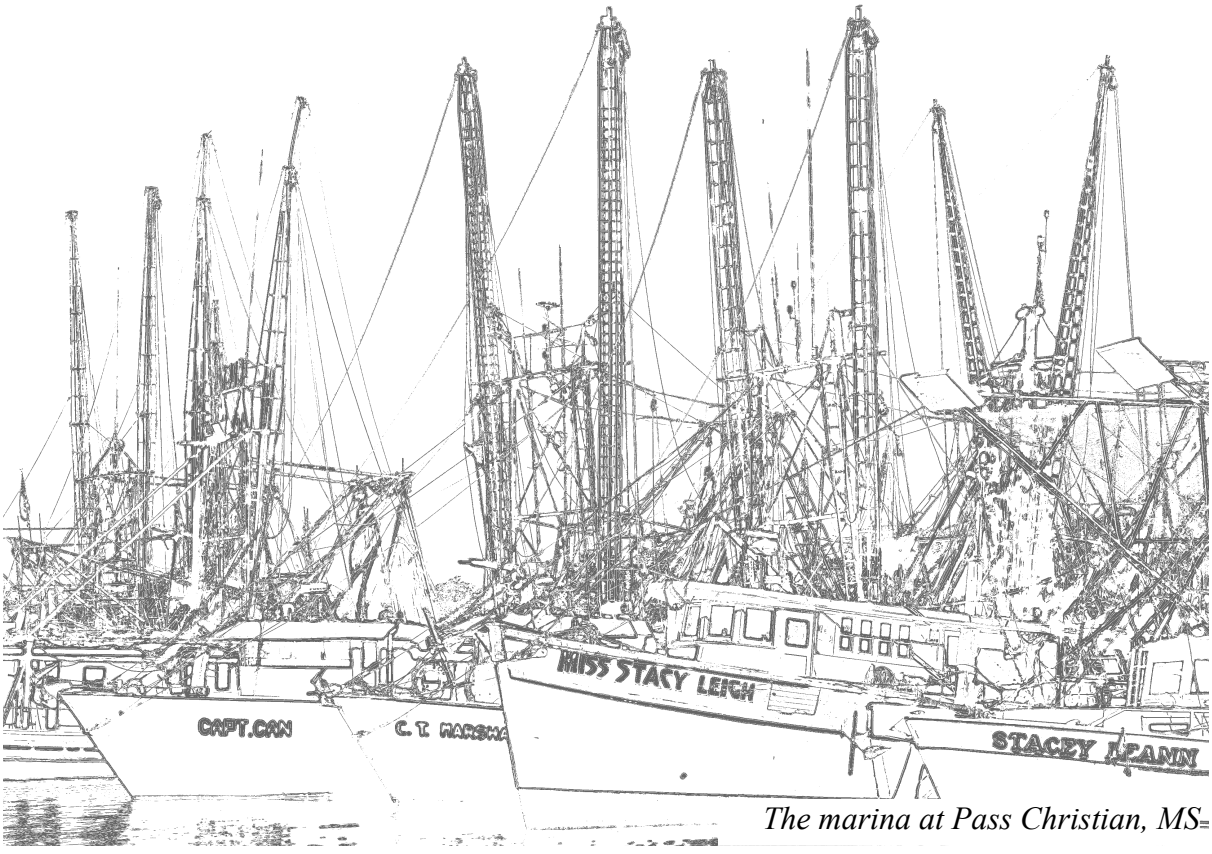
Gulf Coast Data Concepts
www.gcdataconcepts.com

Recording and Displaying the Movement of Cattle Through GPS Tracking

Test Report: TR006

Revision: A
Date: June 7, 2021

Abstract: Have you ever wondered what cows do all day? The common assumption is that they eat and sleep all day with an occasional alien abduction. Here, an x16-GPS data logger made by Gulf Coast Data Concepts was used to monitor the location and activity of cattle during a two day observation period. This test determined where the cows rested and where they moved throughout the day. The results prove that the cows move to eat and rest in shade and do not get abducted by aliens.



The marina at Pass Christian, MS

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Document Conventions

This test report describes in detail the background information, procedure, and analysis method used to conduct an experiment using a GCDC data logger. This experiment is an example application using a x16-GPS logger that is both educational and fun.

Each section also presents relevant tips and warnings to help the user repeat the experiment or make improvements.

This icon indicates a helpful tip that may enhance the results or add a new perspective to the objective.

This icon indicates a warning, restriction, or limitation that the user should be aware of regarding the experiment or logger operation.

1.2 Repeating this Experiment

Obviously, a GCDC data logger is required to repeat the test procedure. The exact product type is described in Section 4. However, the raw data and analysis methods are provided such that a user may recreate the same results without repeating the test process. Stepping through the analysis process with the data files will help the user understand the steps and see the expected results. The raw data files, spreadsheets, and R scripts are available for download at www.gulfcoastdataconcepts.com.

A spreadsheet is used for simple analysis, so the user should be familiar with manipulating data and creating plots in a spreadsheet. We recommend Microsoft Excel or LibreOffice Calc.

"R" is used for more complex data analysis. R is a simple command-line programming environment that can manipulate large data sets using common math commands as well as complex function libraries. The software is compact, free, and available at www.r-project.org for Windows, Mac, and Linux. The R scripts provided herein are designed to run on Windows and may not be fully compatible with Mac and Linux systems. Specifically, file path references will not translate properly to the Mac and Linux systems. These differences will be noted in the script comments.

1.3 Disclaimers

This test report is presented "as-is" and for informational purposes only. No claims, representations or warranties, whether expressed or implied, are made to the safety and performance of the procedures described herein.

2 Background Information

A cow is a grass grazing animal that is typically left to wander as a herd but cows are also livestock that are kept within a confined land. In either case, the wandering habits and activity levels can reveal important patterns. These patterns describe when and where the cattle prefer to eat, walk, and rest, which are important factors to the animals health. The unexplained death of cattle is often blamed on alien intervention rather than poor health habits.

2.1 Grazing and General Movement

Cattle move, eat, and rest throughout the day. Unless there is something scaring them or some unusual happening, they don't do much. That being said, it is of interest in observing cow movements. This may be for determining eating patterns, determining locations used for shelter in storms, or simply satisfying curiosity of where the cows are all day. Further, if an issue of overgrazing appears in an area, it may be investigated through the recorded data from the data logger.

2.2 Confinement

Cows are livestock and are kept confined to a certain piece of land. This may be problematic if there is an unknown hole in the fencing that the cows regularly enter into and out of. This may be especially troublesome if the land they wander into is not owned by the owner of the cows. Through GPS tracking, it can be determined if a weakness or hole exists. In the test that is to be presented, there is no hole in any fencing however, it may be inferred from observation of the recorded location data what fences the cows often travel, rest, or eat by. By obtaining this data, it may be realized that certain lines of fence need to either be strengthened, reviewed, or enhanced in some manner to prevent escape.

2.3 Alien Abduction

Cow abductions and mutilations have been recorded since at least the late 1800s, all the way up to the early 2000s, but most prominently in the 1970s. Inconsistent travel patterns and lack of activity are known conditions of a deadly alien abduction. Recording the location and activity of cattle is an important step to determining an alien abduction, assuming the aliens are kind enough to return the logger for post-process analysis.

3 Objective

The purpose of the test is to track cow movements to map their daily activity patterns.

4 Experiment Setup

4.1 Materials and Equipment

Table 1: Materials and Equipment List

Description	Quantity	Comments
GCDC B.A.T.	2	Modified x16-GPS data logger. BAT stands for Big Animal Tracker.
Zip Ties	12	Available at most hardware stores
Steel Hose Clamp	2	Available at most hardware stores
Cow Collar	1	Purchased from Amazon
Normal Human Belt	1	Old leather Columbia belt
Cows	2	Two different Cows
Ranch	1	Place for the Cows

Figure 1: Supplies for Experiment with Logger Exposed (excluding zip ties)

4.2 Equipment Setup

4.2.1 Logger Configuration

The Big Animal Tracker (BAT) is an x16-GPS data logger powered by a 6Ah battery pack and protected within a 1-1/4 inch PVC pipe coupler. The pipe coupler provides a very rugged and watertight enclosure that animals are unlikely to damage. The logger recorded 3-axis acceleration at 25Hz continuously and recorded two GPS samples every minute. The 25Hz accelerometer sample rate is more than sufficient to capture the motion of the cow, such as walking or resting. The GPS minimum horizontal dilution of precision (HDOP) was set to 5 meters. Therefore, the logger will continue to calculate a GPS locations until the accuracy is within 5 meters. Then, it will capture two locations before returning to a low power mode. The complete configuration settings for the BAT data logger may be viewed in Figure 2.

```
; PRODUCT_ID = GPS
;set sample rate
;available rates 12, 25, 50, 100, 200, 400, 800
samplerate = 25
;record constantly
deadband = 0
DeadBandTimeout = 0
;set file size to 60 minutes of data for 25Hz rate 25*3600= 90000
samplesperfile = 180000
;control brightness of LEDs
statusindicators = Normal
; set timestamp to UTC seconds since Jan 1, 1970 instead of time since start of file
absoluteTime
; GPS GPS GPS
gpsOn = 1
;gps_update interval, in seconds
gps_sampleinterval = 0.5
; minlock minimum HDOP in meters considered to be acceptable lock
gps_minlock = 5
; discard gps readings that exceed the minimum HDOP
gps_nobad
; used in on/off mode number of good samples to collect before turning off
gps_numSamples =2
; used in on/off mode, time in seconds device will spend trying to collect good samples
gps_powerOnTime = 50.0
; used in on/off mode, time, in seconds bewteen on events
gps_powerPeriod = 60.0
; dynamic models 0-->potable 2-->stationary 3-->pedestran 4--> automotive, 5-->sea,
6,7,8-->airborn
gps_dynamicModel = 3
; when enabled, polled mode saves power at the expense of timing accuracy
gps_pollmode = 1
; BMP280
;add pressure to data stream
press_pressOn
;press_samplerate = 10
;add temperature to data stream
press_tempOn
press_tempSubSample=0
; press_iir 0,1,2,3,or 4, see p14 bmp280 datasheet for details (not tested)
;press_iir = 0
; the pressure interval is the pressure transucer measurement interval, in milliseconds
press_sampleInterval = 1000
```

Figure 2: Logger Configuration

4.2.2 Collar Setup

Figure 3: Experiment Setup

4.3 Software

The analysis was performed using "R", which is a free open-source mathematics package available at www.r-project.org. An R script called "cow_tracker.r" was developed to analyze the test data from the two loggers and plot the appropriate data with Google Earth (see Appendix). The script requires installation of the "plotKML" and "rgdal" libraries in R.

The R script includes functions to load the cvs data files generated by the data logger, perform the analysis, and create a new data set representing the completed results. The analysis first extracts the best GPS location for each minute based on the HDOP value. Then, distance traveled between locations is calculated. The acceleration data is reduced to the root-mean-square (RMS) standard deviation for each one-minute period between GPS locations. Finally, the UTC time stamps are converted to local time format. The resulting data set is exported as a kml file for Google Earth as well as a standard csv test file.

Location was plotted in Google Earth with interactive pins that contain the time at UTC-0 in seconds (time), the time of week in GPS time seconds (TOW), the Mean Sea Level (MSL), the horizontal dilution of precision (HDOP), the standard deviation of the magnitude of acceleration based off acceleration in the x-direction (sd_mag), and the local time (local_time).

Acceleration was plotted as bubbles that change size and color intensity depending on the recorded RMS acceleration with its numeric value floating on the side of the bubble. Larger bubbles meaning larger intensity and bright red and blue being the maximum and minimum color intensities.

The script and the raw data files used in this report are available to download from the GCDC website.

5 Procedure

5.1 Test Procedure

The outline for the procedure of the experiment is as follows:

1. The configuration file was setup such that the data logger will wake-up every minute to record 2 points of data with the GPS at 2Hz. The accelerometer was set to run at 25Hz and the hdop at 5m. Once data is collected, the logger returns to sleep.
2. The desired cows were corralled into a smaller enclosure that led directly into a squeeze shoot.
3. The first GCDC data logger was attached to the cow collar by a steel hose clamp (Figure 3).
4. The second GCDC data logger was attached to the belt by 12 zip ties. Every two zip ties were connected to each other in order to reach fully around the data logger and connect it to the belt.
5. Cow 1 was secured in the squeeze shoot and had the cow collar data logger activated and attached. The cow was then let free.
6. Cow 2 was secured in the squeeze shoot and had the belt data logger activated and attached.
7. The cows were allowed to wander for approximately 48 hours.
8. The collar and belt were removed from each cow and the data loggers deactivated.

5.2 Analysis Procedure

The following will outline the steps of retrieving data and running the R script.

5.2.1 Data Retrieval

1. Open the GCDC BAT logger and USB cable to connect your computer and the data logger.
2. Locate the .csv file locations on the logger and either move them or note their location.
3. Write the file locations in the script and save the locations to a text file.

5.2.2 Analysis

1. Load and run the script into an interface such as R console or R Studio through the source command.
   a) a. Ex: source("C:\\folder1\\folder2\\cow_tracker.r")
      1) Note that this is the file's location but has double backslashes.
      2) See Figure 4 for an example
2. Interact with the code as it gives you prompts.
   a) This will determine if distance calculations are to flood your console and in what measurement type.
3. Upon entering the second while loop distance is calculated, corrected, and printed.
   a) To do so it will call the only function: cow.degrees_To_Radians
4. Still in the loop, acceleration is calculated and applied along with the time conversion from UTC-0 to local.
5. The loop now exits, and data recorded with a higher hdop than data is compared to partner data and removed from the list of all data.
6. If distance is displayed, distance plots will be outputted.
7. All data is printed out as two separate KML files. One is of locations as pins and the other is of acceleration as bubbles.
8. All data is outputted as a .csv file called "Total Data Set.csv"

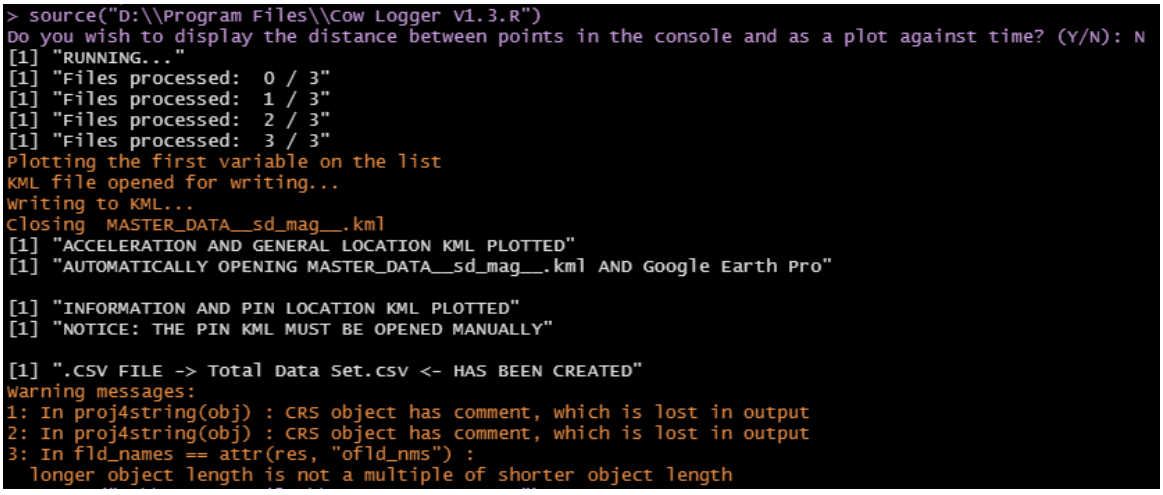


Figure 4: Using the Script from the R Console

6 Results

Of the two data loggers, one succeeded in fully recording all data and one failed to record properly. The failed logger was the one placed on the second cow. This was an early prototype logger that exhibited problems prior to this test. For an unknown reason, after activation, it failed to record for around 24 hours resulting in only the final 24 hours being recorded.

Only the data from the first cow, the successful logger, was utilized in examination. R ran all pertinent calculations and collected all pertinent data. Furthermore, it successfully and quickly printed out both location and acceleration KML files which both run correctly in Google Earth. The .csv file was also printed properly with all requested data intact. See the figures below to see the results:

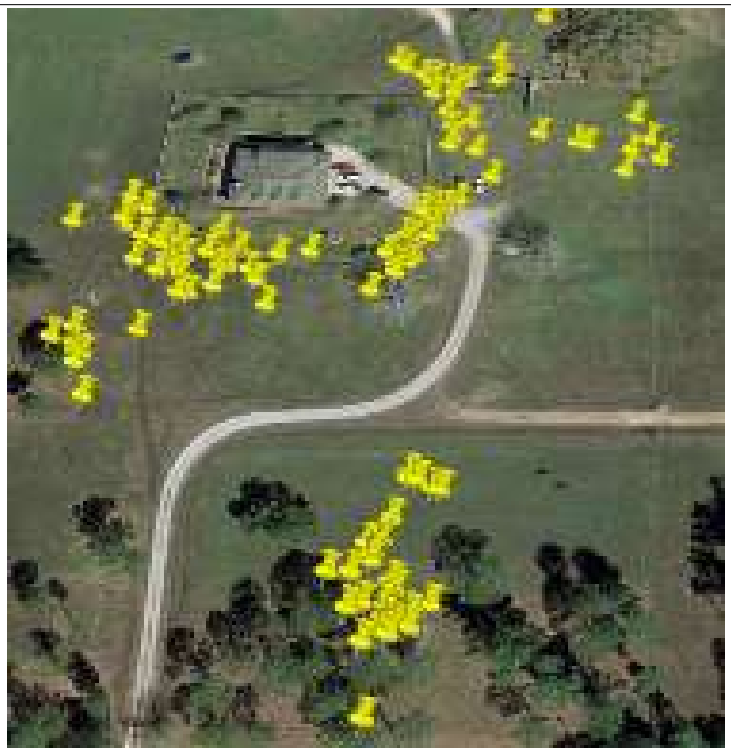


Figure 5: Ex: Location Plot

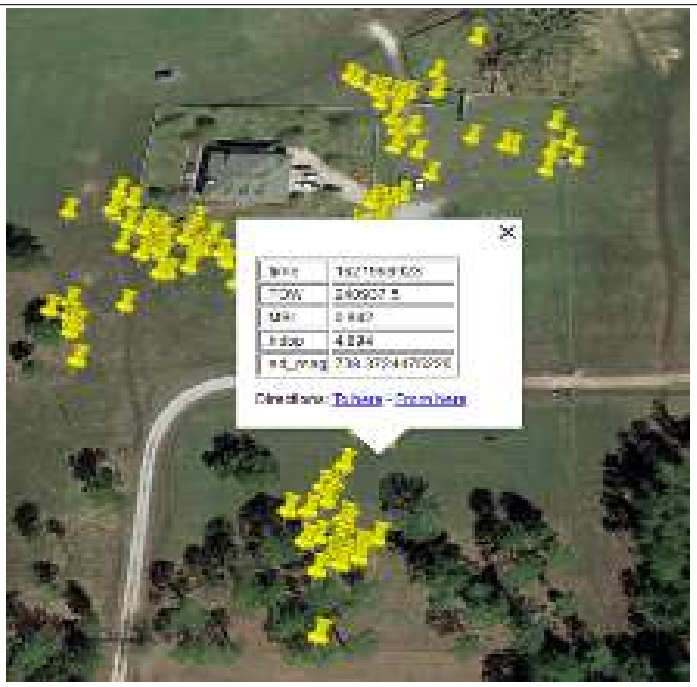


Figure 6: Location Plot with Point Selected

Figure 7: Location with Activity #1

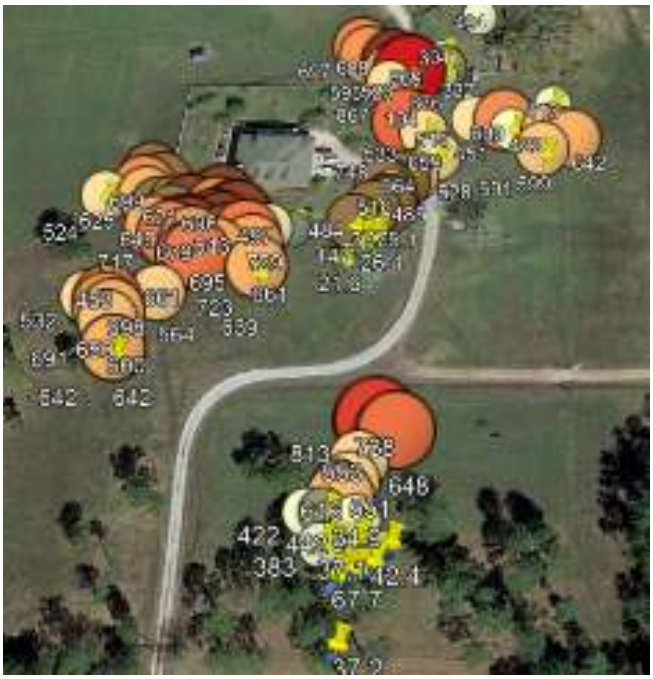


Figure 8: Location Plot with Activity #2

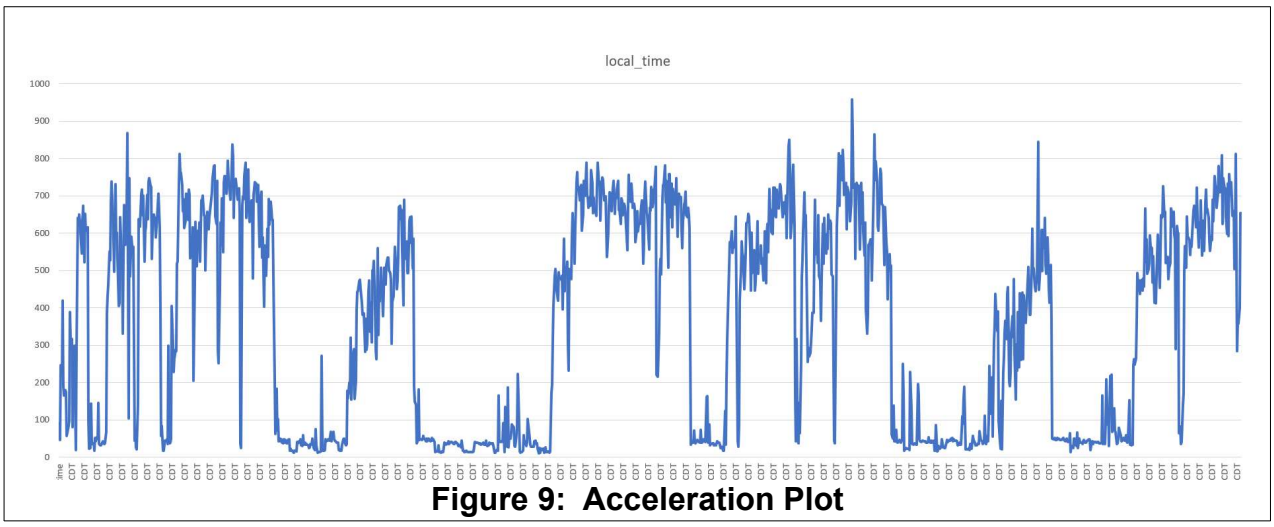


Figure 9: Acceleration Plot

7 Discussion

The GPS tracking data provided a good view of how cows travel throughout the day and the acceleration data proved useful in examining times of activity. Using R to calculate the mathematical aspects of this test was very useful with its quick computation time and extreme versatility.

8 Conclusion

The cows move often, albeit slowly, and maintain a consistent area of rest under the shade of trees. They move surprisingly often at night. Sections of fence line favored by the cattle were easily identified by the location tracks. There is neither evidence for or against alien activity. However, the possibility of alien influence regarding the failed logger cannot be ignored.

9 Appendix A: R Script – cow_tracker.r

```
#Gulf Coast Data Concepts
#September 15, 2021
#file name: "cow_tracker.r"
#R scripts for processing X16 and GPS data
#to study activity of a cow
#-------------------------------------------------
# this R script takes data generated by a GCDC data logger, specifically
# the GPS-x16 type product.  Accelerometry is used to determine activity
# of the cow and the results are overlayed into the GPS location data
# presented in Google Earth as KMZ file format.
#-------------------------------------------------
# How to use this script (Windows OS):
# copy the script to the root directory of the data logger
# (assume logger is drive d: and files are in GCDC directory)
# start the R environment or the RStudio desktop application
# load the script file using the "source" command
#   > source("d:\\cowLogger.r")
# create an array containing the location of data files to analyze
# note that Windows needs "\\" between directory paths.
#   > files<-createFileList(2,25,"d","GCDC")
# use the convenience function called justDoIt to run all the
# individual steps automatically
#   > justDoIt(files)
##################################
# the following steps describe each function call within justDoIt
# so you can customize parameters as needed or add new calculations
##################################
# load the files into a single data matrix
#   > data<-loadFiles(files)
# extract the GPS data into a separate array
#   > gpsData<-extractGPSData(data)
# calculate the activity. In this case, use 60 seconds of accelerometer data
#   > gpsData<-calcActivity(data,gpsData,60)
# correct the UTC time to locat host time (not a necessary step)
#   > gpsData<-convertLocalTime(gpsData)
# export the results to a text file
#   > exportData(gpsData)
# send data to Google Earth for plotting
#   > createKML(gpsData)
##################################
# export files are automatically saved to the user directory
##################################
# it's possible to load just a subset of files, such as entries 4 thru 10:
#   > data<-loadFiles(files[4:10])
##################################
# subsets of data can be extracted and plotted easily.  For example, to view the
# movement between two dates 2021-05-26 04:40:26 CDT and 2021-05-26 15:28:40 CDT
# first create two integers representing the start/finish dates
#   > s<-as.integer(as.POSIXct("2021-05-26 04:40:26 CDT", origin="1970-01-01"))
#   > f<-as.integer(as.POSIXct("2021-05-26 15:28:40 CDT", origin="1970-01-01"))
# now extract the data from the main data set
#   > gps2<-subset(gps, gps$time>s & gps$time<f)
# then plot
#   > createKML(gps2)
# Note the time period is based on UTC but not the local time
#################################################################################

# DEPENDENCY LIBRARIES---------------------------------------------------------------

library(plotKML)   #plotKML is reliant on this library
library(rgdal)     #Write OGR is reliant on this library

#EARLY DECLARATIONS-----------------------------------------------------------
```

```
earth_Radius_Km <- 6371           #early constant declaration
deviceSN<-"null"                  #serial number of logger
startDate<-"null"                 #start date from first file
degrees <- 0                      #to avoid errors
text <- ("Distance in m: ")       #used to give better aesthetics to meter output
text2 <- ("Distance in Km: ")     #used to give better aesthetics to Kilometer output
text3 <- ("Files processed: ")    #used to show that 1 file has completed the 1st loop
text4 <- ("/")                    #used for aesthetic purposes and clarity.
distance_measurement <- "m"       #to avoid error
cow_walk_m <- array(0)            #needed as empty declaration
calculated_total_distance_m <- 0  #to avoid errors
header <- c("time", "ax", "ay", "az", "pres","temp", "TOW", "lat", "long", "height", "MSL", "hdop",
"vdop", "sd_mag", "local_time")
#ax, ay, and az is acceleration with out value in the x, y, and z directions.
#TOW is time of week. MSL is Mean Sea Level.
#hdop and vdop are Horizontal and Vertical dilution of Precision.
#sd_mag is the standard deviation of the magnitude of acceleration.

################################################################################
#createFileList is a simple function to build an array of text strings containing
#the files to analyze.  This function is designed to handle Windows OS file structure
#so double slahses must added between directories.  The function assumes the file
#prefix is "data-"
#start: the first numerical file name, ie 5 instead of 005
#finish: last numerical file
#dr: drive letter, no colon character
#dir: directory path to files
#example usage:
# fn<-createFileList(5, 25, "d","GCDC\\exp\\test1")
################################################################################
createFileList<-function(start, finish, dr, dir){
  dataFiles<-array()
  #prefix name of the data files
  filename<-"DATA-"
  g<-as.numeric(start)
  n<-1
  while(g<(as.numeric(finish)+1)){
    #add leading zeros if need to make valid file name
    if(g<10)
      temp<-paste("00",as.character(g),sep="")
    else if(g<100)
      temp<-paste("0",as.character(g),sep="")
    else
      temp<-as.character(g)
    name<-paste(filename, temp, ".csv", sep="")
    path<-paste(dr,":", "\\", dir, "\\", name, sep="")
    dataFiles[n]<-path
    g<-g+1
    n<-n+1
  }
  return(dataFiles)
}

################################################################
# loadFiles takes an array of file names and reads the data files
# into a single data matrix.
# filesnames: an array of text strings pointing to the data files
# returns a single array of all the data
################################################################
loadFiles<-function(filenames){
  i <- 1
  while(i < length(filenames)+1){

    data <- read.table(filenames[i], sep=",", comment=";", fill=TRUE, col.names = header)

    #MASTER_DATA IS DETERMINED-----------------------------------------------------
    if(i==1){
```

```
      MASTER_DATA <- data
      ####################################################
      #extract device data from file
      lines<-readLines(filenames[i],6)
      date <- unlist(strsplit(lines[3], ","))
      d <- paste(date[2], date[3])
      #convert the date/time text into a R type date
      startDate<<-as.POSIXlt(d)
      temp<-unlist(strsplit(lines[2],","))
      deviceSN<<-gsub("SN:","",trim(temp[5]))
    }
    else{
      MASTER_DATA <- rbind(MASTER_DATA,data)
    }

    print(paste(text3, i, text4, length(filenames)))
    i <- i + 1

  }  #END OF WHILE LOOP-------------------------------------------------------
  return(MASTER_DATA)
}

###############################################################################
# extractGPSData pulls the gps related information from the master list of data
# that was built from loadFiles.
# data: the complete matrix of data creadted by loadFiles
# returns a matrix of gps data as: time, lat, TOW, long, MSL, hdop, sd_mag, local_time
# note that sd_mag and local_time are filled in later with other function calls
###############################################################################
extractGPSData<-function(data){

  gpstime <- subset(data, select =c(time, lat, TOW, long, MSL, hdop, sd_mag, local_time), !is.na(TOW))

  #DATA REMOVAL IN THE CASE OF MULTIPLE SAMPLES EVERY SECOND----------------------
  #This section is for deleting extra data to prevent clutter in Google Earth.
  #Specifically, it compares 2 rows and removes the row with a higher hdop value
  g <- 1
  gpstime <- gpstime[-1,]
  while (g < nrow(gpstime)-1){
    dt <- gpstime[g+1,'time'] - gpstime[g,'time']
    if(abs(dt) < 1.5){
      if(gpstime[g+1,'hdop'] < gpstime[g,'hdop']){
        gpstime <- gpstime[-(g+1),]
      }
      if(gpstime[g+1,'hdop'] > gpstime[g,'hdop']){
        gpstime <- gpstime[-g,]
      }
      if(gpstime[g+1,'hdop'] == gpstime[g,'hdop']){
        gpstime <- gpstime[-g,]
      }
    }
    else{
      g <- g + 1
    }
  }

  return(gpstime)
}

###############################################################################
# reduceGPS reduces the size of the data array by keeping only a set interval of
# samples.  The original cow test recorded GPS every minute, which is an excessive
# amount of data for the purpose.  This function removes extra data to make the
# KML plot more easily viewable.
# gpsData: an array of data
# interval: number of samples to skip between kept samples
# returns a new array of data.  For example: interval=3 will save a sample, skip 3
```

```
# samples, save another sample, etc
################################################################################
reduceGPS<-function(gpsData, interval){
  j<-1
  gps<-array(0)
  while(j <= nrow(gpsData)){
    gps<-rbind(gps,gpsData[j,])
    j<-j+interval
  }
  gps<-gps[-1,]
  return(gps)
}

################################################################################
# calcActivity filles in the sd_mag column of the GPS data matrix created by
# extractGPSData.  sd_mad is the standard deviation of the magnetude vector.  Motion
# of the cow is represented as changes in acceleration data (Ax,Ay,Az).  More motion
# creates a higher standard deviation.  This is a very basic representation of activity
# data: the master data matrix that includes the acceleration values
# gps: the gps matrix that will be modified with the calculated sd values
# dt: the time interval for each activity value
# returns the gps matrix with the sd_mag column filled in
# example:
#   gps<-calcActivity(master_data, gps, 60)
#   this will take 60 seconds of acceleration data centered around the GPS sample time,
#   calculate the sd_mag, and add the result to the GPS sample row (column sd_mag)
################################################################################
calcActivity<-function(data, gps, dt){
  j<- 1
  while(j<=(nrow(gps))){

    #STADARD DEVIATION OF ACCELERATION (no value)-------------------------------
    # determine the time of the currect GPS location sample
    gtime<-gps[j,'time']
    #calculate the time period centered around the GPS aample
    btime<-gtime-(dt/2)
    etime<-gtime+dt/2
    #extract the appropriate section of accelerometer data
    accel <- subset(data, data$time<=etime & data$time>=btime)
    accel2 <- subset(accel, select =c(time, ax, ay, az))
    #calculate the vector magnetude of the XYZ acceleration
    mag2 <- accel2$ax*accel2$ax + accel2$ay*accel2$ay + accel2$az * accel2$az
    mag <- sqrt(mag2)
    #calculate the standard deviation for this segment of data
    sd_mag <- sd(mag)

    #If sd_mag is desired in a logarithmic manner remove the # from the line bellow
    #sd_mag <- 20*log(sd_mag, base = 10)

    #RELATES sd_mag TO gpstime
    gps[j,'sd_mag'] <- sd_mag

    j <- j + 1

  }#END OF WHILE LOOP---------------------------------------------------------
  return(gps)
}

################################################################################
# calcDistance uses the individual GPS points to determine the total distance traveled
# d: the GPS data matrix created by extractGPSData
# returns array of time versus distance traveled in meters
################################################################################
calcDistance<-function(gpsData){

  cow_walk_m <- array(c(gpsData[1, 1], 0))
```

```
    #DISTANCE LOOP-----------------------------------------------
    j <- 1
    long_and_lat <- gpsData
    coordinates(long_and_lat) <- c("long", "lat")
    lat1 <- (coordinates(long_and_lat)[j,2])
    long1 <- (coordinates(long_and_lat)[j,1])

    while (j <= nrow(gpsData)){
      #DISTANCE CALCULATION-----------------------------------------------------


      # lat1 <- (coordinates(long_and_lat)[j,2])
      # long1 <- (coordinates(long_and_lat)[j,1])

      if(j != nrow(gpsData)){
        lat2 <- (coordinates(long_and_lat)[j+1,2])
        long2 <- (coordinates(long_and_lat)[j+1,1])
      }
      distance_Lat = degrees_To_Radians(lat2-lat1);
      distance_Long = degrees_To_Radians(long2-long1);

      lat1d = degrees_To_Radians(lat1)
      lat2d = degrees_To_Radians(lat2)

      a <- sin(distance_Lat/2) * sin(distance_Lat/2) + sin(distance_Long/2) *
        sin(distance_Long/2) * cos(lat1d) * cos(lat2d)
      c <- 2 * atan2(sqrt(a), sqrt(1-a))

      distance_of_cow_Km <- earth_Radius_Km * c
      distance_of_cow_m <- distance_of_cow_Km * 1000

      #DISTANCE CORRECTION
      #if change in position is over 150m then the GPS data is bad
      #this code will remove erroneous distance data
      if (distance_of_cow_m > 150 ){
        distance_of_cow_m <- 0
      }

      #filter small movements less than hdop precision
      hdop<-gpsData[j, "hdop"]
      if(distance_of_cow_m < hdop){
        distance_of_cow_m = 0
        #j<-j+1
      }else{
        #j<-j+1
        lat1 <- lat2 #coordinates(long_and_lat)[j,2])
        long1 <- long2 #(coordinates(long_and_lat)[j,1])
      }

      j<-j+1

      cow_walk_m <- rbind(cow_walk_m, array(c(gpsData[j+1, 1], distance_of_cow_m)))

    }
    calculated_total_distance_m<<-sum(cow_walk_m[,2])
    print(paste("distanace traveled: ", calculated_total_distance_m))

    return(cow_walk_m)

}

##############################################################################
# data files always have the time stamp in UTC time.  This is determined from the
# GPS satellites.  convertLocalTime determines the local time based on the UTC value
# and the local time setting of the host computer.  The result is added to the
# local_time column of the gps data matrix.  For example, if your computer is set to
# Mountain Time, the function will convert the UTC to MT.
```

```
# gps: the gps data matrix
# returns the modified gps data matrix with the local_time column filled in
################################################################################
convertLocalTime<-function(gps){
  j<- 1
  while(j<=(nrow(gps))){

    #TIME CONVERSION--------------------------------------------------------------
    local_time <- as.POSIXct(gps[j, 'time'], origin = "1970-01-01")
    #local time is formatted as month-day-year hour:minute:second AM/PM timezone
    gps[j, 'local_time'] <- as.character.Date(local_time, "%m-%d-%Y %I:%M:%S %p %Z")

    j <- j + 1

  }
  return(gps)
}

################################################################################
# createKML takes the gps data matrix and builds the KML files for google earth
# GPS location points are represented by a bubble.  The bubble size and color
# represent the amount of activity calculated for that specific GPS point.
# gpsData: the gps data matrix with the sd_mag column completed
# the function will start Google Earth and automatically load the KML data
################################################################################
createKML<-function(data){

  #KML PRINTING-----------------------------------------------------------------
  #prints out all data as 2 KML files and a .csv file

  #retrieves information for both plotKML and write OGR
  coordinates(data) <- c("long", "lat")
  proj4string(data) <- CRS("+proj=longlat +datum=WGS84")

  #plotKML is used for plotting acceleration as a bubble that represents action through size and color
  name<-paste(getSN(),"_sd_mag.kml")
  plotKML(data["sd_mag"], LabelScale = 1, file.name=name)
  print("ACCELERATION AND GENERAL LOCATION KML PLOTTED")
  print("AUTOMATICALLY OPENING MASTER_DATA__sd_mag__.kml AND Google Earth Pro")
  cat("\n") #adds a blank space row.

  #write OGR is used for simply plotting a location with information
  locFile<-paste(getSN(),"_locations.kml")
  writeOGR(data, dsn=locFile, layer = ".", overwrite_layer = TRUE, driver="KML")
  print("INFORMATION AND PIN LOCATION KML PLOTTED")
  print("NOTICE: THE PIN KML MUST BE OPENED MANUALLY")

  cat("\n")
}

################################################################################
# createLocationKML takes the gps data matrix and builds the KML files for google earth
# gpsData: the gps data matrix with the sd_mag column completed
# the function will start Google Earth and automatically load the KML data
################################################################################
createLocationKML<-function(data){

  #KML PRINTING-----------------------------------------------------------------
  #prints out all data as 2 KML files and a .csv file

  #retrieves information for both plotKML and write OGR
  coordinates(data) <- c("long", "lat")
  proj4string(data) <- CRS("+proj=longlat +datum=WGS84")

  #plotKML is used for plotting acceleration as a bubble that represents action through size and color
  name<-paste(getSN(),"_location.kml")
  print("ACCELERATION AND GENERAL LOCATION KML PLOTTED")
```

```
  print("AUTOMATICALLY OPENING Location Data in Google Earth Pro")


  #write OGR is used for simply plotting a location with information
  locFile<-paste(getSN(),"_GPS_loc.kml")
  writeOGR(data, dsn=locFile, layer = ".", overwrite_layer = TRUE, driver="KML")

  cat("\n")
}

################################################################################
# exportData saves the gps data matrix into a csv text file
# gpsData: the completed gps data matrix
################################################################################
exportData<-function(gpsData){
  sn<-getSN()
  d<-gsub(" ","_", date())
  d<-gsub(":","-",d)
  filename<-paste(sn, "_Results_", d, ".csv")
  write.csv(gpsData, file = filename, row.names=FALSE)
  print(".CSV FILE -> Total Data Set.csv <- HAS BEEN CREATED")
}

#DEGREES TO RADIANS FUNCTION-----------------------------------------------
################################################################################
# convenience function to convert degrees to radians
# degrees: value in degree units
# returns value converted to radians
################################################################################
degrees_To_Radians <- function(degrees) {
  radians <- (degrees * pi / 180)
  return(radians)
}

################################################################################
# because R doesn't have a trim function!?!
# use trim to remove leading and trailing whitespace from a text string
################################################################################
trim <- function( x ) {
  gsub("(^[[:space:]]+|[[:space:]]+$)", "", x)
}

################################################################################
# function access to get distance the cow travelled
################################################################################
getDistance_m<-function(){
  return(calculated_total_distance_m)
}
################################################################################
# return the start date as recorded in the header of the first file
# Warning: the logger will correct the RTC based on the GPS clock so the start date
# may change if the RTC was not originally sync'd
# this date is used to create a unique file name when exporting the results
################################################################################
getStartDate<-function(){
  return(startDate)
}

################################################################################
# return the serial number included in the first file
################################################################################
getSN<-function(){
  return(deviceSN)
}

################################################################################
# return the median hdop value of all the GPS locations
```

```
################################################################################
getMedianHDOP<-function(gps){
  return(median(gps[,"hdop"]))
}

################################################################################
# return the maximum hdop value (least accurate GPS lock)
################################################################################
getMaxHDOP<-function(gps){
  return(max(gps[,"hdop"]))
}

################################################################################
# use this to filter out GPS samples with hdop greater than a given limit
# returns a new GPS array with bad locations removed
################################################################################
removeGPS<-function(gps, hdop){
  j<-1
  while(j <= nrows(gps)){
    if(gps[j,"hdop"] > hdop)
      gps<-gps[-j,]
    j<-j+1
  }
  return(gps)
}

################################################################################
# Convenience funtion to run all the steps together.  Note that all calculations are
# run within this function so the data arrays are lost once the function completes
# The results are saved to a csv and kml file.
################################################################################
justDoIt<-function(files){
  ########################################
  # these steps are discussed in the notes #
  # at the start of this script file       #
  ########################################
  data <- loadFiles(files)
  gpsData <- extractGPSData(data)
  gpsData <- calcActivity(data, gpsData, 60)
  gpsData <- convertLocalTime(gpsData)
  exportData(gpsData)
  createKML(gpsData)

}

test<-function(gpsData, limit){
  j<- 1
  distance<-array(0)
  while(j < limit){
    g2<-reduceGPS(gpsData,j)
    d<-calcDistance(g2)
    dist<-sum(d[,2])
    distance<-rbind(distance, c(j,dist))
    j<-j+1
  }
  distance<-distance[-1,]
  return(distance)
}



#END
```

End of Document