ΔΗΜΟΚΡΙΤΟΣ

ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΕΡΕΥΝΑΣ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

MSC Data Science

Applied Data Mining

Preliminary data analysis - Automata-based CER with Flink - Logic-based CER with RTEC

Alevizopoulou Sofia 2022201704002

Avgeros Giannis 2022201704003

Tsiatsios George 2022201704024

Athens 2018

Contents

Table of figures

Table of tables

1. Entity Relation model

The tables that we are going to use in this assignment are listed below. They contain real-world data from the maritime domain [https://zenodo.org/record/1167595#.W9BgcFUzapo] which monitor, analyze and visualize the sea movements. There are tables that contain info of the whole maritime activities and their impact on the environment, tables that contain a set of complementary data having spatial and temporal information and tables with information about ships positions within Celtic sea, the Channel and Bay of Biscay (France).

More specifically, there are four categories of data: Navigation data, vessel-oriented data, geographic data, and environmental data. It covers a time span of six months, from October 1st, 2015 to March 31st, 2016.

Tables are:

- aton
- pg_catalog
- country_codes
- pg_temp_1
- geographic_features
- pg_toast.
- geography_columns
- pg_toast_temp_1
- geometry_columns
- ports.
- information_schema
- public.
- raster_overviews
- receiver
- ship_types_detailled_list
- natura2000
- ship_types_list
- navigational_status
- spatial_ref_sys
- raster_columns
- nari_ais_static
- nari_dynamic
- nari_dynamic_aton
- nari_dynamic_sar

We are going to use most of the tables at the next section at which we will try to understand the data.

Create tables

We have added all the above tables at the database.

2. Preliminary data analytics

2.1. Null values

By observing the data, we discovered many columns that contain multiple null values. We decided it would be a good practice not to remove them in this step, but include them in the dataset and decide in one of the next steps how to handle those cases.

Preliminary data analysis refers to some simple and basic analysis by running some SQL queries. This analysis will helpful in understanding the data.

2.2. SQL commands:

1. How many ships there are based on shipname?

```
SELECT Count (DISTINCT shipname)
FROM   nari_ais_static;
```

Count: 4824

Note: Extract this info from static information about ships

2. How many ships have imo number?

```
SELECT Count (DISTINCT imo)
FROM   nari_ais_static;
```

Count: 4033

Note: There are ships with no imo number

3. How many ships there are based on mmsi(dynamic table)?

```
SELECT Count (DISTINCT mmsi)
FROM   nari_dynamic;
```

Count: 5055

Note: Extract this info from dynamic information about ships (ais messages). Here we can see that there are ships which send ais messages but there is no info about them at the static table of ships (nari_static)

4. How many ships there are based on mmsi (static table)?

```
SELECT Count (DISTINCT sourcemmsi)
FROM   nari_ais_static;
```

Count: 4842

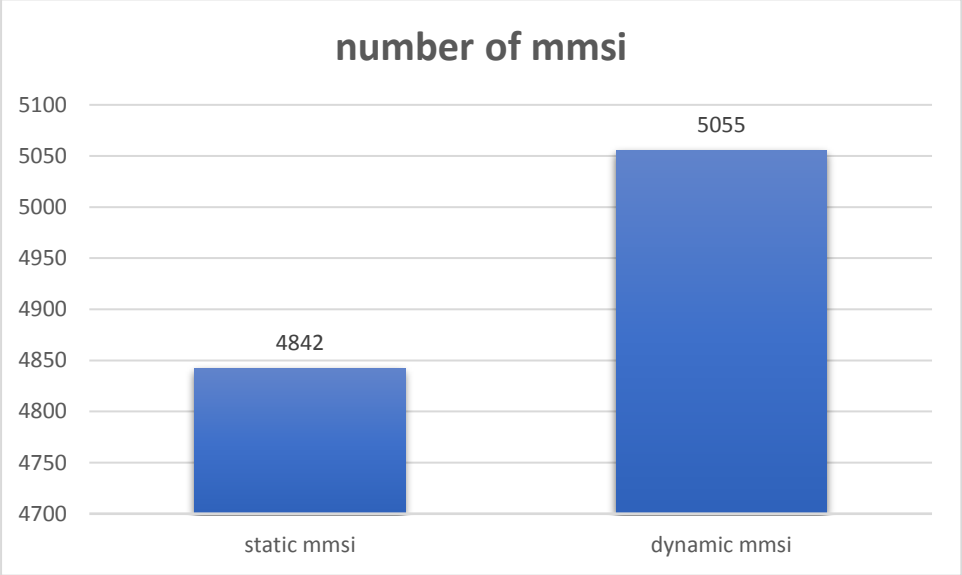

Figure 1:Number of mmsis at 2 tables (static, dynamic)

5. How many ships have mouthermmsi (ships that other ships use them as start point)?

```
SELECT Count (DISTINCT mothershipmmsi)
FROM   nari_ais_static;
```

Count: 228

6. Display 5 ships that have mouthermmsi?

```
SELECT shipname,
       callsign,
       imo,
       sourcemmsi,
       mothershipmmsi,
       detailed_type
FROM   public.nari_ais_static,
       public.ship_types_list,
       public.ship_types_detailled_list
WHERE  nari_ais_static.shiptype = ship_types_detailled_list.id
_detailedtype
       AND ship_types_detailled_list.id_shiptype = ship_types_
list.id_shiptype
       AND nari_ais_static.mothershipmmsi > 0
LIMIT  5;
```

	shipname text	callsign text	imo integer	sourcemmsi integer	mothershipmmsi integer	detailed_type text
1	KINGSTON	MZXE7		235013963	16847107	Suction Dredger
2	IZAR	DGYN		211232180	33558785	Inland Dredger
3	CARRIED AWAY	2HGE4		235103401	23072961	Inland Dredger
4	IZAR	DGYN		211232180	33558785	Inland Dredger
5	KINGSTON	MZXE7		235013963	16847107	Suction Dredger

Figure 2: Ships with mothermmsi

7. Display the most popular mothershipmmsi ship.

```
SELECT mothershipmmsi,
       Count(DISTINCT sourcemmsi) AS num
FROM   nari_ais_static
WHERE  mothershipmmsi > 0
GROUP  BY mothershipmmsi
ORDER  BY num DESC;
```

Mothershipmmsi | count

6320258 | 6

4223106 | 5

8. Display the ships that have as mothershipmmsi the most popular mothershipmmsi ship

```
SELECT DISTINCT sourcemmsi,
                shipname,
                callsign,
                imo,
                mothershipmmsi,
                detailed_type
FROM   PUBLIC.nari_ais_static,
       PUBLIC.ship_types_list,
       PUBLIC.ship_types_detailled_list
WHERE  nari_ais_static.shiptype = ship_types_detailled_list.id
_detailedtype
       AND ship_types_detailled_list.id_shiptype = ship_types_
list.id_shiptype
       AND nari_ais_static.mothershipmmsi = 6320258;
```

	sourcemmsi integer	shipname text	callsign text	imo integer	mothershipmmsi integer	detailed_type text
1	227316100	JEANCANI	FG9660		6320258	Cutter Suction Dredger
2	227591030	SPONTUS	FU5007		6320258	Cutter Suction Dredger
3	227736540	ELORN	FGF6010		6320258	Waste Disposal Vessel
4	227549890	LAITERIE MALO-ESPOIR	FGD2321		6320258	Inland Dredger
5	227635650	ARTEMIS 3	FS5834		6320258	Cutter Suction Dredger
6	227322690	MAM GOZ	FI3738		6320258	Cutter Suction Dredger

Figure 3:Ships with the same mothermmmsi

9. How many ship types there are at the static table?

```
SELECT Count (DISTINCT shiptype)
FROM   nari_ais_static;
```

Count: 45

10. Display the number of ships based on ship type

```
SELECT type_name,
       Count(DISTINCT sourcemmsi) AS num
FROM   PUBLIC.nari_ais_static,
       PUBLIC.ship_types_list
WHERE  ship_types_list.shiptype_min <= nari_ais_static.sh
iptype
       AND ship_types_list.shiptype_max >= nari_ais_stati
c.shiptype
GROUP  BY type_name
ORDER  BY num DESC;
```

	type_name text	num bigint
1	Cargo	2450
2	Tanker	787
3	Fishing	357
4	Cargo - Hazard A (Major)	293
5	Sailing Vessel	222
6	Tanker - Hazard B	123
7	Other	113

Figure 4: Ships per type

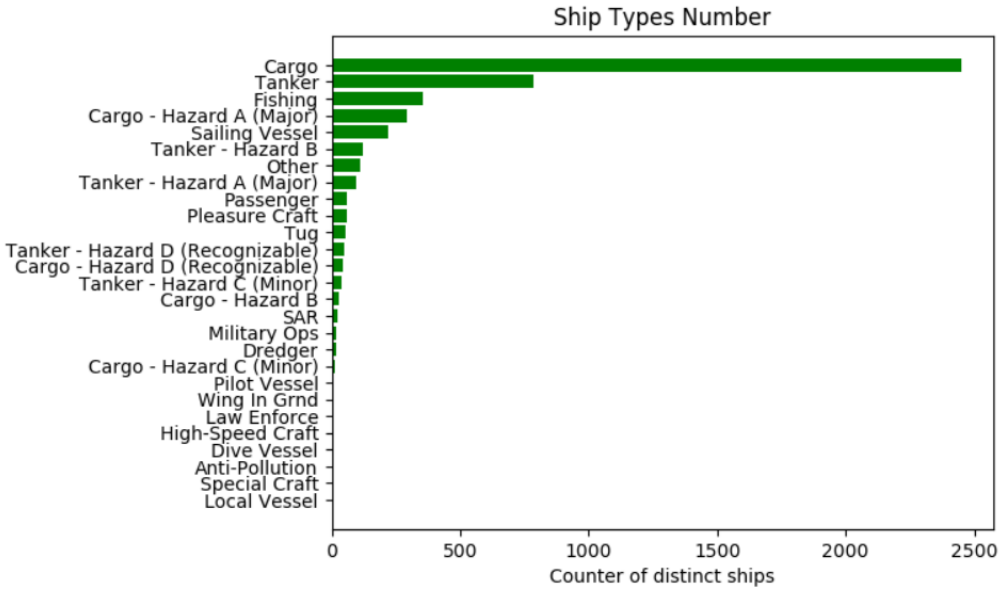

Figure 5 Visualization of number of each vessel type.

11. Find the shipnames that have no ship type defined

```
SELECT DISTINCT shipname
FROM   nari_ais_static
WHERE  nari_ais_static.shiptype IS NULL;
```

	shipname text
1	TORR PENN
2	SEA-EYE
3	RAW ONE
4	NADIA TONY
5	MANEVAI
6	TAMARIND
7	DOLENN

Figure 6:Ships with no type defined

12. Display the number of vessels per country. The first 3 digits of MMSI declare the region country of the ship.

```
SELECT Count(DISTINCT sourcemmsi),
       country
FROM   public.country_codes,
       public.nari_ais_static
WHERE  Left(nari_ais_static.sourcemmsi :: text, 3) :: INTEGER
=
       country_codes.mmsi_country_code
GROUP  BY country;
```

	num bigint	country text
1	715	France
2	368	Netherlands (Kingdom of the)
3	347	Liberia (Republic of)
4	336	Marshall Islands (Republic of the)
5	303	Malta
6	298	Antigua and Barbuda
7	243	Panama (Republic of)

Figure 7:Ships per counrty

13. Display the number of ships and their type per country

```
SELECT Count(DISTINCT( sourcemmsi )),
       country,
       ais_type_summary
FROM   public.country_codes,
       public.nari_ais_static,
       public.ship_types_list
WHERE  Left(nari_ais_static.sourcemmsi :: text, 3) :: INT
EGER =
              country_codes.mmsi_country_code
       AND ship_types_list.shiptype_min <= nari_ais_stati
c.shiptype
       AND ship_types_list.shiptype_max >= nari_ais_stati
c.shiptype
GROUP  BY country,
          ais_type_summary;
```

	num bigint	country text	ais_type_summary text
1	324	France	Fishing
2	307	Netherlands (Kingdom of the)	Cargo
3	284	Antigua and Barbuda	Cargo
4	252	Liberia (Republic of)	Cargo
5	196	Panama (Republic of)	Cargo
6	174	Marshall Islands (Republic of the)	Cargo

Figure 8:Detailed info for ships per counrty

14. Find the ship types for French vessels.

```
SELECT DISTINCT mmsi_country_code
FROM   PUBLIC.country_codes
WHERE  country LIKE 'France';
```

Note: mmsi codes for France are: 226, 227,228

```
SELECT shiptype,
       detailed_type,
       Count(DISTINCT shipname)
FROM   public.nari_ais_static,
       public.ship_types_list,
       public.ship_types_detailled_list
WHERE  nari_ais_static.shiptype = ship_types_detailled_list.id
_detailedtype
       AND ship_types_detailled_list.id_shiptype = ship_types_
list.id_shiptype
       AND ( Left(nari_ais_static.sourcemmsi :: text, 3) :: IN
TEGER = 226
              OR Left(nari_ais_static.sourcemmsi :: text, 3) :
: INTEGER = 227
              OR Left(nari_ais_static.sourcemmsi :: text, 3) :
: INTEGER = 228 )
GROUP  BY shiptype,
          detailed_type;
```

	shiptype integer	detailed_type text	count bigint
1	20	Tractor Tug	1
2	30	Cutter Suction Dredger	331
3	31	Cutter Suction Hopper Dredger	1
4	33	Bucket Dredger	9
5	34	Trailing Suction Hopper Dredge	1
6	35	Trailing Suction Dredger	3

Figure 9:Ships of France

15. Display the most popular destinations

```
SELECT destination,
       Count(DISTINCT sourcemmsi) AS num
FROM   nari_ais_static
WHERE  Length(destination) > 0
GROUP  BY destination
ORDER  BY num DESC;
```

	destination text	num bigint
1	ROTTERDAM	431
2	ANTWERP	231

Figure 10:Most popular destinations

16. Display the type of messages that are sent from ship with shiptype=0.

```
SELECT    Count(*),
          n.status
FROM      (
                 SELECT mmsi,
                        status
                 FROM   PUBLIC.nari_dynamic) a,
          (
                           SELECT DISTINCT(sourcemmsi),
                                           shiptype
                           FROM            nari_ais_static)
 b,
          navigational_status AS n mmsi=sourcemmsi
AND       b.shiptype=0
AND       n.code=a.status
GROUP BY n.status;
```

	count bigint	status text
1	12584	aground
2	8867	at anchor
3	4	constrained by her draught
4	180719	moored
5	1270566	not defined = default (also used by AIS-SART under test)
6	791	not under command
7	210105	restricted manoeuvrability

Figure 11:Messages sent from ships with type=0

17. Display the number of messages that are sent from ships based on their type.

```
SELECT Count(*),
       n.status,
       shl.ais_type_summary
FROM   (SELECT mmsi,
               status
        FROM   PUBLIC.nari_dynamic) a,
       (SELECT DISTINCT( sourcemmsi ),
                        shiptype
        FROM   nari_ais_static) b,
       ship_types_list AS shl,
       navigational_status AS n
WHERE  mmsi = sourcemmsi
       AND shl.shiptype_min <= b.shiptype
       AND shl.shiptype_max >= b.shiptype
       AND b.shiptype != 0
       AND n.code = a.status
GROUP  BY n.status,
          shl.ais_type_summary;
```

	count bigint	status text	ais_type_summary text
1	2055	aground	Fishing
2	12584	aground	Other
3	316	aground	Passenger
4	72556	at anchor	Cargo
5	105	at anchor	Fishing
6	3472	at anchor	Other
7	9	at anchor	Passenger
8	1187	at anchor	Sailing Vessel
9	24004	at anchor	Search and Rescue

Figure 12:Number of messages sent based on ship type

18. Display the number of fishing messages per month

```
SELECT    Count(*) AS num,
          m_month
FROM      (
                    SELECT status,
                           date_part('month',timestamp 'epoch' + t
 * interval '1 second') AS m_month
                    FROM   nari_dynamic) m
WHERE     m.status=7
GROUP BY m_month
ORDER BY num DESC;
```

	num bigint	m_month double precision
1	266961	2
2	190729	3
3	169513	11
4	151189	10
5	119942	12
6	115142	1

Figure 13:Fishing messages per month

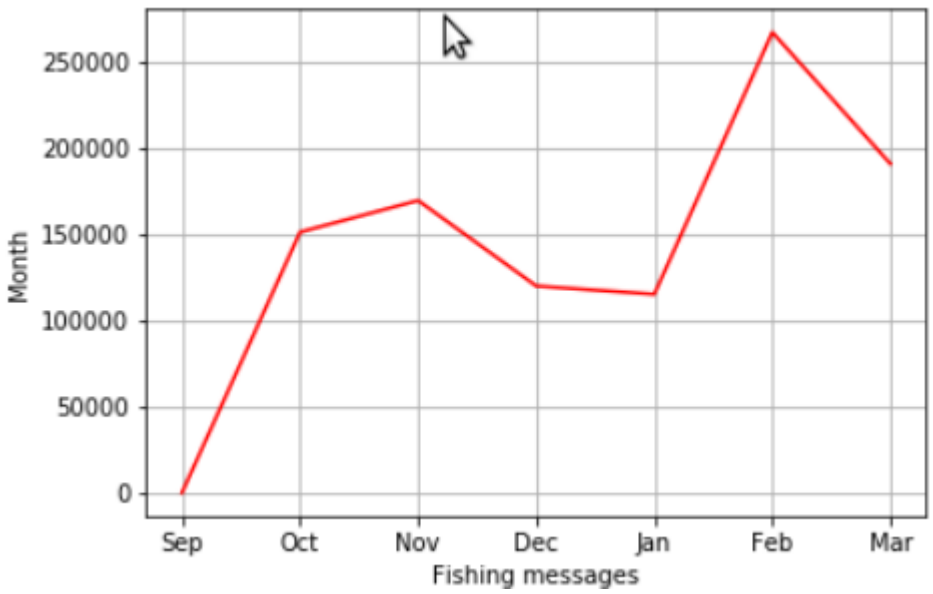

Figure 14:Visualization of fishing messages per month

19. Display the number of messages per month

```
SELECT    Count(*) AS num,
          m_month,
          n.status
FROM      (
                SELECT status,
                       date_part('month',timestamp 'epoch
' + t * interval '1 second') AS m_month
                FROM   nari_dynamic) m,
          navigational_status AS n
WHERE     m.status=n.code
GROUP BY m_month,
          n.status
ORDER BY num DESC;
```

	num bigint	m_month double precision	status text
1	1501657	1	not defined = default (also used by AIS-SART under test)
2	1396348	3	not defined = default (also used by AIS-SART under test)
3	1243249	2	not defined = default (also used by AIS-SART under test)
4	758044	11	not defined = default (also used by AIS-SART under test)
5	735396	10	not defined = default (also used by AIS-SART under test)
6	621065	12	not defined = default (also used by AIS-SART under test)

Figure 15;Number of messages per month

20. Display the types of messages inside a fishing area

```
SELECT Count(*) AS num,
       n.status
FROM   (SELECT DISTINCT maxlong,
                        maxlat,
                        minlong,
                        minlat
        FROM   geographic_features.fishing_areas_eu) g,
       (SELECT lat,
               lon,
               status
        FROM   nari_dynamic) d,
       navigational_status AS n
WHERE  maxlong >= lon
       AND minlong <= lon
       AND maxlat >= lat
       AND minlat <= lat
       AND n.code = d.status
GROUP  BY n.status
ORDER  BY num DESC;
```

	num bigint	status text
1	6260535	not defined = default (also used by AIS-SART under test)
2	1039810	moored
3	1013596	engaged in fishing
4	391680	restricted manoeuvrability
5	179548	at anchor
6	95417	reserved for future amendment of navigational status for ships carrying DG, HS, or MP, or IMO hazard or pollutant category C, high speed craft (HSC)
7	72165	under way sailing

Figure 16:Type of messages inside a fishing area

21. Display the number of fishing messages inside a fishing area

```
FROM    (SELECT DISTINCT maxlong,
                         maxlat,
                         minlong,
                         minlat
         FROM   geographic_features.fishing_areas_eu) g,
        (SELECT lat,
                lon,
                status
         FROM   nari_dynamic) d,
        navigational_status AS n
WHERE   maxlong >= lon
        AND minlong <= lon
        AND maxlat >= lat
        AND minlat <= lat
        AND n.code = d.status
        AND n.code = 7;
```

Count: 1013596

Whereas the total number of messages inside this area is:

```
SELECT Count(*)
FROM    (SELECT DISTINCT maxlong,
                         maxlat,
                         minlong,
                         minlat
         FROM   geographic_features.fishing_areas_eu) g,
        (SELECT lat,
                lon,
                status
         FROM   nari_dynamic) d,
        navigational_status AS n
WHERE   maxlong >= lon
        AND minlong <= lon
        AND maxlat >= lat
        AND minlat <= lat
        AND n.code = d.status;
```

Count: 9092197

Note: fishing messages is 11,147977% of the total messages

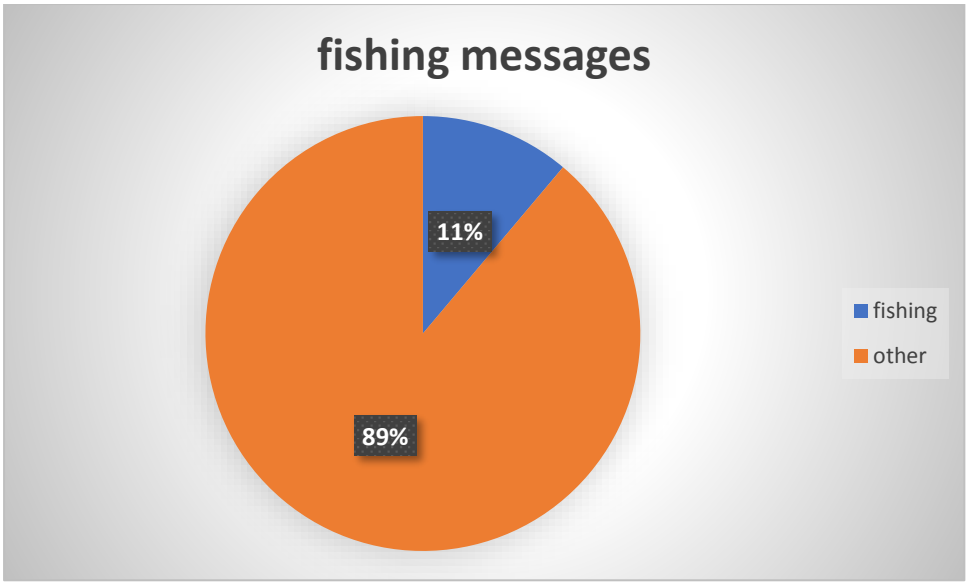

Figure 17:Visualization of total and fishing messages inside a fishing area

22. Display the number of ships that have navigational status "fishing" but out of fishing areas.

```
SELECT Count(*) AS num,
       n.status
FROM   (SELECT DISTINCT maxlong,
                        maxlat,
                        minlong,
                        minlat
        FROM   geographic_features.fishing_areas_eu) g,
       (SELECT lat,
               lon,
               status
        FROM   nari_dynamic) d,
       navigational_status AS n
WHERE  ( maxlong < lon
       OR minlong > lon )
       AND (maxlat < lat

       OR  minlat > lat)
       AND n.code = d.status

       AND n.code = 7
GROUP  BY n.status
ORDER  BY num DESC;
```

Count:0

Note: All ships that have navigational status "fishing" are inside fishing areas.

23. Display the number of ships per type that send fishing messages

```
SELECT type_name,
       Count(DISTINCT sourcemmsi) AS num
FROM   PUBLIC.nari_ais_static,
       PUBLIC.ship_types_list,
       PUBLIC.nari_dynamic
WHERE  ship_types_list.shiptype_min <= nari_ais_static.shiptyp
e
       AND ship_types_list.shiptype_max >= nari_ais_static.shi
ptype
       AND status = 7
GROUP  BY type_name
ORDER  BY num DESC;
```

24. Display the traffic per month

```
SELECT Count(mmsi),
       Extract(month FROM To_timestamp(t)) a
FROM   nari_dynamic
GROUP  BY a
ORDER  BY Count(mmsi) DESC;
```

	count bigint	a double precision
1	3358741	1
2	3353163	3
3	3243176	11
4	3183877	2
5	3130207	10
6	2762531	12
7	3935	4

Figure 18:Traffic per month

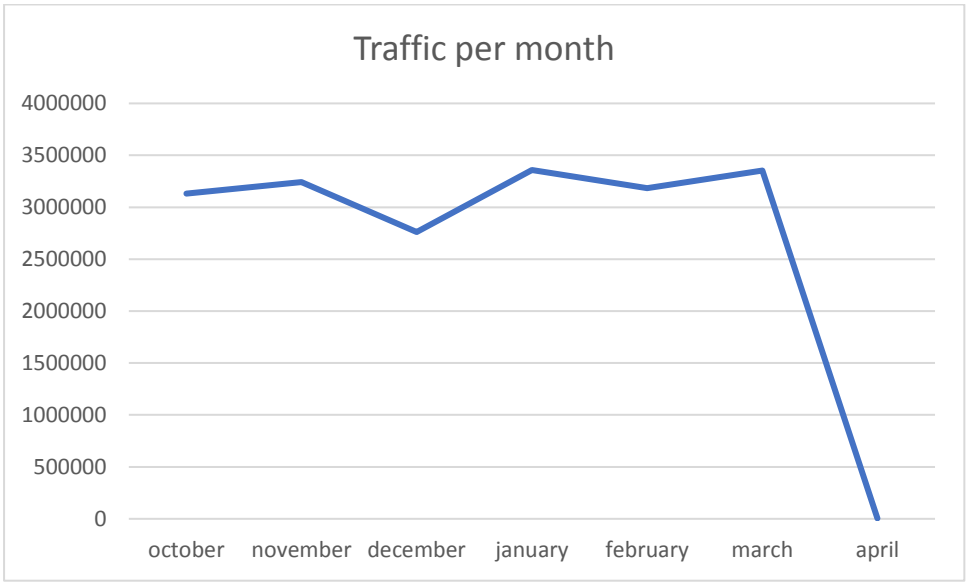

Figure 19:Visualization of traffic per month

25. Display the average speed per Month

```
SELECT Avg(speed),
       Extract(month FROM To_timestamp(t)) a
FROM   nari_dynamic
GROUP  BY a;
```

avg	a
17.0591974493276	1
10.554037420406	2
11.9571499208403	3
13.799288437103	4
4.09282450010901	10
4.28085302802243	11
4.82032302986167	12

Figure 20:Average speed of vessels per month

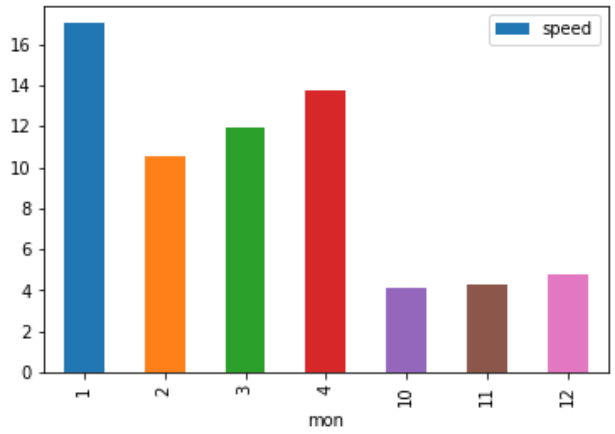

Figure 21: Visualization of Average speed of vessels per month

26. Display average speed, draught, length and width per ship type

```
SELECT b.shiptype,
       type_name,
       Avg(speed),
       Avg(b.draught),
       Avg(b.to_stern),
       Avg(b.to_starboard)
FROM   (SELECT mmsi,
               speed
        FROM   nari_dynamic) a,
       (SELECT DISTINCT( sourcemmsi ),
                        shiptype,
                        draught,
                        to_stern,
                        to_starboard
        FROM   nari_ais_static) b,
       (SELECT type_name,
               shiptype_min
        FROM   ship_types_list
        WHERE  shiptype_min > 0
               AND shiptype_max < 100) c
WHERE  a.mmsi = b.sourcemmsi
       AND c.shiptype_min = b.shiptype
GROUP  BY b.shiptype,
          type_name;
```

shiptype	type_name	avg	avg	avg	avg
20	Wing In Grnd	1.59777777777778	3.69333333333333	3.26666666666666667	2.33333333333333333
30	Fishing	4.93609060771588	1.69934889735548	8.7148782672699574	2.9680948984657168
31	Tug	1.13113637359268	4.10060673763447	19.9533666272396723	5.0599071275244228
32	Tug	0.401223575806678	8.22813871889022	11.4028552649679776	12.0715191904987543
33	Dredger	5.86106486779064	5.2487830952067	28.5842464311174592	7.4229458788190912
34	Dive Vessel	4.06092478311422	7.4	15.9828280118057419	2.9967802522135766
35	Military Ops	2.02956284041059	0.295931659940151	48.6979140566410599	6.3613189084889254
36	Sailing Vessel	3.30649045114843	2.67791987122098	5.1056841225280591	2.4062964239113858
37	Pleasure Craft	1.35407869610842	2.36412262655531	3.8067907187618868	0.900863344181167800076
40	High-Speed Craft	8.27206595538279	2.15322156020505	12.0644312041014272	6.0000000000000000
50	Pilot Vessel	13.1632146855069	0.00544528576859713	2.0970330122924851	1.9973862628310734
51	SAR	56.5769604530518	6.0189955040196	55.6060352110197233	9.5298548616081381

Figure 22:Average speed, draught, length and width per ship type

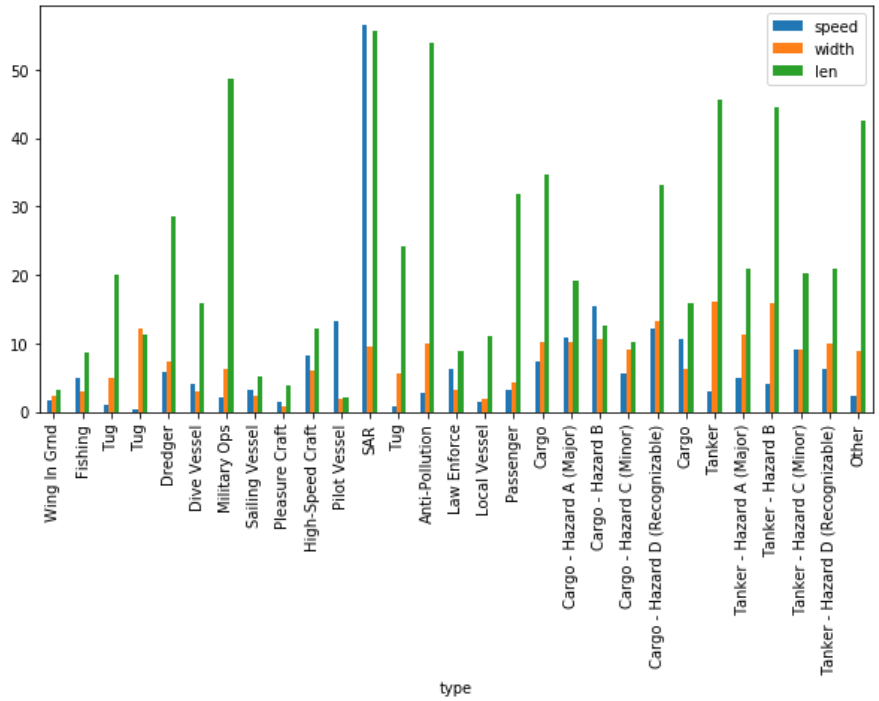

Figure 23:Visualization of average speed, draught, length and width per ship type

27. Display the traffic of Brest port

```
SELECT Count(*)
FROM   nari_ais_static
WHERE  destination LIKE '%BREST%'
        OR destination LIKE '%BES'
        OR destination LIKE ' ';
```

Count of vessels connected with Brest port :640060

Count of all vessels; 1078617

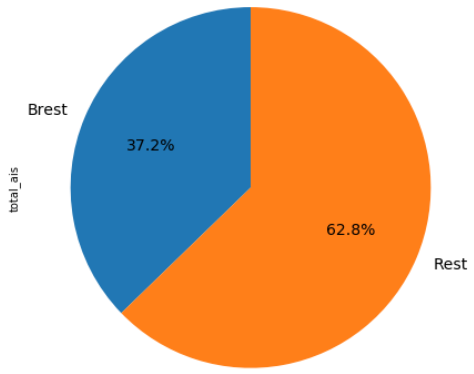

Figure 24:Visualization with the vessels that are connected with Brest Port

28. Display the average of draught, width per month for each ship type

```
SELECT b.shiptype,
       type_name,
       Avg(b.draught)       avg_draught,
       Avg(b.to_starboard) avg_len,
       a.c
FROM   (SELECT mmsi,
               speed,
               Extract(month FROM To_timestamp(t)) c
        FROM   nari_dynamic) a,
       (SELECT DISTINCT( sourcemmsi ),
                        shiptype,
                        draught,
                        to_starboard
        FROM   nari_ais_static) b,
       (SELECT type_name,
               shiptype_min
        FROM   ship_types_list
        WHERE  shiptype_min > 0
               AND shiptype_max <= 100) c
WHERE  a.mmsi = b.sourcemmsi
       AND c.shiptype_min = b.shiptype
GROUP  BY b.shiptype,
          type_name,
          a.c,
          type_name
ORDER  BY a.c;
```

shiptype	type_name	avg_draught	avg_len	c
30	Fishing	1.43644674187529	2.9665094031885342	1
31	Tug	4.10019957015558	5.0379183297513049	1
32	Tug	8.25	12.7500000000000000	1
33	Dredger	5.6850466537058	5.9940454279713314	1
34	Dive Vessel		3.0000000000000000	1
35	Military Ops	0.0013614388398278	6.1129593600895586	1
36	Sailing Vessel	3.48623101036894	3.0082228813877520	1
37	Pleasure Craft	2.29964285714285	1.1814503415659485	1
40	High-Speed Craft	3.7	6.0000000000000000	1
50	Pilot Vessel	0.00258456923846026	2.0021163139476754	1
51	SAR	6.097275797042	9.7340892585258838	1
52	Tug	5.09820913560387	5.5713431697314346	1
54	Anti-Pollution	5.84999999999912	10.0000000000000000	1
55	Law Enforce	1.4	3.5000000000000000	1
57	Local Vessel	2.20000000000024	2.0000000000000000	1
60	Passenger	1.15724124482264	3.9223039814669126	1
70	Cargo	6.65079479312786	10.3411853085653853	1
71	Cargo - Hazard A (Major)	7.34735017483756	10.2153479215794378	1

Figure 25:Average of draught,width per month for each ship type

29. Display the usage of SAR vessels every month

```
SELECT Count(mmsi),
       Extract(month FROM To_timestamp(t)) a,
       Avg(speed)
FROM   nari_dynamic_sar
GROUP  BY a
ORDER  BY Count(mmsi) DESC;
```

count	a	avg
1336	2	178.172155688623
1027	3	211.499513145083
918	10	139.102396514161
792	1	105.010101010101
447	11	71.3243847874720
46	12	26

Figure 26:SAR vessels per month

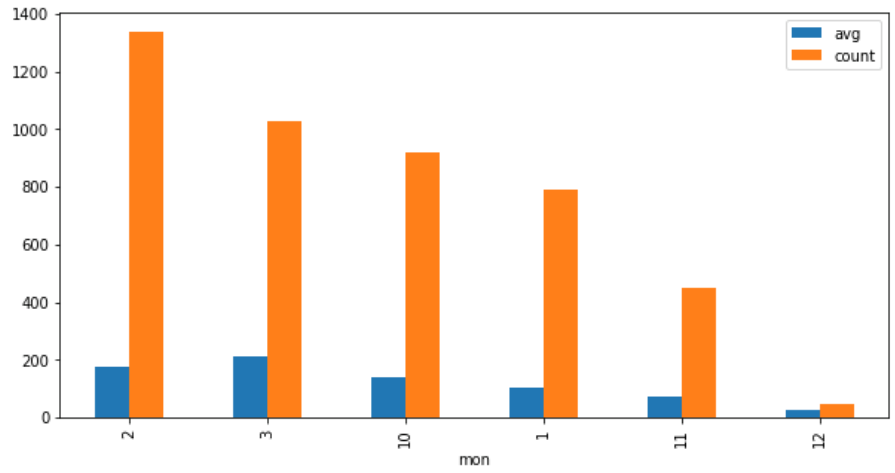

Figure 27:Visualization of SAR vessels per month

30. Display the type of atons

```
SELECT typeofaid,
       Count(typeofaid),
       at.definition,
       virtual
FROM   nari_dynamic_aton a,
       aton at
WHERE  a.typeofaid = at.code
GROUP  BY typeofaid,
          at.definition,
          virtual;
```

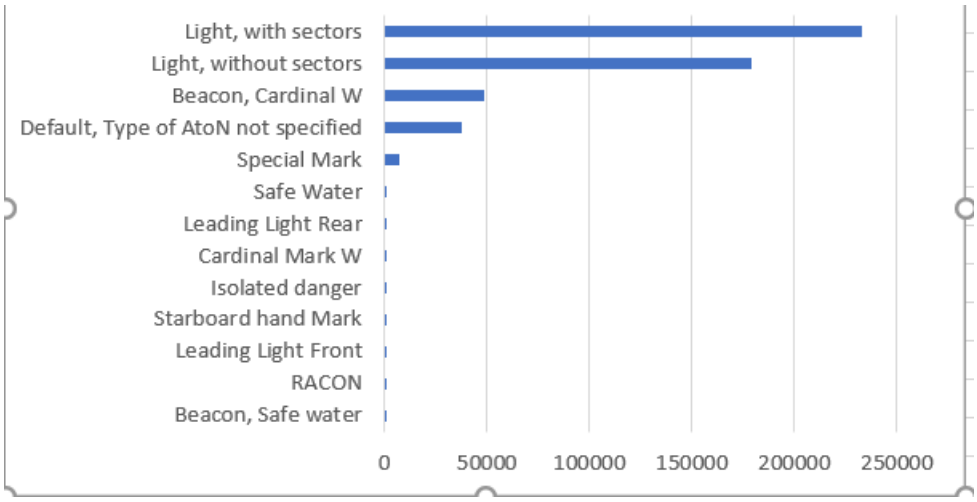

Figure 28: Visualization of type of atons

31. Make a plot for a specific ship according its ais messages (python)

Source code: vessel_route.py

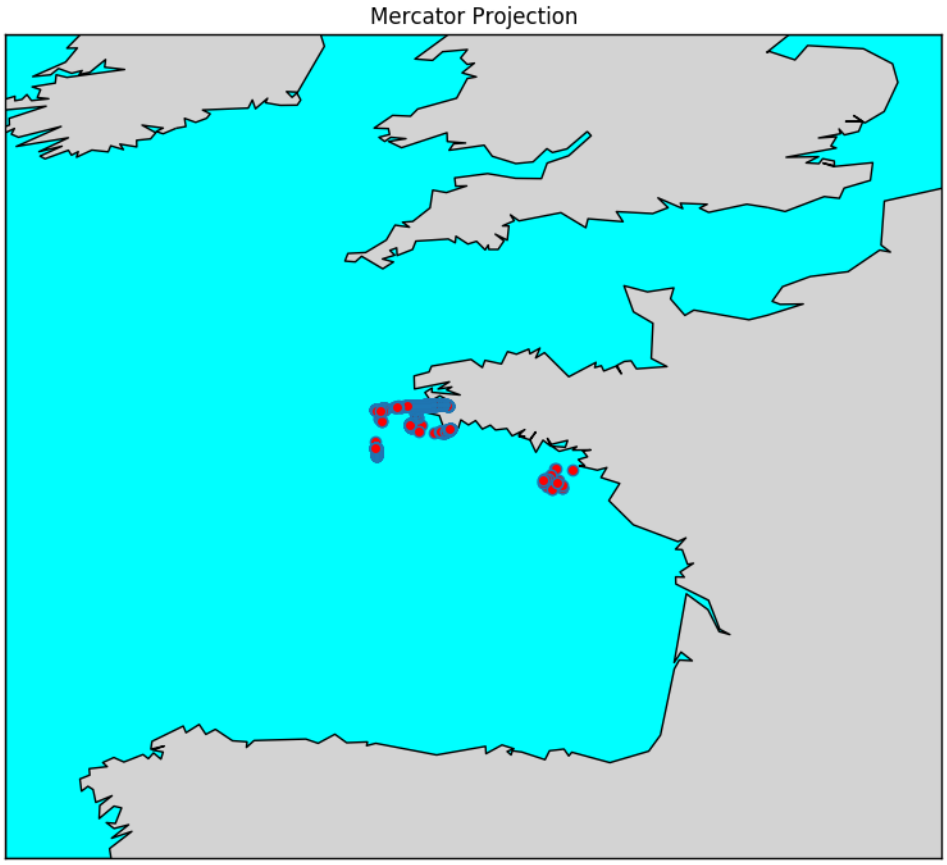

Figure 29:Route of a specific vessel with mmsi= 228931000

32. Make a plot for a specific ship according its fishing ais messages. All these messages are inside a fishing area (QJIS)

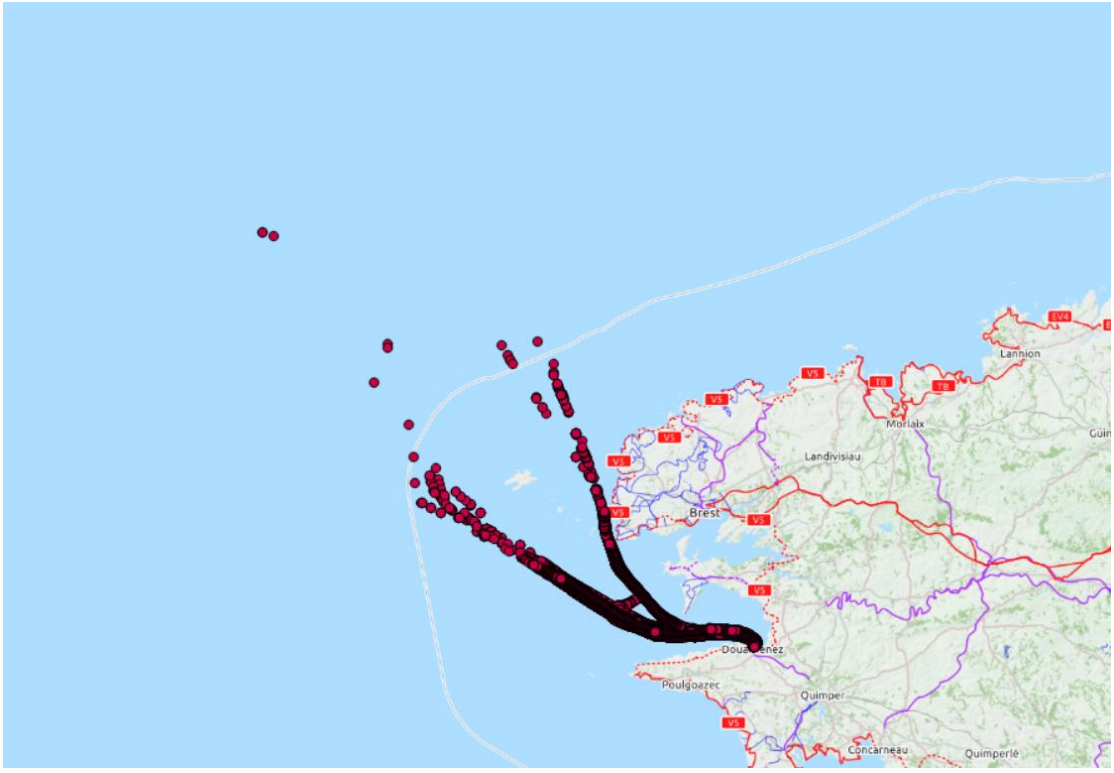

Figure 30: Route of a specific vessel with mmsi= 228931000 with fishing messages

33. Make a plot for a specific ship according its fishing ais messages. All these messages are inside a fishing area. Fishing areas are presented. (QJIS)

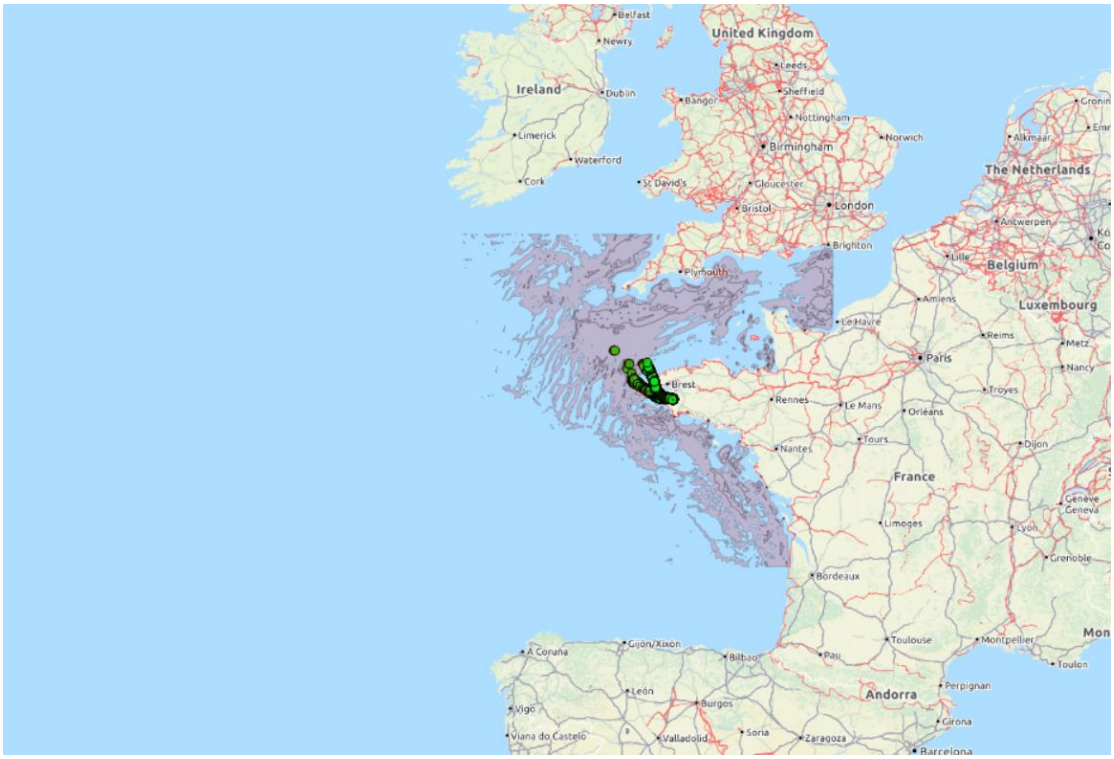

Figure 31: Route of a specific vessel with mmsi= 228931000 with fishing messages, fishing areas are presented

34. Make a plot for a specific ship according its fishing ais messages. All these messages are inside a fishing area. Fishing areas and constraint fishing areas are presented. (QJIS)

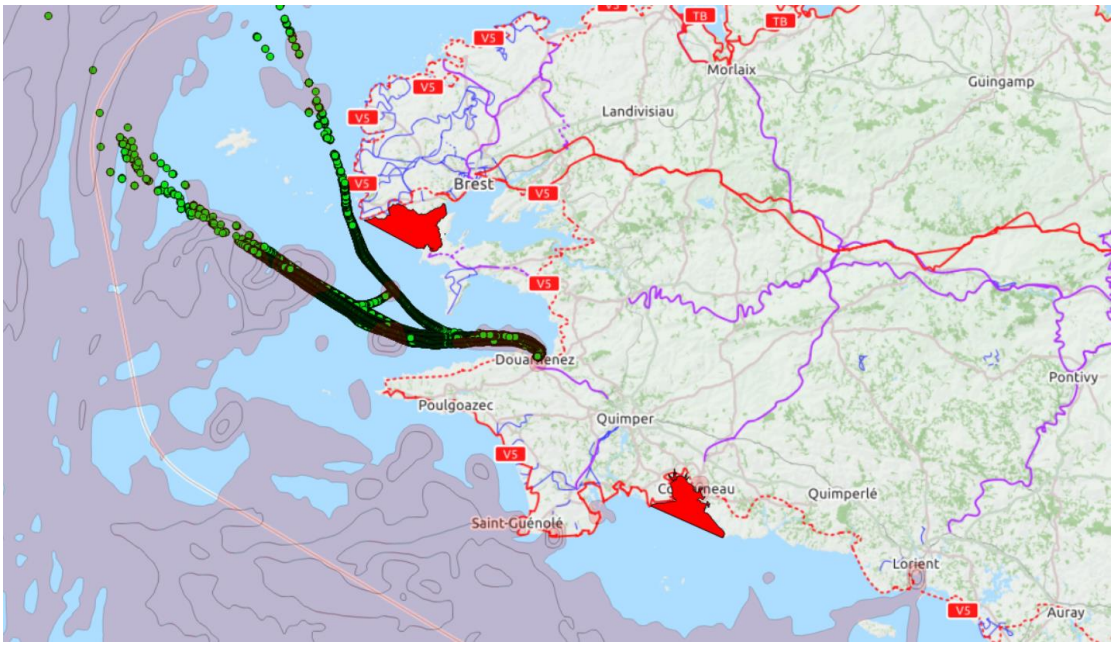

Figure 32:Route of a specific vessel with mmsi= 228931000 with fishing messages, fishing areas and constrained fishing areas are presented (fishing: purple, constraint: red)

35. Plot the route of a specific vessel inside a fishing area (python)

Source code: vessel_at_fishing_area.py

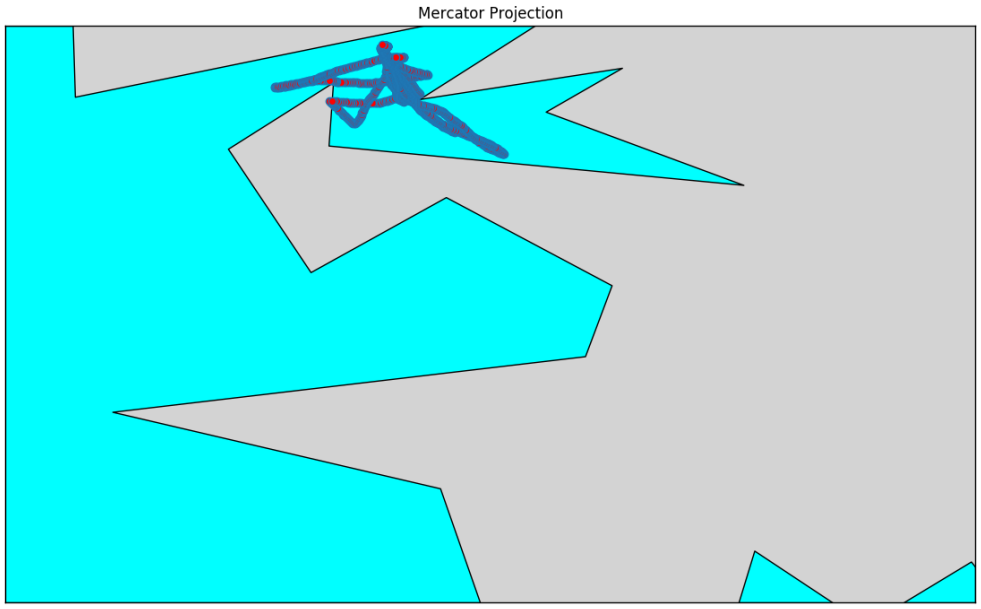

Figure 33:Specific route inside a fishing area for vessel with mmsi= 227741610

36. Plot constrain fishing areas (python)

Source code: constrain_fishing/constrain_fish.py

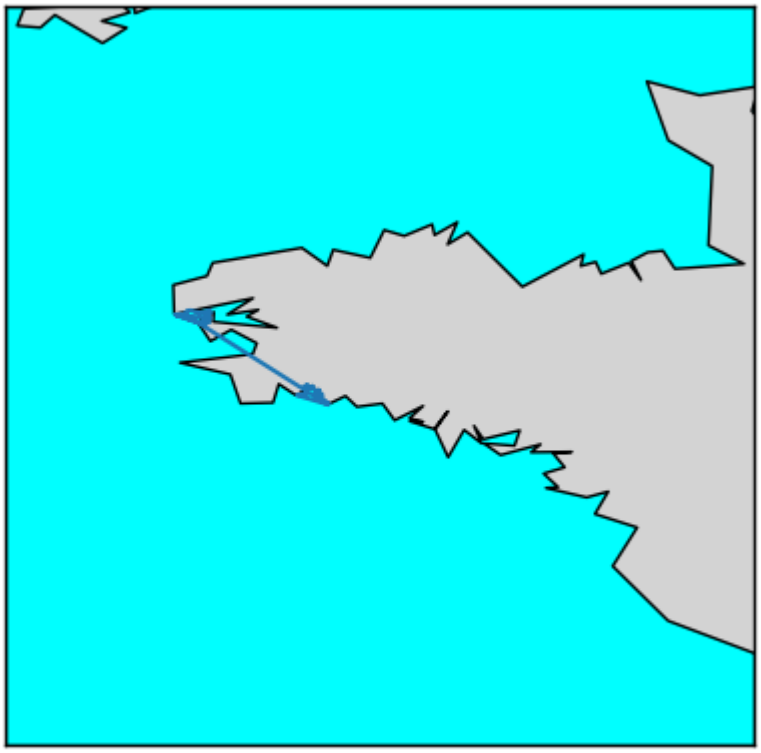

Figure 34: Fishing constraint areas

37. Plot fishing areas(python)

Source code: fishing_area/fishing.py

Figure 35:Fishing areas

38. Plot ports of Brittany(python)

Source code: port_britany/ port_britany.py

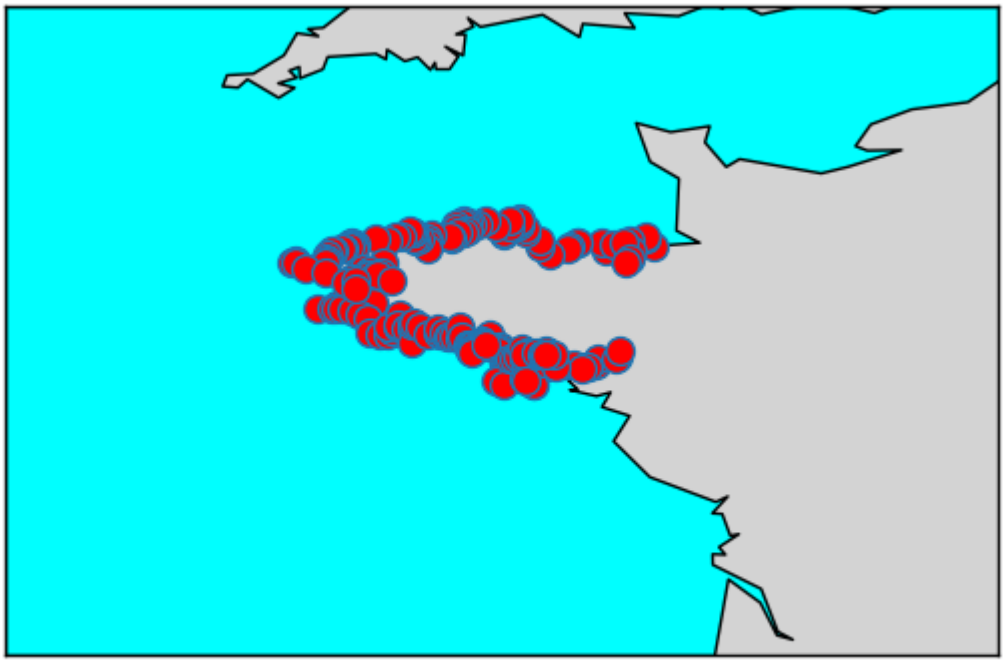

Figure 36:Ports of Brittany

39. Plot the ports of the Europe (python)

Source code: world_port_index/wpi.py

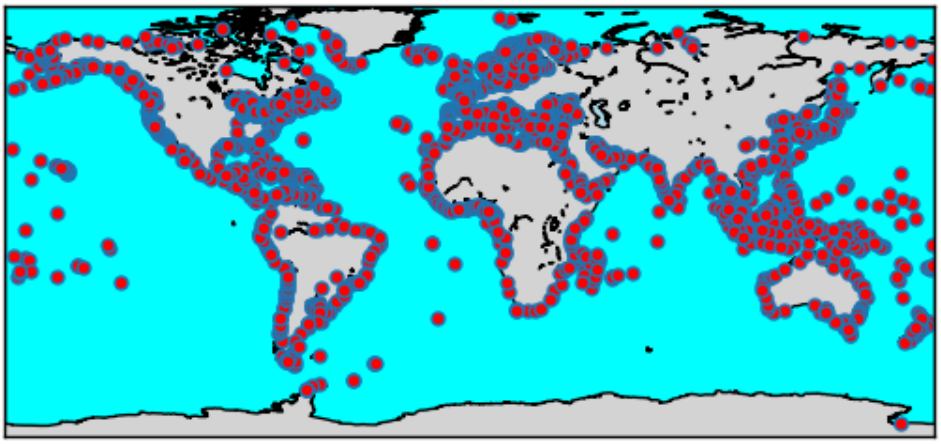

Figure 37:Ports all over the world

2.3. Repository

- The source code for the preliminary data analysis is located at: https://github.com/salevizo/cer.git

3. Automata-based CER with Flink

3.1. Complex event detecting with automata-based framework

Since we have already understand the dataset, a system for online monitoring marine activity over streaming positions from numerous vessels sailing at sea will be presented. The system detects abnormal behavior in AIS messages emitted from vessels across time. AIS messages will be used to recognize suspicious events of one or multiple vessels.

3.1.1. System Architecture

The online monitoring system is a combination of the following systems: kafka and Apache Flink. Kafka is an open source streaming platform which is used for stream processing and apache Flink is a real time processing engine for stateful computations.

A Flink program is defined data streams and their transformations. A stream is a flow of events, and a transformation is an operation that takes one or more streams as input, and produces one or more output streams as an output. When executed, Flink programs are mapped to directed acyclic graphs, consisting of streams and transformation operators. Each graph starts with one or more sources and ends in one or more sinks.

3.1.2. System Deployment

The system implemented detects complex patterns in a stream of AIS messages. AIS messages are stored in a PSQL database. A python script has been written in order to fetch these messages from database and forward them to a kafka topic named “AIS ”.

The system consists of 2 different jobs. The 1st one is used to detect trajectory events for one vessel and the 2nd to detect complex event for more than one vessels based on the trajectory events that have been already detected at the previous step. The 1st project’s “flinkcep” input is what is written at topic AIS (the AIS messages fetched from database). The output of this project is forwarded to other kafka topics like OUT_GAP, OUT_COTRAVEL, OUT_COURSE that given as input to the kafka producer of the 2nd project “cep_flinkcep”. The results of both projects are written in txt and csv files. Each row of the txt/csv files contains information for the detected event. Csv files will be used to make some visualizations.

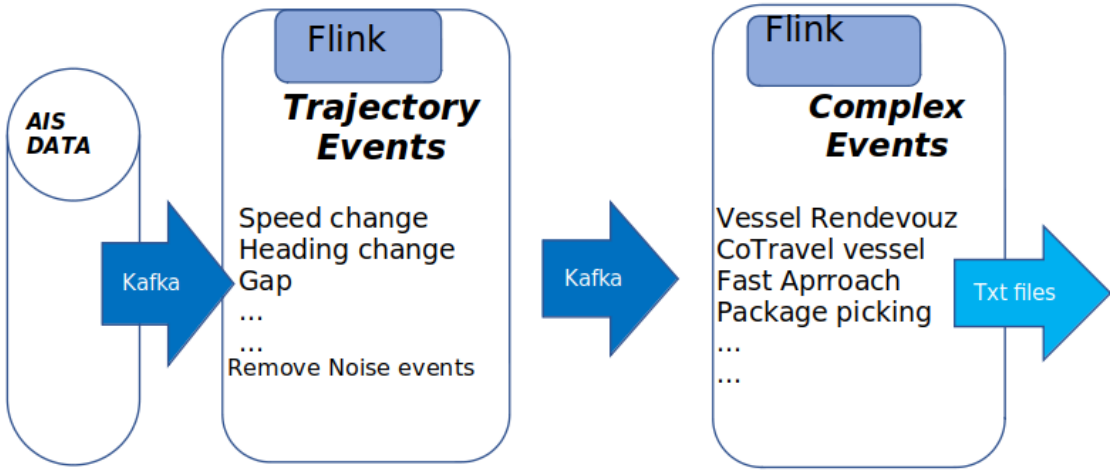

Figure 38: System deployment

3.1.3. Online Noise Reduction

AIS dataset, is a dataset that contains data which can be noisy, and as a result, difficult to be processed. Aiming to have better and more accurate results in the monitoring system, we should remove these noisy messages. In this implementation critical conditions have been added at the flinkcep patterns to identify noisy AIS messages, for examples emitted messages where the shift of the heading is more than 60 degrees. These conditions are checked while processing the patterns in order to improve the performance of the system.

Out of order events

As it is known, transmission delays may frequently occur between the original message and its arrival. Successive positional messages from a single vessel may often arrive intermingled at a distorted order. In our case, we used custom watermarks to handle the timestamp of events and we specifically decided to not accept events arriving out of order. The main reason behind the decision was that the program's execution time was exponentially bigger if we should consider as pattern candidates the unordered events. If this system was in a real world at which we would have delays, we should take care of messages that arrived out of order by adding a small time window of accepted events despite the fact of having out of order timestamps.

3.1.4. Grid partitioning

We decided to translate use the coordinates of a vessel and check how far is from an area or how far is from other vessels *geohash*, a grid portioning method has been used. Geohash is a geocoding system based on a hierarchical spatial data structure which subdivides space into buckets of grid (github.com/davidmoten/geo).

Each cell is labeled using a geohash which is of user-definable precision:
● High precision geohash have a long string length and represent cells that cover only a small area.
● Low precision geohash have a short string length and represent cells that each cover a large area.

GeoHash, can have a choice of precision between 1 and 12. As a consequence of the gradual precision degradation, nearby places will often present similar prefixes. The longer a shared prefix is, the closer the two places are. In the current implementation the below cell dimensions have been used.

We preferred geohash over haversine distance because it produced must faster results for the given latitudes/longitudes.

The following table is presented as an example of the translation between geohash string length and actual distance metric units.

Precision	**Cell dimension**
4	39,1 km x 19,5 km
5	4,9 km x 4,9 km
6	1,2 km x 600,4m
7	152,9km x 152.4m

3.2. Trajectory Events

Trajectory Detection is the main module of the project since the Complex Event Recognition module uses its outcome to compute the patterns that satisfy the conditions predefined. Trajectory Movement can be vessel Speed Change Events, gap in communication and generally events that related with the movement and the behavior of one vessel or more, but mainly serve as a filter for complex events that will be checked for pattern recognition in the following project.

At this phase, system deduce instantaneous events by examining the trace of each vessel alone. This system consumes a stream of AIS tracking messages from vessels and continuously detects important patterns that characterize their movement.

A sample AIS message that is consumed by the patterns written for trajectory detection is:

"lat":2.541122,"lon":3.90484,"mmsi":14,"status":7,"speed":30,"turn":,"heading":36, "course":13,1, "t": 1443650402

Trajectory detection creates a stream of important pattern of events that will be used from the complex event recognition system. Each such event is accompanied by the coordinates or some other characteristics of each corresponding vessel. At each class of trajectory events there is a message serializer class that serializes the accepted events and create another stream. These produced streams will either be used from the 2nd complex event recognition system or either they will be written on txt/csv files if there is no need to forward the stream to the 2nd monitoring system.

Below are described the trajectory events we have implemented.

1. Gaps in communication

System detects vessels which has communication gaps on sending AIS messages. We can define a gap as "*the absence of emitted AIS messages from a specific vessel for a specific period*". In our approach, the ground truth of this time is 600 seconds . So, if the system won't receive an AIS message from a vessel between 600 seconds an its far away from a port (grid 1,2 km x 600,4m), this vessel is tagged as suspicious. This pattern characterized by the critical events "gap_start" – "gap_end". Information about the geohash of ports has been fetched from database.

The flinkcep pattern is written bellow. As time window we have used 900 seconds. That all the continuous AIS messages have to differ 600 seconds in order to be detected from the pattern and the message gap must not exceed the value of 900.For larger windows we faced problem with the ram (heap exceeded).

Table 1: Gap Pattern

```
private static int gapTime=600;

private static int geoHashLen=6;

public static Pattern<AIS Message, ?> patternGap(){

Pattern<AIS Message, ?> rendezvouzPattern = Pattern.<AIS Message>begin("gap_start",
AfterMatchSkipStrategy.skipPastLastEvent())

  .next("gap_end")

   .where(new IterativeCondition<AIS Message>()

@Override

public boolean filter(AIS Message event,Context<AIS Message> ctx) throws Exception {  for (AIS Message
ev : ctx.getEventsForPattern("gap_start")) {

    if ((event.getT() - ev.getT()) > gapTime && (event.getT() - ev.getT()) > 0

      && listOfPorts.contains(GeoHash.encodeHash(event.getLat(),    event.getLon(), geoHashLen)) ==
false)

    {return true;

     }

  }).within(Time.seconds(1200));
```

2. Vessel speed according its type

In this case, the system detects vessels which have very small or very big speed for the vessel type. Each vessel type has its own min and max speeds defined, so in any case that a vessel overcomes these limits can be considered as an event worth recognizing. We obtained and store the information about the ship type of each vessel we have created a csv file which contains all vessel types and all the mmsis for each type (these data fetched from nari_static table on database). For the maximum and minimum speed of each vessel type, there is also a csv file that contains max and min speed per vessel type. This file created from information that is available on web.

The flinkcep pattern is written bellow. As time window we have 30 seconds . That means at the time window of 30 seconds all the AIS messages comes from one mmsi have to satisfy the following conditions in order to be detected from the pattern.

Table 2: Too slow or too fast vessel pattern

```
public static Pattern<AIS Message, ?> patternSpeedVesselType(){
  Pattern<AIS Message, ?> spaciousSpeedPattern = Pattern.<AIS
Message>begin("speed_spacicious_start",

AfterMatchSkipStrategy.skipPastLastEvent())
      .followedBy("speed_spacicious_stop")
      .where(new IterativeCondition<AIS Message>() {
       public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
          for (AIS Message ev : ctx.getEventsForPattern("suspicious_heading_start")) {
                  String mmsi_ = String.valueOf(event.getMmsi());
                  String type="";
                  for (Object o : listOfVesselsType.keySet()) {
                      if (Arrays.asList(listOfVesselsType.get(o)).contains(mmsi_))
                          {
                                  type = o.toString();
                          }
                  }
                  if (type.equals("")==false) {
                      String speed [] = listOfVesselsMaxMinSpeed.get(type);
                      if ( ((event.getSpeed()<Float.valueOf(speed[0]))  ||
                      (event.getSpeed()>Float.valueOf(speed[1])))   && (event.getT() - ev.getT()) > 0) {

                          return true;
                      } else {
                      ……….……
}).within(Time.seconds(30));
```

3. Adrift - Course of ground (COG) differentiates from the heading of a vessel

In this case, the system detects vessels with different heading and course. This can happen when there are extreme weather conditions and vessels are unable to follow the desired route. Heading describes the direction that a vessel is pointed at any time relative to the magnetic north pole or geographic north pole, of 511 degrees indicates noise. As such, a stationary vessel ex. a vessel which has been tied to a dock will have a heading associated with the vessel's orientation. COG describes the direction of motion with respect to the ground that a vessel has moved relative to the magnetic north pole or geographic north pole. An alert sign could be sent in this case. COG equals to 511 means unavailable gps, and these messages will be excluded. Accepted difference between heading and course over ground is every difference less than 10 degrees whereas suspicious can be considered every difference bigger than 10 and smaller than 60 degrees. At this case we also check the speed of a vessel, if the speed of the vessel is less than 1 KNOT it means that the vessel is anchored, and we do not accept those events. We care about vessels whose speed is between 1-48.6 KNOTS, this indicates that the vessel is under way. [1]

The flinkcep pattern is a simple condition and there is no window time.

Table 3: Heading and CoG pattern

```
static int minDiff=10;
static int maxDiff=60;
static double max_speed=48.6;
static double min_speed=1;
public static Pattern<AIS Message, ?> patternSpaciousHeading(){
 Pattern<AIS Message, ?> spaciousHeading = Pattern.<AIS Message>begin("suspicious_heading_start")
      .where(new SimpleCondition<AIS Message>() {
     @Override
      public boolean filter(AIS Message event) throws Exception {
           float courseDiffToHead=(Math.abs(event.getHeading()-event.getCourse()));
           if (courseDiffToHead>minDiff && courseDiffToHead<maxDiff && event.getSpeed()>min_speed
                && event.getSpeed()<max_speed) {            return true;      …….
```

4. High speed near port

At this case system detects vessels whose speed is bigger than 5KNOTS, they are near ports (grid 1,2 km x 600,4m) and at least 10 consecutive messages of a vessel satisfy those conditions. From the moment that we identify this behavior, the program does not search for intermediate patterns and follows "*skip after match strategy*". Ships that overcome these speed limits can be considered as an ongoing suspicious vessel following a dangerous route. Information about the geohash of ports has been fetched from database.

The flinkcep pattern is written bellow. As time window we have 600 seconds . That means at the time window of 600 seconds all the AIS messages from one mmsi, have to satisfy the following conditions in order to be detected from the pattern.

Table 4:High Speed near port

```
static int indexNearPorts=6;

static int maxSpeed=5;

static int patternTime=600;

static int patternTime=600; public static Pattern<AIS Message, ?> patternSpeedNearPort(){

  Pattern<AIS Message, ?> fastForwardPattern = Pattern.<AIS Message>begin("acceleration_start",
AfterMatchSkipStrategy.skipPastLastEvent())

      .where(new SimpleCondition<AIS Message>() {

       @Override

       public boolean filter(AIS Message event) throws Exception {

          if (listOfPorts.contains(GeoHash.encodeHash(event.getLat(), event.getLon(),
indexNearPorts))== true

                    && event.getSpeed()>maxSpeed) {

                       return true;

          }return false;

             }}).times(10)

          .consecutive()

          .within(Time.seconds(patternTime));
```

5. Long Term Stop

Long-Term stop is only triggered if the vessel is noticed to move slowly after a pause. Pause means that vessel's speed is less than 1KNOT. [1] If the next M messages, within 200 seconds are inside a predefined area (grid 1,2 km x 600,4m) , a long term stop is identified. More specifically, this pattern first detects vessels whose speed is smaller than 1KNOT and far away from ports (grid 1,2 km x 600,4m). After check if the detected vessels don't change their speed more than 2 KNOTS within 1800 seconds and they send AIS messages inside an area (grid 1,2 km x 600,4m), means they haven't moved away. Information about the geohash of ports has been fetched from database.

The flinkcep pattern is written bellow. As time window we have 7200 seconds . That means at the time window of 7200 seconds all the AIS messages from one mmsi have to satisfy the following conditions in order to be detected from the pattern.

Table 5: Long Term Stop

```
public static Pattern<AIS Message, ?> patternLongStop() {
    int longterm = 3600;
    int time_window=7200;
    Pattern<AIS Message, ?> LongTermStopPattern = Pattern.<AIS Message>begin("start",
AfterMatchSkipStrategy.skipPastLastEvent())
        .next("stop")
        .where(new SimpleCondition<AIS Message>() {
          @Override
          public boolean filter(AIS Message event) throws Exception {
            if((event.getSpeed() <1 && listOfPorts.contains(GeoHash.encodeHash(event.getLat(),
event.getLon(), 8)) == false)){
              String geoHash1=GeoHash.encodeHash(event.getLat(),event.getLon(),8);
              return true;
            }
            return false;
          }
        })
        .followedBy("stop_ends")
        .where(new IterativeCondition<AIS Message>() {
          @Override
          public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
```

```
for (AIS Message ev : ctx.getEventsForPattern("stop")) {
   if ( ev.getMmsi() == event.getMmsi()) {
      String geoHash1=GeoHash.encodeHash(event.getLat(),event.getLon(),6);
      String geoHash2=GeoHash.encodeHash(ev.getLat(),ev.getLon(),6);
      if((geoHash1.equals(geoHash2)) && (event.getSpeed()<2) && ev.getSpeed()<2  &&
(Math.abs(event.getT() - ev.getT()) > 1800)){
         return true;
      }
      else{
         return false;
      }
   }.within(Time.seconds(time_window));
      .....
      return false;    }})
```

6. Vessels with false status

The dataset contains vessels with wrong status for example vessels with speed more than 5KNOTS and status moored or at anchor (status =1, status=5). Those ships could be identified as noise.

The flinkcep pattern is a simple condition and there is no window time.

Table 6: Vessels with false status

```
public static Pattern<AIS Message, ?> patternFalseType() {
    Pattern<AIS Message, AIS Message> alarmPattern = Pattern.<AIS
Message>begin("first")
        .where(new SimpleCondition<AIS Message>() {
         @Override
         public boolean filter(AIS Message event) {
            if ((event.getStatus() == 1 || event.getStatus() == 5)) {
               if (event.getSpeed() > 5) {
                  return true;
               } else
                  return false;
              ..... });
```

3.3. Complex event

Complex event Recognition module consumes the output of the Trajectory detection module process the results and recognize in real time potentially complex maritime situations. Below will be described some patterns used for complex event recognition. For some of them, two monitoring systems needed. The 1st system for the trajectory events that are analyzed at the previous section and the 2nd one for combining these events to detect more complex patterns.

1. Two vessels co-travelling

The pattern, checks that the AIS messages from 2 different vessels are between a time period of 15 seconds . The pattern will detect events that happened closely in the time dimension. Next thing that will be checked is if the ships are on route and not paused. The speed of the vessel should be bigger than 1KNOT (trajectory event - 1 KNOT is the minimum speed of a vessel that isn't in anchor). [1] At the end, the geohash of the 2 vessels will be checked, ensuring by that, that the two vessels were located in the same grid. The precision of geohash is 6 (grid: 1.2 km x 600m). Two vessels seem to cotravel if they are close enough (grid: 1.2 km x 600m) for more than 180 seconds.This pattern characterized from events "vessel_1" – "vessel_2".

In the end, all the detected patterns will be inserted into a kafka topic and will be given as input stream at the 2nd online monitoring system "cep_flincep" to detect the desired events.

In order to make the second job more efficient, we decided to insert the id of the vessel that is the smallest between mmsi 1 and 2 as MMSI_1 and to collect all the events per vessel. With that, instead of search through all the accepted events we will consider only the pairs that at least one of the MMSI does not change.

As time window we have used 35 seconds for the 1st pattern and 300 seconds for the 2nd. 35 seconds of the 1st pattern means all the AIS messages (for all vessels) in a time window of 35 seconds should be at the same geohash in order to be detected from the system. The second pattern checks that vessels emitted from the first pattern, follow the following rules:

- Two different messages for two different ships that co-exist in the same grid must at worst be differentiated by a margin of 35 seconds.
- Going backwards to the accepted messages following the first rule, we check whether those messages surpass in total the 180 seconds margin. Base event is considered the latest, and we move iteratively backwards until we find a series of messages that the difference between an event and the base event is more than 180 seconds.
- All this should happen between 300 seconds.

The 1st pattern is :

Table 7: Cotravel for 2 vessels pattern – trajectory events

```
static int minSpeed=1;
static int geoHashLength=6;
static int timeBetweenVesselsMsgs=15;
public static Pattern<AIS Message, ?> patternCoTravel(){
   Pattern<AIS Message, ?> coTravelPattern = Pattern.<AIS Message>begin("vessel_1")
  .subtype(AIS Message.class)
  .followedBy("vessel_2")
  .subtype(AIS Message.class)
  .where(new IterativeCondition<AIS Message>() {
 @Override
 public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
 for (AIS Message ev : ctx.getEventsForPattern("vessel_1")) {
    if(ev.getSpeed()>minSpeed
     && event.getSpeed()>minSpeed
     && ev.getMmsi()!=event.getMmsi()){
      String geoHash1=
      GeoHash.encodeHash(ev.getLat(),ev.getLon(),geoHashLength);
      String geoHash2=
 GeoHash.encodeHash(event.getLat(),event.getLon(),geoHashLength);
        if(geoHash1.equals(geoHash2)==true){
          return true;
         }else{return false;
          }……………
           .within(Time.seconds(timeBetweenVesselsMsgs));
```

The 2nd pattern is:

Table 8: Cotravel for 2 vessels pattern – complex events

```
static int coTravelTime=35;

static int coTravellingTotalTime=180;

Pattern<CoTravelInfo, ?> coTravelattern =
Pattern.<CoTravelInfo>begin("msg_1",AfterMatchSkipStrategy.skipPastLastEvent())

  .subtype(CoTravelInfo.class)

  .oneOrMore()

  .followedBy("msg_2")

  .where(new IterativeCondition<CoTravelInfo>() {

  @Override

 public boolean filter(CoTravelInfo event, Context<CoTravelInfo> ctx) throws Exception{

  int base = event.getTimestamp();

  int currTime = event.getTimestamp();

 List<CoTravelInfo> l = Lists.newArrayList(ctx.getEventsForPattern("msg_1"));

  for (CoTravelInfo ev : Lists.reverse(l)) {

    if ((currTime - ev.getTimestamp()) < coTravelTime) {

      if (event.getMmsi_2() == ev.getMmsi_2()) {

        if ((base - ev.getTimestamp()) > coTravellingTotalTime) {

           return true;

         } else {

            currTime = ev.getTimestamp();

         }}} else {

          return false;

    }…….

          .within(Time.seconds(300));
```

2. Fishing Activity

This pattern combines two trajectory events in order to detect a complex event. Specifically, it checks the sequence of continuous changes of the heading (changes between 15 and 60 degrees) , followed by a gap in communication (600 seconds) and a final turn in the end (change between 15 and 60 degrees). This can be considered as an alert sign for illegal fishing. The captain checks the area and afterwards closes its GPS. At that point the system will check just the events that have been characterized by the tag "gap_start" – "gap_end" in order to detect some more changes in heading of the vessels. The trajectory events of heading change and gap in communication have been already analyzed.

The flinkcep pattern is written bellow. As time window we have used 1200 seconds . That means at the time window of 1200 seconds the AIS messages from one mmsi have to satisfy the following conditions in order to be detected from the pattern.

Table 9: Fishing pattern

```
static int headingChangeMin=15;
static int headingChangeMax=60;
static int gapTime=600;
 public static  Pattern<AIS Message, ?> patternFishing(){
  Pattern<AIS Message, ?> fishingPattern = Pattern.<AIS Message>begin("start")
.subtype(AIS Message.class)
.followedBy("gap_start")
.subtype(AIS Message.class)
.where(new IterativeCondition<AIS Message>() {
 @Override
 public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
   for (AIS Message ev : ctx.getEventsForPattern("start")) {
    if(Math.abs(ev.getHeading()-vent.getHeading())>headingChange)
    {return true;
     }else{
        return false;…… }})
.subtype(AIS Message.class)
 .followedBy("gap_end")
 .subtype(AIS Message.class)
```

```
.where(new IterativeCondition<AIS Message>() {
  @Override
  public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
    for (AIS Message ev : ctx.getEventsForPattern("gap_start")) {
        if((event.getT()-ev.getT())>gapTime){
            return true;
         }else{
            return false; }}
    return false;}})
 .followedBy("change in heading  again")
 .subtype(AIS Message.class)
 .where(new IterativeCondition<AIS Message>() {
  @Override
 public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
    for (AIS Message ev : ctx.getEventsForPattern("gap_end")) {
if(Math.abs(ev.getHeading()-event.getHeading())>headingChange){
        return true;
}else{
       return false; }}
  return false;}})
           .within(Time.seconds(600));
```

3. Vessel Rendezvous

The pattern, checks that the AIS messages from 2 different vessels if they have gap in their communication are in the same geohash grid. The precision of geohash is 6 (grid: 1,2 km x 600,4m). The information about the geohash of each vessel is given by the previous pattern "gap-communication" whose outcome is used as input stream ain this pattern. Two vessels may have arranged a rendezvous while on the sea, when they have both gap in their communication at the same time as long as they are at the same geohash grid.
This pattern characterized from events "vessel_1" – "vessel_2".

The flinkcep patterns are written bellow. As time window we have used 900 seconds for the 1st pattern and 120 seconds for the 2nd. 900 seconds of the 1st pattern means all the continuous AIS messages of a mmsi have to differ 600 seconds in order to be detected. With the second pattern, we aim to identify vessels that their gap end event differs at most to 120 seconds.

The 1st pattern is:

Table 10: Vessel Rendezvous pattern - trajectory events

```
private static int gapTime=600;

private static int geoHashLen=6;

public static Pattern<AIS Message, ?> patternGap(){

Pattern<AIS Message, ?> rendezvouzPattern = Pattern.<AIS Message>begin("gap_start", AfterMatchSkipStrategy.skipPastLastEvent())

  .followedBy("gap_end")

   .where(new IterativeCondition<AIS Message>()

@Override

public boolean filter(AIS Message event,Context<AIS Message> ctx) throws Exception {  for (AIS Message ev : ctx.getEventsForPattern("gap_start")) {

    if ((event.getT() - ev.getT()) > gapTime && (event.getT() - ev.getT()) > 0

     && listOfPorts.contains(GeoHash.encodeHash(event.getLat(),    event.getLon(), geoHashLen)) == false)

    {

      return true;

     }

      ……..

  }).within(Time.seconds(1200));
```

The 2nd pattern is:

Table 11: Vessel Rendezvous pattern - complex events

```
public static int gapTime=120;

public static Pattern<GapMessage, ?> patternRendezvouz(){

    Pattern<GapMessage, ?> rendevouzPattern =
Pattern.<GapMessage>begin("Vessel_1")

  .subtype(GapMessage.class)

  .followedBy("Vessel_2")

  .subtype(GapMessage.class)

  .where(new IterativeCondition<GapMessage>() {

  @Override

  public boolean filter(GapMessage event, Context<GapMessage> ctx) throws Exception {

   for (GapMessage ev : ctx.getEventsForPattern("Vessel_1")) {

     if ((ev.getGeoHash().equals(event.getGeoHash()) == true)

       && ev.getMmsi()!=event.getMmsi()) {

         return true;

     } else { return false;

     }}

   return false;}})

          .within(Time.seconds(gapTime));
```

4. Adrift

The pattern, detects vessels that may have a problematic route. The vessels checked at this pattern are already detected as vessels with suspicious difference between heading and course of ground.
As far as the second pattern is concerned, the following prerequisites must be met:

- Two consecutive messages for a vessel must be on a time range of 20 seconds.
- Going backwards to the accepted messages following the first rule, we check whether those messages surpass in total the 180 seconds margin. Base event is considered the latest, and we move iteratively backwards until we find a series of messages that the difference between an event and the base event is more than 120 seconds.
- All this should happen between 240 seconds.

The 1st pattern is:

Table 12: Big difference between heading and course – trajectory event

```
static int minDiff=10;

static int maxDiff=60;

static double max_speed=48.6;

static double min_speed=1;

public static Pattern<AIS Message, ?> patternSpaciousHeading(){

 Pattern<AIS Message, ?> spaciousHeading = Pattern.<AIS
Message>begin("suspicious_heading_start")

      .where(new SimpleCondition<AIS Message>() {

     @Override

      public boolean filter(AIS Message event) throws Exception {

           float courseDiffToHead=(Math.abs(event.getHeading()-event.getCourse()));

           if (courseDiffToHead>minDiff && courseDiffToHead<maxDiff &&
event.getSpeed()>min_speed

                && event.getSpeed()<max_speed) {

                     return true;

           } else {

                     return false; …….
```

The 2nd pattern is:

Table 13: Problematic route - complex events

```
static int messagesTimeGap=20;
 static int problematicRouteTime=120;
   public static Pattern<courseHead, ?> patternSpaciousHeading(){
      Pattern<courseHead, ?> spaciousHeading =
Pattern.<courseHead>begin("suspicious_heading_start", AfterMatchSkipStrategy.skipPastLastEvent())
           .oneOrMore()
           .followedBy("suspicious_heading_stop")
           .where(new IterativeCondition<courseHead>() {
              @Override
              public boolean filter(courseHead event, Context<courseHead> ctx) throws Exception {
                 int base = event.getTimestamp();
                 int currTime = event.getTimestamp();
                 List<courseHead> l =
Lists.newArrayList(ctx.getEventsForPattern("suspicious_heading_start"));
                 for (courseHead ev : Lists.reverse(l)) {
                    if ((currTime - ev.getTimestamp()) < messagesTimeGap) {
                       if ((base - ev.getTimestamp()) > problematicRouteTime) {
                          return true;
                       } else {
                          currTime = ev.getTimestamp();
                       }} else {
                       return false; ……..
}).within(Time.seconds(240));
```

5. Package Picking

This pattern detects possible interaction between two vessels. A possible interaction is when one of them drops a package at some area and another vessel appears later in order to pick it up. By joining the previous long stop events using geohash area (grid: 1,2 km x 600,4m) we find ships that have long stops in the same area where the package picking is possible. As described before the ships should be away from ports. The timestamp between the vessels to be in the same geohash area is 60seconds .

The flinkcep patterns are written bellow. As time window we have used 3600 seconds . That means at the time window of 3600 seconds the AIS messages satisfy the following conditions in order to be detected from the pattern.

```
Pattern<SuspiciousLongStop, ?> alarmPattern = Pattern.<SuspiciousLongStop>begin("first")
    .where(new SimpleCondition<SuspiciousLongStop>() {
   @Override
  public boolean filter(SuspiciousLongStop suspiciousLongStop) throws Exception {
      if(suspiciousLongStop.getMmsi()>0) {
          return true; }else{
        return false; }}})
    .followedBy("picking")
    .where(new IterativeCondition<SuspiciousLongStop>() {
    @Override
  public boolean filter(SuspiciousLongStop event, Context<SuspiciousLongStop> ctx) throws Exception {
      String geoHash1= GeoHash.encodeHash(event.getLat(),event.getLon(),6);
            for (SuspiciousLongStop ev : ctx.getEventsForPattern("first")) {
              if ( ev.getMmsi() != event.getMmsi()) {
              String geoHash2=GeoHash.encodeHash(ev.getLat(),ev.getLon(),6);
                 if((geoHash1.equals(geoHash2)) && (event.getGapEnd() - ev.getGapEnd())<60){
                    return true; }
                else{return false; }}) ……..
        .within(Time.seconds(3600));
```

6. Loitering

Loitering is the act of remaining in a particular area for a long period without purpose. Vessels with low speed, anchored or moored must be filtered out. If the messages from a single vessel are in the same area for 1800 seconds (loitering time , Ltrtime) this vessel is considered to be loitering. So firstly detects vessels that are far away from ports (grid: 1,2 km x 600,4m) with speed bigger than 2.87 KNOTS and smaller than 8 KNOTS and after check id this vessel remain at the same area. Information about the geohash of ports has been fetched from database. [1]

The flinkcep patterns are written bellow. No pattern window is applied we simply search for vessels for loitering. Optional time window would be 3600 (1 hour)

```
Pattern<AIS Message, ?> Loitering = Pattern.<AIS Message>begin("stop")
    .subtype(AIS Message.class)
    .where(new SimpleCondition<AIS Message>() {
      @Override
      public boolean filter(AIS Message event) throws Exception {
        boolean near_ports = false;
        for(String str: Ports) {
          String ship_geohash = GeoHash.encodeHash(event.getLat(),event.getLon(),6);
          if(str.equals(ship_geohash))
            near_ports = true;
        }
        if((event.getSpeed() >2.87 && near_ports == false)){
          if(event.getSpeed() < 8){
            String geohash1=GeoHash.encodeHash(event.getLat(),event.getLon(),6);
            return true;}
          else{
              return false;
          …… })
    .followedByAny("stop_ends")
    .where(new IterativeCondition<AIS Message>() {
      @Override
      public boolean filter(AIS Message event, Context<AIS Message> ctx) throws Exception {
        for (AIS Message ev : ctx.getEventsForPattern("stop")) {
          String geoHash1=GeoHash.encodeHash(event.getLat(),event.getLon(),6)
          if ( ev.getMmsi() == event.getMmsi()) {
            String geoHash2=GeoHash.encodeHash(ev.getLat(),ev.getLon(),6);
            if((geoHash1.equals(geoHash2)) && ev.getSpeed()< 8 && (event.getSpeed()< 8 && (event.getT()-
ev.getT()>ltrtime))){
              if(event.getSpeed()>2.87 && ev.getSpeed() > 2.87){
                return true;
              }
```

3.4. Producer of AIS messages for the kafka topic

The AIS messages are fetched from database with a python script and after given as input at a kafka topic. The python script is inside the path flinkcep/producer and the script is: AIS .py. Running the script kafka topic's name should be defined.

3.5. Watermark Pattern

We take care of the time in our events by introducing the watermark functionality. Watermark is the assigning of a timestamp into an event by using one of its fields instead of using the time that the event was consumed by the system.

Thus, the following implementation is made to retrieve and assign the timestamp of the ais message to the event.

We also extended this functionality to the second project. We wanted also there to have events based on event time and not on the system time. The only variation with respect to the first project is that we use a different field for time extraction.

Table 14: Watermark pattern

```
DataStream<AIS Message> messageStream = env .addSource(new
FlinkKafkaConsumer09<>(
        parameterTool.getRequired("INPUT"),new AIS MessageDeserializer(),props))
        .assignTimestampsAndWatermarks(new Watermarks());
```

3.6. Running Apache Flink Jobs

As analyzed before system consists of 2 projects. For each project there is a different job running on flinkcep. For each job there are different patterns that are checked. The following screenshots show a sample execution of them. The system is running for 1h 48 min, that means system receives AIS messages for 1h 48 min. During this period of time, system detects trajectory and complex events.

You can check how long the job runs as long as the number of events which has been detected.

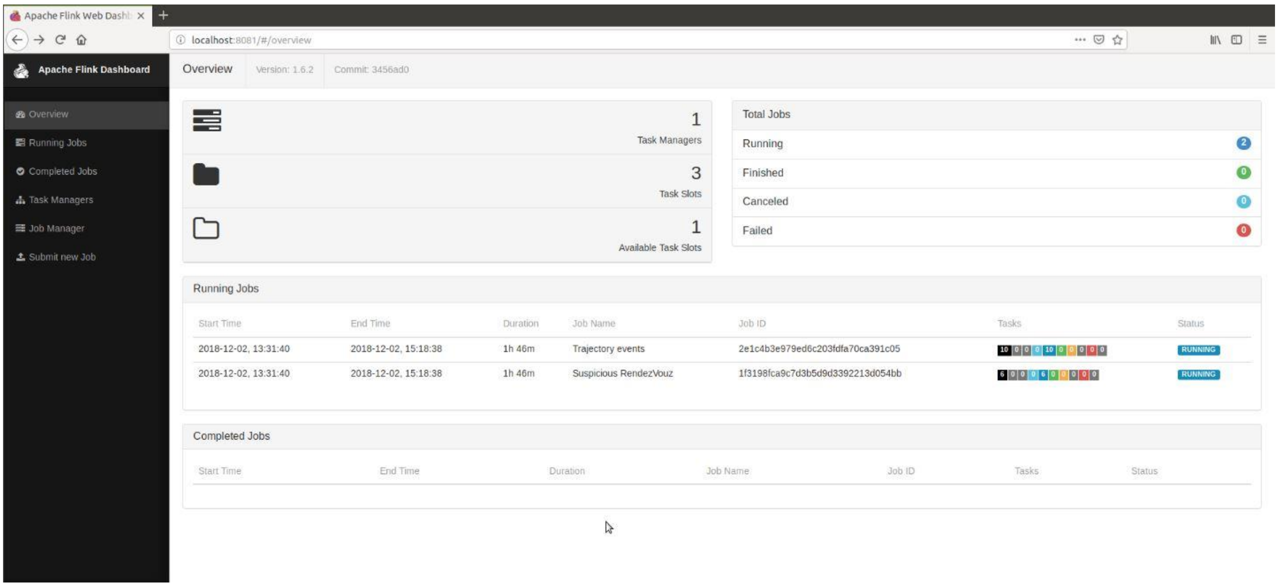

Figure 39:Two jobs are running: trajectory and complex events

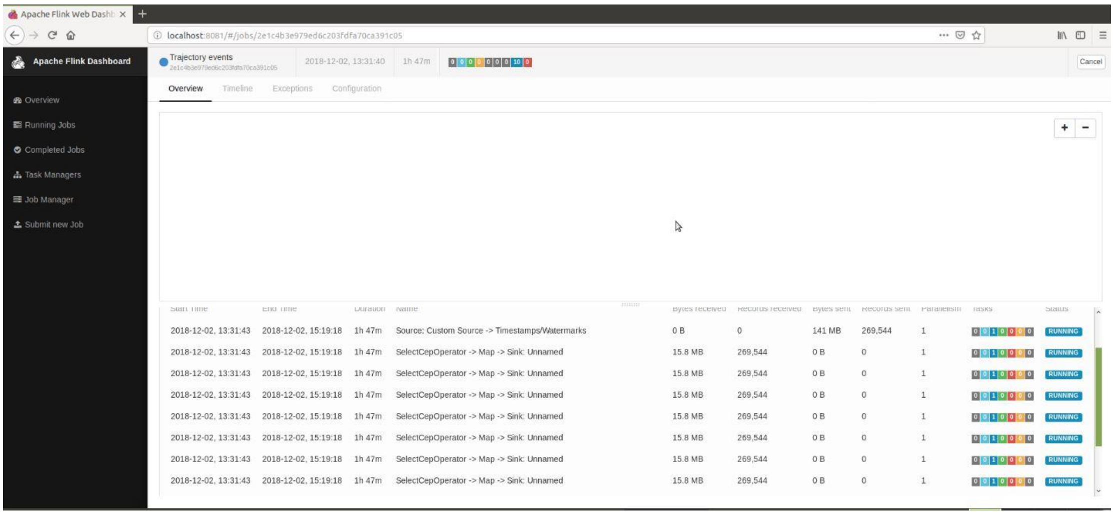

Figure 40: Patterns for job Id: trajectory event (part a)

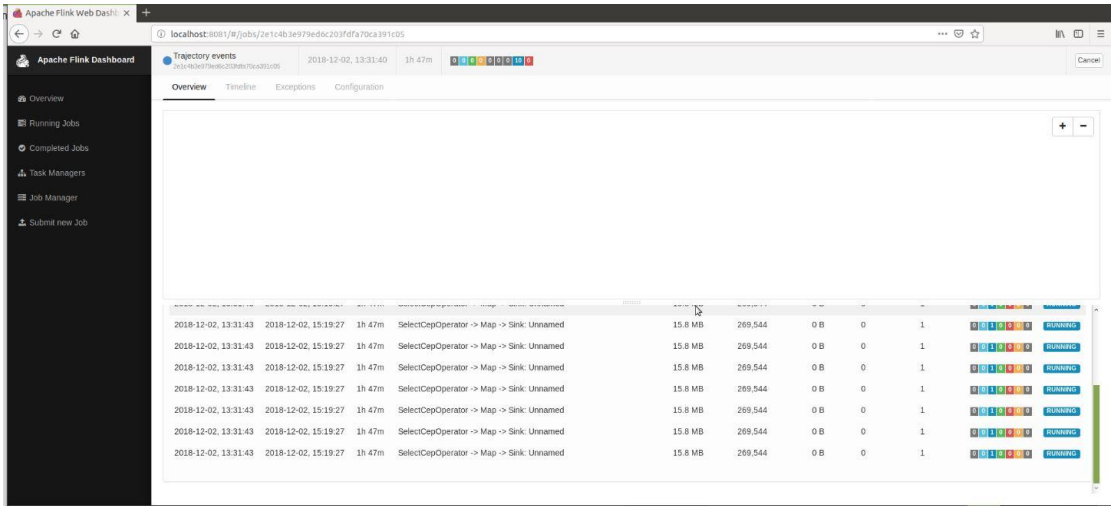

Figure 41:Patterns for job Id: trajectory event (part b)

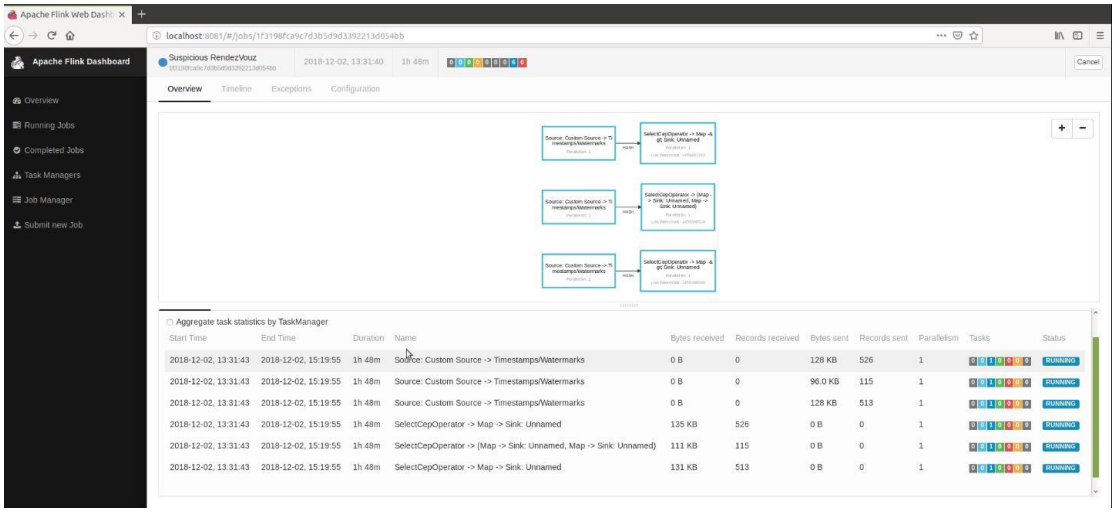

Figure 42: Patterns for job Id: complex event

At this job, which takes as input the output of the 1st job the acyclic graphs are shown, consisting of streams and transformation operators. Each graph starts with one source and end in one sink.

3.7. Outcomes of the running jobs

All the above patterns, while running they write their results (information about the detected events) into .txt files at the path /Desktop/temp. After that, some python scripts will be executed (at the path /conver_txt_to_csv) for parsing these .txt files and make .csv files which contain just the values of fields that are needed to make visualizations on qgis.

The .txt files are listed below:

- speed_near_port.txt, fishing.txt, false_speed.txt, loitering.txt, longstop.txt, packages.txt, rendezvous.txt, cotravel.txt, coursehead.txt, suspicious_speed.txt

The scripts used for these files are:

files	script
coursehead.txt	decode.py
cotravel.txt	decode5.py
loitering.txt	decode1.py
Packages.txt	decode2.py
rendezvous.txt	decode3.py
speed_near_port.txt	decode6.py
fishing.txt	decode4.py

3.8. Empirical Evaluation

Changing the values on mobility tracking parameters eg: seconds of gap in communication, seconds of gap in a suspected fishing etc we can see that the number of events that are detected changes dramatically. Doing some tests with 1.000.000 AIS messages, we conclude at the below results. Due to the fact that some patterns required excessive amount of memory, affected the pattern processing and slow down the whole process. The selection below are for patterns whose output was emitted in a reasonable time for the above amount of AIS messages. We decided to use as parameters values these that were giving us more realistic results (the bold values).

The below results came after running the monitoring system for **1.000.000** ais messages and the time period of ['1443650402','1444660250'].

Rendezvous

GeoHash	Detected events
5 *4.9km x 4.9km*	476
6 ***1.2km x 609.4m***	432
7 *152m x 152m*	**195**
8 *38.2m x 19m*	58

Illegal Fishing

Seconds of gap in a suspected fishing activity(max 1200)	Detected illegal fishing events
300	16
600	**10**
900	9
1200	8

Speed near port

GeoHash for near port	Detected events
5 *4.9km x 4.9km*	1545
6 ***1.2km x 609.4m***	**41**
7 *152m x 152m*	0
8 *38.2m x 19m*	0

3.9. Visualization of the detected events

The below pictures come from QGIS framework using the .csv files with the several complex or trajectory events. These visualizations represent the detected patterns using the bold empirical values on patterns (see previous Section). These are also the values that are written at the patterns' description on this report.

1. Cotraveling activity

The involved mmsis at this plot are 7: 227730220,227005550, 28051000,356101000,227004390,235095836,244740921 and the pattern is the one described at the complex event section.

Figure 43: Cotraveling activity (a)

Figure 44: Cotraveling activity with zoom

2. Loitering activity

The involved mmsi at this plot is 1: 228037700 and the pattern is the one described at the section of trajectory events.

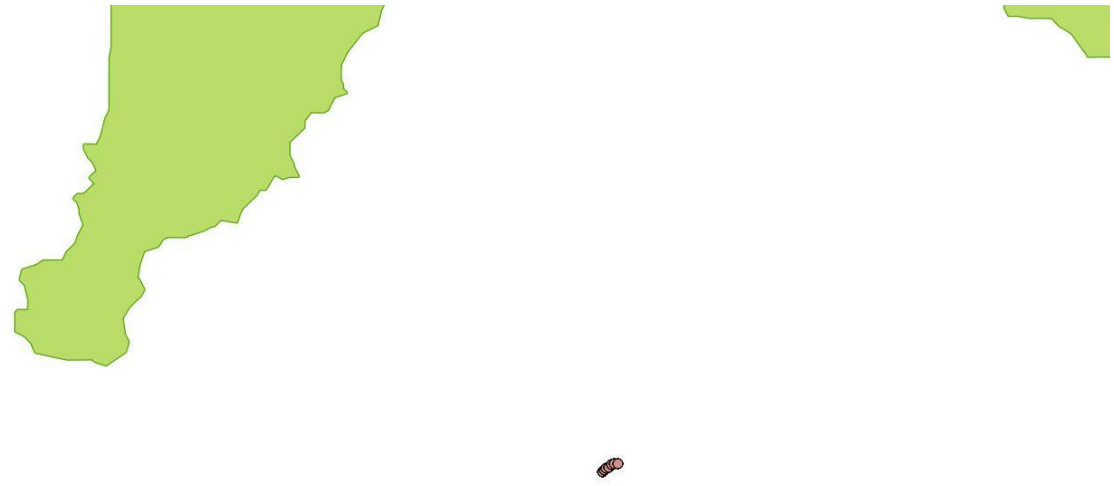

Figure 45: Loitering activity

3. Adrift activity

The involved mmsi at this plot is 1: 228064900 and the pattern is the one described at the complex events section.

Figure 46: Adrift activity

4. High speed near port

The first plot has as geohash index 6 (1.2 km x 609.4 m near port) and the involved mmsis are 6: 227577000, 227002330,228051000, 246254000,227730220,227005550. The detected events at this case are 41.

Figure 47: High speed near port, grid of 1.2km x 609.4m

The second plot has as geohash index 5 (4.9 km x 4.9km near port) and the involved mmsis are 108. The detected events at this case are 1545.

Figure 48: High speed near port, grid of 4.9km x 4.9km (1545 detected events)

As we can see the vessels that detected at 2nd case are also detected at the 1st. Using bigger grid as accepted distance more vessels will be detected. Events that have been detected in distance_a are also detected in distance_b where distance_a<= distance _b. We decided to choose the 1st case.

5. Fishing activity

The blue points on the plot used to describe the case of 300 secs as gap of communication and the involved mmsis are 11: ['228929000', '227003050', '228160000', '226177000', '228017700', '228213700', '227142200', '227318010', '227008170', '228064900', '227300000']. The detected events at this case are 16.

The purple points on the plot used to describe the case of 600 seconds as gap in communication and the involved mmsis are 7: ['227003050', '228160000', '228017700', '228213700', '227142200', '227318010', '227008170']. The detected events at this case are 10.

We can see that the 10 detected events on the 2nd case (600 secs) are also detected in the 1st case (300 secs – 16 events).

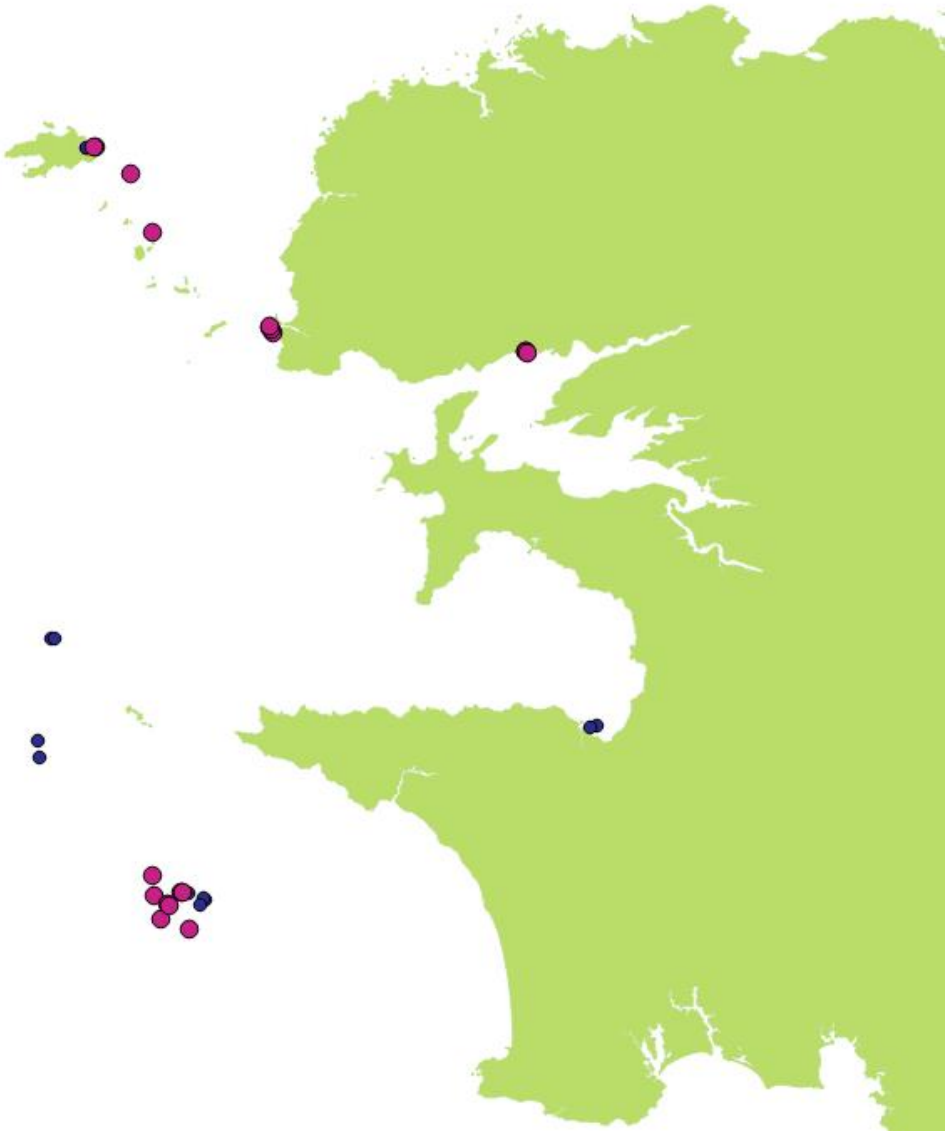

Figure 49: Fishing activity, 300 – 600 seconds gap in communication (blue:300 secs, purle:600 secs)

The black points on the plot used to describe the case of 900 secs as gap of communication and the involved mmsis are 6: ['227003050', '228160000', '228017700', '227142200', '227318010', '227008170']. The detected events at this case are 9.

The purple points on the plot used to describe the case of 600 secs gap in communication (10 detected events).

We can see that the 9 detected events on the 2nd case (900 secs) are also detected in the 1st case (600 secs – 10 events).

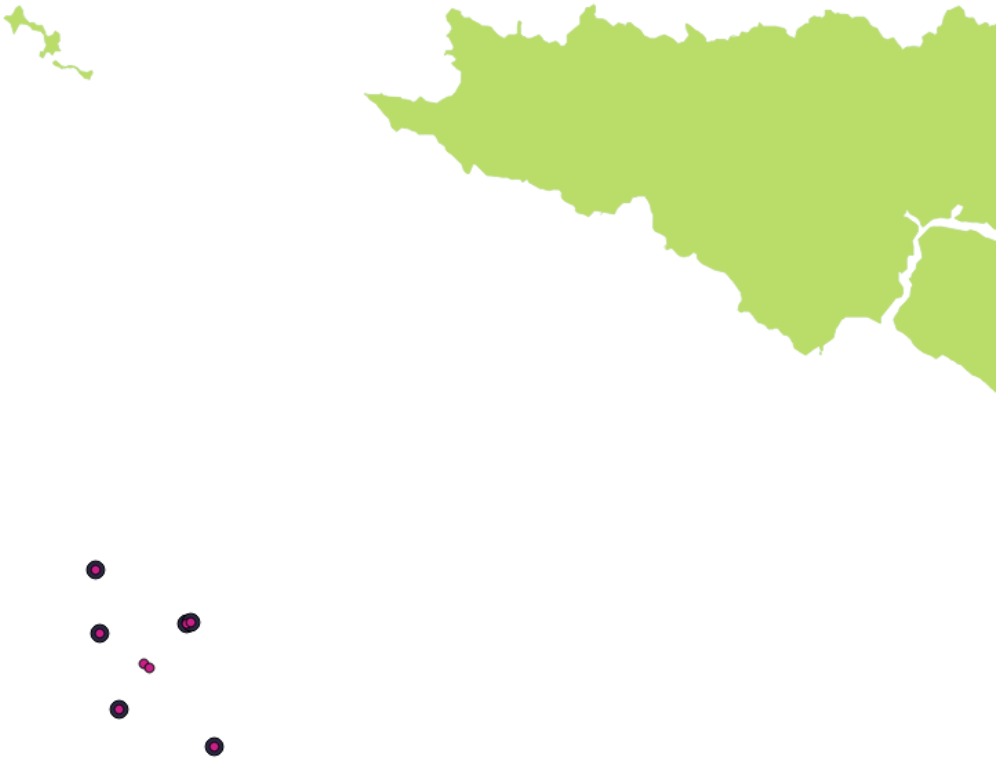

Figure 50: Fishing activity, 600 – 900 seconds gap in communication (purple:600 secs, black:900 secs)

The green points on the plot used to describe the case of 1200 secs of gap in communication and the involved mmsis are 5:['228160000', '227008170', '227003050', '227318010', '228017700']. The detected events at this case are 8.

The black points on the plot used to describe the case of 900 secs of gap in communication (9 detected events).

We can see that the 8 detected events on the 2nd case (1200 secs) are also detected in the 1st case (900 secs – 9 events).

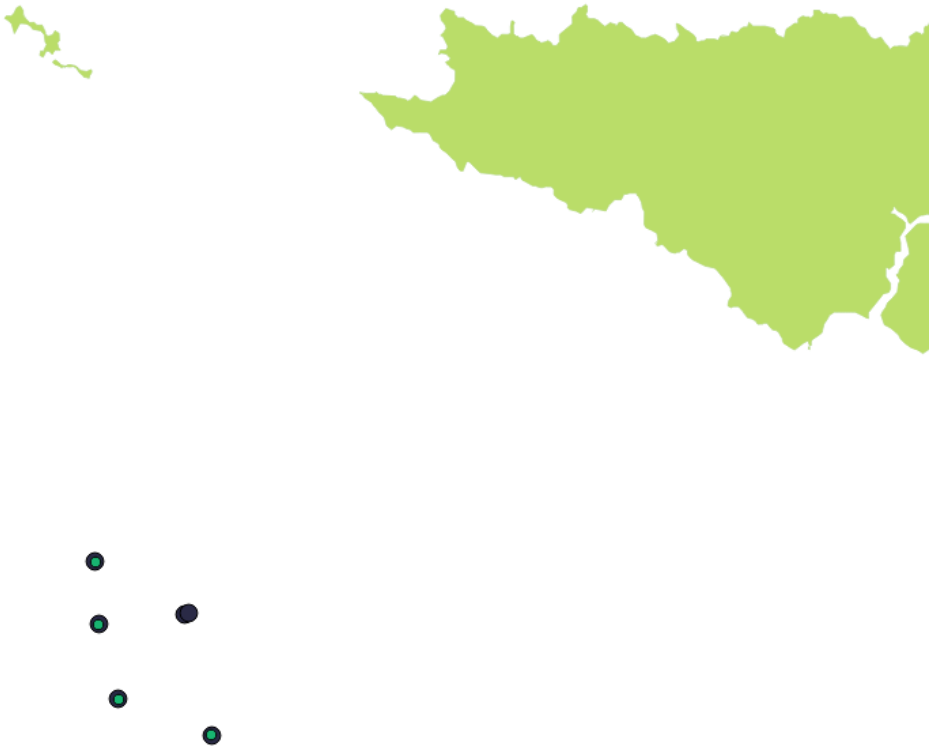

Figure 51: Fishing activity, 900 - 1200 seconds gap in communication (green:1200 secs, black:900 secs)

In other words, increasing the seconds, the pattern detects less events. Events that have been detected in sec_b are also detected in sec_a where sec_a<=sec_b. We decided to use 600 secs (2nd case).

6. Vessels rendezvous

The blue points on the plot used to describe the case of 4.9km x 4.9km as grid and the involved mmsis are 209. The detected events at this case are 476.

The purple points on the plot used to describe the case of 1.2km x 609.4m as grid and the involved mmsis are 179. The detected events at this case are 432.

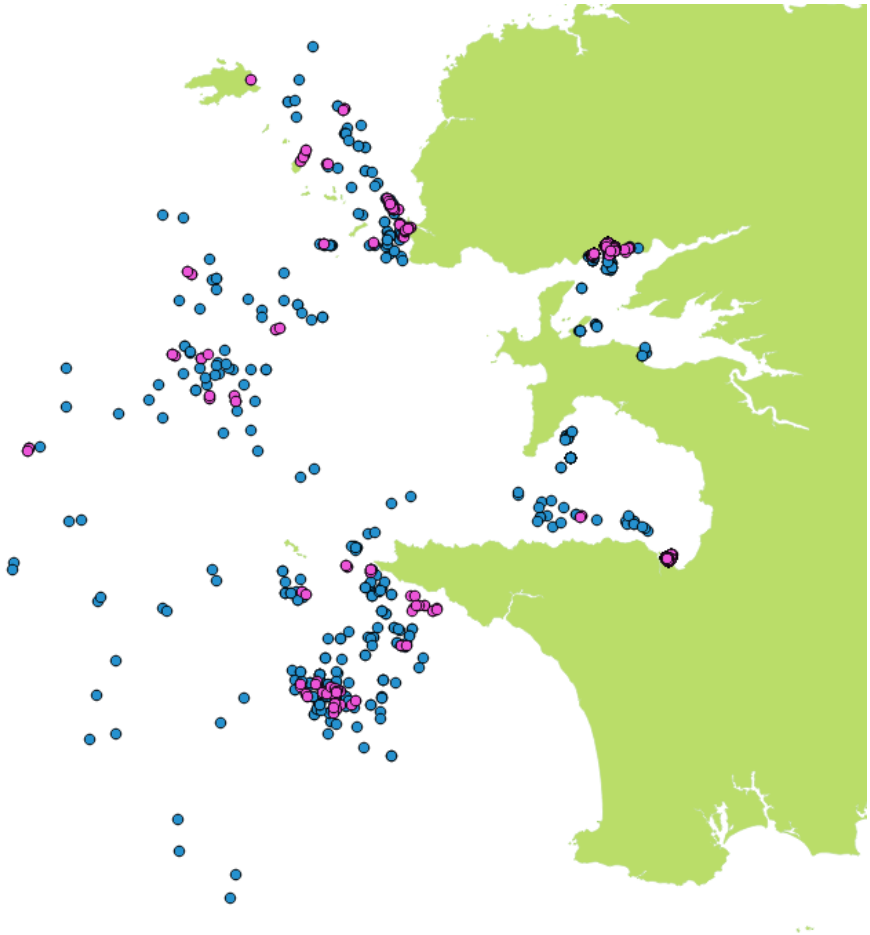

Figure 52: Vessels Rendezvous (blue: grid of 4.9km x 4.9km , purple: grid 1.2km x 609.4m)

The black points on the plot used to describe the case of 152m x 152m as grid and the involved mmsis are 101. The detected events at this case are 195.

The purple points on the plot used to describe the case of 1.2km x 609.4m as grid (432 detected events).

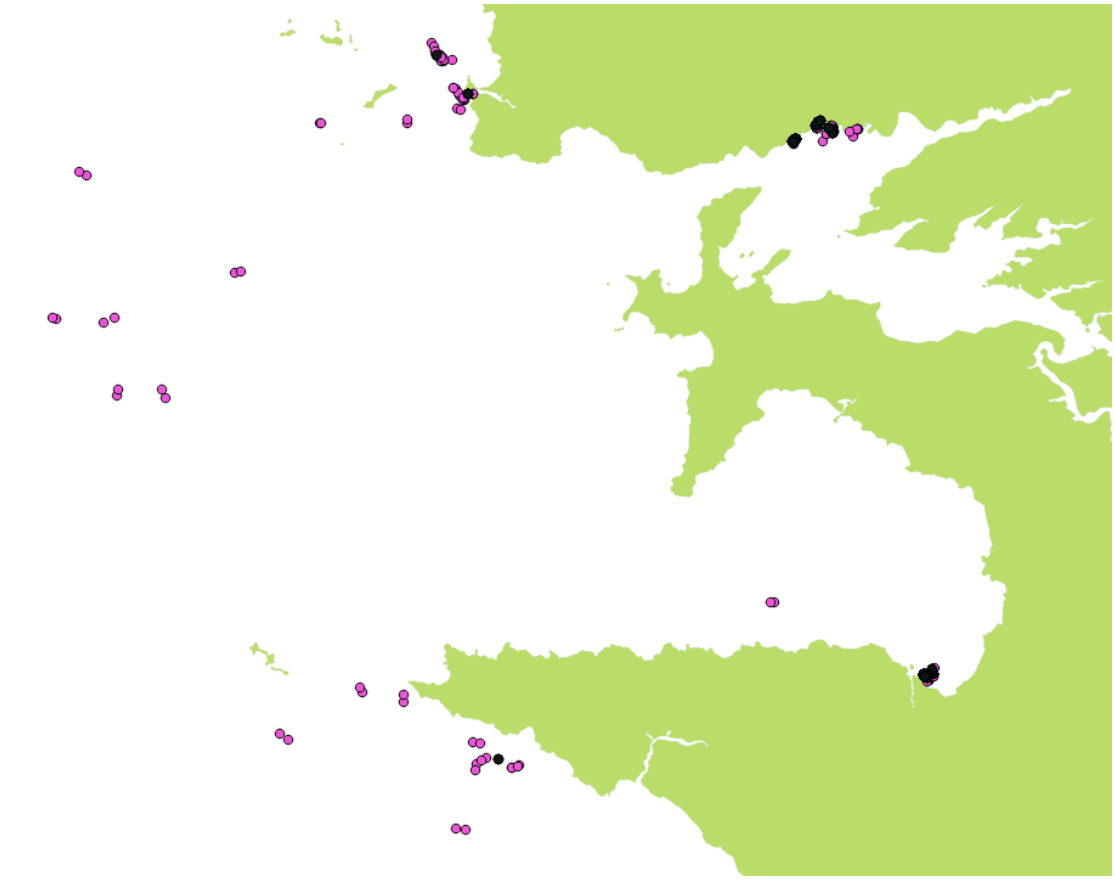

Figure 53: Vessels Rendezvous (black: grid of 152 m x 152m , purple: grid 1.2km x 609.4m)

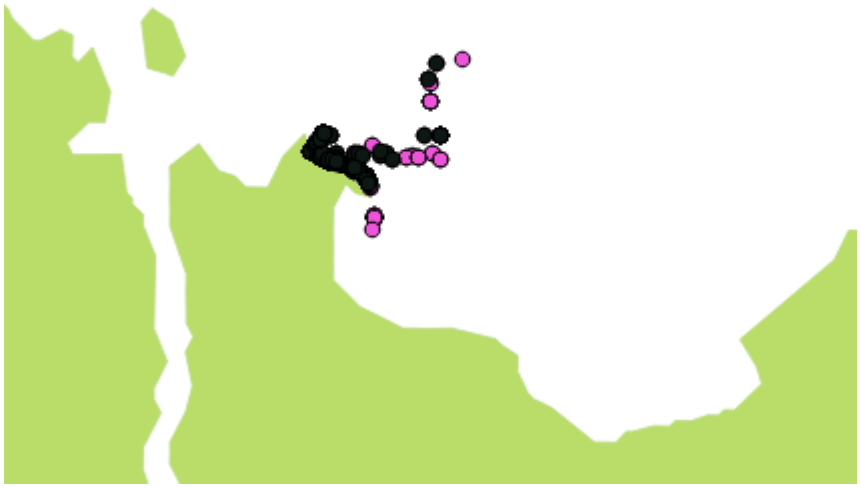

Figure 54: Vessels Rendezvous (black: grid of 152 m x 152m , purple: grid 1.2km x 609.4m) with zoom

The yellow points on the plot used to describe the case of 38.2m x 19m as grid and the involved mmsis are 40. The detected events at this case are 58.

The black points on the plot used to describe the case of 152m x 152m as grid (195 detected events).

Figure 55:Vessels Rendezvous (black: grid of 152 m x 152m , yellow: grid 38,2 m x 19m)

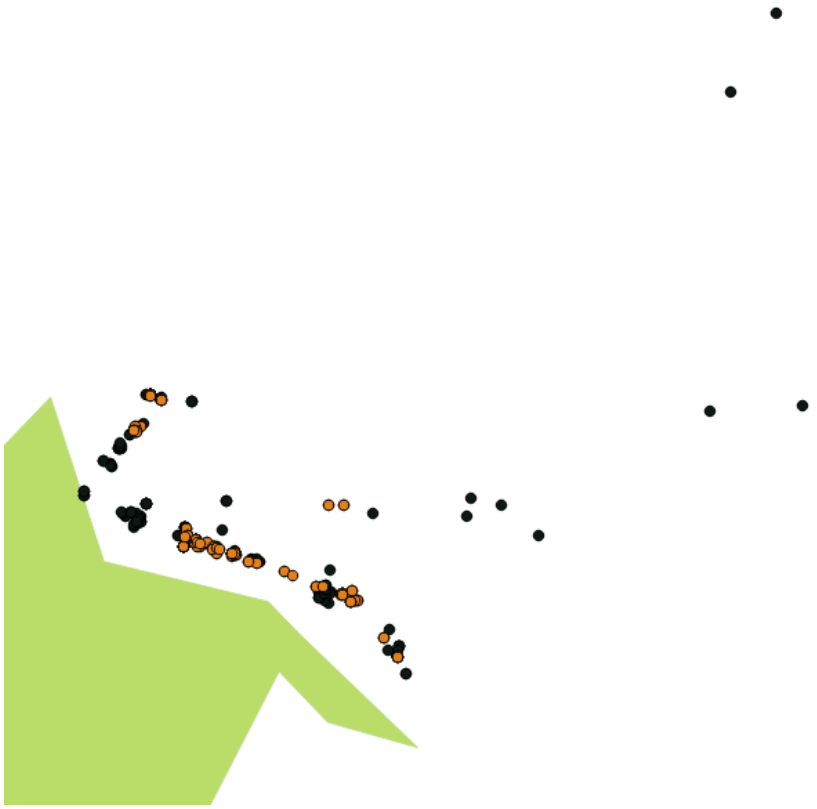

Figure 56:Vessels Rendezvous (black: grid of 152 m x 152m , yellow: grid 38,2 m x 19m) with zoom

Figure 57:Vessels Rendezvous with zoom for all different grid values

3.10. Time of execution

The monitoring of about 1000000 ais messages and the detection of complex events based on the patterns described on the Sections took more than 1h.

3.11. Running commands

First you need to run 3 servers: flink cluster server, zookeeper and kafka server. The next step is to create 4 topics on kafka. Also, there are 2 java projects that will run and one python script which will be used as producer for the main topic of the system (the one contains AIS messages).

The commands you need to execute are described below:

- ***$ flink_1.6.2:bin/start-cluster.sh***
  - start the cluster of flink
  - Check that server is running on http://localhost:8081/#/overview
- ***$ kafka2.2:bin/zookeeper-server-start.sh config/zookeeper.properties***
- ***$ kafka2.2:bin/kafka-server-start.sh config/server.properties***
  - Start the kafka and zookeeper servers
- ***$ kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 partitions 1 --topic AIS***
- ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic OUT_GAP***
- ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic OUT_COTRAVEL***
- ***$kafka2.2:bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic OUT_COURSE***
  - Create the topics on kafka
- ***sudo /home/cer/Desktop/flink-1.6.2/bin/flink run /home/cer/Desktop/cer/flinkcep/cep_flinkcep/target/flinkicu_cep-1.0-jar-with-dependencies.jar --IN_GAP GAP --bootstrap.servers localhost:9092 --zookeeper.connect localhost:2181 --IN_COTRAVEL COTRAVEL --IN_COURSE COURSE***
- ***sudo /home/cer/Desktop/flink-1.6.2/bin/flink run /home/cer/Desktop/cer/flinkcep/flinkcep/target/flinkicu-1.0-jar-with-dependencies.jar calhost:9092 --zookeeper.connect localhost:2181 --OUT_GAP GAP --OUT_COTRAVEL COTRAVEL --OUT_COURSE COURSE***
- ***$ Kafka2.2:bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 topic AIS***
  - You can check the events that detected at each job at this url. http://localhost:8081/#/overview
  - You can see what is send at each topic running the consumer of each topic. For example for topic AIS , running the consumer you will see the AIS messages
- ***./AIS .py AIS***
  - Run the python script in order to full fill the topic that contains all the AIS messages and 1st module starts receiving AIS messages
  - Producer is inside the project at the path ‘flinkcep/producer’

Running the script suspicious events will start to be detected. All of these will be written at the path *home/cer/Desktop/temp/* as .txt /.csv files. There will be a .txt /.csv file for each trajectory and suspicious event.

3.12. Repository

- The source code for the flink project is located at: https://github.com/salevizo/flinkcep.git

3.13. Source code of flink project

- 2 java maven projects: implement flink patterns for detecting trajectory or complex events
    - *Flinkcep*
    - *Cep_flinkcep*
- 1 python script *AIS .py* inside flinkcep that send AIS messages inside kafka producer
- 7 python scripts to convert the outcomes of the projects .txt files to csv in order to plot them on qgis, located inside folder */convert_txt_to_csv*.
- 1 folder */outcomes* which contains all the generated outcomes of 4 different executions of the project.
- Running the project, the outcomes will be saved at the path /home/cer /Desktop/temp
- README file
- The project is located at the path:
    - /home/cer/Desktop/cer_2/flinkcep/
- Kafka is located at the path:
    - /home/cer/Desktop/
- Flink is located at the path:
    - /home/cer/Desktop/

3.14. Running environment

OS	PRETTY_NAME="Ubuntu 18.04.1 LTS" VERSION_ID="18.04"
Postgres	postgres=# SELECT version(); PostgreSQL 10.5 (Ubuntu 10.5-0ubuntu0.18.04) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 7.3.0-16ubuntu3) 7.3.0, 64-bit
QGIS	2.18.0
Python	Python 2.7.15
Kafka	kafka_2.11-1.0.0
Flink	flink-1.6.2
Scala	Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/E
Prolog	YAP 6.2.2 (x86_64-linux):

3.14.1. Computer specifications

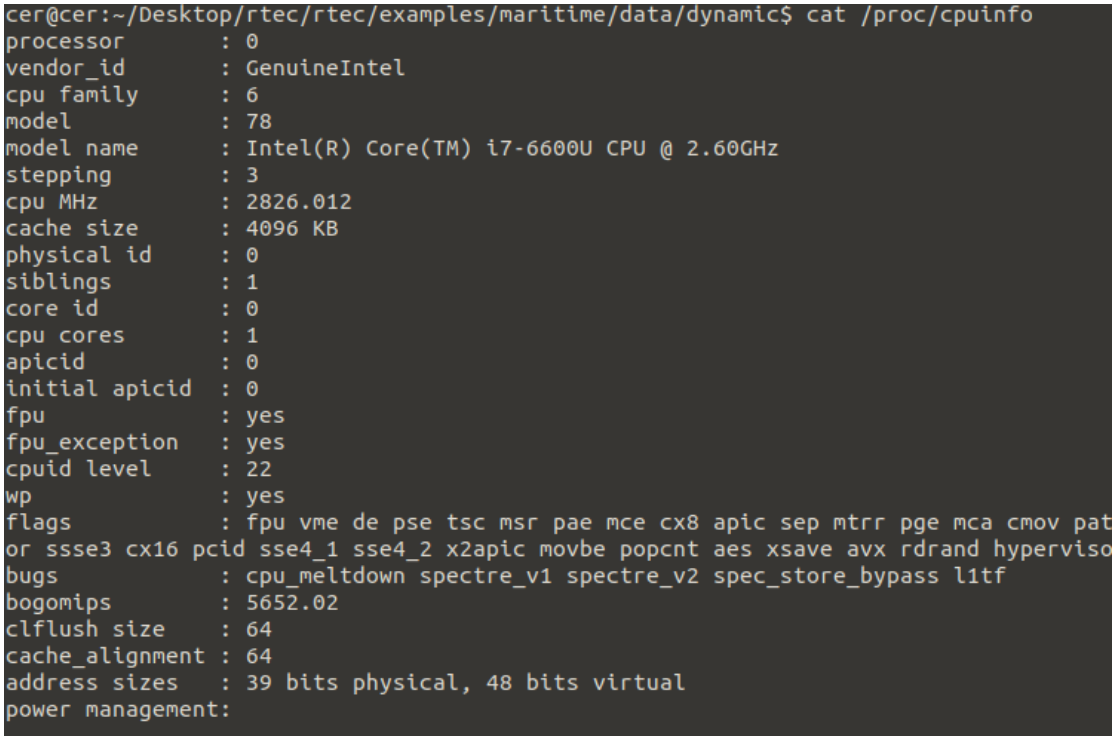

Figure 58: Cpu info

```
MemAvailable:    6185232 kB
Buffers:           63284 kB
Cached:           462744 kB
SwapCached:            0 kB
Active:           644748 kB
Inactive:         272044 kB
Active(anon):     394864 kB
Inactive(anon):    25956 kB
Active(file):     249884 kB
Inactive(file):   246088 kB
Unevictable:          32 kB
Mlocked:              32 kB
SwapTotal:       1134260 kB
SwapFree:        1134260 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        390824 kB
Mapped:           176672 kB
Shmem:             30060 kB
Slab:              60608 kB
SReclaimable:      36500 kB
SUnreclaim:        24108 kB
KernelStack:        5216 kB
PageTables:        25148 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     4596428 kB
Committed_AS:    2584964 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       72640 kB
DirectMap2M:     7053312 kB
```

Figure 59:Memory info

4. Logic-based CER with RTEC

4.1. Complex event detecting with RTEC

Since we have already implemented a system for online monitoring marine activity over streaming positions from numerous vessels sailing at sea, at that point we will present another framework of monitoring. RTEC [3] a logic-based engine written in Prolog. More specifically RTEC it's an event Calculus for Run-Time reasoning, designed to compute continuous queries on data streams. Event calculus represents the effects of actions on fluents. The predicate *HoldsAt* is used to tell which fluents hold at a given time point. Events are also represented as terms. The effects of events are given using the predicates *Initiates* and *Terminates*. An event won't be terminated alone, *Terminates* predicate should be call.

RTEC accepts as input a stream of time-stamped simple, derived events (SDE)s. A SDE is the result of applying a computational derivation process to some other event, such as an event coming from a sensor, in our case the ais messages from the vessels. An advantage of RTEC system is that provides Windowing, that supports real-time query computation. The system detects abnormal behavior using the critical events that which have been detected. As critical events assumed for example, the "stop" or the "pause" of a vessel.

4.1.1. System Architecture

The online monitoring system uses the prolog patterns which have been compiled and an input file which contains critical events, critical events are the trajectory events we have explained on Sec. 3.2. in order to detect complex events. The output of the system is written on a txt file. There are declarations files which contain information about all the events and fluents of the scenarios we would like to detect.

There is a function 'continuousER' that executes all the prolog patterns on the given critical events. The time period of the events that we would like to detect should be declared on this function as long as the window size. We run the monitoring systems for window size 600secs in order to do the same executions as we have already done on flinkcep.

4.2. Trajectory events

We have analyzed on the previous Sections (see Sec. 3.2) what is a trajectory events. At this system, trajectory events are written on a txt file and will be given as input on the monitoring system.

A sample critical event is shown here. For example, “velocity” event means that a vessel change its speed with a suspicious manner.

Event	timestamp_ start	timestamp_end	mmsi	other information based on the event
Velocity	1443677750	1443677750	227008170	Xxx|xxx|xxx

As we would like to compare the number of detected events between the two monitoring systems we have written a python script that filters the timestamp of critical events. The file final_preprocessed_dataset_RTEC_critical_nd.csv which located at the path /home/cer/Desktop/rtec/rtec/examples/maritime/data/dynamic/ will contain all the critical events related to the time period ['1443650402','1444660250'] (same duration has been used for the ais message on flinkcep) and will be given as input on the system.

Critical events are described on the declarations files. On declaration files are declared the entities of the event description: events, simple and statically determined fluents.

4.3. Complex event

Complex event Recognition module consumes the input file with critical events in order to detect complex events of maritime activities. The complex event patterns are located at the path RTEC/examples/maritime/CE\ patterns/patterns.prolog. In order to run them firstly you should compile them.

The implemented patterns are listed below:

- gap
- stopped
- changingSpeed
- withinArea
- underWay
- highSpeedNearCoast
- anchored or moored
- Movement ability affected
- adrift
- rendezVous
- fishing
- lowspeed

Some of the above patterns for complex events have also been implemented and run in flinkcep so we can compare their results and highlight the differences between the 2 monitoring systems, whereas for some of them there is no special pattern on flinkcep as their functionality is included in other patterns. These patterns cannot be compared.

Looking the patterns on flink (Java patterns) and the patterns on RTEC (prolog patterns) we can see that the source code for RTEC patterns are less complex and more user friendly. Their format is like a pseudocode and it is easy to be maintained. Each pattern has an "Initiated" and one or more "Terminated" predicates. For example, a vessel detected as "stopped" with the following pattern:

```
InitiatedAt(stopped(Vessel)=Status,T):-
        happensAt(coord(Vessel,Lon,Lat),T),
        portDistance(Lon,Lat,Status).
terminatedAt(stopped(Vessel) = Status, T) :-
        happensAt(stop_end(Vessel), T).
terminatedAt(stopped(Vessel) = _Status, T) :-
         happensAt(gap_init(Vessel), T).
```

Below are described the patterns running in RTEC monitoring system. For each detected event will be presented the coordinates at which this event took place. We will use them to visualize the results on QGIS. Coordinates found after searching on psql database for the specific timestamp.

1. Stopped & lowspeed

This pattern has not been implemented as special pattern on flinkcep as we check for stopped vessels inside other patterns. For example: in the Long stop pattern "stopped" is included., see Sec 3.2.5

2. WithinArea

WithinArea pattern on flinkcep is implemented via the grid partioning, see Sec 3.1.4. Here in RTEC "within area" is a pattern which detects that a vessel is inside an area for a time period. Similarly, there is no standalone pattern on flinkcep for this.

3. Underway & anchored or moored

On flinkcep these patterns are included in other patterns by checking if the speed of a vessel is bigger than 5KNOTS -under way or smaller than 1 KNOT- anchored or moored. For example, see Sec 3.2.6 which detects vessels with false status. There aren't special patterns for these.

4. ChangingSpeed

ChangingSpeed pattern hasn't been implemented as a standalone pattern on flinkcep, but inside other patterns like: HighspeednearPort, see Se 3.2.4 there is a statement which check if a vessel has changed its speed rapidly.

5. HighSpeedNearCoast

The highSpeedNearCoast pattern is implemented somehow different on flinkcep. On flinkcep, we have written a pattern which detect vessels near port, more specific than detect vessels near cost.

6. Gap in communication

This pattern is implemented at both systems with the same way .

7. Adrift

This pattern is implemented at both systems. Nevertheless, on flinkcep we have used different approach about when a vessel seems to be in adrift, so we have changed the RTEC prolog pattern according our implementation on flinkcep in order to have a fair comparison.

8. Rendezvous , Fishing

These patterns are implemented at both systems with the same way. The results of 2 systems will be compared.

4.4. Outcomes of the system

Running RTEC system for detecting suspicious events we can see that we don't have exactly the same results with the previous monitoring system, Flink even though we used the same input data. It should be noticed that patterns between flinkcep and RTEC have not exactly the same context.

The below results came after running the monitoring system for time period of ['1443650402','1444660250'].

4.4.1. Running patterns for different window size

Detected event	300 secs	600 secs
gap	82003	41508
stopped	25064	25299
changingSpeed	6456	7281
atAnchorOrMoored	3821	3726
fishing	2969	5625
adrift	2367	5103
speedLTMin	34589	21447
highSpeedNearCoast	168	252
rendezVous	2	45

It is obvious that with bigger window size less trajecory events are detected as the patterns become stricter, eg: gap event. On the other hand, with bigger window size the detected events come from complex events whose duration should be bigger are more. Looking for example on rendezvous complex event, using 300 secs as window size only 2 events are detected whereas when window size increased from 300 secs to 600 secs, 45 evets are detected. Meaning that inside 300 secs the "terminates" predicated hasn't happened.

The results from several running are located at the path:/results in the folders results_win300 and results_win600 (window size 600 and 300 secs).

4.4.2. Ground truth for detected events with window size: 600 secs

We have used as window size 600 secs in order to be the same with flinkcep patterns. From the detected events on flinkcep project, we have concluded on that window size of 600 secs detects events that correspond to the ground truth.

1. *High Speed near port / coast*

Monitoring System	Detected events
FlinkCep – near port	41
RTEC – near coast	252

2. *Rendezvous*

Monitoring System	Detected events
FlinkCep	58
RTEC	45

Comparing the outcomes of the monitoring system we see that they have some differences.

4.5. Visualization of the detected events

The below pictures come from QGIS framework using the results from RTEC. These visualizations represent the detected events for the patterns which describes in previous chapter, see SEC 4.3.. We searched on psql database at which we have stored AIS messages and searched latitude and longitude coordinates for each timestamp in combination with vessel mmsi. We have also used as window size 600 secs in order to be the same with flinkcep patterns.

1. High Speed Near Coast

The blue points are results from RTEC (252 detected events) and the red from flinkcep (41 detecetd events). As ports is a subgroup of coast, the detected events "near coast" are more than events "near port", as it was expected.

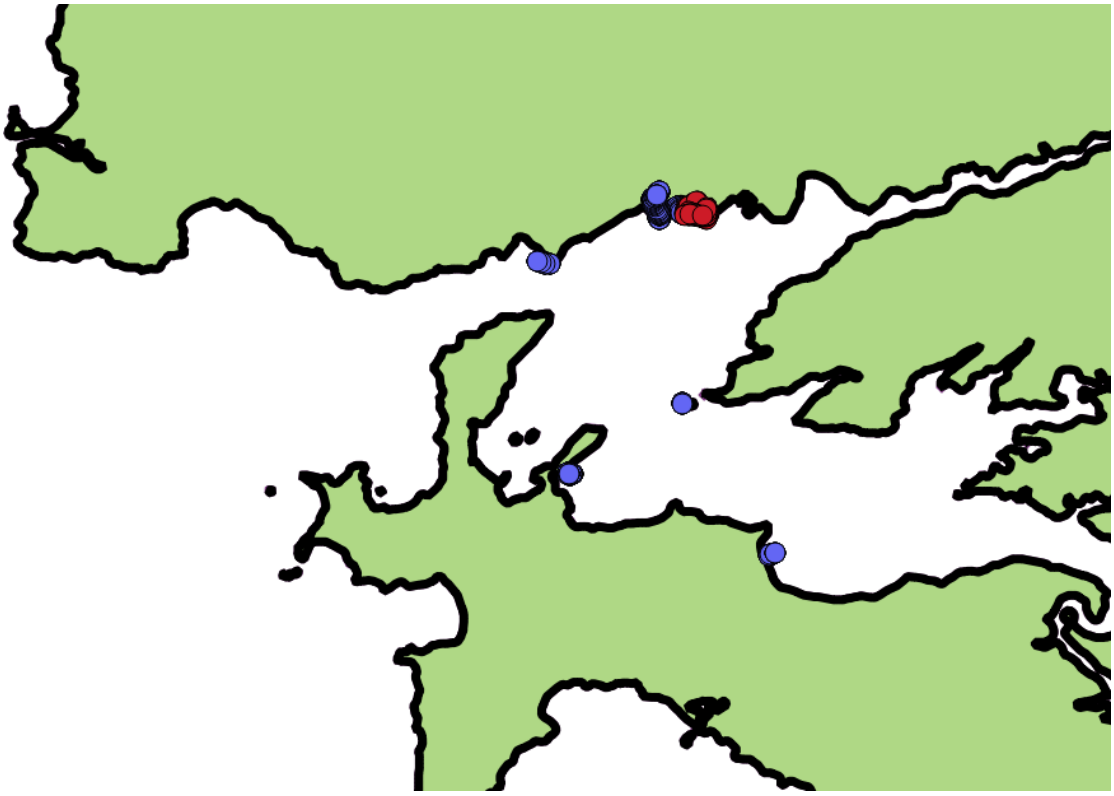

Figure 60: High speed near coast (blue: RTEC, red :flink)

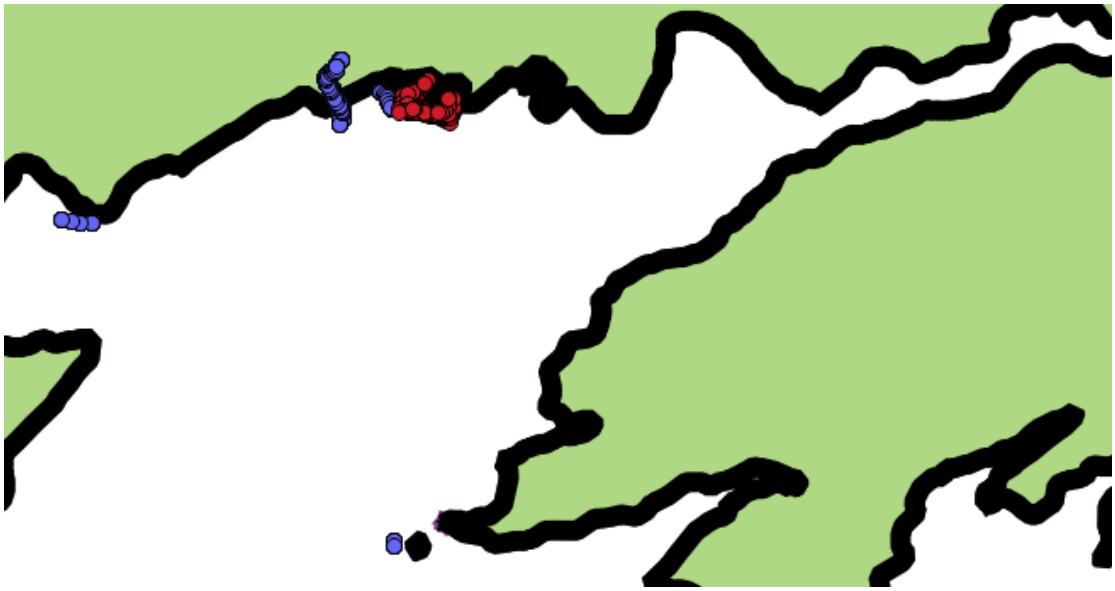

Figure 61: High speed near coast in zoom (blue: RTEC, red :flink)

2. rendezVous

The blue points are results from RTEC (45 detecetd events) and the red from flinkcep (58 detected events). We compared RTEC results with the pattern running on flinkcep for index 8 (grid of *38.2m x 19m).* Flincep detects more events because of the distance rule it uses.

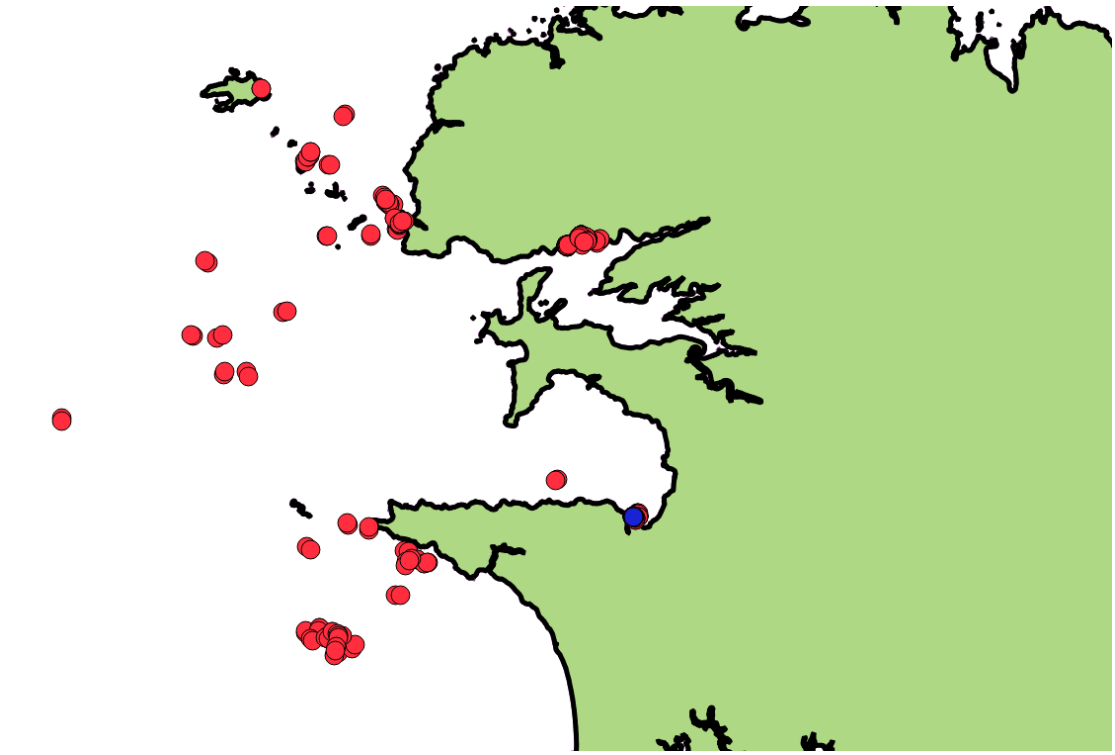

Figure 62: Vessels Rendezvous (blue: RTEC, red :flink)

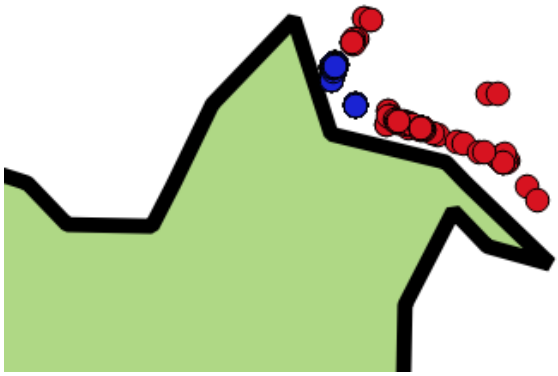

Figure 63:Vessels Rendezvous in zoom (blue: RTEC, red: flink)

3. Adrift

Below is displayed a plot with all adrift detected events on RTEC monitoring system.

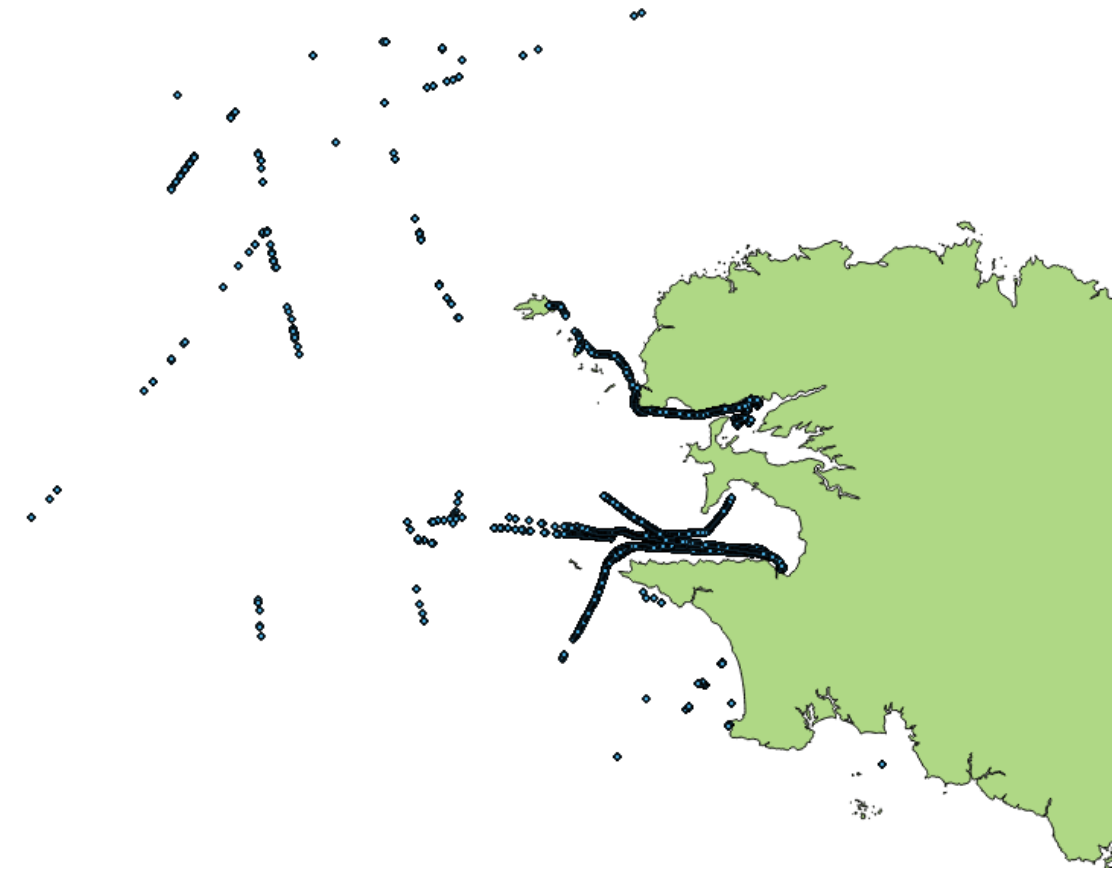

Figure 64: Adrift from RTEC

The involved mmsi at the next plot is 228064900. The blue points are results from RTEC and the red from flinkcep.

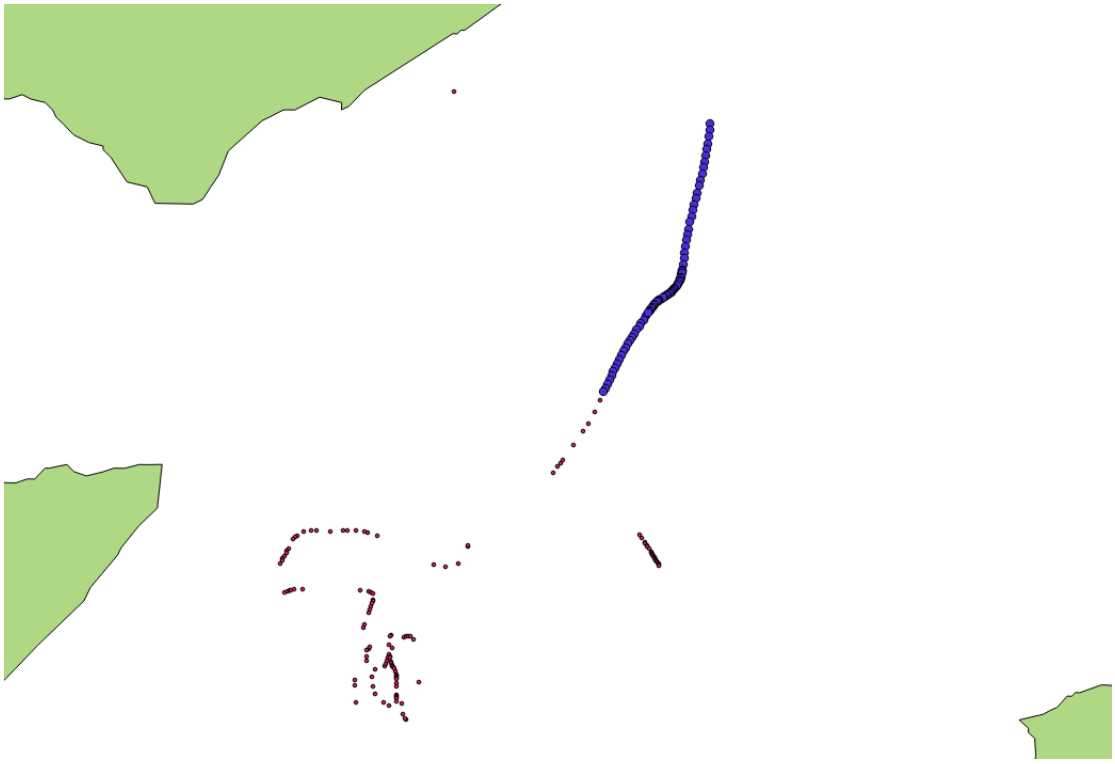

Figure 65:Adrift activity for mmsi 228064900 (blue: RTEC, red: flink)

The results from the 2 monitoring systems for the same mmsi are different enough.

4.6. Time of execution

The monitoring of about 1000000 critical events and the detection of complex events based on the patterns described on the Sections took less than 1 h. Faster running than flinkcep monitoring system.

4.7. Running commands

In order to execute the prolog patterns, they should be compiled first. Below are listed the commands you must execute.

- **Go at RTEC/examples/maritime/CE\ patterns/**
- **$yap**
    - open YAP prolog
- **?- ['../../../src/RTEC.prolog'].**
    - Load the main file of RTEC.
- **?- compileEventDescription('declarations.prolog','patterns.prolog','compiled_patterns.prolog').**
- **Go at RTEC/examples/maritime/execution\ scripts/**
- **$yap**
- **?- ['continuousQueries.prolog'].**
- **?- ['../CE\ patterns/declarations.prolog'].**
- **?- ['../CE\ patterns/compiled_patterns.prolog'].**
    - load the declarations, and the compiled patterns...
- **?- ['../data/static/loadStaticData.prolog'].**
- **?- ['../utils/loadUtils.prolog'].**
    - load static data, and some utils, both necessary for recognition.
- **%continuousER(DatasetFile,RecognisedCEsFile,RecognitionTimesFile,InputStatisticsFile,StartTime,EndTime,Step,Window)**
    - continuousER('../data/dynamic/final_preprocessed_dataset_RTEC_critical_nd.csv','../results/maritime_results.txt','stats_times.txt','stats_input.txt',1443650402,1444660250,x,x).
    - Call 'continuousER' with the appropriate parameters to perform recognition for a specific time period and window size.

4.8. Repository

- The source code for the RTEC project is located at: https://github.com/salevizo/rtec.git

4.9. Running environment

4.9.1. Computer specifications

Running on the same computer which we run the flink monitoring system. For the specifications see Sec 3.14.1.

References

[1] Online Event Recognition from Moving Vessel Trajectories Kostas Patroumpas · Elias Alevizos · Alexander Artikis · Marios Vodas · Nikos Pelekis · Yannis Theodoridis.

[2] COMPARISON OF INS HEADING AND GPS COG, R. Michael Reynolds November 6, 20

http://www.rmrco.com/docs/m1227_Compare-cog-hdg.pdf

[3] https://github.com/aartikis/RTEC