ExploreDuplicates

April 6, 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.

[1]:
```
# imports

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

[2]:
```
def count_decimal_places(value):
    if "." in str(value):  # Check if there is a decimal point
```

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

[3]:
```
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)
```

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

[ ]:
```python
### MAIN

# Disable rounding to see the distance properly.
# export the database to csv and read it in.
df = pd.read_csv("/ibtd/20250406/20250406.csv")

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

[12]:
```python
# make a duplicate indicator
df["dup"] = 0
df.loc[(df["speed"] == 0) & (df["direction"].isna()), "dup"] = 1
```

[13]:
```python
print(  f"{df.dup.sum()} or {df.dup.sum()/len(df):.8%} 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)
```

```
74590 or 8.97922824% 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:

[14]:
```python
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)

[16]:
```
df.groupby("lat_d").size()
```

[16]:
```
lat_d
1       4878
2      34013
3     278696
4     198913
5      81738
6     155024
7        847
8      35715
9          1
10      1250
13     30450
14      9170
dtype: int64
```

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

[17]:
```
df.groupby("lon_d").size()
```

[17]:
```
lon_d
1       1817
2      30847
3     268560
4     215490
5      90707
6     155337
7        923
8      35346
9          1
10      1253
13     30388
14         7
15        19
dtype: int64
```

The percent of records that are duplicated by precision

[18]:
```
df.loc[df["dup"] == 1].groupby("lat_d").size() / df.groupby("lat_d").size() *␣
↪100
```

[18]:
```
lat_d
1      4.202542
2      5.156852
3      7.282846
4     22.851196
5      6.268810
```

```
6       0.382521
7            NaN
8       3.189136
9            NaN
10           NaN
13           NaN
14      0.261723
dtype: float64
```

[19]:
```
df.loc[df["dup"] == 1].groupby("lon_d").size() / df.groupby("lon_d").size() *␣
↪100
```

[19]:
```
lon_d
1       2.036324
2      10.150096
3       3.367962
4      25.744582
5       5.816530
6       0.364369
7            NaN
8       2.990437
9            NaN
10      0.079808
13           NaN
14           NaN
15           NaN
dtype: float64
```

Next figure out which beacon tracks have the most duplicates:

[24]:
```
# 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_excel("/ibtd/metadata/track_metadata_raw.ods")
mf = mf[["beacon_id", "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()
```

```python
#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)
```

[25]:
```python
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)
```

[25]:
```
                beacon_id                        model  total_n   dups_n  \
115   2018_300234066545280                      FT-2000     6058   4938.0
109   2017_300234063516450                       iCALIB     2429   1840.0
70    2015_300234061762030                       iCALIB     3427   2298.0
62    2014_300234060544160                 SVP-I-BXGS-LP     3107   1923.0
106   2017_300234062325760              SVP-I-BXGSA-L-AD      461    282.0
108   2017_300234062328750              SVP-I-BXGSA-L-AD     3165   1923.0
68    2015_300234060104820                 SVP-I-BXGS-LP     8533   4688.0
223   2023_300534064036660                       iCALIB     8220   4504.0
63    2014_300234061763040                       iCALIB     2654   1271.0
38   2011_300234010035940b                 SVP-I-XXGS-LP    31900  14723.0
105   2017_300234062324750              SVP-I-BXGSA-L-AD      481    221.0
44    2012_100000000000000                     DMR-800L      603    248.0
89    2016_300234063513450                       iCALIB    11638   4669.0
42   2011_300234010958690b                 SVP-I-XXGS-LP     8124   2977.0
90    2016_300234063515450                       iCALIB    12554   4238.0
69    2015_300234060435010                 SVP-I-BXGS-LP     2311    716.0
86    2016_300234062950220                 SVP-I-BXGS-LP    13676   4142.0
88    2016_300234062957250                 SVP-I-BXGS-LP    10882   3066.0
21    2010_300034013723350  Model 703 Ice Tracking Buoy      456    126.0
20    2010_300034013721340  Model 703 Ice Tracking Buoy       67     13.0
107   2017_300234062327750              SVP-I-BXGSA-L-AD     3117    565.0
85    2016_300234061768060                       iCALIB     2874    517.0
51    2012_300234010082470                 SVP-I-XXGS-LP     2342    328.0
15    2009_300034012571050                     ICEB-I-XA     3858    422.0
```

203	2021_300234011751690	iCALIB	18022	1731.0
207	2021_300234060725890	iCALIB	11516	1100.0
204	2021_300234011751700	iCALIB	12752	1128.0
202	2021_300234011750710	iCALIB	12650	1046.0
201	2021_300234011750690	iCALIB	12947	1020.0
34	2011_300034013463170	Model 703 Ice Tracking Buoy	5433	384.0
87	2016_300234062951220	SVP-I-BXGS-LP	12275	774.0
22	2010_300034013726340	Model 703 Ice Tracking Buoy	524	32.0
56	2013_300034013464170	Model 703 Ice Tracking Buoy	4922	286.0
55	2013_300034013464160	Model 703 Ice Tracking Buoy	7647	433.0
205	2021_300234011752700	iCALIB	13718	775.0
200	2019_300434063495310	ITB v2.0	2909	161.0
199	2019_300434063494100	ITB v2.0	5171	281.0

	dups_percent	lon_d	lat_d
115	81.512050	5.0	5.0
109	75.751338	4.0	4.0
70	67.055734	4.0	4.0
62	61.892501	4.0	4.0
106	61.171367	4.0	3.0
108	60.758294	4.0	4.0
68	54.939646	4.0	4.0
223	54.793187	4.0	4.0
63	47.889977	4.0	4.0
38	46.153605	4.0	4.0
105	45.945946	3.0	4.0
44	41.127695	4.0	4.0
89	40.118577	4.0	3.0
42	36.644510	4.0	4.0
90	33.758165	4.0	4.0
69	30.982259	4.0	4.0
86	30.286634	4.0	3.0
88	28.174968	4.0	4.0
21	27.631579	4.0	4.0
20	19.402985	4.0	4.0
107	18.126404	4.0	4.0
85	17.988866	4.0	4.0
51	14.005124	4.0	4.0
15	10.938310	5.0	5.0
203	9.604927	4.0	4.0
207	9.551928	4.0	4.0
204	8.845671	4.0	4.0
202	8.268775	4.0	4.0
201	7.878273	4.0	4.0
34	7.067918	4.0	4.0
87	6.305499	4.0	4.0
22	6.106870	4.0	4.0

```
56        5.810646    8.0    8.0
55        5.662351    8.0    8.0
205       5.649512    4.0    4.0
200       5.534548    6.0    6.0
199       5.434152    6.0    6.0
```

1 Conclusions

The GPSDELAY has now been accounted for, this has allayed concerns a lot. Clearly there were many metocean beacons that had reporting frequencies that were much less then their transmission interval. See below for hte head of the table above.

Note that before GPSDELAY was addressed the table was like this (first few rows). Before at least 23 tracks had duplicates > 10%.

beacon_id	beacon_model	total_n	dups_n	dups_percent
2015_300234061762030	iCALIB	10280	9150	89.01
2014_300234060544160	SVP-I-BXGS-LP	9324	8139	87.29
2017_300234062325760	SVP-I-BXGSA-L-AD	1381	1202	87.04
2017_300234062328750	SVP-I-BXGSA-L-AD	9499	8255	86.90
2015_300234060104820	SVP-I-BXGS-LP	25682	21836	85.02
2014_300234061763040	iCALIB	7961	6577	82.62
2017_300234062324750	SVP-I-BXGSA-L-AD	1442	1182	81.97

After reviewing the current top offenders above (which do not show ‘jerky’ stop start motion, it seems that the duplicates here are due to non-motion (which is the way it is supposed to be).

Closing the case on this.

[ ]: