to store all our areas as
# they are being produced
areasall <- NULL
```

Now we are ready for the loop that will generate each frame of the video or gif, one by one. Be warned this is not a particularly efficient loop. Essentially it boils down to a repeated, and steadily increasing, subset of the data having a kernel and MCP calculated. Each frame gets its own unique kernel and MCP.

Everything is placed within a saveVideo() function. Except for the filename. saveVideo() will give use a .mp4 file at the end. Alternatively you can run this to produce a GIF using saveGIF; take care to change the file extension. I have found that GIFs made using this code are substantially larger files.

If at any point during this process the whole thing fails, you may need to run:

```r
dev.off()
```

This will stop and close the current device, which in the case of a failure would be a device attempting to generate a .mp4 file.

```r
saveVideo({
  # the first step of the loop is defining how long and the maximum amount
  # of data to use
  # it would be wise to replace length(animal$ID) with a lower number for
  # the first run with your own data
  # before committing to running a full dataset
  # it starts at five because we need at least five locations to
  # calculate the kernels
  for (i in 5:length(animaldata$ID)) {
    # first step is to subset the data by i
    # i is the changing variable that will increase by one every time
    # the loop is repeated
    CurrSet <- animaldata[1:i,]

    # now the code essentially follows what has been described above
    # but for only the subset defined by the above line
    coords <- cbind(CurrSet$Easting, CurrSet$Northing)

    # next five lines should be familiar
    SP <- SpatialPoints(coords, proj4string = utm)
    MCP <- mcp(SP, percent = 95, unin = "m", unout = "ha")

    # do make sure that these values make sense for the dataset you are working with
    K <- kernelUD(SP, h = "href", grid = 1000, extent = 10)

    # the next two generate contours based on 50 and 95% probabilities pulled from
    # the kernel utilisation distribution
    K50 <- getverticeshr(K, 50)
    K95 <- getverticeshr(K, 95)

    # next two lines pull the areas of the MCP and the 50 and 95% contours
```

```r
MCParea <- MCP$area
kernelareas <- kernel.area(K, percent = c(50, 95), unin = "m", unout = "ha")

# next few lines creates a new object to store them in and
# renames them so more can be added
areas <- cbind(MCParea, kernelareas[1], kernelareas[2])
areas <- as.data.frame(areas)
areas <- cbind(areas, CurrSet$Tracking_date[length(CurrSet$ID)])
names(areas) <- c("MCP", "K50", "K95", "Date")
row.names(areas) <- "Area (ha)"
# after we have given them a workable-with name we can attatch it to the
# empty object created before the saveVideo was run
# every run of the loop, with every i value, adds one more row to the areasall object
# we need all of them as it grows to plot on the graphs
areasall <- rbind(areasall, areas)

# same as before,
# code that converts the UTM based shapes to WGS84 to help with plotting in ggmap
SPlatlong <- spTransform(SP, latlong)
MCPlatlong <- spTransform(MCP, latlong)
K50latlong <- spTransform(K50, latlong)
K95latlong <- spTransform(K95, latlong)

## the next chunk is purely defining how the ggmap plot should look
# I recommend people looking up ggplot cheatsheets and the like to
# investigate the various options that ggplot can provide
# ggplot is pretty great and there are lots of aesthetic options available,
# experiment...
# I would recommend experimentation outside of the saveVideo function to
# save time and to quick re-run changes made

# first line is telling r to store it in fullmap and to base it on the
# previously stored map object
# map was defined earlier using get_map()
fullmap <- ggmap(map) +
  # these three geom_spatial()s plot all our defined shapefiles
  geom_spatial(data = K95latlong, aes(),
               fill = "white",  alpha = 0.3,
               linetype = 1, size = 0.5, colour = "black") +
  geom_spatial(data = K50latlong, aes(),
               fill = "white",  alpha = 0.3,
               linetype = 1, size = 0.5, colour = "black") +
  geom_spatial(data = MCPlatlong, aes(),
               colour="black",  alpha = 0,
               linetype = 1, size = 1.5) +
  geom_segment(data = as.data.frame(SPlatlong),
               # geom_segment() draws a line between two points,
               # the first segment is between the last and penultimate locations
               aes(x = coords.x1[i-1], xend = coords.x1[i],
                   y = coords.x2[i-1], yend = coords.x2[i]),
               size = 2, colour = "red4") +
  geom_segment(data = as.data.frame(SPlatlong),
               # and pen-penultimate locations # this is between the penultimate
```

```r
                 aes(x = coords.x1[i-2], xend = coords.x1[i-1],
                     y = coords.x2[i-2], yend = coords.x2[i-1]),
                 size = 1.5, alpha = 0.5, colour = "red4") +
  geom_segment(data = as.data.frame(SPlatlong), # and again
                 aes(x = coords.x1[i-3], xend = coords.x1[i-2],
                     y = coords.x2[i-3], yend = coords.x2[i-2]),
                 size = 1, alpha = 0.25, colour = "red4") +
  geom_point(data = as.data.frame(SPlatlong),
               aes(x = coords.x1, y = coords.x2),
               colour = "black", pch = 3, size = 2) +
  geom_point(data = as.data.frame(SPlatlong),
               # plots the points of the SP object we defined,
               # but we need to coerce it to a dataframe to get at the real coords
               aes(x = coords.x1[i], y = coords.x2[i]),
               colour = "red", pch = 16, size = 5) +
  # a line that prints a title with the animal ID
  labs(colour = "white", title = CurrSet$ID[1],
       # adds the time and date below the title
       subtitle = paste(CurrSet$Tracking_date[length(CurrSet$ID)],
                        CurrSet$Tracking_time[length(CurrSet$ID)]),
       x = "Longitude", y = "Latitude") +
  # a line that neatens the borders
  theme(panel.border = element_rect(colour = "black", fill = NA, size = 3),
        title = element_text(face = 1, size = 40), # deals with all text first
        # overwrites previous line for title only
        plot.title = element_text(size = 70, face = 2, hjust = 0,
                                  margin = margin(t = 20, r = 0, b = 0, l = 0)),
        # overwrites for subtitle
        plot.subtitle = element_text(face = 3, size = 50,
                                     margin = margin(t = 20, r = 0, b = 20, l = 0)),
        text=element_text(size = 20, colour = "black"),
        # changes the scales labels
        axis.text.x = element_text(angle = 45, hjust = 1, colour="black",
                                   margin = margin(t = 15, r = 0, b = 10, l = 0)),
        axis.text.y = element_text(angle = 45, hjust = 1, colour = "black",
                                   margin = margin(t = 0, r = 10, b = 0, l = 15)),
        plot.background = element_rect(fill = "white"),
        axis.line = element_line(colour = "black", size = 0.5, linetype = "solid"),
        axis.ticks = element_line(colour = "black", size = 2, linetype = "solid"),
        axis.ticks.length = unit(0.5, "cm"))
# all this theme stuff can be modified and experimented with

# this line just makes sure that Tracking_date is being read as a date,
# if it wasn't the x axis would not work
# it would be labelled with numbers and we couldn't define breaks as a period of time
CurrSet$Tracking_date <- as.Date(CurrSet$Tracking_date, format = "%Y-%m-%d")

# now we can generate the distance moved graph
distanceVtime  <- ggplot(CurrSet) +
  geom_segment(aes(x = as.Date("1971-01-01", format = "%Y-%m-%d"),
                   xend = max(CurrSet$Tracking_date),
                   y = CurrSet$CsumMoves[length(CurrSet$ID)]/1000,
                   yend = CurrSet$CsumMoves[length(CurrSet$ID)]/1000),
               size = 0.5, colour = "grey45", alpha = 0.2) +
  # this creates the horizontal line that links the location to left side of
  # the graph. The 1970 date seems odd but that just ensures to goes all the
  # way to the y axis regardless of the data's timescale
  geom_point(aes(x = Tracking_date, y = CsumMoves/1000), pch = 43, size = 1.5) +
  # we are dividing all the movements by 1000 to get them into km,
  # this is appropriate for these data, but may need to be changed
  geom_point(aes(x = Tracking_date[length(CurrSet$ID)],
                 y = CsumMoves[length(CurrSet$ID)]/1000),
             pch = 21, colour = "red", size = 2.5) + # this is the geom_point
  # that highlights the most recent added point
  geom_line(aes(x = Tracking_date, y = CsumMoves/1000), alpha = 0.8, size = 2) +
  theme_bw() + # a ggplot theme that simplfies the look fo the graph
  coord_cartesian(xlim = c(min(animaldata$Tracking_date),
                           max(animaldata$Tracking_date)),
                  ylim = c(min(animaldata$CsumMoves)/1000,
                           max(animaldata$CsumMoves)/1000)) +
  # here we define the scale limits, coord_cartesian does not remove data,
  # only moves it from view. Therefore, be careful you are not missing data
  scale_x_date(breaks = date_breaks("4 weeks"),
               # making sure the x axis is something sensible using the
               # above mentions scales package
               labels = date_format("%d %m %Y")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, colour="black",
                                   margin = margin(t = 6, r = 0, b = 12, l = 0),
                                   size = 15),
        axis.title.x = element_text(size = 30),
        axis.text.y = element_text(angle = 0, hjust = 1,
                                   margin = margin(t = 0, r = 6, b = 0, l = 12),
                                   size = 20),
        axis.title.y = element_text(size = 30),
        axis.line = element_line(colour = "black",
                                 size = 0.5, linetype = "solid"),
        axis.ticks = element_line(colour = "black", size = 1.5,
                                  linetype = "solid"),
        axis.ticks.length=unit(0.25, "cm")) +
  # all some more theme stuff changing the aesthetics of the graph
  annotate("text", x = as.Date(animaldata$Tracking_date[length(animaldata$ID)]),
           y = max(animaldata$CsumMoves)/10000, size = 20,
           label = paste(round(CurrSet$CsumMoves[length(CurrSet$ID)]/1000,
                               digits = 2), "km"),
           hjust = 1) + # this little chunk simply places the total distance
  # moved on the graph and will update per frame.
  # Again in km so may need to be changed for other datasets
  scale_y_continuous(breaks = seq(0, max(animaldata$CsumMoves)/1000, 5)) +
  # y and x axis breaks will need to be changed dependent on your study species
  labs(title = "", x = "Date", y = "Distance Moved (km)")

# once again making sure r is reading this variable as a date in the correct format
areasall$Date <- as.Date(areasall$Date, format = "%Y-%m-%d")

# melt is an odd function that sort of flattens a dataframe
# we need to do this so we can use can split the colour by a factorised
# variable in a second
meltedareas <- melt(areasall)
# Now I had some difficulties here getting the date to read properly
# as you can see from the melt results it has been returned as a numeric value
# this next line is to get it into the correct data format, you can use this
# to test for the correct origin and date format
meltedareas[meltedareas$variable == "Date",][,2] <-
  as.Date(meltedareas[meltedareas$variable == "Date",][,2],
          origin = as.Date("1969-01-01", format = "%Y-%m-%d"))

# this creates a new column filled with the date ready for ggplot,
## DOUBLE CHECK this stage and make sure that the origin is correct for your data
# THIS WILL BE THE ONE DISPLAYED
# I know there are some irritating differences between the origin used
# by Mac and Windows Excel versions,
# something like a 4 year difference
meltedareas$Date <- as.Date(meltedareas[meltedareas$variable == "Date",][,2],
                         origin = as.Date("1971-01-01", format = "%Y-%m-%d"))
# we can now get rid of that bottom row so we only have values in 'variable'
# that we want to use as a factor in the ggplot
meltedareas <- meltedareas[!meltedareas$variable == "Date",]

# similar situation as above, a whole series of lines generating the ggplot for the areas
EstimationAreas <- ggplot(meltedareas) +
  geom_segment(aes(
    x = as.Date("1971-01-01", format = "%Y-%m-%d"),
    xend = max(meltedareas$Date),
    y = tail(meltedareas[meltedareas$variable == "MCP",],1)[1,2],
    yend = tail(meltedareas[meltedareas$variable == "MCP",],1)[1,2]),
    size = 0.5, colour = "grey45", alpha = 0.2) +
  geom_segment(aes(
    x = as.Date("1971-01-01", format = "%Y-%m-%d"),
    xend = max(meltedareas$Date),
    y = tail(meltedareas[meltedareas$variable == "K95",],1)[1,2],
    yend = tail(meltedareas[meltedareas$variable == "K95",],1)[1,2]),
    size = 0.5, colour = "grey45", alpha = 0.2) +
  geom_segment(aes(
    x = as.Date("1971-01-01", format = "%Y-%m-%d"),
    xend = max(meltedareas$Date),
    y = tail(meltedareas[meltedareas$variable == "K50",],1)[1,2],
    yend = tail(meltedareas[meltedareas$variable == "K50",],1)[1,2]),
    size = 0.5, colour = "grey45", alpha = 0.2) +
  geom_point(aes(x = Date, y = value, colour = as.factor(variable)),
             size = 1.5) +
  # the geom_line and _point lines display the actual progress of the home range areas
  geom_line(aes(x = Date, y = value, colour = as.factor(variable)),
            alpha = 0.8, size = 1) +
  geom_point(aes(x = max(meltedareas$Date),
                 y = tail(meltedareas[meltedareas$variable == "MCP",],1)[1,2]),
             pch = 21, colour = "red", size = 3) +
  # these lines are used to created the red highlight around the most recent point
  geom_point(aes(x = max(meltedareas$Date),
                 y = tail(meltedareas[meltedareas$variable == "K95",],1)[1,2]),
         pch = 21, colour="red", size = 3) +
geom_point(aes(x = max(meltedareas$Date),
             y = tail(meltedareas[meltedareas$variable == "K50",],1)[1,2]),
         pch = 21, colour = "red", size = 3) +
coord_cartesian(xlim = c(min(animaldata$Tracking_date),
                         max(animaldata$Tracking_date)),
                ylim = c(min(animaldata$CsumMoves), ylimforHRgraph)) +
# once again setting our limits based on what was calculated for the
# entire dataset. This can pose problems for kernel estimations that are
# actually larger for certain subsets. They will exceed the Y limits,
# and to be fixed require some manual adjustments.
scale_y_continuous(breaks = seq(0, ylimforHRgraph,
                              round(ylimforHRgraph/10, digits = -1))) +
# note the rounding function within there to make the breaks more sensible,
# this may require modification per dataset
scale_x_date(breaks = date_breaks("4 weeks"),
           labels = date_format("%d %m %Y")) +
# making our x axis a little easier to read
guides(color = guide_legend(override.aes = list(size = 3))) +
# the next section deals with the legend, it adds the actual area from
# these estimation too
scale_color_hue(
l = 65,
c = 100,
h = c(130, 250),
labels = c(paste0("MCP: ",
                  round(tail(meltedareas[meltedareas$variable == "MCP",],1)[1,2],
                        digits = 2), " ha  "),
           paste0("K50: ",
                  round(tail(meltedareas[meltedareas$variable == "K50",],1)[1,2],
                        digits = 2), " ha  "),
           paste0("K95: ",
                  round(tail(meltedareas[meltedareas$variable == "K95",],1)[1,2],
                        digits = 2), " ha  "))) +
# you can change the values in this line to change the colours used for
# each estimation method
theme_bw() +
labs(title = "",
   x = "", y = "Home Range Area (ha)") +
theme(axis.text.x = element_text(
angle = 45, hjust = 1, colour="black",
margin = margin(t = 6, r = 0, b = 12, l = 0),
size = 15),
axis.title.x = element_text(size = 30),
axis.text.y = element_text(
  angle = 0, hjust = 1,
  margin = margin(t = 0, r = 6, b = 0, l = 12),
  size = 20),
axis.title.y = element_text(size=30),
axis.line = element_line(colour = "black",
                         size = 0.5, linetype = "solid"),
axis.ticks = element_line(colour = "black", size = 1.5, linetype = "solid"),
axis.ticks.length = unit(0.25,"cm"),
      legend.text = element_text(size = 25, face = 3),
      legend.title = element_blank(),
      panel.background = element_rect(fill = alpha('white', 0.5), colour = NA),
      plot.background = element_rect(fill = alpha('white', 0.1), colour = NA),
      legend.background = element_rect(fill = alpha('white', 0), colour = NA),
      legend.key = element_rect(fill = alpha('white', 0), colour = NA),
      legend.position = "top")

  # now we can use cowplot to plot all of the graphs together onto a frame ready for printing
  # I have opted to use ggdraw here because of the control you can get over spacing
  # I found it easier to work with ggdraw as opposed to plot_grid and the
  # like when working to a set resolution and aspect ratio.
  # ggmap in particular does not always play nice with others.
  Grapharrange <- ggdraw() +
    draw_plot(fullmap, x = -0.25, y = 0., width = 1, height = 1) +
    draw_plot(EstimationAreas, x = 0.52, y = 0.5, width = 0.45, height = 0.5) +
    draw_plot(distanceVtime, x = 0.52, y = 0, width = 0.45, height = 0.5)

  # next few lines will print a few lines to the console as the code
  # runs keeping you updated on progress
  # each frame generated will have its own print out
  # it starts at five because the kernel estimations require at least 5 points
  # to be calculated
  print(paste("Datapoint", i,"/",length(animaldata$ID), "Complete"))
  print(paste(CurrSet$ID[1],
                CurrSet$Tracking_date[length(CurrSet$ID)],
                CurrSet$Tracking_time[length(CurrSet$ID)]))
  # here we need it to print the ggplot results
  # usually ggplot graphs will print without the print() function,
  # but within this loop it is required
  print(Grapharrange)
} # this is where the loop that generates each frame closes
beep(2) # a beep when complete
# once this sounds there will be some time for the file to be compiled
}, video.name = paste0(animaldata$ID[1], "animated.mp4"))
# and here we close the expression passed to saveVideo
# final line pulls the animal ID to put in the final output's filename
```

You may see a great deal of error messages. The ones reading

"No id variables; using all as measure variables Warning message: attributes are not identical across measure variables; they will be dropped"

This occurs at the line:

```
meltedareas <- melt(areasall)
```

If you investigate what that line does you'll see that it causes no real issue regarding our purposes.

Hopefully, once this loop has finished and you are relayed a message concerning an output you will be able to find a new .mp4 file of the animation. There is an example output from the fakesnake data included to see if there were any problems.

Alternative map only plot

I'm sure that there are times when you will not want the accompanying graphs. Below is the code that will render just the map to a 1920x1920 format.

In this chunk you will also fine a couple of alternatives to the contours used, as viewable in the map only exmaple animation.

```r
for(contr in c(seq(20, 90, 10), 95)){
      contr_name <- paste0("K_UTM_", contr)
      K_contr <- getverticeshr(K, contr)
      assign(contr_name, K_contr)
    }

    for(kutm in ls(pattern = "K_UTM_*")){
      K_Number <- sub("K_UTM_", "", kutm)
      K_To_Convert <- get(kutm)
      K_LatLong <- spTransform(K_To_Convert, latlong)
      assign(paste0("K_LatLong_", K_Number), K_LatLong)
    }
```

This will slow the entire process down and make be more complicated than needed for many maps. Also if using LSCV smoothing or lower grid/extent parameters you may run into issues trying to define a 20% contour.

```r
ani.options(interval = .3, ani.width = 1920, ani.height = 1920, autoplay = FALSE,
            ffmpeg = "/usr/local/bin/ffmpeg")
# Do not take this directory location as correctly formatted if you are working on windows.


# Before we begin we need to create a location to store all our areas as
# they are being produced
areasall <- NULL

saveVideo({
  for (i in 5:length(animaldata$ID)) {
    CurrSet <- animaldata[1:i,]
    coords <- cbind(CurrSet$Easting, CurrSet$Northing)
    SP <- SpatialPoints(coords,proj4string = utm)
    MCP <- mcp(SP, percent = 95, unin = "m", unout = "ha")
    K <- kernelUD(SP, h = 'href', grid = 1000, extent = 10)

    # here's an alternative to simply having a 50 and 95% contour,
    # this will make one every 10%
    # BEWARE - this will make the entire process much much longer
    for(contr in c(seq(20, 90, 10), 95)){
      contr_name <- paste0("K_UTM_", contr)
      K_contr <- getverticeshr(K, contr)
      assign(contr_name, K_contr)
    }

    MCParea <- MCP$area
    kernelareas <- kernel.area(K, percent = c(50, 95), unin = "m", unout = "ha")
    areas <- cbind(MCParea, kernelareas[1], kernelareas[2])
    areas <- as.data.frame(areas)
    areas <- cbind(areas, CurrSet$Tracking_date[length(CurrSet$ID)])
```

```r
names(areas) <- c("MCP", "K50", "K95", "Date")
row.names(areas) <- "Area (ha)"
areasall <- rbind(areasall, areas)
SPlatlong <- spTransform(SP, latlong)
MCPlatlong <- spTransform(MCP, latlong)

# This is the corresponding loop to convert all the contours generated
# into lat long format.
for(kutm in ls(pattern = "K_UTM_*")){
  K_Number <- sub("K_UTM_", "", kutm)
  K_To_Convert <- get(kutm)
  K_LatLong <- spTransform(K_To_Convert, latlong)
  assign(paste0("K_LatLong_", K_Number), K_LatLong)
}

fullmap=ggmap(map, padding = 0)+
  geom_spatial(data = K_LatLong_95, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 1, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_90, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_80, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_70, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_60, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_50, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_40, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_30, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = K_LatLong_20, aes(),
               fill = "white",  alpha = 0.05,
               linetype = 2, size = 0.5, colour = "black") +
  geom_spatial(data = MCPlatlong, aes(),
               colour = "black",  alpha = 0,
               linetype = 1, size = 2.5) +
  geom_segment(data = as.data.frame(SPlatlong),
               aes(x = coords.x1[i-1], xend = coords.x1[i],
                   y = coords.x2[i-1], yend = coords.x2[i]),
               size = 2, colour = "red4") +
  geom_segment(data = as.data.frame(SPlatlong),
               aes(x = coords.x1[i-2], xend = coords.x1[i-1],
                   y = coords.x2[i-2], yend = coords.x2[i-1]),
                  size = 1.5, alpha = 0.5, colour = "red4") +
      geom_segment(data = as.data.frame(SPlatlong),
                  aes(x = coords.x1[i-3], xend = coords.x1[i-2],
                      y = coords.x2[i-3], yend = coords.x2[i-2]),
                  size = 1, alpha = 0.25, colour = "red4") +
      geom_point(data = as.data.frame(SPlatlong),
                aes(x = coords.x1, y = coords.x2),
                colour = "black", pch = 3, size = 3) +
      geom_point(data = as.data.frame(SPlatlong),
                aes(x = coords.x1[i], y = coords.x2[i]),
                colour = "red", pch = 16, size = 5) +
      labs(colour = "white", title = CurrSet$ID[1],
           subtitle = paste(CurrSet$Tracking_date[length(CurrSet$ID)],
                            CurrSet$Tracking_time[length(CurrSet$ID)]),
           x = "Longitude", y = "Latitude") +
      theme_bw()+
      theme(panel.border = element_rect(colour = "black", fill = NA, size = 3),
            text = element_text(size = 30, colour = "black"),
            title = element_text(face = 1, size = 40),
            plot.title = element_text(size = 70, face = 2,
                                      margin = margin(t = 20, r = 0, b = 0, l = 0)),
            plot.subtitle = element_text(face = 3, size = 50,
                                         margin = margin(t = 20, r = 0, b = 20, l = 0)),
            axis.text.x = element_text(angle = 45, hjust = 1, colour = "black",
                                       margin = margin(t = 15, r = 0, b = 25, l = 0)),
            axis.text.y = element_text(angle = 45, hjust = 1, colour = "black",
                                       margin = margin(t = 0, r = 15, b = 0, l = 25)),
            plot.background = element_rect(fill = "white"),
            axis.line = element_line(colour = "black",
                                     size = 0.5, linetype = "solid"),
            axis.ticks = element_line(colour = "black", size = 3, linetype = "solid"),
            axis.ticks.length = unit(1, "cm"))

    print(paste("Datapoint", i,"/",length(animaldata$ID), "Complete"))
    print(paste(CurrSet$ID[1],
                CurrSet$Tracking_date[length(CurrSet$ID)],
                CurrSet$Tracking_time[length(CurrSet$ID)]))
    print(fullmap)
  }
  beep(2)
}, video.name = paste0(animaldata$ID[1], "MAPONLYanimated.mp4"))
```

I hope this how-to has proven useful and you can apply it to your own data. There is much that can be expanded on from this code but hopefully the example data will present a good reference point to work from.