Chapter 8
Data Quality: Detection and Management of Outliers

Ferdinando Urbano, Mathieu Basille and Francesca Cagnacci

Abstract Tracking data can potentially be affected by a large set of errors in different steps of data acquisition and processing. Erroneous data can heavily affect analysis, leading to biased inference and misleading wildlife management/conservation suggestions. Data quality assessment is therefore a key step in data management. In this chapter, we especially deal with biased locations, or 'outliers'. While in some cases incorrect data are evident, in many situations, it is not possible to clearly identify locations as outliers because although they are suspicious (e.g. long distances covered by animals in a short time or repeated extreme values), they might still be correct, leaving a margin of uncertainty. In this chapter, different potential errors are identified and a general approach to managing outliers is proposed that tags records rather than deleting them. According to this approach, practical methods to find and mark errors are illustrated on the database created in Chaps. 2, 3, 4, 5, 6 and 7.

Keywords Outlier detection · GPS accuracy · Animal movement · Erroneous data

F. Urbano (✉)
Università Iuav di Venezia, Santa Croce 191 Tolentini, 30135 Venice, Italy
e-mail: ferdi.urbano@gmail.com

M. Basille
Fort Lauderdale Research and Education Center, University of Florida,
3205 College Avenue, Fort Lauderdale, FL 33314, USA
e-mail: basille@ase-research.org

F. Cagnacci
Biodiversity and Molecular Ecology Department, Research and Innovation Centre,
Fondazione Edmund Mach, via E. Mach 1, 38010 S.Michele all'Adige, TN, Italy
e-mail: francesca.cagnacci@fmach.it

F. Urbano and F. Cagnacci (eds.), *Spatial Database for GPS Wildlife Tracking Data*,
DOI: 10.1007/978-3-319-03743-1_8,

Introduction

Tracking data can potentially be affected by a large set of errors in different steps of data acquisition and processing, involving malfunctioning or poor performance of the sensor device that may affect measurement, acquisition and recording; failure of transmission hardware or lack of transmission due to network or physical conditions; and errors in data handling and processing. Erroneous location data can substantially affect analysis related to ecological or conservation questions, thus leading to biased inference and misleading wildlife management/conservation conclusions. The nature of positioning error is variable (see Special Topic), but whatever the source and type of errors, they have to be taken into account. Indeed, data quality assessment is a key step in data management.

In this chapter, we especially deal with biased locations or 'outliers'. While in some cases incorrect data are evident, in many situations it is not possible to clearly identify locations as outliers because although they are suspicious (e.g. long distances covered by animals in a short time or repeated extreme values), they might still be correct, leaving a margin of uncertainty. For example, it is evident from Fig. 8.1 that there are at least three points of the GPS data set with clearly incorrect coordinates.

In the exercise presented in this chapter, different potential errors are identified. A general approach to managing outliers is proposed that tags records rather than deleting them. According to this approach, practical methods to find and mark errors are illustrated.

Review of Errors that Can Affect GPS Tracking Data

The following are some of the main errors that can potentially affect data acquired from GPS sensors (points 1 to 5), and that can be classified as GPS location bias, i.e. due to a malfunctioning of the GPS sensor that generates locations with low accuracy (points 6 to 9):

1. Missing records. This means that no information (not even the acquisition time) has been received from the sensor, although it was planned by the acquisition schedule.
2. Records with missing coordinates. In this case, there is a GPS failure probably due to bad GPS coverage or canopy closure. In this case, the information on acquisition time is still valid, even if no coordinates are provided. This corresponds to 'fix rate' error (see Special Topic).
3. Multiple records with the same acquisition time. This has no physical meaning and is a clear error. The main problem here is to decide which record (if any) is correct.
4. Records that contain different values when acquired using different data transfer procedures (e.g. direct download from the sensor through a cable vs. data transmission through the GSM network).

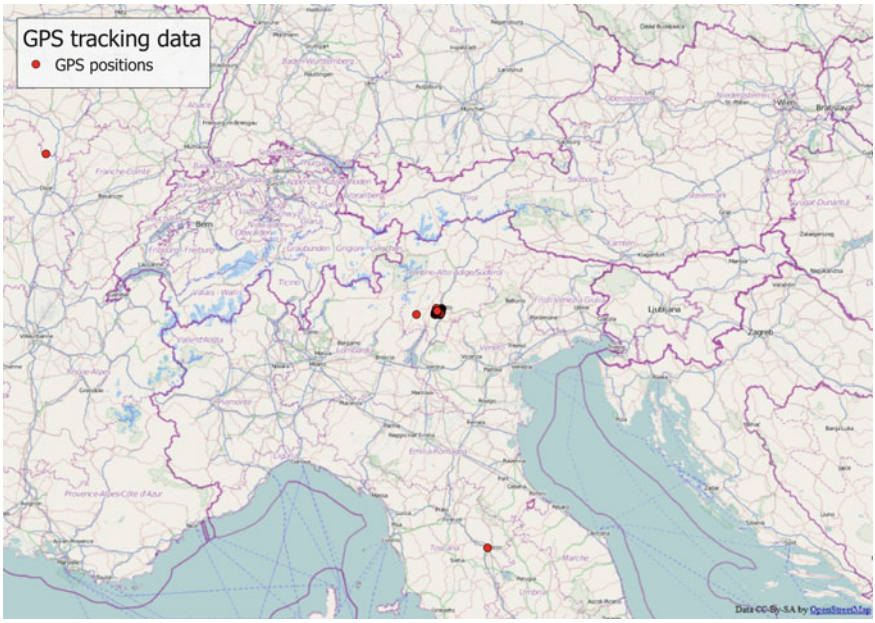

Fig. 8.1 Visualisation of the GPS data set, where three points are evident erroneous positions

5. Records erroneously attributed to an animal because of inexact deployment information. This case is frequent and is usually due to an imprecise definition of the deployment time range of the sensor on the animal. A typical result is locations in the scientist's office followed by a trajectory along the road to the point of capture.
6. Records located outside the study area. In this case, coordinates are incorrect (probably due to malfunctioning of the GPS sensor) and outliers appear very far from the other (valid) locations. This is a special case of impossible movements where the erroneous location is detected even with a simple visual exploration. This can be considered an extreme case of location bias, in terms of accuracy (see Special Topic).
7. Records located in impossible places. This might include (depending on species) sea, lakes or otherwise inaccessible places. Again, the error can be attributed to GPS sensor bias.
8. Records that imply impossible movements (e.g. very long displacements, requiring movement at a speed impossible for the species). In this case, some assumptions on the movement model must be made (e.g. maximum speed).
9. Records that imply improbable movements. In this case, although the movement is physically possible according to the threshold defined, the likelihood of the movement is so low that it raises serious doubts about its reliability. Once the location is tagged as suspicious, analysts can decide whether it should be considered in specific analyses.

GPS sensors usually record other ancillary information that can vary according to vendors and models. Detection of errors in the acquisition of these attributes is not treated here. Examples are the number of satellites used to estimate the position, the dilution of precision (DOP), the temperatures as measured by the sensor associated with the GPS and the altitude estimated by the GPS. Temperature is measured close to the body of the animal, while altitude is not measured on the geoid but as the distance from the centre of the earth: thus in both cases the measure is affected by large errors.

Special Topic: **Dealing with localisation errors associated with the GPS sensor**

A source of uncertainty associated with GPS data is the positioning error of the sensor. GPS error can be classified as bias (i.e. average distance between the 'true location' and the estimated location, where the average value represents the accuracy while the measure of dispersion of repeated measures represents the precision) and fix rate, or the proportion of expected fixes (i.e. those expected according to the schedule of positioning that is programmed on the GPS unit) compared to the number of fixes actually obtained. Both these types of errors are related to several factors, including tag brand, orientation, fix interval (e.g. cold/warm or hot start), and topography and canopy closure. Unfortunately, the relationship between animals and the latter two factors is the subject of a fundamental branch of spatial ecology: habitat selection studies. In extreme synthesis, habitat selection models establish a relationship between the habitat used by animals (estimated by acquired locations) versus available proportion of habitat (e.g. random locations throughout study area or home range). Therefore, a habitat-biased proportion of fixes due to instrumental error may hamper the inferential powers of habitat selection models. A series of solutions have been proposed. For a comprehensive review see Frair et al. (2010). Among others, a robust methodology is the use of spatial predictive models for the probability of GPS acquisition, usually based on dedicated local tests, the so-called Pfix. Data can then be weighted by the inverse of Pfix, so that positions taken in difficult-to-estimate locations are weighted more. In general, it is extremely important to account for GPS bias, especially in resource selection models.

A General Approach to the Management of Erroneous Locations

Once erroneous records are detected, the suggested approach is to keep a copy of all the information as it comes from the sensors (in *gps_data* table), and then mark records affected by each of the possible errors using different tags in the table where locations are associated with animals (*gps_data_animals*). Removing data seems often not so much a problem with GPS data sets, since you probably have thousands of locations anyway. Although keeping incorrect data as valid could be much more of a problem and bias further analyses, suspicious locations, if correct, might be exactly the information needed for a specific analysis (e.g. rutting excursions). The use of a tag to identify the reliability of each record can solve these problems. Records should never be deleted from the data set even when they are completely wrong, for the following reasons:

- If you detect errors with automatic procedures, it is always a good idea to be able to manually check the results to be sure that the method performed as expected, which is not possible if you delete the records.
- If you delete a record, whenever you have to resynchronise your data set with the original source, you will reintroduce the outlier, particularly for erroneous locations that cannot be automatically detected.
- A record can have some values that are wrong (e.g. coordinates), but others that are valid and useful (e.g. timestamp).
- The fact that the record is an outlier is valuable information that you do not want to lose (e.g. you might want to know the success rate of the sensor according to the different types of errors).
- It is important to differentiate missing locations (no data from sensor) from data that were received but erroneous for another reason (incorrect coordinates). As underlined in the Special Topic, the difference between these two types of error is substantial.
- It is often difficult to determine unequivocally that a record is wrong, because this decision is related to assumptions about the species' biology. If all original data are kept, criteria to identify outliers can be changed at any time.
- What looks useless in most cases (e.g. records without coordinates) might be very useful in other studies that were not planned when data were acquired and screened.
- Repeated erroneous data are a fairly reliable clue that a sensor is not working properly, and you might use this information to decide whether and when to replace it.

In the following examples, you will explore the location data set hunting for possible errors. First, you will create a field in the GPS data table where you can store a tag associated with each erroneous or suspicious record. Then, you will define a list of codes, one for each possible type of error. In general, a preliminary visual exploration of the spatial distribution of the entire set of locations can be useful for detecting the general spatial patterns of the animals' movements and evident outlier locations.

To tag locations as errors or unreliable data, you first create a new field (*sensor_validity_code*) in the *gps_data_animals* table. At the same time, a list of codes corresponding to all possible errors must be created using a lookup table *gps_validity*, linked to the *sensor_validity_code* field with a foreign key. When an outlier detection process identifies an error, the record is marked with the corresponding tag code. In the analytical stage, users can decide to exclude all or part of the records tagged as erroneous. The evident errors (e.g. points outside the study area) can be automatically marked in the import procedure, while some other detection algorithms are better run by users when required because they imply a long processing time or might need a fine tuning of the parameters. First, add the new field to the table:

```
ALTER TABLE main.gps_data_animals
  ADD COLUMN gps_validity_code integer;
```

Now create a table to store the validity codes, create the external key and insert the admitted values:

```
CREATE TABLE lu_tables.lu_gps_validity(
  gps_validity_code integer,
  gps_validity_description character varying,
  CONSTRAINT lu_gps_validity_pkey
    PRIMARY KEY (gps_validity_code));
COMMENT ON TABLE lu_tables.lu_gps_validity
IS 'Look up table for GPS positions validity codes.';

ALTER TABLE main.gps_data_animals
  ADD CONSTRAINT animals_lu_gps_validity
  FOREIGN KEY (gps_validity_code)
  REFERENCES lu_tables.lu_gps_validity (gps_validity_code)
  MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION;

INSERT INTO lu_tables.lu_gps_validity
  VALUES (0, 'Position with no coordinate');
INSERT INTO lu_tables.lu_gps_validity
  VALUES (1, 'Valid Position');
INSERT INTO lu_tables.lu_gps_validity
  VALUES (2, 'Position with a low degree of reliability');
INSERT INTO lu_tables.lu_gps_validity
  VALUES (11, 'Position wrong: out of the study area');
INSERT INTO lu_tables.lu_gps_validity
  VALUES (12, 'Position wrong: impossible spike');
INSERT INTO lu_tables.lu_gps_validity
  VALUES (13, 'Position wrong: impossible place (e.g. lake or sea)');
INSERT INTO lu_tables.lu_gps_validity
  VALUES (21, 'Position wrong: duplicated timestamp');
```

Some errors are already contained in the five GPS data sets previously loaded into the database, but a new (fake) data set can be imported to verify a wider range of errors. To do this, insert a new animal, a new GPS sensor, a new deployment record and finally import the data from the .csv file provided in the test data set.

Insert a new animal, called '*test*':

```
INSERT INTO main.animals
  (animals_id, animals_code, name, sex, age_class_code, species_code, note)
  VALUES (6, 'test', 'test-ina', 'm', 3, 1, 'This is a fake animal, used to
  test outliers detection processes.');
```

Insert a new sensor, called '*GSM_test*':

```
INSERT INTO main.gps_sensors
  (gps_sensors_id, gps_sensors_code, purchase_date, frequency, vendor, model,
  sim)
  VALUES (6, 'GSM_test', '2005-01-01', 1000, 'TNT', 'top', '+391441414');
```

Insert the time interval of the deployment of the test sensor on the test animal:

```
INSERT INTO main.gps_sensors_animals
  (animals_id, gps_sensors_id, start_time, end_time, notes)
  VALUES (6, 6, '2005-04-04 08:00:00+02', '2005-05-06 02:00:00+02', 'test
  deployment');
```

The last step is importing the data set from the .csv file:

```
COPY main.gps_data(
  gps_sensors_code, line_no, utc_date, utc_time, lmt_date, lmt_time, ecef_x,
  ecef_y, ecef_z, latitude, longitude, height, dop, nav, validated, sats_used,
  ch01_sat_id, ch01_sat_cnr, ch02_sat_id, ch02_sat_cnr, ch03_sat_id,
  ch03_sat_cnr, ch04_sat_id, ch04_sat_cnr, ch05_sat_id, ch05_sat_cnr,
  ch06_sat_id, ch06_sat_cnr, ch07_sat_id, ch07_sat_cnr, ch08_sat_id,
  ch08_sat_cnr, ch09_sat_id, ch09_sat_cnr, ch10_sat_id, ch10_sat_cnr,
  ch11_sat_id, ch11_sat_cnr, ch12_sat_id, ch12_sat_cnr, main_vol, bu_vol, temp,
  easting, northing, remarks)
FROM
  'C:\tracking_db\data\sensors_data\GSM_test.csv'
    WITH (FORMAT csv, HEADER, DELIMITER ';');
```

Now you can proceed with outlier detection, having a large set of errors to hunt for. You can start by assuming that all the GPS positions are correct (value '1'):

```
UPDATE main.gps_data_animals
  SET gps_validity_code = 1;
```

Missing Records

You might have a missing record when the device was programmed to acquire the position but no information (not even the acquisition time) is recorded. In this case, you can use specific functions (see Chap. 9) to create 'virtual' records and, if needed, compute and interpolate values for the coordinates. The 'virtual' records should be created just in the analytical stage and not stored in the reference data set (table *gps_data_animals*).

Records with Missing Coordinates

When the GPS is unable to receive sufficient satellite signal, the record has no coordinates associated. The rate of GPS failure can vary substantially, mainly according to sensor quality, terrain morphology and vegetation cover. Missing coordinates cannot be managed as location bias, but have to be properly treated in the analytical stage depending on the specific objective, since they result in an

erroneous 'fix rate' (see Special Topic—how to deal with erroneous fix rates is beyond the scope of this chapter). Technically, they can be filtered from the data set, or an estimated value can be calculated by interpolating the previous and next GPS positions. This is a very important issue, since several analytical methods require regular time intervals. Note that with no longitude/latitude, the spatial attribute (i.e. the *geom* field) cannot be created, which makes it easy to identify this type of error. You can mark all the GPS positions with no coordinates with the code '0':

```
UPDATE main.gps_data_animals
  SET gps_validity_code = 0
  WHERE geom IS NULL;
```

Multiple Records with the Same Acquisition Time

In some (rare) cases, you might have a repeated acquisition time (from the same acquisition source). You can detect these errors by grouping your data set by animal and acquisition time and asking for multiple occurrences. Here is an example of an SQL query to get this result:

```
SELECT
  x.gps_data_animals_id, x.animals_id, x.acquisition_time
FROM
  main.gps_data_animals x,
  (SELECT animals_id, acquisition_time
  FROM main.gps_data_animals
  WHERE gps_validity_code = 1
  GROUP BY animals_id, acquisition_time
  HAVING count(animals_id) > 1) a
WHERE
  a.animals_id = x.animals_id AND
  a.acquisition_time = x.acquisition_time
ORDER BY
  x.animals_id, x.acquisition_time;
```

This query returns the id of the records with duplicated timestamps (having *count(animals_id)* > 1). In this case, it retrieves two records with the same acquisition time ('2005-05-04 22:01:24+00'):

```
gps_data_animals_id | animals_id |    acquisition_time
--------------------+------------+------------------------
              28177 |          6 | 2005-05-05 00:01:24+02
              28176 |          6 | 2005-05-05 00:01:24+02
```

At this point, the data manager has to decide what to do. You can keep one of the two (or more) GPS positions with repeated acquisition time, or tag both (all) as

unreliable. The first possibility would imply a detailed inspection of the locations at fault, in order to possibly identify (with no guarantee of success) which one is correct. On the other hand, the second case is more conservative and can be automated as the user does not have to take any decision that could lead to erroneous conclusions. As for the other type of errors, a specific *gps_validity_code* is suggested. Here is an example:

```
UPDATE main.gps_data_animals
  SET gps_validity_code = 21
  WHERE
    gps_data_animals_id in
      (SELECT x.gps_data_animals_id
       FROM
        main.gps_data_animals x,
        (SELECT animals_id, acquisition_time
         FROM main.gps_data_animals
         WHERE gps_validity_code = 1
         GROUP BY animals_id, acquisition_time
         HAVING count(animals_id) > 1) a
       WHERE
        a.animals_id = x.animals_id AND
        a.acquisition_time = x.acquisition_time);
```

If you rerun the above query to identify locations with the same timestamps, it will now return an empty output.

Records with Different Values When Acquired Using Different Acquisition Sources

It may happen that data are obtained from sensors through different data transfer processes. A typical example is data received in near real time through a GSM network and later downloaded directly via cable from the sensor when it is physically removed from the animal. If the information is different, it probably means that an error occurred during data transmission. In this case, it is necessary to define a hierarchy of reliability for the different sources (e.g. data obtained via cable download are better than those obtained via the GSM network). This information should be stored when data are imported into the database into *gps_data* table. Then, when valid data are to be identified, the 'best' code should be selected, paying attention to properly synchronise *gps_data* and *gps_data _animals.* Which specific tools will be used to manage different acquisition sources largely depends on the number of sensors, frequency of updates and desired level of automation of the process. No specific examples are provided here.

Records Erroneously Attributed to Animals

This situation usually occurs for the first and/or last GPS positions because the start and end date and time of the sensor deployment are not correct. The consequence is that the movements of the sensor before and after the deployment are attributed to the animal. It may be difficult to trap this error with automatic methods because incorrect records can be organised in spatial clusters with a (theoretically) meaningful pattern (the set of erroneous GPS positions has a high degree of spatial autocorrelation as it contains 'real' GPS positions of 'real' movements, although they are not animal's movements). It is important to stress that this kind of pattern, e.g. GPS positions repeated in a small area where the sensor is stored before the deployment (e.g. the researcher's office) and then a long movement to where the sensor is deployed on the animal, can closely resemble the sequence of GPS positions for animals just released in a new area.

To identify this type of error, the suggested approach is to visually explore the data set in a GIS desktop interface. Once you detect this situation, you should check the source of information on the date of capture and sensor deployment and, if needed, correct the information in the table *gps_sensors_animals* (this will automatically update the table *gps_data_animals*). In general, a visual exploration of your GPS data sets, using as representation both points and trajectories, is always useful to help identify unusual spatial patterns. For this kind of error, no *gps_validity_code* are used because, once detected, they are automatically excluded from the table *gps_data_animals*.

The best method to avoid this type of error is to get accurate and complete information about the deployment of the sensors on the animals, for example, verifying not just the starting and ending date, but also the time of the day and time zone.

Special attention must be paid to the end of the deployment. For active deployments, no end is defined. In this case, the procedure can make use of the function *now()* to define a dynamic upper limit when checking the timestamp of recorded locations (i.e. the record is not valid if $acquisition\_time > now()$).

The next types of error can all be connected to GPS sensor malfunctioning or poor performance, leading to biased locations with low accuracy, or a true 'outlier', i.e. coordinates that are distant or very distant from the 'true location'.

Records Located Outside the Study Area

When the error of coordinates is due to reasons not related to general GPS accuracy (which will almost always be within a few dozen metres), the incorrect positions are often quite evident as they are usually very far from the others (a typical example is the inversion of longitude and latitude). At the same time, this error is random, so erroneous GPS positions are hardly grouped in spatial clusters.

When a study area has defined limits (e.g. fencing or natural boundaries), the simplest way to identify incorrect GPS positions is to run a query that looks for those that are located outside these boundaries (optionally, with an extra buffer area). Though animals have no constraints to their movements, they are still limited to a specific area (e.g. an island), so you can delineate a very large boundary so that at least GPS positions very far outside this area are captured. In this case, it is better to be conservative and enlarge the study area as much as possible to exclude all the valid GPS positions. Other, more fine-tuned methods can be used at a later stage to detect the remaining erroneous GPS positions. This approach has the risk of tagging correct locations if the boundaries are not properly set, as the criteria are very subjective. It is important to note that erroneous locations will be identified in any case as impossible movements (see next sections). This step can be useful in cases where you don't have access to movement properties (e.g. VHF data with only one location a week). Another element to keep in mind, especially in the case of automatic procedures to be run in real time on the data flow, is that very complex geometries (e.g. a coastline drawn at high spatial resolution) can slow down the intersection queries. In this case, you can exploit the power of spatial indexes and/or simplify your geometry, which can be done using the PostGIS commands *ST_Simplify*[1] and *ST_SimplifyPreserveTopology*[2]. Here is an example of an SQL query that detects outliers outside the boundaries of the *study_area* layer, returning the IDs of outlying records:

```
SELECT
  gps_data_animals_id
FROM
  main.gps_data_animals
LEFT JOIN
  env_data.study_area
ON
  ST_Intersects(gps_data_animals.geom, study_area.geom)
WHERE
  study_area IS NULL AND
  gps_data_animals.geom IS NOT NULL;
```

There result is the list of the six GPS positions that fall outside the study area:

```
 gps_data_animals_id
---------------------
               15810
               27945
               28094
               28111
               20540
               23030
```

[1] http://www.postgis.org/docs/ST_Simplify.html.

[2] http://www.postgis.org/docs/ST_SimplifyPreserveTopology.html.

The same query could be made using *ST_Disjoint*, i.e. the opposite of *ST_Intersects* (note, however, that the former does not work on multiple polygons). Here is an example where a small buffer (*ST_Buffer*) is added (using *Common Table Expressions*[3]):

```
WITH area_buffer_simplified AS
  (SELECT
    ST_Simplify(
      ST_Buffer(study_area.geom, 0.1), 0.1) AS geom
   FROM
    env_data.study_area)
SELECT
  animals_id, gps_data_animals_id
FROM
  main.gps_data_animals
WHERE
  ST_Disjoint(
    gps_data_animals.geom, (SELECT geom FROM area_buffer_simplified));
```

The use of the syntax with *WITH* is optional, but in some cases can be a useful way to simplify your queries, and it might be interesting for you to know how it works.

In this case, just five outliers are detected because one of the previous six is very close to the boundaries of the study area:

```
animals_id | gps_data_animals_id
-----------+--------------------
         3 |               15810
         6 |               27945
         6 |               28111
         1 |               20540
         5 |               23030
```

This GPS position deserves a more accurate analysis to determine whether it is really an outlier. Now tag the other five GPS positions as erroneous (validity code '11', i.e. 'Position wrong: out of the study area'):

```
UPDATE main.gps_data_animals
  SET gps_validity_code = 11
  WHERE
    gps_data_animals_id in
    (SELECT gps_data_animals_id
     FROM main.gps_data_animals, env_data.study_area
     WHERE ST_Disjoint(
       gps_data_animals.geom,
       ST_Simplify(ST_Buffer(study_area.geom, 0.1), 0.1)));
```

[3] http://www.postgresql.org/docs/9.2/static/queries-with.html.

Using a simpler approach, another quick way to detect these errors is to order GPS positions according to their longitude and latitude coordinates. The outliers are immediately visible as their values are completely different from the others and they pop up at the beginning of the list. An example of this kind of query is:

```
SELECT
  gps_data_animals_id, ST_X(geom)
FROM
  main.gps_data_animals
WHERE
  geom IS NOT NULL
ORDER BY
  ST_X(geom)
LIMIT 10;
```

The resulting data set is limited to ten records, as just a few GPS positions are expected to be affected by this type of error. From the result of the query, it is clear that the first two locations are outliers, while the third is a strong candidate:

```
 gps_data_animals_id |    st_x
---------------------+------------
               15810 |  5.0300699
               23030 | 10.7061637
               27948 | 10.9506126
               17836 | 10.9872122
               17835 | 10.9875451
               17837 | 10.9876942
               17609 | 10.9884574
               18098 | 10.9898182
               18020 | 10.9899461
               20154 | 10.9900441
```

The same query can then be repeated in reverse order, and then doing the same for latitude:

```
SELECT gps_data_animals_id, ST_X(geom)
FROM main.gps_data_animals
WHERE geom IS NOT NULL
ORDER BY ST_X(geom)
DESC LIMIT 10;

SELECT gps_data_animals_id, ST_Y(geom)
FROM main.gps_data_animals
WHERE geom IS NOT NULL
ORDER BY ST_Y(geom)
LIMIT 10;

SELECT gps_data_animals_id, ST_Y(geom)
FROM main.gps_data_animals
WHERE geom IS NOT NULL
ORDER BY ST_Y(geom) DESC
LIMIT 10;
```

Records Located in Impossible Places

When there are areas not accessible to animals because of physical constraints (e.g. fencing, natural barriers) or environments not compatible with the studied species (lakes and sea, or land, according to the species), you can detect GPS positions that are located in those areas where it is impossible for the animal to be. Therefore, the decision whether or not to mark the locations as incorrect is based on ecological assumptions (i.e. non-habitat). In this example, you mark, using validity code '13', all the GPS positions that fall inside a water body according to Corine land cover layer (Corine codes '40', '41', '42', '43' and '44'):

```
UPDATE main.gps_data_animals
  SET gps_validity_code = 13
  FROM
    env_data.corine_land_cover
  WHERE
    ST_Intersects(
      corine_land_cover.rast,
      ST_Transform(gps_data_animals.geom, 3035)) AND
    ST_Value(
      corine_land_cover.rast,
      ST_Transform(gps_data_animals.geom, 3035))
      in (40,41,42,43,44) AND
    gps_validity_code = 1 AND
    ST_Value(
      corine_land_cover.rast,
      ST_Transform(gps_data_animals.geom, 3035)) != 'NaN';
```

For this kind of control, you must also consider that the result depends on the accuracy of the land cover layer and of the GPS positions. Thus, at a minimum, a further visual check in a GIS environment is recommended.

Records that Would Imply Impossible Movements

To detect records with incorrect coordinates that cannot be identified using clear boundaries, such as the study area or land cover type, a more sophisticated outlier filtering procedure must be applied. To do so, some kind of assumption about the animals' movement model has to be made, for example, a speed limit. It is important to remember that animal movements can be modelled in different ways at different temporal scales: an average speed that is impossible over a period of 4 h could be perfectly feasible for movements in a shorter time (e.g. 5 minutes). Which algorithm to apply depends largely on the species and the environment in which the animal is moving and the duty cycle of the tag. In general, PostgreSQL window functions can help.

Special Topic: **PostgreSQL window functions**

A window function[4] performs a calculation across a set of rows that are somehow related to the current row. This is similar to an aggregate function, but unlike regular aggregate functions, window functions do not group rows into a single output row, hence they are still able to access more than just the current row of the query result. In particular, it enables you to access previous and next rows (according to a user-defined ordering criteria) while calculating values for the current row. This is very useful, as a tracking data set has a predetermined temporal order, where many properties (e.g. geometric parameters of the trajectory, such as turning angle and speed) involve a sequence of GPS positions. It is important to remember that the order of records in a database is irrelevant. The ordering criteria must be set in the query that retrieves data.

In the next example, you will make use of window functions to convert the series of locations into steps (i.e. the straight-line segment connecting two successive locations), and compute geometric characteristics of each step: the time interval, the step length, and the speed during the step as the ratio of the previous two. It is important to note that while a step is the movement between two points, in many cases, its attributes are associated with the starting or the ending point. In this book, we use the ending point as reference. In some software, particularly the adehabitat[5] package for *R* (see Chap. 10), the step is associated with the starting point. If needed, the queries and functions presented in this book can be modified to follow this convention.

```
SELECT
  animals_id AS id,
  acquisition_time,
  LEAD(acquisition_time,-1)
    OVER (
      PARTITION BY animals_id
      ORDER BY acquisition_time) AS acquisition_time_1,
  (EXTRACT(epoch FROM acquisition_time) -
  LEAD(EXTRACT(epoch FROM acquisition_time), -1)
    OVER (
      PARTITION BY animals_id
      ORDER BY acquisition_time))::integer AS deltat,
  (ST_Distance_Spheroid(
    geom,
    LEAD(geom, -1)
      OVER (
        PARTITION BY animals_id
        ORDER BY acquisition_time),
```

[4] http://www.postgresql.org/docs/9.2/static/tutorial-window.html.

[5] http://cran.r-project.org/web/packages/adehabitat/index.html.

```
    'SPHEROID["WGS 84",6378137,298.257223563]'))::integer AS dist,
  (ST_Distance_Spheroid(
    geom,
    LEAD(geom, -1)
      OVER (
        PARTITION BY animals_id
        ORDER BY acquisition_time),
    'SPHEROID["WGS 84",6378137,298.257223563]')/
  ((EXTRACT(epoch FROM acquisition_time) -
  LEAD(
    EXTRACT(epoch FROM acquisition_time), -1)
    OVER (
      PARTITION BY animals_id
      ORDER BY acquisition_time))+1)*60*60)::numeric(8,2) AS speed
FROM main.gps_data_animals
WHERE gps_validity_code = 1
LIMIT 10;
```

The result of this query is

id	*acquisition_time*	*acquisition_time_1*	*deltat*	*dist*	*speed*
1	*2005-10-18 22:00:54+02*				
1	*2005-10-19 02:01:23+02*	*2005-10-18 22:00:54+02*	*14429*	*97*	*24.15*
1	*2005-10-19 06:02:22+02*	*2005-10-19 02:01:23+02*	*14459*	*430*	*107.08*
1	*2005-10-19 10:03:08+02*	*2005-10-19 06:02:22+02*	*14446*	*218*	*54.40*
1	*2005-10-20 22:00:53+02*	*2005-10-19 10:03:08+02*	*129465*	*510*	*14.17*
1	*2005-10-21 02:00:48+02*	*2005-10-20 22:00:53+02*	*14395*	*97*	*24.22*
1	*2005-10-21 06:00:53+02*	*2005-10-21 02:00:48+02*	*14405*	*69*	*17.26*
1	*2005-10-21 10:01:42+02*	*2005-10-21 06:00:53+02*	*14449*	*478*	*119.20*
1	*2005-10-21 18:01:16+02*	*2005-10-21 10:01:42+02*	*28774*	*150*	*18.77*
1	*2005-10-21 22:01:23+02*	*2005-10-21 18:01:16+02*	*14407*	*688*	*172.02*

As a demonstration of a possible approach to detecting ‘impossible movements’, here is an adapted function that implements the algorithm presented in Bjorneraas et al. (2010). In the first step, you compute the distance from each GPS position to the average of the previous and next ten GPS positions, and extract records that have values bigger than a defined threshold (in this case, arbitrarily set to 10,000 m):

```
SELECT gps_data_animals_id
FROM
  (SELECT
    gps_data_animals_id,
    ST_Distance_Spheroid(geom,
      ST_setsrid(ST_makepoint(
        avg(ST_X(geom))
          OVER (
```

```
            PARTITION BY animals_id
            ORDER BY acquisition_time rows
              BETWEEN 10 PRECEDING and 10 FOLLOWING),
        avg(ST_Y(geom))
          OVER (
            PARTITION BY animals_id
            ORDER BY acquisition_time rows
              BETWEEN 10 PRECEDING and 10 FOLLOWING)), 4326),'SPHEROID["WGS
              84", 6378137,298.257223563]') AS dist_to_avg
   FROM
     main.gps_data_animals
   WHERE
     gps_validity_code = 1) a
WHERE
  dist_to_avg > 10000;
```

The result is the list of IDs of all the GPS positions that match the defined conditions (and thus can be considered outliers). In this case, just one record is returned:

```
gps_data_animals_id
--------------------
              27948
```

This code can be improved in many ways. For example, it is possible to consider the median instead of the average. It is also possible to take into consideration that the first and last ten GPS positions have incomplete windows of 10 + 10 GPS positions. Moreover, this method works fine for GPS positions at regular time intervals, but in the case of a change in acquisition schedule might lead to unexpected results. In these cases, you should create a query with a temporal window instead of a fixed number of GPS positions.

In the second step, the angle and speed based on the previous and next GPS position are calculated (both the previous and next location are used to determine whether the location under consideration shows a spike in speed or turning angle), and then GPS positions below the defined thresholds (in this case, arbitrarily set as cosine of the relative angle <-0.99 and speed >2,500 m per hour) are extracted:

```
SELECT
  gps_data_animals_id
FROM
  (SELECT gps_data_animals_id,
  ST_Distance_Spheroid(
    geom,
    LAG(geom, 1) OVER (PARTITION BY animals_id ORDER BY acquisition_time),
    'SPHEROID["WGS 84",6378137,298.257223563]') /
    (EXTRACT(epoch FROM acquisition_time) - EXTRACT (epoch FROM
    (lag(acquisition_time, 1) OVER (PARTITION BY animals_id ORDER BY
```

```
    acquisition_time))))*3600 AS speed_FROM,
  ST_Distance_Spheroid(
    geom,LEAD(geom, 1) OVER (PARTITION BY animals_id ORDER BY acquisition_time),
    'SPHEROID["WGS 84",6378137,298.257223563]') /
    ( - EXTRACT(epoch FROM acquisition_time) + EXTRACT (epoch FROM
    (lead(acquisition_time, 1) OVER (PARTITION BY animals_id ORDER BY
    acquisition_time))))*3600 AS speed_to,
  cos(ST_Azimuth((
    LAG(geom, 1) OVER (PARTITION BY animals_id ORDER BY
    acquisition_time))::geography,
    geom::geography) -
  ST_Azimuth(
    geom::geography,
    (LEAD(geom, 1) OVER (PARTITION BY animals_id ORDER BY
    acquisition_time))::geography)) AS rel_angle
  FROM main.gps_data_animals
  WHERE gps_validity_code = 1) a
WHERE
  rel_angle < -0.99 AND
  speed_from > 2500 AND
  speed_to > 2500;
```

The result returns the list of IDs of all the GPS positions that match the defined conditions. The same record detected in the previous query is returned. These examples can be used as templates to create other filtering procedures based on the temporal sequence of GPS positions and the users' defined movement constraints.

It is important to remember that this kind of method is based on the analysis of the sequence of GPS positions, and therefore results might change when new GPS positions are uploaded. Moreover, it is not possible to run them in real time because the calculation requires a subsequent GPS position. The result is that they have to be run in a specific procedure unlinked with the (near) real-time import procedure.

Now you run this process on your data sets to mark the detected outliers (validity code '12'):

```
UPDATE
  main.gps_data_animals
SET
  gps_validity_code = 12
WHERE
  gps_data_animals_id in
    (SELECT gps_data_animals_id
    FROM
      (SELECT
```

```
      gps_data_animals_id,
      ST_Distance_Spheroid(geom, lag(geom, 1) OVER (PARTITION BY animals_id
      ORDER BY acquisition_time), 'SPHEROID["WGS 84",6378137,298.257223563]') /
      (EXTRACT(epoch FROM acquisition_time) - EXTRACT (epoch FROM
      (lag(acquisition_time, 1) OVER (PARTITION BY animals_id ORDER BY
      acquisition_time))))*3600 AS speed_from,
      ST_Distance_Spheroid(geom, lead(geom, 1) OVER (PARTITION BY animals_id
      order by acquisition_time), 'SPHEROID["WGS 84",6378137,298.257223563]') /
      ( - EXTRACT(epoch FROM acquisition_time) + EXTRACT (epoch FROM
      (lead(acquisition_time, 1) OVER (PARTITION BY animals_id ORDER BY
      acquisition_time))))*3600 AS speed_to,
      cos(ST_Azimuth((lag(geom, 1) OVER (PARTITION BY animals_id ORDER BY
      acquisition_time))::geography, geom::geography) - ST_Azimuth(geom::geography,
      (lead(geom, 1) OVER (PARTITION BY animals_id ORDER BY
      acquisition_time))::geography)) AS rel_angle
    FROM main.gps_data_animals
    WHERE gps_validity_code = 1) a
  WHERE
    rel_angle < -0.99 AND
    speed_from > 2500 AND
    speed_to > 2500);
```

Records that Would Imply Improbable Movements

This is similar to the previous type of error, but in this case, the assumption made in the animals' movement model cannot completely exclude that the GPS position is correct (e.g. same methods as before, but with reduced thresholds: cosine of the relative angle <-0.98 and speed >300 m per hour). These records should be kept as valid but marked with a specific validity code that can permit users to exclude them for analysis as appropriate.

```
UPDATE
  main.gps_data_animals
SET
  gps_validity_code = 2
WHERE
  gps_data_animals_id IN
    (SELECT gps_data_animals_id
    FROM
      (SELECT
        gps_data_animals_id,
        ST_Distance_Spheroid(geom, lag(geom, 1) OVER (PARTITION BY animals_id
        ORDER BY acquisition_time), 'SPHEROID["WGS 84",6378137,298.257223563]') /
        (EXTRACT(epoch FROM acquisition_time) - EXTRACT (epoch FROM
        (lag(acquisition_time, 1) OVER (PARTITION BY animals_id ORDER BY
        acquisition_time))))*3600 AS speed_FROM,
        ST_Distance_Spheroid(geom, lead(geom, 1) OVER (PARTITION BY animals_id
      ORDER BY acquisition_time), 'SPHEROID["WGS 84",6378137,298.257223563]') /
      ( - EXTRACT(epoch FROM acquisition_time) + EXTRACT (epoch FROM
      (lead(acquisition_time, 1) OVER (PARTITION BY animals_id ORDER BY
```

```
    acquisition_time))))*3600 AS speed_to,
    cos(ST_Azimuth((lag(geom, 1) OVER (PARTITION BY animals_id ORDER BY
    acquisition_time))::geography, geom::geography) - ST_Azimuth(geom
    ::geography,(lead(geom, 1) OVER (PARTITION BY animals_id ORDER BY
    acquisition_time))::geography)) AS rel_angle
   FROM main.gps_data_animals
   WHERE gps_validity_code = 1) a
WHERE
  rel_angle < -0.98 AND
  speed_from > 300 AND
  speed_to > 300);
```

The marked GPS positions should then be inspected visually to decide if they are valid with a direct expert evaluation.

Update of Spatial Views to Exclude Erroneous Locations

As a consequence of the outlier tagging approach illustrated in these pages, views based on the GPS positions data set should exclude the incorrect points, adding a *gps_validity_code = 1* criteria (corresponding to GPS positions with no errors and valid geometry) in their *WHERE* conditions. You can do this for *analysis.view_convex_hulls*:

```
CREATE OR REPLACE VIEW analysis.view_convex_hulls AS
SELECT
  gps_data_animals.animals_id,
  ST_ConvexHull(ST_Collect(gps_data_animals.geom))::geometry(Polygon,4326)
  AS geom
FROM
  main.gps_data_animals
WHERE
  gps_data_animals.gps_validity_code = 1
GROUP BY
  gps_data_animals.animals_id
ORDER BY
  gps_data_animals.animals_id;
```

You do the same for *analysis.view_gps_locations*:

```
CREATE OR REPLACE VIEW analysis.view_gps_locations AS
SELECT
  gps_data_animals.gps_data_animals_id,
  gps_data_animals.animals_id,
  animals.name,
```

```
    timezone('UTC'::text, gps_data_animals.acquisition_time) AS time_utc,
    animals.sex,
    lu_age_class.age_class_description,
    lu_species.species_description,
    gps_data_animals.geom
  FROM
    main.gps_data_animals,
    main.animals,
    lu_tables.lu_age_class,
    lu_tables.lu_species
  WHERE
    gps_data_animals.animals_id = animals.animals_id AND
    animals.age_class_code = lu_age_class.age_class_code AND
    animals.species_code = lu_species.species_code AND
    gps_data_animals.gps_validity_code = 1;
```

Now repeat the same operation for *analysis.view_trajectories*:

```
CREATE OR REPLACE VIEW analysis.view_trajectories AS
SELECT
  sel_subquery.animals_id,
  st_MakeLine(sel_subquery.geom)::geometry(LineString,4326) AS geom
FROM
  (SELECT
    gps_data_animals.animals_id,
    gps_data_animals.geom,
    gps_data_animals.acquisition_time
  FROM main.gps_data_animals
  WHERE gps_data_animals.gps_validity_code = 1
  ORDER BY gps_data_animals.animals_id, gps_data_animals.acquisition_time)
  sel_subquery
GROUP BY sel_subquery.animals_id;
```

If you visualise these layers in a GIS desktop, you can verify that outliers are now excluded. An example for the MCP is illustrated in Fig. 8.2, which can be compared with Fig. 5.4.

Update Import Procedure with Detection of Erroneous Positions

Some of the operations to filter outliers can be integrated into the procedure that automatically uploads GPS positions into the table *gps_data_animals*. In this example, you redefine the *tools.new_gps_data_animals()* function to tag GPS positions with no coordinates (*gps_validity_code = 0*) and GPS positions outside of the study area (*gps_validity_code = 11*) as soon as they are imported into the database. All the others are set as valid (*gps_validity_code = 1*).

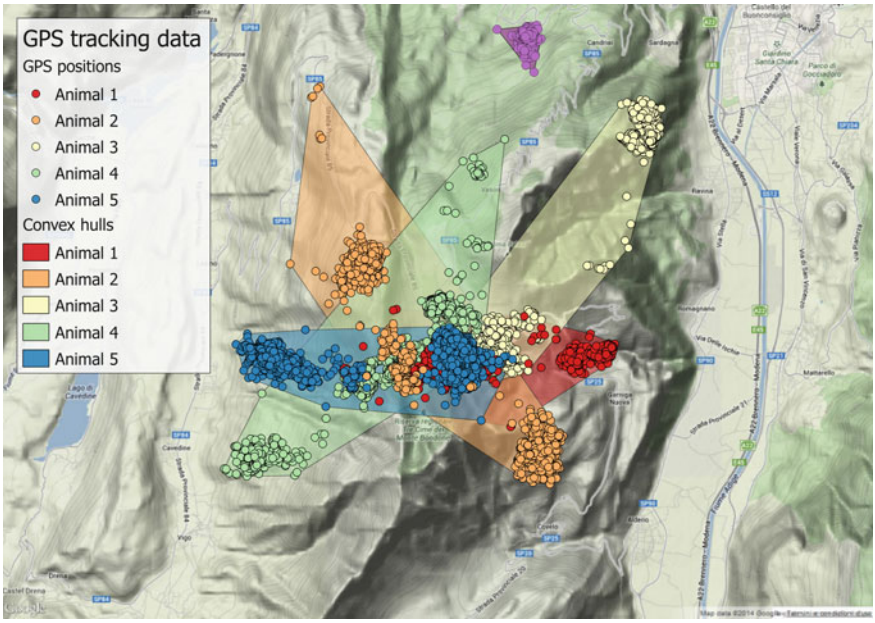

Fig. 8.2 Minimum convex polygons without outliers

```
CREATE OR REPLACE FUNCTION tools.new_gps_data_animals()
RETURNS trigger AS
$BODY$
DECLARE
thegeom geometry;
BEGIN
IF NEW.longitude IS NOT NULL AND NEW.latitude IS NOT NULL THEN
  thegeom = ST_setsrid(ST_MakePoint(NEW.longitude, NEW.latitude), 4326);
  NEW.geom =thegeom;
  NEW.gps_validity_code = 1;
    IF NOT EXISTS (SELECT study_area FROM env_data.study_area WHERE
ST_intersects(ST_Buffer(thegeom,0.1), study_area.geom)) THEN
      NEW.gps_validity_code = 11;
    END IF;
  NEW.pro_com = (SELECT pro_com::integer FROM env_data.adm_boundaries WHERE
ST_intersects(geom,thegeom));
  NEW.corine_land_cover_code = (SELECT ST_Value(rast, ST_Transform(thegeom,
  3035)) FROM env_data.corine_land_cover WHERE
ST_Intersects(ST_Transform(thegeom,3035), rast));
  NEW.altitude_srtm = (SELECT ST_Value(rast,thegeom) FROM env_data.srtm_dem
```

```
WHERE ST_intersects(thegeom, rast));
  NEW.station_id = (SELECT station_id::integer FROM env_data.meteo_stations
ORDER BY ST_Distance_Spheroid(thegeom, geom, 'SPHEROID["WGS
  84",6378137,298.257223563]') LIMIT 1);
  NEW.roads_dist = (SELECT ST_Distance(thegeom::geography,
  geom::geography)::integer FROM env_data.roads ORDER BY
ST_Distance(thegeom::geography, geom::geography) LIMIT 1);
  NEW.ndvi_modis = (SELECT ST_Value(rast, thegeom)FROM env_data_ts.ndvi_modis
WHERE ST_Intersects(thegeom, rast)
AND EXTRACT(year FROM acquisition_date) = EXTRACT(year FROM
  NEW.acquisition_time)
AND EXTRACT(month FROM acquisition_date) = EXTRACT(month FROM
  NEW.acquisition_time)
AND EXTRACT(day FROM acquisition_date) = CASE
WHEN EXTRACT(day FROM NEW.acquisition_time) < 11 THEN 1
WHEN EXTRACT(day FROM NEW.acquisition_time) < 21 THEN 11
ELSE 21
END);
ELSE
NEW.gps_validity_code = 0;
END IF;
RETURN NEW;
END;$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
COMMENT ON FUNCTION tools.new_gps_data_animals()
IS 'When called by the trigger insert_gps_locations (raised whenever a new GPS
position is uploaded into gps_data_animals) this function gets the new
longitude and latitude values and sets the field geom accordingly, computing a
set of derived environmental information calculated intersection the GPS
position with the environmental ancillary layers. GPS positions outside the
study area are tagged as outliers.';
```

You can test the results by reloading the GPS positions into *gps_data_animals* (for example, modifying the *gps_sensors_animals* table). If you do so, do not forget to rerun the tool to detect GPS positions in water, impossible spikes, and duplicated acquisition time, as they are not integrated in the automated upload procedure.

References

Bjorneraas K, van Moorter B, Rolandsen CM, Herfindal I (2010) Screening GPS location data for errors using animal movement characteristics. J Wild Manage 74:1361–1366. doi:10.1111/j.1937-2817.2010.tb01258.x

Frair JL, Fieberg J, Hebblewhite M, Cagnacci F, DeCesare NJ, Pedrotti L (2010) Resolving issues of imprecise and habitat-biased locations in ecological analyses using GPS telemetry data. Philos Trans R Soc B 365:2187–2200. doi:10.1098/rstb.2010.0084