ExploreDuplicates

January 3, 2025

This file is an exploration of duplicate values that are seen in the Iceberg Tracking Beacon Database.

Derek Mueller

Several beacons have data with repeat positions but with consecutive dates, which seem suspicious. For example, the repeats seem to appear in patterns. In my experience, there is almost always some jitter in the GPS data and a lot of jitter in ARGOS data. If they are erroneous, is there a way to reprocess these? See for example:

- 2017_300234062328750 SVP-I-BXGSA-L-AD
- 2016_300234063515450 iCALIB
- 2009_300034012571050 ICEB-I-XA

Here is a bit of the 300234062328750 SVP data. Note that the distance and direction rounding was turned off to generate this.

datetime_data	latitude	longitude	temperature_air	distance	speed	direction
2017-07-25 18:00:00	76.3194	-75.0602	5.4	1607.1913	0.446	92.36
2017-07-25 19:00:00	76.3194	-75.0602	7.0	0	0.0	180
2017-07-25 20:00:00	76.3194	-75.0602	9.8	0	0.0	180
2017-07-25 21:00:00	76.295	-74.993	9.8	3251.8739	0.903	146.86
2017-07-25 22:00:00	76.295	-74.993	9.3	0	0.0	180
2017-07-25 23:00:00	76.295	-74.993	8.8	0	0.0	180
2017-07-26 00:00:00	76.2778	-74.9708	9.7	2007.9831	0.557	162.97
2017-07-26 01:00:00	76.2778	-74.9708	8.9	0	0.0	180
2017-07-26 02:00:00	76.2778	-74.9708	5.4	0	0	180

The following notebook will review duplicates in the ITDB.

[30]:
```
# imports

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import pyproj
```

[31]:
```
def count_decimal_places(value):
    if "." in str(value):  # Check if there is a decimal point
```

```python
        return len(
            str(value).split(".")[-1]
        )  # Count the characters after the decimal point
    return 0
```

[32]:

```python
def lon_precision_v_distance(lat):
    """
    Generate distances represented by a 1 sd change in longitude.

    Parameters
    ----------
    lat : float
        A valid latitude

    Returns
    -------
    dist :

    """
    geodesic = pyproj.Geod(ellps="WGS84")

    lons = [
        -90.1,
        -90.2,
        -80.01,
        -80.02,
        -70.001,
        -70.002,
        -60.0001,
        -60.0002,
        -50.00001,
        -50.00002,
        -40.000001,
        -40.000002,
        -30.0000001,
        -30.0000002,
    ]

    decimal_places = pd.Series(range(1, int(len(lons) / 2) + 1))
    az, baz, dist = geodesic.inv(
        [np.nan] + lons[:-1],
        [lat] * len(lons),
        lons,
        [lat] * len(lons),
    )
    dist = pd.Series(dist)
    distance_m = dist[dist < dist.quantile(0.51)].reset_index(drop=True)
```

```
    return pd.DataFrame({"decimal_places": decimal_places, "distance_m":␣
 ↪distance_m})
```

[33]:
```
### MAIN

# Disable rounding to see the distance properly.
# export the database to csv and read it in.
df = pd.read_csv("/home/dmueller/Desktop/cis_iceberg_beacon_database_0.3/
 ↪alltracks_05_3sd.csv")

print(f"The database has {len(df)} iceberg positions")

# make a duplicate indicator
df["dup"] = 0
df.loc[(df["speed"] == 0) & (df["direction"] == 180), "dup"] = 1

print(
    f"{df.dup.sum()} or {df.dup.sum()/len(df):.2%} of these positions are␣
 ↪duplicates, where there is no apparent movement of the iceberg"
)

# Apply function to the column to get decimal places
df["lat_d"] = df["latitude"].apply(count_decimal_places)
df["lon_d"] = df["longitude"].apply(count_decimal_places)
```

```
/tmp/ipykernel_2586541/903319785.py:5: DtypeWarning: Columns (2) have mixed
types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("/home/dmueller/Desktop/cis_iceberg_beacon_database_0.3/alltr
acks_05_3sd.csv")
```

```
The database has 885576 iceberg positions
149436 or 16.87% of these positions are duplicates, where there is no apparent
movement of the iceberg
```

The main question is:

Are the duplicates there because there is actually no movement or are there artifacts in the data?

The precision of the data makes a big difference, when detecting duplicates. For more info, see https://xkcd.com/2170/, but realize that the latitude has a big effect. For example:

[34]:
```
print(
    f"At 80 deg N, the longitude decimal place affects distance as follows:␣
 ↪\n\n {lon_precision_v_distance(80)}"
)
print(
    f"At 70 deg N, the longitude decimal place affects distance as follows:␣
 ↪\n\n {lon_precision_v_distance(70)}"
```

```
)
print(
    f"At 60 deg N, the longitude decimal place affects distance as follows:␣
  ↪\n\n {lon_precision_v_distance(60)}"
)
print(
    f"At 50 deg N, the longitude decimal place affects distance as follows:␣
  ↪\n\n {lon_precision_v_distance(50)}"
)
```

```
At 80 deg N, the longitude decimal place affects distance as follows:

    decimal_places   distance_m
0                1  1939.348314
1                2   193.934855
2                3    19.393486
3                4     1.939349
4                5     0.193935
5                6     0.019393
6                7     0.001939
At 70 deg N, the longitude decimal place affects distance as follows:

    decimal_places   distance_m
0                1  3818.653700
1                2   381.865412
2                3    38.186541
3                4     3.818654
4                5     0.381865
5                6     0.038187
6                7     0.003819
At 60 deg N, the longitude decimal place affects distance as follows:

    decimal_places   distance_m
0                1  5579.999626
1                2   558.000015
2                3    55.800002
3                4     5.580000
4                5     0.558000
5                6     0.055800
6                7     0.005580
At 50 deg N, the longitude decimal place affects distance as follows:

    decimal_places   distance_m
0                1  7169.574828
1                2   716.957536
2                3    71.695754
3                4     7.169575
```

```
4              5    0.716958
5              6    0.071696
6              7    0.007170
```

Given the location precision of a single frequency (L1) GPS receiver is typically +/- 3 to 10 m, it is quite possible that movement below ~15 m would not be detectable and therefore we cannot expect to separate duplicates that are legitimate from those that are caused by artifacts at SD <= 4.

The number of recorded latitudes by precision (number of decimal places)

[35]:
```python
df.groupby("lat_d").size()
```

[35]:
```
lat_d
1       4918
2      34474
3     290700
4     252455
5      83137
6     142156
7        847
8      35721
9          1
10      1250
13     30453
14      9464
dtype: int64
```

The number of recorded longitudes by precision (number of decimal places)

[36]:
```python
df.groupby("lon_d").size()
```

[36]:
```
lon_d
1       1874
2      31497
3     277337
4     271475
5      92899
6     142486
7        924
8      35351
9          1
10      1253
13     30390
14        70
15        19
dtype: int64
```

The percent of records that are duplicated by precision

[37]:
```
df.loc[df["dup"] == 1].groupby("lat_d").size() / df.groupby("lat_d").size() *␣
↪100
```

[37]:
```
lat_d
1      5.002033
2      6.430933
3     11.133471
4     39.412569
5     16.048210
6      0.417147
7           NaN
8      3.188601
9           NaN
10          NaN
13          NaN
14     0.369822
dtype: float64
```

[12]:
```
df.loc[df["dup"] == 1].groupby("lon_d").size() / df.groupby("lon_d").size() *␣
↪100
```

[12]:
```
lon_d
1      4.962647
2     12.045592
3      6.439098
4     41.238788
5     15.184232
6      0.397232
7           NaN
8      2.990014
9           NaN
10     0.079808
13          NaN
14    11.428571
15          NaN
dtype: float64
```

Next figure out which beacon tracks have the most duplicates:

[40]:
```
# find the total number of positions by track
total_counts = df.groupby("beacon_id").size().reset_index()
total_counts.columns = ["beacon_id", "total_n"]

# find the number of duplicates by track
dup_counts = df.loc[df.dup == 1].groupby("beacon_id").size().reset_index()
dup_counts.columns = ["beacon_id", "dups_n"]

# get the beacon models
```

```
mf = pd.read_csv("/home/dmueller/Desktop/cis_iceberg_beacon_database_0.3/
 ↪database/metadata.csv")
mf = mf[["beacon_id", "beacon_model"]]

# get the median precision for the duplicates in the track
dp_median = df.loc[df.dup == 1].groupby("beacon_id").agg({'lon_d':
 ↪'median','lat_d':'median'}).reset_index()
#df_median = df.groupby("beacon_id").agg({'lon_d':'median','lat_d':'median'}).
 ↪reset_index()
#dp_dup_stats = dup.groupby("beacon_id").agg({"lon_d": ["min","mean","max"],␣
 ↪"lat_d":["min","mean","max"]}).reset_index()
#dp_df_stats = df.groupby("beacon_id").agg({"lon_d": ["min","mean","max"],␣
 ↪"lat_d":["min","mean","max"]}).reset_index()


# merge dfs to make one with duplicate counts, total counts, beacon model and␣
 ↪percent
totdf = pd.merge(mf, total_counts, how="left")
dupdf = pd.merge(totdf, dup_counts, how="left")
# get the % of track that has duplicates
dupdf["dups_percent"] = dupdf["dups_n"] / dupdf["total_n"] * 100
# merge with the median precision
dupdp = pd.merge(dupdf, dp_median, how="left")
# sort
dupdp.sort_values("dups_percent", inplace=True, ascending=False)
```

[43]:
```
n=  pd.get_option('display.max_rows')
pd.set_option('display.max_rows', len(dupdp.loc[dupdp["dups_percent"] > 5]))
dupdp.loc[dupdp["dups_percent"] > 5]
#pd.set_option('display.max_rows', n)
```

[43]:
```
                     beacon_id           beacon_model  total_n  \
70     2015_300234061762030                 iCALIB    10280
62     2014_300234060544160         SVP-I-BXGS-LP     9324
106    2017_300234062325760      SVP-I-BXGSA-L-AD     1381
108    2017_300234062328750      SVP-I-BXGSA-L-AD     9499
68     2015_300234060104820         SVP-I-BXGS-LP    25682
63     2014_300234061763040                 iCALIB     7961
105    2017_300234062324750      SVP-I-BXGSA-L-AD     1442
115    2018_300234066545280                FT-2000     6058
69     2015_300234060435010         SVP-I-BXGS-LP     6966
15     2009_300034012571050             ICEB-I-XA    14647
109    2017_300234063516450                 iCALIB     2429
107    2017_300234062327750      SVP-I-BXGSA-L-AD     9351
85     2016_300234061768060                 iCALIB     8692
52     2012_300234010132070         SVP-I-BXGS-LP     3369
83     2016_300234061761040                 iCALIB     1301
```

84	2016_300234061763030	iCALIB	443
170	2023_300534064036660	iCALIB	8220
38	2011_300234010035940-PII-B	SVP-I-XXGS-LP	31997
44	2012_100000000000000	DMR800L	603
89	2016_300234063513450	iCALIB	11651
42	2011_300234010958690-PII-B	SVP-I-XXGS-LP	8136
90	2016_300234063515450	iCALIB	12560
86	2016_300234062950220	SVP-I-BXGS-LP	13675
88	2016_300234062957250	SVP-I-BXGS-LP	10881
21	2010_300034013723350	Model 703 Ice Tracking Buoy	456
20	2010_300034013721340	Model 703 Ice Tracking Buoy	67
51	2012_300234010082470	SVP-I-XXGS-LP	2342
154	2021_300234011751690	iCALIB	18087
157	2021_300234060725890	iCALIB	11516
155	2021_300234011751700	iCALIB	12752
153	2021_300234011750710	iCALIB	12650
152	2021_300234011750690	iCALIB	12970
22	2010_300034013726340	Model 703 Ice Tracking Buoy	447
34	2011_300034013463170	Model 703 Ice Tracking Buoy	5433
87	2016_300234062951220	SVP-I-BXGS-LP	12274
156	2021_300234011752700	iCALIB	13771
56	2013_300034013464170	Model 703 Ice Tracking Buoy	4923
55	2013_300034013464160	Model 703 Ice Tracking Buoy	7647
145	2019_300434063495310	ITB v2.0	2909
144	2019_300434063494100	ITB v2.0	5173

	dups_n	dups_percent	lon_d	lat_d
70	9150.0	89.007782	4.0	4.0
62	8139.0	87.290862	4.0	4.0
106	1202.0	87.038378	4.0	3.0
108	8255.0	86.903885	4.0	4.0
68	21836.0	85.024531	4.0	4.0
63	6577.0	82.615249	4.0	4.0
105	1182.0	81.969487	3.0	4.0
115	4938.0	81.512050	5.0	5.0
69	5369.0	77.074361	4.0	4.0
15	11195.0	76.432034	5.0	5.0
109	1840.0	75.751338	4.0	4.0
107	6799.0	72.708801	4.0	4.0
85	6265.0	72.077773	4.0	4.0
52	2285.0	67.824280	4.0	4.0
83	873.0	67.102229	4.0	4.0
84	295.0	66.591422	4.0	4.0
170	4504.0	54.793187	4.0	4.0
38	14815.0	46.301216	4.0	4.0
44	248.0	41.127695	4.0	4.0
89	4681.0	40.176809	4.0	3.0

42	2988.0	36.725664	4.0	4.0
90	4244.0	33.789809	4.0	4.0
86	4142.0	30.288848	4.0	3.0
88	3066.0	28.177557	4.0	4.0
21	126.0	27.631579	4.0	4.0
20	13.0	19.402985	4.0	4.0
51	328.0	14.005124	4.0	4.0
154	1794.0	9.918726	4.0	4.0
157	1100.0	9.551928	4.0	4.0
155	1128.0	8.845671	4.0	4.0
153	1046.0	8.268775	4.0	4.0
152	1042.0	8.033924	4.0	4.0
22	32.0	7.158837	4.0	4.0
34	384.0	7.067918	4.0	4.0
87	774.0	6.306013	4.0	4.0
156	828.0	6.012635	4.0	4.0
56	286.0	5.809466	8.0	8.0
55	433.0	5.662351	8.0	8.0
145	161.0	5.534548	6.0	6.0
144	281.0	5.432051	6.0	6.0

[ ]:

```

```