Protocol for generating distance data from camera trap images using a simple computer vision approach

Marcus Rowcliffe

20 June 2019

## Background

The random encounter model, distance sampling and related methods require data on animal positions relative to camera in order to estimate camera detection zone size and (for REM) animal speed of movement. This vignette sets out a process for generating animal position and speed data from camera trap images using:  
1. Images of calibration objects to inform computer vision models;

1. The html tool **animaltracker** to digitise images (see <https://robinfreeman.github.io/animaltracker>);
2. Work-in-progress R package **CTtracking** to fit calibration models and predict animal positions and speeds (see <https://github.com/MarcusRowcliffe/CTtracking>).

## 1. Making calibration images

### Camera calibration

The goal is to take pictures of objects of known size at a range of known distances from the camera in order to calculate the camera’s intrinsic properties, which then allow us to calculate the positions of objects of known size, as used in site calibration. This needs to be done for each combination of camera model and image resolution setting used in the field (Note: it’s best to keep to keep image resolution consistent throughout; if you do this, and use a consistent camera model, you only need to calibrate one camera). The steps are as follows:

1. Make calibration pole(s), with regular length intervals clearly marked. A 1 m pole with 20 cm intervals marked works well (Fig. 1). Calibration can either be done using a static array of poles (Fig. 1a), or using a single pole moved to different positions in front of the camera (Fig. 1b). A static array is harder to set up, but is quicker to take images once in place.

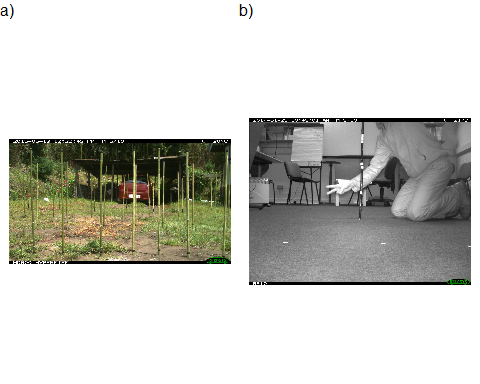


Fig. 1. Example images of camera calibration poles for a) a static array, and b) a single mobile pole

1. On an open arena, mark out at least nine positions at a range of different radial and angular distances from the camera, measuring the distances from camera accurately. Fig. 2 gives an example of placement positions, with poles at three distances (1, 2 and 4 m), and a range of angles. It’s not necessary to measure angle, but it should be variable, and within the camera’s field of view (usually about 20 degrees either side of the mid line, but you may need to check the effective field of view for your camera and setting). If using a static pole array, firmly fix a pole at each position.

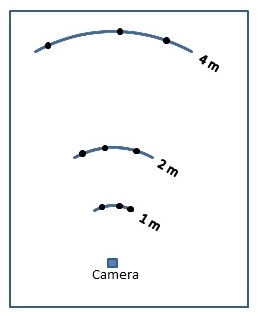


Fig. 2. Diagram of an example layout for a camera calibration pole grid.

1. With a camera positioned in front of the arena and armed, either take one image of the static pole array, or take one image of the moveble pole at each position on the array. In the latter case, holding up some visible marker of the distance. For example, in Fig. 1b, the pole is placed at 2 m from the camera, with distance indicated by fingers displayed.

### Site calibration

Site calibration models can be created either in a single step, using calibration poles placed at known distance from camera, or with an initial step to fit a camera calibration model, which is then used to calculate site calibration pole distances. The advantage of the latter is that it is not necessary to measure distances when setting up cameras in the field. *Further details to follow*

## 2. Digitising images

### Animaltracker basics

1. Organise images into folders specific to the deployment (for animal and site calibration images) or camera (for camera calibration images), naming folders with unique identifiers.
2. Open animal tracker: <https://robinfreeman.github.io/animaltracker>
3. Click *Browse* -> navigate to your chosen image directory -> select all images (*Ctrl+a*) -> OK
4. Digitise points:

* Scroll through images using Next / Previous buttons or -> / <- cursor keys to find those you want to digitise.
* To digitise a point or sequence: click *Add sequence* (keyboard shortcut *a*) and click on the desired point on each image until the sequence ends. Type any required data (e.g. species) into the annotation box (replacing the default “new” text). BEWARE USING KEYBOARD SHORTCUTS WHEN THE CURSOR IS ACTIVE IN THE ANNOTATION BOX. Note that it is not possible to digitise a multi-point sequence on a single image.
* If you want to delete or move a point, scroll back to the image on which the point was digitised (you’ll see it highlighted when viewing the correct image) and click *Add mode* (keyboard shortcut *m*), then click and drag the point to move it, or click *Delete point* (keyboard shortcut d) to delete. Once finished, click *Edit mode* to go back and continue digitising.
* To show or hide a sequence, click on its ‘eye’ icon; to delete a sequence, click on its ‘x’ icon.

1. When finished digitising a set of images, click *Export CSV* -> *Open* -> *OK* -> save the resulting file to the the relevant directory with an informative name (see below on organising and naming different types of data). If you need to take a break and shut down before completing a set of images, make a note of the last image digitised. Then, when re-starting, open images from that point for the next session, export the csv file as above when ready, and cut and paste the resulting data into the partially completed csv file created earlier.
2. To start a new image directory, start a new browser tab before repeating from step 2 (this avoids a bug that causes new image sets in the same tab to open from the last image number in the previous set).

### Camera calibration

1. For each pole, digitise two points a known height above ground, each as a new single-point sequence, and for each recording the following in the annotations window:

* distance from camera in metres;
* height of digitised point in metres.

Separate the entries with semicolons. For example, in Fig. 1b, the pole is 2 m from the camera, and the uppermost marker is 60 cm above the ground, so digitising the pole base and upper marker would respectively be annotated as 2;0 and 2;0.6. Make sure that the different types of information are consistently entered in the same order. If an array of poles is used, and and a single image digitised, you would also need a unique identifier for each pole, in which case an example annotation might be 2;0;pole\_2A.

1. Save data for each camera in a separate csv file, named with the unique identifier matching the name of the folder from which the images came, and placed in a single camera calibration data folder.

### Site calibration

1. For each pole seen to be in position resting on the ground, digitise the upper- and lower-most visible reference points, each as a new single-point sequence. For each point, record the height of the point in metres in the sequence annotation box (a single number). If the single-step calibration process is used, with measured pole distances, also record pole distance in metres in the annotation box, with data separated by a semicolon. For example, if a point 60 cm high on a pole 3.5 m away is digitised, record 0.6;3.5. When recording multiple entries like this, take care to ensure that the height and distance are consistently entered in the same order.
2. Save data for each deployment in a separate csv file, named with the unique identifier matching the name of the folder from which the images came, and placed in a single deploymnet calibration data folder.

### Animal sequences

1. Digitise animal positions at points on the ground directly beneath the animals’ centre of mass, as best you can judge.
2. If the animal is only partially visible, digitise the position on the canvas outside the image itself. To do this, you may need to move (click and drag) or zoom (mouse scroll wheel) the image to make space on the canvas. Click *Reset view* (keyboard shortcut *r*) to return image to full size following this.
3. During digitisations, look out for moments at which the camera clearly changes its orientation. It may be necessary to truncate the deployment at that time, as positions for any points digitised after that cannot be accurately estimated unless there has been a subsequent set of calibration images taken that apply to that period. If you know there have been no applicable pole images taken, the deployment end time will need to be updated to the point at which the shift is first seen, and there is no need to digitise any further animal positions. If on the other hand there has been a cubsequent set of calibration images taken, you will need to create a new deployment, separating the images into separate folders and creating separate digitisation data files.

## 3. Generating position and speed data

The CTtracking package is under development and not (yet) available on CRAN. Downnload the code file CTtracking.r from <https://github.com/MarcusRowcliffe/CTtracking>, and load it, (first adding full path to the file name, or setting the working directory):

source("CTtracking.r")

### Fitting camera calibration models

This step is necessary if site calibration images are collected without distance measurements. A camera calibration model is needed for each type of camera, where type is defined by the combination of camera parameters and image resolution setting. In practice, camera parameters may vary between makes, but are likely to be consistent across minor model variations within makes (although this may be worth checking). To check which types exist in your data, it may help to create the camera lookup table described under step 3 of the following section on fitting deployment calibration models. Running unique(camtable) will then list the types. The steps to fit camera calibration models are:

1. Create a dataframe of metadata from the digitised camera calibration images. The CTtracking function read.exif can be used, but before using it for the first time, you need to download the exiftool executable to your computer. For windows (the function isn’t currently geared up for Mac OS), rename the executable file exiftool.exe and place in a new directory named C:/Exiftool. You can then run read.exif with a path to the image folder as the sole argument, for example:

path <- "D:/Survey\_xxx"  
exifdat.cam <- read.exif(file.path(path, "CameraImages"))

1. Create a dataframe of digitisation data, pooling across camera-specific files, using read.digidat, with arguments:

* path: a path to the folder containing the csv files
* exifdat: the dataframe of exifdata created above
* annotations: a set of column headings for the data extracted from sequence annotations
* pair: set to TRUE, pairs up points from the same pole to create a dataframe with one row per pole

camdat <- read.digidat(file.path(path, "CameraData"),   
 exifdat=exifdat.cam,   
 annotations=c("distance", "height"),   
 pair=TRUE)

1. Fit the camera calibration models using cal.cam. Supply the digitisation dataframe created above as the sole argument:

cmods <- cal.cam(camdat)

1. Check model fit before proceeding, by plotting the resulting models. Two plots are created for each camera:

* A plot of length (m) per pixel as a function of distance from camera, with points shaded by distance from the image centre, and fitted lines for the edge and centre of the image.
* A plot of poles as they appeared on the image according to the data supplied, with shading coded by distance.

Any clearly out of place points or poles may indicate a digitising or pole placement problem that needs fixing. In this case the fit seems good.

plot(cmods)

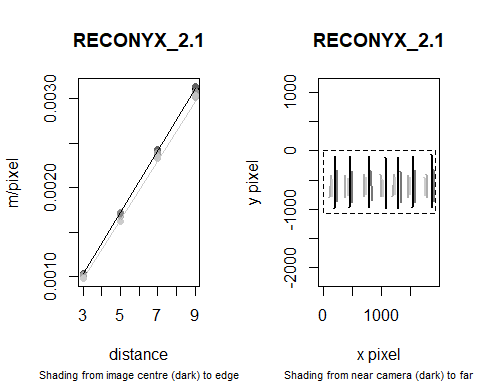


Fig. 3. Diagnostic plots for a camera calibration model

### Fitting deployment calibration models

The steps required are essentially the same as those for camera calibration above:

1. Create a dataframe of metadata from the deployment calibration (and animal) images. This will take some time, so you may want to go and get a cup of tea, then save the resulting dataframe for re-loading in subsequent sessions, rather than having to re-run metadata extraction. The process may also run out of memory and abort if number of images is very large. In this case you may need to reduce the size of the task by separating out the digitised images if appropriate, or processing the images in batches.

##Run this the first time:  
#exifdat <- read.exif(file.path(path, "DeploymentImages"))  
#write.csv(exifdat, "D:/RegentsPark17/exifdata.csv", row.names = FALSE)  
  
##In subsequent sessions just run this:  
exifdat <- read.csv(file.path(path, "exifdata.csv"), stringsAsFactors = FALSE)

1. Create a dataframe of digitisation data, pooling across csv files from multiple deployments using read.digidat, with arguments:

* path: a path to the folder containing the csv files
* exifdat: the dataframe of exifdata created above
* exifcols: additional data columns to extract from exifdat (these will be needed in the next step)
* annotations: a set of column headings for the data extracted from sequence annotations, in this case simply height
* pair: set to TRUE, pairs up points from the same pole to create a dataframe with one row per pole

There may be a warning here that flags up some problematic poles (e.g. ony one, or more than two points apparently digitised on a single pole). You may want go back and check for errors in the digitisation and fix these before proceeeding.

caldat <- read.digidat(file.path(path, "DeploymentData"),   
 exifdat=exifdat,  
 exifcols=c("Make", "Megapixels"),  
 annotations="height",  
 pair=TRUE)

1. Fit the deployment calibration models using cal.site, supplying the pole digitisation dataframe created above as the sole argument. To do this, you first need to create a lookup table, specifying which camera type (defined by make/model and/or image resolution setting) is used at each deployment (even if the same type is used for all deployments). There may be warnings here if some models could not be fitted (e.g. if fewer than three poles were digitised).

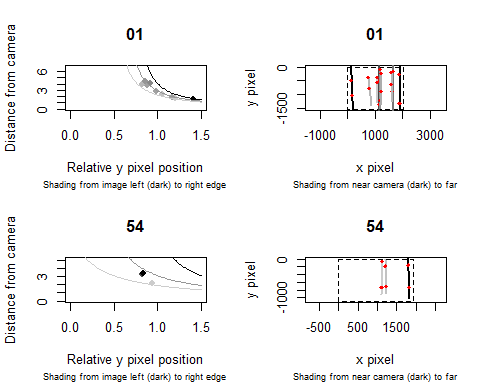
camtable <- make.camtable(caldat, camcolumns=c("Make","Megapixels"))  
smods <- cal.site(caldat, cmods, camtable)

1. Check model fit before proceeding by plotting the resulting models. Two plots are created for each deployment:

* A plot of distance from camera as a function of y pixel position, with points shaded by x pixel position, and fitted lines at the edges and centre of the image.
* A plot of poles as they appeared on the image according to the data supplied, with shading coded by distance and digitised points indicated in red.

Any clearly out of place points or poles may indicate a digitising problem that needs fixing. In the figure below, deployment 01 has a good fit, but deployment 54 has a very poor fit: only three poles were digitised, and the deployment may have been on uneven ground. In this case, it is unlikely that the fit could be improved by revisiting the digitisation, and it may be sensible to remove all animal records at this deployment from analysis in the next step.

plot(smods)



### Generating animal position and speed data

1. Create a dataframe of animal digitisation data using read.digidat, this time with arguments only for path, exifdat and annotations. If any images specified in the digitisation data could not be found in the exifdat metadata, this will be stripped out of the result with a warning.

animdat <- read.digidat(file.path(path, "AnimalData"),  
 exifdat=exifdat,  
 annotations="species")

1. Using the animal digitisation dataframe and deployment calibration models created above, add predicted radial and angular positions for each point to the dataframe using function predict.pos. If any records in the digitisation data come from deployments that do not have a matching deployment calibration model, these records will be stripped out with a warning.

posdat <- predict.pos(animdat, smods)

It is likely that radial distances for some records will have been extremely poorly estimated, so it is worth exploring the distribution, perhaps going back to the digitisations to check for problems (e.g. substantial shift in camera orientation during the deployment), and removing any remaining extreme observations.

1. Using the augmented dataframe created in the last step, create a dataframe summarising sequences with distances travelled over given times and the resulting speed estimates.

seqdat <- seq.summary(posdat)