Protocol for generating distance data from camera trap images using a simple computer vision approach, CTtracking V0.2

Marcus Rowcliffe

18 November 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;
2. The html tool **animaltracker** to digitise images (see https://lauravzarco.github.io/animaltracker);
3. 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 a calibration pole, with regular length intervals clearly marked. A 1 m pole with 20 cm intervals marked works well (Fig. 1). Note that the gradations on this pole are indicated by the number of bands, so for example, the top of the triple band is at 60 cm from the tip, whereas the single band is at 20 cm.
2. On an open arena, mark out nine or more 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).
3. With a camera positioned in front of the arena and armed, take one image of the pole at each position on the array, holding up some visible marker of the distance. For example, in Fig. 1, the pole is placed at 2 m from the camera, with distance indicated in metres by the number of fingers displayed.

Figure 1: Example image of a camera calibration pole in position 2 m from the camera.

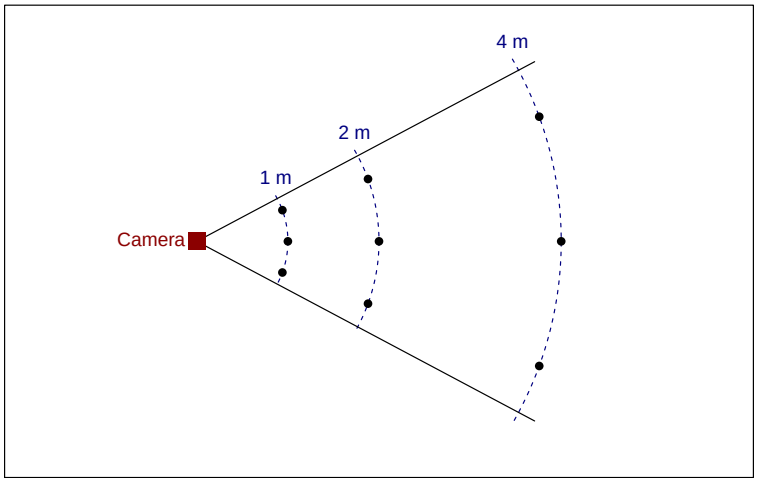

Figure 2: Diagram of an example layout for a camera calibration pole grid.

Figure 3: Example image of a deployment calibration pole placement.

Deployment calibration

The goal is to take images of calibration poles in the field, as above but without having to measure distance. Using the same calibration pole as above, carry out the following procedure at each camera deployment, either when setting, checking or removing the camera. Note that if the camera is changed or moved, even slightly, the calibration process should be repeated for that deployment.

1. With the camera in its final operating position, arm the camera.
2. Starting about 1m directly in front of the camera, hold the pole vertically with its base on the ground so that it is clearly visible by the camera, and held still long enough to ensure a clear image (generally a few seconds). Fig. 3 shows an example of a resulting image.
3. Repeat this for further pole placements across the field of view and away from the camera with placements spaced about 1 m apart. Continue away from the camera to the maximum extent that any animals are likely to be captured, or if possible a bit beyond. As you reach greater distances, it may help to have a second person next to the camera to keep it triggering. Aim for 10-20 placements, perhaps more if visibility is very good Fig. 4 shows an example configuration of deployment calibration pole placements.

2. Digitising images

Animaltracker basics

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

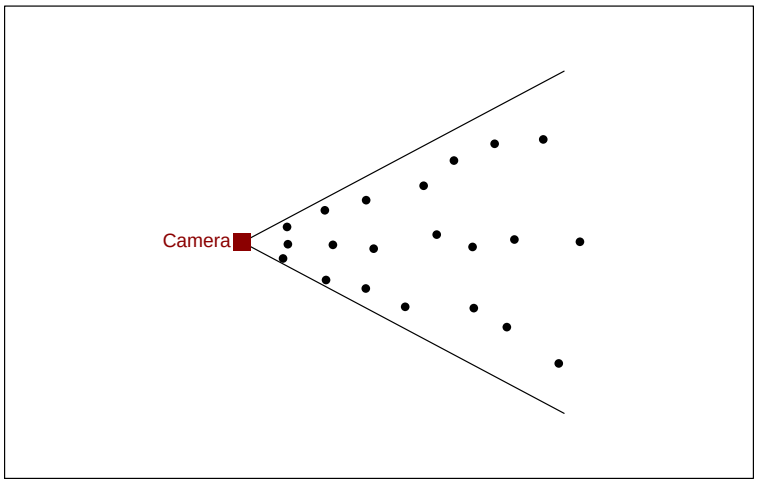

Figure 4: Diagram of an example set of pole locations for a deployment calibration. Note this can be set out without measuring anything.

- 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* and click on the desired point on each image until the sequence ends. Note that it is not possible to digitise a multi-point sequence on a single image.
- Once you have digitised a point, a data table will appear. Use the default *name* field to tag species, and add any additional fields by typing the field name into the box above the table and clicking *Add field to all sequences*.
- 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*, then click and drag the point to move it, or click *Delete point* 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.
- Use the mouse scroll wheel to zoom the image, or click and drag to move it. The *Reset view* button returns the image to full screen.

5. 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.

6. To start a new image directory, refresh your browser and repeat from step 3.

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:

- distance from camera in metres, in a new field called "distance";
- height of digitised point in metres, in a new field called "height".

2. 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.

Deployment 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 a new field called "height".
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 or zoom the image to make space on the canvas. Fig. 7 shows an example of this.
3. At least the first position of every individual animal sequence should be digitised, but it is not necessary to digitise every subsequent image. Only digitise movement sequences if:

- the animal shows significant change in position over the sequence; AND for as long as:
- the animal is continually observed with no more than 2 seconds or so between images; AND
- the animal's position is reasonably easy to locate within the image.

4. 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")
```

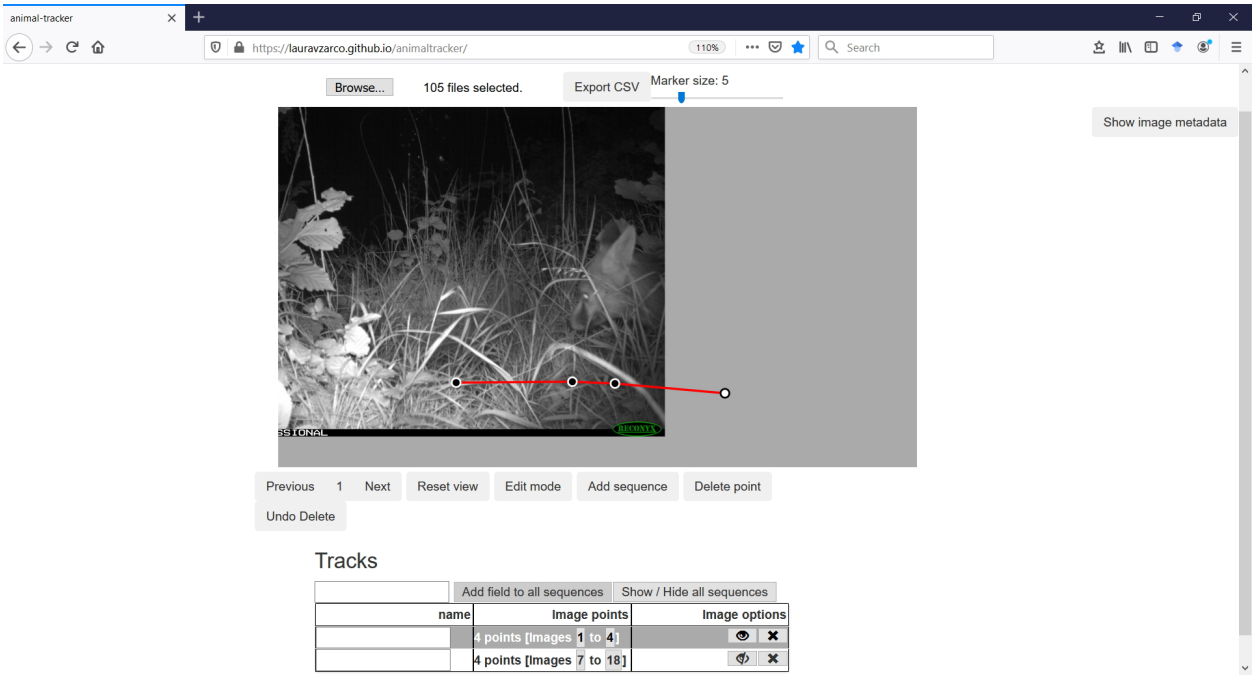

Figure 5: Animaltracker screenshot, showing the first image of a sequence with the animal only partially visible, and the subsequently digitised track showing.

Fitting camera calibration models

This step is necessary if (as recommended above) deployment 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). 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 (https://www.sno.phy.queensu.ca/~phil/exiftool). 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:

```
exifdat.cam <- read.exif("./Survey_yyy/CameraImages")
```

2. 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
- datatype: "pole" indicates that the input data contains only pole digitisations

```
camdat <- read.digidat("./Survey_yyy/CameraData",
                       exifdat=exifdat.cam,
                       datatype="pole")
```

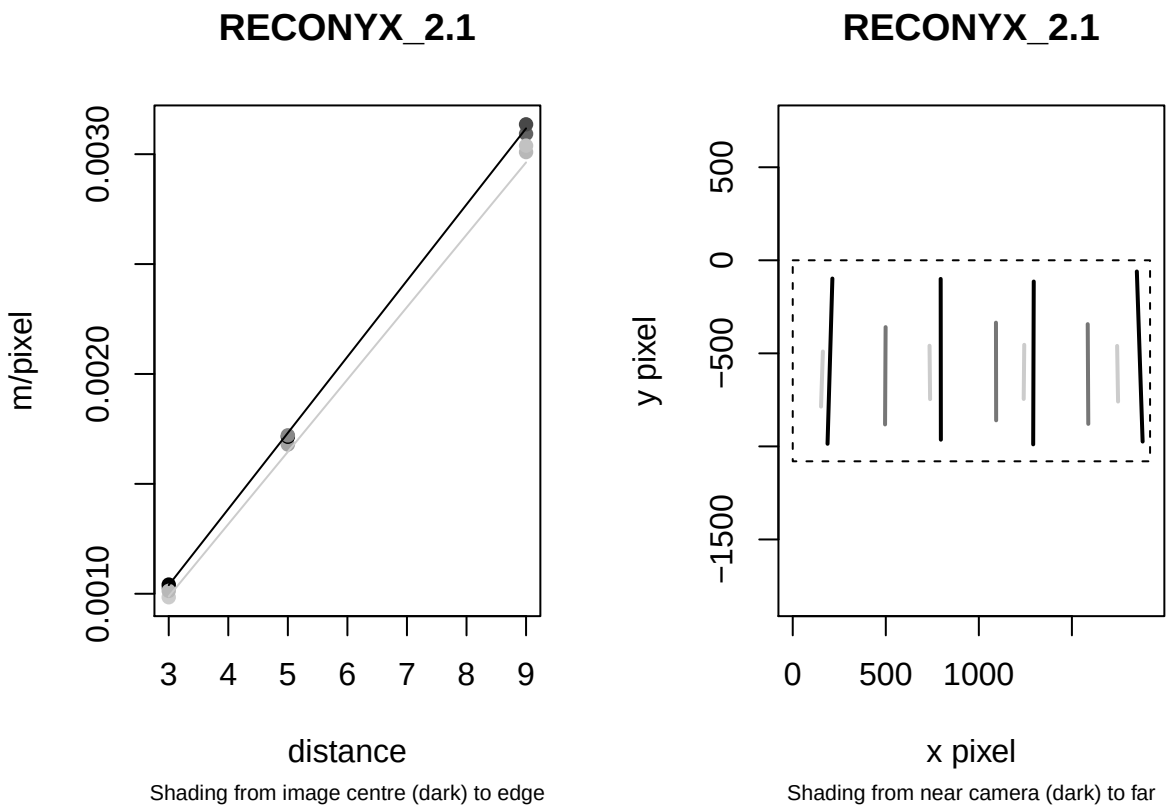

Figure 6: Diagnostic plots for a camera calibration model

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

```
cmods <- cal.cam(camdat)
```

4. 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)
```

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("./Survey_yyy/DeploymentImages"))
#write.csv(exifdat, "D:/RegentsPark17/exifdata.csv", row.names = FALSE)

##In subsequent sessions just run this:
exifdat <- read.csv("./Survey_yyy/exifdata.csv", stringsAsFactors = FALSE)
```

2. Create a dataframe of deployment 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
- datatype: "both" indicates that the input data contains both pole and animal digitisations. In this case, the output is a list with elements *animal* and *pole*, respectively dataframes of animal and pole digitisation data.

There may be a warning here that flags up some problematic poles (e.g. only 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 proceeding.

```
depdat <- read.digidat(path="./Survey_yyy/DeploymentData",
                       exifdat=exifdat,
                       datatype="both")
```

3. 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 deployment table, specifying which camera type (as named in the camera calibration models) was used at each deployment (even if the same type is used for all deployments). In the example below, note the specific field names, which must be exactly as given (deployment start and stop times/dates are also given but are not used at this stage in the analysis). `cal.site` may throw warnings here if some models could not be fitted (e.g. if fewer than three poles were digitised).

```
deptab <- read.csv("./Survey_yyy/deptable.csv")
head(deptab)
```

```
##   deploy_id               start                stop      cam_id
## 1    Site01 2017:10:02 09:06:43 2017:10:13 06:11:46 RECONYX_2.1
## 2    Site02 2017:10:23 20:10:55 2017:10:29 05:17:38 RECONYX_3.1
```

```
smods <- cal.site(depdat$pole, cmods, deptab)
```

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

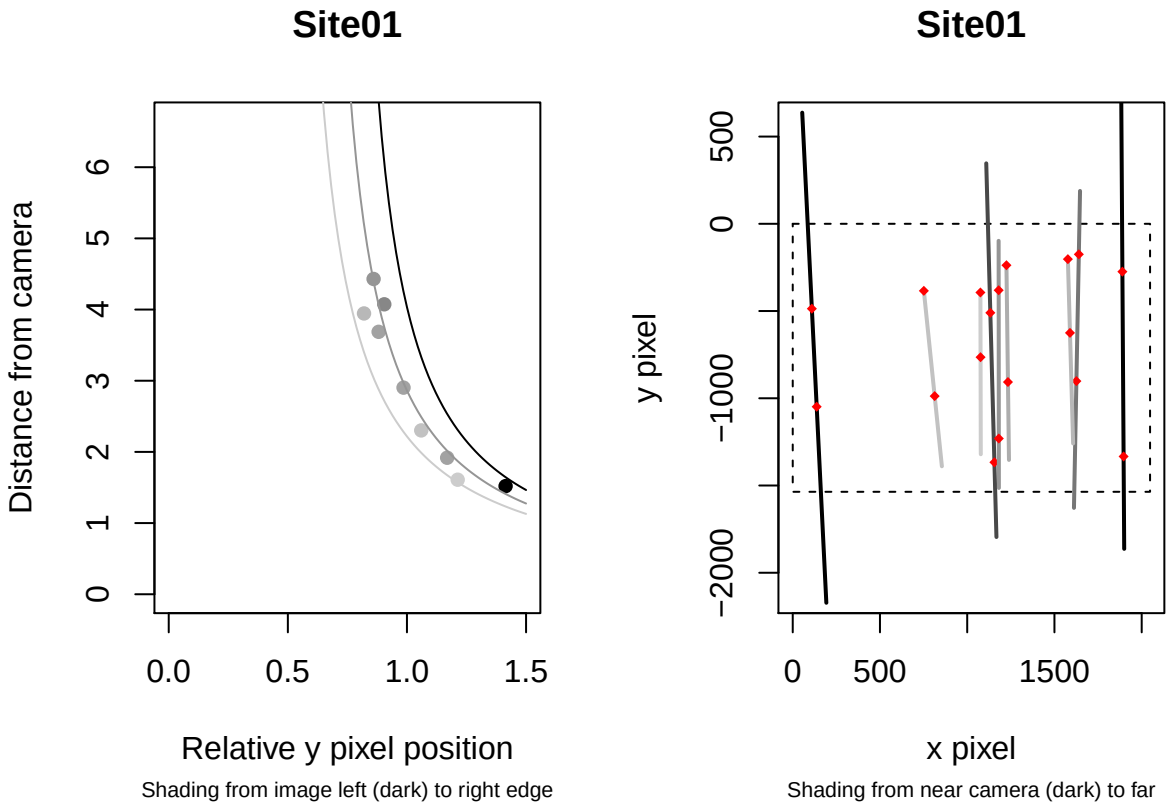

Figure 7: Diagnostic plots for a deployment calibration model

- 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 reasonable fit.

```
plot(smods)
```

Generating animal position and speed data

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(depdat$animal, smods)
```

It is likely that radial distances for some records will have been poorly estimated, for example where animals move beyond the deployment calibration grid, so it is worth exploring the distributions of distance and angle

estimates, perhaps going back to the digitisations to check for problems (e.g. substantial shift in camera orientation during the deployment), and judiciously removing any remaining extreme observations from subsequent analyses.

2. 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)
```