IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 29, 2021, accepted December 26, 2021, date of publication December 30, 2021,
date of current version January 12, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3139767

Electronic Navigational Charts for Visualization, Simulation, and Autonomous Ship Control

SIMON BLINDHEIM, (Member, IEEE), AND TOR ARNE JOHANSEN, (Senior Member, IEEE)
Centre for Autonomous Marine Operations and Systems, Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), 7491 Trondheim, Norway

Corresponding author: Simon Blindheim (simon.blindheim@ntnu.no)

This work was supported in part by the Online Risk Management and Risk Control for Autonomous Ships (ORCAS) Project funded by the Research Council of Norway under Grant 280655, and in part by the Kongsberg Maritime, Det Norske Veritas (DNV), and the Centre for Autonomous Marine Operations and Systems (AMOS) at NTNU funded by the Research Council of Norway under Grant 223254.

ABSTRACT Autonomous control and high-level decision-making for autonomous ships in complex environments are largely dependent on real-time navigational data from sensory systems and nautical charts. Visualization and manipulation of data contained within electronic navigational charts (ENC) or hydro/geographic information systems (HIS/GIS) is generally handled on the application or operations level. However, accessible and open-source application programming interfaces (API) for displaying and managing spatial bathymetry or other related sea-faring data for research and development are scarce. This work presents an open-source ENC visualization and manipulation API implemented in Python, with heavy emphasis on accessibility and simplicity. The current version of the package provides tools for displaying marine polygon data such as ships, ocean depths, reefs, and shallows, using the transverse Mercator projection. Additionally, polygon- and point-based transformation and calculation methods for application development based on spatial geometry, path planning and numerical optimal control are implemented. Usage of the spatial methods are demonstrated by examples involving high-level path or trajectory planning, optimization, and assisted decision-making for autonomous and remote-controlled ships.

INDEX TERMS Autonomous ships, bathymetry, decision-making, electronic navigational charts (ENC), geographic information systems (GIS), maps, maritime systems, nautical charts, optimal control, path planning, polygon data, risk management, sea charts, simulation, trajectory planning, 2D visualization.

I. INTRODUCTION

Electronic navigational charts (ENC) have today become the digital standard to replace printed navigational charts. Such formats allow for a range of possibilities with regards to data handling. However, the task of efficiently showing and manipulating ENC data has arguably become more challenging with increased data sizes and degrees of freedom, and solutions are largely developed on a case-to-case basis. More specifically, open application programming interfaces (API) for public use are scarce. Given the limited resources for polygon-based maritime environments available today, it is apparent that there is a need for an open-source ENC API for future research and software development efforts.

The associate editor coordinating the review of this manuscript and approving it for publication was Mu-Yen Chen.

A. LITERATURE REVIEW

Active development of increasingly more advanced ENC has been steadily moving forward, since the emergence of digitally stored bathymetry data and specifications of international exchange standards for hydrographic data were formulated [1]. There are several objectives associated with the use of ENC; applications related to pure visualization for navigation purposes, geometric calculations and spatial data operations, automatic control and decision support for manned or unmanned operations such as anti-grounding and obstacle avoidance, safety or risk analysis, and route or path planning. However, there seems to be a lack of open API which sufficiently provide necessary and/or convenient tools for research and development of such systems. Though recent works have been carried out to specify and solidify forms of systematization, standardization and classification of ENC [2]–[5], API solutions for both visualization and

direct spatial ENC data manipulation are still limited. The following sections present an analysis of some of the relevant resources available in the literature today.

1) VISUALIZATION

Norms and standards for visualization of two-dimensional (2D) bathymetry data has been consistent with practices used for printed navigational charts, and numerous applications have emerged for various areas, such as pure visualization of ENC data [6]–[8], radar display imaging [9], [10], three-dimensional (3D) visualization of bathymetry data [11]–[13], and even endeavors on virtual or augmented reality [14], [15]. However, visualization of spatial data is merely a tool for evaluation and affirmation within research and development on applications for autonomous decision-making or decision support. Thus, the development of such stand-alone user applications or API is not sufficient for research and development within this area, and variations of such solutions are consequently not considered in this work.

2) DATA EXTRACTION

In order to be able to utilize environment data for autonomous navigation, it is necessary to make the intrinsic spatial data contained within the ENC accessible for external applications through a programmable software interface, in addition to simultaneous environment visualization and information display. ENC may as such be utilized as direct data inputs for active decision support systems [16], and may be formulated as a vector-based architecture [17] given a spatial point- or polygon-based database from e.g. surveys [18], [19].

3) PATH PLANNING

Decision support systems using operational parameters related to mission objectives and spatial hydrographic ENC data are mainly concerned with path or trajectory planning in a maritime environment. Path planning in accordance with mission objectives is a known problem, and as such there exists a rich collection of research on efficient path planning for autonomous or unmanned surface vehicles (ASV / USV), such as long-distance mission planning [20], dynamic or adaptive re-planning [21], [22], control based on objective optimization and vector fields [23]–[27], and optimal paths using the established A* algorithm [28]–[30].

It is noted that path planning in the context of maritime operations in itself is only a tool commonly used to achieve higher levels of autonomy. As such, path planning for this purpose is combined with (sub-)systems for situational awareness (i.e. structured spatial data and prediction or simulation capabilities) based on ENC as well as sensor data, to be used in schemes for decision-making and risk analysis. However, the results of this research indicate some of the functionality required of an adequate ENC API. It is important that objective optimization is facilitated by making relevant ENC data available to the algorithm, such as depth values and distance calculations. Moreover, the prevalent prominence of methods based on vector fields suggests that it is desirable to include a sufficient range of vector-based methods in the ENC API, as well as to preserve the ability for scalable resolutions of the data sets loaded by the system. If methods based on raster images or spatial grids is used for decision-making, such grids may be directly constructed by interpolation between coordinate points through sampling along the spatial axes.

4) AUTONOMOUS NAVIGATION, RISK & SAFETY

The overarching main objective of autonomous navigation systems (ANS) or decision support systems is to increase maritime safety [31], [32] as well as efficiency, with respect to risks and/or expected cost analyses. Additionally, effective visualization of important risk factors for humans involved both during the design, planning and operational phases of maritime navigation is an integral part in achieving this goal, further highlighting the need for practical and flexible ENC API for the purpose of additional information overlay visualization. Moreover, risk analysis and risk management of (semi-) autonomous vessels are of increased importance with the development of unmanned operations [33], [34]. With higher levels of autonomy, the requirements for data quality and performance demanded by the involved decision-making systems increase significantly, and as such must be provided by the underlying ENC API.

5) AUTONOMOUS OBSTACLE & COLLISION AVOIDANCE

Path planning and autonomous obstacle or collision avoidance for complex maritime environments remains a challenging problem [35]. In order to further advance ANS algorithms for ASV, decision-making should include considerations related to environmental disturbances, identification and dynamic or adaptive planning with respect to static and dynamic obstacles, COLREG compliance, and utilization of vessel safety domains [36]. These systems must be highly flexible and robust, and be able to handle both long-term and short-term (reactive) planning for obstacle avoidance based on reliable information, i.e. sensors and detailed bathymetric ENC data constructed as polygons [37], [38].

Recent example applications include traffic monitoring with respect to collision detection and risk assessment [39], obstacle tracking and reactive collision avoidance based on sensor fusion [40], and the development of adaptive safety domains for identification of grounding risks [41] and collision avoidance during vessel interactions (COLREG) [42]. Thus, it is clear that the demand for open and accessible ENC software and resources for real-time simultaneous visualization and efficient computation of hydrographic data is evermore increasing.

6) CURRENT AVAILABLE API SOLUTIONS

In general, web searches for ENC or hydrographical API yield few relevant results, and there currently exists only a handful of ENC API today. Moreover, most of these are proprietary or closed-source, making it exceedingly difficult to identify individual resources suitable for feature-by-feature

comparison between these, general desired API capabilities, and the solution presented in this paper. Due to this fundamental lack of accessibility and transparency, such alternatives are inherently considered inadequate for open research and development. The available open-source software for ocean mapping is currently varying in quality and functionality, and no single API is identified as an all-in-one solution [43].

Pydro [44] is identified as the most promising candidate solution supporting the International Hydrographic Oranization S-57 standard [45]. However, Pydro is not strictly a pure API for Python programming, but a suite of software tools built for hydrography and cartography. The software is focused on enhancing and automating the hydrographic workflow from data acquisition to hydrographic survey review and nautical chart compilation [46] which leaves the work of visualization and high-level data abstraction to the researcher. Additionally, the application is currently only supported in Windows, and may not be repackaged or redistributed due to license restrictions. These are significant constraints with respect to open research and development, and Pydro is as such considered unsuitable for this purpose.

The conclusion of the literature review is thus that no truly open-source API packages or ENC solutions for open research and development currently exists.

B. PROBLEM & CONTRIBUTION

Unambiguously structured and visualized spatial data are needed for clear interpretation and efficient computation in autonomous ENC-based maritime navigation. This work addresses the research question: How can the research and development process for autonomous control and decision-making of autonomous and remote-controlled ships be improved and made more efficient for future works? It is proposed that an open-source Python-based API may serve as a framework and/or a valuable support tool for further development of hydrographic information systems (HIS), simulation or control algorithms for ANS based on ENC.

Thus, the purpose of this paper is to provide an open and user-friendly API package intended for fast and easy prototyping during software development such that researchers more quickly may get to work on development on (autonomous) navigation systems, rather than spending time on writing basic GIS functionality such as polygon manipulation or simply displaying a maritime environment. The package is focused on ease of use and fast prototyping capabilities through high-level spatial computation of polygon data and flexible visualization methods, and may play a part in facilitating more productive and targeted research on the topic of autonomous ships, by allowing the researcher to concentrate more on other aspects than application programming of a baseline simulation environment for testing purposes. Intelligent design of the ENC package may also lead to significant improvements to computation speeds for dynamic path planning and simulation.

TABLE 1. Class diagram of the main SeaCharts ENC module in Python.

ENC
- environment: Environment - display: Display + supported_crs: **str** + supported_layers: **str** + land: Land + shore: Shore + seabed: **dict**
+ ENC(size: **tuple**, origin: **tuple**, center: **tuple**, buffer: **int**, tolerance: **int**, layers: **list**, depths: **list**, files: **list**, new_data: **bool**, raw_data: **bool**, border: **bool**, verbose: **bool**, multiprocessing: **bool**): ENC + show_display(duration: **float**): **None** + refresh_display( ): **None** + close_display( ): **None** + save_image(name: **str**, scale: **float**, extension: **str**): **None** + colorbar(arg: **bool**): **None** + dark_mode(arg: **bool**): **None** + fullscreen_mode(arg: **bool**): **None** + add_vessels(args: **list**): **None** + clear_vessels( ): **None** + add_ownship(easting: **int**, northing: **int**, heading: **float**, hull_scale: **float**, lon_scale: **float**, lat_scale: **float**): **None** + remove_ownship( ): **None** + add_hazards(depth: **int**, buffer: **int**): **None** + draw_arrow(start: **tuple**, end: **tuple**, color: **str**, width: **float**, head_size: **float**, thickness: **float**, edge_style: **str**): **None** + draw_circle(center: **tuple**, radius: **float**, color: **str**, fill: **bool**, thickness: **float**, edge_style: **str**): **None** + draw_line(points: **list**, color: **str**, width: **float**, thickness: **float**, edge_style: **str**): **None** + draw_polygon(geometry: **Any**, color: **str**, interiors: **list**, fill: **bool**, thickness: **float**, edge_style: **str**): **None** + draw_rectangle(center: **tuple**, size: **tuple**, color: **str**, rotation: **float**, fill: **float**, thickness: **float**, edge_style: **str**): **None**

An overview of the contributions included in the proposed API is listed in Section II, and shown as implemented class instance methods in Table 1.

C. SCOPE

Only 2D visualization is considered, in accordance with the goal of a simple and comprehensible interface. As marine environments inherently are concerned with depth-related data and representations, the core objective is thus to visualize objects and ocean depth through the use of distinctly colored 2D shapes. Moreover, all given coordinates are projected onto a locally flat plane via the Universal Transverse Mercator (UTM) coordinate system, subsequently allowing all polygon or other shape-based operations performed by the application to assume a flat 2D plane. Significant efforts in this work are targeted toward the development of clear and uncomplicated methods for straightforward geometric

polygon manipulation and operations for spatial depth data sets, such as polygon simplification, intersections, unions, dilution, erosion, convex hulls, and distance and bounding box calculations. Finally, several example algorithms are defined and discussed in order to showcase usage of the implemented methods of the ENC API package, within the fields of autonomous path planning, collision avoidance, and online risk management.

D. INSTALLATION & USAGE

Python is chosen as the preferred programming language for the development of an open-source API for elegant visualization and straightforward manipulation of spatial data, given its design philosophy with regard to its readability, object oriented structure and readily available libraries for management of GIS resources.

The package is given the name *SeaCharts*, and may be installed through the Python Package Index (PyPI) [47] by executing the command **pip install seacharts**, making it readily available through the standard **import** statements in Python files within an environment. Interested readers may find maintained links to the SeaCharts homepage, source code repository, documentation and usage instructions at pypi.org/project/seacharts.

E. DESIGN & STRUCTURE

Table 1 presents a class diagram of the main class **ENC** to be instantiated by the user, from which all other processes are initiated. An instance of this main class serves as a single-point interface, which is initialized by arranging and encapsulating polygonal features extracted from external Esri File Geodatabase (FGDB) files as local shapefiles [48]. These files are subsequently read and fed into a private **environment** variable containing all available data computation methods, accessed by the user through the top-level hypsometry attributes (**land**, **shore**, **seabed**) and direct utilization of geometric operations provided by the *Shapely* library [49].

The **display** variable may if desired be utilized through the provided class methods to display user-selected spatial features produced by the environment variable, serially or in parallel with data computations performed by a possible third-party navigation or optimization program. Users of the SeaCharts package are advised to refer to its *Readme* file in the maintained repository for detailed usage instructions. The following sections nevertheless provide an overview of the currently available API methods and functionality, while leaving most of the programming-related details to the code documentation.

II. METHODS

This section presents the main contributions of this work. The initialization and creation of an ENC interface instance is firstly described by its data extraction, polygon handling and feature extraction processes, followed by demonstrations of most of the user methods available in the API package as seen in Table 1, at the time of writing.

The constructor of the **ENC** class is denoted at the top of the third compartment. Its input arguments are used during initialization, and will be thoroughly discussed in Sections II-A, II-B and II-C. Built-in Python types are denoted in bold.

The methods related to the display are summarized as follows: The display utility methods, namely the **show_display**, **refresh_display**, **close_display** and **save_image**, are used to show and save images from the interactive display during runtime. Additional visual configuration settings are available through the **colorbar**, **dark_mode** and **fullscreen_mode** methods.

Next, methods for adding (drawing and maintaining references to) vessels and a controllable interactive ownship with toggleable hazardous areas within a given horizon are included through the methods **add_vessels**, **clear_vessels**, **add_ownship**, **remove_ownship** and **add_hazards**.

Lastly, opaque or transparent geometric shape overlays may be drawn onto the displayed environment through the draw-methods at the bottom of Table 1. Note that the **draw_polygon** method may in general be used to draw *any* polygon-based shape, the rest are provided for convenience.

A. DATA EXTRACTION

The API is initialized through the creation of the single ENC instance, which reads and stores geometric shapes for both future offline re-reading and dynamic shape handling during runtime. This section illustrates how the API handles the data during this initialization process, based on any combination of the class constructor arguments of Table 1. For all remaining figures in this paper, the **border** argument is set to **True** in order to produce an encompassing black border around the image edges, and the **multiprocessing** variable is set to **False** such that environment plotting is dependent on the running demonstration applications. The **verbose** argument may if desired be toggled on for the purpose of information printing in the terminal during runtime.

The SeaCharts package supports loading and reading spatial data structured in the FGDB 10.0 format. For demonstration purposes, the open-source ENC data used in this work is downloaded from the Norwegian Mapping Authority, which contains 2D ENC polygons of the coast of Norway projected in *EUREF89 UTM zone 33N*. However, any region of the world may be read and extracted as an ENC, provided a properly structured FGDB for the area in question is available.

At the time of writing, the **depth data** files for Norway may be downloaded as a whole or divided into specific county areas. These files are to be placed in a specific folder relative to the working directory of the Python application such that the Data module may reach it, as detailed in the SeaCharts package Readme. Processing of the downloaded FGDB files is performed during startup when the application is first run, or if the user has explicitly requested a manual data extraction

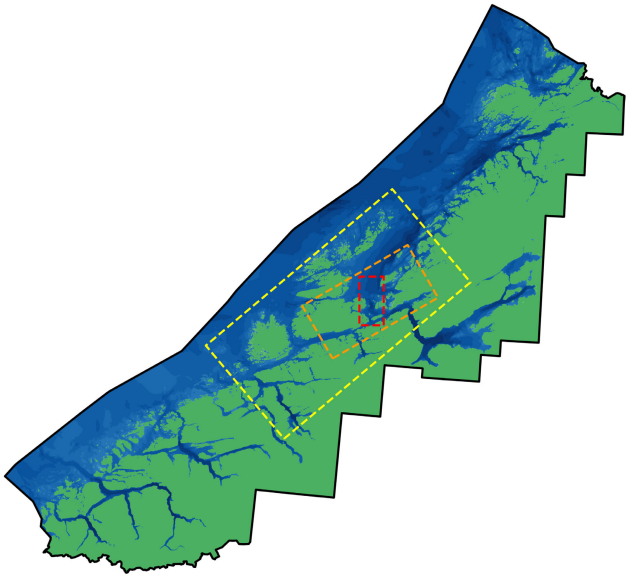

FIGURE 1. Data extraction window hierarchy of the SeaCharts package.

during a subsequent run by setting the **new_data** argument value equal to **True**. During this process, the polygonal or point data files specified by the **files** argument (a list of file names) are loaded into memory and spatially filtered given a user-specified bounding box calculated from the **size** and **origin** (or **center**) arguments, as well as the depth bin groups defined by the **depth** argument (see Section II-B). The specified data is subsequently written to shapefiles and stored in memory for direct access during runtime.

Figure 1 presents a visualization of the intended data extraction window hierarchy of the SeaCharts package. The figure shows an example view of the Norwegian counties *Møre og Romsdal* and *Trøndelag* merged together. The black border is the abstraction of any (digital) ENC downloaded or constructed for any given region of the world, and may contain one or several specific regions of spatial data such as e.g. countries or municipalities. Next, the yellow rectangle represents an arbitrary region of *extracted depth data* from the downloaded ENC in black, stored as shapefiles on the system's hard drive, as well as being available through the corresponding top-level hypsometry layers of the **ENC** class instance. This region may be considerably smaller than the initial raw data sets, and contains only the optimized features (Section II-B) selected by the user during the initial FGDB file extraction phase. The orange rectangle shows the *shape handling* subregion or area for which specific polygon- or point-based computations may be performed during runtime, e.g. to extract and merge all depth layers deeper than ten meters in order to construct a (binary) seafaring domain feasible for any particular ship. Lastly, the red rectangle represents a *dynamic horizon* subregion of the orange area, which may be used by some external path planning, collision avoidance, or simulation algorithm. This area is intended to be dynamic during external algorithm cycles and e.g. follow the ship(s) in question during a voyage, such that appropriate domains may be extracted and/or scaled for feasible optimization or planning given some real-time requirements and mission objectives. Note that both the shape handling and dynamic horizon regions may be freely constructed as any arbitrary polygon based on the application for which the API is used, and may if desired simply be identical.

In addition to selecting subregions of spatial data (even across data set boundaries such as e.g. county borders), the package explicitly specifies and selects each desired spatial features or layers to be extracted from the data sets during initialization through the **layers** argument containing a list of valid layer names. If the user of the API is e.g. only concerned specifically with the seabed depth polygons included in the depth data set, all other features like docks or other marine structures are disregarded and filtered out from the constructed shapefiles. The resulting shapefile data set read by the application after the initial startup thus only contains requested data, which allows for faster performance for dynamic filtering and spatial computing during runtime. Additional polygonal data optimization or simplifications are further described in the following sections.

B. POLYGON OPTIMIZATION

Downloaded depth data sets are generally large, and regularly contain suboptimal numbers of additional polygons due to region splits produced by some mapping algorithm or e.g. county boundaries. Thus, the SeaCharts package performs various standard GIS simplification procedures on the raw data in order to speed up runtime calculations and visualization, unless the **raw_data** argument is set to **True**. This process is included and described in this work for completeness, such that users of the package may become familiar with common GIS techniques in the context of the provided API functionality for research and development.

1) POLYGON MERGING

Figure 2 shows a SeaCharts visualization of the raw polygon data extracted from the downloaded FGDB files surrounding the Norwegian city of *Ålesund* (with **raw_data = True**), in which polygon edges are white for demonstration purposes. It is clear that the depth data polygon regions are divided into orthogonally adjacent subrectangles, creating situations in which there exists more than one polygon for any single disjoint body of land, shore, or seabed depth. In addition to the increased computational complexity introduced by this data structure, such redundant polygon definitions may also produce unexpected, undesirable or potentially incorrect results when performing numerical spatial operations, e.g. irregularities where only half an island is included for path planning purposes due to excluded polygon areas which are not properly intersected with a dynamic extraction window. Calculation performance during runtime may thus be increased significantly by merging or calculating the unions of all orthogonally adjacent polygons

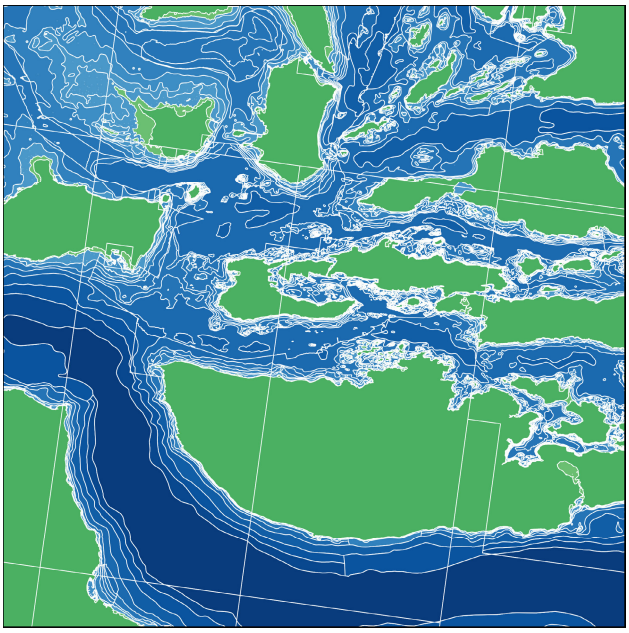

FIGURE 2. SeaCharts visualization of raw downloaded polygon data with depth bins of 5, 10, 20, 50, 100, 200, 350 and 500 m.

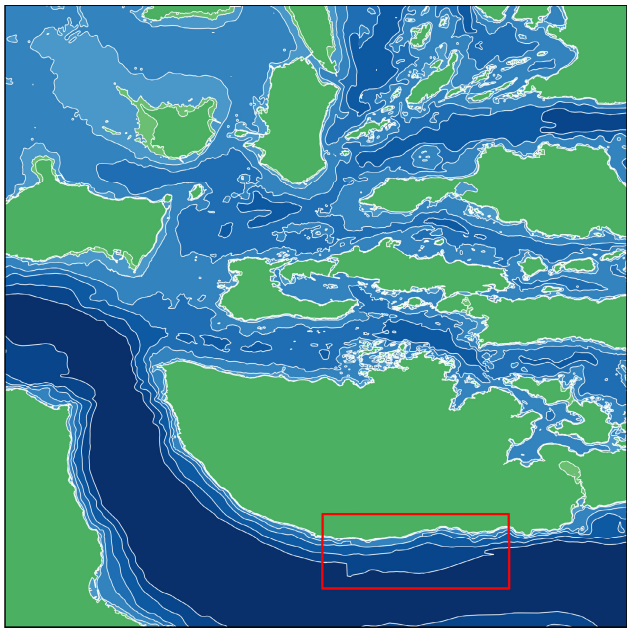

FIGURE 3. Polygon merging with depth bins of 10, 50, 100, 200, 300 m.

with depth data within the same depth bins present in the extracted data sets.

Figure 3 presents the result after performing the *Shapely* operation **unary_union** on the region from Figure 2. Notice how the polygons of large areas across the white rectangular divisions have been merged together, such that no continuous regions of the same depths are split. Each seabed polygon is constructed such that it fully envelops any and all polygons with depths *deeper* than its depth value, that it might contain. Thus, the total area of all polygons of depths between e.g. 5 m and 10 m will consequently be larger than the total area of all seabed layers deeper than 10 m. Note that this behavior is specific to the SeaCharts package, and is chosen as such with the purpose of intuitive alternative views during visualization - e.g. to toggle off deeper depth layers such that the remaining encompassing and more shallow layers are still complete. All land polygons are however entirely contained within the complete set of shore polygons, for the analogous inverse reason. This layer structure facilitates an intuitive and effortless lookup interface with respect to the fundamental question of identifying which areas are safe, hazardous or simply impassable for any given ship to navigate within. If e.g. the maximum draft of a ship is 5 m, it is thus straightforward to simply extract the full seabed layer of e.g. 7 m including an additional safety depth margin.

In addition to the merged regional shapes, there are fewer different values of depth polygons shown in the resulting environment in Figure 3 compared to the original downloaded data in Figure 2 due to the user-specified depth bins, i.e. ranges for which each depth measurement is grouped and separated by, through the **depths** argument list of integers. This feature may serve as another flexible layer of data management for further computation optimization, e.g. by disregarding depth range resolutions outside the scope of a path planning problem. In Figure 3, depths of 5 m, 20 m and deeper than 300 m were disregarded, such that the deeper *Sulafjorden* regions shown in dark blue in the bottom are consolidated into a region of a larger range of depth values. Notice however how the different resolutions present in the original raw data may produce polygon artifacts at the boundaries between the resolution partitions, e.g. along the southern coast of the *Sula* island/peninsula, marked by a red rectangle: The gradually deeper depth contours in Figure 2 reveals a noticeable resolution discrepancy edge where two depth bins are merged into one. These unavoidable artifacts emerge from the use of several different spatial resolutions in the downloaded data set, and should be treated with care.

2) POLYGON SIMPLIFICATION

The next step of the polygon optimization process is to simplify the topology of the polygon edges and vertices. This operation is provided by the Shapely method **simplify**, which removes vertices and edges from the polygon that are within the distance defined by the optional **tolerance** initialization argument. The resulting polygon may have a significantly reduced number of vertices, and consequently may reduce the time complexity of the spatial algorithms performed on the polygon data significantly. Moreover, the geometric shapes may be further simplified by buffering (dilating or eroding) all polygons by providing the optional **buffer** argument.

Figure 4 presents an oversimplification example of the same Ålesund fjord area from the previous section, showcasing why the simplification of the polygons must be performed with care. If the tolerance distance is too large, the polygons may become significantly distorted and thus

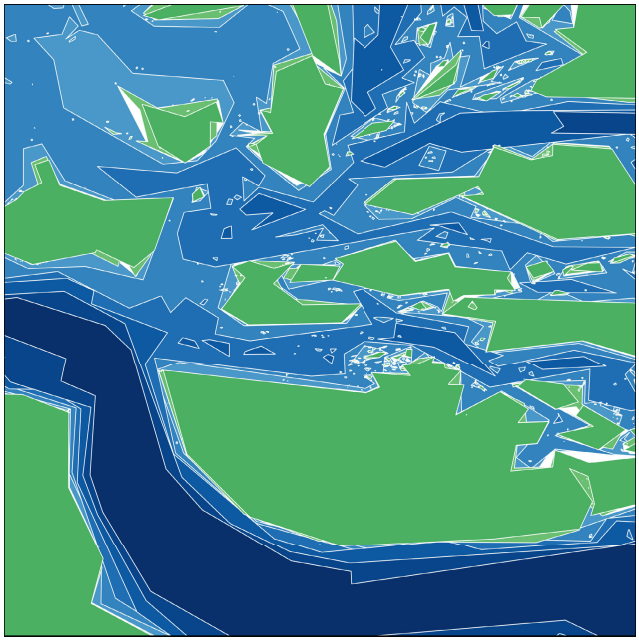

FIGURE 4. Polygon simplification example with a tolerance of 300 m.

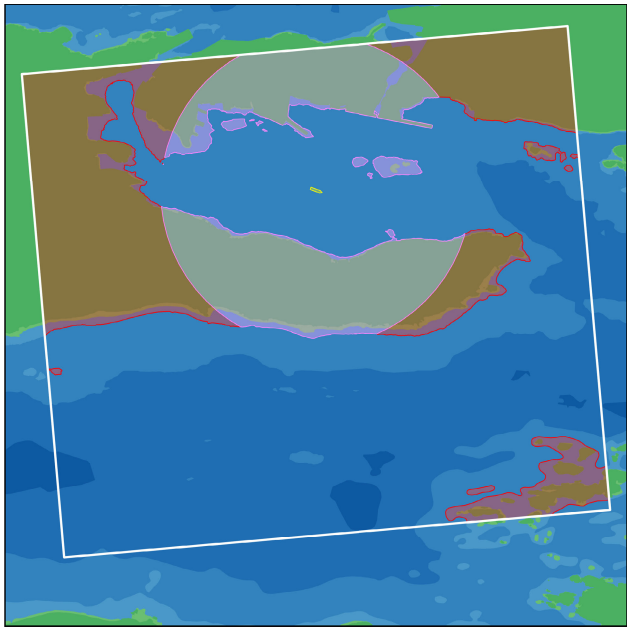

FIGURE 5. Bounded shape handling and dynamic horizon example.

no longer sufficiently represent the real-life obstacles in the environment.

Additionally, oversimplification may lead to some artifact regions not being covered by a depth polygon (as seen in white). These areas inherently have no ocean depth data associated with them, introducing irregularities and holes in the spatial data which are not easily fixed. Thus, the tolerance distance given to the simplification method is during the rest of this work set to be smaller than the width of the ship used for demonstration purposes, such that the topology of any polygon is guaranteed to be of the same or a higher resolution than the ship navigating the environment. This facilitates faster computation on shapes with lower counts of vertices and edges, while simultaneously keeping the resolution within reasonable bounds for the example applications demonstrated later in this paper. Lastly, the Shapely method **simplify** also provides the option **preserve_topology**, and is in this work set to true in order to avoid polygonal deformation.

C. FEATURE SELECTION

In this section, the data extraction hierarchy mentioned in Section II-A will be discussed in more detail. By selecting specific regions for the extraction of depth data (the yellow rectangle of Figure 1), the package may filter out any unnecessary regions from the full ENC data set, such that only the points and polygons of the area of interest are constructed and stored as local shapefiles. An example extraction view of such an area is presented in Figure 5, showing a square region of Ålesund smaller than the region presented in Figures 2, 3 and 4.

Next, one may isolate an even smaller shape handling subregion (in Figure 5 shown as the white square defined by some coordinates provided by the user) after reading the shapefiles to memory during runtime, and perform spatial operations on these points or polygons. Furthermore, if the package is used by e.g. an ANS, it may construct another artificial extraction window or dynamic horizon polygon defined by e.g. the pink disk as in Figure 5, or any other shapes like rectangles or general polygons. This dynamic region may subsequently be utilized to isolate all features of interest within the horizon such as e.g. grounding obstacles or nearby vessels for collision avoidance efforts, using methods provided by the SeaCharts package or the Shapely library directly.

The white square is intersected with all of the data points located within it, effectively filtering out all features not inside the rotated square. Here, the polygon merging and seabed range consolidation methods (i.e. the user-specified depth ranges from Section II-B) may be used as a filtering technique to extract ''safe'' or feasible sea-faring regions for any ship, given user-specified parameters such as a minimum allowed seabed depth e.g. based on the maximum draft of the ship in question. For this purpose, the seabed polygon layer of e.g. 10 m contains any region of the processed data sets that are *deeper* than 10 m, by construction.

Inversely, all areas of insufficient depths may be calculated by taking the spatial **difference** (Shapely) between the white square polygon and the seabed layer, yielding the inverted shallower areas in red within which all depths are *less* than 10 m. Thus, both the dark green and lighter green land and shore polygons, as well as all seabed depths down to 10 m, are merged together and shown as red polygons in this example. The exterior boundary of the horizon disk around the ship of Figure 5 has an excessively small radius of 1 km, for demonstration purposes only. The interior of the disk is calculated by performing the Shapely operation **intersection** on the horizon disk polygon and the red polygons, resulting in the isolated pink overlay separated from the red.

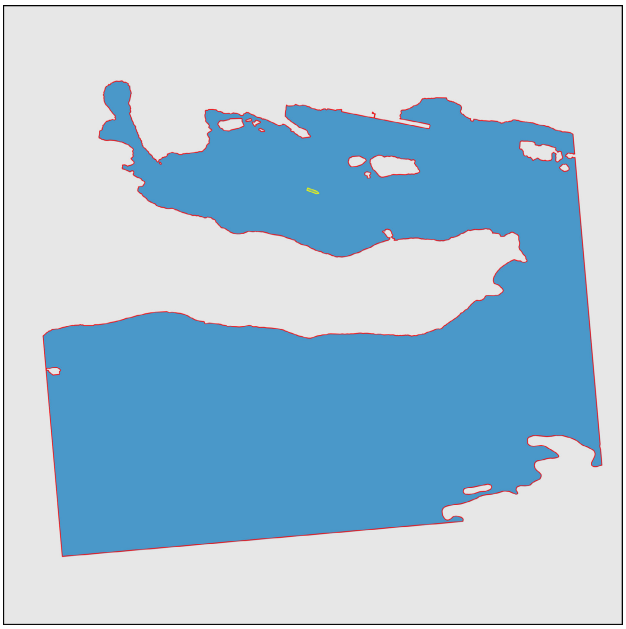

FIGURE 6. Inverted bounded minimum depth region transformation.

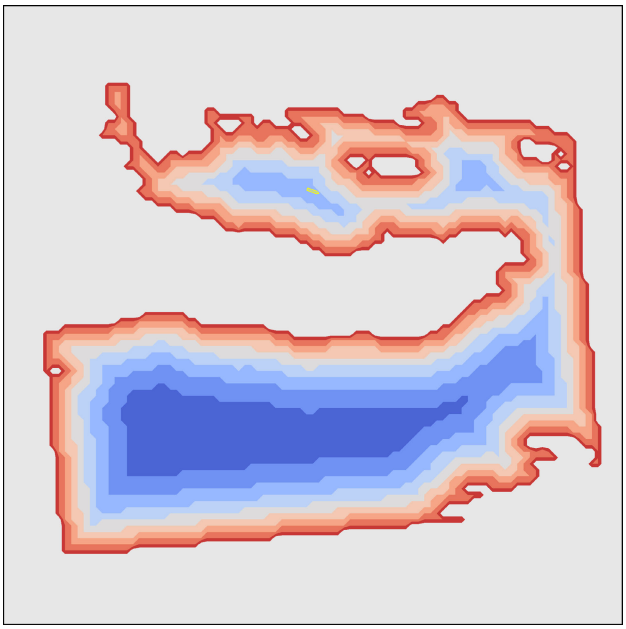

FIGURE 7. Distance-based grounding risk topology example.

An important detail to note in this example is how the selection of the 10 m seabed effectively closes off both of the narrow straits between the individual islands of Ålesund, northwest and northeast of the vessel. This is indeed the expected and desired result, given that both passages are not continuously deeper than the selected seabed layer. As such, appropriate depth bins or depth filters must be selected with care and thorough consideration, based on the possible consequences for any given navigation application.

D. BOUNDED DEPTH REGIONS

Extracted regions of arbitrary shape may be transformed into non-convex or convex regions for path or trajectory planning algorithms, by intersecting the polygon of a bounded region – like e.g. a rectangular view window – with any polygon of interest within it.

Figure 6 demonstrates a variation of such an application, based on the white square region from Figure 5. A new single polygon is constructed by the traversable open water area around the ship within the square with depths deeper than 10 m, using the Shapely methods mentioned in Section II-C. The problem is thus inverted from being based on an open environment with many land polygons, into utilizing a closed environment consisting of a single ocean polygon with islands represented as its inner holes. This transformed topology is inherently finite and bounded by its construction, and may be more feasible for use in autonomous navigation or path planning algorithms. Several overlapping or joined sets of these smaller bounded areas may subsequently be used to calculate subpaths from intermediate points along a larger route, possibly yielding significantly faster solver performances than global techniques.

This inverted topology may furthermore be used to facilitate methods for local path planning or risk analysis through distance-based artificial potential fields (APF) [23], [50], in which e.g. the distance from the ship to the nearest shore may be used to indicate risk levels during transit.

An example plot of a rasterized and distance-based risk topology is presented in Figure 7, based on the traversable polygon in Figure 6. Notice how the resolution of the point samples providing the base for the risk contours is lowered and exaggerated in this example for demonstration purposes. This lowering of spatial resolution may alongside with other techniques be used as a tool to facilitate faster performance. Points of higher risk closer to the shoreline or the interior and exterior boundaries of the traversable polygon are shown in red, and points far away from any boundary are given shades of blue. The gray area outside of the polygon is non-traversable land or seabed with insufficient depth. This type of plot may be useful for visualization of distance-based navigation algorithms, or to define constraints and cost functions in optimization techniques performed by external programs. Such rasterized data points may additionally be used directly as an alternative to the vectorized polygon data stored in the shapefiles, but is currently not a supported feature at the time of writing.

E. FEATURES VISUALIZATION

This section presents an introductory overview of the visualization methods currently supported by the SeaCharts package. All figures are produced by utilizing the **show_display**, **refresh_display**, **close_display** during testing, and saving the resulting images through the **save_image** method in an appropriate image format.

1) VESSELS PLOTTING

To visualize the environment, the user firstly creates an instance of the SeaCharts main class. This class must be instantiated by defining an **origin** or **center** coordinate pair

FIGURE 8. Example plots of various vessels in different colors.

(easting and northing) as well as the bounding box **size** in meters, specifying any region of interest. After the environment polygon data produced from depth measurements are displayed, other features such as vessels and other physical structures along with abstract entities such as path references, pointer arrows, enclosing circles or area overlays of interest may be visualized on top.

A magnified view of the Ålesund region from Figures 2-7 is presented in Figure 8, showing a simplistic scene produced and displayed by merely selecting a desired ENC region and plotting a collection of various vessels in different colors within the environment using the **add_vessels** and **clear_vessels** methods. This is considered the main feature of the package.

Furthermore, the procedure for plotting each ship position is designed such that the *Display* may be created and run in parallel with the main calling process by setting the **multiprocessing** initialization argument equal to **True**, in order to update the visualization plot in real time. This separate process repeatedly reads the CSV file containing all coordinate pairs, heading angles and color names as well as other options for each single ship plot to be displayed in the environment. Thus, the normally convoluted operation of visualizing ships in a maritime environment is reduced to writing (or removing) a collection of values to a plain file during runtime.

2) INFORMATION PRESENTATION

In addition to simply showing vessels in the environment, supplementary information overlays may be added to the plot in order to present various aspects of e.g. vessel(s) risks, intent, planning, predictions, or other relevant factors like weather conditions or mission objectives. For this purpose, several basic shape-based visualization methods have been added to the package. Specifically, it is possible to add lines, arrows, paths, circles, rectangles and any other general polygon shape on top of the environment plot through the collection of **draw** methods of Table 1. Various adjustment options are available to these methods. These will however not be noted in detail in this work, and the user is referred to the maintained repository Readme and code documentation for updated versions of all available input arguments for each function. An example demonstration of these are shown in Figure 9.

Centered in the middle, a clock-like structure of various elements is plotted for the purpose of demonstration. Behind the rotated and semi-transparent rectangle in blue, a yellow disk is drawn with a dashed edge, which again encompasses another smaller green circle. The difference in border style and the non-filled interior color of the smaller circle highlights the flexibility of the available plotting variations, and is possible to adjust similarly for all shapes. Likewise, the pair of two-part straight lines in white and magenta are showing the different variations of straight line segments. The thickness of all bordering edges of shapes as well as for lines may be chosen by the user, as is readily apparent from the large plotted arrow in orange. For this shape, the size of the arrow head may be adjusted as well.

Next, there are additionally shown three groups of polygonal overlays in Figure 9. The lighter land and shore polygons of a single island due north in Ålesund is simply a polygon overlay of white transparent color to highlight a region of interest. In the middle of the fjord to the southwest, a cyan shape has the contours of a seabed polygon with a minimum depth of 100 m and a maximum depth of 200 m, given the user-specified depth bins for this example. Thus, this shape outlines an area for which the depth measurements of the ocean are within 100 and 200 meters (assuming the collected data are valid). More interestingly, the group of red isles to the southeast are intersected with the large square with pink dashed edges, once more using the **intersection** method of the Shapely library. These shapes contain areas for which the depth is more *shallow* than 10 m, and are shown with red solid boundaries in addition to a semi-transparent interior color. Note how the demonstration shows that the user may select any subset of shapes resulting from an intersection (or any other spatial operation), as not all of the areas with the same depth within the pink square are highlighted in red.

The optional colorbar to the right of the environment plot is enabled by calling the **colorbar** method, and shows how the colors of the land, shore, and seabed polygons correspond to the depth measurements as grouped by the defined depth bins. The dark green is simply all *land* polygons with a height of 0 meters and above relative to the mean sea level, as indicated by the upwards triangle shape. The lighter green is however chosen to represent all data polygons labeled as *shore* in the downloaded data sets, and is denoted as a value range between 0 m and 1 m on the depth bar legend. The seabed

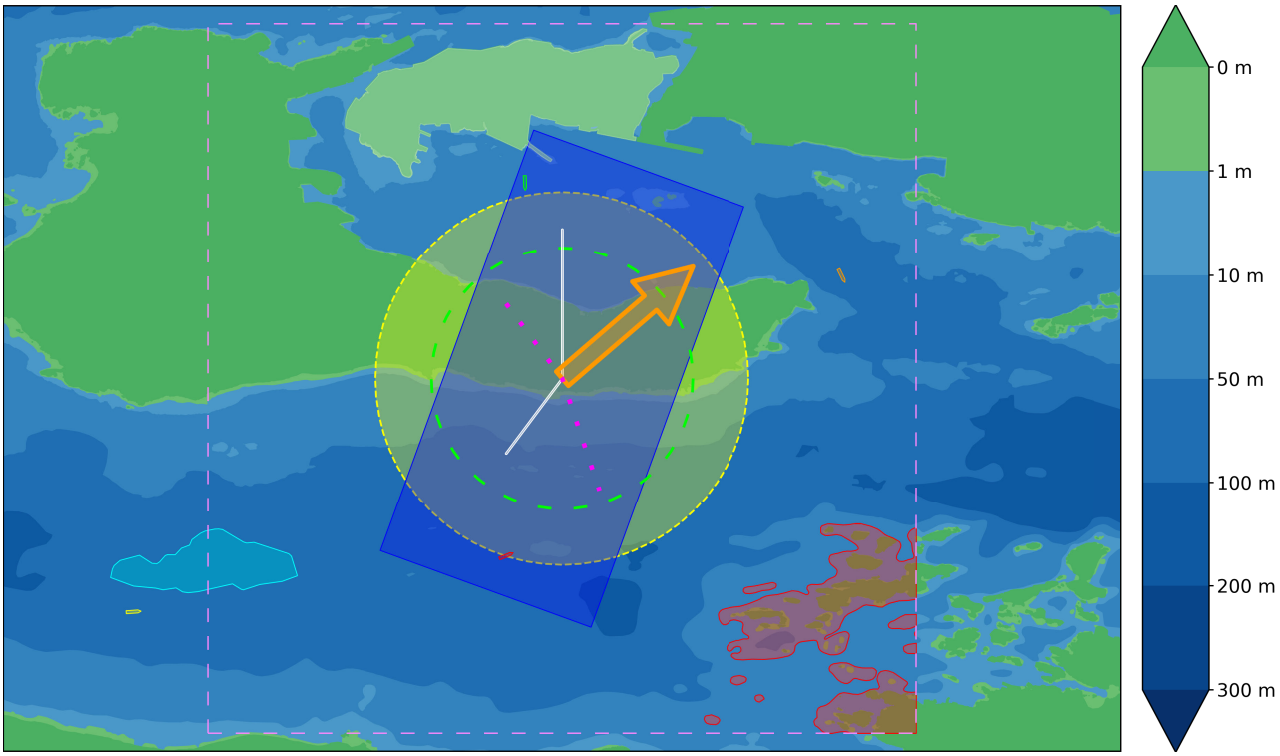

FIGURE 9. Example demonstration of various available shapes and overlays, including a depth bar legend.

depths in blue are intuitively increasing in color intensity in a downward direction, down to the darkest depth at the bottom. As indicated by the legend shape, the bottom layer (here at 300 m) also contains all depth measurements larger, or deeper, than its value. Note however that the depths of the color legend are not required to be present in the drawn environment, and is only dependent on the user-specified depth bins, for consistency.

Lastly, Figure 10 shows the same environment plot with a bounding box window equal to the pink square of Figure 9, with no colorbar added and dark mode toggled on by calling the **dark_mode** method. This optional darker view may aid in providing starker contrasts for the color of the informational shapes and vessels, as well as complying with the established industry standard for e.g. vessel and environment plotting on commercial ship bridges during the night. Thus, this mode remains activated for all remaining figures in this work for increased readability.

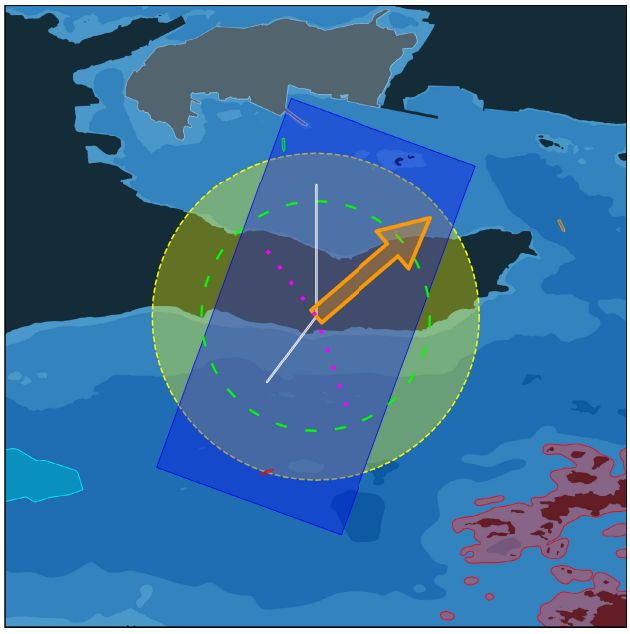

FIGURE 10. Example demonstration of a square dark mode view.

3) PLANNED PATHS

Path plotting is an essential tool for visualizing navigation and ship routes within an environment. Figure 11 presents an example demonstration of a coarse planned path shown in a dashed white line. The start position of a single vessel is displayed in green, and the target destination (before docking) is added as a pink disk. These straight line segments and supplementary shapes may readily visualize a vessel route e.g. given by defined mission objectives or other subsequent and automated path planning schemes. Moreover, a simple arrow pointing from the vessel's start position and to its target crosses a land mass or island located in between, highlighted in red. Additional visualizations such as these may be used to communicate to the viewer e.g. how a path planning algorithm (Section III-E) performs or utilizes available data resources during execution, like in this example.

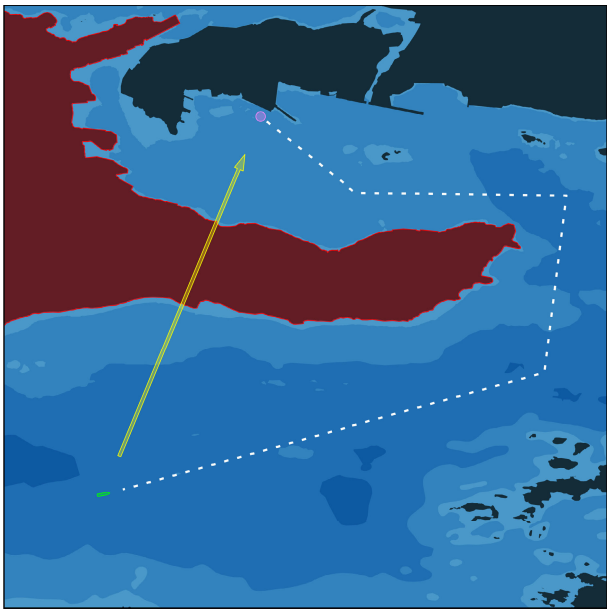

FIGURE 11. Example demonstration of path plotting and additional overlays.

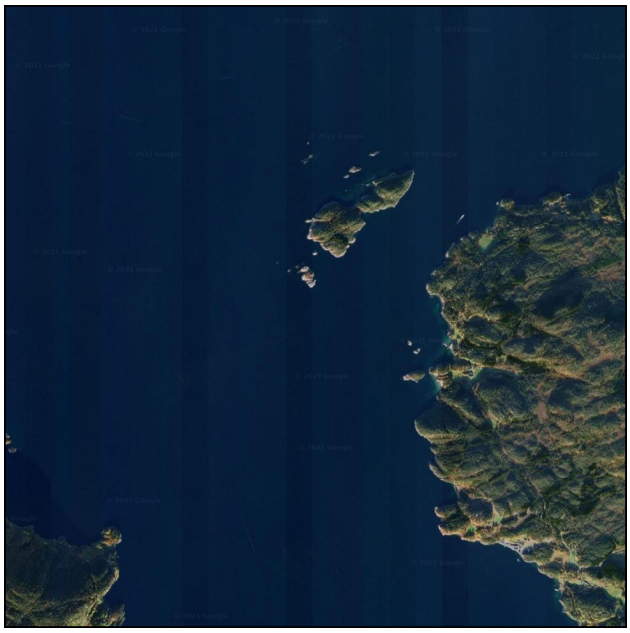

FIGURE 12. Vessel mission area example (Google Maps - satellite view).

III. EXAMPLE USAGE

The following sections present example demonstrations of various API usage within contexts for which the SeaCharts package is intended. The example usages include plotting of trajectories (trails) and simulations of target vessels, visualization of points of interest, intent, and safety domains, utilizing the interactive capabilities of the API for fast ad hoc prototyping, line of sight calculation, and example applications within the fields of collision avoidance, path planning, online risk analysis, and optimal control.

A. TRAJECTORIES & TRAILS

For the rest of this work, a particular area along the coast of Norway is used for demonstration purposes, in order to highlight the capabilities of the SeaCharts package. Figure 12 shows an example view of a fjord area northwest of the ferry dock of *Halsa*, at the border between the Norwegian counties *Møre og Romsdal* and *Trøndelag*. For example, an interesting scenario is path or trajectory planning on each side of the small isle group shown in the middle of the image, with regard to risk analysis and energy consumption optimization purposes.

Figure 13 shows an example visualization of three different ship trajectories with the same origin, in the environment presented in Figure 12. Each color represents a separate trajectory, and are as the other shapes made semi-transparent in order to more easily distinguish between overlapping trajectories, and to make the temporal aspects of each ship path more discernible if vessel polygons overlap. The original planned path for all vessels is represented as green lines with circular waypoints through the isles strait. This straightforward usage provides the user with a basis for plotting of vessel trajectories, which may be useful for risk analysis or more advanced optimization visualization.

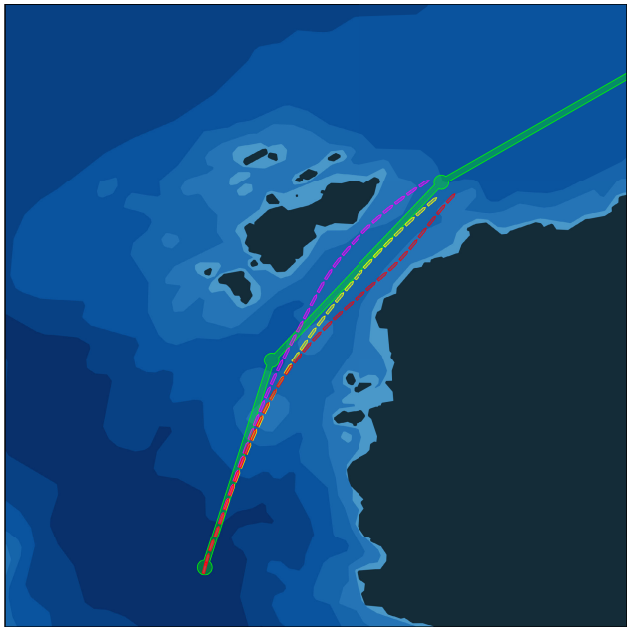

FIGURE 13. Example plot of three overlapping vessel trajectories.

Another natural application of the package is to show several different types of alternative paths or trajectories for a single ship, such as future predictions or past trails of ship poses. Figure 14 presents a visualization of a planned ship path with power blackout simulations at regular intervals. In the environment plot, yellow ship poses are shown along a green pre-planned path with waypoints given as discs. Note that in contrast to the ship trajectories of Figure 13, these ship poses are not produced by simulation of a ship dynamics model, and are generated simply by discretizing the planned path and calculating the appropriate ship heading between each intermediate interval along the path using trigonometry.

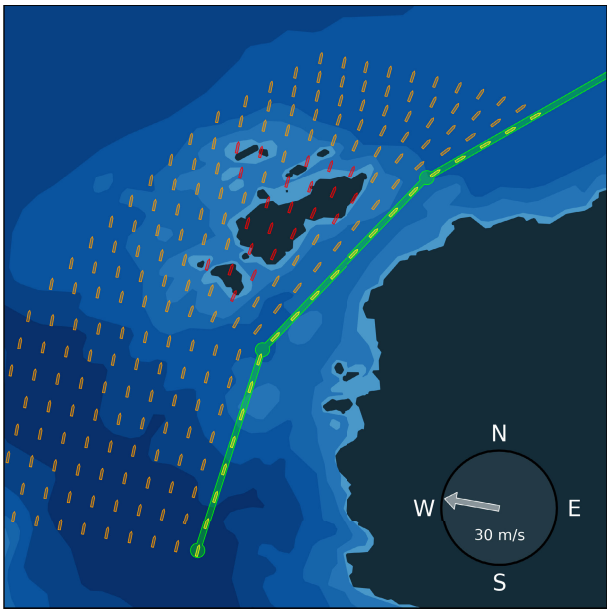

FIGURE 14. Pre-planned path example with wind disturbance simulations.

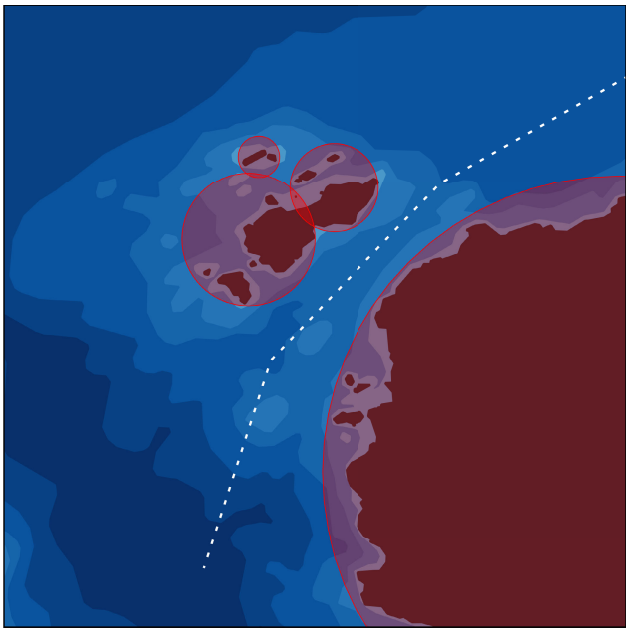

FIGURE 15. Simple vessel path example with danger area disks.

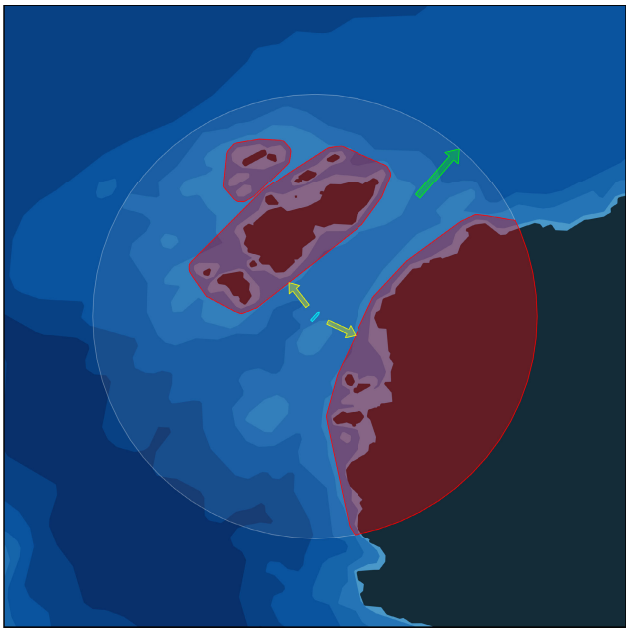

FIGURE 16. Visualization example with red local danger areas and yellow arrows of minimum distances to observable grounding obstacles.

It is thus apparent that the package is not dependent on any past or future (accumulating) inputs for plotting, and may be used quite flexibly for visualization of a large selection of various objectives or information of interest at any time instant during the program cycle.

The ship trajectories in orange are however in this example computed by setting the ship velocity to zero at each intermediate waypoint, and iteratively calculating the next ship pose given a simple ship dynamics model, including a wind disturbance force driving the ship westward e.g. as a result of power blackout or machinery failure. The wind direction and velocity are given in the bottom right corner. All ship poses intersecting with seabed polygons of depths less than 5 m are in this example given a red color, for illustrative purposes. This combined visualization of both pre-planned and simulated paths are shown in a single demonstrative display, to further showcase the capabilities of the package.

B. CONTROL & SIMULATION

For planning and/or control applications that require reduced complexity, the user may wish to display information accordingly. Figure 15 presents a simple environment in which ship poses are replaced by a single dashed line denoting a planned path, along with overlapping red disc overlays with various diameters. Such environment plots may be useful for fast prototyping or simple problem formulations, and are readily available by the provided package methods. The red overlays may e.g. represent abstract circular grounding obstacles for proof-of-concept planning algorithm research [51], allowing for simple radius-based proximity calculations and faster algorithm computations.

For more advanced usage, Figure 16 presents a snapshot of a larger temporal simulation run exported as the graphics interchange format (GIF), showcasing how the spatial operations provided by the Shapely package may be used to visualize dangerous or hazardous areas calculated based on a dynamic horizon radius around the ship. The horizon is shown as a white disk with a radius of 1.5 km around the cyan pose of the ship, and represents the red dynamic extraction window from Figure 1. Similarly to the pink disk of Figure 5, the horizon is also here made excessively small for demonstration purposes only.

The red regions shown inside the disk are generated by first taking the intersection between the circular horizon and

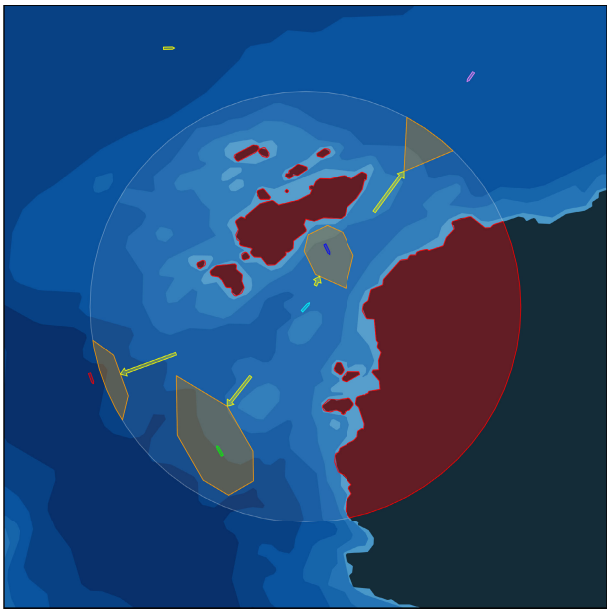

FIGURE 17. Vessel safety domains visualization within ownship horizon in orange, land obstacles in red and yellow distance arrows.

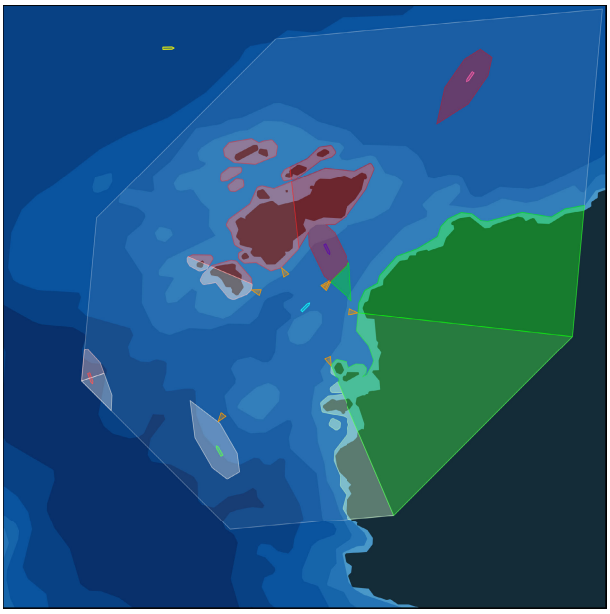

FIGURE 18. Interactive ownship visualization with alternative safety domains for vessels within a sector-based horizon.

all areas with depths less than the maximum ship draft of 5 m, adding a spatial safety buffer of 30 m, and extracting the *convex hull* from each resulting polygon. This is done to demonstrate that the principle of which areas are considered hazardous or impassable is entirely decided by the user.

A green arrow is shown pointing to a target location along some route through the strait. Moreover, the yellow arrows pointing from the ship center to the closest point on each grounding obstacle are calculated by the Shapely method **nearest_points**. Notice however that there is no yellow arrow pointing toward the smallest red shape, as that polygon is not labeled ''observable'' by the ship as a result of the larger red polygon between them. This may be verified in a user-defined algorithm by e.g. utilizing the Shapely method **intersects** on each present obstacle shape with respect to the straight line of sight between the ship and any other obstacle.

C. COLLISION AVOIDANCE

In addition to the concepts related to anti-grounding described in previous sections, a user may also visualize how an autonomous or controlled ship perceives and/or interacts with other vessels within its vicinity. Figure 17 demonstrates how the positions and headings of nearby vessels are used to construct safety domains [41] for each corresponding vessel pose based on given proportional parameters and an ownship horizon. Here, the red overlays highlight only physical landmasses disregarding seabed depths, and the yellow arrows are pointing toward all safety domains of any nearby vessel within the horizon. Notice how in this example the safety domains of each vessel shown in orange are scaled with its own velocity vector by some factor. Thus, the distant ship in pink has its safety domain polygon visible within the ownship horizon due to a significantly high velocity.

Another example demonstration of vessel safety domains with respect to collision avoidance is presented in Figure 18, serving as a base of discussion for the following section. The user is referred to the SeaCharts Readme for further usage details.

D. INTERACTIVE MODE

In addition to the shape plotting methods described in the previous section, the SeaCharts package also includes two interactive programs; ownship and path plotting. Figures 18 and 19 show example plots during such interactive sessions.

1) CONTROLLABLE OWNSHIP

In order to easily plan around or estimate the outcome of various scenarios of interest, the user may activate a controllable ownship to move around in the environment using keyboard keystrokes. Figure 18 exchanges the circular ownship horizon of Figure 17 for a larger ownship domain split into sectors, based on the concept of navigational lights. Here, the safety domain polygons of nearby vessels are of constant size, and are colored according to their orientation with respect to the cyan ownship.

The diamond-like ship horizon polygon is constructed in a fashion similar to the principles of a vessel's navigation lights, split into seven subregions according to the orientation of the navigation lights on the vessel: The starboard side has two regions in green and lighter green respectively from the forward axis of the ship and to the 112.5° mark, and the red regions are similarly mirrored on the opposite port side. In white, the aft direction is split into three such that one may differentiate between objects located within the different subregions relative to the heading of the ship.

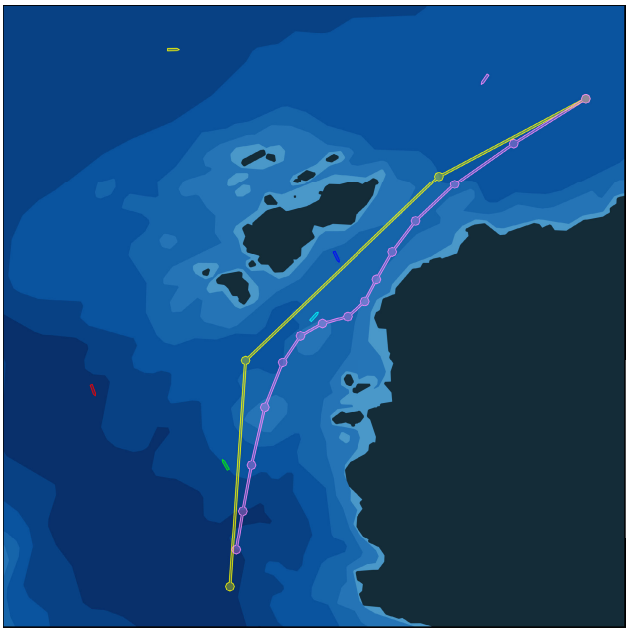

FIGURE 19. Interactive ownship and paths visualization, with original waypoints in yellow and a less crude path shown in pink.

The orange triangles denote the closest point of any selected type of polygon located in each subregion, if it exists. In this example, the user has chosen the seabed layer with 10 m depths. All seabed, land, or shore polygons with depths less than 10 m within the horizon diamond of transparent white are consequently highlighted in tints of green, red and white according to their corresponding subregion color. Thus, there are exactly seven arrows (two overlapping) pointing toward each of the identified regions of different navigation light colors, calculated and stored in a simple CSV file during runtime. The application or user in question is at liberty to freely decide how these closest polygon points are to be utilized for further interpretation. The ownship, arrow triangles, nearby vessels and horizon hazards may all be toggled off and hidden from the environment, if desired. Additionally, the size and proportions of the vessel horizon and hazardous depth filter may be dynamically adjusted during the interactive session by using keyboard keys.

2) PATH DRAWING AND MANIPULATION

Figure 19 presents another snapshot of an interactive session. Here, two independent examples of planned paths are drawn between the same ownship and nearby vessels from Figure 18. One may view the larger yellow path as the coarse ship path planned for the ship through the isle strait, for which four separate waypoints are given. These waypoints may be decided or planned as a subset of a larger mission objective from port to port, expanding further into the environment at either side. The denoted waypoints demonstrated in Figure 19 are part of a subplanning problem constrained within the environment shown, with the ownship horizon and hazards toggled off for clarity.

Algorithm 1 PlanRoutes

Input: grounding obstacles G, safety distance Δd_s, start point σ, end point χ
Output: binary tree R of alternative routes from σ to χ

```
procedure PlanRoutes(G, σ, χ)
    H ← convex hulls of all polygons in G
    I ← dilate H by Δd_s
    J ← spatial unions of all polygons in I
    K ← convex hulls of unions J
    ρ ← straight line segment from σ to χ
    R ← new tree of line nodes with root ρ
    while ∃P ∈ K intersects ∃ρ ∈ R do
        P ← largest intersecting polygon
        ρ ← remove intersecting line from R
        V ← visible vertices of P
        Λ, Γ ← group V into left and right wrt. ρ
        λ, γ ← vertices of Λ and Γ farthest from ρ
        δ ← start point of ρ
        α_{1,2} ← linear line segments from δ to χ via λ
        β_{1,2} ← linear line segments from δ to χ via γ
        R ← add α_{1,2} and β_{1,2} as new line nodes
    end while
end procedure
```

The smoother path in pink may furthermore show how mission control or the autonomous planner aims to follow the main yellow path, in which the time intervals are shorter and the resulting trajectory has a higher resolution. Based on some given operational costs or thresholds, an algorithm may e.g. produce a more detailed and smooth path compared to the coarse main path for a given part of a larger route. Notice how the path in pink attempts to avoid the safety domain of the blue vessel (as shown in Figure 18), creating another significant deviation from the intended path.

The path waypoints for both colors created by the user during the interactive mode are stored in CSV files, and the package may conversely plot paths given by an external program through simple reading of these files. Additionally, the user is able to both move and delete existing path points at any location by appropriate mouse and keyboard commands, allowing for flexible makeshift planning during testing or programming of e.g. ANS.

E. PATH PLANNING

In addition to makeshift planning or prototyping during interactive sessions, the user may also want to display or showcase autonomous path planning e.g. produced by an ANS during or after simulations. Thus, an example path planning algorithm as well as examples of information visualization is presented in this section.

For demonstration purposes, a simple path planning algorithm for constructing a tree of possible route alternatives between two waypoints is presented in Algorithm 1. Note that it is not intended to be a complete path planning algorithm,

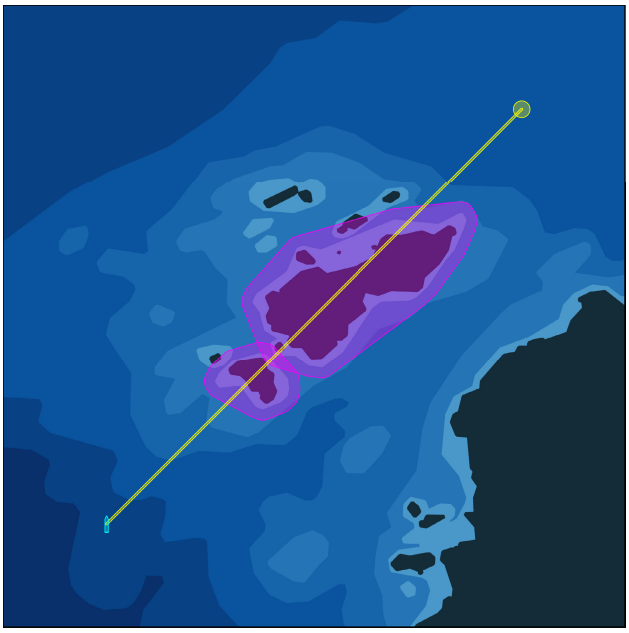

FIGURE 20. Two overlapping polygons from the set of polygons I, which overlap the initial straight line route ρ between start point σ and end point χ.

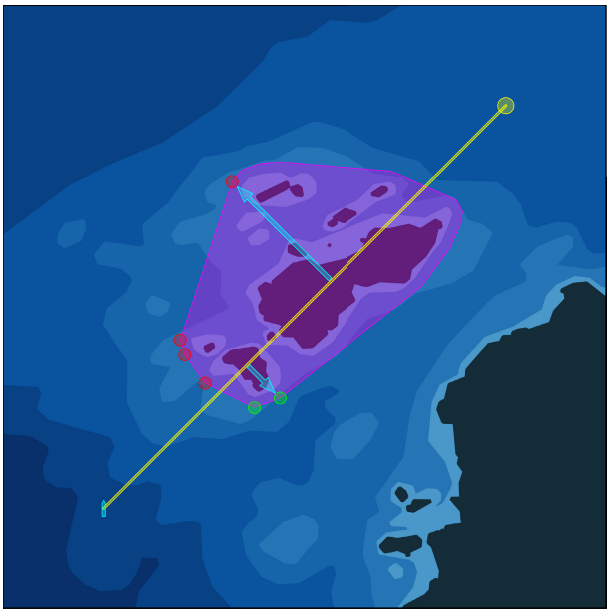

FIGURE 21. Main loop step visualization of the path planning algorithm.

but is merely included in this work to showcase example usage of a selection of SeaCharts methods. The algorithm is given a set of grounding obstacle polygons G, a safety distance Δd_s, an initial starting waypoint σ, and a single end target waypoint χ to which a path with several potential route alternatives is to be planned. The grounding obstacles G may be any two-dimensional polygons of arbitrary shape with any optional depth(s) of interest, and is defined by the user as an input to the algorithm. Similarly, the start point σ may be any point e.g. along a route or the current position of a vessel, the end point χ may be any user-selected target point, and Δd_s is defined as any desired buffer distance.

Figure 20 shows an example in which a vessel intends to navigate around a collection of smaller isles, i.e. the set of grounding obstacles G. The start point σ is represented by the vessel hull in cyan, and the end point χ is denoted by the yellow disk. The initial route path ρ intersecting G is shown as a yellow line from σ to χ. In this example, G is defined by extracting all nearby areas of seabed depths <10 m, which consists of the union of all dark gray island masses as well as the light blue ocean polygons sharing at least one edge with (and fully encompassing) the land masses.

An initialization phase of six steps sets up the algorithm before the main loop is initiated, and consists of the following: The convex hulls H of all polygons in G are computed by accessing the Shapely property **convex_hull**, and the resulting new set of polygons H are subsequently dilated by the safety distance Δd_s (here defined as 50 m), using the Shapely method **buffer** to produce the polygon set of I. In Figure 20, the pink overlay is isolated by selecting all (in this case two) polygons of I that intersect the initial route line segment ρ, for the purpose of visualization only.

The next step calculates the spatial unions J of all polygons in I using the Shapely method **unary_union**, such that any overlapping polygons are merged. The convex hulls K of J are lastly computed similarly to the first step, yielding the final set of polygons to be used in the main loop. This is done to potentially save a significant number of subsequent algorithm iterations, by reducing the number of considered polygons and disregarding all non-convex areas contained within or between the dilated (larger) obstacle polygons of I. The only such polygon in K intersecting ρ is shown as the pink region in Figure 21, demonstrating how the group of isles is reduced to a single convex polygon. Lastly, the initial straight line segment ρ (shown as the yellow line in Figures 20 and 21) is defined by the start point σ and the end point χ, and a new binary tree R with ρ as its root node is created.

After initialization, the main loop of the algorithm identifies the largest (if any) polygon $P \in K$ that intersects with any line segment $\rho \in R$ and extracts all *visible* vertices V of P, filtered as described in Section III-F. Next, these vertices are split into two sets of *left* and *right* (Λ and Γ, respectively) based on their positions with respect to the line segment ρ. These are shown in Figure 21, given the colors red (port) and green (starboard), respectively. This grouping is computed by constructing a triangular polygon between σ, χ and each vertex, in that order. If the resulting polygon is counter-clockwise oriented (asserted using the Shapely method **object.is_ccw**), the vertex is located on the left side of ρ, and vice versa.

The vertices with the maximum distance from ρ in each group (shown in Figure 21 as cyan perpendicular arrows from ρ to each respective vertex) are selected as the new intermediate route waypoints λ and γ, i.e. the minimum distance required to circumnavigate the visible part of the

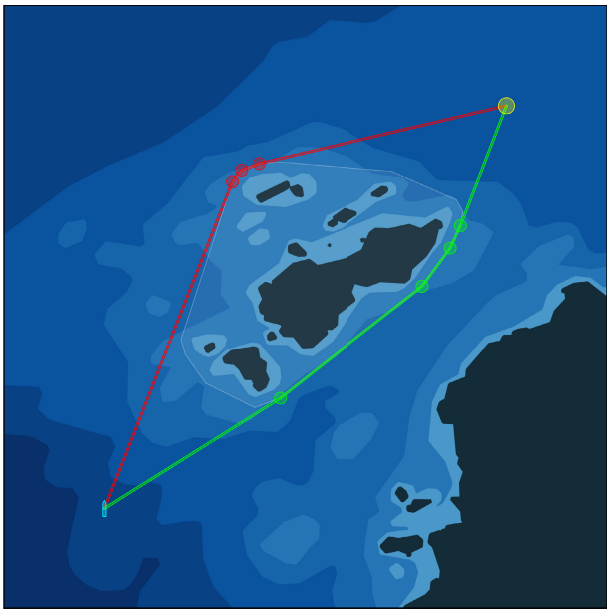

FIGURE 22. Path planning end result visualization of two alternative routes.

obstacle P at each iteration. These waypoints are used to construct two separate splines of straight lines α and β consisting of two linear line segments each, from σ to χ via λ and γ.

These new line segments are subsequently added to the root node of the R tree, leaving two new leaf nodes of line segments sharing the same end target point at χ. If any of the line segments in the resulting tree intersects with any polygon P of K, this process is repeated for that particular line segment, potentially creating more branching nodes along its respective route alternative.

Note that redundant further branching along one side of the obstacle is prevented by the fact that all vertex candidates in that special case are sorted into the same Λ or Γ set. The end result of the algorithm for the demonstrative example case is presented in Figure 22, in which two route alternatives in red and green have been constructed with several intermediate waypoints, generated by repeated iterations of the main loop of the algorithm. These path alternatives may subsequently be used by other navigational optimization schemes, e.g. to select the optimal path with respect to resource consumption or time. Thus, several relevant methods and visualization capabilities of the SeaCharts package is demonstrated.

F. VISIBILITY-BASED ENCLOSING CIRCLES

Throughout development of various applications such as ANS, simplifications may be implemented in order to facilitate faster computation. Advanced shapes such as polygons with many vertices or irregular forms can be transformed into circular approximations, such that only a single point in space along with a radius can be used for rapid spatial calculations in an otherwise complex environment. This may be useful for formulating convex constraints and optimization costs.

Extending this further, any complicated environment may be closely approximated through disjoint or overlapping sets of circles or disks of various desired resolutions. Consequently, it may be useful to demonstrate an approximation technique for simplifying polygons into circles, using available methods of the SeaChart package.

Figure 23 presents the results of the example algorithm **EnclosingCircles**, which calculates *local* polygon approximation circles analogous to the concept of minimal enclosing circles, based only on the currently visible shoreline from any given ship position.

Algorithm 2 outlines an overview of the construction of the visibility-based enclosing circles. Similarly to solving the smallest-circle or minimal enclosing circle problem, the algorithm attempts to construct an enclosing circle that spans all of the *visible* vertices of a grounding obstacle (i.e. island or land) polygon within a relatively small horizon circle around the ship, such that the overlap between the area of the open water seen from the center of the ship and the constructed circle is minimized. Inversely, the open water seen between the ship and the constructed circle is maximized, such that there is minimal discrepancy between the constructed circle and its enveloped polygon. This is based on the assumption that the enclosing circles should map to its original polygon most accurately along the shoreline closest to the ship, as viewed by the perspective of the onboard navigator. Thus, for short term obstacle avoidance purposes, only the immediate surrounding grounding obstacles are acknowledged, disregarding unnecessary considerations of land masses hidden behind obstacles the ship might potentially hit if moving in any straight line from its current position.

Algorithm 2 EnclosingCircles

Input: grounding obstacles G, horizon disk D, ship center s, distance buffer Δd
Output: enclosing circles C of obstacles as seen from ship

```
procedure EnclosingCircles(G, D, s, Δd)
    C ← ∅
    I ← spatial intersection of G and D
    for all P ∈ I do
        K ← ∅
        H ← convex hull of P
        V ← remove nonvisible vertices from H
        for all combinations of v1, v2, v3 ∈ V do
            b1 ← perpendicular bisector of line v1-v2
            b2 ← perpendicular bisector of line v2-v3
            p ← intersection point of b1 and b2
            r ← distance between p and v2 + Δd
            c ← circle with center point p and radius r
            K ← candidate circle c
        end for
        C ← max(K) w.r.t. open water*
    end for
end procedure
```

area of unobstructed water between ship and circle

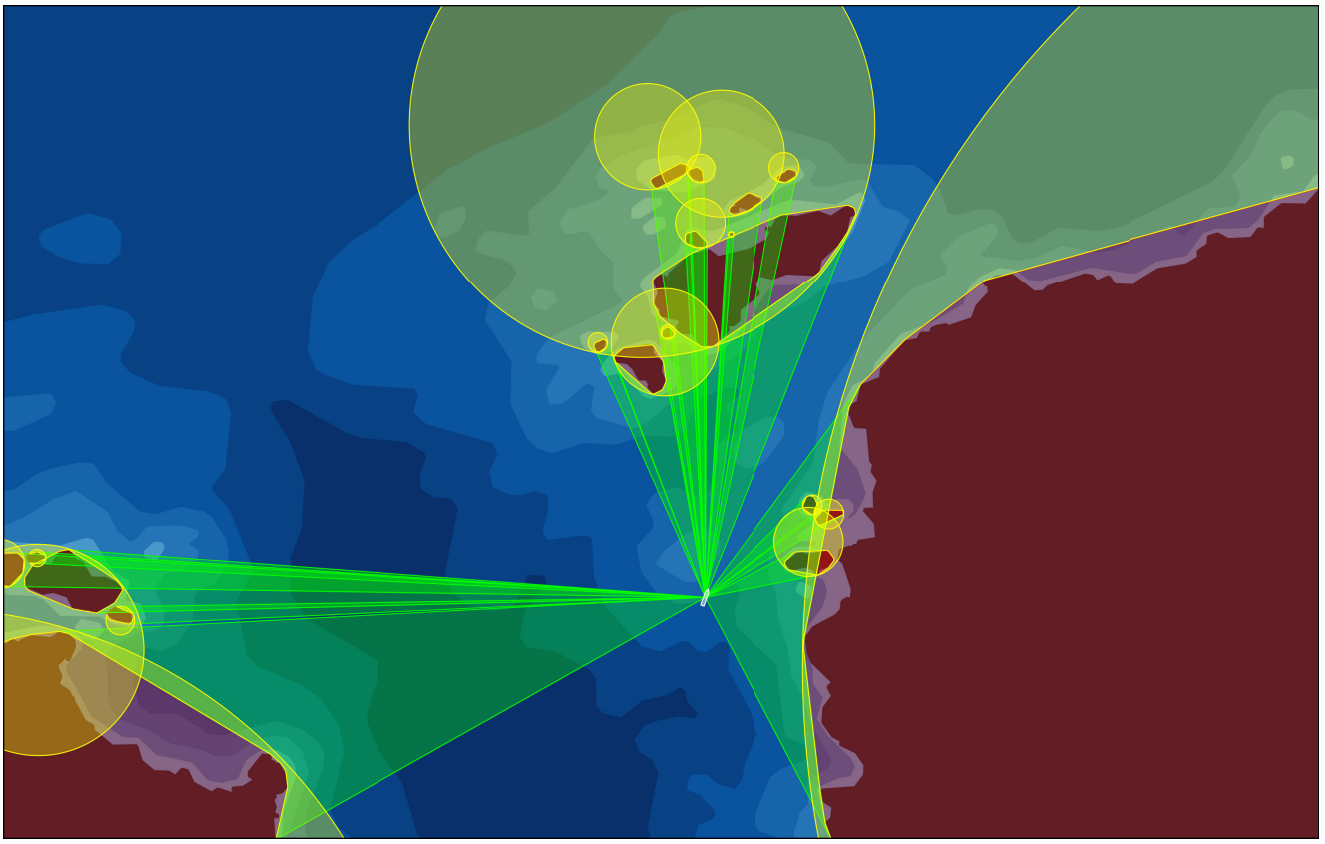

FIGURE 23. End result demonstration of overlapping visibility-based enclosing circles.

Figure 24 presents a diagram showing some of the variables from Algorithm 2 for an example of a single non-convex grounding obstacle $\in G$. The algorithm is firstly initialized by intersecting all grounding obstacles (land and shore polygons in this example) with the dynamic horizon disk D, producing a new set of polygons I. In Figure 24, the resulting example polygon is shown in red color, in which the original grounding obstacle with eight vertices is intentionally clipped by the horizon disk D (here shown as a quadrant) in dashed lines. In Figure 23, however, the entire environment is chosen as the horizon given to the algorithm. In the main loop, each polygon P is used as a basis to compute its individual enclosing circle.

In order to reduce the number of vertices for further computations, the convex hull H of P is calculated – concave crevices or pockets of any polygon are ignored for high-level navigation purposes. The resulting polygon may be identified as the union of the red and orange regions in Figure 24. Thus, it is immediately clear why all polygons are trimmed along the horizon boundary. In a situation in which there exists e.g. a land mass significantly encompassing the current ship position such that the vessel is located within a non-convex crevice, the convex hull of this polygon would remove the feasible navigation area of interest in its entirety.

Moving forward, only *visible* vertices V of H are considered, in accordance with the principle of disregarding

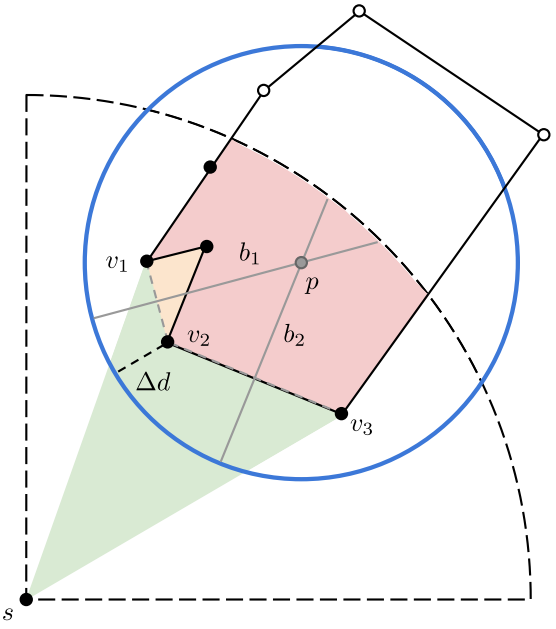

FIGURE 24. Diagram of the construction of perpendicular bisectors in Algorithm 2, with open water* shown in green, I in red, H as the union of red and orange, and the resulting enclosing circle c in blue.

all obstacle topology hidden behind the closest shoreline in any direction from the ship center s. The visibility of each vertex is readily examined by asserting that the line

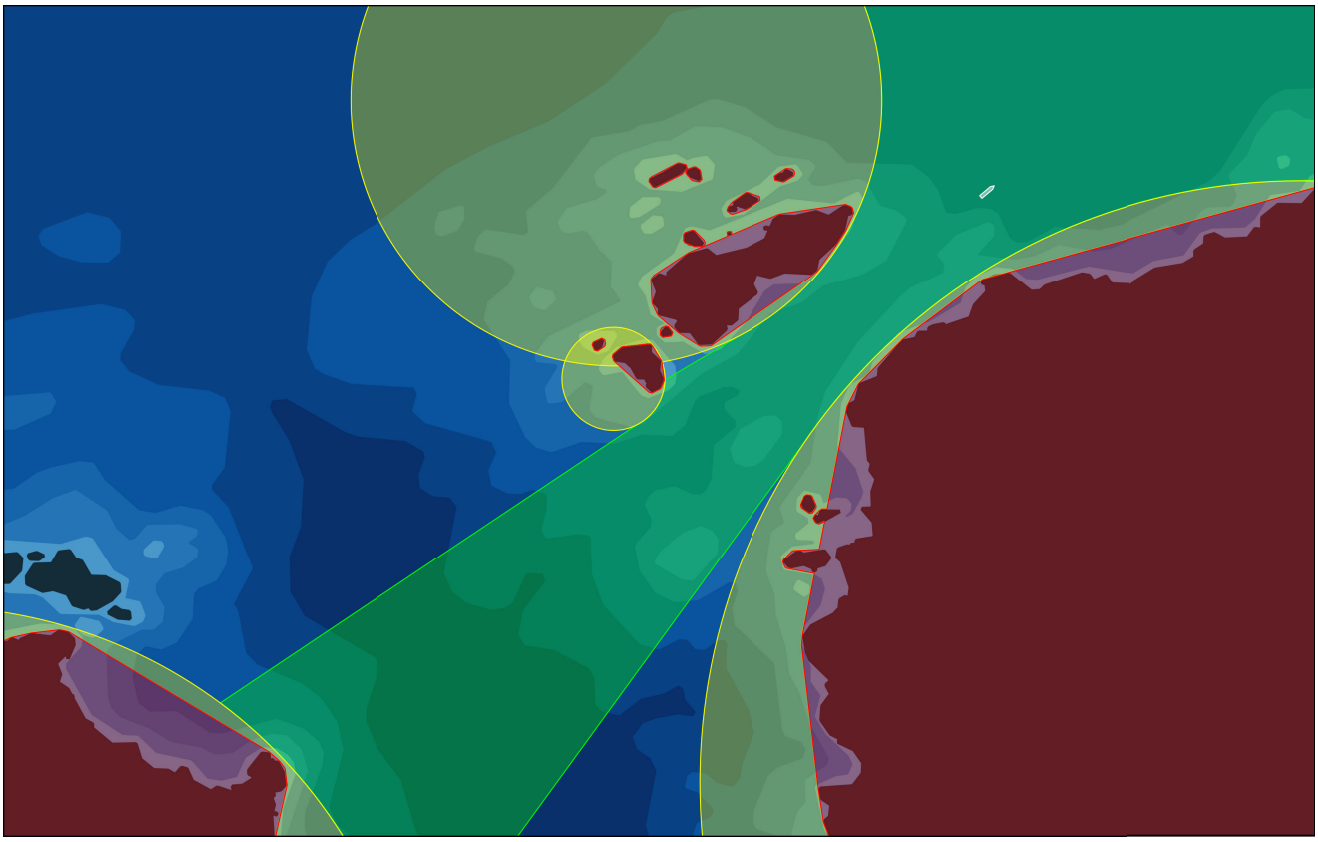

FIGURE 25. End result demonstration of filtered visibility-based enclosing circles and field of view visualization.

constructed from s to the vertex in question does not intersect the interior of H, not including its exterior (boundary). In the simple example in Figure 24, the only visible vertices are v_1, v_2, and v_3 exactly. The secondary inner loop repeats the final steps of the procedure for *any* combination of three vertices from V, i.e. all possible ways to construct two lines from three visible obstacle vertices.

Next, perpendicular bisectors $b_{1,2}$ for both visible vertex lines between v_1 and v_2, and v_2 and v_3, respectively, are calculated. By definition, $b_{1,2}$ are perpendicular lines passing through the midpoint of the pair-wise vertex lines, and as such are extended in each direction with respect to the original environment scope. Thus, the intersection point p between b_1 and b_2 is in general computable, unless the vertex lines are parallel. If this occurs, the candidate is silently disregarded.

Lastly, a candidate circle c is constructed from the intersection point p as its center point, and the radius r of c is set equal to the distance from p to v_2 plus the given input distance buffer Δd. All of the enclosing circles constructed by the inner loop are stored in a set of circle candidates K, which is ultimately sorted by the area of open water left between the ship and the obstacle by each c (shown in green). The more open water is still remaining between the ship and the constructed circle, the more accurate the circle approximates the obstacle boundary shape given the perspective of the ship at the current time instant. The circle with the maximum area of unobstructed water between the ship and itself is thus added to the set of enclosing circles C.

The results of Algorithm 2 may be verified in the example demonstrations displayed in Figures 23 and 25, with different reference points as ship centers. In Figure 23, the convex hulls of each polygon considered in Algorithm 2 are shown in red, and the end result enclosing C are shown in yellow. Green polygons highlight the visible "open water" between each red convex hull and the ship center, i.e. the water surface between all visible shorelines as seen from the ship. It is clear that every vertex of *visible* land or obstructed water is strictly contained within the horizon disk D, and that each obstruction polygon is assigned exactly one enclosing circle. Note however that the entirety of the convex hull of each grounding obstacle need not be fully enclosed by the resulting circle, as is slightly discernible on the second small isle from the west boundary of the environment. Using this method, any radial sector around the ship not covered in green or yellow is in effect considered completely open water, given the specific horizon and ship center given.

This intuitive interpretation and visibility classification may also be useful for field of view procedures, e.g. simulating radar images of a ship's surrounding environment. By identifying all visible shoreline edges within the horizon, one may construct and apply radar-based techniques to a separate layer of the simulated environment for additional

Algorithm 3 ModelPredictiveControl

Input: ownship state x_0, grounding obstacles G
Output: simulated ship trajectory along planned path

```
procedure ModelPredictiveControl(x_0, G)
    d(x, g) ← Shapely distance-to-polygons method
    f(x) ← formulate risk cost function using d(x, g)
    x_s ← x_0
    while not arrived and risk < threshold do
        Π ← construct new NLP using f(x_s)
        s ← optimal solution of solved Π
        u ← first control step of s
        x_s ← apply u to simulate next ownship state
    end while
end procedure
```

situational awareness and decision-making algorithms. Figure 25 shows an example visualization in which a different reference ship center is used to produce a *filtered* view of the resulting enclosing circles and open waters according to the principle of sight lines, further demonstrating the capabilities of the SeaCharts package.

Here, all minor circles fully encompassed in or located behind other circles are disregarded, and the polygons of visible open waters in green are merged and adjusted appropriately, producing the ship's field of view with respect to nearby obstacles. Notice how the circle of the large land mass to the east has changed considerably given the relocated reference point, and that the small group of isles to the west are hidden behind the yellow circles to the north of the plot.

Furthermore, the green open water polygons once again serve as the optimization objective for the end results of Algorithm 2. In Figure 25, only the resulting green region is considered unobstructed or navigable waters. This metric is in this work selected on the premise that only the visible exterior of any nearby polygons is considered e.g. with respect to reactive anti-grounding or collision avoidance, and that any circular boundary completely covering an irregular shoreline should minimize its overlap with otherwise unobstructed open water. The effects of this area maximization are considered adequately sufficient, by comparison of the red polygons against the constructed enclosing circles within the horizon.

G. DYNAMIC RISK OPTIMIZATION

Research on autonomous ships involve (online) risk analysis with respect to anti-grounding and collision avoidance. In this section, the package methods established in the previous sections are further demonstrated by a numerical gradient-based ANS. The example application utilizes functionality and attributes of the *SeaCharts* package to construct an optimal control problem (OCP), transform it into a nonlinear program (NLP) and repeatedly solve it during runtime. Algorithm 3 presents a simplified overview of the main ANS procedure for demonstration purposes, based on model predictive control (MPC).

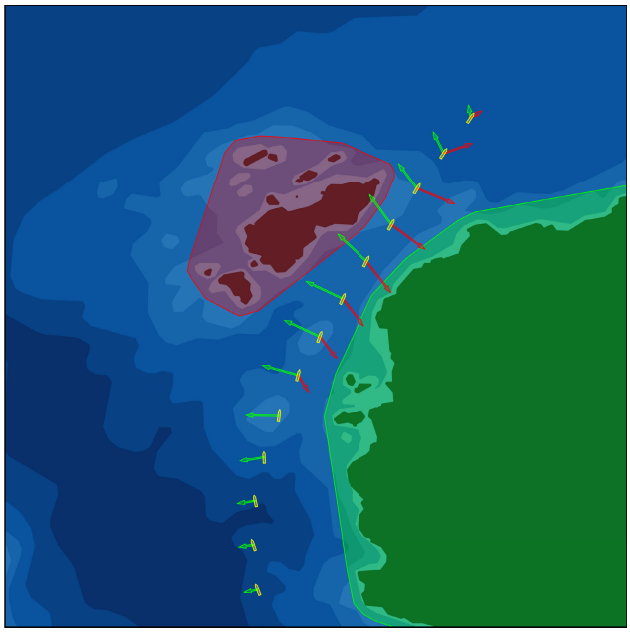

FIGURE 26. Distance-based risk gradient vectors visualization.

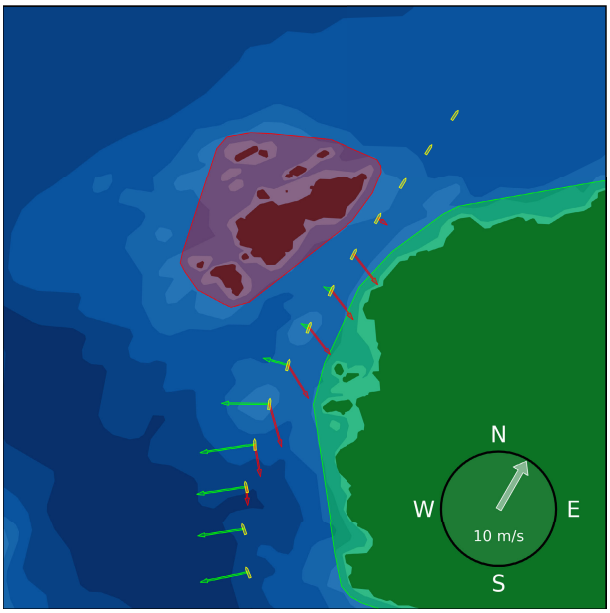

FIGURE 27. Wind disturbance scalar products visualization.

A grounding risk cost function (recall Figure 7) is formulated mathematically as $f(x)$ where x is a state vector, based on the Shapely **distance** method applied to all nearby grounding obstacles G. The distance function is denoted as $d(x, g)$, where $g \in G$ is each individual grounding obstacle polygon within the horizon. Additional mission constraints and risk thresholds for emergency management [51] is considered outside the scope of this discussion.

In this work, $f(x) = \sum_g \gamma \cdot \exp(-\frac{d(x,g)}{\lambda}) + w(x, g)$ where γ and λ are tuning parameters. This form is chosen to scale an abstract measure of grounding risk cost by the distance to all grounding obstacles such that the risk

gradient is exponentially larger closer to land. $w(x, g)$ is an additional wind disturbance cost to be discussed later. In the environment plot of Figure 26, subsequent ship poses of a simulated ship trajectory are shown in yellow.

The colored arrows attached to each ship pose are visual representations of the risk gradients produced by each respective obstacle polygon on each side of the ship path. The direction of each arrow at every time interval is equal to the direction of the unit vector from the closest point of an obstacle polygon and to the center of the ship. The magnitude or length of each arrow increases closer to land due to the inverse exponential scaling, and may as such aid in demonstrating the effects of the risk-based anti-grounding costs for autonomous control.

Figure 27 shows an alternate view of the same simulation plot, this time visualizing the wind-related risk gradients produced by the $w(x, g)$ term of $f(x)$ for a wind disturbance with velocity equal to 10 m/s and direction equal to 30° relative to the North axis. Here, the risk magnitude is proportional to the (positive only) scalar product between the wind disturbance vector and the vector from the ship to each grounding obstacle g, and are similarly to previously displayed as risk gradients directed away from the grounding obstacles [51].

Note how the length of the vectors are only significant when the obstacles are located in an onshore wind direction relative to the ship position, given the scaling based on the scalar product between the wind vector and the vector to the nearest point of an obstacle. Thus, the diminishing vector arrows defined by the green (starboard) obstacle shown for the earlier time intervals of the simulation are negligible and consequently not visible during the later intervals. The scaling factors used between Figures 26 and 27 are not proportional to the terms of the cost function utilized by the ANS, and are adjusted for visual clarity in the example demonstrations.

H. PATH FOLLOWING

Figure 28 considers a more complex path following example in order to further demonstrate potential usage of the visualization tools in the SeaCharts package. Here, the red ship pose is the initial ship state and the yellow ship poses are part of a discretized pre-planned path (i.e. before optimization or simulations) calculated by the ANS given four route waypoints in green (the last one off-screen), written to the ship pose file for some time horizon and an appropriate sampling interval. Similarly to Figure 14 in Section II-E, the planned ship poses are calculated only by simple trigonometry as an initialization step (warm start) of the MPC algorithm, and may as well be valuable to visualize during algorithm demonstrations.

Another example view of the same environment and simulation run is shown in Figure 29, in which the past ship pose trail in white is also shown behind the red ship pose for the current simulation time step. Notice how the yellow future predictions in this example have been computed to comply with the dynamics of the ship, as a result of the trajectory optimization performed by the external ANS. This simple two-part example highlights the flexibility resulting from the visualization module of the SeaCharts package not being dependent on any past or future inputs for plotting of temporal information, and is considered one of the main contributions of this work.

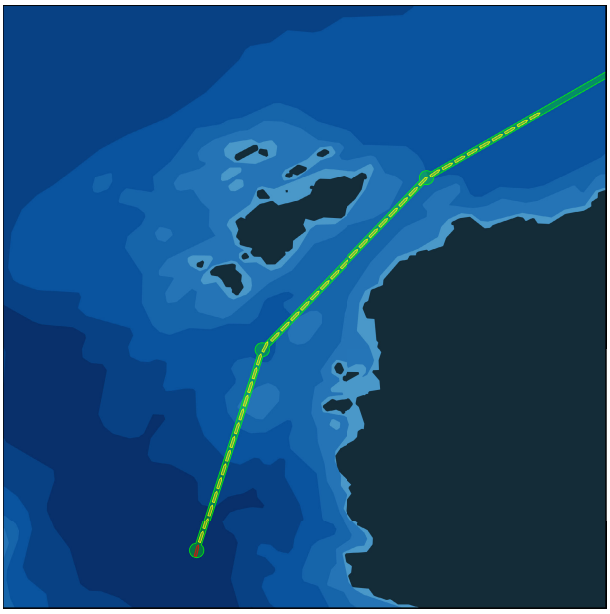

FIGURE 28. Vessel trajectory initialization along the pre-planned path.

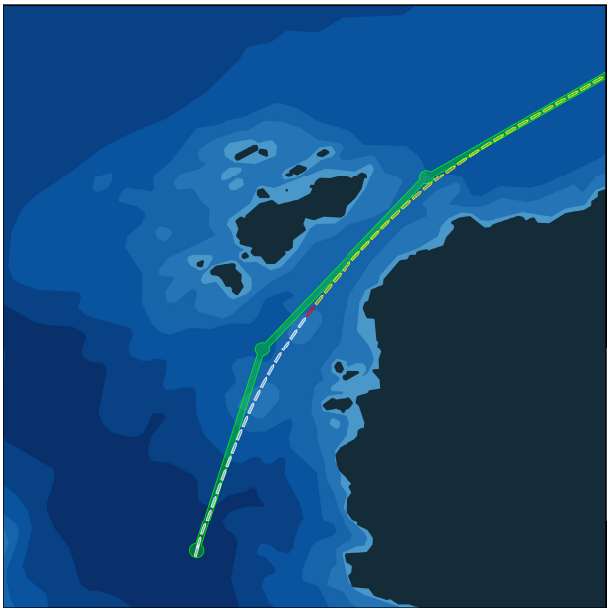

FIGURE 29. Vessel future predictions and past trail during MPC simulation.

The methods discussed above are focused on online analysis of distance-based risk related to grounding obstacles. However, it is proposed that the anti-grounding approaches described in this section may similarly be applied to collision avoidance e.g. based on vessel safety domains [41],

given the general spatial formulations presented and the methods available to the SeaCharts package [52]. If external procedures for predicting or measuring velocities and/or intent of other vessels are included in an ANS, polygon-based risk analysis techniques for grounding obstacles may in general be fused with reactive collision avoidance methods to further enhance the potential of autonomous path and trajectory planning for autonomous ships.

IV. DISCUSSION

This section sums up some of the limitations and areas of improvement for the developed API, as well as points that may provide the basis for future work.

The SeaCharts API is a Python-based package for spatial visualization and computation, targeted at providing methods for fast prototyping and efficient research. As this work presents the very first version of this package, there is vast potential for improvements. Specifically, the API currently only supports the UTM coordinate system / map projection and the FGDB format for spatial data. Moreover, the features that are inherently loaded by the package are currently defined to comply with the feature labels defined by the Norwegian Mapping Authority. Note however that these labels if desired may be added or replaced directly in the source code after installation, for any feature names provided in the FGDB format. Lastly, the installation process may for some be cumbersome if users attempt to install and use the package in non-empty (virtual) environments, due to possible package version mismatches or support conflicts. The package currently uses the *Intel® oneAPI Math Kernel Library* [53] for Numpy/Scipy, which must be properly supported by the environment using the API.

Given the current status of the SeaCharts package and the notes above, it is recommended to continue the development of the API by improving upon its limitations, as well as adding new features and interface methods. For instance, support for the most commonly used coordinate systems or map projections (such as the equirectangular or plate carrée projection) may be useful to integrate into the package. Similarly, support for other file formats for spatial databases may prove beneficial to many potential users. Another suggestion for future work is to attempt to streamline the installation process, e.g. by utilizing the Intel® Distribution for Python. Several additional interface methods may also be added, such as e.g. convenience methods for depth sampling at any location in the plane, and the possibility to change the coordinates and size of the bounding box of the ENC main class during runtime. These are just few of the potential changes and additions that may be made to the SeaCharts package, in order to make the API useful and more practical for researchers and developers within maritime path planning, optimal control, and obstacle avoidance.

V. CONCLUSION

A shortage of versatile and open-source API with simple and user-friendly methods for spatial visualization of maritime environments for research and development has been observed in the literature. In an attempt to fill this need, the open-source Python package *SeaCharts* was implemented and presented in this paper. The package includes demonstrated methods for reading and parsing depth data of known formats, spatial operations for polygon merging and simplification, user-specified features filtering and extraction, visualization of environments, vessels and mission objectives, as well as interface methods for use by external programs such as autonomous systems using spatial data for navigation. Additionally, algorithms for enclosing circle approximations and simplified procedures for path planning and optimization was presented in order to demonstrate potential usage of the package. Ultimately, this API may prove useful for high-level autonomous path planning, control, obstacle avoidance and simulation in maritime environments, by facilitating combined usage of convenient spatial computation and visualization methods for autonomous navigation.

REFERENCES

[1] B. D. MacRae, R. Stephenson, T. Leadholm, and I. Gonin, "Digital chart database conversion into a system electronic navigational chart," Environ. Res. Inst., Ann Arbor, MI, USA, Tech. Rep. #CC-D-15-92, 1992.

[2] A. Weintrit, "The electronic chart systems and their classification," *Annu. Navigat.*, vol. 4, pp. 127–140, Oct. 2001.

[3] A. Weintrit, "Clarification, systematization and general classification of electronic chart systems and electronic navigational charts used in marine navigation. Part 1—Electronic chart systems," *Int. J. Mar. Navigat. Saf. Sea Transp.*, vol. 12, no. 3, pp. 471–482, 2018.

[4] A. Weintrit, "Clarification, systematization and general classification of electronic chart systems and electronic navigational charts used in marine Navigation. Part 2—Electronic navigational charts," *Int. J. Mar. Navigat. Saf. Sea Transp.*, vol. 12, no. 4, pp. 769–780, 2018.

[5] A. Palikaris and A. K. Mavraeidopoulos, "Electronic navigational charts: International standards and map projections," *J. Mar. Sci. Eng.*, vol. 8, no. 4, p. 248, Apr. 2020.

[6] M. R. Mahmud, N. Ibrahim, A. A. Rahman, R. Othman, U. Din, and A. H. Omar, "The development of a low-cost integrated marine navigation system for leisure crafts and small boats," Univ. Teknologi Malaysia, Johor Bahru, Malaysia, Tech. Rep., 2006.

[7] F. Zhu, Y. Zhang, and W. Sang, "Web marine spatial information service based on electronic nautical charts," in *Proc. 8th ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw., Parallel/Distrib. Comput. (SNPD)*, Jul. 2007, pp. 131–136.

[8] G. Park, D. Park, and S. Park, "Design and implementation of display module for electronic navigational chart data," in *Proc. Int. Conf. IT Converg. Secur. (ICITCS)*, Oct. 2014, pp. 1–3.

[9] A. Weintrit, "Radar image overlay in ECDIS display versus electronic navigational chart overlay on radar screen," *Prace Wydziału Nawigacyjnego*, no. 22, pp. 1–5, Oct. 2008.

[10] M. Waz and K. Naus, "Electronic Navigational Chart in aid of generation of multi-dimensional radar display," *Int. J. Circuits Electron.*, vol. 2, p. 4, Dec. 2017.

[11] K. Naus and A. Makar, "Conception of spatial presentation of ENC," in *Proc. XIV Int. Sci. Tech. Conf. Navigat. Support Hum. Activity*, Gdynia, Poland, 2004, pp. 1–5.

[12] L. Hui, X. Shengwei, and Z. Yingjun, "Inland waterway three-dimensional visualization based on 3D-GIS technology," in *Proc. IEEE Int. Conf. Service Oper. Logistics, Informat.*, Oct. 2008, pp. 564–568.

[13] T. Liu, D. Zhao, and M. Pan, "Generating 3D depiction for a future ECDIS based on digital earth," *J. Navigat.*, vol. 67, no. 6, p. 1049, 2014.

[14] J.-C. Morgère, J.-P. Diguet, and J. Laurent, "Electronic navigational chart generator for a marine mobile augmented reality system," in *Proc. Oceans-St. John's*, Sep. 2014, pp. 1–9.

[15] M. Lager, E. A. Topp, and J. Malec, "Remote operation of unmanned surface vessel through virtual reality low cognitive load approach," in *Proc. 1st Int. Workshop Virtual, Augmented, Mixed Reality*, 2018, pp. 1–5.
[16] J. Mkaka and J. Magaj, "Data extraction from an electronic S-57 standard chart for navigational decision systems," in *Proc. Zeszyty Naukowe/Akademia Morska Szczecinie*, 2012, pp. 83–87.
[17] Y. Yu, H. Zhu, L. Yang, and C. Wang, "Spatial indexing for effective visualization of vector-based electronic nautical chart," in *Proc. Int. Conf. Ind. Informat.-Comput. Technol., Intell. Technol., Ind. Inf. Integr. (ICIICII)*, Dec. 2016, pp. 323–326.
[18] S. M. Smith, L. Alexander, and A. A. Armstrong, "The navigation surface: A new database approach to creating multiple products from high-density surveys," *Int. Hydrographic Rev.*, vol. 3, no. 2, pp. 1–15, 2002.
[19] M. Wlodarczyk-Sielicka, "Interpolating bathymetric big data for an inland mobile navigation system," *Inf. Technol. Control*, vol. 47, no. 2, pp. 338–348, Jun. 2018.
[20] B. C. Shah and S. K. Gupta, "Long-distance path planning for unmanned surface vehicles in complex marine environment," *IEEE J. Ocean. Eng.*, vol. 45, no. 3, pp. 813–830, Jul. 2020.
[21] M. Candeloro, A. M. Lekkas, and A. J. Sørensen, "A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels," *Control Eng. Pract.*, vol. 61, pp. 41–54, Apr. 2017.
[22] T. Wilson and S. B. Williams, "Adaptive path planning for depth-constrained bathymetric mapping with an autonomous surface vessel," *J. Field Robot.*, vol. 35, no. 3, pp. 345–358, May 2018.
[23] C. Shi, M. Zhang, and J. Peng, "Harmonic potential field method for autonomous ship navigation," in *Proc. 7th Int. Conf. ITS Telecommun.*, Jun. 2007, pp. 1–6.
[24] Y. Liu and R. Bucknall, "Path planning algorithm for unmanned surface vehicle formations in a practical maritime environment," *Ocean Eng.*, vol. 97, pp. 126–144, Mar. 2015.
[25] R. Song, Y. Liu, and R. Bucknall, "A multi-layered fast marching method for unmanned surface vehicle path planning in a time-variant maritime environment," *Ocean Eng.*, vol. 129, no. 1, pp. 301–317, Jan. 2017.
[26] Y. Liu and R. Bucknall, "Efficient multi-task allocation and path planning for unmanned surface vehicle in support of ocean operations," *Neurocomputing*, vol. 275, pp. 1550–1566, Jan. 2018.
[27] Y. Ma, M. Hu, and X. Yan, "Multi-objective path planning for unmanned surface vehicle with currents effects," *ISA Trans.*, vol. 75, pp. 137–156, Apr. 2018.
[28] B. Shah and S. Gupta, "Speeding up A* search on visibility graphs defined over quadtrees to enable long distance path planning for unmanned surface vehicles," in *Proc. Int. Conf. Automated Planning Scheduling*, 2016, vol. 26, no. 1, pp. 527–535.
[29] Y. Singh, S. Sharma, R. Sutton, D. Hatton, and A. Khan, "A constrained A* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents," *Ocean Eng.*, vol. 169, pp. 187–201, Dec. 2018.
[30] R. Song, Y. Liu, and R. Bucknall, "Smoothed A* algorithm for practical unmanned surface vehicle path planning," *Appl. Oceans Res.*, vol. 83, pp. 9–20, 2019.
[31] R. Goralski, C. Ray, and C. Gold, "Applications and benefits for the development of cartographic 3D visualization systems in support of maritime safety," in *Proc. Int. Recent ECDIS, E-Navigat. Saf., Mar. Navigat. Saf. Sea Transp.*, vol. 77, 2011, pp. 1–5.
[32] X. Gao, S. Shiotani, and H. Makino, "The study of effective communication of water depth information for prevention of accidents in marine traffic," in *Proc. 5th Int. Conf. Emerg. Trends Eng. Technol.*, Nov. 2012, pp. 265–269.
[33] I. B. Utne, A. J. Sørensen, and I. Schjølberg, "Risk management of autonomous marine systems and operations," in *Proc. Int. Conf. Offshore Mech. Arctic Eng.*, vol. 57663, 2017, pp. 1–5.
[34] O. A. V. Banda, S. Kannos, F. Goerlandt, P. H. van Gelder, M. Bergström, and P. Kujala, "A systemic hazard analysis and management process for the concept design phase of an autonomous vessel," *Rel. Eng. Syst. Saf.*, vol. 191, Nov. 2019, Art. no. 106584.
[35] A. Vagale, R. Oucheikh, R. T. Bye, O. L. Osen, and T. I. Fossen, "Path planning and collision avoidance for autonomous surface vehicles I: A review," *J. Mar. Sci. Technol.*, vol. 4, pp. 1–15, Jan. 2021.
[36] A. Vagale, R. T. Bye, R. Oucheikh, O. L. Osen, and T. I. Fossen, "Path planning and collision avoidance for autonomous surface vehicles II: A comparative study of algorithms," *J. Mar. Sci. Technol.*, vol. 26, no. 4, pp. 1307–1323, Dec. 2021.
[37] J. Larson, M. Bruch, and J. Ebken, "Autonomous navigation and obstacle avoidance for unmanned surface vehicles," *Proc. SPIE Unmanned Syst. Technol.*, vol. 6230, Apr. 2006, Art. no. 623007.
[38] M. P. Vitus, S. L. Waslander, and C. J. Tomlin, "Locally optimal decomposition for autonomous obstacle avoidance with the tunnel-MILP algorithm," in *Proc. 47th IEEE Conf. Decis. Control*, May 2008, pp. 540–545.
[39] R. Zhen, M. Riveiro, and Y. Jin, "A novel analytic framework of real-time multi-vessel collision risk assessment for maritime traffic surveillance," *Ocean Eng.*, vol. 145, pp. 492–501, Nov. 2017.
[40] E. F. Brekke, E. F. Wilthil, B.-O.-H. Eriksen, D. K. M. Kufoalor, Ø. K. Helgesen, I. B. Hagen, M. Breivik, and T. A. Johansen, "The autosea project: Developing closed-loop target tracking and collision avoidance systems," *J. Phys., Conf. Ser.*, vol. 1357, Oct. 2019, Art. no. 012020.
[41] A. Bakdi, I. K. Glad, E. Vanem, and Ø. Engelhardtsen, "AIS-based multiple vessel collision and grounding risk identification based on adaptive safety domain," *J. Mar. Sci. Eng.*, vol. 8, no. 1, p. 5, Dec. 2019.
[42] J. Zhou, C. Wang, and A. Zhang, "A COLREGs-based dynamic navigation safety domain for unmanned surface vehicles: A case study of Dolphin-I," *J. Mar. Sci. Eng.*, vol. 8, no. 4, p. 264, Apr. 2020.
[43] J. M. Cordero and C. Kastrisios, "Characterizing free and open-source tools for ocean-mapping," in *Proc. ResearchGate*, 2020, pp. 1–4.
[44] C. Barry, N. P. H. Branch, S. Legeer, G. Parker, N. A. H. Branch, and K. VanSant, "Us office of coast survey's re-engineered process for application of hydrographic survey data to NOAA charts," in *Proc. 10th Int. User Group Conf. Educ. Sessions*, Nova Scotia, BC, Canada, 2005, pp. 1–5.
[45] L. Alexander and M. Huet, "Relationship of marine information overlays (MIOs) to current/future IHO standards," Int. Hydrographic Org., 2007.
[46] G. Masetti, B. R. Calder, and M. J. Wilson. (2017). *Pydro White Paper*. [Online]. Available: https://www.hydroffice.org/manuals/whitepaper.pdf
[47] *Python Package Index—PyPI*. Accessed: Nov. 29, 2021. [Online]. Available: https://pypi.org/
[48] R. Renger, A. Cimetta, S. Pettygrove, and S. Rogan, "Geographic information systems (GIS) as an evaluation tool," *Amer. J. Eval.*, vol. 23, no. 4, pp. 469–479, 2002.
[49] S. Gillies. (2007). *Shapely: Manipulation and Analysis of Geometric Objects*. [Online]. Available: https://github.com/Toblerity/Shapely
[50] P. Wu, S. Xie, H. Liu, M. Li, H. Li, Y. Peng, X. Li, and J. Luo, "Autonomous obstacle avoidance of an unmanned surface vehicle based on cooperative manoeuvring," *Ind. Robot. Int. J.*, vol. 44, no. 1, pp. 64–74, Jan. 2017.
[51] S. Blindheim, S. Gros, and T. A. Johansen, "Risk-based model predictive control for autonomous ship emergency management," *IFAC-Papers Line*, vol. 53, no. 2, pp. 14524–14531, 2020.
[52] *PSB-MPC Collision Avoidance with Anti-Grounding*, NTNU ITK, Trondheim, Norway, 2021.
[53] Intel Corporation. (2021). *Intel One API Math Kernel Library*. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html#gs.gy20fm

SIMON BLINDHEIM (Member, IEEE) received the M.Sc. degree in engineering cybernetics. He is currently pursuing the Ph.D. degree in autonomous risk-based decision-making with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Norway. He is also a Researcher at the Department of Engineering Cybernetics, NTNU.

TOR ARNE JOHANSEN (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in engineering cybernetics from the Norwegian University of Science and Technology (NTNU), in 1989 and 1994, respectively. He is currently a Professor at the Department of Engineering Cybernetics, NTNU. He is also a Key Scientist at the Centre for Autonomous Marine Operations and Systems (AMOS), NTNU.