TRAFFIC

a toolbox for processing and analysing air traffic data

Xavier Olive Luis Basora

The 7th OpenSky Workshop, 21/22 November 2019

EARLY DAYS WITH OPENSKY

```
/* DB for departures and arrivals did not exist yet */
select callsign, s.ITEM as serial, hour
from state_vectors_data4, state_vectors_data4.serials s
where hour>=1488322800.0 and hour<=1501538400.0
and lat>=43.59 and lat<=43.68 and lon>=1.3 and lon<=1.43
and s.ITEM in (-1498608228, 1433801924)
and baroaltitude<300 group by icao24, callsign, s.ITEM, hour;

/* Extended Mode S */
select hour, rawmsg, message, s.mintime, s.serial
from rollcall_replies_data4, rollcall_replies_data4.sensors s
where hour>=1488322800.0 and hour<=1501538400.0
and s.serial in (-1498608228, 1433801924) and message is not null;
```

EARLY DAYS WITH OPENSKY

```
script -f
-c "ssh -p 2230 -l USERNAME data.opensky-network.org"
log.txt

cat log.txt | grep "^|.*" |
sed -e 's/\s*|\s*/,/g' -e 's/^,\|,$//g' -e 's/NULL//g' |
awk '!seen[$0]++' >> log.csv
```

then in Python
(or R, or your favourite programming environment)

```
import pandas as pd
df = pd.read_csv("log.csv")
df['latitude'] = df['latitude'].astype(float)
# followed by more or less efficient code
```

DEFINITION OF NEEDS

You may also need something among:

- airspaces description
- airport's apron layout,
- runway information,
- SID/STAR procedures,
- etc.

DEFINITION OF NEEDS

- A lot of manual process is costly and error prone.
  Processes are not reproducible nor generalisable to different use cases.
- As scientists, we need tools to represent complex ideas concisely, elegantly and safely.
- Learn from mistakes, in terms of performance and design

- An easy, intuitive and programmatic access to common sources of air traffic data (incl. OpenSky Network)
- Common methods to apply on trajectories and airspaces
- A declarative grammar to describe data processing
- Facilities to visualise data in common frameworks

traffic implements such principles in Python, but the principles are language agnostic.

ACCESS TO DATA SOURCES

```
from traffic.data import eurofirs, opensky

switzerland_raw = opensky.history(
    "2018-08-01 05:00",  # UTC time by default
    "2018-08-01 22:00",
    bounds=eurofirs["LSAS"],  # FIR for Switzerland
    # other possible arguments include
    # callsign: Union[None, str, Iterable[str]] = None,
    # icao24: Union[None, str, Iterable[str]] = None,
    # serials: Union[None, str, Iterable[str]] = None,
)
```

ACCESS TO DATA SOURCES

DATA SOURCES - BASIC DATA

European FIRs are available in the library, together with

- a database of aircraft;
- a database of airports;
- a database of navigational waypoints;
- a database of ATS routes;

If you provide a path to these files, you may parse:

- Eurocontrol AIRAC files (DDR or B2B)
- data from other providers (easy to adapt)

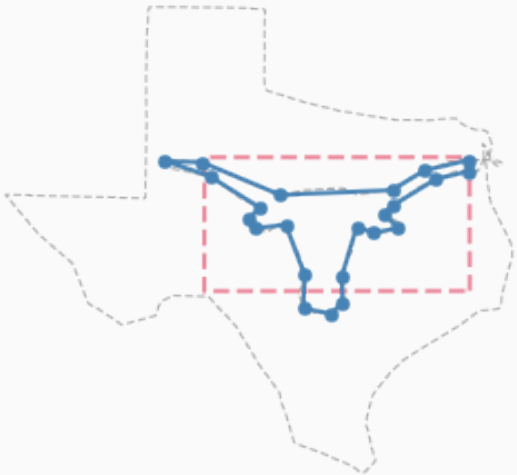
Sample trajectories are provided for demonstration purposes

```
from traffic.data.samples import texas_longhorn
```

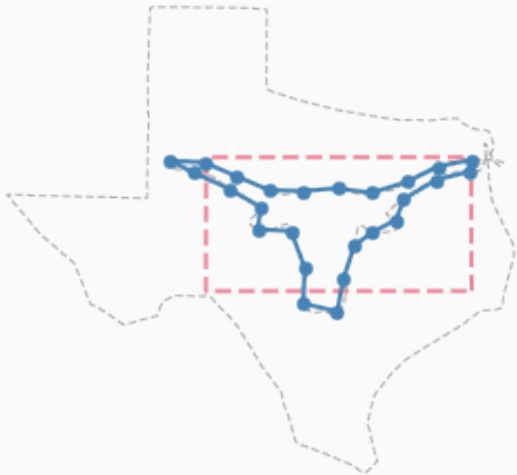
- Methods to apply on trajectories (chainable)

☞ Flight → Optional[Flight]

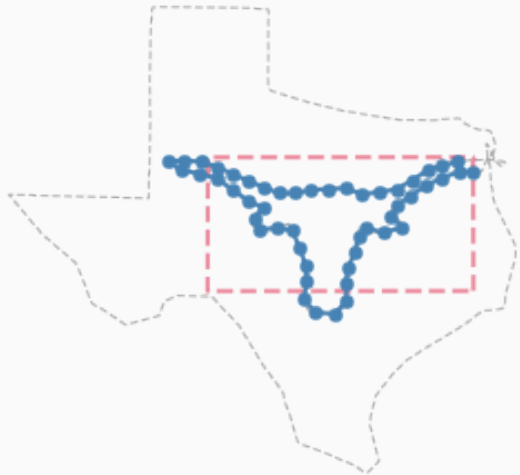
f.clip(box).simplify(1e3)
Douglas-Peucker algorithm



f.clip(box).resample(25)
equally distributed



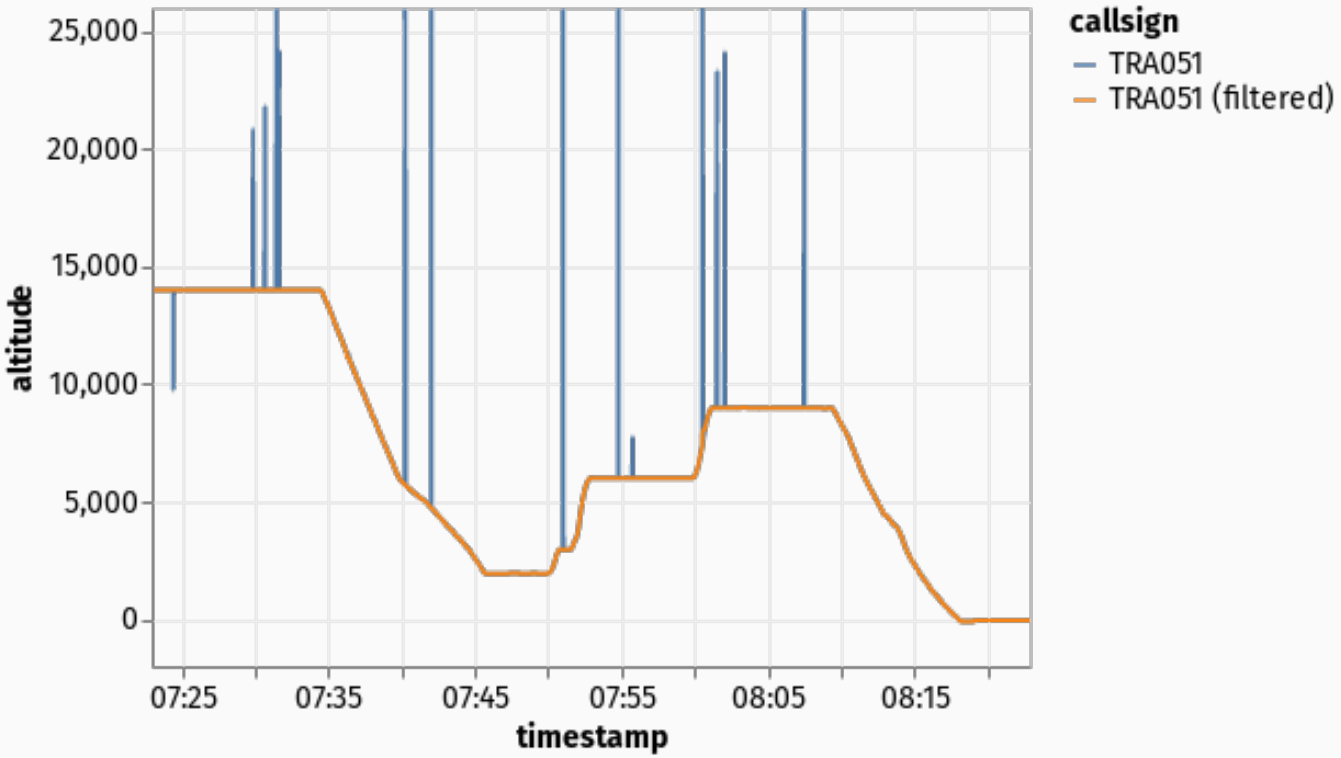
f.clip(box).resample('1T')
one point per minute



```
from traffic.data.samples import belevingsvlucht
```

- Methods for time series, e.g. cascaded median filters

☞ Flight → Optional[Flight]



ITERATION

- Automatic iteration (customisable heuristics), based on:
  - `icao24`, `callsign` and detected gaps in timestamps;
  - a custom `flight_id` feature

```
for flight in switzerland_raw:  # default heuristics
    pass

for flight in switzerland_raw.iterate(by=...):  # custom heuristics
    pass
```

- Automatic indexation based on `icao24`, `callsign`, `flight_id` fields (unique or as lists), integers, slices...

```
switzerland_raw[:50]  # first 50 flights
switzerland_raw['3c6750']  # based on transponder code
```

A DECLARATIVE GRAMMAR

```
switzerland = (
    switzerland_raw
    # a set of heuristics to remove most faulty data
    .clean_invalid()
    # assign identifiers (default pattern: {callsign}_{index})
    .assign_id()
    # cascade of median filters to remove spikes
    .filter().filter(altitude=53)
    # keep only en-route flights
    .filter_if(enroute)
    # resample to one point every 10 seconds
    .resample("10s")
    # multiprocessed evaluation using 4 cores
    .eval(desc="preprocessing", max_workers=4)
)
```

Easily enrich grammar with custom methods

```
import pandas as pd

def enroute(flight: "Flight") -> bool:
    "Returns True if flight is most likely enroute."
    return (
        flight.duration > pd.Timedelta("10 minutes")
        # filter ground vehicles with no track angles (NaN)
        and flight.min("track").notnull()
        # we consider enroute flights never fly below FL300
        and flight.min("altitude") > 30000
    )
```

- Full support of standard visualisation frameworks, namely Matplotlib/Cartopy and altair
- More framework available as plugins, including Leaflet, CesiumJS, Google Earth, Kepler GL, and more
- Self-registration plugin mechanism: plugins may provide custom methods through monkey-patching.
  Plugins can be developped as separate (private?) packages and activated in a configuration file.

RESOURCES

Code base and documentation

- https://github.com/xoolive/traffic/
- https://traffic-viz.github.io/

Supporting notebook

- https://tinyurl.com/opensky2019

PERSPECTIVES

- New data sources/formats (other parts of the world)
- Richer (pragmatic) grammar of preprocessing
- Efficient algorithms for detection of specific events
- Big(ger) data tools
- Interactive large scale visualisation
- Implementations of compatible principles/grammar
  e.g. R, Scala, yet another fun and hype language, etc.

Help and/or feedback welcome!

https://github.com/xoolive/traffic/issues