RVG DSS Manual

Luis Fernando Sanchez Martin [1]

July 2023

[1]luissanchezmtn@gmail.com

Contents

Chapter 1

Introduction

This document is meant to serve as a technical manual on the modules, functionality, and operation of the RVG DSS Repo/ Application

This project is meant to be a Decision Support System for Research Vessel Gunnerus. It receives real-time data from RVG's onboard computer containing NMEA and AIS data, processes it, and provides a visual output as an Electronic Chart (EC) accessible through a web application. Additionally, a server-like application is used, in order to make this web application remotely available.

On top of this, the project includes modules to allow replay of previously obtained data. run 4DOF simulations based on RVG models, and collision avoidance algorithms. These functionalities shall be further expanded in the following sections.

A monolithic repository containing the project can be found here,

1.1 Overview

The project is divided into three main components, the `rvg_leidarstein_frontend`, the `rvg_leidarstein_msg_relay`, and the `rvg_leidarstein_core`. This last one is the most extensive of the three.

A short breakdown of the functionality of these components is as follows:

- `rvg_leidarstein_frontend`: this is the front-end implementation, which provides visual information related to RVG, This receives data via WebSocket through the `rvg_leidarstein_msg_relay` and displays it on a digital electronic chart. This web application is based on React.js.

- `rvg_leidarstein_msg_relay`: This functions as a bridge that allows communicating the output of the `rvg_leidarstein_core` to the `rvg_leidarstein_frontend`. Although establishing direct communication between the `rvg_leidarstein_core` and the `rvg_leidarstein_frontend` is possible; the `rvg_leidarstein_msg_relay` allows for remote access to the data in both directions. This implementation should be revised.

- `rvg_leidarstein_core`: this is the most extensive component in the stack. this handles the communication between RVG and the Application, the processing of the raw messages into usable data, the execution of the DSS algorithms, and finally the relay of pertinent data to the `rvg_leidarstein_frontend`. This module is due to be refactored in order to improve its functionality.

The following sections will present each of the main components, beginning with the installation and execution steps.

1.2 Quickstart

- install the necessary dependencies via conda or any other method. these can be found in the provided .yml or text files and if necessary activate the conda environment

In the `RVG_DSS` directory:

```
conda env create -f rvgdss.yml
conda activate rvgdss
```

In addition to these dependencies, there are two more that need to be installed locally, these are the `model4dof` and `rvg_ledarstein_core`. In order to install `model4dof` download the repository then `pip install` the `.tar.gz` file contained in the dist directory.

```
pip install .\model4dof-0.0.1.tar.gz
```

Next, go to the `rvg_ledarstein_core` directory and build and install the package there. `cd` into the `rvg_ledarstein_core` directory and:

```
python -m build
pip install .\dist\rvg_dss-0.0.1.tar.gz
```

Note that this step should be repeated when changes are performed in the code contained in this directory

- start the `rvg_leidarstein_msg_relay`

`cd` into the `rvg_leidarstein_msg_relay` directory and:

```
python .\rvg_leidarstein_msg_relay.py
```

- launch the webapp `rvg_leidarstein_frontend`

In a new terminal, `cd` into the `rvg_leidarstein_frontend` directory and:

```
npm install
npm start
```

- launch the `run_leidarstein.py` script

In a new terminal `cd` into `\RVG_DSS\rvg_leidarstein_core\scripts` and:

```
python .\run_leidarstein.py
```

Chapter 2

Frontend

`rvg_leidarstein_frontend`

2.1 Installation and execution

Node is required to run this component: `cd` into the `rvg_leidarstein_frontend` directory and

```
npm install
npm start
```

after the first install, only the `npm start` command is necessary for its execution.

This should launch the react web application in a browser 2.1, although it will do nothing unless the `rvg_leidarstein_msg_relay` and the `rvg_leidarstein_core` components are also running.

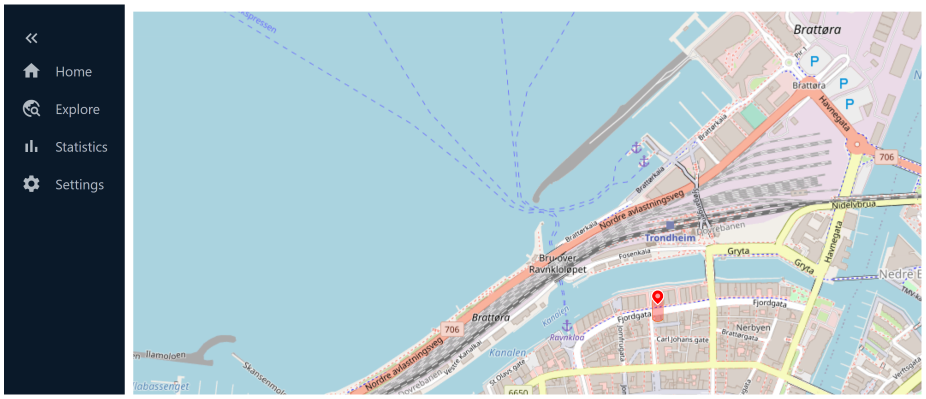

Figure 2.1: App running in browser

Chapter 3

Message Relay

`rvg_leidarstein_msg_relay`

3.1 Installation and Execution

`cd` into the `rvg_leidarstein_msg_relay` directory and

```
python .\scripts\run_leidarstein.py
```

Following this, the connections made to the server will be displayed in the same console.

Chapter 4

Core

`rvg_leidarstein_core`

4.1 Installation and Execution

Although installation in and of itself is not required for running this component, it is necessary to have the Python dependencies required for this project. At the time of writing this manual, it is suggested to use the conda environment manager to install the required packages by creating an environment from the provided `rvgdss.yml` file. However, a python `venv` solution is being worked on. At any rate the packages described in the yml file are required in addition to the packages mentioned in the quickstart guide.

Once these dependencies are met, the component can be executed by

```
python .\scripts\run_leidarstein.py
```

Alternatively, a log containing raw data can be passed as an argument for the component to use as a data source by using `-f`

```
python .\scripts\run_leidarstein.py -f path/to/file
```

Upon execution, the following messages will be displayed in the console indicating that it is running:

```
StreamParser running.
DataLogger running.
Simulation 4DOF Client running...
Colav Manager running...
```

4.2 Description

The entry point for this component is the `rvg_leidarstein_frontend` script. This script contains the main execution loop and instantiates the main components of the `rvg_leidarstein_core`. The `ColavManager` component is also instantiated here since it is included in the main loop in order to run it as an individual process, as `cbf_process` requires additional resources to run in a timely fashion. Colav Manager aside, most of the functionality of this application is wrapped inside of the `DataModel` component itself. It is worth mentioning that some of the naming conventions in this app are holdovers from the development of the project and could now use different names since they have diverted from its original intent.

The `core.py` component encapsulates the instances of the four main components required for the execution of the program. These components run concurrently in separate threads, namely:

- `thread_websocket_receive`: receives and stores feedback messages coming from the frontend web application.
- `thread_datastream`: receives data from RVG UDP stream, otherwise servers as the data source for RVG data. Performs decoding and decrypting operations on the received data.
- `thread_serialize_data`: serializes the messages into data structures usable by the `simulation_manager` [1]
- `thread_sim_manager`: Takes the parsed data structures originating from the aforementioned thread and performs some filtering operations before sending the data via WebSockets. Alternately RVG data can originate from here via the using the 4dof simulation model.

Additionally, the `colav_manager` is executed in a separate process in `run_leidarstein.py` in order to generate the DSS data.

Message Reference

The following section will contain a description of the messages being received ad transmitted by the components executing the threads and processes described in the previous section.

4.3 Websocket Message Reference

The WebSocket sends messages when prompted by other modules. This component also receives messages from the frontend, these are the feedback for the `core` module.

Message for simulation type:

```
{
  type: "datain",
  content: {
```

[1] This part of the process could be revised.

```
    message_id: "data_mode",
    val: "value",
  },
}
```

Inputs for 4DOF simulation:

```
{
  type: "datain",
  content: {
    message_id: "control_azi",
    val: value,
  },
};

{
  type: "datain",
  content: {
    message_id: "control_thrust",
    val: value,
  },
};
```

4.4 Datastream Manager Message Reference

Input

The input of the `datastream_manager` can be any of the strings described in the appendix A.

Output

the output of this component is a list containing the objects as parsed by the `pyais` and `pynmea2` libraries. An example of a AIS object is

```
MessageType1(
msg_type=1,
repeat=0,
mmsi=257131220,
status=
<NavigationStatus.Undefined: 15>,
turn=<TurnRate.NO_TI_DEFAULT: -128.0>,
speed=0.0,
accuracy=True,
lon=10.361998,
lat=63.432855,
course=360.0,
heading=511,
second=36,
maneuver=<ManeuverIndicator.NotAvailable: 0>,
spare_1=b'\x00',
raim=True,
radio=66921
)
```

The NMEA object depends on the kind of message that is being received and it contains the relevant attributes and values. An example of a NMEA object for a GPRMC message is:

```
GPRMC(
timestamp=datetime.time(16, 23, 17, 500000, tzinfo=datetime.timezone.utc)
status='A',
lat='6326.3036',
lat_dir='N',
lon='01024.5386',
lon_dir='E',
spd_over_grnd=0.0,
true_course=195.5,
datestamp=datetime.date(2023, 8, 6),
mag_variation='4.7',
mag_var_dir='E',
mode_indicator='A',
nav_status='S'
)
```

4.5 Serializer Message Reference

Input

The serializer takes as input the list of objects provided by the `datastream_manager`

Output

The output of the `fast_serializer` is a list composed of the serialized NMEA and AIS messages (dictionaries). The following messages can be expected to be in the output list of the component:

AIS message:

```
{
'msg_type': 1,
'repeat': 0,
'mmsi': 258342000,
'status': <NavigationStatus.Moored: 5>,
'turn': 0.0,
'speed': 0.0,
'accuracy': True,
'lon': 10.408968,
'lat': 63.453595,
'course': 335.2,
'heading': 449,
'second': 0,
'maneuver': <ManeuverIndicator.NotAvailable: 0>,
'spare_1': b'\x00',
'raim': False,
'radio': 0,
'unix_time': 1691340458.702,
'seq_num': 11449976,
'src_id': 2,
'src_name': '@10.0.8.10:35481',
'message_id': '!AIVDO_ext_258342000'
}
```

NMEA GPRMC message:

```
{
'timestamp': datetime.time(16, 47, 37, 500000,...),
'status': 'A',
'lat': '6326.3037',
'lat_dir': 'N',
'lon': '01024.5381',
'lon_dir': 'E',
'spd_over_grnd': 0.0,
'true_course': 190.8,
'datestamp': datetime.date(2023, 8, 6),
'mag_variation': '4.7',
'mag_var_dir': 'E',
'mode_indicator': 'A',
'nav_status': 'S',
'unix_time': 1691340457.962,
'seq_num': 11449967,
'src_id': 1,
'src_name': 'a10.0.8.1',
'message_id': '$GPRMC_ext'
}
```

NMEA PSIMSNS message:

```
{
'msg_type': 'SNS',
'timestamp': '164742.292',
'unknown_1': '',
'tcvr_num': '1',
'tdcr_num': '1',
'roll_deg': '0.28',
'pitch_deg': '-0.82',
'heave_m': '0.01',
'head_deg': '199.44',
'empty_1': '',
'unknown_2': '40',
'unknown_3': '0.001',
'empty_2': '',
'checksum': 'M121',
'unix_time': 1691340458.27,
'seq_num': 11449970,
'src_id': 3,
'src_name': '@10.0.8.10:39816',
'message_id': '$PSIMSNS_ext'
}
```

NMEA GPGGA message:

```
{
'timestamp': datetime.time(16, 47, 37, 500000,...),
'lat': '6326.3037',
'lat_dir': 'N',
'lon': '01024.5381',
'lon_dir': 'E',
'gps_qual': 1,
'num_sats': '11',
'horizontal_dil': '0.7',
'altitude': 14.2,
'altitude_units': 'M',
'geo_sep': '41.4',
'geo_sep_units': 'M',
'age_gps_data': '',
'ref_station_id': '',
'unix_time': 1691340458.274,
'seq_num': 11449971,
'src_id': 1,
'src_name': '@10.0.8.10:42796',
'message_id': '$GPGGA_ext'
}
```

It is worth noting that the following attributes are appended to the messages as part of the encrypted data being relayed by the Olex computer:

- `'unix_time'`

- 'seq_num'
- 'src_id'
- 'src_name'

4.6 Simulation Manager Message Reference

Input

The input for the `simulation_manager` is the list of dictionaries provided by the `fast_serializer`

Output

The output of this component is strings of JSON messages for the WebSocket to relay. Additionally, this component keeps track of the latest data for each type of message and updates the Colav Manager attributes.

AIS message example:

```
{
"type": "datain",
"content": {
    "msg_type": 1,
    "repeat": 0,
    "mmsi": 258342000,
    "status": 5,
    "turn": 0.0,
    "speed": 0.0,
    "accuracy": true,
    "lon": 10.408952,
    "lat": 63.45423,
    "course": 72.8,
    "heading": 262,
    "second": 1,
    "maneuver": 0,
    "spare_1": "b'\\x00'",
    "raim": false,
    "radio": 0,
    "unix_time": 1691353146.021,
    "seq_num": 11595825,
    "src_id": 2,
    "src_name": "@10.0.8.10:35481",
    "message_id": "!AIVDO_ext_258342000",
    "pos_history": [[10.408952, 63.454228], (...), [10.408952, 63.45423]]
    }
}
```

NMEA PSIMSNS message:

```
{
"type": "datain",
"content": {
    "msg_type": "SNS",
    "timestamp": "22:19:07.837150",
    "unknown_1": "",
    "tcvr_num": "1",
    "tdcr_num": "1",
    "roll_deg": 0.0,
    "pitch_deg": 0,
    "heave_m": "0.00",
    "head_deg": -40.0,
    "empty_1": "",
    "unknown_2": "40",
    "unknown_3": "0.000",
    "empty_2": "",
    "checksum": "M121",
    "unix_time": 1691353147.8371496,
    "seq_num": 39,
    "src_id": 1,
    "src_name": "@10.0.8.10:39816",
    "message_id": "$PSIMSNS_ext"
    }
}
```

NMEA GPGGA message:

```
{
"type": "datain",
"content": {
    "timestamp": "22:19:07.837150",
    "lat": 6326.305279636112,
    "lat_dir": "N",
    "lon": 1024.5376641753296,
    "lon_dir": "E",
    "gps_qual": 1,
    "num_sats": "10",
    "horizontal_dil": "1.0",
    "altitude": 12.6,
    "altitude_units": "M",
    "geo_sep": "41.4",
    "geo_sep_units": "M",
    "age_gps_data": "",
    "ref_station_id": "",
    "unix_time": 1691353147.8371496,
    "seq_num": 40,
    "src_id": 3,
    "src_name": "@10.0.8.10:34340",
    "message_id": "$GPGGA_ext"
    }
}
```

NMEA GPRMC message:

```
{
"type": "datain",
"content": {
    "timestamp": "22:19:07.237150",
    "status": "A",
    "lat": 6326.305135176538,
    "lat_dir": "N",
    "lon": 1024.5379348907738,
    "lon_dir": "E",
    "spd_over_grnd": 1.0929829421120387,
    "true_course": -40.0,
    "datestamp": "2023-08-06",
    "mag_variation": "4.7",
    "mag_var_dir": "E",
    "unknown_0": "A",
    "unknown_1": "S",
    "unix_time": 1691353147.2371497,
    "seq_num": 38,
    "src_id": 3,
    "src_name": "a10.0.8.1",
    "message_id": "$GPRMC_ext"
    }
}
```

4.7 Colav Manager Message Reference

The `colav_manager` consists of two main components, the `arpa` component for obtaining ARPA parameters and the `cbf` component for obtaining the information related to Control Barrier Functions.

Input

This component acquires data by storing NMEA and AIS data as attributes (`self._ais_data[mmsi] = data`, `self._gunnerus_data = data`). these attributes are updated when new data is received by the Simulation Manager.

Example of AIS data stored in the colav manager:

```
self._ais_data[258342000] = {
    'msg_type': 1,
    'repeat': 0,
    'mmsi': 258342000,
    'status': <NavigationStatus.Moored: 5>,
    'turn': 0.0,
    'speed': 0.0,
    'accuracy': True,
    'lon': 10.40896,
    'lat': 63.453905,
    'course': 79.2,
    'heading': 266,
    'second': 1,
    'maneuver': <ManeuverIndicator.NotAvailable: 0>,
    'spare_1': b'\x00',
    'raim': False,
    'radio': 0,
    'unix_time': 1691355550.095,
    'seq_num': 11622741,
    'src_id': 2,
    'src_name': '@10.0.8.10:35481',
    'message_id': '!AIVDO_ext_258342000',
    'pos_history': [[10.408958, 63.453898], (...), [10.40896, 63.453905]]
    }
```

Similarly, the following data is stored for Gunnerus:

```
self._gunnerus_data = {
    'timestamp': datetime.time(22, 59, 9, 616093),
    'status': 'A',
    'lat': 6326.305969691541,
    'lat_dir': 'N',
    'lon': 1024.5363710191984,
    'lon_dir': 'E',
    'spd_over_grnd': 1.5323130672165104,
    'true_course': -40.0,
    'datestamp': datetime.date(2023, 8, 6),
    'mag_variation': '4.7',
    'mag_var_dir': 'E',
    'unknown_0': 'A',
    'unknown_1': 'S',
    'unix_time': 1691355549.6160932,
    'seq_num': 53,
    'src_id': 3,
    'src_name': 'a10.0.8.1',
    'message_id': '$GPRMC_ext'
    }
```

This data is used by the ARPA component.

Output

The output of the Colav Manager is the information relayed via WebSockets by the ARPA and CBF components.

4.7.1 ARPA

ARPA Input

This component takes as input the data stored in the aforementioned attributes.

ARPA Output

The `arpa` component produces two main outputs, one made from the resulting calculations converted to geodetic coordinates that are relayed via websockets to the frontend:

```
{
"type": "datain",
"content": {
    "message_id": "arpa",
        "data": {
            "2570221": {
                "self_course": -40.0,         #rvg course
                "course": 0,                  #targer ship course
                "t_2_cpa": 558.8694955229263, #time to cpa
                "lat_o": 63.442017999990306,  #origin lat of target vessel
                "lon_o": 10.403963000013439,  #origin lon of target vessel
                "uo": 0.0,                    #speed of target vessel
                "zo": "[[0.]\n [1.]]",        #orientation of target vessel
                "d_at_cpa": 66.63601614808934, #distance at cpa
                "d_2_cpa": 465.2980767097945,   #distance to cpa
                "lat_at_cpa": 63.4416337020036, #target vessel lat at cpa
                "lon_at_cpa": 10.402940216628501, #target vessel lon at cpa
                "lat_o_at_cpa": 63.442017999990306, #rvg lat at cpa
                "lon_o_at_cpa": 10.403963000013439, #rvg lon at cpa
                "safety_params": true, #true if rvg will get closer than saf rad at cpa
                "t_2_r": 332.37490497999767, #time to safety radius
                "lat_o_at_r": 63.442017999990306, #rvg lat at safety rad
                "lon_o_at_r": 10.403963000013439, #rvg lon at safety rad
                "lat_at_r": 63.440337857723435,   #target vessel lon at safety rad
                "lon_at_r": 10.405369196577027,   #target vessel lon at safety rad
                "d_2_r": 276.7254346009467,       # distance before the safety radius
                "safety_radius": 200
                        }, (...)
            "2572222": {
                "self_course": -40.0,
                "course": 0,
                "t_2_cpa": 767.8156155573266,
                "lat_o": 63.44410799996819,
                "lon_o": 10.404103000027073,
                "uo": 0.0,
                "zo": "[[0.]\n [1.]]",
                "d_at_cpa": 221.75117238015486,
                "d_2_cpa": 639.2603855615433,
                "lat_at_cpa": 63.442829109372276,
                "lon_at_cpa": 10.400699235848839,
                "lat_o_at_cpa": 63.44410799996819,
                "lon_o_at_cpa": 10.404103000027073,
                "safety_params": false
                    },
                }
            }
}
```

And the data still in NED frame used for the CBF component for gunnerus:

```
{
'p': array([[0],[0]]), #position (located at origin)
'u': 0.7882831342988617, #speed
'ux': -0.5066986316521784, #speed x component
'uy': 0.6038599146340541,  #speed y component
'z': array([[-0.64278761], [ 0.76604444]]), #rvg orientation
'tq': array([[-0.64278761],[ 0.76604444]]), #rvg target orientation
'lon': 10.40893951698664,
'lat': 63.43843282819234,
'course': -40.0
}
```

And the data in NED frame used for the CBF component for AIS vessels:

```
[
    { #target vessel 1
    'po_x': -248.4036616045414, #starting x position
    'po_y': 399.5323145183838,  #starting y position
    'uo': 0.0,                  #speed
    'zo': array([[0.],[1.]]),   #orientation
    'uo_x': 0.0,                #speed x component
    'uo_y': 0.0,                #speed y component
    'course': 0,
    'mmsi': 2570221,
    'cpa': {
```

```
    'd_at_cpa': 66.5261768192365,    # distance at cpa
    'd_2_cpa': 465.73030526343894,   # distance to cpa
    't_2_cpa': 590.8160215525639,    # time to cpa
    'x_at_cpa': -299.36566967886813, # target vessels x pos at cpa
    'y_at_cpa': 356.7701123391627,   # target vessels y pos at cpa
    'o_x_at_cpa': -248.4036616045414,# rvg x pos at cpa
    'o_y_at_cpa': 399.5323145183838  # rvg y pos at cpa
    },
'safety_params': {
    't_2_r': 351.54739850532616,   # time to safety radius
    't_x_at_r': -248.4036616045414,# target vessel x at safety rad
    't_y_at_r': 399.5323145183838, # target vessel y at safety rad
    'x_at_r': -178.12858578353183, # rvg x at safety rad
    'y_at_r': 212.28538205125005,  # rvg y at safety rad
    'd_2_r': 277.11888514838944}   # distance to safety radius
    },
{ #target vessel n
'po_x': -241.34943827321155,
'po_y': (...)
}
]
```

4.8 CBF

CBF Input

The input for the CBF is the NED data provided by the ARPA component as mentioned in the previous subsection.

CBF Output

The output of the CBF component is a JSON message containing the geodetic points required to draw the resulting trajectory.

```
{
"type": "datain",
"content": {
    "message_id": "cbf",
    "data": {
        "cbf": [[10.408939516986642, 63.43843282819234], (...),
                [10.408939516986642, 63.43843282819234]]
            }
        }
}
```

Appendix A

Received UDP Messages

A.1 AIS messages from Gunnerus

Class A AIS Position Report (Messages 1, 2, and 3)

1. Message ID: Identifier for this message 1, 2 or 3
2. Repeat indicator: Used by the repeater to indicate how many times a message has been repeated. See Section 4.6.1, Annex 2; 0-3; 0 = default; 3 = do not repeat any more.
3. User ID: MMSI number
4. Navigational status:
   (a) 0 = under way using engine
   (b) 1 = at anchor
   (c) 2 = not under command
   (d) 3 = restricted maneuverability
   (e) 4 = constrained by her draught
   (f) 5 = moored
   (g) 6 = aground
   (h) 7 = engaged in fishing
   (i) 8 = under way sailing
   (j) 9 = 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)
   (k) 10 = reserved for future amendment of navigational status for ships carrying dangerous goods (DG), harmful substances (HS) or marine pollutants (MP), or IMO hazard or pollutant category A, wing in ground (WIG)

   (l) 11 = power-driven vessel towing astern (regional use)
   (m) 12 = power-driven vessel pushing ahead or towing alongside (regional use)
   (n) 13 = reserved for future use
   (o) 14 = AIS-SART (active), MOB-AIS, EPIRB-AIS
   (p) 15 = undefined = default (also used by AIS-SART, MOB-AIS and EPIRB-AIS under test)

5. Rate of turn ROTAIS : 0 to +126 = turning right at up to 708 deg per min or higher 0 to -126 = turning left at up to 708 deg per min or higher Values between 0 and 708 deg per min coded by ROTAIS = 4.733 SQRT(ROTsensor) degrees per min where ROTsensor is the Rate of Turn as input by an external Rate of Turn Indicator (TI). ROTAIS is rounded to the nearest integer value. +127 = turning right at more than 5 deg per 30 s (No TI available) -127 = turning left at more than 5 deg per 30 s (No TI available) -128 (80 hex) indicates no turn information available (default). ROT data should not be derived from COG information.

6. SOG: Speed over ground in 1/10 knot steps (0-102.2 knots) 1 023 = not available, 1 022 = 102.2 knots or higher

7. Position accuracy: The position accuracy (PA) flag should be determined in accordance with the table below: 1 = high (¡= 10 m) 0 = low (¿ 10 m) 0 = default

8. Longitude: Longitude in 1/10 000 min (+/-180 deg, East = positive (as per 2's complement), West = negative (as per 2's complement). 181= (6791AC0h) = not available = default)

9. Latitude: Latitude in 1/10 000 min (+/-90 deg, North = positive (as per 2's complement), South = negative (as per 2's complement). 91deg (3412140h) = not available = default)

10. COG: Course over ground in 1/10 = (0-3599). 3600 (E10h) = not available = default. 3 601-4 095 should not be used

11. True heading: Degrees (0-359) (511 indicates not available = default)

12. Time stamp: UTC second when the report was generated by the electronic position system (EPFS) (0-59, or 60 if time stamp is not available, which should also be the default value, or 61 if positioning system is in manual input mode, or 62 if electronic position fixing system operates in estimated (dead reckoning) mode, or 63 if the positioning system is inoperative)

13. special maneuvre indicator: 0 = not available = default 1 = not engaged in special maneuver 2 = engaged in special maneuver (i.e.: regional passing arrangement on Inland Waterway)

14. Spare: Not used. Should be set to zero. Reserved for future use.
15. RAIM-flag: Receiver autonomous integrity monitoring (RAIM) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use. See Table
16. Communication state:
    (a) Sync state: 0 UTC direct (sync from own integral GPS receiver) 1 UTC indirect (own GPS unavailable - UTC sync from GPS receiver on nearby ship or base station) 2 Station is synchronized to a base station (base direct - GPS unavailable)

        3 Station is synchronized to another station based on the highest number of received stations or to another mobile station, which is directly synchronized to a base station (GPS unavailable)
    (b) Slot time-out: Specifies frames remaining until a new slot is selected 0 means that this was the last transmission in this slot 1-7 means that 1 to 7 frames respectively are left until slot change
    (c) Sub message: The sub message depends on the current value in slot time-out
17. Number of bits

A.2 NMEA messages from Gunnerus

YCMTW: (YC) Transducer Temp. (MTW) Mean Temperature of Water
Format: $–MTW,x.x,C*hh<CR><LF> Format Description:

1. Temperature, degrees
2. Unit of Measurement, Celsius
3. Checksum

SDDBT: (SD) Depth Sounder. (DBT) Depth below transducer
Format: $–DBT,x.x,f,x.x,M,x.x,F*hh<CR><LF> Format Description:

1. Water depth, feet
2. f = feet
3. Water depth, meters
4. M = meters
5. Water depth, Fathoms
6. F = Fathoms
7. Checksum

SDDBS: (SD) Depth Sounder. (DBS) Below Surface
Format: $–DBS,x.x,f,x.x,M,x.x,F*hh<CR><LF> Format Description:

1. Water depth, feet
2. f = feet
3. Water depth, meters
4. M = meters
5. Water depth, Fathoms
6. F = Fathoms
7. Checksum

SDDPT: (SD) Depth Sounder. (DPT) Depth of Water
Format: $–DPT,x.x,x.x,x.x*hh<CR><LF> Format Description:

1. Water depth relative to transducer, meters
2. Offset from transducer, meters positive means distance from transducer to water line negative means distance from transducer to keel
3. Maximum range scale in use (NMEA 3.0 and above)
4. Checksum

PFEC: Propietary sentence
Format: $PFEC,,xxxxx,x,x,x,x,x,x,x*hh<CR><LF> Format Description: unknown
PSIMSSB: Propietary sentence, SSB SSBL position
Format: $PSIMSSB,hhmmss.ss,cc__,A,cc_,aa__,aa__,aa__,x.x,x.x,x.x,x.x,*hh<CR><LF> Format Description:

1. = empty or time of reception
2. cc__ = Tp code (Examples: B01, B33, B47)
3. A = Status, A for OK and V for not OK. When the measurement is not OK the Error code field contains the error code
4. cc__ = Error code. Empty or a three character error code
5. aa__ = Coordinate system. C for Cartesian, P for Polar and U for UTM
6. aa__ = Orientation. H for vessel head up, N for North and E for East
7. aa__ = SW filter. M for Measured, F for Filtered and P for Predicted
8. x.x = X coordinate. The Coordinate system and Orientation fields should be used to decode the X and Y coordinate field

9. x.x = Y coordinate. The Coordinate system and Orientation fields should be used to decode the X and Y coordinate fields

10. x.x =Depth in metres

11. x.x =Expected accuracy of the position

12. aa__ = Additional information. N for None, C for Compass and I for Inclinometer

13. x.x = First additional value. Empty, Tp compass or Tp x inclination

14. x.x = First additional value. Empty or Tp y inclination

15. * = checksum delimiter

16. hh = empty or checksum

GPGBS: (GP) Global Positioning System receiver. (GBS) GPS Satellite Fault Detection
Format: $–GBS,hhmmss.ss,x.x,x.x,x.x,x.x,x.x,x.x,x.x*hh<CR><LF> Format Description:

1. UTC time of the GGA or GNS fix associated with this sentence. hh is hours, mm is minutes, ss.ss is seconds

2. Expected 1-sigma error in latitude (meters)

3. Expected 1-sigma error in longitude (meters)

4. Expected 1-sigma error in altitude (meters)

5. ID of most likely failed satellite (1 to 138)

6. Probability of missed detection for most likely failed satellite

7. Estimate of bias in meters on most likely failed satellite

8. Standard deviation of bias estimate

9. Checksum

GPRMC: (GP) Global Positioning System receiver. (RMC) Recommended Minimum Navigation Information
This is one of the sentences commonly emitted by GPS units.
Format: $–RMC,hhmmss.ss,A,ddmm.mm,a,dddmm.mm,a,x.x,x.x,xxxx,x.x,a*hh<CR><LF>
Format Description:

1. UTC of position fix, hh is hours, mm is minutes, ss.ss is seconds.

2. Status, A = Valid, V = Warning

3. Latitude, dd is degrees. mm.mm is minutes.

4. N or S
5. Longitude, ddd is degrees. mm.mm is minutes.
6. E or W
7. Speed over ground, knots
8. Track made good, degrees true
9. Date, ddmmyy
10. Magnetic Variation, degrees
11. E or W
12. FAA mode indicator (NMEA 2.3 and later)
13. Nav Status (NMEA 4.1 and later) A=autonomous, D=differential, E=Estimated, M=Manual input mode N=not valid, S=Simulator, V = Valid
14. Checksum

IIMWV: (II) Integrated Instrumentation. (MWV) Wind Speed and Angle
Format: $–MWV,x.x,a,x.x,a*hh<CR><LF> Format Description:

1. Wind Angle, 0 to 359 degrees
2. Reference, R = Relative, T = True
3. Wind Speed
4. Wind Speed Units, K/M/
5. Status, A = Data Valid, V = Invalid
6. Checksum

GPVTG: (GP) Global Positioning System receiver. (VTG) Track Made Good and Ground Speed
This is one of the sentences commonly emitted by GPS units.
Format: $–VTG,x.x,T,x.x,M,x.x,N,x.x,K,m*hh<CR><LF> Format Description:

1. Course over ground, degrees True
2. T = True
3. Course over ground, degrees Magnetic
4. M = Magnetic
5. Speed over ground, knots

6. N = Knots
7. Speed over ground, km/hr
8. K = Kilometers Per Hour
9. FAA mode indicator (NMEA 2.3 and later)
10. Checksum

GPZDA: (GP) Global Positioning System receiver. (ZDA) Time and Date
This is one of the sentences commonly emitted by GPS units.
Format: $–ZDA,hhmmss.ss,xx,xx,xxxx,xx,xx*hh<CR><LF> Format Description:

1. UTC time (hours, minutes, seconds, may have fractional subseconds)
2. Day, 01 to 31
3. Month, 01 to 12
4. Year (4 digits)
5. Local zone description, 00 to +- 13 hours
6. Local zone minutes description, 00 to 59, apply same sign as local hours
7. Checksum

PSIMSNS: (SNS) sensor values
compliant telegram designed by Kongsberg Maritime. It holds sensor values from a acoustic positioning transceivers. This telegram is typically generated by a HiPAP or HPR system.
Format: $PSIMSNS,hhmmss.ss,c–c,xx__,xx__,x.x,x.x,x.x,x.x, , , , , ,*hh<CR><LF>
Format description

1. PSIMSNS = talker identifier and telegram identifier
2. hhmmss.ss = coordinated universal time (UTC) of position. The decimal fraction is optional
3. xx__ = Transceiver number
4. xx__ = Transducer number
5. x.x = The roll angle in degrees
6. x.x = The pitch angle in degrees
7. x.x= The heave in metres
8. x.x= The heading in degrees. A value between 0 and 360
9. = empty field. May be used in the future

10. = empty field. May be used in the future
11. = empty field. May be used in the future
12. = empty field. May be used in the future
13. *= checksum delimiter
14. hh= checksum

VDVBW: (VD) Velocity Sensor, Doppler, other/general (VBW) Dual Ground/Water Speed
Format: $–VBW,x.x,x.x,A,x.x,x.x,A,x.x,A,x.x,A*hh<CR><LF> Format Description:

1. Longitudinal water speed, ”-” means astern, knots
2. Transverse water speed, ”-” means port, knots
3. Status, A = Data Valid
4. Longitudinal ground speed, ”-” means astern, knots
5. Transverse ground speed, ”-” means port, knots
6. Status, A = Data Valid
7. Stern traverse water speed, knots *NMEA 3 and above)
8. Status, stern traverse water speed A = Valid (NMEA 3 and above)
9. Stern traverse ground speed, knots *NMEA 3 and above)
10. Status, stern ground speed A = Valid (NMEA 3 and above)
11. Checksum

GPGGA: (GP) Global Positioning System receiver. (GGA) Global Positioning System Fix Data
Format: $–GGA,hhmmss.ss,ddmm.mm,a,ddmm.mm,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh<CR><LF>
Format Description:

1. UTC of this position report, hh is hours, mm is minutes, ss.ss is seconds.
2. Latitude, dd is degrees, mm.mm is minutes
3. N or S (North or South)
4. Longitude, dd is degrees, mm.mm is minutes
5. E or W (East or West)
6. GPS Quality Indicator (non null)

7. Number of satellites in use, 00 - 12
8. Horizontal Dilution of precision (meters)
9. Antenna Altitude above/below mean-sea-level (geoid) (in meters)
10. Units of antenna altitude, meters
11. Geoidal separation, the difference between the WGS-84 earth ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below ellipsoid
12. Units of geoidal separation, meters
13. Age of differential GPS data, time in seconds since last SC104 type 1 or 9 update, null field when DGPS is not used
14. Differential reference station ID, 0000-1023
15. Checksum

VDVLW : (VD) Velocity Sensor, Doppler, other/general. (VLW) Distance Traveled through Water
Format: $–VLW,x.x,N,x.x,N,x.x,N,x.x,N*hh<CR><LF> Format Description:

1. Total cumulative water distance, nm
2. N = Nautical Miles
3. Water distance since Reset, nm
4. N = Nautical Miles
5. Total cumulative ground distance, nm (NMEA 3 and above)
6. N = Nautical Miles (NMEA 3 and above)
7. Ground distance since reset, nm (NMEA 3 and above)
8. N = Nautical Miles (NMEA 3 and above)
9. Checksum

RAOSD: (RA) RADAR and/or ARPA. (OSD) Own Ship Data
Format: $–OSD,x.x,A,x.x,a,x.x,a,x.x,x.x,a*hh¡CR¿¡LF¿*hh<CR><LF> Format Description:

1. Heading, degrees True
2. Status, A = Data Valid, V = Invalid
3. Vessel Course, degrees True
4. Course Reference B/M/W/R/P

5. Vessel Speed
6. Speed Reference B/M/W/R/P
7. Vessel Set, degrees True
8. Vessel drift (speed)
9. Speed Units K/N
10. Checksum

RATTM: (RA) RADAR and/or ARPA. (TTM) Tracked Target Message
Format: $–TTM,xx,x.x,x.x,a,x.x,x.x,a,x.x,x.x,a,c–c,a,a,hhmmss.ss,a*hh<CR><LF>
Format Description:

1. Target Number (0-99)
2. Target Distance
3. Bearing from own ship
4. T = True, R = Relative
5. Target Speed
6. Target Course
7. T = True, R = Relative
8. Distance of closest-point-of-approach
9. Time until closest-point-of-approach "-" means increasing
10. Speed/distance units, K/N
11. Target name
12. Target Status
13. Reference Target
14. UTC of data (NMEA 3 and above) hh is hours, mm is minutes, ss.ss is seconds.
15. Type, A = Auto, M = Manual, R = Reported (NMEA 3 and above)
16. Checksum