Estimation of satellite tag pop-up location using a collection of customized R functions

W. Gaeuman and A. Nault
Alaska Department of Fish & Game
Division of Commercial Fisheries, Kodiak

Overview

The R workspace *Additional file 3.RData* contains 16 customized functions for estimation of Wildlife Computers MiniPAT and mrPAT satellite tag pop-up location. The functions operate primarily on Argos data contained in the associated *Locations*, *Status*, *Summary*, and, for MiniPAT tags, *Series* csv files. Files for each tag should occupy a single tag-specific directory, with all tag directories contained in a single model-specific directory (e.g., either *MiniPAT* or *mrPAT*). It is expected that the required files within an individual tag directory are named by hyphenating the directory name and file type so that, for example, the *Locations* file associated with MiniPAT tag 202492 and contained in subdirectory *202492* of the *MiniPAT* directory would be designated *202492-Locations.csv.* Also needed for the analysis are three csv files with two providing either MiniPAT or mrPAT transmission data in three columns named *tag* (tag ID), *test.transmits* (pre-deployment transmit count), and *rep.rate* (transmission repetition rate) and an additional file providing the date-time of the first uplink attempt (FUA) for mrPAT tags in two columns named *tag* (tag ID) and *FUA*, with the latter given in mdy_hm compatible format (e.g., 12-1-2021 15:22). In contrast to the required tag-specific Argos files, each of these files will contain information about multiple tags. These files should occupy the corresponding model-specific directory and contain records for all tags in the associated tag-specific subdirectories therein. Table 1 provides a summary of all input files required for the analysis.

Table 1. Required satellite-tag analysis files by tag model.

MiniPAT	mrPAT
1) *X-Locations.csv* 2) *X-Status.csv* 3) *X-Summary.csv* 4) *X-Series.csv* 5) miniPATtransmits.csv with columns *tag*, *test.transmits*, and *rep.rate*	1) *X-Locations.csv* 2) *X-Status.csv* 3) *X-Summary.csv* 4) mrPATtransmits.csv file with columns *tag*, *test.transmits*, and *rep.rate* 5) mrPAT-FUA.csv with columns *tag* and *FUA*

Note: *X* represents tag-specific subdirectory name (e.g., *202492*) within the model-specific directory (e.g., *MiniPAT*).

Analysis functions

The following list gives the names of the 16 functions contained in the R workspace *Additional file 3.RData* together with both required and optional parameters and a brief description of the function's output/action. Optional parameters are those supplied with a default value, such as *T1* in function ***get_MiniPAT.t0***. An example analysis illustrating use of the functions is given later in this document. Function code is provided in Appendix A.

1. **get_tag.status**("*directory*")
Constructs and returns a data frame of required *Status* file data for all tags of a given model.

2. **get_MiniPAT.series**(*"directory"*)
Constructs and returns a data frame of required *Series* file data for MiniPAT tags.

3. **get_tag.transmits**(*"file.pathname"*)
Constructs and returns a data frame of required transmission data for all tags of a given model.

4. **get_mrPAT.FUA**(*"file.pathname"*)
Constructs and returns a data frame of required first uplink attempt data for all mrPAT tags.

5. **get_MiniPAT.t0**(*MiniPAT.status, MiniPAT.test.transmits, MiniPAT.series, T1 = 45, T2 = 90*)
Constructs and returns a data frame containing estimates of MiniPAT pop-up time and related quantities.

6. **get_mrPAT.t0**(*mrPAT.status, mrPAT.test.transmits, mrPAT.FUA, TRANSvsFUA = 1*)
Constructs and returns a data frame containing estimates of mrPAT pop-up time and related quantities.

7. **get_tag.locations**(*"directory", "tag.model"*)
Constructs and returns a data frame of Argos location information for all tags of a given model.

8. **get_tag.HQlocs**(*tag.locs, tag.t0, HQ1.MAX.S.MAJ = 2250*)
Constructs and returns a data frame of high-quality locations for all tags of a given model.

9. **get_tag.HQpairs**(*tag.locs, tag.t0, HQ1.MAX.S.MAJ = 2250, HQ2.MAX.S.MAJ = 1250, HQ2A = 0.375, HQ2B = 1.0*)
Constructs and returns a data frame of high-quality location pairs for all tags of a given model.

10. **get_pop.locs**(*tag.HQpairs, tag.HQlocs, P = 0.63*)
Constructs and returns a data frame of estimated pop-up locations and related quantities for all tags of a given model based on high-quality locations of the target tag. The argument P, with default 0.63, allows user to choose error ellipse theoretical contour probability.

11. **get_inf.pop.locs**(*tag.HQlocs, all.locs, MAX.S.MAJ = 2000, MAX.T.DIFF = 1, MAX.DIST = 60, P = 0.63*)
Constructs and returns a data frame of estimated pop-up locations and related quantities for all tags of a given model based on locations of one or more suitable informer (i.e., proxy) tags. The argument P, with default 0.63, allows user to choose error ellipse theoretical contour probability.

12. **augment_pop.locs**(*"directory", pop.locs*)
Returns an updated, final version of the output *pop.locs* of function **get_pop.locs** that includes ancillary information about tag deployment and release timing contained in the Argos *Summary* files.

13. **augment_inf.pop.locs**(*"directory", inf.pop.locs*)
Returns an updated, final version of the output *inf.pop.locs* of function **get_inf.pop.locs** that includes ancillary information about tag deployment and release timing contained in the Argos *Summary* files.

14. **make_displays**(*pop.locs, inf.pop.locs, directory = "DISPLAYS", N = 6*)
For each tag represented in the input data frames *pop.locs* and *inf.pop.locs*, makes a visual display of estimated pop-up location based on target-tag locations together with up to N estimated pop-up locations based on informer (i.e., proxy) tag locations.

15. **estimate_errors**(*x.pop.locs, P = 0.63*)
Calculates error estimates associated with pop-up locations based on target-tag location information given in the input data frame *x.pop.locs*, a version of which is returned with error estimates appended. The argument P, with default 0.63, allows user to choose error ellipse theoretical contour probability. This function is designed to support "manual" analysis of tag data. Use of this function is described at the end of this document.

16. **estimate_errors.proxy**(*x.inf.pop.locs, P = 0.63*)
Calculates error estimates associated with pop-up locations based on informer (i.e., proxy) tag location information given in the input data frame *x.inf.pop.locs*, a version of which is returned with error estimates appended. The argument P, with default 0.63, allows user to choose error ellipse theoretical contour probability. This function is designed to support "manual" analysis of tag data. Use of this function is described at the end of this document.

Performing the analysis in R

These instructions assume that the analysis is being run using the standard R graphical user interface (GUI), though they easily extend to the RStudio integrated development environment. After starting the R GUI, some preliminary steps are necessary before the analysis can proceed. The first is to set the desired working directory as the location containing the relevant model-specific directories, e.g. *MiniPAT* and *mrPAT* and the R workspace *Additional file 3.RData*. The final preliminary step is to load the three R packages *tidyverse, geosphere,* and *lubridate* needed by the analysis functions.

The analysis involves running the functions in the correct order with the appropriate inputs. An example analysis is presented below as a sequence of five subtasks. For this example, it is assumed that: 1) the study includes both MiniPAT and mrPAT tags; 2) individual tag file subdirectories are contained in directories *MiniPAT* and *mrPAT* in accordance with tag model; 3) tag transmission data are contained in files *MiniPATtransmits.csv* and *mrPATtransmits.csv* located in directories *MiniPAT* and *mrPAT*, respectively; and 4) mrPAT first-uplink-attempt data are contained in file *mrPAT-FUA.csv* located in directory *mrPAT*.

Task 1: Estimate tag pop-up times

```
MiniPAT.status <- get_tag.status("MiniPAT")
mrPAT.status <- get_tag.status("mrPAT")

MiniPAT.series <- get_MiniPAT.series("MiniPAT")

MiniPAT.transmits <- get_tag.transmits("MiniPAT/MiniPATtransmits.csv")
mrPAT.transmits <- get_tag.transmits("mrPAT/mrPATtransmits.csv")

mrPAT.FUA <- get_mrPAT.FUA("mrPAT/mrPAT-FUA.csv")

MiniPAT.t0 <- get_MiniPAT.t0(MiniPAT.status, MiniPAT.transmits, MiniPAT.series)

mrPAT.t0 <- get_mrPAT.t0(mrPAT.status, mrPAT.transmits, mrPAT.FUA)

rm(MiniPAT.status, mrPAT.status, MiniPAT.series, MiniPAT.transmits,
      mrPAT.transmits, mrPAT.FUA)
```

The data frames *MiniPAT.t0* and *mrPAT.t0* containing estimated pop-up times are retained in the workspace for use in Task 2.

Task 2: Determine high-quality drift locations

```
MiniPAT.locs <- get_tag.locations("MiniPAT", tag.model = "MiniPAT")
mrPAT.locs <- get_tag.locations("mrPAT", tag.model = "mrPAT")

MiniPAT.HQlocs <- get_tag.HQlocs(MiniPAT.locs, MiniPAT.t0)
mrPAT.HQlocs <- get_tag.HQlocs(mrPAT.locs, mrPAT.t0)

MiniPAT.HQpairs <- get_tag.HQpairs(MiniPAT.locs, MiniPAT.t0)
mrPAT.HQpairs <- get_tag.HQpairs(mrPAT.locs, mrPAT.t0)

rm(MiniPAT.t0, mrPAT.t0)
```

All six data frames generated in this part of the analysis are used in Task 3.

Task 3: Estimate pop-up locations

```
MiniPAT.pop.locs <- get_pop.locs(MiniPAT.HQpairs, MiniPAT.HQlocs)
mrPAT.pop.locs <- get_pop.locs(mrPAT.HQpairs, mrPAT.HQlocs)

all.locs <- rbind(MiniPAT.locs, mrPAT.locs)
MiniPAT.inf.pop.locs <- get_inf.pop.locs(MiniPAT.HQlocs, all.locs)
mrPAT.inf.pop.locs <- get_inf.pop.locs(mrPAT.HQlocs, all.locs)
```

```
rm(all.locs, MiniPAT.locs, mrPAT.locs, MiniPAT.HQlocs, mrPAT.HQlocs, MiniPAT.HQpairs,
      mrPAT.HQpairs)
```

The four data frames *MiniPAT.pop.locs*, *mrPAT.pop.locs*, *MiniPAT.inf.pop.locs*, and *mrPAT.inf.pop.locs* are retained in the workspace. These contain estimated pop-up locations for the two tag models.

Task 4: Include information from the Summary.csv files in final datasets.

```
MiniPAT <- augment_pop.locs("MiniPAT", MiniPAT.pop.locs)
mrPAT <- augment_pop.locs("mrPAT", mrPAT.pop.locs)

MiniPAT.proxy <- augment_inf.pop.locs("MiniPAT", MiniPAT.inf.pop.locs)
mrPAT.proxy <- augment_inf.pop.locs("mrPAT", mrPAT.inf.pop.locs)

rm(MiniPAT.pop.locs, mrPAT.pop.locs, MiniPAT.inf.pop.locs, mrPAT.inf.pop.locs)
```

The four data frames *MiniPAT*, *mrPAT*, *MiniPAT.proxy*, and *mrPAT.proxy* containing the final results have been retained in the workspace. Field names are defined in Table 2.

Task 5: Make visual displays of tag pop-up location estimates for each tag.

```
make_displays(MiniPAT, MiniPAT.proxy, directory = "MiniPAT/DISPLAYS", N = 2)

make_displays(mrPAT, mrPAT.proxy, directory = "mrPAT/DISPLAYS", N = 2)
```

In the two function calls above, the values of the *directory* and *N* arguments have been modified from their default values *directory* = "DISPLAYS" and *N* = 6. The argument *N* determines the maximum number of informer (i.e., proxy) tag pop-up locations included in a display. Individual displays will be created for each tag (e.g., *Tag202504.png*). Figure 1 provides an example of such a display for *N* = 2.

Calculation of pop-up location error ellipses in "manual" analysis

The two functions ***estimate_errors*** and ***estimate_errors.proxy*** can be used to compute pop-up location error ellipses for any cases in which pop-up locations have been estimated outside of the automated analysis described above (i.e., manually). Required pop-up location information should be contained in a csv file. For function ***estimate_errors***, the input file must contain fields *SemMaj1, SemMin1, Orientation1, SemMaj2, SemMin2, Orientation2, DriftTime1*, and *DriftTime2* and for function ***estimate_errors.proxy***, the file must contain fields *SemMaj1, SemMin1, Orientation1, corrSemMaj1, CorrSemMin1, corrOrientation1, corrSemMaj2, CorrSemMin2, corrOrientation2, DriftTime1*, and *corrDriftTime.* Alternatively, field names can be re-specified; relations between these fields and those defined in Table 2 are found in Table 3. The functions ***estimate_errors*** and ***estimate_errors.proxy*** operate on a data frame containing the specified fields to estimate pop-up location error ellipse information and append it to the data frame in three columns called *s.maj0, s.min0*, and *angle0*.

Table 2. Field name definitions for final data frames produced by functions 12 and 13.

Field	Definition
tag	Argos platform terminal transmitter (PTT) ID number
model	tag type (e.g., MiniPAT or mrPAT)
deploy.t	deployment timestamp (UTC)
rel.t	timestamp at tag burn initiation (UTC)
deploy.days	number of days between tag deployment and tag reaching surface
t0	timstamp (mrPAT) or estimated timestamp (MiniPAT) at tag reaching surface (functions 5 and 6) (UTC)
t0.type	source of t0, based on selection criteria defined in functions 5 and 6
t1	timestamp of target tag first high-quality (HQ) location estimate defined in function 8 (UTC)
lat1	latitude of first HQ location estimate
lon1	longitude of first HQ location estimate
s.maj1	semi-major axis of first HQ location error ellipse (m)
s.min1	semi-minor axis of first HQ location error ellipse (m)
angle1	orientation of first HQ location error ellipse major axis (0–180°)
t2	timestamp of target tag next location estimate meeting criteria defined in function 9 (UTC)
lat2	latitude of next location estimate
lon2	longitude of next location estimate
s.maj2	semi-major axis of next location error ellipse (m)
s.min2	semi-minor axis of next location error ellipse (m)
angle2	orientation of next location error ellipse major axis (0–180°)
x.tag	proxy tag Argos platform terminal transmitter (PTT) ID number
x.dist	distance between the proxy tag and target tag (km)
x.t1	timestamp of proxy tag location estimate 1 meeting criteria defined in function 11 (UTC)
x.lat1	latitude of proxy tag location estimate 1
x.lon1	longitude of proxy tag location estimate 1
x.s.maj1	semi-major axis of proxy tag location 1 error ellipse (m)
x.s.min1	semi-minor axis of proxy tag location 1 error ellipse (m)
x.angle1	orientation of proxy tag location 1 error ellipse major axis (0–180°)
x.t2	timestamp of proxy tag location estimate 2 meeting criteria defined in function 11 (UTC)
x.lat2	latitude of proxy tag location estimate 2
x.lon2	longitude of proxy tag location estimate 2
x.s.maj2	semi-major axis of proxy tag location 2 error ellipse (m)
x.s.min2	semi-minor axis of proxy tag location 2 error ellipse (m)
x.angle2	orientation of proxy tag location 2 error ellipse major axis (0–180°)

Table 2. Continued.

Field	Definition
driftT	drift time from target tag or proxy tag for period used to calculate driftD (h)
driftD	estimated drift distance from target tag or proxy tag over driftT (m)
driftR	estimated drift rate from target tag or proxy tag during driftT (m/h)
driftT0	drift time between tag reaching surface and first HQ location estimate (h)
driftD0	estimated drift distance over driftT0 (m)
inv.driftB	bearing between tag's first HQ location estimate and the estimated pop-up location (°)
lat0	latitude of target tag estimated pop-up location
lon0	longitude of target tag estimated pop-up location
s.maj0	semi-major axis of estimated pop-up location error ellipse (m)
s.min0	semi-minor axis of estimated pop-up location error ellipse (m)
angle0	orientation of estimated pop-up location error ellipse major axis (0–180°)

Table 3. Relations between fields required for functions 15 and 16 and those in Table 2.

Auto	Manual	Method
s.maj1	SemMaj1	both
s.min1	SemMin1	both
angle1	Orientation1	both
s.maj2	SemMaj2	target
s.min2	SemMin2	target
angle2	Orientation2	target
driftT0	DriftTime1	both
driftT	DriftTime2	target
x.s.maj1	corrSemMaj1	proxy
x.s.min1	corrSemMin1	proxy
x.angle1	corrOrientation1	proxy
x.s.maj2	corrSemMaj2	proxy
x.s.min2	corrSemMin2	proxy
x.angle2	corrOrientation2	proxy
driftT	corrDriftTime	proxy

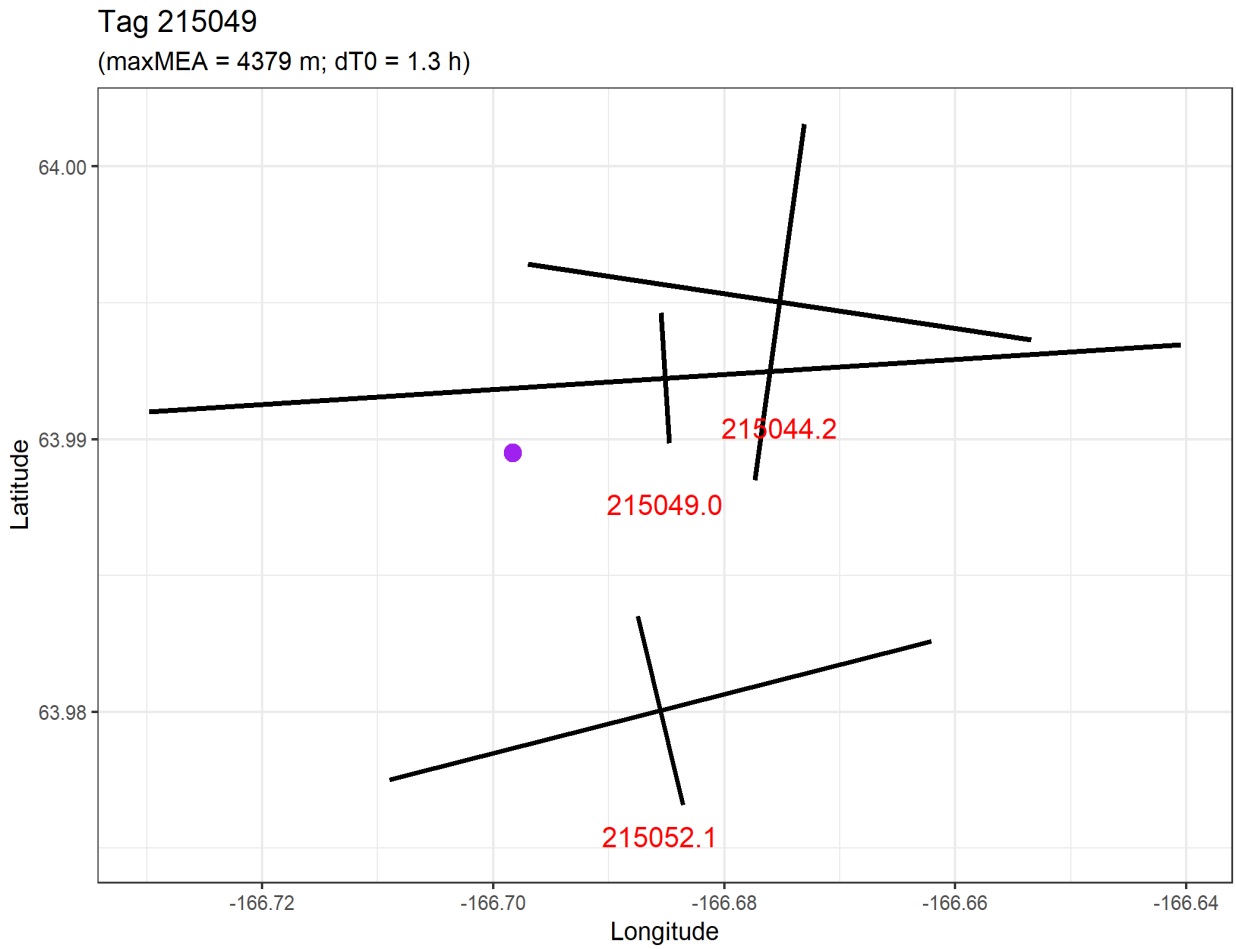

Figure 1. Pop-up location display for MiniPAT tag 215049 showing pop-up location estimated from target-tag location information (center of cross at top labeled 215049.0) and pop-up locations based on location information of the two proxy tags 215052 and 215044. Label decimal digits > 0 indicate ordering of proxy tags according to spatial proximity to the target tag. Lines of crosses are proportional to major and minor axes of estimated pop-up location error ellipses. The purple point near the center of the display is the target-tag first high-quality Argos location estimate. Drift time between the tag reaching the surface and first high-quality Argos location estimate (i.e., *dT0*) and the maximum length of the major error axes shown in the display (i.e., *maxMEA*) are reported in the display subtitle to facilitate interpretation.

Appendix A: Satellite tag analysis function code

```
# MiniPAT/mrPAT SATELLITE TAG ANALYSIS FUNCTIONS
# ******************************************
# w.gaeuman
# last update: 2 Nov 2023
#
# Required Packages:
# 1) tidyverse
# 2) lubridate
# 3) geosphere
# ----

# FUNCTION 1
# **********
get_tag.status <- function(directory){
# ---------------------------------------------
setwd(directory)
DF <- lapply(paste0(list.files(), "/", list.files(), "-Status.csv"),
                    function(x){if(file.exists(x)) read.csv(x, as.is = TRUE) %>%
                    rename(tag = Ptt) %>%
                    slice(1)}) %>% # only need first record
          do.call(rbind, .)
setwd("..")
return(DF)
}
# ----

# FUNCTION 2
# **********
get_MiniPAT.series <- function(directory){
# -----------------------------------------------------
setwd(directory)
DF <- lapply(paste0(list.files(), "/", list.files(), "-Series.csv"),
                    function(x){if(file.exists(x)) read.csv(x, as.is = TRUE) %>%
                    slice(nrow(.))}) %>%
          do.call(rbind, .) %>%
          transmute(tag = Ptt, depth = Depth, depth.t0 = dmy_hms(paste(Day, Time)) +
                    minutes(10))
setwd("..")
return(DF)
}
# ----

# FUNCTION 3
# **********
get_tag.transmits <- function(file.pathname){
# ------------------------------------------------------------
```

```
DF <- read.csv(file.pathname, as.is = TRUE) %>%
        select(tag, test.transmits, rep.rate)
return(DF)
}
# ----

# FUNCTION 4
# **********
get_mrPAT.FUA <- function(file.pathname){
# ------------------------------------------------------
DF <- read.csv(file.pathname, as.is = TRUE, na.strings = c(NA, "")) %>%
        transmute(tag = tag, FUA = mdy_hm(FUA))
return(DF)
}
# ----

# FUNCTION 5
# **********
get_MiniPAT.t0 <- function(MiniPAT.status, MiniPAT.transmits, MiniPAT.series, T1 = 45, T2 = 90){
# ------------------------------------------------------------------------------------------------------------------------
DF <- left_join(MiniPAT.status, MiniPAT.transmits) %>%
        transmute(tag = tag, model = "MiniPAT",
                rec.t = dmy(substr(Received, 10, nchar(Received))) + hms(substr(Received, 1, 8)),
        transmits = Transmits,
        trans.t0 = rec.t - minutes(round((transmits - test.transmits) * rep.rate/60, 0)),
        rel.t = dmy(substr(ReleaseTime, 10, nchar(ReleaseTime))) + hms(substr(ReleaseTime, 1, 8)),
        burn = BurnMinutes,
        rel.t0 = rel.t + minutes(burn)) %>%
        full_join(MiniPAT.series)

choose.t0 <- function(burn, depth.t0, trans.t0, rel.t0){
# ------------------------------------------------------------------
T1 = minutes(T1)
T2 = minutes(T2)
t0.type <- "DEPTH"
d.t = abs(depth.t0 - trans.t0)
d.r = abs(depth.t0 - rel.t0)
t.r = abs(trans.t0 - rel.t0)
if(is.na(d.t + d.r + t.r)){
        return("E3")
} else {
        if(burn < 450 & d.r > T1)
                if(min(t.r, d.t) < T2){
                        if(t.r < d.t) t0.type <- "REL"
                } else {
                        t0.type <- "E1"
                }
        if(burn == 450 & d.t > T2) t0.type = "E2"
```

```
        return(t0.type)
}
}
# ----

t0.type  <- character()
for(k in 1:nrow(DF))
        t0.type <- c(t0.type, with(DF[k,], choose.t0(burn, depth.t0, trans.t0, rel.t0)))
DF <- cbind(DF, t0.type) %>%
        mutate(t0 = if_else(t0.type == "REL", rel.t0,
                if_else(t0.type == "TRANS", trans.t0, depth.t0)))
return(DF)
}
# ----

# FUNCTION 6
# **********
get_mrPAT.t0 <- function(mrPAT.status, mrPAT.transmits, mrPAT.FUA, TRANSvsFUA = 1){
# ------------------------------------------------------------------------------------------------------------------
TRANSvsFUA <- hours(TRANSvsFUA)
DF <- left_join(mrPAT.status, mrPAT.transmits) %>%
        transmute(tag = tag, model = "mrPAT",
                rec.t = dmy(substr(Received, 10, nchar(Received))) + hms(substr(Received, 1, 8)),
                transmits = Transmits,
                trans.t0 = rec.t - minutes(round((transmits - test.transmits) * rep.rate/60, 0))) %>%
        full_join(mrPAT.FUA) %>%
        rename(FUA.t0 = FUA) %>%
        mutate(t0.type = ifelse(is.na(trans.t0) | is.na(FUA.t0), "E3",
                        ifelse(abs(trans.t0 - FUA.t0) > TRANSvsFUA, "TRANS", "FUA")),
                t0 = if_else(t0.type == "TRANS" | (t0.type == "E3" & !is.na(trans.t0)), trans.t0, FUA.t0))
return(DF)
}
# ----

# FUNCTION 7
# **********
get_tag.locations <- function(directory, tag.model){
# ------------------------------------------------------------
setwd(directory)
DF <- lapply(paste0(list.files(), "/", list.files(), "-Locations.csv"),
                function(x){if(file.exists(x)) read.csv(x, as.is = TRUE)}) %>%
        do.call(rbind, .) %>%
        transmute(tag = Ptt, model = tag.model,
                t = dmy(sub(".+\\s+", "", Date)) +  hms(sub("\\s+.+", "", Date)),
                lat = Latitude, lon = Longitude, s.maj = Error.Semi.major.axis,
                s.min = Error.Semi.minor.axis,
                angle = Error.Ellipse.orientation) %>%
        drop_na()
```

```
setwd("..")
return(DF)
}
# ----

# FUNCTION 8
# **********
get_tag.HQlocs <- function(tag.locs, tag.t0, HQ1.MAX.S.MAJ = 2250){
# ------------------------------------------------------------------------------------
DF <- filter(tag.locs, s.maj < HQ1.MAX.S.MAJ) %>%
        group_by(tag) %>%
        slice(1) %>%
        ungroup() %>%
        left_join(select(tag.t0, tag, model, t0, t0.type), .) %>%
        transmute(tag = tag, model = model, t0 = t0, t0.type = t0.type, t1 = t,
                lat1 = lat, lon1 = lon, s.maj1 = s.maj, s.min1 = s.min, angle1 = angle,
                t2 = NA, lat2 = NA, lon2 = NA, s.maj2 = NA, s.min2 = NA, angle2 = NA,
                driftT0 = as.numeric(difftime(t1, t0, units = "h")))
return(DF)
}
# ----

# FUNCTION 9
# **********
get_tag.HQpairs <- function(tag.locs, tag.t0, HQ1.MAX.S.MAJ = 2250, HQ2.MAX.S.MAJ = 1250,
        HQ2A = 0.375, HQ2B = 1.0){
# ---------------------------------------------
DF <- filter(tag.locs, s.maj < HQ1.MAX.S.MAJ) %>%
        left_join(select(tag.t0, tag, model, t0, t0.type), .) %>%
        group_by(tag) %>%
        mutate(t1 = t[1], lat1 = lat[1], lon1 = lon[1], s.maj1 = s.maj[1],
                s.min1 = s.min[1], angle1 = angle[1],
                driftT0 = difftime(t1, t0, units = "h")) %>%
        filter(s.maj < HQ2.MAX.S.MAJ,
                t > (t1 + HQ2A * driftT0),
                t < (t1 + HQ2B * driftT0)) %>%
        mutate(k = which(s.maj == min(s.maj))[1], t2 = t[k], lat2 = lat[k],
                lon2 = lon[k], s.maj2 = s.maj[k], s.min2 = s.min[k],
                angle2 = angle[k], driftT0 = as.numeric(driftT0)) %>%
                slice(1) %>%
                select(tag, model, t0, t0.type, t1, lat1, lon1, s.maj1, s.min1, angle1,
                        t2, lat2, lon2, s.maj2, s.min2, angle2, driftT0) %>%
        ungroup()
return(DF)
}
# ----

# FUNCTION 10
```

```
# ***********
get_pop.locs <- function(tag.HQpairs, tag.HQlocs, P = 0.63){
# ------------------------------------------------------------------------
DF <- mutate(tag.HQpairs, driftT = as.numeric(difftime(t2, t1, units = "h")),
                    driftD = distHaversine(cbind(lon2, lat2), cbind(lon1, lat1)),
                    driftR = driftD / driftT, # m/hr
                    inv.driftB = bearing(cbind(lon2, lat2), cbind(lon1, lat1)),
                    driftD0 = driftR * driftT0,
                    lat0 = destPoint(cbind(lon1, lat1), inv.driftB, driftD0)[,2],
                    lon0 = destPoint(cbind(lon1, lat1), inv.driftB, driftD0)[,1])

get.error1 <- function(x){
# ----------------------------
if(any(is.na(x))){
        return(rep(NA, 3))
} else {
        S1 <- x[1]; s1 <- x[2]; angle1 <- x[3]; S2 <- x[4]; s2 <- x[5]
        angle2 <- x[6]; driftT0 <- x[7]; driftT <- x[8]

        C <- sqrt(2) # for 63% prob contour

        lam1 <- (S1/C)^2; lam2 <- (s1/C)^2; theta <- pi/2 - pi/180*angle1
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma1 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        lam1 <- (S2/C)^2; lam2 <- (s2/C)^2; theta <- pi/2 - pi/180*angle2
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma2 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        joint.Sigma <- rbind(cbind(Sigma1, diag(0,2)), cbind(diag(0,2), Sigma2))
        a <- (driftT0 + driftT)/driftT; b <- (-driftT0/driftT)
        M <- rbind(c(a,0,b,0), c(0,a,0,b))
        Sigma <- M%*%joint.Sigma%*%t(M)
        SVD <- eigen(Sigma)

        CC <- sqrt(qchisq(P,2)) # for Px100% prob contour

        S <- sqrt(SVD$val[1])*CC; s <- sqrt(SVD$val[2])*CC
        angle <- 90 - atan(SVD$vec[2,1]/SVD$vec[1,1])*180/pi
        return(c(S, s, angle))
}
}
# ----

Xtemp <- select(DF,
                    s.maj1, s.min1, angle1, s.maj2, s.min2, angle2, driftT0, driftT) %>%
        as.matrix()
error.inf <- t(apply(Xtemp, 1, get.error1)) %>%
```

```
        as.data.frame()
names(error.inf) <- c("s.maj0", "s.min0", "angle0")
DF <- cbind(DF, error.inf)
DF <- left_join(tag.HQlocs[, 1:10], DF)
}
# ----

# FUNCTION 11
# ***********
get_inf.pop.locs <- function(tag.HQlocs, all.locs, MAX.S.MAJ = 2000, MAX.T.DIFF = 1,
MAX.DIST = 60, P = 0.63){
# ------------------------------
MAX.T.DIFF = hours(MAX.T.DIFF)

get.inf.tags <- function(all.locs, TAG){
# ----------------------------------------------
Xtemp <- mutate(all.locs, t0.diff = abs(t - TAG$t0),
        t1.diff = abs(t - TAG$t1))
x1 <- filter(Xtemp, t0.diff < MAX.T.DIFF) %>%
        slice_min(order_by = s.maj, with_ties = FALSE) %>%
        transmute(x.tag = tag, x.t1 = t, x.lat1 = lat, x.lon1 = lon,
                        x.s.maj1 = s.maj, x.s.min1 = s.min, x.angle1 = angle)
x2 <- filter(Xtemp, t1.diff < MAX.T.DIFF, t != x1$x.t1[1]) %>%
        slice_min(order_by = s.maj, with_ties = FALSE) %>%
        transmute(x.t2 = t, x.lat2 = lat, x.lon2 = lon,
                        x.s.maj2 = s.maj, x.s.min2 = s.min, x.angle2 = angle)
if(nrow(x1) * nrow(x2) == 0){
        return(data.frame(tag = TAG$tag, x.tag = NA, x.dist = NA, x.t1 = NA,
                x.lat1 = NA, x.lon1 = NA, x.s.maj1 = NA, x.s.min1 = NA,
                x.angle1 = NA, x.t2 = NA, x.lat2 = NA, x.lon2 = NA, x.s.maj2 = NA,
                x.s.min2 = NA, x.angle2 = NA))
} else {
        x.dist <- distHaversine(c(TAG$lon1, TAG$lat1), c(x2$x.lon2, x2$x.lat2)) / 1000
        return(cbind(tag = TAG$tag, x1[1], x.dist, x1[2:7], x2))
}
}
# ----

all.locs <- filter(all.locs, s.maj < MAX.S.MAJ)
inf.tags <- data.frame()
for(k in tag.HQlocs$tag){
        TAG <- tag.HQlocs[tag.HQlocs$tag == k,]
        temp <- filter(all.locs, tag != k) %>%
                group_by(tag) %>%
                do(get.inf.tags(., TAG)) %>%
                drop_na() %>%
                filter(x.dist < MAX.DIST) %>%
                .[order(.$x.dist), ]  %>%
```

```
                as.data.frame()
        inf.tags <- rbind(inf.tags, temp)
}
DF <- group_by(inf.tags, tag) %>%
        #slice_head(n = N.inf.tags) %>%
        left_join(tag.HQlocs, .) %>%
        mutate(driftT = as.numeric(difftime(x.t2, x.t1, units = "h")),
                driftD = distHaversine(cbind(x.lon2, x.lat2), cbind(x.lon1, x.lat1)),
                driftR = driftD / driftT, # m/hr
                inv.driftB = bearing(cbind(x.lon2, x.lat2), cbind(x.lon1, x.lat1)),
                driftD0 = driftR * driftT0,
                lat0 = destPoint(cbind(lon1, lat1), inv.driftB, driftD0)[,2],
                lon0 = destPoint(cbind(lon1, lat1), inv.driftB, driftD0)[,1])

get.error2 <- function(x){
# ------------------------------
if(any(is.na(x))){
        return(rep(NA, 3))
} else {
        S0 <- x[1]; s0 <- x[2]; angle0 <- x[3]; S1 <- x[4]; s1 <- x[5]; angle1 <- x[6]
        S2 <- x[7]; s2 <- x[8]; angle2 <- x[9]; driftT0 <- x[10]; driftT <- x[11]

        C <- sqrt(2) # for 63% prob contour

        lam1 <- (S0/C)^2; lam2 <- (s0/C)^2; theta <- pi/2 - pi/180*angle0
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma0 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        lam1 <- (S1/C)^2; lam2 <- (s1/C)^2; theta <- pi/2 - pi/180*angle1
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma1 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        lam1 <- (S2/C)^2; lam2 <- (s2/C)^2; theta <- pi/2 - pi/180*angle2
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma2 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        joint.Sigma <- rbind(cbind(Sigma0, diag(0,2), diag(0,2)),
                cbind(diag(0,2), Sigma1, diag(0,2)),
                cbind(diag(0,2), diag(0,2), Sigma2))
        a <- driftT0/driftT
        M <- rbind(c(1,0,a,0,-a,0), c(0,1,0,a,0,-a))
        Sigma <- M%*%joint.Sigma%*%t(M)
        SVD <- eigen(Sigma)

        CC <- sqrt(qchisq(P,2)) # for Px100% prob contour

        S <- sqrt(SVD$val[1])*CC; s <- sqrt(SVD$val[2])*CC
        angle <- 90 - atan(SVD$vec[2,1]/SVD$vec[1,1])*180/pi
```

```
        return(c(S, s, angle))
}
}
# ----

Xtemp <- select(DF, s.maj1, s.min1, angle1, x.s.maj1,
                x.s.min1, x.angle1, x.s.maj2, x.s.min2, x.angle2, driftT0, driftT) %>%
        as.matrix()
error.inf <- t(apply(Xtemp, 1, get.error2)) %>%
        as.data.frame()
names(error.inf) <- c("s.maj0", "s.min0", "angle0")
DF <- cbind(DF, error.inf)
return(DF)
}
# ----

# FUNCTION 12
# ***********
augment_pop.locs <- function(directory, pop.locs){
# ----------------------------------------------------------------
setwd(directory)
Xtemp <- lapply(paste0(list.files(), "/", list.files(), "-Summary.csv"),
                function(x){if(file.exists(x)) read.csv(x, as.is = TRUE)}) %>%
        do.call(rbind, .) %>%
        transmute(tag = Ptt,
                deploy.t = dmy(substr(DeployDate, 10, nchar(DeployDate))) +
                        hms(substr(DeployDate, 1, 8)),
                rel.t = dmy(substr(ReleaseDate, 10, nchar(ReleaseDate))) +
                        hms(substr(ReleaseDate, 1, 8)))
setwd("..")
DF <- left_join(pop.locs, Xtemp, by = "tag") %>%
        mutate(deploy.days = round(as.numeric(difftime(t0, deploy.t, units = "d")), 1)) %>%
        select(1:2, 28:30, 3:27)
return(DF)
}
# ----

# FUNCTION 13
# ***********
augment_inf.pop.locs <- function(directory, inf.pop.locs){
# ----------------------------------------------------------------
setwd(directory)
Xtemp <- lapply(paste0(list.files(), "/", list.files(), "-Summary.csv"),
                function(x){if(file.exists(x)) read.csv(x, as.is = TRUE)}) %>%
        do.call(rbind, .) %>%
        transmute(tag = Ptt,
                deploy.t = dmy(substr(DeployDate, 10, nchar(DeployDate))) +
                        hms(substr(DeployDate, 1, 8)),
```

```
                    rel.t = dmy(substr(ReleaseDate, 10, nchar(ReleaseDate))) +
                            hms(substr(ReleaseDate, 1, 8)))
setwd("..")
DF <- left_join(inf.pop.locs, Xtemp, by = "tag") %>%
          mutate(deploy.days = round(as.numeric(difftime(t0, deploy.t, units = "d")), 1)) %>%
          select(1:2, 42:44, 3:41) %>%
          select(-(14:19))  #drop HQ2 location columns
return(DF)
}
# ----

# FUNCTION 14
# ***********
make_displays <- function(pop.locs, inf.pop.locs, directory = "DISPLAYS", N = 6){
# ------------------------------------------------------------------------------------------------

f <- function(df){
# ---------------
driftT0 <- df$driftT0[1]
df <- mutate(df, s.majlon1 = destPoint(cbind(lon0, lat0), angle0, s.maj0)[,1],
                    s.majlat1 = destPoint(cbind(lon0, lat0), angle0, s.maj0)[,2],
                    s.majlon2 = destPoint(cbind(lon0, lat0), angle0 + 180, s.maj0)[,1],
                    s.majlat2 = destPoint(cbind(lon0, lat0), angle0 + 180, s.maj0)[,2],
                    s.minlon1 = destPoint(cbind(lon0, lat0), angle0 + 90, s.min0)[,1],
                    s.minlat1 = destPoint(cbind(lon0, lat0), angle0 + 90, s.min0)[,2],
                    s.minlon2 = destPoint(cbind(lon0, lat0), angle0 - 90, s.min0)[,1],
                    s.minlat2 = destPoint(cbind(lon0, lat0), angle0 - 90, s.min0)[,2],
                    x.tag = {if(x.tag[1] == tag[1]) paste0(x.tag, ".", 0:(n()-1))
                            else paste0(x.tag, ".", 1:n())})
ggplot(data = df) +
          geom_point(aes(x = lon1, y = lat1), size = 3, color = "purple") +
          geom_segment(aes(x = s.majlon1, y = s.majlat1, xend = s.majlon2, yend = s.majlat2),
          size = 1.0) +
          geom_segment(aes(x = s.minlon1, y = s.minlat1, xend = s.minlon2, yend = s.minlat2),
                    size = 1.0) +
          coord_quickmap() +
          geom_text(aes(x = lon0, y = lat0, label = x.tag), vjust = 0,
                    nudge_y = -0.005, color = "red") +
          labs(title = paste0("Tag ", df$tag[1]),
                    subtitle = paste0("(maxMEA = ", as.integer(2*max(df$s.maj0)), " m; dT0 = ",
                            round(driftT0, 1), " h)"),
                    x = "Longitude", y = "Latitude") +
          theme_bw(base_size = 10)
}
# ----

DF <- transmute(pop.locs, tag = tag, x.tag = tag, lon0 = lon0, lat0 = lat0,
                    s.maj0 = s.maj0, s.min0 = s.min0, angle0 = angle0, lon1 = lon1,
```

```
                    lat1 = lat1, driftT0 = driftT0)
DF <- group_by(inf.pop.locs, tag) %>%
                    slice_head(n = N) %>%
                    ungroup() %>%
                    transmute(tag = tag, x.tag = x.tag, lon0 = lon0, lat0 = lat0,
                    s.maj0 = s.maj0, s.min0 = s.min0, angle0 = angle0, lon1 = lon1,
                    lat1 = lat1, driftT0 = driftT0) %>%
                    rbind(DF, .) %>%
                    drop_na()

if(!dir.exists(directory)) dir.create(directory, recursive = TRUE)
cdir <- getwd()
setwd(directory)
out <- group_by(DF, tag) %>%
          do(plot = f(.))
for(k in 1:nrow(out))
          ggsave(paste0("Tag", out$tag[k], ".png"), out$plot[[k]])
setwd(cdir)
}
# ----

# FUNCTION 15
# ***********
estimate_errors <- function(pop.locs, P = 0.63){
# ------------------------------------------------------------
f <- function(x){
# -----------------
if(any(is.na(x))){
          return(rep(NA, 3))
} else {
          S1 <- x[1]; s1 <- x[2]; angle1 <- x[3]; S2 <- x[4]; s2 <- x[5]
          angle2 <- x[6]; driftT0 <- x[7]; driftT <- x[8]

          C <- sqrt(2) # for 63% prob contour

          lam1 <- (S1/C)^2; lam2 <- (s1/C)^2; theta <- pi/2 - pi/180*angle1
          e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
          Sigma1 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

          lam1 <- (S2/C)^2; lam2 <- (s2/C)^2; theta <- pi/2 - pi/180*angle2
          e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
          Sigma2 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

          joint.Sigma <- rbind(cbind(Sigma1, diag(0,2)), cbind(diag(0,2), Sigma2))
          a <- (driftT0 + driftT)/driftT; b <- (-driftT0/driftT)
          M <- rbind(c(a,0,b,0), c(0,a,0,b))
          Sigma <- M%*%joint.Sigma%*%t(M)
          SVD <- eigen(Sigma)
```

```
        CC <- sqrt(qchisq(P,2)) # for Px100% prob contour

        S <- sqrt(SVD$val[1])*CC; s <- sqrt(SVD$val[2])*CC
        angle <- 90 - atan(SVD$vec[2,1]/SVD$vec[1,1])*180/pi
        return(c(S, s, angle))
}
}
# ----

Xtemp <- transmute(pop.locs, s.maj1 = SemMaj1, s.min1 = SemMin1,
                angle1 = Orientation1, s.maj2 = SemMaj2, s.min2 = SemMin2,
                angle2 = Orientation2, driftT0 = DriftTime1, driftT = DriftTime2)%>%
        as.matrix()
Xtemp <- t(apply(Xtemp, 1, f)) %>%
        as.data.frame()
names(Xtemp) <- c("s.maj0", "s.min0", "angle0")
data <- cbind(pop.locs, Xtemp)
}
# ----

# FUNCTION 16
# ***********
estimate_errors.proxy <- function(inf.pop.locs, P = 0.63){
# ------------------------------------------------------------------------
f <- function(x){
# -----------------
if(any(is.na(x))){
        return(rep(NA, 3))
} else {
        S0 <- x[1]; s0 <- x[2]; angle0 <- x[3]; S1 <- x[4]; s1 <- x[5]; angle1 <- x[6]
        S2 <- x[7]; s2 <- x[8]; angle2 <- x[9]; driftT0 <- x[10]; driftT <- x[11]

        C <- sqrt(2) # for 63% prob contour

        lam1 <- (S0/C)^2; lam2 <- (s0/C)^2; theta <- pi/2 - pi/180*angle0
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma0 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        lam1 <- (S1/C)^2; lam2 <- (s1/C)^2; theta <- pi/2 - pi/180*angle1
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma1 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        lam1 <- (S2/C)^2; lam2 <- (s2/C)^2; theta <- pi/2 - pi/180*angle2
        e1 <- c(cos(theta), sin(theta)); e2 <- c(-sin(theta), cos(theta))
        Sigma2 <- lam1 * e1%*%t(e1) + lam2 * e2%*%t(e2)

        joint.Sigma <- rbind(cbind(Sigma0, diag(0,2), diag(0,2)),
```

```
                cbind(diag(0,2), Sigma1, diag(0,2)),
                cbind(diag(0,2), diag(0,2), Sigma2))
        a <- driftT0/driftT
        M <- rbind(c(1,0,a,0,-a,0), c(0,1,0,a,0,-a))
        Sigma <- M%*%joint.Sigma%*%t(M)
        SVD <- eigen(Sigma)

        CC <- sqrt(qchisq(P,2)) # for Px100% prob contour

        S <- sqrt(SVD$val[1])*CC; s <- sqrt(SVD$val[2])*CC
        angle <- 90 - atan(SVD$vec[2,1]/SVD$vec[1,1])*180/pi
        return(c(S, s, angle))
}
}
# ----

Xtemp <- transmute(inf.pop.locs, s.maj1 = SemMaj1, s.min1 = SemMin1,
                angle1 = Orientation1, x.s.maj1 = corrSemMaj1,
                x.s.min1 = corrSemMin1, x.angle1 = corrOrientation1,
                x.s.maj2 = corrSemMaj2, x.s.min2 = corrSemMin2,
                x.angle2 = corrOrientation2, driftT0 = DriftTime1,
                driftT = corrDriftTime) %>%
        as.matrix()
Xtemp <- t(apply(Xtemp, 1, f)) %>%
        as.data.frame()
names(Xtemp) <- c("s.maj0", "s.min0", "angle0")
pop.data <- cbind(inf.pop.locs, Xtemp)
}
# ----
```