Visualizing fish movements with animated videos in R: The ggplot2 and gganimate packages

Workshop Handout - GLATOS, March 27, 2018

Todd Hayden, Christopher Holbrook, Mike Lowe

Updated: 2019-02-24

Introduction

The `glatos` package includes several functions that facilitate the creation of animated videos of animal movements from acoustic telemetry data. The primary functions for building animated videos are the `interpolate_path` and `make_frames` functions. `interpolate_path` calculates interpolated positions of animals between detections and `make_frames` creates a series of snapshots of interpolated and detected fish locations over time from the output of `interpolate_path`. In addition to creating and outputting frames, `make_frames` can also stitch frames together to create an animated video of movements using the open source FFmpeg software. Although `make_frames` was developed to allow customization of the output content and layout of frames, some may find the level of customization to be insufficient for their needs. In this vignette, we present a short example of an alternative approach for creating animations using the `ggplot2` and `gganimate` R packages. [1] `ggplot2` is a popular and powerful package for plotting figures based on The Grammer of Graphics approach to visualizing data and developed for R by Hadley Wickham. [2] `ggplot2` differs from base plotting functions in R by its layered approach where graphical output is defined by combing layers of plot objects. In `ggplot2` variables in data are mapped to aesthetic categories to build the plot. In contrast, calls to base R plotting functions are rendered immediately after they are submitted. The `ggplot2` framework enables complex plots to be specified with few lines of code however in our experience, customized `ggplot2` figures often require substantial effort. Base R plotting functions often require extra effort to create but are easier customize. Both frameworks will produce publication-quality graphical output and have large user-bases. The `gganimate` package is an extension to `ggplot2` that provides support for animated graphical output [3]. In the following code, we use the `make_transition` function on an example dataset included in `glatos` to create a custom transition layer necessary for interpolating fish movements. We then use the `interpolate_path` function in `glatos` to interpolate fish positions in space and time [4].Finally, we end the vignette with a demonstration of code to create an animation of fish movements using `ggplot2` and `gganimate`.

In the first chunk we load necessary packages, obtain example data from `glatos`, clean up the data, and create a figure of lakes Huron and Erie (Figure 1). The `data.table` R package is used for data manipulations in this vignette to improve efficiency. [5]

```
library(glatos)
library(sp)
library(sf)
library(raster)
```

[1] For more information about obtaining `gganmiate` and `ggplot2`, see https://gganimate.com/ and https://ggplot2.tidyverse.org/

[2] Leland Wilkinson,https://www.amazon.com/Grammar-Graphics-Statistics-Computing/dp/0387245448

[3] see https://gganimate.com/articles/gganimate.html

[4] See `glatos` package help for more information about functional arguments and outputs of `make_frames`, `make_transition`, and `interpolate_path`. Additional details about creating animated videos of fish movements using the `glatos` package is found in the workshop manual "A guide to R for analyzing Acoustic Telemetry Data V2.0". A pdf of manual can be downloaded here https://gitlab.oceantrack.org/GreatLakes/glatos/wikis/Past%20r%20workshops%20and%20manuals

[5] Data.table information: https://github.com/Rdatatable/data.table/wiki

```
library(gganimate)
library(ggplot2)
library(data.table)

# get polygon of the Great Lakes
data(greatLakesPoly)

# convert to sf spatial object
lakes <- st_as_sf(greatLakesPoly)

# crop polygon to extract Lake Huron and Lake Erie
lakes <- st_crop(lakes, xmin = -84.8, xmax = -79.6, ymin = 41.3, ymax = 46.6)
plot(st_geometry(lakes), col = "grey")
```

```
# get example walleye detection data
det_file <- system.file("extdata", "walleye_detections.csv", package = "glatos")
det <- read_glatos_detections(det_file)

# bring in receiver file and convert to sf spatial object
rec_file <- system.file("extdata", "sample_receivers.csv", package = "glatos")
recs <- read_glatos_receivers(rec_file)
setDT(recs)  # convert to data.table object- necessary for sf package
recs <- recs[deploy_long < -80, ]
recs_sf <- st_as_sf(recs, coords = c("deploy_long", "deploy_lat"), crs = 4326)
```

In the next bit of code, we create a transition layer from the `lakes` object. The transition layer is needed to calculate non-linear interpolated movement paths that avoid impossible overland movements of fish. We set the `all_touched` argument = TRUE to allow any pixel that touches the polygon outline of the Great Lakes to be coded as water in the resulting transition layer. If `all_touched` = FALSE, then more than half of the pixel must be in the polygon in order to be coded as water.

```
# temporarily convert sf spatial to sp make_transition only accepts sp
# SpatialPolygonsDataFrame
lakes <- as(lakes, "Spatial")

# make transition layer.
tran <- make_transition(lakes, res = c(0.01, 0.01), all_touched = TRUE)

# convert back to sf
lakes <- st_as_sf(lakes)
```

```
# create quick plot of fish and receivers to make sure they are 'on the map'
# (Figure 2)
unique_fish <- unique(det, by = c("deploy_lat", "deploy_long"))
plot(tran$rast)
plot(st_geometry(lakes), add = TRUE)
plot(st_geometry(recs_sf), pch = 20, col = "orange", add = TRUE)
points(unique_fish$deploy_long, unique_fish$deploy_lat, pch = 20, col = "red")
```

Next we interpolate detection data and do some cleanup.

Figure 1: Lakes Huron and Erie extracted from Great Lakes polygon included in the `glatos` package

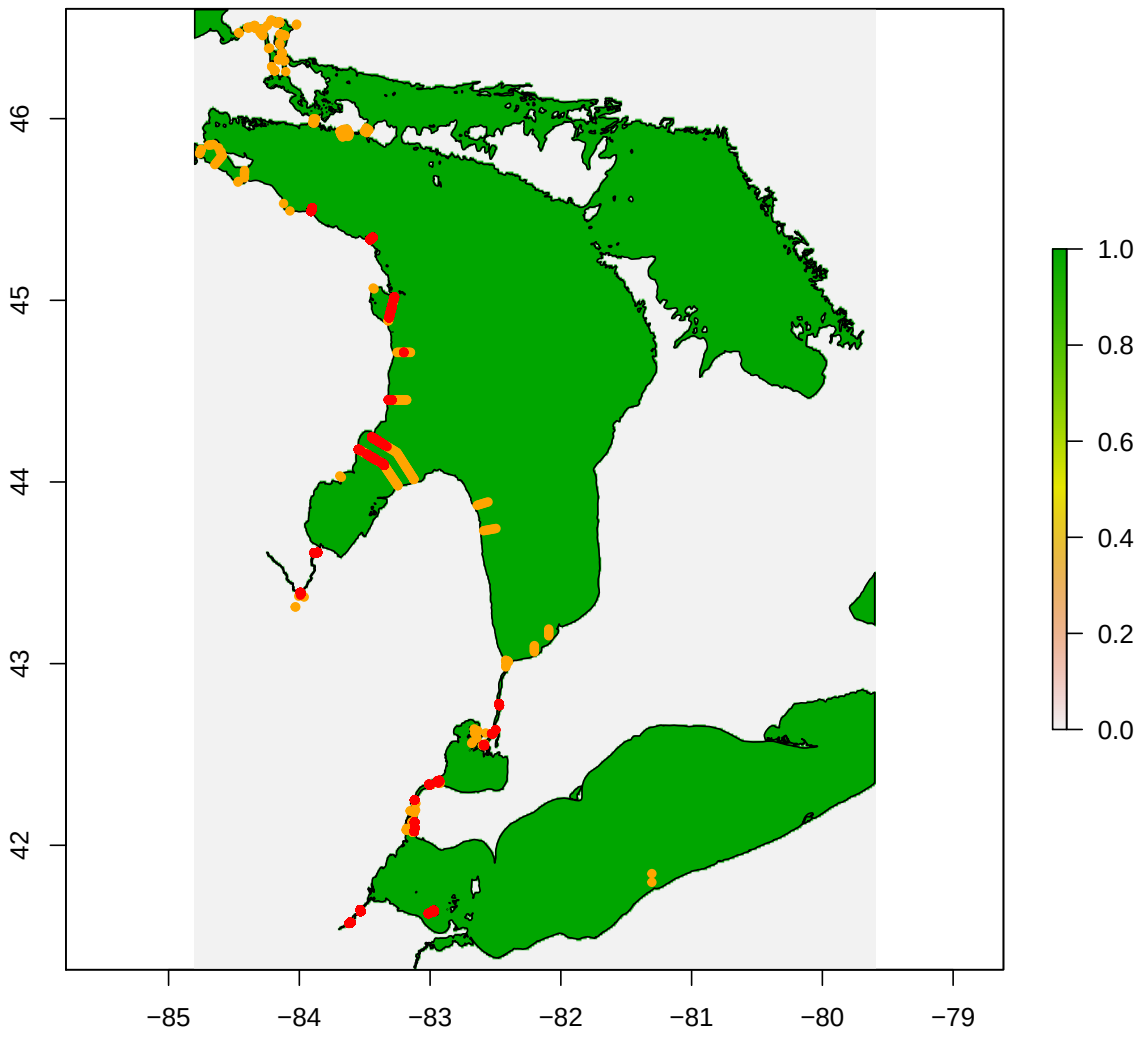

Figure 2: Plot of transition layer, receivers (orange circles), fish detection location (red circles) in lakes Huron and Erie.

```
# interpolate detections.
pos2 <- interpolate_path(det, trans = tran$transition, lnl_thresh = 0.99)
```

```
# convert pos2 to data.table
setDT(pos2)

# extract unique bin_timestamp from the interpolated data
int <- unique(pos2, by = "bin_timestamp")
```

In the next code chunk, we use a `data.table` join to add the unique timestamps extracted from `pos2` in the last line of code above (named `int`) to the receiver object. We use receiver deployment and recovery timestamps to identify if the receiver was active in that `bin_timestamp`. In other words, we need to know what receivers were deployed during each `bin_timestamp` interval so we can display receiver deployment and recovery in the animation. This step is really easy with `data.table` and would be a bit complicated using base R functions.

```
# Add bin_timestamp to receivers, based on deploy/recover timestamps.
# Removes unnecessary columns in output to simplify This is a data.table
# non-equi join...
recs <- recs[int, .(deploy_lat = x.deploy_lat, deploy_long = x.deploy_long,
    station = x.station, deploy_date_time = x.deploy_date_time, recover_date_time = x.recover_date_time
    bin_timestamp = i.bin_timestamp), on = .(deploy_date_time <= bin_timestamp,
    recover_date_time >= bin_timestamp), allow.cartesian = TRUE]
```

We are ready to create the animation. The first step is to create a static image of all receivers, detections, and interpolated detections with `ggplot2` (all frames combined). We changed the color of fish detections to red circles, interpolated positions to blue circles and made detections larger than interpolated detections.

```
ggplot(data = lakes) +
  geom_sf(color = "black", fill = "lightgrey") +
  geom_point(data = recs, aes(x = deploy_long, y = deploy_lat), size = 2, color = "orange", inherit.aes
  geom_point(data=pos2, aes(x=longitude, y=latitude, group=animal_id, color=record_type, size=record_ty
  xlab("Longitude") +
  ylab("Latitude") +
  scale_color_manual(values=c("red", "blue")) +
  scale_size_manual(values=c(2,1))
```

Now we will animate the plot with `gganimate`. We repeat the code from the previous code chunk and add an additional layer to create the animation and a layer to display a clock (`transition_time`, `ggtitle`).

```
ggplot(data = lakes) +
  geom_sf(color = "black", fill = "lightgrey") +
  geom_point(data = recs, aes(x = deploy_long, y = deploy_lat),
             size = 2, color = "orange", inherit.aes = FALSE) +
  geom_point(data=pos2, aes(x=longitude, y=latitude,
                            group=animal_id, color=record_type,
                            size=record_type), inherit.aes=FALSE) +
  xlab("Longitude") +
  ylab("Latitude") +
  scale_color_manual(values=c("red", "blue"))+
  scale_size_manual(values=c(2,1)) +
  transition_time(bin_timestamp) +
  ggtitle('{frame_time}')
```

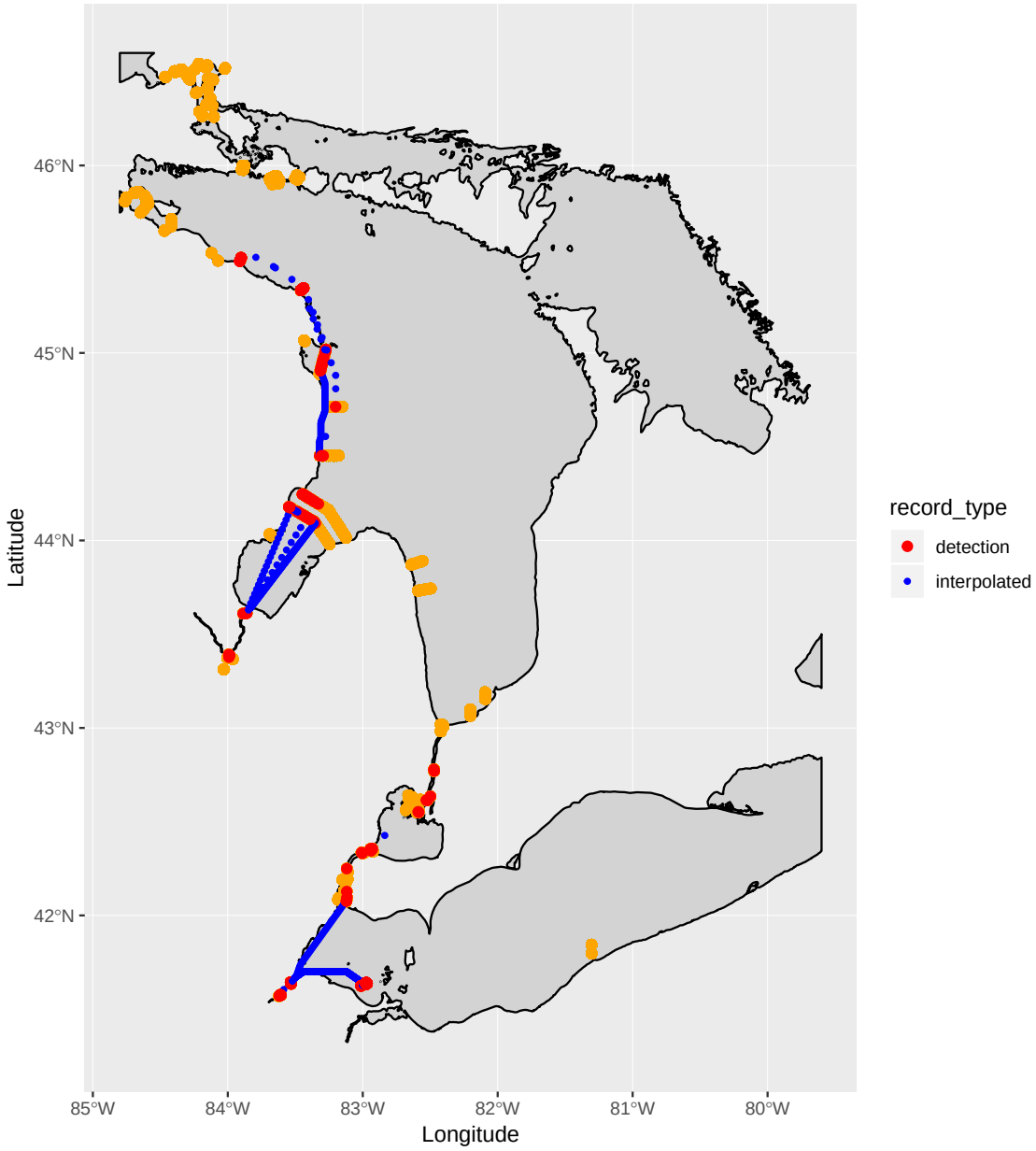

Figure 3: Static figure of fish detections (red), interpolated positions (blue), and receivers (orange) plotted using `ggplot2`

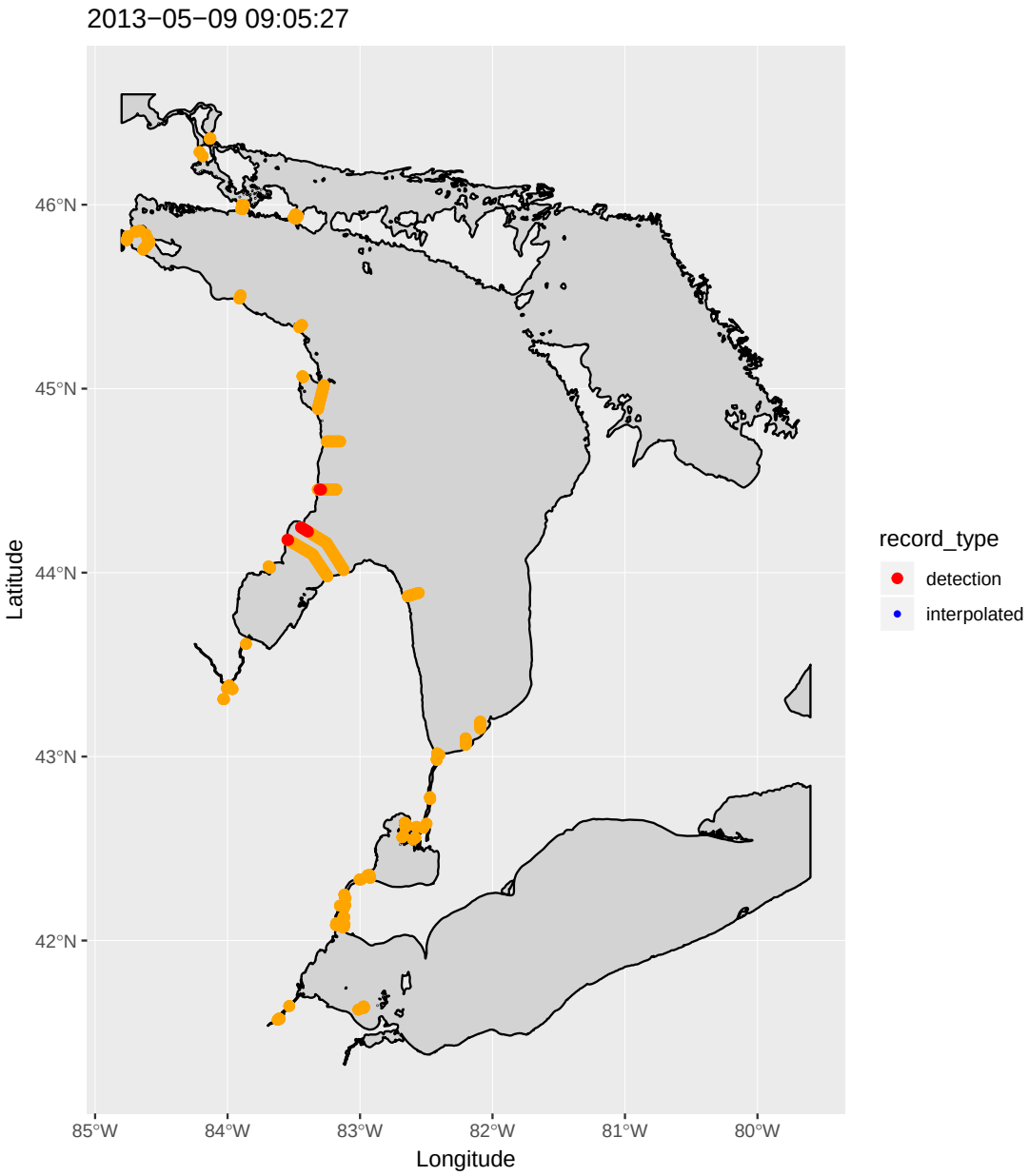

Figure 4: Animated figure of fish movements in lakes Huron and Erie. Only last image of animation is included in this vignette

As mentioned previously, the addition of a `transition_time` and `ggtitle` component in the code above creates the animation. The resulting animation may be customized (number of frames, duration of animation, frames per second, etc) by specifying arguments to the `animate` function. Please see `?animate` for more information. Also, specifying `anim_save` may be used to write the animation object to disk.

This vignette provides a basic, high-level summary of creating an animated video with `ggplot2` and `gganimate`. These packages provides much more functionality than included in this vignette. Please refer to documentation for `gganimate` and `ggplot2` for additional examples and customization that may be useful for visualizing fish movements.