A Python package for decoding Automatic Identification System (AIS) data from the Canadian Coast Guard

Lanli Guo, Jinshan Xu, Shihan Li and Jessica Wingfield

Ocean and Ecosystem Sciences Division
Maritimes Region
Fisheries and Oceans Canada

Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, Nova Scotia
Canada B2Y 4A2

2023

Canadian Technical Report of Hydrography and Ocean Sciences 360

Fisheries and Oceans Canada   Pêches et Océans Canada

Canada

Canadian Technical Report of Hydrography and Ocean Sciences

Technical reports contain scientific and technical information of a type that represents a contribution to existing knowledge but which is not normally found in the primary literature. The subject matter is generally related to programs and interests of the Oceans and Science sectors of Fisheries and Oceans Canada.

Technical reports may be cited as full publications. The correct citation appears above the abstract of each report. Each report is abstracted in the data base *Aquatic Sciences and Fisheries Abstracts.*

Technical reports are produced regionally but are numbered nationally. Requests for individual reports will be filled by the issuing establishment listed on the front cover and title page.

Regional and headquarters establishments of Ocean Science and Surveys ceased publication of their various report series as of December 1981. A complete listing of these publications and the last number issued under each title are published in the *Canadian Journal of Fisheries and Aquatic Sciences*, Volume 38: Index to Publications 1981. The current series began with Report Number 1 in January 1982.

Rapport technique canadien sur l'hydrographie et les sciences océaniques

Les rapports techniques contiennent des renseignements scientifiques et techniques qui constituent une contribution aux connaissances actuelles mais que l'on ne trouve pas normalement dans les revues scientifiques. Le sujet est généralement rattaché aux programmes et intérêts des secteurs des Océans et des Sciences de Pêches et Océans Canada.

Les rapports techniques peuvent être cités comme des publications à part entière. Le titre exact figure au-dessus du résumé de chaque rapport. Les rapports techniques sont résumés dans la base de données *Résumés des sciences aquatiques et halieutiques*.

Les rapports techniques sont produits à l'échelon régional, mais numérotés à l'échelon national. Les demandes de rapports seront satisfaites par l'établissement auteur dont le nom figure sur la couverture et la page de titre.

Les établissements de l'ancien secteur des Sciences et Levés océaniques dans les régions et à l'administration centrale ont cessé de publier leurs diverses séries de rapports en décembre 1981. Vous trouverez dans l'index des publications du volume 38 du *Journal canadien des sciences halieutiques et aquatiques*, la liste de ces publications ainsi que le dernier numéro paru dans chaque catégorie. La nouvelle série a commencé avec la publication du rapport numéro 1 en janvier 1982.

Canadian Technical Report of
Hydrography and Ocean Sciences 360

2023

A Python package for decoding Automatic Identification System (AIS) data from the Canadian Coast Guard

by

Lanli Guo, Jinshan Xu, Shihan Li and Jessica Wingfield[1]

Ocean and Ecosystem Sciences Division
Fisheries and Oceans Canada
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, Nova Scotia
Canada B2Y 4A2

[1] Marine Planning and Conservation Division, Fisheries and Oceans Canada, Maritimes Region, Bedford Institute of Oceanography, P.O. Box 1006, Dartmouth, Nova Scotia, Canada, B2Y 4A2

Cat. No. Fs97-18/360E-PDF ISBN 978-0-660-67831-3 ISSN 1488-5417

Correct Citation for this publication:

Guo, L., Xu, J., Li, S., and Wingfield, J. 2023. A Python package for decoding Automatic Identification System (AIS) data from the Canadian Coast Guard. Can. Tech. Rep. Hydrogr. Ocean Sci. 360: vi + 57 p.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

Guo, L., Xu, J., Li, S., and Wingfield, J.. 2023. A Python package for decoding Automatic Identification System (AIS) data from the Canadian Coast Guard. Can. Tech. Rep. Hydrogr. Ocean Sci. 360: vi + 57 p.

The Automatic Identification System (AIS) is an automated tracking system that is used in the shipping industry for monitoring global vessel traffic. Beyond its primary purpose of supporting the safety of life at sea by reducing vessel collision risk, AIS data can be used to better understand vessel impacts on the environment, such as underwater noise and vessel strike risks in whale habitat and migration routes. Before AIS data can be used for these purposes, the raw datasets must be decoded from ASCII formats. This report describes a package of Python scripts and programs which were developed to decode AIS data collected and provided by the Canadian Coast Guard. The package decodes 26 types of AIS messages, which comprise more than 94% of the messages received by the Canadian Coast Guard. The scripts can run by either serial or parallel methods to improve the efficiency. This package decodes more message types than previously written packages, thus enhancing the potential of AIS data for use in analysis and research, such as that to inform conservation efforts for NARWs.

Résumé

Guo, L., Xu, J., Li, S., and Wingfield, J.. 2023. A Python package for decoding Automatic Identification System (AIS) data from the Canadian Coast Guard. Can. Tech. Rep. Hydrogr. Ocean Sci. 360: vi + 57 p.

Le système d'identification automatique (AIS) est un système de suivi automatisé utilisé dans l'industrie du transport maritime pour surveiller le trafic mondial des navires. Au-delà de leur objectif principal de soutenir la sécurité de la vie humaine en mer en réduisant le risque de collision des navires, les données AIS peuvent être utilisées pour mieux comprendre les impacts des navires sur l'environnement, tels que le bruit sous-marin et les risques de collision avec les navires dans l'habitat des baleines et les routes de migration. Avant que les données AIS puissent être utilisées à ces fins, les ensembles de données brutes doivent être décodés à partir des formats ASCII. Ce rapport décrit un ensemble de scripts et de programmes Python qui ont été développés pour décoder les données AIS recueillies et fournies par la Garde côtière canadienne. Le progiciel décode 26 types de messages AIS, qui représentent plus de 94 % des messages reçus par la Garde côtière canadienne. Les scripts peuvent être exécutés par des méthodes série ou parallèles pour améliorer l'efficacité. Ce package décode plus de types de messages que les packages précédemment écrits, améliorant ainsi le potentiel des données AIS pour une utilisation dans l'analyse et la recherche, telles que celles pour informer les efforts de conservation des NARW.

1 INTRODUCTION

The Automatic Identification System (AIS) is an automated vessel tracking and identification system intended to enhance the safety of life at sea, the safety and efficiency of navigation, and the protection of the marine environment (International Maritime Organization (IMO) A 29/Res. 1106). Since 2002, IMO's International Convention for the Safety of Life at Sea (SOLAS) requires operating Class A AIS transponders on all vessels of 300 gross tonnage (GT) or greater on an international voyage and vessels of 500 GT or greater not on an international voyage (IMO A 29/Res. 1106). In Canada, Class A AIS transponders are also required for vessels that are 20 m or greater in length (other than pleasure crafts), towboats that are 8 m or greater in length, vessels carrying more than 50 passengers, dredges or floating plants located in a place where they pose a collision hazard, and vessels carrying dangerous goods or pollutants (Navigation Safety Regulations SOR/2020-216). Class A or B transponders are required for passenger vessels and vessels of 8 m or greater in length carrying a passenger that are traveling outside of sheltered waters (Navigation Safety Regulations SOR/2020-216). Class A devices (SOLAS compliant, using Self-Organizing Time Division Multiple Access (TDMA) broadcast mode) transmit at a power level of 12.5 watts, while Class B devices (using Carrier-Sense TDMA broadcast mode) transmit at a power level of 2 watts (IMO A 29/Res. 1106). Signals from Class A transopnders are given priority over signals from Class B transponders. Since Class B transceivers are more affordable and interoperable, they are more commonly used by nonmandated vessels, such as fishing boats, recreational boats, small domestic ships, and even artisanal craft.

AIS signals are transmitted over Very High Frequency (VHF) radio using TDMA technology. Terrestrial AIS datasets have a much higher temporal resolution than satellite AIS datasets, as land-based stations are able to constantly receive signals while the receipt of signals by satellites is contingent upon the frequency of orbital passes (Iacarella et al. 2020). However, as terrestrial receivers must have vessels in their line of sight in order to receive the signals, some signals can be lost if the vessel moves behind a land mass or another vessel. The height of the receiver will greatly affect the distance at which a AIS will be received. The same is true for the transponders on the vessel. The spatial extent of the data also depends on how many land-based receivers there are along the nearby coastline. The Canadian Coast Guard has been collecting AIS data from a network of terrestrial receivers since 2012.

Researchers have used AIS data to address a variety of conservation issues, such as characterizing vessel traffic in North Atlantic right whale (*Eubalaena glacialis*; hereafter NARW) critical habitat and migration route (van der Hoop et al. 2012; Conn and Silber 2013); examining the threat of vessel presence and acoustic disturbance within Southern Resident killer whale (*Orcinus orca*) critical habitat in the Salish Sea and Swiftsure Bank area (Vagle et al. 2021); Assessing the risk of ships striking large whales, like humpback (*Megaptera novaeangliae*), blue (*Balaenoptera musculus*), and fin (*Balaenoptera physalus*) whales off Southern California (Redfern et al., 2013) and the risk of chronic shipping noise to baleen whales (Redfern et al., 2017); investigating the vessel risks to marine wildlife in the Tallurutiup Imanga National Marine Conservation Area and the eastern entrance to the Northwest Passage (Halliday et al., 2022). AIS data has been used as an input for fishing monitoring systems and is increasingly being used as a means to assess ambient noise levels resulting from shipping (Fournier et al., 2018). Improving the availability of AIS data can improve the efficacy of such analyses.

AIS data is ideal for characterizing vessel presence for both historical and real-time analytics. AIS data is originally transmitted and recorded in coded messages. In order to use the data, the messages must first to be decoded. This can present a unique challenge, as an extremely large number of messages are received and recorded, ranging from 10 million to 40 million messages for just a single day. There are two existing AIS decoder packages that were developed at DFO Maritimes Region. The first was written in Fortran by Norman A. Cochrane, and the second was written in R (R Core Team 2021) by Angelia Vanderlaan using functions from the packages “stringr” (Wickham 2019) and “compositions” (van den Boogaart et al. 2021) in addition to several base functions. These packages can decode typical AIS messages, like location related message type 1, 2, 3, 5, and 18. However, there are additional, less common AIS message types that are also of interest, like safety related message type 12 and 14. The processing speed of the previous decoder is not efficient, which is another reason for us to create a new package. Therefore, building on this work, we developed a package using the Python programming language (Python Software Foundation) that is able to decode additional, these less typical AIS message types in addition to those decoded by the previously developed packages and improve the processing speed. Python is an interpreted, high-level, general-purpose dynamic programming language. Python is freely available and distributable for many operating systems.

The purpose of this report is to provide details of the development and usage of this new set of Python AIS decoding scripts. We present a brief introduction of the AIS data format in Section 2. Section 3 describes the structure of the decoding system. Instructions for running this package are provided in Section 4. A summary of the performance of this system and issues that have yet to be addressed are presented in Section 5, and an overall summary is given in Section 6.

2 AIS MESSAGE FORMAT

AIS data are reported as ASCII data packets using the NMEA 0183 or NMEA 2000 data formats (NMEA 0183 is more commonly used in mobile devices). AIS packets use the introducer "!AIVDM" for reports from other ships and "!AIVDO" for reports from the ship receiving its own signal. The standard for the AIVDM/AIVDO messages is the ITU Recommendation M.1371, "Technical Characteristics for a Universal Shipborne Automatic Identification System Using Time Division Multiple Access in the VHF maritime mobile frequency band " (ITU1371). Issued in 2001, this standard first described the bit-level format of AIS radio messages. ITU-R M.1371-4 defines 27 different AIS messages shown in Table 2. The recommendation was expanded upon and clarified by the "IALA Technical Clarifications on Recommendation ITU-R M.1371-5", which is freely available.

2.1 AIVDM/AIVDO SENTENCE STRUCTURE

AIS messages are relayed in ASCII using the NMEA 0183 format, the standard for data interchange in marine navigation systems. An example of a typical AIS message is as follows:

!AIVDM,1,1,,A,400TcdiuiT7VDR>3nIfr6>i00000,0*78

The numbers and corresponding definitions of each field (separated by a comma) are as follows:

- Field 1, *!AIVDM*, identifies this as an AIVDM packet.
- Field 2, *1* in this example, is the number of fragments in the message. The payload size of each sentence is limited by NMEA 0183's 82-character maximum, and therefore some payloads must be split into several fragments.
- Field 3, *1* in this example, is the fragment number. Therefore, a sentence with a fragment count of 1 and a fragment number of 1 is complete in and of itself.
- Field 4, empty in this example, is a sequential message ID for multi-sentence messages.

- Field 5, *A* in this example, is a radio channel code. AIS uses the high side of the duplex from two VHF radio channels: AIS Channel A is 161.975Mhz (87B) and AIS Channel B is 162.025Mhz (88B) (IMO A 29/Res. 1106). Channel codes '1' and '2' may also be encountered instead of A or B.
- Field 6, *400TcdiuiT7VDR>3nIfr6>i00000* in this example, is the data payload. The process for decoding the payload is described in Section 3.
- Field 7, *0*78* in this example, number before * is the number of fill bits required to pad the data payload to a 6 bit boundary, ranging from 0 to 5 (0 in this example), followed by the NMEA 0183 data-integrity checksum for the sentence (78 in this example). The checksum is computed using the entire sentence, after the leading "!" to before "*".

The syntax and semantics of fields 1 to 4 are fixed and the fill-bit field and NMEA checksum are required. A correct checksum is the first criterion used for deciding whether the message should be decoded. AIVDM/AIVDO messages have a two-layer protocol. The fields listed above are the outer layer information. The payload from Field 6 contains the inner layer information, which is an ASCII-encoded bit vector. Each character represents six bits of data. The character data is converted into an ASCII character value. To determine the six bits, 48 must be subtracted from the ASCII character value; if the result is still greater than 40, subtract 8. The data are then converted from the resulting decimal into a 6-bit binary format. Characters, their ASCII values, the calculated decimal and resulting bits are shown in Table 1. For the payload from Field 6, each character was transferred to 6 bits based on Table 1. After all six-bit quantities found in the payload were concatenated together, bit fields will be interpreted and converted based on different types of messages (Table 6 -28). The decoding process is illustrated in Figure 1.

Table 1. ASCII payload armoring.

Char	ASCII	Decimal	Bits	Char	ASCII	Decimal	Bits	Char	ASCII	Decimal	Bits
"0"	48	0	000000	**"F"**	70	22	010110	**"d"**	100	44	101100
"1"	49	1	000001	**"G"**	71	23	010111	**"e"**	101	45	101101
"2"	50	2	000010	**"H"**	72	24	011000	**"f"**	102	46	101110
"3"	51	3	000011	**"I"**	73	25	011001	**"g"**	103	47	101111
"4"	52	4	000100	**"J"**	74	26	011010	**"h"**	104	48	110000
"5"	53	5	000101	**"K"**	75	27	011011	**"i"**	105	49	110001
"6"	54	6	000110	**"L"**	76	28	011100	**"j"**	106	50	110010
"7"	55	7	000111	**"M"**	77	29	011101	**"k"**	107	51	110011
"8"	56	8	001000	**"N"**	78	30	011110	**"l"**	108	52	110100
"9"	57	9	001001	**"O"**	79	31	011111	**"m"**	109	53	110101

":"	58	10	001010	**"P"**	80	32	100000	**"n"**	110	54	110110
";"	59	11	001011	**"Q"**	81	33	100001	**"o"**	111	55	110111
"<"	60	12	001100	**"R"**	82	34	100010	**"p"**	112	56	111000
"="	61	13	001101	**"S"**	83	35	100011	**"q"**	113	57	111001
">"	62	14	001110	**"T"**	84	36	100100	**"r"**	114	58	111010
"?"	63	15	001111	**"U"**	85	37	100101	**"s"**	115	59	111011
"@"	64	16	010000	**"V"**	86	38	100110	**"t"**	116	60	111100
"A"	65	17	010001	**"W"**	87	39	100111	**"u"**	117	61	111101
"B"	66	18	010010	**"`"**	96	40	101000	**"v"**	118	62	111110
"C"	67	19	010011	**"a"**	97	41	101001	**"w"**	119	63	111111
"D"	68	20	010100	**"b"**	98	42	101010				
"E"	69	21	010101	**"c"**	99	43	101011				

2.2 AIS MESSAGE TYPES

The first 6 bits of the payload are the message type. There are 27 AIS message types (Table 2). In practice, message types other than 1, 3, 4, 5, 18, and 24 are rare; many AIS transmitters do not emit them. In normal operation, AIS transponders broadcast a position report (type 1, 2, or 3 from vessels with Class A transponders) every 2 to 10 seconds depending on the vessel’s speed while underway, and every 3 minutes while the vessel is stationary. These messages may include a vessel's MMSI number, longitude, latitude, rate of turn, speed, true heading, and other parameters. Vessel's location and motion could also be transmitted through message type 4 every 3 1/3 seconds from a base station and message type 18 every 30 to 180 seconds from vessels with Class B transponders. AIS transponders also send static and voyage related data (type 5 from Class A vessels and type 24 from Class B vessels) every 6 minutes. Different from the other message types, both message type 6 and type 8 have many subtypes (kinds). Message type 6 is used for unencrypted structured extension messages conforming to the Inland AIS standard, and by local authorities such as the St. Lawrence Seaway and the U.S Coast Guard’s PAWSS (the Ports and Waterways Safety System). Type 8 messages are private encrypted messages, used for location transmission in military exercises and other sensitive operations. Type 8 messages can also be used for unencrypted structured extension messages by Inland AIS, and by local authorities such as the St. Lawrence Seaway and PAWSS. Due to their specialized purpose, we only decode two kinds of message types 6 and 8. All message types are described in detail in Tables 6 through 28 in Appen_ dix 1. Since we did not receive any type 13 messages and there is currently not enough information for how to properly decode these messages, we did not include this type in our decoder.

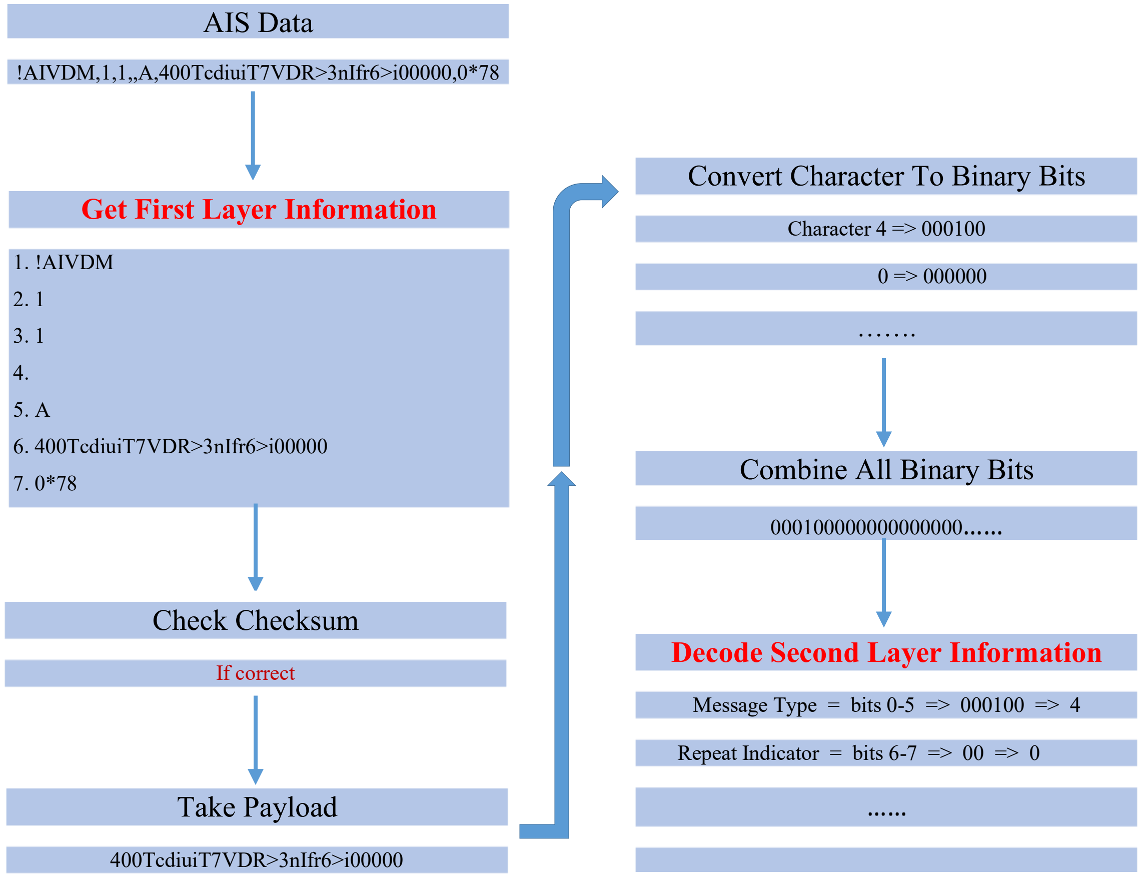


Figure 1. The AIS message payload decoding process.

Table 2. Descriptions of each of the AIS message types.

01	Position Report Class A
02	Position Report Class A (Assigned schedule)
03	Position Report Class A (Response to interrogation)
04	Base Station Report
05	Static and Voyage Related Data
06	Binary Addressed Message
07	Binary Acknowledge
08	Binary Broadcast Message
09	Standard SAR Aircraft Position Report
10	UTC and Date Inquiry
11	UTC and Date Response
12	Addressed Safety Related Message
13	Safety Related Acknowledgement
14	Safety Related Broadcast Message

15	Interrogation
16	Assignment Mode Command
17	DGNSS Binary Broadcast Message
18	Standard Class B CS Position Report
19	Extended Class B Equipment Position Report
20	Data Link Management
21	Aid-to-Navigation Report
22	Channel Management
23	Group Assignment Command
24	Static Data Report
25	Single Slot Binary Message
26	Multiple Slot Binary Message With Communications State
27	Position Report For Long-Range Applications

2.3 OPEN RESOURCES FOR AIS DECODING

Kurt Schwehr, a research scientist at the Center for Coastal and Ocean Mapping at the University of New Hampshire, provides a collection of Python scripts for decoding and analyzing AIVDM sentences on his website (http://vislab-ccom.unh.edu/~schwehr/).

AISHub is a free and publically accessible AIS feed pool, which allows for the exchange of AIS data in raw NMEA format (https://www.aishub.net/api). AISHub members share their AIS data and receive the merged feed from all other participating parties.

The source-code repository of the GPSD project (GPS service Daemon, a suite of tools for managing collections of GPS devices and other sensors including AIS) contains a conforming standalone Python decoder, named “ais.py”. More information on this decoder can be found here: https://gpsd.gitlab.io/gpsd/AIVDM.html.

The Maritec decoder (https://maritec.co.za/tools/aisvdmvdodecoding) is a high-quality decoder and can be exercised through public website.

At DFO Maritimes Region, Norman A. Cochrane developed Fortran codes for decoding message types 1, 2, 3, 5 and 18, and Angelia Vanderlaan developed an R package for decoding message types 1, 2, 3, 5, 18, 19, 24, and 27. The R package was specifically designed to capture location information of vessels and information on the vessels themselves, which is why not all message types are decoded.

Rely on these resources, we build up our phython decoding system which is used to decode the AIS data with unique format received from the Canadian Coast Guard. We set up different modules to make the decoding system easy to maintain and modify. Meanwhile, we used parallel method which make the system more efficient.

3 STRUCTURE OF THE PYTHON DECODING SYSTEM

The structure of the Python decoding system is shown in Figure 2. There are two ways to run the decoder either in serial (main code is Process_AIS_Serial.py) or in parallel (main code is Process_AIS_Parallel.py). The remaining files are specific modules for decoding.

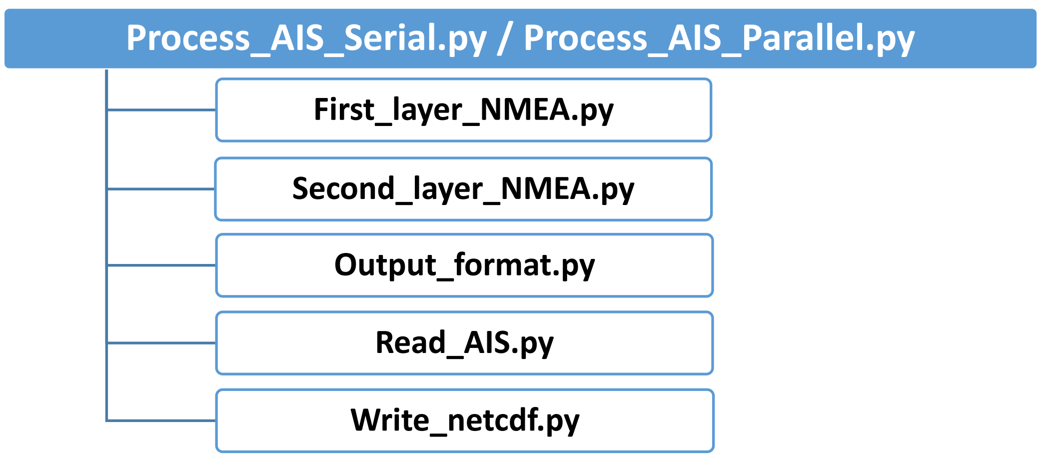


Figure 2. Structure of the Python decoding system described in this report.

3.1 SERIAL RUN

Process_AIS_Serial.py contains the main code for controlling the decoding process. This file is specifically designed for dealing with terrestrial AIS data in the format provided by the Canadian Coast Guard. The typical format of the AIS data starts with the indicator “c:”. The AIS data from the Canadian Coast Guard are contained in either a .txt file or a .csv file, the patterns for which are different. For a .txt file, the pattern is as follows:

```
c:1506815999,C:2234,s:P-Calvert*4F
!AIVDM,1,1,9,B,H4eGD9PP5=@D000000000000000,2*01
```

For a .csv file, the pattern is as follows:

```
\ c:1506815999,C:2234,s:P-Calvert*4F\!AIVDM,1,1,9,B,H4eGD9PP5=@D000000000000000,2*01
```

In addition to the standard AIS message, other information is provided. Following the indicator “c:”, “1506815999 ” is the time of the message received in elapsed seconds since January

1 1970, 00:00:00 (UTC). “C:” is the slot number indicator. In this case, the slot number is 2234. “s:” is the location indicator. The uppercase letter following the location indicator represents the region. In this example, the region is “P”, which means Pacific. The other region codes are “M” for Maritimes, “Q” for Quebec, “C” for Central, and “N” for Newfoundland. After “-”, the name of the terrestrial receiving station is provided. In this example, the station name is “Calvert”. These information are decoded in **Process_AIS_Serial.py** and **Process_AIS_Parrallel.py**. There are other formatted AIS data from Canadian Coast Guard. Our code can also handle the format starting with the indicator “s:”.

First_layer_NMEA.py includes functions for dealing with the first layer of information in the AIS message. The original author of these functions is Pierre Payen and they can be found here: https://github.com/pirpyn/pyAISm.

The functions are:

sign_int converts signed pack of bytes (as a string) to signed integer.

compute_checksum checksum, the *-separated suffix (field 7), is an extra information that is included with each sentence of the data sent by AIS device. The checksum is a value calculated based on the contents of the sentence. The decoder makes the same calculation and compares its value with the one received. If the calculated and received values do not match, the sentence should be discarded. Computing the checksum of an AIS sentence is to exclusive OR (XOR) all of the characters (including the commas) between the '!' and the '*'. XOR or exclusive disjunction is a logical operation that is true if and only if its arguments differ, for example XOR 101001 and 011110 results in 110111 (because from the left, bits 1, 2, 4, 5 and 6 are different, and bits 3 are the same). To start this process, we set zero (000000) as the initial XOR’d output. We take the 6-bit binary format of each character and XOR it with the previous XOR’d output. This process is repeated until the very last character is XOR’d. The final XOR result will be output in hexadecimal and compared with the received checksum value (number after *).

get_msg_type reads the AIS sentence and returns the message introducer (i.e. "!AIVDM" or "!AIVDO"; Field 1 of section 2.1).

get_payload reads the AIS sentence and returns the payload (Field 6 of section 2.1).

get_sentence_number reads the AIS sentence and returns the count of fragments in the message (Field 2 of section 2.1).

get_sentence_count reads the AIS sentence and returns the fragment number of the sentence (Field 3 of section 2.1).

get_checksum reads the AIS sentence and returns the checksum of the sentence (Field 7 of section 2.1).

decod_payload converts the payload from ASCII characters to their 6-bit counterparts.

decod_6bits_ascii decodes 6-bits into an ASCII character, with respect to the 6-bits ASCII table (Table 3) below.

decod_str decodes a string of bits to ASCII characters with respect to the 6-bits ASCII table (Table 3). It uses decod_6bits_ascii above to decode each 6-bits to an ASCII character.

Second_layer_NMEA.py includes the functions for dealing with the second layer information of the AIS message. The original functions, which decode 9 message types, are from https://github.com/pirpyn/pyAISm. We expanded the code to deal with 26 message types, including two kinds of message type 8 and two kinds of message type 6. This file also includes the code for combining the sentence fragments.

Output_format.py includes the functions for dealing with the format of the decoded AIS data.

Read_AIS.py is used to collect the decoded data from each message and prepare to save them as netCDF (network Common Data Form, .nc) files. NetCDF is a machine-independent data format which has been adopted as a standard way to represent array-oriented scientific data by many organizations and scientific groups in different countries. Interfaces to netCDF are available in many popular languages which include Fortran, R, Perl, Python, Ruby, Haskell, Mathematica, MATLAB, IDL, Julia and Octave. Meanwhile, the collected data can be readily converted to other format, like csv.

Write_netcdf.py saves the output of the decoded AIS data in .nc files.

Table 3. 6-bits ASCII.

Bits	Decimal	ASCII	Bits	Decimal	ASCII	Bits	Decimal	ASCII	Bits	Decimal	ASCII
000000	0	"@"	**010000**	16	"P"	**100000**	32	" "	**110000**	48	"0"
000001	1	"A"	**010001**	17	"Q"	**100001**	33	"!"	**110001**	49	"1"
000010	2	"B"	**010010**	18	"R"	**100010**	34	""""	**110010**	50	"2"
000011	3	"C"	**010011**	19	"S"	**100011**	35	"\#"	**110011**	51	"3"
000100	4	"D"	**010100**	20	"T"	**100100**	36	"$"	**110100**	52	"4"
000101	5	"E"	**010101**	21	"U"	**100101**	37	"%"	**110101**	53	"5"
000110	6	"F"	**010110**	22	"V"	**100110**	38	"&"	**110110**	54	"6"

000111	7	"G"	**010111**	23	"W"	**100111**	39	"\""	**110111**	55	"7"
001000	8	"H"	**011000**	24	"X"	**101000**	40	"("	**111000**	56	"8"
001001	9	"I"	**011001**	25	"Y"	**101001**	41	")"	**111001**	56	"9"
001010	10	"J"	**011010**	26	"Z"	**101010**	42	"*"	**111010**	58	":"
001011	11	"K"	**011011**	27	"["	**101011**	43	"\+"	**111011**	59	";"
001100	12	"L"	**011100**	28	"\"	**101100**	44	","	**111100**	60	"<"
001101	13	"M"	**011101**	29	"]"	**101101**	45	"-"	**111101**	61	"="
001110	14	"N"	**011110**	30	"\^"	**101110**	46	"."	**111110**	62	">"
001111	15	"O"	**011111**	31	"_"	**101111**	47	"/"	**111111**	63	"?"

3.2 PARALLEL RUN

Process_AIS_Parallel.py is the main code for running the decoding package using the parallel method. This is an updated version of Process_AIS_Serial.py, and is more efficient than the serial method. The serial method is convenient for a test run to debug. The functions from First_layer_NMEA.py, Second_layer_NMEA.py, Output_format.py, Read_AIS.py and Write_netcdf.py are also used in parallel run. Both the serial and parallel code are available to the user.

4 RUNNING THE PYTHON DECODER

4.1 SYSTEM SET-UP

To run the decoding package, Python libraries such as Numpy, netCDF4, and Ray must be installed. The “Input_Directory”, “Output_Directory” and “Outlog_Directory” must be setup in Process_AIS_Serial.py (or Process_AIS_Parallel.py).

4.2 MODIFYING THE CODE TO ADD MORE MESSAGE TYPES

This package can decode 26 message types, including two kinds of message type 6 and two kinds of message type 8. More subtypes/kinds of message type 6 and 8 and message type 13 can be added by modifying the file ‘Second_layer_NMEA.py’. The code should be added within the function ‘decod_data’. If adding a completely new type, like type 13, the function ‘decod_type’ should also be modified.

Table 4 provides a summary of the information from the decoded AIS data and the output we generated. The “Full Name” listed in the table is the normal name of a variable, meanwhile the “Short Name” is the name we used in the code and output file. The dash (-) through the fields means that variable is not included in the dynamic or static outputs. To delete some output variables (e.g. Type, MMSI), the easy way is to delete them in Write_netcdf.py. To add some output variables, the first step is to define the variables, and modify the calling functions read_dyn, read_sta, write_dyn, and write_sta in Process_AIS_Serial.py (Process_AIS_Parallel.py); the second step is to add the definitions of those variables in Read_AIS.py and Write_netcdf.py; third, the formats of the new variables need to be defined in Output_format.py.

Table 4. Description of the output from the decoded AIS message.

Main Decoded AIS Parameter		**Output Variable**	
Full Name	**Short Name**	**Dynamic**	**Static**
Message Type	Type	Type	Type
Repeat Indicator	Repeat	Repeat	Repeat
MMSI	MMSI	MMSI	MMSI
Rate of Turn (ROT)	Turn (°/min)	Turn (°/min)	-
Navigation Status	Status	Status	-
Speed Over Ground (SOG)	Speed (knots)	Speed (knots)	-
Position Accuracy	Accuracy	Accuracy	-
Longitude	Lon (°)	Lon (°)	-
Latitude	Lat (°)	Lat (°)	-
Course over Ground (COG)	Course (°)	Course (°)	-
True Heading (HDG)	Heading (°)	-	-
Time Stamp	Timestamp	-	-
Maneuver Indicator	Maneuver	-	-
RAIM flag	RAIM	-	-
Radio status	Radio	-	-
ETA Year (UTC)	Year	-	-
ETA Month (UTC)	Month	-	-
ETA Day (UTC)	Day	-	-
ETA Hour (UTC)	Hour	-	-
ETA Minute (UTC)	Minute	-	-
Draught	Draught (m)	-	-
Destination	Destination	-	-
Destination MMSI	Dest_MMSI	-	-
Designated Area Code	Dac	-	-
Functional ID	Fid	-	-
Retransmit flag	Retransmit	-	-
NE Longitude	Ne_lon (°)	-	-
NE Latitude	Ne_lat (°)	-	-
SW Longitude	Sw_lon (°)	-	-
SW Latitude	Sw_lat (°)	-	-
Vessel Name	Shipname	-	Shipname
Ship Type	Shiptype	-	Shiptype
Dimension to Bow	To_bow (m)	-	To_bow (m)
Dimension to Stern	To_stern (m)	-	To_stern (m)
Dimension to Port	to_port (m)	-	to_port (m)
Dimension to Starboard	To_starboard (m)	-	To_starboard (m)

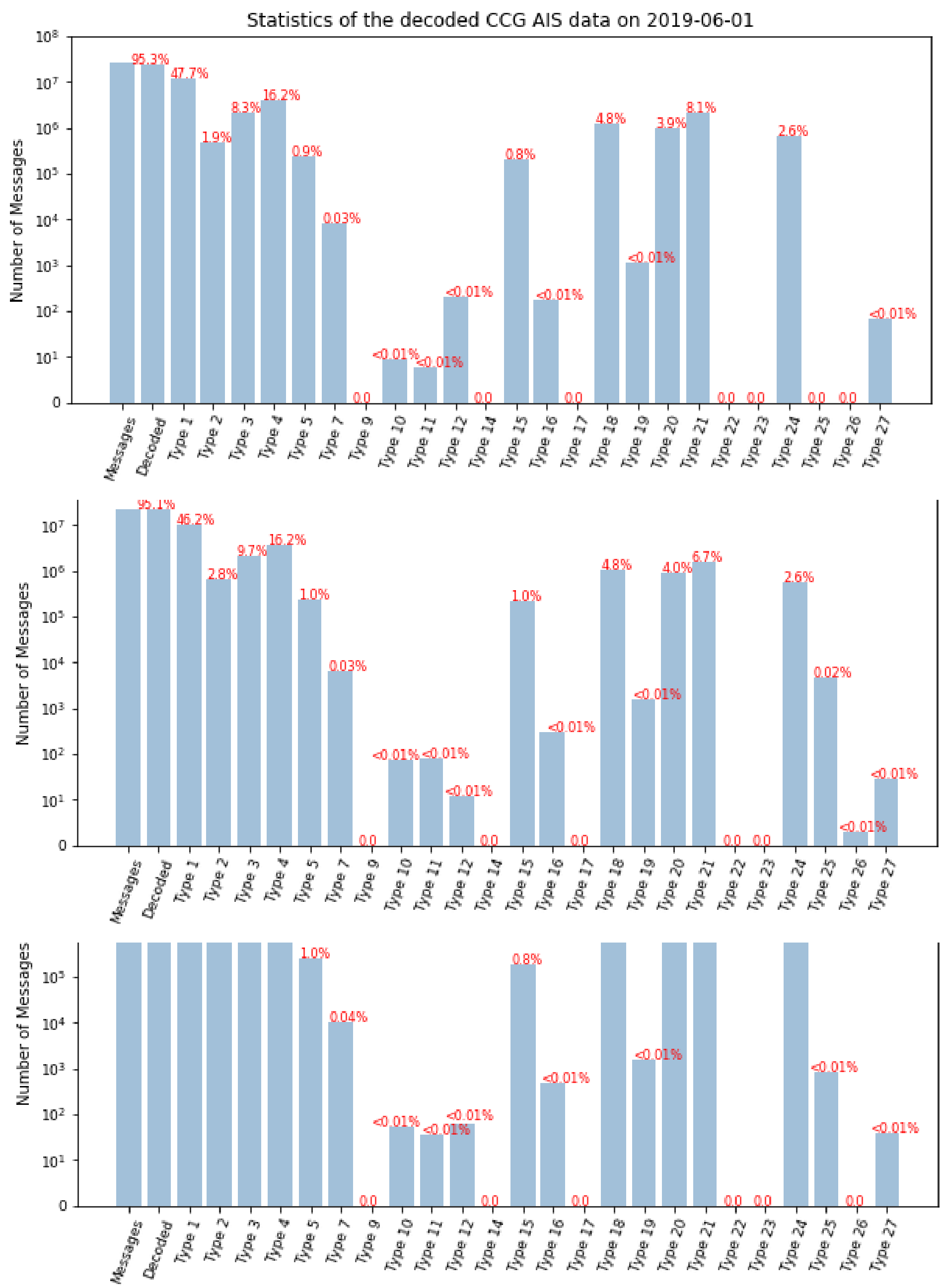
IMO Number	Imo	-	-
		Station_region	Station_region
		Station_name	Station_name
		Station_time	Station_time

5 DECODER PERFORMANCE

5.1 RESULTS

We used decoded data from June 1 to 5, 2019 as examples to check the performance of this system. At Bedford Institute of Oceanography, the received AIS messages from the Canadian Coast Guard were saved as a daily data. Since this python package is capable of decoding only two kinds of message types 6 and 8, we did not include types 6 and 8 in the following calculations. Figure 3 provides the results from the Python decoding system. The percent values shown on the figure are the ratios of the decoded messages to the total received messages. The number of lines differs from number of messages on days when messages were separated into multiple lines. The Python system decoded more than 94% of the messages received each day from the Canadian Coast Guard. Percentages decoded for each message type were similar across days, and more than 75% of the decoded messages were of type 1, 2, 3, 4, or 5. The decoded results were also summarized in Table 5. Because some message types are rare and occasionally appeared, the Python decoding system decoded 17-19 message types each day. To decode 1 day's data, it usually took around half an hour by using serial method, and less than 1 minute by using parrell method with 50 processers.

For the location related messages, the performance of this system was compared against the previous decoding system originally developed by Angelia Vanderlaan et al. in R. To compare with the output from R system, we used the same time range as R system each day. Figure 4 shows positions of vessels over the eastern Canadian shelf region, using the decoded vessel locations from two different systems. The positions of vessels from both systems are nearly the same, but the Python decoding system provides additional position information near the coastline (Figure 4). In Figure 5, red dots shows extra vessel locations resulting from the Python decoding system that were missing from the R decoding system output, and vice versa the blue dots are the extra vessel lovations results from the R decoding system. Since the Python decoder decodes more messages from type 1, 2, 3, 4 and 21, it seems that the Python decoding system therefore provides better spatial converage than the R decoding system.



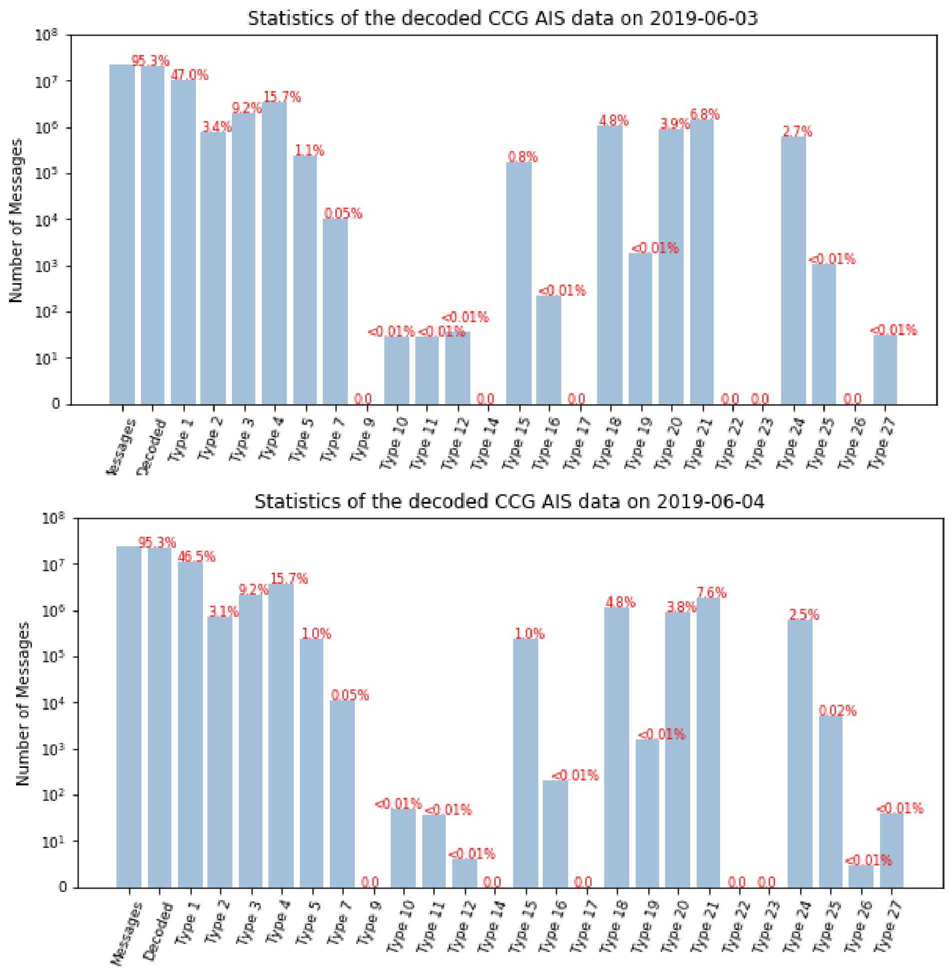


Figure 3. Messages decoded for each message type using the Python decoder. The red digits are the percentages of messages that were decoded by Python decoding system to the total received messages.

Table 5. The number of decoded messages and unique MMSI numbers resulting from the Python decoding system described in this report using data from the Canadian Coast Guard from June 1 to June 5 2019.

Date	Python Decoding System	
	Decoded messages	MMSI numbers
2019-06-01	25,188,773	4,230
2019-06-02	24,700,827	4,293
2019-06-03	21,796,274	3,732
2019-06-04	23,306,122	3,768
2019-06-05	22,365,458	3,755

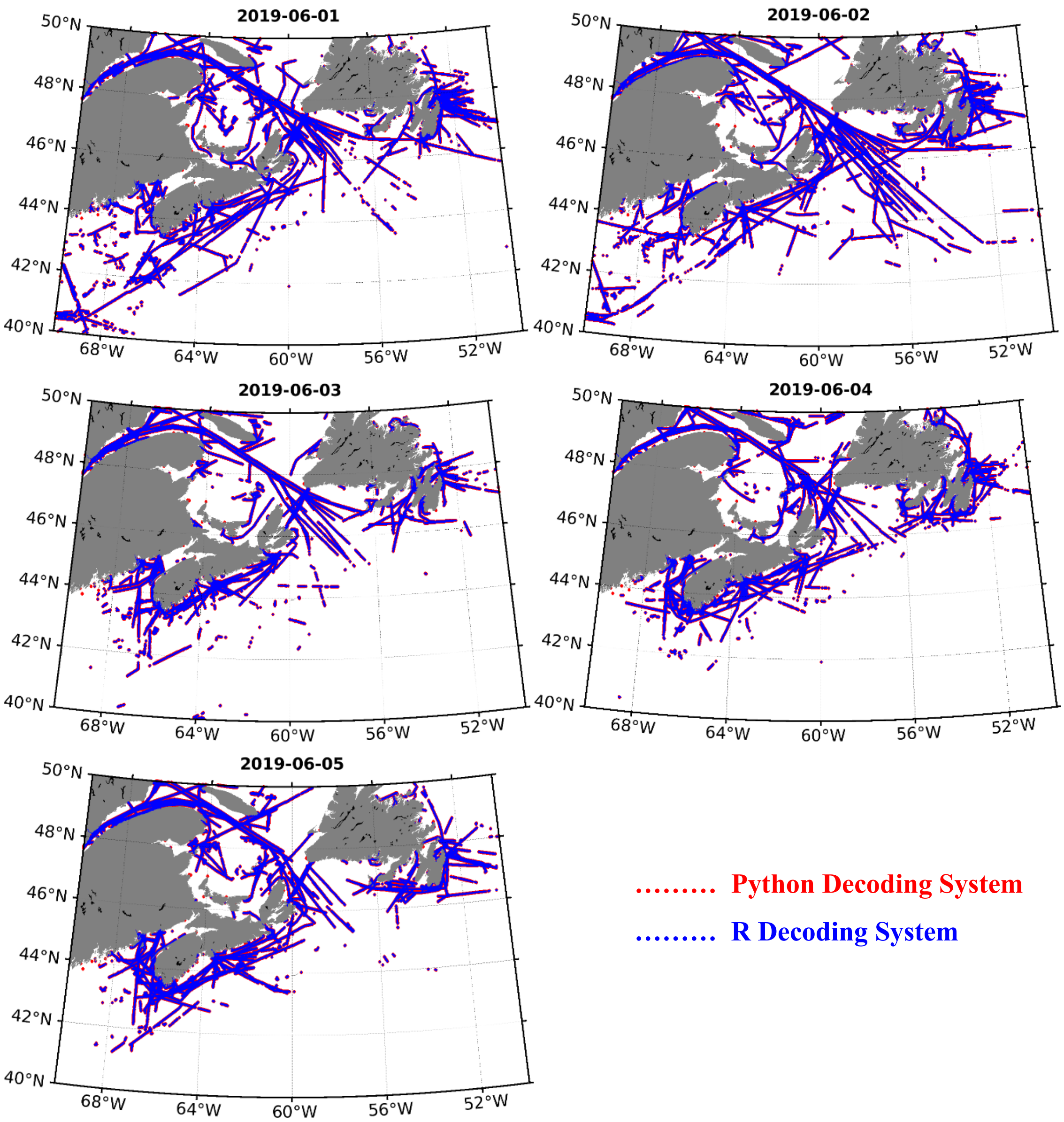


Figure 4. Positions of vessels using decoded data from the Python and R decoding systems from June 1 to June 5 2019.

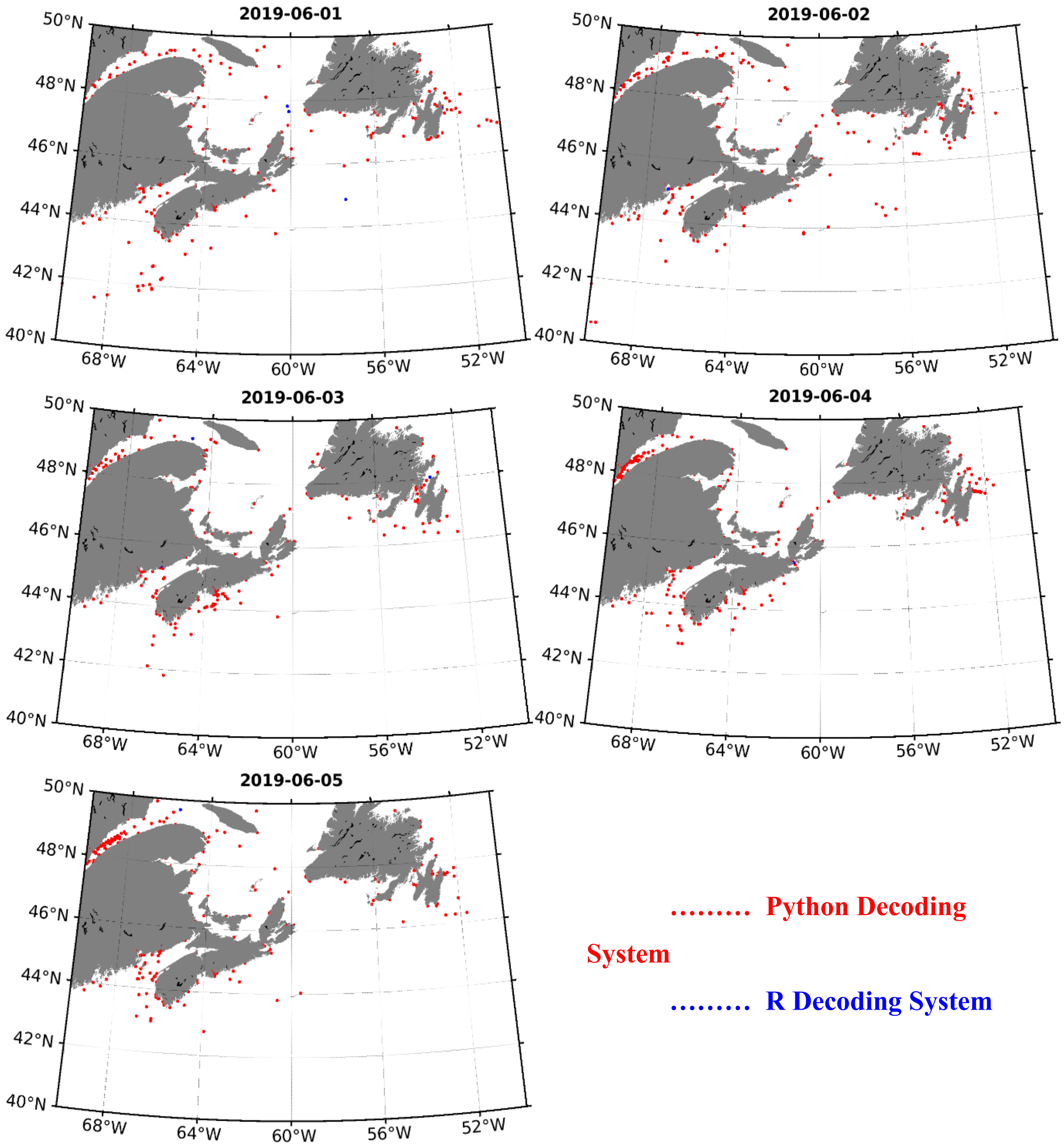


Figure 5. Extra decoded positions resulting from the Python decoding system and R decoding system from June 1 to June 5 2019.

5.2 PYTHON DECODING SYSTEM ISSUES

There are two issues we would like to flag for potential users of the Python decoding system. First, for message type 8, we built the code based on the document found here: https://gpsd.gitlab.io/gpsd/AIVDM.html. We compared our decoded data with the results from the Maritec decoder, and found that our decoded results for water level and current speed were 10 times smaller. The reason for this is unknown and requires further investigation. Second, we were unable to decode the slot binary data for message types 25 and 26 as there was not enough information available to properly do so.

6 SUMMARY

This Python decoding package is capable of decoding AIS message types 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26 and 27, which account for more than 94% of the messages received by the Canadian Coast Guard. The package can be run using serial or parallel methods. The code for message types 6, 8, 25, and 26 is incomplete and will continue to be developed. There are several updates for R decoding system, which include scripts have been modified to ensure all data decoded from a day and able to decode message type 21.

7 ACKNOWLEDGEMENTS

This work was supported by the Oceans Protection Plan - Marine Environmental Quality (OPP-MEQ) project, Ocean and Ecosystem Sciences Division, DFO Maritimes Region. We thank Norman A. Cochrane for providing technical support and his Fortran code for decoding AIS data. We thank Angelia Vanderlaan for providing her R decoding package and resulting decoded data. We also thank Angelia Vanderlaan and Christine Konrad Clarke for their careful and insightful review and efforts towards improving our manuscript.

RFERENCES

Conn, P. B. and Silber, G. K. 2013. Vessel speed restrictions reduce risk of collision-related mortality for North Atlantic right whales. Ecosphere 4: 1-16.

Fournier, M., Hilliard, C., Rezaee, S., Pelot, R. 2018. Past, present and future of the satellite-based automatic identification system: areas of applications (2004-2016). J. Marit. Aff. 17: 311-345,

Government of Canada. 2005. Navigation Safety Regulations. SOR/2020-216. Available from: https://laws-lois.justice.gc.ca/PDF/SOR-2020-216.pdf [accessed 10 February 2021]

Halliday, W. D., Dawson, J., Yurkowski, D.J., Doniol-Valvroze, T., Ferguson, S.H., and et al. 2022. Vessel risks to marine wildlife in the Tallurutiup Imanga National Marine Conservation Area and the eastern entrance to the Northwest Passage. Environmental Science and Policy. 127: 181-195.

Iacarella, J.C., Clyde, G. and Dunham, A. 2020. Vessel tracking datasets for monitoring Canada's conservation effectiveness. Can. Tech. Rep. Fish. Aquat. Sci. 3387: viii + 31 p.

International Maritime Organization (IMO). 2015. Revised guidelines for the onboard operational use of shipborne automatic identification systems (AIS). Resolution A.1106(29). Available from https://wwwcdn.imo.org/localresources/en/KnowledgeCentre/IndexofIMOResolutions/AssemblyDocuments/A.1106(29).pdf [accessed 10 February 2021].

International Telecommunications Union (ITU). 2014. Technical characteristics for an automatic identification system using time-division multiple access in the VHF maritime mobile frequency band. Recommendation ITU-R M.1371-5. Available from https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.1371-5-201402-I!!PDF-E.pdf

R Core Team. 2021. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

Redfern, J. V., McKenna, M. F., Moore, T. J., Calambokidis, J., Deangelis, M. L., Becker, E. A., Barlow, J., Forney, K. A., Fiedler, P. C., and Chivers, S. J. 2013. Assessing the risk of ships striking large whales in marine spatial planning. Conserv. Biol. 27(2): 292-302. doi: 10.1111/cobi.12029.

Redfern, J. V., Hatch, L. T., Caldow, C., DeAngelis, M. L., Gedamke, J., Hastings, S., and et al.

2017. Assessing the Risk of Chronic Shipping Noise to Baleen Whales Off Southern California, USA. Endanger. Species. Res. 32: 153–167. doi: 10.3354/esr00797

Vagle, S., Burnham, R., Thupaki, P., Konrad, C., Toews, S., and Thornton, S.J. 2021. Vessel presence and acoustic environment within Southern Resident Killer Whale (Orcinus orca) critical habitat in the Salish Sea and Swiftsure Bank area. DFO Can. Sci. Advis. Sec. Res. Doc. 2021/058. x + 66 p.

van den Boogaart, K.G., Tolosana-Delgado, R. and Bren, M. 2021. compositions: Compositional data analysis. R package version 2.0-2. https://CRAN.Rproject.org/package=compositions

van der Hoop, J.M., Vanderlaan, A.S. and Taggart, C.T. 2012. Absolute probability estimates of lethal vessel strikes to North Atlantic right whales in Roseway Basin, Scotian Shelf. Ecol. Appl. 22: 2021-2033.

Wickham, H. 2019. stringr: Simple, consistent wrappers for common string operations. R package version 1.4.0. https://CRAN.R-project.org/package=stringr

APPENDIX 1: AIS PAYLOAD INTERPRETATION

Table 6. Class A position report (Messages 1, 2, and 3)

Parameter	Bits	Description
Message type	0-5	Identifier for this message 1, 2 or 3
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more.
MMSI	8-37	MMSI number
Navigational status	38-41	0 = under way using engine, 1 = at anchor, 2 = not under command, 3 = restricted maneuverability, 4 = constrained by her draught, 5 = moored, 6 = aground, 7 = engaged in fishing, 8 = under way sailing, 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), 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), 11 = power-driven vessel towing astern (regional use), 12 = power-driven vessel pushing ahead or towing alongside (regional use), 13 = reserved for future use,14 = AIS-SART (active), MOB-AIS, EPIRB-AIS, 15 = undefined = default (also used by AIS-SART, MOB-AIS and EPIRB-AIS under test)
Rate of turn ROT	42-49	0 = not turning 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.
SOG	50-59	Speed over ground in 1/10 knot steps (0-102.2 knots) 1 023 = not available, 1 022 = 102.2 knots or higher
Position accuracy	60-60	The position accuracy (PA) flag should be determined in accordance with the table below: 1 = high (<= 10 m) 0 = low (> 10 m) 0 = default
Longitude	61-88	Longitude in 1/10 000 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement). 181°= (0x6791AC0 hex) = not available = default)
Latitude	89-115	Latitude in 1/10 000 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement). 91° (0x3412140 hex) = not available = default)
COG	116-127	Course over ground in 1/10 = (0-3599). 3600 (0xE10) = not available = default. 3 601-4 095 should not be used
True heading	128-136	Degrees (0-359) (511 indicates not available = default)
Time stamp	137-142	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)
Special maneuvre indicator	143-144	0 = not available = default 1 = not engaged in special maneuver 2 = engaged in special maneuver (i.e.: regional passing arrangement on Inland Waterway)

Spare	145-147	Not used. Should be set to zero. Reserved for future use.
RAIM-flag	148-148	Receiver autonomous integrity monitoring (RAIM) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use. See Table
Communication state (see below)	149-167	See Rec. ITU-R M.1371-5 Table 49
Number of bits	168	

Table 7. Base station report (Message 4) and UTC/Date response (Message 11)

Parameter	Bits	Description
Message type	0-5	Identifier for this Message 4 or 11 4 = UTC and position report from base station: 11 = UTC and position response from mobile station
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
UTC year	38-51	1-9999; 0 = UTC year not available = default
UTC month	52-55	1-12; 0 = UTC month not available = default; 13-15 not used
UTC day	56-60	1-31; 0 = UTC day not available = default
UTC hour	61-65	0-23; 24 = UTC hour not available = default; 25-31 not used
UTC minute	66-71	0-59; 60 = UTC minute not available = default; 61-63 not used
UTC second	72-77	0-59; 60 = UTC second not available = default; 61-63 not used
Position accuracy	78-78	1 = high (<= 10 m) 0 = low (>10 m) 0 = default
Longitude	79-106	Longitude in 1/10 000 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement); 181 = ($6791AC0_h$) = not available = default)
Latitude	107-133	Latitude in 1/10 000 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement); 91 = (3412140_h) = not available = default)
Type of electronic position fixing device	134-137	Use of differential corrections is defined by field position accuracy above: 0 = undefined (default) 1 = global positioning system (GPS) 2 = GNSS (GLONASS) 3 = combined GPS/GLONASS 4 = Loran-C 5 = Chayka 6 = integrated navigation system 7 = surveyed 8 = Galileo 9-14 = not used 15 = internal GNSS
Spare	138-147	Not used. Should be set to zero. Reserved for future use
RAIM-flag	148-148	RAIM (Receiver autonomous integrity monitoring) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use see Table 50
Communication state	149-167	SOTDMA communication state
Number of bits	168	

Table 8. Ship static and voyage related data (Message 5)

Parameter	Bits	Description
Message type	0-5	Identifier for this Message
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
AIS version indicator	38-39	0 = station compliant with Recommendation ITU-R M.1371-1 1 = station compliant with Recommendation ITU-R M.1371-3 (or later) 2 = station compliant with Recommendation ITU-R M.1371-5 (or later) 3 = station compliant with future editions
IMO number	40-69	0 = not available = default – Not applicable to SAR aircraft 0000000001-0000999999 not used 0001000000-0009999999 = valid IMO number; 0010000000-1073741823 = official flag state number.
Call sign	70-111	7 six-bit ASCII characters, @@@@@@@ = not available = default Craft associated with a parent vessel, should use "A" followed by the last 6 digits of the MMSI of the parent vessel. Examples of these craft include towed vessels, rescue boats, tenders, lifeboats and liferafts.
Name	112-231	Maximum 20 six-bit ASCII characters "@@@@@@@@@@@@@@@@@@@@" = not available = default The Name should be as shown on the station radio license. For SAR aircraft, it should be set to "SAR AIRCRAFT NNNNNNN" where NNNNNNN equals the aircraft registration number.
Type of ship and cargo type	232-239	0 = not available or no ship = default 1-19 = Reserved for future use, 20 =Wing in ground (WIG), all ships of this type, 21-24 = Wing in ground (WIG), Hazardous category A-D, 25-29 = Wing in ground (WIG), Reserved for future use, 30 = Fishing, 31= Towing, 32= Towing: length exceeds 200m or breadth exceeds 25m, 33= Dredging or underwater ops, 34= Diving ops, 35= Military ops, 36= Sailing, 37= Pleasure Craft, 38-39= Reserved, 40= High speed craft (HSC), all ships of this type, 41-44= High speed craft (HSC), Hazardous category A-D, 45-48= High speed craft (HSC), Reserved for future use, 49=High speed craft (HSC), No additional information, 50= Pilot Vessel, 51= Search and Rescue vessel, 52= Tug, 53= Port Tender, 54= Anti-pollution equipment, 55= Law Enforcement, 56-57= Spare - Local Vessel, 58=Medical Transport, 59= Noncombatant ship according to RR Resolution No. 18, 60= Passenger, all ships of this type, 61-64=Passenger, Hazardous category A-D, 65-58= Passenger, Reserved for future use, 69=Passenger, No additional information, 70=Cargo, all ships of this type, 71-74=Cargo, Hazardous category A-D, 75-78=Cargo, Reserved for future use, 79=Cargo, No additional information, 80=Tanker, all ships of this type, 81-84=Tanker, Hazardous category A-D, 85-88= Tanker, Reserved for future use, 89= Tanker, No additional information, 90= Other Type, all ships of this type, 91-94=Other Type, Hazardous category A-D, 95-98= Other Type, Reserved for future use, 99= Other Type, no additional information 100-199 = reserved, for regional use 200-255 = reserved, for future use, Not applicable to SAR aircraft
Dimension to Bow	240-248	Meters
Dimension to Stern	249-257	Meters
Dimension to Port	258-263	Meters
Dimension to Starboard	264-269	Meters
	270-273	0 = undefined (default) 1 = GPS

Type of electronic position fixing device		2 = GLONASS 3 = combined GPS/GLONASS 4 = Loran-C 5 = Chayka 6 = integrated navigation system 7 = surveyed 8 = Galileo, 9-14 = not used 15 = internal GNSS
ETA	274-293	Estimated time of arrival; MMDDHHMM UTC Bits 274-277: month; 1-12; 0 = not available = default Bits 278-282: day; 1-31; 0 = not available = default Bits 283-287: hour; 0-23; 24 = not available = default Bits 288-293: minute; 0-59; 60 = not available = default For SAR aircraft, the use of this field may be decided by the responsible administration
Maximum present static draught	294-301	In 1/10 m, 255 = draught 25.5 m or greater, 0 = not available = default; in accordance with IMO Resolution A.851 Not applicable to SAR aircraft, should be set to 0
Destination	302-421	Maximum 20 characters using 6-bit ASCII; @@@@@@@@@@@@@@@@@@@@ = not available For SAR aircraft, the use of this field may be decided by the responsible administration
DTE	422-422	Data terminal equipment (DTE) ready (0 = available, 1 = not available = default)
Spare	423-423	Spare. Not used. Should be set to zero. Reserved for future use.
Number of bits	424	Occupies 2 slots

Table 9. Addressed binary message (Message 6)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 6
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; default = 0; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station
Sequence number	38-39	0-3
Destination MMSI	40-69	MMSI number of destination station
Retransmit flag	70	Retransmit flag should be set upon retransmission: 0 = no retransmission = default; 1 = retransmitted
Spare	71	Not used. Should be zero. Reserved for future use
Designated Area Code	72-81	Unsigned integer
Functional ID	82-87	Unsigned integer
Binary data	88-1007	Binary data May be shorter than 920 bits. For decoding this part, check https://gpsd.gitlab.io/gpsd/AIVDM.html#_type_6_binary_addressed_message
Number of bits	Maximum 1 008	

Table 10. Binary acknowledge (Message 7)

Parameter	Bits	Description
Message type	0-5	Constant: 7
Repeat indicator	6-7	As in common navigation block, Used by the repeater to indicate how many times a message has been repeated.
Source MMSI	8-37	9 decimal digits
Spare	38-39	Not used
MMSI number 1	40-69	9 decimal digits
Sequence for MMSI 1	70-71	Not used
MMSI number 2	72-101	9 decimal digits
Sequence for MMSI 2	102-103	Not used
MMSI number 3	104-133	9 decimal digits
Sequence for MMSI 3	134-135	Not used
MMSI number 4	136-165	9 decimal digits
Sequence for MMSI 4	166-167	Not used
Number of bits	72-168	

Table 11. Addressed binary message (Message 8)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 8
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; default = 0; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station
Spare	38-39	0-3
Designated Area Code	40-49	Unsigned integer
Functional ID	50-55	Unsigned integer
Binary data	56-1007	Binary data May be shorter than 952 bits. For decoding this part, check https://gpsd.gitlab.io/gpsd/AIVDM.html#_type_8_binary_broadcast_message
Number of bits	Maximum 1 008	

Table 12. Standard SAR aircraft position report (Message 9)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 9; always 9
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
Altitude (GNSS)	38-49	Altitude (derived from GNSS or barometric (see altitude sensor parameter below)) (m) (0-4 094 m) 4 095 = not available, 4 094 = 4 094 m or higher
SOG	50-59	Speed over ground in knot steps (0-1 022 knots) 1 023 = not available, 1 022 = 1 022 knots or higher
Position accuracy	60-60	1 = high (<=10 m) 0 = low (>10 m) 0 = default
Longitude	61-88	Longitude in 1/10 000 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement). 181°= (0x6791AC0 hex) = not available = default)

Latitude	89-115	Latitude in 1/10 000 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement). 91° (0x3412140 hex) = not available = default)
COG	116-127	Course over ground in 1/10 = (0-3 599). 3 600 (0xE10) = not available = default; 3 601-4 095 should not be used
Time stamp	128-133	UTC second when the report was generated by the 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)
Regional reserved	134-141	Reserved
DTE	142-142	Data terminal ready (0 = available 1 = not available = default)
Spare	143-145	Not used. Should be set to zero. Reserved for future use
Assigned mode flag	146-146	0 = Station operating in autonomous and continuous mode = default 1 = Station operating in assigned mode
RAIM-flag	147-147	RAIM flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use
Communication state	148-167	SOTDMA communication state if communication state selector flag is set to 0, or ITDMA communication state if communication state selector flag is set to 1
Number of bits	168	

Table 13. UTC/Date Inquiry (Message 10)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 10; always 10
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated; 0-3; 0 = default; 3 = do not repeat anymore; should be 0 for "CS" transmissions
Source MMSI	8-37	MMSI number
Spare	38-39	Not used
Destination MMSI	40-69	9 decimal digits
Spare	70-71	Not used
Number of bits	72	

Table 14. Addressed safety-related message (Message 12)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 12; always 12
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source ID	8-37	MMSI number of station which is the source of the message.
Sequence number	38-39	0-3
Destination MMSI	40-69	MMSI number of station which is the destination of the message
Retransmit flag	70	Retransmit flag should be set upon retransmission: 0 = no retrans-mission = default; 1 = retransmitted
Spare	71	Not used. Should be zero. Reserved for future use
Safety related text	72-1007	1-156 six-bit ASCII characters. May be shorter than 936 bits.
Number of bits	Maximum 1 008	Occupies up to 3 slots, or up to 5 slots when able to use FATDMA reservations. For Class B "SO" mobile AIS stations the length of the message should not exceed 3 slots For Class B "CS" mobile AIS stations the length of the message should not exceed 1 slot

Table 15. Safety-related broadcast message (Message 14)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 14; always 14.
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station of message
Spare	38-39	Not used. Should be set to zero. Reserved for future use
Safety related text	40-1007	1-161 six-bit ASCII characters. May be shorter than 968 bits.
Number of bits	Maximum 1 008	Occupies up to 3 slots, or up to 5 slots when able to use FATDMA reservations. For Class B “SO” mobile AIS stations the length of the message should not exceed 3 slots For Class B “CS” mobile AIS stations the length of the message should not exceed 1 slot

Table 16. Interrogation (Message 15)

Parameter	Bits	Description
Message Type	0-5	Identifier for Message 15; always 15.
Repeat Indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station of message
Spare	38-39	Not used. Should be set to zero. Reserved for future use
Interrogated MMSI	40-69	9 decimal digits
First message type	70-75	Unsigned integer
First slot offset	76-87	Unsigned integer
Spare	88-89	Not used. Should be set to zero. Reserved for future use
Second message type	90-95	Unsigned integer
Second slot offset	96-107	Unsigned integer
Spare	108-109	Not used. Should be set to zero. Reserved for future use
Interrogated MMSI	110-139	9 decimal digits
First message type	140-145	Unsigned integer
First slot offset	146-157	Unsigned integer
Spare	158-159	Not used. Should be set to zero. Reserved for future use
Number of bits	88-160	There are four use cases for this message. A decoder must dispatch on the length of the data packet to determine which it is seeing: 1. One station is interrogated for one message type. Length is 88 bits. 2. One station is interrogated for two message types, Length is 110 bits. There is a design error in the standard here; according to the ITU1371 requirement for padding to 8 bits, this should have been 112 with a 4-bit trailing spare field, and decoders should be prepared to handle that length as well. See the discussion of byte alignment elsewhere in this document for context. 3. Two stations are interrogated for one message type each. Length is 160 bits. The second message type and second slot offset associated with the first queried MMSI should be zeroed. 4. One station is interrogated for two message types, and a second for one message type. Length is 160 bits.

Table 17. Assignment mode command (Message 16)

Parameter	Bits	Description
Message Type	0-5	Identifier for Message 16; always 16.
Repeat Indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station of message
Spare	38-39	Not used. Should be set to zero. Reserved for future use
Destination A MMSI	40-69	9 decimal digits
Offset A	70-81	Check ITU-R M.1371-5 for details
Increment A	82-91	Check ITU-R M.1371-5 for details
Destination B MMSI	92-121	9 decimal digits
Offset B	122-133	Check ITU-R M.1371-5 for details
Increment B	134-143	Check ITU-R M.1371-5 for details
Number of bits	96 or 144	If the message is 96 bits long, it should be interpreted as an assignment for a single station (92 bits) followed by 4 bits of padding reserved for future use. If the message is 144 bits long it should be interpreted as a channel assignment for two stations; no padding follows.

Table 18. DGNSS broadcast binary message (Message 17)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 17; always 17
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source MMSI	8-37	MMSI of the base station
Spare	38-39	Spare. Should be set to zero. Reserved for future use
Longitude	40-57	Surveyed longitude of DGNSS reference station in 1/10 min (±180°, East = positive, West = negative). If interrogated and differential correction service not available, the longitude should be set to 181°
Latitude	58-74	Surveyed latitude of DGNSS reference station in 1/10 min (±90°, North = positive, South = negative). If interrogated and differential correction service not available, the latitude should be set to 91°
Spare	75-79	Not used. Should be set to zero. Reserved for future use
Data	80-815	Differential correction data (see below). If interrogated and differential correction service not available, the data field should remain empty (zero bits). This should be interpreted by the recipient as DGNSS data words set to zero
Number of bits	80-816	80 bits: assumes N = 0; 816 bits: assumes N = 29 (maximum value)

Table 19. Standard Class B position report (Message 18)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 18; always 18
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated; 0-3; 0 = default; 3 = do not repeat anymore; should be 0 for “CS” transmissions
MMSI	8-37	MMSI number
Spare	38-45	Not used. Should be set to zero. Reserved for future use
SOG	46-55	Speed over ground in 1/10 knot steps (0-102.2 knots) 1 023 = not available, 1 022 = 102.2 knots or higher
Position accuracy	56-56	1 = high (<= 10 m) 0 = low (> 10 m) 0 = default
Longitude	57-84	Longitude in 1/10 000 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement). 181°= (0x6791AC0 hex) = not available = default)
Latitude	85-111	Latitude in 1/10 000 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement). 91° (0x3412140 hex) = not available = default)
COG	112-123	Course over ground in 1/10= (0-3 599). 3 600 (0xE10) = not available = default; 3 601-4 095 should not be used
True heading	124-132	Degrees (0-359) (511 indicates not available = default)
Time stamp	133-138	UTC second when the report was generated by the 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) 61, 62, 63 are not used by "CS" AIS
Spare	139-140	Not used. Should be set to zero. Reserved for future use
Class B unit flag	141-141	0 = Class B SOTDMA unit 1 = Class B "CS" unit
Class B display flag	142-142	0 = No display available; not capable of displaying Message 12 and 14 1 = Equipped with integrated display displaying Message 12 and 14
Class B DSC flag	143-143	0 = Not equipped with DSC function 1 = Equipped with DSC function (dedicated or time-shared)
Class B band flag	144-144	0 = Capable of operating over the upper 525 kHz band of the marine band 1 = Capable of operating over the whole marine band (irrelevant if "Class B Message 22 flag" is 0)
Class B Message 22 flag	145-145	0 = No frequency management via Message 22 , operating on AIS1, AIS2 only 1 = Frequency management via Message 22
Mode flag	146-146	0 = Station operating in autonomous and continuous mode = default 1 = Station operating in assigned mode
RAIM-flag	147-147	RAIM (Receiver autonomous integrity monitoring) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use
Communication state	148-167	SOTDMA communication state. Because Class B "CS" does not use any Communication State information, this field shall be filled with the following value: 1100000000000000110.
Number of bits	168	Occupies one slot

Table 20. Extended Class B position report (Message 19)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 19; always 19
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
Spare	38-45	Not used. Should be set to zero. Reserved for future use
SOG	46-55	Speed over ground in 1/10 knot steps (0-102.2 knots) 1 023 = not available, 1 022 = 102.2 knots or higher
Position accuracy	56-56	1 = high (> 10 m) 0 = low (< 10 m) 0 = default
Longitude	57-84	Longitude in 1/10 000 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement); 181° (6791AC0h) = not available = default)
Latitude	85-111	Latitude in 1/10 000 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement); 91º = (3412140h) = not available = default)
COG	112-123	Course over ground in 1/10= (0-3 599). 3 600 (E10h) = not available = default; 3 601-4 095 should not be used
True heading	124-132	Degrees (0-359) (511 indicates not available = default)
Time stamp	133-138	UTC second when the report was generated by the 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) Note: CSTDMA devices do not transmit if position information is not available.
Spare	139-142	Not used. Should be set to zero. Reserved for future use
Ship name	143-262	Maximum 20 six-bit ASCII characters. @@@@@@@@@@@@@@@@@@@@ = not available = default
Type of ship and cargo type	263-270	0 = not available or no ship = default 1-19 = Reserved for future use, 20 =Wing in ground (WIG), all ships of this type, 21-24 = Wing in ground (WIG), Hazardous category A-D, 25-29 = Wing in ground (WIG), Reserved for future use, 30 = Fishing, 31= Towing, 32= Towing: length exceeds 200m or breadth exceeds 25m, 33= Dredging or underwater ops, 34= Diving ops, 35= Military ops, 36= Sailing, 37= Pleasure Craft, 38-39= Reserved, 40= High speed craft (HSC), all ships of this type, 41-44= High speed craft (HSC), Hazardous category A-D, 45-48= High speed craft (HSC), Reserved for future use, 49=High speed craft (HSC), No additional information, 50= Pilot Vessel, 51= Search and Rescue vessel, 52= Tug, 53= Port Tender, 54= Anti-pollution equipment, 55= Law Enforcement, 56-57= Spare - Local Vessel, 58=Medical Transport, 59= Noncombatant ship according to RR Resolution No. 18, 60= Passenger, all ships of this type, 61-64=Passenger, Hazardous category A-D, 65-58= Passenger, Reserved for future use, 69=Passenger, No additional information, 70=Cargo, all ships of this type, 71-74=Cargo, Hazardous category A-D, 75-78=Cargo, Reserved for future use, 79=Cargo, No additional information, 80=Tanker, all ships of this type, 81-84=Tanker, Hazardous category A-D, 85-88= Tanker, Reserved for future use, 89= Tanker, No additional information, 90= Other Type, all ships of this type, 91-94=Other Type, Hazardous category A-D, 95-98= Other Type, Reserved for future use, 99= Other Type, no additional information 100-199 = reserved, for regional use 200-255 = reserved, for future use, Not applicable to SAR aircraft

Dimension to Bow	271-279	Meters
Dimension to Stern	280-288	Meters
Dimension to Port	289-294	Meters
Dimension to Starboard	295-300	Meters
Type of electronic position fixing device Provided by Message 24B	301-304	0 = Undefined (default); 1 = GPS, 2 = GLONASS, 3 = combined GPS/GLONASS, 4 = Loran-C, 5 = Chayka, 6 = integrated navigation system, 7 = surveyed; 8 = Galileo, 9-15 = not used
RAIM-flag Provided by Message 18	305-305	RAIM (Receiver autonomous integrity monitoring) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use see Table 47
DTE Provided by Message 18 (Display Flag)	306-306	Data terminal ready (0 = available 1 = not available; = default) (see § 3.3.1)
Assigned mode flag Provided by Message 18 (Display Flag)	307-307	0 = Station operating in autonomous and continuous mode = default 1 = Station operating in assigned mode
Spare	308-311	Not used. Should be set to zero. Reserved for future use
Number of bits	312	Occupies two slots.

Table 21. Data link management message (Message 20)

Parameter	Bits	Description
Message Type	0-5	Identifier for Message 20; always 20
Repeat Indicator	6-7	Used by the repeater to indicate how many times a message has been repeated.; 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
Spare	38-39	Not used. Should be set to zero. Reserved for future use
Offset number 1	40-51	Reserved offset number
Reserved slots	52-55	Consecutive slots
Time-out	56-58	Allocation timeout in minutes
Increment	59-69	Repeat increment
Offset number 2	70-81	Reserved offset number
Reserved slots	82-85	Consecutive slots
Time-out	86-88	Allocation timeout in minutes
Increment	89-99	Repeat increment
Offset number 3	100-111	Reserved offset number
Reserved slots	112-115	Consecutive slots
Time-out	116-118	Allocation timeout in minutes
Increment	119-129	Repeat increment
Offset number 4	130-141	Reserved offset number
Reserved slots	142-145	Consecutive slots
Time-out	146-148	Allocation timeout in minutes
Increment	149-159	Repeat increment
Number of bits	72-160	Length varies from 72-160 depending on the number of slot reservations (1 to 4) in the message.

Table 22. Aids to navigation report (Message 21)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 21
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated.
MMSI	8-37	MMSI number
Type of aids-to-navigation	38-42	0 = not available = default; refer to appropriate definition set up by IALA; see Table below
Name of Aids-to-Navigation (AtoN)	43-162	Maximum 20 six-bit ASCII characters "@@@@@@@@@@@@@@@@@@@@" = not available = default. The name of the AtoN may be extended by the parameter "Name of Aid-to-Navigation Extension" below
Position accuracy	163-163	1 = high (<10 m) 0 = low (>10 m) 0 = default
Longitude	164-191	Longitude in 1/10 000 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement). 181°= (0x6791AC0 hex) = not available = default)
Latitude	192-218	Latitude in 1/10 000 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement). 91° (0x3412140 hex) = not available = default)
Dimension to Bow	219-227	Meters
Dimension to Stern	228-236	Meters
Dimension to Port	237-242	Meters
Dimension to Starboard	243-248	Meters
Type of electronic position fixing device	249-252	0 = Undefined (default) 1 = GPS 2 = GLONASS 3 = Combined GPS/GLONASS 4 = Loran-C 5 = Chayka 6 = Integrated Navigation System 7 = surveyed. For fixed AtoN and virtual AtoN, the charted position should be used. The accurate position enhances its function as a radar reference target 8 = Galileo 9-14 = not used 15 = internal GNSS
Time stamp	253-258	UTC second when the report was generated by the 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)
Off-position indicator	259-259	For floating AtoN, only: 0 = on position; 1 = off position. NOTE 1 – This flag should only be considered valid by receiving station, if the AtoN is a floating aid, and if time stamp is equal to or below 59. For floating AtoN the guard zone parameters should be set on installation
AtoN status	260-267	Reserved for the indication of the AtoN status 0 = Default, Type of AtoN not specified, 1= Reference point, 2= RACON (radar transponder marking a navigation hazard), 3= Fixed structure off shore, such as oil platforms, wind farms, rigs. (Note: This code should identify an obstruction that is fitted with an Aid-to-Navigation AIS station.), 4= Emergency Wreck Marking Buoy, 5= Light, without sectors, 6= Light, with sectors, 7= Leading Light Front,

		8= Leading Light rear, 9= Beacon, Cardinal N, 10= Beacon, Cardinal E, 11= Beacon, Cardinal S, 12= Beacon, Cardinal W, 13= Beacon, Port hand, 14= Beacon, Starboard hand, 15= Beacon, Preferred Channel port hand, 16= Beacon, Preferred Channel starboard hand, 17= Beacon, Isolated danger, 18= Beacon, Safe water, 19= Beacon, Special mark, 20= Cardinal Mark N, 21= Cardinal Mark E, 22= Cardinal Mark S, 23= Cardinal Mark W, 24= Port hand Mark, 25= Starboard hand Mark, 26= Preferred Channel Port hand, 27= Preferred Channel Starboard hand, 28= Isolated danger, 29= Safe Water, 30= Special Mark, 31= Light Vessel / LANBY / Rigs
RAIM-flag	268-268	RAIM (Receiver autonomous integrity monitoring) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use see Table 50
Virtual AtoN flag	269-269	0 = default = real AtoN at indicated position; 1 = virtual AtoN, does not physically exist.
Assigned mode flag	270-270	0 = Station operating in autonomous and continuous mode = default 1 = Station operating in assigned mode
Spare	271-271	Spare. Not used. Should be set to zero. Reserved for future use
Name of Aid-to-Navigation Extension	272-359	This parameter of up to 14 additional 6-bit-ASCII characters for a 2-slot message may be combined with the parameter "Name of Aid-to-Navigation" at the end of that parameter, when more than 20 characters are needed for the name of the AtoN. This parameter should be omitted when no more than 20 characters for the name of the A-to-N are needed in total. Only the required number of characters should be transmitted, i.e. no @-character should be used
Number of bits	272-360	Occupies two slots

Table 23. Channel management (Message 22)

Parameter	Bits	Description
Message Type	0-5	Identifier for Message 21
Repeat Indicator	6-7	Used by the repeater to indicate how many times a message has been repeated.
MMSI	8-37	MMSI number
Spare	38-39	Not used
Channel A	40-51	Channel number
Channel B	52-63	Channel number
Tx/Rx mode	64-67	Transmit/receive mode
Power	68-68	Low=0, high=1
NE Longitude	69-86	NE longitude to 0.1 minutes
NE Latitude	87-103	NE latitude to 0.1 minutes
SW Longitude	104-121	SW longitude to 0.1 minutes
SW Latitude	122-138	SW latitude to 0.1 minutes
MMSI1	69-98	MMSI of destination 1
MMSI2	104-133	MMSI of destination 2
Addressed	139-139	0=Broadcast, 1=Addressed
Channel A Band	140-140	0=Default, 1=12.5kHz
Channel B Band	141-141	0=Default, 1=12.5kHz
Zone size	142-144	Size of transitional zone
Spare	145-167	Reserved for future use
Number of bits	168	If the message is broadcast (addressed field is 0), the NE Longitude, NE Latitude, SW Longitude, and SW Latitude fields are the corners of a rectangular jurisdiction area over which control parameters are to be set. If it is addressed (addressed field is 1), the same span of data is interpreted as two 30-bit MMSIs beginning at bit offsets 69 and 104 respectively.

Table 24. Group assignment command (Message 23)

Parameter	Bits	Description
Message Type	0-5	Identifier for Message 23
Repeat Indicator	6-7	Used by the repeater to indicate how many times a message has been repeated.
MMSI	8-37	MMSI number
Spare	38-39	Not used
NE Longitude	40-57	NE longitude to 0.1 minutes
NE Latitude	58-74	NE latitude to 0.1 minutes
SW Longitude	75-92	SW longitude to 0.1 minutes
SW Latitude	93-109	SW latitude to 0.1 minutes
Station type	110-113	0= All types of mobiles (default), 1= Reserved for future use, 2= All types of Class B mobile stations, 3= SAR airborne mobile station, 4= Aid to Navigation station, 5= Class B shipborne mobile station (IEC62287 only), 6-9= Regional use and inland waterways, 10-15= Reserved for future use
Ship Type	114-121	0 = not available or no ship = default 1-19 = Reserved for future use, 20 =Wing in ground (WIG), all ships of this type, 21-24 = Wing in ground (WIG), Hazardous category A-D, 25-29 = Wing in ground (WIG), Reserved for future use, 30 = Fishing, 31= Towing, 32= Towing: length exceeds 200m or breadth exceeds 25m, 33= Dredging or underwater ops, 34= Diving ops, 35= Military ops, 36= Sailing, 37= Pleasure Craft, 38-39= Reserved, 40= High speed craft (HSC), all ships of this type, 41-44= High speed craft (HSC), Hazardous category A-D, 45-48= High speed craft (HSC), Reserved for future use, 49=High speed craft (HSC), No additional information, 50= Pilot Vessel, 51= Search and Rescue vessel, 52= Tug, 53= Port Tender, 54= Anti-pollution equipment, 55= Law Enforcement, 56-57= Spare - Local Vessel, 58=Medical Transport, 59= Noncombatant ship according to RR Resolution No. 18, 60= Passenger, all ships of this type, 61-64=Passenger, Hazardous category A-D, 65-58= Passenger, Reserved for future use, 69=Passenger, No additional information, 70=Cargo, all ships of this type, 71-74=Cargo, Hazardous category A-D, 75-78=Cargo, Reserved for future use, 79=Cargo, No additional information, 80=Tanker, all ships of this type, 81-84=Tanker, Hazardous category A-D, 85-88= Tanker, Reserved for future use, 89= Tanker, No additional information, 90= Other Type, all ships of this type, 91-94=Other Type, Hazardous category A-D, 95-98= Other Type, Reserved for future use, 99= Other Type, no additional information 100-199 = reserved, for regional use
Spare	122-143	Not used
Tx/Rx Mode	144-145	0 = TxA/TxB, RxA/RxB (default), 1= TxA, RxA/RxB, 2= TxB, RxA/RxB, 3= Reserved for Future Use
Report Interval	146-149	0= As given by the autonomous mode, 1=10 Minutes, 2=6 Minutes, 3=3 Minutes, 4=1 Minutes, 5=30 Seconds, 6= 15 Seconds, 7=10 Seconds, 8=5 Seconds, 9= Next Shorter Reporting Interval, 10= Next Longer Reporting Interval
Quiet Time	150-153	0 = none, 1-15 quiet time in minutes
Spare	154-159	Not used
Number of bits	160	This message is intended to be broadcast by a competent authority (an AIS network-control base station) to set operational parameters for all mobile stations in an AIS coverage region.

Table 25. Static data report (Message 24A)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 24; always 24
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
Part number	38-39	Identifier for the message part number; always 0 for Part A
Name	40-159	Name of the MMSI-registered vessel. Maximum 20 six-bit ASCII characters, @@@@@@@@@@@@@@@@@@@@ = not available = default For SAR aircraft, it should be set to “SAR AIRCRAFT NNNNNNN” where NNNNNNN equals the aircraft registration number
Number of bits	160	Occupies one-time period

Table 26. Static data report (Message 24B)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 24; always 24
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
Part number	38-39	Identifier for the message part number; always 1 for Part B
Type of ship and cargo type	40-47	0 = not available or no ship = default 1-19 = Reserved for future use, 20 =Wing in ground (WIG), all ships of this type, 21-24 = Wing in ground (WIG), Hazardous category A-D, 25-29 = Wing in ground (WIG), Reserved for future use, 30 = Fishing, 31= Towing, 32= Towing: length exceeds 200m or breadth exceeds 25m, 33= Dredging or underwater ops, 34= Diving ops, 35= Military ops, 36= Sailing, 37= Pleasure Craft, 38-39= Reserved, 40= High speed craft (HSC), all ships of this type, 41-44= High speed craft (HSC), Hazardous category A-D, 45-48= High speed craft (HSC), Reserved for future use, 49=High speed craft (HSC), No additional information, 50= Pilot Vessel, 51= Search and Rescue vessel, 52= Tug, 53= Port Tender, 54= Anti-pollution equipment, 55= Law Enforcement, 56-57= Spare - Local Vessel, 58=Medical Transport, 59= Noncombatant ship according to RR Resolution No. 18, 60= Passenger, all ships of this type, 61-64=Passenger, Hazardous category A-D, 65-58= Passenger, Reserved for future use, 69=Passenger, No additional information, 70=Cargo, all ships of this type, 71-74=Cargo, Hazardous category A-D, 75-78=Cargo, Reserved for future use, 79=Cargo, No additional information, 80=Tanker, all ships of this type, 81-84=Tanker, Hazardous category A-D, 85-88= Tanker, Reserved for future use, 89= Tanker, No additional information, 90= Other Type, all ships of this type, 91-94=Other Type, Hazardous category A-D, 95-98= Other Type, Reserved for future use, 99= Other Type, no additional information 100-199 = reserved, for regional use 200-255 = reserved, for future use, Not applicable to SAR aircraft
Vendor ID	48-65	Unique identification of the Unit by a number as defined by the manufacturer (option; "@@@@@@@" = not available = default)
Unit model code	66-69	
Serial number	70-89	
Call sign	90-131	Call sign of the MMSI-registered vessel. 7 X 6 bit ASCII characters, "@@@@@@@" = not available = default Craft associated with a parent vessel should use “A” followed by the

		last 6 digits of the MMSI of the parent vessel. Examples of these craft include towed vessels, rescue boats, tenders, lifeboats and life rafts
Dimension to Bow	132-140	Meters
Dimension to Stern	141-149	Meters
Dimension to Port	150-155	Meters
Dimension to Starboard	156-161	Meters
Mothership MMSI	132-161	Interpretation of the 30 bits 132-162 in Part B is variable. If the MMSI at 8-37 is that of an auxiliary craft, the entry is taken to refer to a small attached auxiliary vessel and these 30 bits are read as the MMSI of the mother ship. Otherwise the 30 bits describe vessel dimensions as in Message Type 5.
Spare	162-167	Spare. Not used. Should be set to zero. Reserved for future use
Number of bits	168	Occupies one-time period

Table 27. Single slot binary message (Message 25)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 25; always 25
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station
Destination indicator	38	0 = Broadcast (no Destination MMSI field used) 1 = Addressed (Destination MMSI uses 30 data bits for MMSI)
Binary data flag	39	0 = unstructured binary data (no Application Identifier bits used) 1 = binary data coded as defined by using the 16-bit Application identifier
Destination MMSI	40-70	Destination MMSI (if used) , If Destination indicator = 0 (Broadcast); no data bits are needed for the Destination MMSI. If Destination indicator = 1; 30 bits are used for Destination MMSI and spare bits for byte alignment.
Application ID	40-56 or 70-86	If the 'structured' flag is on, a 16-bit application identifier is extracted; this field is to be interpreted as a 10 bit DAC and 6-bit FID as in message types 6 and 8. Otherwise that field span becomes part of the message payload.
Binary data	40-167	Maximum of 128 bits.
Number of bits	168	Maximum of 168 bits (a single slot). Fields after the Destination MMSI are at variable offsets depending on that flag and the Destination Indicator; they always occur in the same order but some may be omitted.

Table 28. Multi slot binary message (Message 26)

Parameter	Bits	Description
Message type	0-5	Identifier for Message 26; always 26
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
Source MMSI	8-37	MMSI number of source station
Destination indicator	38	0 = Broadcast (no Destination MMSI field used) 1 = Addressed (Destination MMSI uses 30 data bits for MMSI)
Binary data flag	39	0 = unstructured binary data (no Application Identifier bits used) 1 = binary data coded as defined by using the 16-bit Application identifier
Destination MMSI	40-70	Destination MMSI (if used) , If Destination indicator = 0 (Broadcast); no data bits are needed for the Destination MMSI. If Destination indicator = 1; 30 bits are used for Destination MMSI and spare bits for byte alignment.
Designated Area Code	40-50 or 70-80	If the 'structured' flag is on, a 16-bit application identifier is extracted; this field is to be interpreted as a 10 bit DAC and 6-bit FID as in message types 6 and 8. Otherwise that field span becomes part of the message payload.
Application ID	?	The data field may span up to five 224-bit slots in addition to the tail end of the base slot. The Application ID field, if present, is to be interpreted as a 10 bit DAC and 6-bit FID as in message types 6 and 8.
Binary data	?	
Radio status	?	The 20 radio status bits are always present after end-of-data in the last slot. The radio status is 20 bits rather than 19 because an extra first bit selects whether it should be interpreted as a SOTDMA or ITDMA state.
Number of bits	Maximum 1 064	Occupies up to 3 slots, or up to 5 slots when able to use FATDMA reservations. For Class B “SO” mobile AIS stations the length of the message should not exceed 3 slots. Class B “CS” mobile AIS stations should not transmit

Table 29. Long-range AIS broadcast message (Message 27)

Parameter	Bits	Description
Message type	0-5	Identifier for this message; always 27
Repeat indicator	6-7	Used by the repeater to indicate how many times a message has been repeated. 0-3; 0 = default; 3 = do not repeat any more
MMSI	8-37	MMSI number
Position accuracy	38-38	The position accuracy (PA) flag should be determined in accordance with the table below: 1 = high (<= 10 m) 0 = low (> 10 m) 0 = default
RAIM flag	39-39	Receiver autonomous integrity monitoring (RAIM) flag of electronic position fixing device; 0 = RAIM not in use = default; 1 = RAIM in use. See Table
Navigational status	40-43	0 = under way using engine, 1 = at anchor, 2 = not under command, 3 = restricted maneuverability, 4 = constrained by her draught, 5 = moored, 6 = aground, 7 = engaged in fishing, 8 = under way sailing, 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), 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), 11 = power-driven vessel towing astern (regional use), 12 = power-driven vessel pushing ahead or towing alongside (regional use), 13 = reserved for future use,14 = AIS-SART (active), MOB-AIS, EPIRB-AIS, 15 = undefined = default (also used by AIS-SART, MOB-AIS and EPIRB-AIS under test)
Longitude	44-61	Longitude in 1/10 min (±180°, East = positive (as per 2's complement), West = negative (as per 2's complement), 181° = position older than 6 hours or not available = default)
Latitude	62-78	Latitude in 1/10 min (±90°, North = positive (as per 2's complement), South = negative (as per 2's complement), 91° = position older than 6 hours or not available = default)
SOG	79-84	Knots (0-62); 63 = not available = default
COG	85-93	Degrees (0-359); 511 = not available = default
Position Latency	94-94	0 = Reported position latency is less than 5 seconds; 1 = Reported position latency is greater than 5 seconds = default
Spare	95-95	Set to zero, to preserve byte boundaries
Number of bits	96	

APPENDIX 2: PYTHON CODE

PROCESS_AIS_SERIAL.PY

```python
#!/usr/bin/env python
"""
The main process to deal with the AIS messages from Canadian Coast Guard:

To run the code need to set up the input and output folders!

The format of the .txt file should be
    c:1506815999,C:2234,s:P-Calvert*4F
    !AIVDM,1,1,9,B,H4eGD9PP5=@D000000000000000,2*01
The format of the .csv file should be
    \c:1573012863,C:135,s:P-Gil*6F\!AIVDM,1,1,2,B,34Vf7j1000njujHNq31<sP840Dg:,0*28
Otherwise the code can't work

Author: Lanli Guo
Publish date: May 15, 2020
"""
###############################################################################
from Second_layer_NMEA import *
from Output_format import *
from Read_AIS import *
from Write_netcdf import *
import time
import os
import csv

###############File Directory################################################
####Need to set up!!!
Input_Directory  = '../Input'
####Need to set up!!!
Output_Directory = 'Output'
####Need to set up!!!
Outlog_Directory = 'Outlog'

def lineDecode(ais_data,Station_region,Station_name,Station_time,dt_time,staregion_dyn,staname_dyn,
               datenum_dyn,type_dyn,repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,
               lat_dyn,course_dyn,heading_dyn,Dindex,staregion_sta,staname_sta,datenum_sta,
               type_sta,repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
               starboard_sta,bow_sta,draught_sta,destination_sta,Sindex):

    if ais_data['type'] == 19:
        ais_format = format_ais(ais_data,'dyn')
        if ais_format == None:
            pass
        else:
            ais_format['Station region'] = Station_region
            ais_format['Station name']   = Station_name
            ais_format['Station time']   = Station_time
            ais_format['Datenum']        = dt_time

            read_dyn(ais_format,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
                     repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,
                     lat_dyn,course_dyn,heading_dyn,Dindex)

        ais_format = format_ais(ais_data,'sta')
        if ais_format == None:
                pass
        else:
            ais_format['Station region'] = Station_region
            ais_format['Station name']   = Station_name
            ais_format['Station time']   = Station_time
            ais_format['Datenum']        = dt_time

            read_sta(ais_format,staregion_sta,staname_sta,datenum_sta,type_sta,
                     repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
                     starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

    elif ais_data['type'] == 5 or ais_data['type'] == 24:
        ais_format = format_ais(ais_data,'sta')
        if ais_format == None:
            pass
        else:
            ais_format['Station region'] = Station_region
            ais_format['Station name']   = Station_name
            ais_format['Station time']   = Station_time
```

```
            ais_format['Datenum']         = dt_time

            read_sta(ais_format,staregion_sta,staname_sta,datenum_sta,type_sta,
                     repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
                     starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

    else:
        ais_format = format_ais(ais_data,'dyn')
        if ais_format == None:
            pass
        else:
            ais_format['Station region'] = Station_region
            ais_format['Station name']   = Station_name
            ais_format['Station time']   = Station_time
            ais_format['Datenum']        = dt_time

            read_dyn(ais_format,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
                     repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,
                     course_dyn,heading_dyn,Dindex)
    return

def main():
    for f_name in os.listdir(Input_Directory):
        infile  = Input_Directory+ "\ ".strip()+f_name
        out_dyn = Output_Directory+ "\ ".strip()+'Dynamic_'+f_name[:-4]+'.nc'
        out_sta = Output_Directory+ "\ ".strip()+'Static_'+f_name[:-4]+'.nc'
        out_log = Outlog_Directory+ "\ ".strip()+'Outlog_'+f_name[:-4]+'.txt'

        staregion_dyn = []
        staname_dyn   = []
        datenum_dyn   = []
        type_dyn      = []
        repeat_dyn    = []
        mmsi_dyn      = []
        status_dyn    = []
        speed_dyn     = []
        accuracy_dyn  = []
        lon_dyn       = []
        lat_dyn       = []
        course_dyn    = []
        heading_dyn   = []
        Dindex        = 0

        staregion_sta = []
        staname_sta   = []
        datenum_sta   = []
        type_sta      = []
        repeat_sta    = []
        mmsi_sta      = []
        shipname_sta  = []
        shiptype_sta  = []
        stern_sta     = []
        port_sta      = []
        starboard_sta = []
        bow_sta       = []
        draught_sta   = []
        destination_sta = []
        Sindex        = 0

    #######To deal with .txt file################################################
        if f_name.endswith('.txt'):

            fin = open(infile, "r")
            Station_region = 'N/A'
            Station_name   = 'N/A'
            Station_time   = 'N/A'
            dt_time        = -9999

            for x in fin:
                if x.split(',')[0][0:2]=='c:' or x.split(',')[0][0:2]=='s:' or x.split(',')[0][0:2]=='C:':
                    Station_time   = 'N/A'
                    dt_time        = -9999
                    x  = x.replace("*",",")
                    x0 = x.replace("c:","$")
                    x1 = x0.split('$')[1]
                    x2 = x1.split(',')[0]
                    dt_obj=time.gmtime(float(x2))
                    dt_time = int(x2)
                    Station_time = time.strftime("%d-%b-%Y %H:%M:%S",dt_obj)

                    s0 = x.replace("s:","$")
```

```
                    if s0.find('-') != -1:
                        s1 = s0.split('$')[1]
                        s2 = s1.split(',')[0]
                        Station_region = s2.split('-')[0]
                        Station_name   = s2.split('-')[1]
                    else:
                        Station_region = 'N/A'
                        Station_name   = 'N/A'

                elif x.split(',')[0][0:5]=='!AIVD' and len(x.split(','))>6:
                    msg = x.rstrip('\n')
                    ais_data = decod_ais(msg,out_log)
                    if ais_data == None:
                        pass
                    else:
                        if len(ais_data)>4:
                            output =
lineDecode(ais_data,Station_region,Station_name,Station_time,dt_time,staregion_dyn,staname_dyn,

datenum_dyn,type_dyn,repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,

lat_dyn,course_dyn,heading_dyn,Dindex,staregion_sta,staname_sta,datenum_sta,

type_sta,repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
                                                starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

            fin.close()
    #######To deal with .csv file################################################
        if f_name.endswith('.csv'):

            with open(infile, newline='') as csvfile:
                spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
                for row in spamreader:
                    xx = ''
                    for i in range(len(row)):
                        xx = xx+row[i]
                    if xx.find('\\',10) !=-1 and xx.count('AIVD')==1 :
                        x = xx.split('!')[0]
                        x = x.replace('\\','')
                        x = x.replace("*",",")
                        if x.find('c:') != -1 and x.count('c:')==1 :
                            Station_time   = 'N/A'
                            dt_time        = -9999
                            x0 = x.replace("c:","$")
                            x1 = x0.split('$')[1]
                            x2 = x1.split(',')[0]
                            dt_obj=time.gmtime(float(x2))
                            Station_time = time.strftime("%d-%b-%Y %H:%M:%S",dt_obj)
                            dt_time = int(x2)

                        s0 = x.replace("s:","$")
                        if s0.find('-') != -1:
                            s1 = s0.split('$')[1]
                            s2 = s1.split(',')[0]
                            Station_region = s2.split('-')[0]
                            Station_name   = s2.split('-')[1]
                        else:
                            Station_region = 'N/A'
                            Station_name   = 'N/A'

                        yy = xx.replace('\\','$')
                        y = yy.split('$')[2]
                        if len(y.split(','))>6 and y.split(',')[6]!='' :
                            if y.split(',')[0][0:5]=='!AIVD' :
                            #    print(y)
                                msg = y.rstrip('\n')

                                ais_data = decod_ais(msg,out_log)
                                if ais_data == None:
                                    pass
                                else:
                                    if len(ais_data)>4:
                                        output =
lineDecode(ais_data,Station_region,Station_name,Station_time,dt_time,staregion_dyn,staname_dyn,

datenum_dyn,type_dyn,repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,

lat_dyn,course_dyn,heading_dyn,Dindex,staregion_sta,staname_sta,datenum_sta,

type_sta,repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
```

```
starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

    #######Save the output to .nc file###########################################
        write_dyn(out_dyn,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,repeat_dyn,mmsi_dyn,status_dyn,
                  speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,course_dyn,heading_dyn,Dindex)

        write_sta(out_sta,staregion_sta,staname_sta,datenum_sta,type_sta,repeat_sta,mmsi_sta,shipname_sta,
                  shiptype_sta,stern_sta,port_sta,starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)


if __name__ == '__main__':

    main()

##############################################################################
```

PROCESS_AIS_PARALLEL.PY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May 22 17:28:44 2020

update AIS processing data with Lanli's May 15 Code
a) netcdf output
b) using ray for parallel processing

@author: xuj
"""

# from process_AIS_Onefile_Parallel_0427 import groupList,lineDecode

from Second_layer_NMEA_0521 import *
from Output_format_0521 import *

from Read_AIS import *
from Write_netcdf import *

import time
import ray
import os
import sys
import csv
import numpy as np

# numCpu=51
numCpu=20

ray.init(num_cpus = numCpu)

def lineDecode(row,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
                  repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,
                  course_dyn,heading_dyn,Dindex,staregion_sta,staname_sta,datenum_sta,type_sta,
                  repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
              starboard_sta,bow_sta,draught_sta,destination_sta,Sindex):

    xx = ''
    for i in range(len(row)):
        xx = xx+row[i]
    if xx.find('\\',10) !=-1 and xx.count('AIVD')==1 :
        x = xx.split('!')[0]
        x = x.replace('\\','')
        x  = x.replace("*",",")
        if x.find('c:') != -1 and x.count('c:')==1 :
            x0 = x.replace("c:","$")
            x1 = x0.split('$')[1]
            x2 = x1.split(',')[0]
            dt_time = float(x2)
        else:
            dt_time  = float('nan')

        s0 = x.replace("s:","$")
        if s0.find('-') != -1:
            s1 = s0.split('$')[1]
            s2 = s1.split(',')[0]
            Station_region = s2.split('-')[0]
            Station_name   = s2.split('-')[1]
        else:
            Station_region = 'N/A'
            Station_name   = 'N/A'

        yy = xx.replace('\\','$')
        y = yy.split('$')[2]
        if len(y.split(','))>6 and y.split(',')[6]!='' :
            if y.split(',')[0][0:5]=='!AIVD' :
                msg = y.rstrip('\n')

                ais_data = decod_ais(msg,out_log)
                if ais_data == None:
                    pass
                else:
                    if len(ais_data)>4:
                        if ais_data['type'] == 19:
                            ais_format = format_ais(ais_data,'dyn')
                            if ais_format == None:
                                pass
```

```
                        else:
                            ais_format['Station region'] = Station_region
                            ais_format['Station name']   = Station_name
                            ais_format['Datenum']        = dt_time

                            read_dyn(ais_format,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
                                     repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,
                                     course_dyn,heading_dyn,Dindex)

                        ais_format = format_ais(ais_data,'sta')
                        if ais_format == None:
                            pass
                        else:
                            ais_format['Station region'] = Station_region
                            ais_format['Station name']   = Station_name
                            ais_format['Datenum']        = dt_time

                            read_sta(ais_format,staregion_sta,staname_sta,datenum_sta,type_sta,
                                     repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
                                     starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

                    elif ais_data['type'] == 5 or ais_data['type'] == 24:
                        ais_format = format_ais(ais_data,'sta')
                        if ais_format == None:
                            pass
                        else:
                            ais_format['Station region'] = Station_region
                            ais_format['Station name']   = Station_name
                            ais_format['Datenum']        = dt_time

                            read_sta(ais_format,staregion_sta,staname_sta,datenum_sta,type_sta,
                                     repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
                                     starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

                    else:
                        ais_format = format_ais(ais_data,'dyn')
                        if ais_format == None:
                            pass
                        else:
                            ais_format['Station region'] = Station_region
                            ais_format['Station name']   = Station_name
                            ais_format['Datenum']        = dt_time

                            read_dyn(ais_format,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
                                     repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,
                                     course_dyn,heading_dyn,Dindex)
    pass

    return


def groupList(total,numGroups):
    countofgroups = np.floor(total/numGroups)
    remain = total%numGroups

    print(remain)

    segs =[]

    for i in np.arange(numGroups):
        segi = np.arange(countofgroups*i,countofgroups*(i+1))
        segs.append(segi)


    segRemain=np.arange(countofgroups*(i+1),countofgroups*(i+1)+remain)

    segs.append(segRemain)
    # print(len(segs))
    # print(segs)

    return(segs)

def process_incremental(sum, result):
    # time.sleep(1) # Replace this with some processing code.
    return sum + result

def groupDecode(rows,i,out_dyn,out_sta):

    staregion_dyn = []
    staname_dyn   = []
    datenum_dyn   = []
```

```
    type_dyn      = []
    repeat_dyn    = []
    mmsi_dyn      = []
    status_dyn    = []
    speed_dyn     = []
    accuracy_dyn  = []
    lon_dyn       = []
    lat_dyn       = []
    course_dyn    = []
    heading_dyn   = []
    Dindex        = 0

    staregion_sta = []
    staname_sta   = []
    datenum_sta   = []
    type_sta      = []
    repeat_sta    = []
    mmsi_sta      = []
    shipname_sta  = []
    shiptype_sta  = []
    stern_sta     = []
    port_sta      = []
    starboard_sta = []
    bow_sta       = []
    draught_sta   = []
    destination_sta = []
    Sindex        = 0

    out_dyni = out_dyn+'_'+str(i).rjust(3,'0')+'.nc'
    out_stai = out_sta+'_'+str(i).rjust(3,'0')+'.nc'

    decodeI=0   # count all the messages been decoded
    for row in rows:

        output = lineDecode(row,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
                  repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,
                  course_dyn,heading_dyn,Dindex,staregion_sta,staname_sta,datenum_sta,type_sta,
                  repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
              starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

        decodeI+=1

    if len(staregion_dyn)> 0:
        write_dyn(out_dyni,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,repeat_dyn,mmsi_dyn,status_dyn,
              speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,course_dyn,heading_dyn,Dindex)

    if len(staregion_sta)> 0:
        write_sta(out_stai,staregion_sta,staname_sta,datenum_sta,type_sta,repeat_sta,mmsi_sta,shipname_sta,
              shiptype_sta,stern_sta,port_sta,starboard_sta,bow_sta,draught_sta,destination_sta,Sindex)

    return decodeI


@ray.remote
def work(spami,ii,out_dyn,out_sta):

    numberofDecoded= groupDecode(spami,ii,out_dyn,out_sta)
    return len(spami),ii,numberofDecoded


def main():
    ##############File Directory##############################
    ####Need to set up!!!
    Input_Directory  = '/home/xuj/work/project/ais/data/'
    ####Need to set up!!!
    Output_Directory = '/home/xuj/work/project/ais/output0522/'
    ####Need to set up!!!
    Outlog_Directory = 'Outlog'

    for f_name in os.listdir(Input_Directory):
        print(f_name)
        out_log = Outlog_Directory+ "\ ".strip()+'Outlog_'+f_name[:-4]+'.txt'
        ####To deal with .csv file
        if f_name.endswith('.csv'):

            infile  = Input_Directory+ f_name
            out_dyn = Output_Directory+ 'Dynamic_'+f_name[:-4]
            out_sta = Output_Directory+ 'Static_'+f_name[:-4]


            with open(infile, newline='') as csvfile:
```

```
            spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
            spamreader = list(spamreader)

        numofProcess= numCpu-1

        totalLines= len(spamreader)
        print(totalLines)

        segs = groupList(totalLines,numofProcess)

        allSpam = np.array(spamreader)
        spamreader =[]

        start = time.time()
        totalSeg = len(segs)
        totalI = np.arange(0,totalSeg)

        spamSeg=[]
        for segi,ii in zip(segs,totalI):
            segIdx= segi.astype(int)
            spami = allSpam[segIdx]
            spamSeg.append(spami)

        allSpam=[]

        result_ids = [work.remote(spamSeg[ii],ii,out_dyn,out_sta) for ii in totalI]


        if True:
            sum = 0
            while len(result_ids):
                done_id, result_ids = ray.wait(result_ids)

                orginalNum,ii,decodedNum = ray.get(done_id[0])

                sum = process_incremental(sum,decodedNum)
                print("duration =", time.time() - start, "\nresult = ", sum)


##############################################################################

if __name__ == '__main__':
    t0 = time.perf_counter()

    main()
    t1 = time.perf_counter()

    print("Time elapsed: ", t1-t0)
```

FIRST_LAYER_NMEA.PY

```
#!/usr/bin/env python
"""
A bunch of python functions to decode ais_data/AIVDM messages:
Original author: Pierre Payen
From https://github.com/pirpyn/pyAISm
The functions were built based on doc : https://gpsd.gitlab.io/gpsd/AIVDM.html

Try with !AIVDO,1,1,,,B00000000868rA6<H7KNswPUoP06,0*6A
"""
##############################################################################
import logging
logger = logging.getLogger(__name__)
logger.setLevel(logging.WARNING)
def sign_int(s_bytes):
    # converts signed pack of bytes (as a string) to signed int
    # @param s_bytes (string) : '1001001010010010...'
    # @return (int) : signed integer
    temp = s_bytes
    if s_bytes[0]=='1':
        l = temp.rfind('1') #find last one
        temp2=temp[:l].replace('1', '2')
        temp2=temp2.replace('0', '1')
        temp2=temp2.replace('2', '0')
        temp=temp2+temp[l:]
        return -int(temp,2)
    else:
        return int(temp,2)

def compute_checksum(msg):
    # compute the checksum of an AIS sentense by XOR every char
    # then confront it to the checksum validator
    # @param (string) msg : one AIS sentense '!AIVDO,1,1,,,B00000000868rA6<H7KNswPUoP06,0*6A'
    # @return (string) : string representation of hexadecimal sum of XORing every char bitwise
    end = msg.rfind('*') # we're gonna read from after '?'' to before '*
    start = 0
    if msg[0] in ('$','!'): start=1 # reading after '!' if it exists
    chcksum = 0
    for c in msg[start:end]: # for every char in the ais sentenses (comma included)
        chcksum = chcksum ^ ord(c) # compare them with the x-or operator '^'
    sumHex = "%x" % chcksum # makes it hexadecimal
    return sumHex.zfill(2).upper()

##############################################################################

def get_msg_type(msg):
    # read the ais sentense and return the message type
    # @param (string) msg : one AIS sentense
'!AIVDM,2,1,3,B,55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53,0*3E'
    # @return (string) : the message type '!AIVDM'
    return msg.split(',')[0]

def get_payload(msg):
    # read the ais sentense and return the payload
    # @param (string) msg : one AIS sentense
'!AIVDM,2,1,3,B,55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53,0*3E'
    # @return (string) : the payload '55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53'
    return msg.split(',')[5]

def get_sentence_number(msg):
    # read the ais sentense and return the number of sentenses the payload is splitted in
    # @param (string) msg : one AIS sentense
'!AIVDM,2,1,3,B,55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53,0*3E'
    # @return (string) number of sentenses: '2'
    return msg.split(',')[1]

def get_sentence_count(msg):
    # read the ais sentense and return the number of the sentense
    # @param (string) msg : one AIS sentense
'!AIVDM,2,1,3,B,55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53,0*3E'
    # @return (string) the number of the current: '1'
    return msg.split(',')[2]

def get_checksum(msg):
    # read the ais sentense and return the number of the sentense
    # @param (string) msg : one AIS sentense
'!AIVDM,2,1,3,B,55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53,0*3E'
    # @return (string) checksum validator: '3E'
    return msg.split('*')[-1]
```

```python
############################################################################

def decod_payload(payload):
    # convert the payload from ASCII char to their 6-bits representation for every char
    # doc : http://catb.org/gpsd/AIVDM.html#_aivdm_aivdo_payload_armoring
    # @param (string) payload : '177KQ' up to 82 chars
    # @return (string) data :  '000001000111000111011011100001'
    data = ''
    for i in range(len(payload)):
        char = ord(payload[i])-48
        if char>40:
            char = char -8
        bit = '{0:b}'.format(char)
        bit = bit.zfill(6) # makes it a full 6 bits
        data = data + bit
    return data

def decod_6bits_ascii(bits):
    # decode 6bits into an ascii char, with respect to the 6bits ascii table
    # doc : http://catb.org/gpsd/AIVDM.html#_ais_payload_data_types
    # @param (string) bits : '101010'
    # @return (char) a ascii charater : '*'
    letter = int(bits,2)
    if letter < 32:
        letter+=64
    return chr(letter)

def decod_str(data):
    # decode a string of bits to an ascii one with respect to the 6bits ascii table
    # doc : http://catb.org/gpsd/AIVDM.html#_ais_payload_data_types
    # @param (string) data : a string of bits  '000001000001000001'
    # @return (string) a string of bits : 'AAA'
    name = ''
    for k in range(len(data)//6):
        letter = decod_6bits_ascii(data[6*k:6*(k+1)])
        if letter != '@':
            name += letter
    return name.rstrip()


def is_auxiliary_craft(mmsi):
    return mmsi//10000000 == 98

############################################################################
```

OUTPUT_FORMAT.PY

```
#!/usr/bin/env python
"""
Python functions to deal with the format of the decoded ais_data/AIVDM messages:
Original author: Pierre Payen
From https://github.com/pirpyn/pyAISm

Edited: Lanli Guo
Publish date: May 01, 2020
"""
##################################################################################
import logging

def format_coord(coord_dec,Dir=''):
    """
    get position in decimal base and return a string with position in arc-base
    :param coord_dec: (int) degree decimal coordinate 43.29492333333334
    :param Dir: (char) to specify the direction like 'N' for North. By default, nothing.
    :return: (string) a degree,minute,second coordinate + direction  43Â°17'41.7"N
    """
#####Format 1 to output in '°'+"'" +'"'
#    tmp = str(coord_dec).split('.')
#    deg = abs(float(tmp[0]))
#    mnt = float('0.'+tmp[1])*60
#    tmp = str(mnt).split('.')
#    sec = float('0.'+tmp[1])*60
#    return str(int(deg))+'°'+str(int(mnt))+"'"+str(sec)[:4]+'"'+Dir
#####Format 2 to output in '°'
#    return str('{:.1f}'.format(coord_dec))+Dir
#####Format 3 to output in float
    return float('{:.1f}'.format(coord_dec))

def format_mmsi(mmsi):
#####Format mmsi in 9 character
    return "{:0>9d}".format(mmsi)

def format_lat(lat):
#####Format 1 to output with 'N' and 'S'
#    return (format_coord(lat,'N') if lat > 0 else format_coord(lat,'S'))
#####Format 2 to output without 'N' and 'S'
    return (format_coord(lat) if lat > 0 else format_coord(lat))

def format_lon(lon):
#####Format 1 to output with 'E' and 'W'
#    return (format_coord(lon,'E') if lon > 0 else format_coord(lon,'W'))
#####Format 2 to output without 'E' and 'W'
    return (format_coord(lon) if lon > 0 else format_coord(lon))

def format_altitude(altitude):
    if speed == 4095:
        return 'N/A'
    elif speed == 4094:
        return ' > 4094 meters'
    elif speed == 0:
        return "0 meters"
    else:
        return "{0:.1f} meters".format(altitude)

def format_course(course):
#####Format 1 to output with '°'
#    return 'N/A' if course == 3600 else "{:3.1f}°".format(course).zfill(6) #with leading zeroes
#####Format 2 to output without '°' in string
#    return 'N/A' if course == 3600 else "{:3.1f}".format(course) #with leading zeroes
#####Format 3 to output without '°' in float
    #return float('nan') if course == 3600 else float("{:3.1f}".format(course)) #with leading zeroes
    return float('nan') if course == 3600 else course

def format_speed(speed):
    if speed == 1023:
        return float('nan')
    else:
        return speed*0.1

def format_heading(heading):
```

```
    #return 'N/A' if heading == 511 else "{:3.1f}°".format(heading).zfill(6) #with leading zeroes
    #return 'N/A' if heading == 511 else "{:3.1f}".format(heading) #with leading zeroes
    return float('nan') if heading == 511 else  heading

def format_month(month):
    return 'N/A' if month == 0 else month

def format_day(day):
    return 'N/A' if day == 0 else day

def format_hour(hour):
    return 'N/A' if hour == 24 else hour

def format_minute(minute):
    return 'N/A' if minute == 60 else minute

def format_second(second):
    if second == 60:
        return 'N/A'
    elif second == 61:
        return 'manual mode'
    elif second == 62:
        return 'EPFS in estimated mode'
    elif second == 63:
        return 'PS inoperative'
    else:
        return second

def format_cs(cs):
    return 'Class B SOTDMA' if cs == '0' else 'Class B CS'

def format_display(display):
    return 'N/A' if display == '0' else 'Display available'

def format_dsc(dsc):
    if dsc == '1':
      return 'VHF voice radio with DSC capability'

def format_band(band):
    if band == '1':
        return 'Can use any frequency of the marine channel'

def format_msg22(msg22):
    if msg22 == '1':
        return 'Accepts channel assignment via Type 22 Message'

def format_assigned(assigned):
    if assigned == '0':
        return 'Autonomous mode'

def format_dte(dte):
    return 'Data terminal ready' if dte == '0' else 'Data terminal N/A'

def format_epfd(epfd):
    epfd_types = [  'Undefined',
                    'GPS',
                    'GLONASS',
                    'GPS/GLONASS',
                    'Loran-C',
                    'Chayka',
                    'Integrated',
                    'Surveyed',
                    'Galileo',
                    'Undefined',
                    'Undefined',
                    'Undefined',
                    'Undefined',
                    'Undefined',
                    'Undefined',
                    'Undefined']
    return epfd_types[epfd]


def format_shiptype(shiptype):
    if shiptype>99 or shiptype<0:
        return float('nan')
```

```
    else:
        return shiptype

def format_turn(rot):
    if rot >= 1 and rot <= 126:
        return "{0:.1f}° Right".format((rot/4.733)**2)
    elif rot >= -126 and rot <= -1:
        return "{0:.1f}° Left".format((rot/4.733)**2)
    elif rot==127:
        return '> 5°/30sec Right (No TI available)'
    elif rot == -127:
        return '> 5°/30sec Left (No TI available)'
    return 'N/A'

def format_status(status):
    status_list = [
        "Under way using engine",
        "At anchor",
        "Not under command",
        "Restricted manoeuverability",
        "Constrained by her draught",
        "Moored",
        "Aground",
        "Engaged in Fishing",
        "Under way sailing",
        "Reserved for future amendment of Navigational Status for HSC", #TODO: Check if this is the
future
        "Reserved for future amendment of Navigational Status for WIG", #TODO: Check if this is the
future
        "Reserved for future use",
        "Reserved for future use",
        "Reserved for future use",
        "AIS-SART is active",
        "Not defined (default)",
    ]
    return status_list[status]

def format_aid_type(aid_type):
    aid_type_list = [
        "Default, Type of Aid to Navigation not specified",
        "Reference point",
        "RACON (radar transponder marking a navigation hazard)",
        "Fixed structure off shore, such as oil platforms, wind farms, rigs. (Note: This code should
identify an obstruction that is fitted with an Aid-to-Navigation AIS station.)",
        "Spare, Reserved for future use",
        "Light, without sectors",
        "Light, with sectors",
        "Leading Light Front",
        "Leading Light Rear",
        "Beacon, Cardinal N",
        "Beacon, Cardinal E",
        "Beacon, Cardinal S",
        "Beacon, Cardinal W",
        "Beacon, Port hand",
        "Beacon, Starboard hand",
        "Beacon, Preferred Channel port hand",
        "Beacon, Preferred Channel starboard hand",
        "Beacon, Isolated danger",
        "Beacon, Safe water",
        "Beacon, Special mark",
        "Cardinal Mark N",
        "Cardinal Mark E",
        "Cardinal Mark S",
        "Cardinal Mark W",
        "Port hand Mark",
        "Starboard hand Mark",
        "Preferred Channel Port hand",
        "Preferred Channel Starboard hand",
        "Isolated danger",
        "Safe Water",
        "Special Mark",
        "Light Vessel / LANBY / Rigs",
    ]
    return aid_type_list[aid_type]

format_list = {#list of all the key that can/want to be formatted
```

```python
#               'lat'      : format_lat,
#               'lon'      : format_lon,
#               'status'   : format_status,
#               'mmsi'     : format_mmsi,
               'course'   : format_course,
               'speed'    : format_speed,
               'heading'  : format_heading,
               'second'   : format_second,
               'cs'       : format_cs,
               'display'  : format_display,
               'dsc'      : format_dsc,
               'band'     : format_band,
               'msg22'    : format_msg22,
               'assigned' : format_assigned,
               'dte'      : format_dte,
               'epfd'     : format_epfd,
               'shiptype' : format_shiptype,
               'month'    : format_month,
               'day'      : format_day,
               'hour'     : format_hour,
               'minute'   : format_minute,
               'turn'     : format_turn,
               'aid_type' : format_aid_type
               }

def format_ais(ais_base,style):
    """
    format the ais_data database to a more user-friendly display
    :param ais_base: (dict) the ais_data base to format
    :return: (dict) ais_format : a dictionary with the same kay as ais_data but other value,
            None if ais_base is None
    """
    if (ais_base == None):
        return None
    ais_format_0 = ais_base.copy()

    if style == 'dyn':
        ais_format = {'Station region':'N/A','Station name':'N/A','datenum': -9999,
                      'type': -9999,'repeat':-9999,'mmsi':-9999,'status':-9999,'turn':'N/A',
                      'speed':float('nan'),'accuracy':-9999,'lon':float('nan'),'lat':float('nan'),
                      'course':float('nan'),'heading':float('nan'),'second':-9999,'maneuver':-9999,
                      'raim':-9999,'radio':-9999}
    else:
        ais_format = {'Station region':'N/A','Station name':'N/A','datenum': -9999,
                      'type': -9999,'repeat':-9999,'mmsi':-9999,'shipname':'N/A','shiptype':-9999,
                      'to_stern':float('nan'),'to_port':float('nan'),'to_starboard':float('nan'),
                      'to_bow':float('nan'),'draught':float('nan'),'destination':'N/A'}


    for key in list(ais_base.keys()):                              #for every key we have
        if key in format_list:                                     #if we can format it
            ais_format_0[key] = format_list[key](ais_format_0[key])#format it

    if len(ais_format_0)<4:                                        #ingore the short messages
        return None
    else:
        for key in list(ais_format.keys()):                        #for the order we want
            if key in list(ais_base.keys()):                       #if we have the keys
                ais_format[key] = ais_format_0[key]                #organize them

        return ais_format

################################################################################
```

READ_AIS.PY

```
#!/usr/bin/env python
"""
This code is to collect the decoded data from each message, and prepare to
save the output of AIS decoded data in nc files:

Author: Lanli Guo
Publish date: May 01, 2020
"""
##########For Dynamic Output################################################
def read_dyn(ais_format,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,repeat_dyn,
             mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,course_dyn,
             heading_dyn,Dindex):

    staregion_dyn.append(ais_format['Station region'])
    staname_dyn.append(ais_format['Station name'])
    datenum_dyn.append(ais_format['Datenum'])
    type_dyn.append(ais_format['type'])
    repeat_dyn.append(ais_format['repeat'])
    mmsi_dyn.append(ais_format['mmsi'])
    status_dyn.append(ais_format['status'])
    speed_dyn.append(ais_format['speed'])
    accuracy_dyn.append(ais_format['accuracy'])
    lon_dyn.append(ais_format['lon'])
    lat_dyn.append(ais_format['lat'])
    course_dyn.append(ais_format['course'])
    heading_dyn.append(ais_format['heading'])
    Dindex += 1

    return staregion_dyn,staname_dyn,datenum_dyn,type_dyn,repeat_dyn,mmsi_dyn,status_dyn,\
           speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,course_dyn,heading_dyn,Dindex

#############For Static Output#############################################
def read_sta(ais_format,staregion_sta,staname_sta,datenum_sta,type_sta,repeat_sta,
             mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,starboard_sta,
             bow_sta,draught_sta,destination_sta,Sindex):

    staregion_sta.append(ais_format['Station region'])
    staname_sta.append(ais_format['Station name'])
    datenum_sta.append(ais_format['Datenum'])
    type_sta.append(ais_format['type'])
    repeat_sta.append(ais_format['repeat'])
    mmsi_sta.append(ais_format['mmsi'])
    shipname_sta.append(ais_format['shipname'])
    shiptype_sta.append(ais_format['shiptype'])
    stern_sta.append(ais_format['to_stern'])
    port_sta.append(ais_format['to_port'])
    starboard_sta.append(ais_format['to_starboard'])
    bow_sta.append(ais_format['to_bow'])
    draught_sta.append(ais_format['draught'])
    destination_sta.append(ais_format['destination'])
    Sindex += 1

    return staregion_sta,staname_sta,datenum_sta,type_sta,repeat_sta,mmsi_sta,shipname_sta,\
           shiptype_sta,stern_sta,port_sta,starboard_sta,bow_sta,draught_sta,destination_sta,Sindex

###########################################################################
```

WRITE_NETCDF.PY

```
#!/usr/bin/env python
"""
This code is to generate the output of AIS decoded data in nc files:

Author: Lanli Guo
Publish date: May 01, 2020
"""
###############################################################################
#from scipy.io import netcdf
import netCDF4
import numpy as np

#############For Dynamic Output############################################
def write_dyn(fnameout,staregion_dyn,staname_dyn,datenum_dyn,type_dyn,
              repeat_dyn,mmsi_dyn,status_dyn,speed_dyn,accuracy_dyn,lon_dyn,lat_dyn,
              course_dyn,heading_dyn,Dindex):

    Dim = Dindex

    ncfout = netCDF4.Dataset(fnameout,'w')
    ncfout.description = "AIS_CCG Dynamic Information"
    ncfout.createDimension('Dindex',Dim)

    station_region =  ncfout.createVariable('station_region','S3',('Dindex',))
    station_region[:]=np.array(staregion_dyn)
    station_region.long_name = "Station Region"
    station_region.missing_value = 'N/A'

    station_name =  ncfout.createVariable('station_name','S20',('Dindex',))
    station_name[:]=np.array(staname_dyn)
    station_name.long_name = "Station Name"
    station_name.missing_value = 'N/A'

    date_num     =  ncfout.createVariable('date_num','i',('Dindex',))
    date_num[:]  =  np.array(datenum_dyn)
    date_num.long_name = "Seconds since 1970-Jan-01 00:00:00"
    date_num.missing_value = -9999

    message_type   =  ncfout.createVariable('message_type','i',('Dindex',))
    message_type[:]=  np.array(type_dyn)
    message_type.long_name = "Message Type"
    message_type.missing_value = -9999

    repeat       =  ncfout.createVariable('repeat','i',('Dindex',))
    repeat[:]    =  np.array(repeat_dyn)
    repeat.long_name = "Repeat"
    repeat.missing_value = -9999

    mmsi       =  ncfout.createVariable('mmsi','i',('Dindex',))
    mmsi[:]    =  np.array(mmsi_dyn)
    mmsi.long_name = "MMSI"
    mmsi.missing_value = -9999
    mmsi.description = "Less than 9 digits, add '0' to the left"

    status     =  ncfout.createVariable('status','i',('Dindex',))
    status[:]  =  np.array(status_dyn)
    status.long_name = "Status"
    status.missing_value = -9999

    speed     =  ncfout.createVariable('speed','f',('Dindex',))
    speed[:]  =  np.array(speed_dyn)
    speed.long_name = "Speed"
    speed.units     = "knot"
    speed.missing_value = np.nan
    speed.description = "Value 102.2 indicates 102.2 knots or higher"

    accuracy  =  ncfout.createVariable('accuracy','i',('Dindex',))
    accuracy[:]  =  np.array(accuracy_dyn)
    accuracy.long_name = "Accuracy"
    accuracy.missing_value = -9999

    longitude  =  ncfout.createVariable('longitude','d',('Dindex',))
    longitude[:]  =  np.array(lon_dyn)
```

```
    longitude.long_name = "Longitude"
    longitude.units     = "degree"
    longitude.missing_value = np.nan

    latitude  =  ncfout.createVariable('latitude','d',('Dindex',))
    latitude[:]  =  np.array(lat_dyn)
    latitude.long_name = "Latitude"
    latitude.units     = "degree"
    latitude.missing_value = np.nan

    course  =  ncfout.createVariable('course','f',('Dindex',))
    course[:]  =  np.array(course_dyn)
    course.long_name = "Course"
    course.units     = "degree"
    course.missing_value = np.nan

    heading =  ncfout.createVariable('heading','f',('Dindex',))
    heading[:]  =  np.array(heading_dyn)
    heading.long_name = "Heading"
    heading.units     = "degree"
    heading.missing_value = np.nan

    ncfout.close()

    return fnameout

################For Static Output###########################################

def write_sta(fnameout,staregion_sta,staname_sta,datenum_sta,type_sta,
              repeat_sta,mmsi_sta,shipname_sta,shiptype_sta,stern_sta,port_sta,
              starboard_sta,bow_sta,draught_sta,destination_sta,Sindex):

    Dim = Sindex

    ncfout = netCDF4.Dataset(fnameout,'w')
    ncfout.description = "AIS_CCG Static Information"

    ncfout.createDimension('Sindex',Dim)

    station_region =  ncfout.createVariable('station_region','S3',('Sindex',))
    station_region[:]=np.array(staregion_sta)
    station_region.long_name = "Station Region"
    station_region.missing_value = "N/A"

    station_name =  ncfout.createVariable('station_name','S20',('Sindex',))
    station_name[:]=np.array(staname_sta)
    station_name.long_name = "Station Name"
    station_name.missing_value = "N/A"

    date_num     =  ncfout.createVariable('date_num','i',('Sindex',))
    date_num[:]  =  np.array(datenum_sta)
    date_num.long_name = "Seconds since 1970-Jan-01 00:00:00"
    date_num.missing_value = -9999

    message_type   =  ncfout.createVariable('message_type','i',('Sindex',))
    message_type[:]=  np.array(type_sta)
    message_type.long_name = "Message Type"
    message_type.missing_value = -9999

    repeat       =  ncfout.createVariable('repeat','i',('Sindex',))
    repeat[:]    =  np.array(repeat_sta)
    repeat.long_name = "Repeat"
    repeat.missing_value = -9999

    mmsi       =  ncfout.createVariable('mmsi','i',('Sindex',))
    mmsi[:]    =  np.array(mmsi_sta)
    mmsi.long_name = "MMSI"
    mmsi.missing_value = -9999
    mmsi.description = "less than 9 digits, add '0' to the left"

    shipname     =  ncfout.createVariable('shipname','S20',('Sindex',))
    shipname[:]  =  np.array(shipname_sta)
    shipname.long_name = "Ship Name"
    shipname.missing_value = "N/A"

    shiptype     =  ncfout.createVariable('shiptype','i',('Sindex',))
```

```python
    shiptype[:]  =  np.array(shiptype_sta)
    shiptype.long_name = "Ship Type"
    shiptype.missing_value = -9999

    stern  =  ncfout.createVariable('stern','f',('Sindex',))
    stern[:]  =  np.array(stern_sta)
    stern.long_name = "Dimension to Stern meters"
    stern.units     = "m"
    stern.missing_value = np.nan

    port  =  ncfout.createVariable('port','f',('Sindex',))
    port[:]  =  np.array(port_sta)
    port.long_name = "Dimension to Port meters"
    port.units     = "m"
    port.missing_value = np.nan

    starboard  =  ncfout.createVariable('starboard','f',('Sindex',))
    starboard[:]  =  np.array(starboard_sta)
    starboard.long_name = "Dimension to Starboard meters"
    starboard.units     = "m"
    starboard.missing_value = np.nan

    bow  =  ncfout.createVariable('bow','f',('Sindex',))
    bow[:]  =  np.array(bow_sta)
    bow.long_name = "Dimension to Bow meters"
    bow.units     = "m"
    bow.missing_value = np.nan

    draught =  ncfout.createVariable('draught','f',('Sindex',))
    draught[:]  =  np.array(draught_sta)
    draught.long_name = "Draught"
    draught.units     = "m"
    draught.missing_value = np.nan

    destination     =  ncfout.createVariable('destination','S20',('Sindex',))
    destination[:]  =  np.array(destination_sta)
    destination.long_name = "Destination"
    destination.missing_value = "N/A"

    ncfout.close()

    return fnameout

###############################################################################
```