Available online at http://www.e-navigation.kr/
e-Navigation Journal

Original article

Building of an Optimal Algorithm for Nearest Navigational Danger Detection in ENC using DP☆

Jisoo KIM[a], Hongrai CHO[a], Unggyu KIM[a], Byunggong HWANG[a]

[a] R&D Center, mapsea Corp., Seoul, Korea

Abstract

Efficiently detecting the nearest navigational dangers in Electronic Chart Display and Information Systems (ECDIS) remains pivotal for maritime safety. However, the software implementation of ADMAR(Automatic Distance measurement and Ranging) functionality faced challenges, necessitating extensive computations across ENC cells and impacting real-time performance. To address this, we present a novel method employing dynamic programming. Our proposed algorithm strategically organizes nodes into a tree structure, optimizing the search process towards nodes likely to contain navigational hazards. Implementation of this method resulted in a notable sevenfold reduction in computation time compared to the conventional Brute Force approach. Our study established a direct correlation between the ADMAR functionality and node count, achieving error margins deemed acceptable for practical navigation scenarios. Despite this theoretical progress, minor errors in results prompt further refinement. Consequently, future iterations will explore varying values for N, considering hierarchy and cell sizes to enhance algorithmic precision. This research signifies a potential advancement in optimizing navigational danger detection within ECDIS, offering a promising avenue for improved efficiency. By introducing a dynamic programming-based approach, we have streamlined the detection process while acknowledging the scope for algorithmic refinement in subsequent studies. Our findings underline the feasibility of employing dynamic programming to enhance navigational danger detection, emphasizing its potential in ensuring maritime safety. This work lays a foundation for future research endeavors, aiming to fine-tune algorithms and advance navigational safety measures in ECDIS.

Keywords: Nearest Navigational Danger, ADMAR, ENC, ECDIS, Brute Force, SENC, Dynamic Programming

https://doi.org/10.52820/j.enavi.2023.21.043

1. Introduction

In the realm of Electronic Chart Display and Information Systems (ECDIS), the quest for precise navigational danger detection has led to an exploration of methodologies, including the application of a Brute Force algorithmic approach (Weintrit A. 2002). While traditional Brute Force algorithms entail exhaustive computations by checking all possible solutions, in the context of ECDIS, a semblance of this method involves a systematic examination of potential hazards in navigational paths.

Notably, within the realm of navigational hazard detection, the Brute Force method has retained significance as a benchmark for accuracy. This approach, outlined by Garcia and team (2006), involves exhaustive and systematic examination of all possible scenarios to determine the nearest navigational danger. Despite its computational intensity, the Brute Force method has been widely accepted as a ground truth for evaluating the accuracy of alternative algorithms due to its comprehensive nature.

While the Brute Force method remains a reliable benchmark, its computational demands often hinder real-time application within ECDIS. Consequently, recent research efforts, such as those by Wang and Liu (2018), have aimed at developing more efficient algorithms capable of approaching the accuracy of Brute Force while significantly reducing computational load, thus enabling practical real-time implementation.

In the context of ECDIS, navigational hazards can encompass various elements that pose risks or threats to safe navigation. Some common navigational hazards within this context might include shoals and underwater obstructions, reefs and coral, wrecks and derelicts, channels and fairways, icebergs or ice floes, offshore installations and other navigationally significant features such as lighthouses, buoys, beacons, or landmarks critical for navigation but could pose risks if inaccurately represented on charts.

These hazards need to be accurately detected, identified, and represented in ECDIS to ensure the safety of maritime navigation. Detecting and managing these hazards play a vital role in preventing accidents or collisions and ensuring safe passage for vessels.

Each type of hazard might demand a specific approach for detection, affecting the time required and the accuracy achieved. Some hazards may be more straightforward to detect accurately and quickly, while others might require more complex or time-consuming methods due to their nature or location. Therefore, considering the diversity of hazards, the detection methods, time, and accuracy levels could differ significantly across various categories within ECDIS.

This research paper delves into the innovative adaptation of a Brute Force-like algorithm for detecting the nearest navigational dangers within ECDIS. The concept involves a systematic analysis of navigational routes or areas by evaluating all potential hazards within the vicinity, akin to a methodical grid-based examination or iterative analysis along the vessel's intended path.

However, the utilization of a Brute Force-like approach in ECDIS for navigational danger detection presents inherent challenges. Its computational intensity poses concerns regarding efficiency and scalability, particularly in managing vast maritime areas and dynamically evolving environments. Despite these challenges, the concept represents an intriguing avenue for refining navigational safety mechanisms within ECDIS.

Therefore, to address this, we propose a method for detecting the nearest navigational danger that suits the data structure of S-57-based ENC. The proposed algorithm involves pre-stratifying the data structure of ENC using dynamic programming, thereby reducing the number of ENC data nodes that need to be searched (Weintrit A. 2002).

2. The Analysis of Navigational Danger Detection in ENC

The ECDIS highlights in new ways four features that are important for safe navigation:

- The own-ship safety contour,
- Depth zone shades,
- The own-ship safety depth,
- Isolated dangers.

The own-ship safety contour is the contour related to the own ship selected by the mariner out of the contours provided for in the SENC (System Electronic Navigational Chart), to be used by ECDIS to distinguish on the display between the safe and the unsafe water, and for generating anti-grounding alarms. The usability of such ECDIS for displaying the nearest navigational danger depends on the performance of the hardware system where the ECDIS is installed and the time complexity of the algorithms embedded in the ECDIS software kernel. Especially considering that the performance of device-specific hardware is fixed, with the recent trend of ECDIS functionalities transitioning to mobile devices, it's essential to be able to run software for detecting the nearest navigational danger on hardware with lower capabilities than before.

2.1. Data Structure of S-57 ENC

Within the ENC, contour lines consist of a list of vector nodes at the same depth, as depicted in Figure 1.

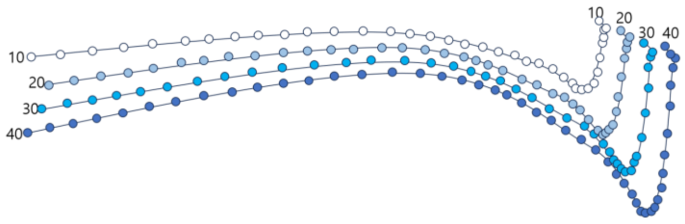

Figure 1. Contour lines vector node data structure in ENC

2.2. ADMAR

ADMAR (Automatic Distance Measurement and Ranging) is a specialized feature of ECDIS that continuously displays the distance to the nearest navigational danger automatically, as illustrated in Figure 2.

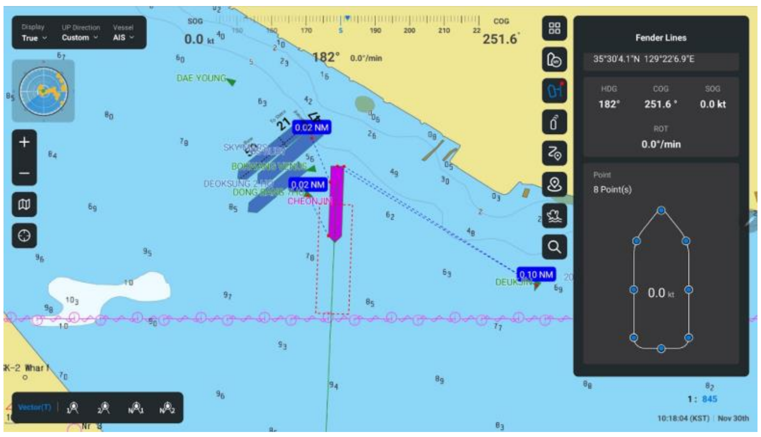

Figure 2, Example of Distance Measurement Function on Mapsea Navigation

To implement a functionality similar to Figure 2 within the software, it involves finding the list of nodes from the contour lines deeper than the vessel's Safety contour, and then identifying the specific node closest to the vessel among this node list.

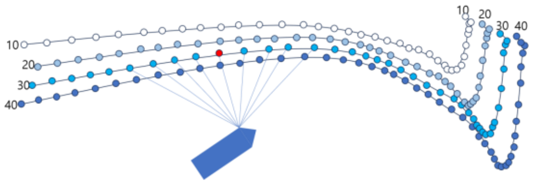

Figure 3. Implementation of ADMAR Functionality in ENC Data Structure

For instance, for a vessel with a safety contour of 24, in an environment similar to Figure 3, the process involves finding the list of contour line nodes at depth 30 that are greater than 24. From this node list, the node closest to the vessel is identified.

In traditional ECDIS systems, locating the node closest to the vessel within a specific node list involved using a brute force method, necessitating a search through all node lists at a particular depth within the ENC cell. This conventional approach incurred significant time complexity as it unnecessarily searched through all nodes, highlighting the need to reduce this complexity.

3. Optimal Algorithm for Nearest Navigational Danger Detection

We propose an algorithm utilizing dynamic programming, which recursively and hierarchically groups nodes within each node list in the ENC. This algorithm averages coordinates of proximate nodes in a certain quantity.

3.1. Hierarchical Aggregation Procedure

The proposed algorithm incorporates a specific positive integer N as an aggregation hyperparameter, which determines the pre-processing procedure involving averaging nodes hierarchically in N increments. Additionally, the number of aggregation levels (L), representing the recursive aggregation iterations, also serves as a hyperparameter.

For example, aggregating ENC data similar to Figure 1 with N=3, L=2 would result in data aggregated in a tree-like structure for each level, as illustrated in Figure 4.

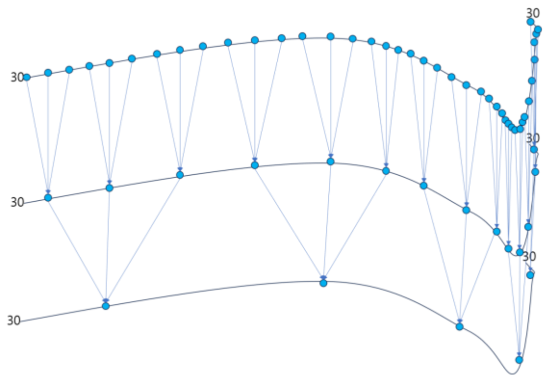

Figure 4 Example of Hierarchical ENC Node Aggregation Method Using Dynamic Programming

The above procedure can be summarized into pseudocode as follows:

Pseudocode 1. Pre-processing Procedure

```
for (int l = 0; l < L; l++)//Loop 1
    for (ArrayList<Point2D> arrayList : DPLayer[l]) {//Loop 2
        ArrayList<Point2D> innerList = new ArrayList<Point2D>();
        int dplength = arrayList.size() / N;
        for(int ii = 0; ii < dplength; ii++)//Loop 3
        {
            double averx = 0;
            double avery = 0;
            for(int jj = 0; jj < N; jj++)//Loop 4
            {
                averx += arrayList.get(ii * N + jj).x;
                avery += arrayList.get(ii * N + jj).y;
            }
            averx /= N;
            avery /= N;
            Point2D ptPoint2d = new Point2D();
            ptPoint2d.x = averx;
            ptPoint2d.y = avery;
            innerList.add(ptPoint2d);
        }
        double averx2 = 0;
        double avery2 = 0;
        for(int jj2 = dplength * N; jj2 < arrayList.size(); jj2++)//Loop 5
        {
            averx2 += arrayList.get(jj2).x;
            avery2 += arrayList.get(jj2).y;
        }
        averx2 /= arrayList.size() - dplength * N;
        avery2 /= arrayList.size() - dplength * N;
        Point2D ptPoint2d2 = new Point2D();
        ptPoint2d2.x = averx2;
        ptPoint2d2.y = avery2;
        if(averx2 != 0.0 && avery2 != 0.0
        && !Double.isNaN(averx2)
        && !Double.isNaN(avery2))
        {
            innerList.add(ptPoint2d2);
        }
        DPLayer[l + 1].add(innerList);
    }
```

The first loop, Loop 1, in Pseudocode 1 is an iteration based on the recursive hierarchy constant L. This means it aggregates from lower to higher levels for each hierarchy. The subsequent loop, Loop 2, iterates through the list of lists of nodes in the ENC data structure. This loop identifies a specific node list. Subsequently, it computes the new length (dplength) for the upper level by dividing the number of nodes in the current hierarchy node list by N. Loops 3, 4, and 5 iterate through this new length, calculating the average of N adjacent nodes in the lower hierarchy and storing the result in the upper hierarchy.

3.2. Search Procedure

The real-time search procedure used in ECDIS software reverses the pre-processing procedure of 3-1. It involves searching through all the top layers and recursively searching only nodes belonging to the lower hierarchy of nodes already searched, which are the shortest distance nodes from the vessel.

Utilizing the dynamically computed hierarchical structure as depicted in Figure 4, to determine the nearest navigational danger to the vessel shown in Figure 3, it follows a search range and procedure similar to Figure 5.

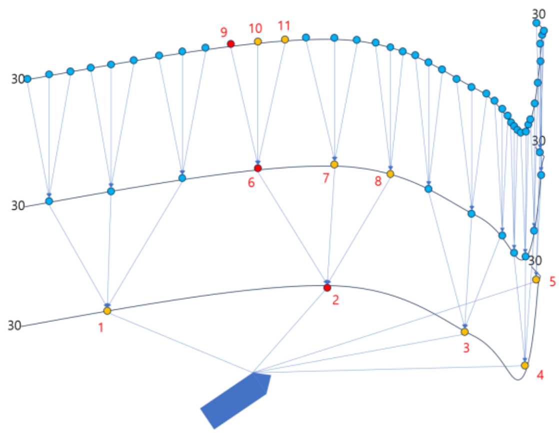

Figure 5 Example of Search Procedure from Dynamic Programming-based ENC Node Aggregation Data

In the case of Figure 3, it was necessary to explore all 43 nodes at depth 30. However, in Figure 5, it can be observed that only a total of 11 nodes were explored to identify the nearest navigational danger.

The above procedure can be summarized into pseudocode as follows:

Pseudocode 2. Real-time Search Procedure

```
int outerIdx = -1;
int innerIdx = -1;
double minDist = Double.MAX_VALUE;
for (int idp = 0; idp < DPLayer[L-1].size(); idp++) {//Loop 1
    for(int jdp = 0; jdp < DPLayer[L-1].get(idp).size(); jdp++)//Loop 2
    {
        double dist = eucDist(MyVessel.x, MyVessel.y, DPLayer[L-1].get(idp).get(jdp).x, DPLayer[L-1].get(idp).get(jdp).y);
        if(dist <= minDist)
        {
            minDist = dist;
            outerIdx = idp;
            innerIdx = jdp;
        }
    }
}
for(int l = L-2; l > -1; l--)//Loop 3
{
    minDist = Double.MAX_VALUE;
    int innerIdx2 = -1;
    for(int ii = 0; ii < N; ii++)//Loop 4
    {
        double dist = eucDist(MyVessel.x, MyVessel.y, DPLayer[l].get(outerIdx).get(innerIdx * N + ii).x, DPLayer[l].get(outerIdx).get(innerIdx * N + ii).y);
        if(dist <= minDist)
        {
            minDist = dist;
            innerIdx2 = innerIdx * DPDivider + ii;
        }
    }
    innerIdx = innerIdx2;
if(l == 0)//If-clause 1
{
print(DPLayer[l].get(outerIdx).get(innerIdx).x,
DPLayer[l].get(outerIdx).get(innerIdx).y))
}
}
```

Loop 1 in Pseudocode 2 extracts the list of nodes from each list of node lists in the ENC data structure that contains the node belonging to the shortest distance. Loop 2 extracts nodes from the list of top-level nodes that represent the shortest distance. Loop 3 traverses the remaining levels, excluding the top-level, and Loop 4 iterates through nodes that are likely to contain the shortest distance nodes in the tree structure. After going through these procedures, finally, If-clause 1 outputs the nearest navigational danger.

4. Application and Validation of Algorithm

4.1. Validation Environment

We conducted experiment using S-57 Electronic Navigational Charts (ENCs) of US waters provided by NOAA (https://www.charts.noaa.gov/ENCs/ENCs.shtml). Our experiment was based on coastal lines, the boundaries between sea and land, to maintain node consistency concerning the safety contour. We utilized 1,162 ENC files out of a total of 2,540 ENCs provided by NOAA that contained coastal lines. The vessel's positions were randomly selected—10,000 points within each cell size of the ENC files, ensuring they fell within the sea points. Hence, we computed the nearest navigational danger for a total of 11,620,000 vessel positions.

We experimented with different hyperparameters for the dynamic programming approach, using aggregation units N as 2, 3, 4, and 5. The total number of aggregation levels L was fixed at 3 for consistency. Additionally, we implemented and experimented with the traditional Brute Force algorithm.

The hardware environment for the validation processes was as follows.

In Table 3, the average processing times for each algorithm are presented. Overall, it is evident that the proposed algorithm significantly outperforms the Brute Force method, especially noting that the case where N equals 3 demonstrates the fastest performance. Based on the average processing time, it seems that the Brute Force algorithm could run smoothly in real-time on both desktop and mobile environments, indicating a less pressing need for the proposed algorithm. However, in the actual execution environment of ECDIS, where multiple ENC cells overlap, measuring the nearest navigational danger from various points of the vessel, the ideal conditions depicted in Table 3 may not persist consistently.

Table 1. Hardware Environment

CPU	Core	Thread	Max. CPU Clock	Types of RAM	RAM Capacity	RAM Clock
intel i5-12400f	6	12	4.4GHz	DDR4	16.0GB	1600MHz(dual channel)

The validation software environment is as follows.

Table 2. Software Environment

Program Language	IDE	ENC SDK
JAVA	Eclipse IDE 2022-09	JOSM(https://github.com/JOSM)[10]

Table 3. Average processing time

Algorithm	BF	DP(N=2)	DP(N=3)	DP(N=4)	DP(N=5)
Average Processing Time(ms)	0.0242	0.0072	0.0035	0.0036	0.0047

Table 4. Top 10 Node Count Processing Times

Algorithm			BF	DP(N=2)	DP(N=3)	DP(N=4)	DP(N=5)
Node Count	72,320	Processing Time	0.4523	0.0860	0.0342	0.0208	0.0148
	64,389		0.4183	0.0852	0.0332	0.0224	0.0206
	63,987		0.3886	0.0797	0.0302	0.0185	0.0150
	61,338		0.4054	0.0969	0.0419	0.0279	0.0269
	60,402		0.3666	0.0786	0.0299	0.0181	0.0167
	60,198		0.3921	0.0826	0.0334	0.0267	0.0202
	59,556		0.2576	0.0663	0.0225	0.0134	0.0102
	58,608		0.3225	0.0669	0.0252	0.0150	0.0127
	53,688		0.1638	0.0680	0.0277	0.0176	0.0149
	52,225		0.2862	0.0584	0.0198	0.0123	0.0103

4.2. Validation Results

I n Table 4, we compiled ten examples where the node count reaches tens of thousands. Comparing these examples with Figure 6, it becomes evident that as the node count increases, the computational load of the Brute Force algorithm escalates linearly, emphasizing the growing necessity for the proposed algorithm

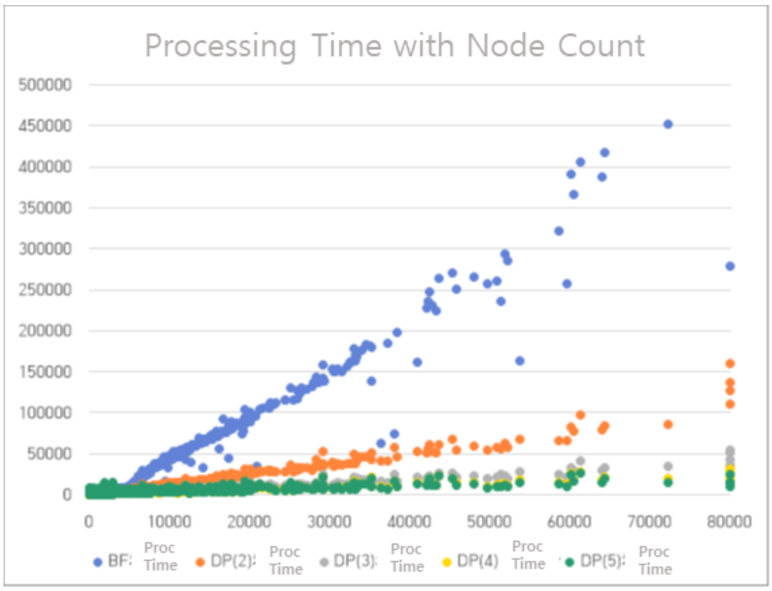

Figure 6. Changes in Processing Time with Node Count

In Figure 6, the graph illustrates the change in processing time for each algorithm concerning the node count. According to this graph, while the proposed algorithm is faster than the Brute Force method, both exhibit a linear complexity of O(C) concerning the number of nodes (C).

Table 5. Average Preprocessing Time

N	2	3	4	5
Average Preprocessing Time (ms)	2.319	2.304	2.356	2.343

In Table 5, we can observe the processing time required for the preprocessing steps utilizing dynamic programming in the proposed algorithm. It indicates a minimal time requirement in the millisecond range for each cell. Additionally, as this computation occurs during the installation of ENC cells rather than in real-time, in terms of computational load, it does not translate to a significant noticeable time for the use

Table 6. Average Error

N	2	3	4	5
Average Error (km)	0.0719	0.1632	0.2544	0.3325

In Table 6, the average error in kilometers is presented. This error, measured in kilometers, is calculated using Brute Force as the ground truth and determining the Euclidean distance, in kilometers, between the actual nearest navigational danger and the result obtained from the proposed algorithm. According to Table 6, even in the best-case scenario with N equal to 2, there is a significant error, averaging 0.0718 kilometers or 71.8 meters. This could pose issues for practical navigation purposes. However, these values are presented without considering the size of the ENC cells. Table 7 outlines errors based on the diagonal length of the chart, providing a perspective considering chart size, and Table 8 summarizes the error rate based on the diagonal size of the ENC cells.

Table 7. Top and Bottom 10 Average Errors based on Chart Diagonal Length

Top 10			DP(N=2)	DP(N=3)	DP(N=4)	DP(N=5)
Diagonal Length (km)	3,344	Average Error (km)	7.686	17.484	27.190	28.126
	3,036		0.241	0.263	0.409	0.308
	2,674		2.228	3.473	5.557	5.879
	2,445		1.979	6.032	5.801	7.204
	1,582		1.004	1.788	2.838	2.848
	1,356		1.224	4.353	7.530	12.294
	1,335		0.849	1.430	1.461	1.293
	1,313		0.430	0.806	1.139	1.072
	872		1.155	1.991	2.847	3.837
	718		0.058	0.565	0.813	0.971
Bottom 10			DP(N=2)	DP(N=3)	DP(N=4)	DP(N=5)

Diagonal Length (km)	0.1940	Average Error (km)	0.0080	0.0020	0.0060	0.0107
	0.1428		0	0	0	0
	0.1182		0.0066	0.0085	0.0059	0.0032
	0.0835		0.0048	0.0030	0.0045	0.0026
	0.0814		0.0053	0.0020	0.0051	0.0064
	0.0607		0.0030	0.0006	0.0017	0.0010
	0.0566		0.0038	0.0042	0.0015	0.0027
	0.0548		0.0004	0.0035	0	0
	0.0221		0.0003	0.0007	0	0
	0.0182		0.0008	0	0	0

Table 8. Average Error Rate

N	2	3	4	5
Average Error Rate(%)	0.0051	0.0090	0.0127	0.0154

According to Tables 7 and 8, the error relative to the ENC cell size at N equals 2 is approximately 0.51%. This indicates that for ENC cell sizes within 1 kilometer, the error is at a level of a few meters, which might not pose significant navigational issues.

Upon consolidating all the results, it became apparent that, in terms of speed, the proposed algorithm outperforms the Brute Force algorithm. Furthermore, to minimize error rates, N equals 2 appears favorable, while for increasing processing speed, N equals 3 seems to be the optimal choice.

5. Conclusions

With advancements in technology, the ADMAR functionality for detecting the nearest navigational danger in ECDIS has become a crucial aid for navigators. However, executing the ADMAR functionality in software posed a challenge, requiring unnecessary exploration of tens of thousands of points on ENC cells. This issue demanded a significant portion of real-time computation performance from ECDIS to resolve.

To address this, we proposed a novel method of detecting the nearest navigational danger using dynamic programming. Our algorithm involves bundling nodes into unit quantities to create a tree structure, allowing for a search procedure that focuses on nodes likely to contain the nearest navigational danger.

By employing this proposed algorithm, we achieved a sevenfold reduction in time compared to the previous Brute Force method for identifying the nearest navigational danger. Furthermore, we demonstrated that the functionality of ADMAR correlates linearly with the number of nodes. Additionally, we analyzed the error rate of our proposed method, showing errors within acceptable margins for practical navigation scenarios.

However, considering some level of errors in the proposed algorithm's results, there is a need for improvement in future iterations. We plan to conduct further research by applying different values for N based on hierarchy and cell sizes to refine the algorithm.

6. Acknowledgement

This research was supported by the Korean Institute of Marine Science and Technology Promotion (KIMST) funded by the Ministry of Oceans and Fisheries, Korea(20220093 & RS-2023-00254860).

References

Vincent Garcia, Sylvain Boltz, E Debrreuve, Michel Barlaud (2006), Contour tracking for rotoscoping based on trajectories of feature points, ECCV Workshop on Statistical Methods in Multi-Image and Video Processing (SMVP), Graz, Autriche, France

Rui Liu, Hong Wang, Xiaomei Yu (2018), Shared-nearest-neighbor-based clustering by fast search and find of density peaks, Information Sciences Volume 450, June 2018, China

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2022), Introduction to Algorithms 4th Edition, The MIT Press Cambridge, Massachusetts London, England

A.N.Cockcroft and J.N.F.Lameijer (2012) A Guide to the Collision Avoidance Rules seventh edition, ScienceDirect

Dimitri P. Bertsekas (2017), Dynamic Programming and Optimal Control Vol. 1, 4th Edition, Massachusetts Institute of Technology, USA

Dimitri P. Bertsekas (2022), Abstract Dynamic Programming, 3rd Edition, EBOOK at Google Play, Massachusetts Institutes of Technology, USA

Weintrit A. (2002), Automatic Measurement of Distance to the Nearest Navigational Danger in ECDIS, XIII-th International Scientific and Technical Conference 'The Part Of Navigation In Support Of Human Activity On The Sea', Naval University of Gdynia

Weintrit A. (2003), Voyage recording in ECDIS. Shipborne simplified version of Voyage Data Recorders (VDRs) for existing cargo ships based on potential of ECDIS, 11th IAIN World Congress 'Smart Navigation – Systems and Services', organised by German Institute of Navigation, International Association of Institutes of Navigation, Berlin

Kolowrocki K., Weintrit A. (2003), A New ADMAR Unit Conception in ECDIS, Saferelnet Workshop: Safety and Reliability in Waterborne Transport. Ship Design and Research Centre, Gdansk

Bellman. R. (1966), Dynamic Programming. Science, 153(3731), 34-37

Olsen, Alexander Arnfinn. (2022) Core Principles of Maritime Navigation (p. 62). CRC Press. Kindle Edition.

Weintrit A., Kopacz P. (2004), Safety Contours on Electronic Navigational Charts, 5th International Symposium 'Information on Ships', ISIS 2004, Hamburg

Adam Weintrit & Piotr Kopacz (2004), The use of Distance Measurement to the nearest navigational Danger in Riute Planning, Rout Monitoring and Voyage Recording in ECDIS, XIVth International Scientific and Technical Conference, The Part of Navigation in Support of Human Activity on The SeaAt: Institute of Navigation and Hydrography, Naval Academy, Gdynia

IHO TRANSFER STANDARD for DIGITAL HYDROGRAPHIC DATA Edition 3.1 - November 2000 Special Publication No. 57

USA office of Coast Survey website (https://www.charts.noaa.gov/ENCs/ENCs.shtml)

Java OpenStreetMap Editor (https://github.com/JOSM)

Received **04 December 2023**
1st Revised **22 December 2023**
2nd Revised **26 December 2023**
3rd Revised **27 December 2023**
Accepted **27 December 2023**