TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Renee Kroon 221523IAPM

APPLICATION FOR GENERATING INPUT DATA FOR COLREG VERIFIER

Master's Thesis

Supervisor: Jüri Vain
PhD

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Renee Kroon 221523IAPM

RAKENDUS COLREGI VERIFITSEERIJA SISENDANDMETE GENEREERIMISEKS

Magistritöö

Juhendaja: Jüri Vain
PhD

Tallinn 2025

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Renee Kroon

19.05.2025

Abstract

As autonomous ships are getting closer to becoming a reality, we face the question of whether or not the current regulations are ready for their adoption and can keep ensuring safety at sea. An existing tool called CVT can be used to verify the COLREG safety rules through simulations, in cases where autonomous ships are involved. The problem with CVT is that the input data creation process is too unproductive to create large and detailed simulations.

The main goal of this master's thesis is to improve the way CVT input data is created. To achieve this we created an application to make it easier and more productive to generate the input data. In the developed application the data can be created and edited directly on a map. We also integrated data from multiple external data sources with our application, making it easy to make use of existing data. Besides analysing existing map applications we also held discussions with experts in maritime safety to improve the functionality of our application.

Our work improves the previous way of creating CVT input data, as it was only possible to create data manually through text. Using our application it is now feasible to create larger amounts and more realistic data, allowing us to verify more situations with CVT. This can help safety experts better analyse COLREG rules in situations where autonomous ships are involved.

The thesis is written in English and is 40 pages long, including 8 chapters, 19 figures and 0 tables.

Annotatsioon

Rakendus COLREGi verifitseerija sisendandmete genereerimiseks

Autonoomsete laevade tehnoloogiate arenedes jõuab nende kasutuselevõtt pidevalt lähemale. Et selleks valmis olla peame me teadma kas hetkel kehtivad regulatsioonid on piisavad, et tagada ohutus merel ka pärast autonoomsete laevade sekkumist liiklusesse. CVT on tööriist mida on võimalik kasutada COLREGi ohutusreeglite verifitseerimiseks simulatsioonide abil autonoomsete laevadega seotud olukordades. CVT kasutamist takistab selle jaoks sisendandmete loomise protsessi ebaproduktiivsus, mistõttu pole realistlik luua suuri ja keerukaid simulatsiooni olukordi.

Käesoleva magistritöö põhieesmärk on CVT sisendandmete loomise protsessi paremaks tegemine. Selle saavutamiseks arendasime me rakenduse mis teeb sisendandmete loomise protsessi lihtsamaks ja produktiivsemaks. Loodud rakenduses on võimalik sisendandmeid luua ja muuta kaardil. Lisaks integreerisime me rakendusega andmeid mitmest välisest allikast, muutes nii eksisteerivate andmete ära kasutamise lihtsaks. Selleks et tagada, et loodud rakenduses oleks vajalik funktsionaalsus olemas, analüüsisime me eksisteerivaid rakendusi ning pidasime nõu merendusohutuse valdkonna ekspertidega.

Meie töö teeb varasema CVT sisendandmete loomise protsessi lihtsamaks, kiiremaks ja produktiivsemaks, kuna varasemalt oli võimalik sisendandmeid luua vaid käsitsi ja teksti kujul. Loodud rakendust kasutades on võimalik luua suuremal hulgal ja realistlikumaid sisendandmeid, mis võimaldab CVT abil rohkem situatsioone verifitseerida. See aitab ohutuse ekspertidel ja -uurijatel paremini COLREGi reegleid autonoomsete laevadega seotud situatsioonides analüüsida.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 40 leheküljel, 8 peatükki, 19 joonist, 0 tabelit.

List of Abbreviations and Terms

AIS	Automatic Identification System
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CLP	Constraint Logic Programming
COLREG	International Regulations for Preventing Collisions at Sea
CORS	Cross-Origin Resource Sharing
CVT	COLREG Verification Tool
EPSG	European Petroleum Survey Group
GEBCO	General Bathymetric Chart of the Oceans
GIS	Geographic Information System
GSHHG	Global Self-consistent, Hierarchical, High-resolution Geography Database
GUI	Graphical User Interface
HIS	Hydrographic Information System
IHO	International Hydrographic Organization
IMO	International Maritime Organization
IOC	Intergovernmental Oceanographic Commission
JSON	JavaScript Object Notation
MASS	Maritime Autonomous Surface Ships
NMA	Navigation Marks Database
OSP	Open Simulation Platform
UI	User Interface
UNESCO	United Nations Educational, Scientific and Cultural Organization
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
WFS	Web Feature Service
WGS	World Geodetic System
WMS	Web Map Service

Table of Contents

1 Introduction **9**

2 Background **12**
2.1 Shipping and MASS 12
2.2 COLREG 12
2.3 CVT 14

3 Solution requirements **16**
3.1 Functional requirements 16
3.2 Non-functional requirements 18

4 Analysis of existing solutions and data sources **20**
4.1 Existing solutions 20
4.1.1 MarineTraffic 20
4.1.2 Nutimeri 20
4.1.3 GIS systems 21
4.1.4 Electronic navigational charts 22
4.1.5 Summary of analysis 22
4.2 Data sources 23
4.2.1 Shoreline data 24
4.2.2 Water depth data 26
4.2.3 Sea marks and other obstacles 27

5 Implemented application **28**
5.1 Overall architecture of the maritime safety analysis system 28
5.2 Application architecture 29
5.2.1 Front-end 29
5.2.2 Back-end 31
5.3 Integrating external data 31
5.4 Implemented functionality 33
5.4.1 Drawing and editing map features 34
5.4.2 Canvas area 38
5.4.3 Conversion of measurement units 40
5.4.4 Saving and loading data 41
5.4.5 Exporting data for CVT 42

6 Validation **44**
6.1 User feedback 45
6.2 Application testing 46

7 Future work **47**

8 Summary **48**

References **49**

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis **54**

Appendix 2 – CVT input data format description **55**

List of Figures

1 *Basic ship encounter maneuvers regulated by COLREG [10].* 13
2 *Visualization of trajectories calculated by CVT in a two vessel head-on situation [11].* 15

3 *MarineTraffic [14] showing current vessel traffic near Estonia.* 21
4 *Nutimeri [17] showing navigational chart data near Tallinn, with its layer selection menu on the right.* 21
5 *QGIS [20] application showing data from GEBCO and HIS.* 22
6 *OpenSeaMap [24] shows important navigational data like sea marks and traffic lanes, but does not show water depth.* 23
7 *OpenStreetMap [27] shoreline highlighted as orange. Entire country shoreline is available as a single multipolygon feature.* 24
8 *Natural Earth [28] (yellow) and GSHHG [29] (red) shoreline data. The islands are missing in Natural Earth data.* 25
9 *Shaded relief of GEBCO data near Tallinn coastline. Individual pixels can be seen as this is the maximum resolution of data available.* 26

10 *Model of the current maritime safety analysis system. Our application, CVT Map Tool, can be seen in the top-left.* 29
11 *View of the main user interface of the created application.* 34
12 *Different features that can be created in our application. From left: obstacle feature, ship feature, leeway feature, and a feature in the process of being added.* 35
13 *Obstacles feature list with 4 features. The parameters of the selected feature are shown at the bottom.* 35
14 *Selecting the ship's direction. The direction selection button is to the right of the heading input field.* 37
15 *A ship feature with both required (red) and optional (blue) waypoints.* . . 37
16 *Two overlapping obstacle features before and after being made convex (top), and being merged together (bottom).* 38
17 *Canvas with multiple features defined. Canvas area is marked with the blue rectangle. The blue circle in the center marks the canvas coordinate origin point.* 39
18 *Obstacle features before (left) and after (right) being clipped to the canvas bounds.* 39

19 *Import form shown to the user when loading saved data from a file. The leeways line is disabled, as the file does not contain any leeway features.* . 42

1. Introduction

Autonomous ships have gained increasingly more attention in recent years. Although autonomous ships are not yet widely used, new technologies related to autonomous ships and working prototypes are created frequently. In spite of long term attempts to increase the safety of maritime traffic, most accidents at sea still happen due to human errors [1]. As such, the adoption of autonomous ships could help to reduce the amount of accidents, making shipping both safer and more efficient.

To regulate maritime traffic and prevent accidents, the International Maritime Organization (IMO) has created the International Regulations for Preventing Collisions at Sea (COLREG), which is followed internationally. COLREG defines rules on how vessels should maneuver and change signals with regards to each other. COLREG rules apply to all vessels at sea and nothing except technical details in COLREG prevents autonomous ships from sailing, as long as it can follow all the given rules [2].

Following the COLREG rules should prevent collisions between ships, but it can be difficult to ensure this for autonomous ships. This is because COLREG rules are formulated in a way that leaves room for interpretation on how exactly to act in specific situations. For example, one of COLREG requirements is following good seamanship, which is not trivial to define for autonomous ships. As such, even when COLREG rules are followed it might not be enough to guarantee safety, depending on the different interpretation of rules between vessels. This means that COLREG rules by themselves are not enough to control the navigation of autonomous ships.

To explore how autonomous ships following COLREG rules would act in different navigation situations, the COLREG Verification Tool (CVT) has been developed at TalTech University. It allows specifying ships and obstacles and calculates safe trajectories for the ships to follow. This tool is useful for examining different complex situations that could happen under rare circumstances, and might result in unexpected outcomes and dangerous situations.

Using the COLREG verifier currently requires the users to manually input all the data in text form, which makes using it rather cumbersome. For spatial features like ship and obstacle positions, it can be difficult to understand how they relate to each other without seeing the wider context. This problem becomes greater the more data we have. Currently

the only way to see the data visually is to run CVT and look at the generated results in the visualisation tool, which is an application separate from CVT.

The current workflow of specifying input data for CVT is not productive, as manually inserting large amounts of data through text takes a significant amount of time and is error prone. The fact that it is not possible to visually see spatial data unless running CVT and looking at the output further reduces the productivity.

In addition to specific situations, it would also be useful to simulate autonomous ships in more realistic situations, which requires even more data about the situation and the surrounding context. For example, we could make use of real world seafloor depth data, shoreline data, or ship locations and their attributes. Simulating ships in areas that are known to have dense maritime traffic and complex obstacles would be a good way to see what issues arise when autonomous ships use COLREG rules as their basis to navigate. What currently makes using real world data difficult is the fact that it would have to be manually prepared in the textual format required by CVT. As realistic data would require many more data points than synthetic data created for just one specific situation, having to manually input it would reduce the productivity and usability of the tool even further.

If we want to create simulations that are more realistic then we should also consider the different parameters that CVT uses. For example, many input parameters are currently given as constant, whereas in reality they might change over time. In this case it could be worth simulating both the best-case and the worst-case scenario separately. Also, in reality ships usually follow sea lanes and have a set destination they are trying to reach, but currently CVT assumes ships do not want to change their course at all, other than for avoiding obstacles and other ships. Longer scenarios would also include routes with multiple pre-planned course changes, for example when navigating between several harbors or performing maintenance of sea marks.

As such the main research questions of this thesis are as follows:

- RQ1: How to make it easier for the users of the COLREG Verification Tool to gather, view, understand and edit the input data?
- RQ2: How to integrate the developed solution with the rest of the system architecture?
- RQ3: How to validate the solution against real world use cases?

To solve the previously mentioned problems, we will be creating an application where the user is able to create and edit input data for the COLREG Verification Tool in a visually

intuitive way, similar to editing map features on different map layers. The application will also have integrations with sources of real world data to make it easier to create realistic situations.

The created application should make it easy for users to create CVT input data and also reuse data that has been created previously. To evaluate the solution user testing with the new application is needed to verify its usability and identify potential missing features which will be planned as future work.

2. Background

2.1 Shipping and MASS

As innovations are made in the artificial intelligence and machine learning fields, using them as the basis of automation solutions is becoming more prevalent. Fully autonomous and remotely controlled ships are currently being developed and tested in the maritime sector, but the regulation does not account for them yet.

Maritime Autonomous Surface Ships (MASS) is a term used by the International Maritime Organization (IMO) to refer to commercial vessels that at least to some degree operate autonomously [3, 4]. MASS can be categorized into four degrees based on the amount of human involvement, with the first degree meaning only some automated processes and crew still on the vessel, second and third degrees meaning remotely controlled ships, and fourth degree meaning fully autonomous ships with no human control or crew on board [3]. In this work when we refer to MASS and autonomous ships, we mean the ships that are not controlled by humans and navigate autonomously.

Adopting MASS can bring many benefits, such as enhanced safety, improved efficiency and reduced operational costs [4]. Improving the efficiency of shipping would also mean reducing the environmental impact through reduced fuel usage, as the shipping sector generates about 3% of the global greenhouse gas emissions [5]. Accidents also contribute to water pollution, which would be improved by increasing safety.

IMO has conducted trials to assess the readiness of the current maritime regulations for the potential future adoptions of MASS. They have concluded that many of the regulations need to be updated, for example the definitions, and operational requirements of the crew and control stations [3]. The IMO has now released a roadmap to address legal issues related to MASS [6]. One of the key points of the upcoming regulations is that there needs to always be a human that is responsible for a MASS, and can intervene in the actions that the MASS takes if necessary.

2.2 COLREG

To prevent collisions between vessels at sea, the International Maritime Organization (IMO) has specified a set of rules called the International Regulations for Preventing Collisions at

Sea (COLREG) [7, 8]. COLREG contains a total of 41 rules for vessel steering and sailing, the use of lights and signals, and some technical requirements for vessels. COLREG rules apply to vessels in all IMO member states, which includes 176 nations around the world [9]. The COLREG rules that regulate navigation in encounter situations between vessels are rules 6 and 13 through 17 [8, 10].

Rule 6 states that every vessel should always move with a safe speed so that it can take necessary action to avoid collision, even coming to a full stop if necessary [8, 10]. When determining safe speed conditions such as visibility, vessel maneuverability, wind speed, water current and draught all need to be taken into account.

Rules 13, 14 and 15 specify how vessels should act in different approach situations between two or more vessels [8, 10]. Rule 13 covers overtaking situations and states that the vessel overtaking should keep out of the way of the vessel being overtaken. Rule 14 covers head-on situations and states that both vessels should alter their course to starboard to pass each other. Rule 15 covers crossing situations and states that the vessel which has the other vessel on its starboard side must keep out of the way and not cross in front of the other vessel, and instead alter its course to starboard and cross behind the other vessel. Figure 1 depicts how vessels should act when following these rules.

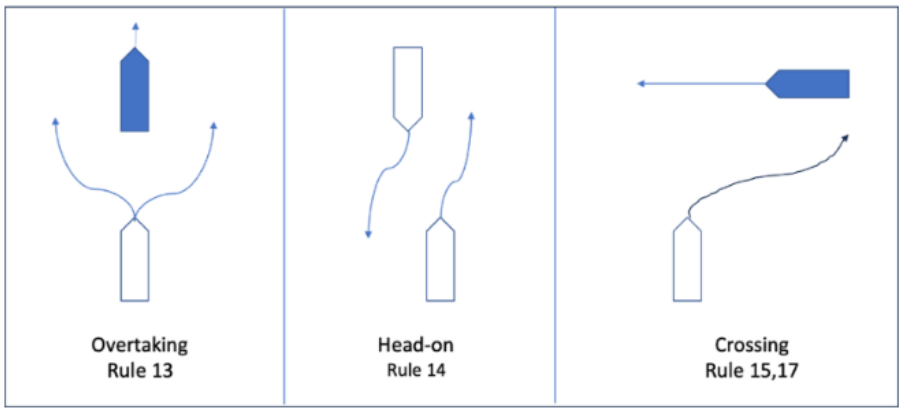

Figure 1. *Basic ship encounter maneuvers regulated by COLREG [10].*

In a situation where one vessel should keep out of the way of the other, the vessel keeping out of the way is called the give-way vessel and the other vessel is called the stand-on vessel. Rule 16 states that the give-way vessel must take action early enough to keep well clear of the stand-on vessel [8, 10]. Rule 17 states that the stand-on vessel should keep its course and speed, only taking action if the give-way vessel does not comply with the COLREG rules and there is a threat of collision [8, 10]. In such cases the stand-on vessel should do its best to avoid collision.

COLREG rules are designed in such a way that they require human interpretation depending

on the situation, as most of the requirements for navigation rules are not precisely specified. Because of this, assuring safety largely depends on the decisions made by the helmsman and crew that operate the vessels. For example, COLREG mentions that vessels should keep well clear and be ready to stop at an appropriate distance, but leaves determining the distances up to the helmsman. As another example, Rule 8 requires that good seamanship should be followed when taking actions to avoid collision.

When considering COLREG rules from MASS perspective, the ambiguity of COLREG means that COLREG by itself is not enough to guarantee safety when driving the decisions of MASS [2]. As COLREG rules are still the basis of navigational safety for human-controlled vessels, MASS should also abide by them to ensure that different kinds of vessels can operate together safely. The decisions made by MASS must be predictable, which requires the rules to be quantitative and unambiguous. This means that a more restrictive and precise decision system based on COLREG rules is required for MASS.

2.3 CVT

The COLREG Verification Tool (CVT), also called CLP-based Verification Tool, is a tool that can calculate close to optimal safe trajectories for vessels [11, 10]. CVT uses COLREG rules as the basis of its trajectory planning, so it can be used to explore how autonomous vessels would navigate. The tool models situations in a simulation environment and is not designed for real-time navigation of actual vessels.

CVT uses Constraint Logic Programming (CLP) to calculate trajectories as constraint system solutions. It uses geometric optimization strategies to constrain the search space, thus reducing the complexity of the difficult optimal path planning problem [11]. CVT can calculate trajectories for a system of multiple vessels and can take many vessel parameters into account. It also allows defining obstacles and leeway areas that will be taken into account when searching for the optimal trajectory. Optimization objective function can be specified in terms of fuel consumption, route length and travel time. The resulting trajectories are verified to be safe and compliant with COLREG rules.

CVT works together with a vessel simulation framework called Open Simulation Platform (OSP) [12]. OSP is an open-source framework for simulating vessels and other maritime systems. It can create lifelike simulations that take vessel dynamic parameters into account.

OSP and CVT are used together to generate more realistic trajectories. CVT will first generate piece-wise linear reference trajectories and OSP iterates on these, taking into account the individual physical characteristics of the vessel and calculating the precise

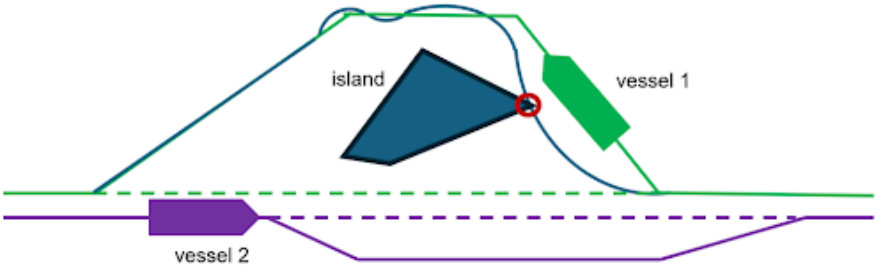

Figure 2. *Visualization of trajectories calculated by CVT in a two vessel head-on situation [11].*

turning curves vessels will take during maneuvers. These new trajectories will then be passed back to CVT and the entire trajectory is recalculated as needed to ensure safety and COLREG rules compliance. Figure 2 displays a situation where trajectories calculated by CVT were deemed as not suitable after the simulation framework calculated the feasible turning radius of vessels.

3. Solution requirements

In this chapter we list the functional and non-functional requirements gathered during the analysis process. The approach taken is to develop a new application, specialized in creating CVT input data via a graphical user interface. The application will be usable as standalone, being independent from the existing CVT visualizer prototypes. From previous experience with other tools and applications, we decided that the best approach to solve the current problems with CVT input data creation is to use a layered map for specifying navigation situations, and have the ability to work with data on the map directly.

In addition to the CVT itself there exist a few other applications in the current CVT ecosystem. CVT GUI [13] is an application that already improves the CVT input data creation process by providing a simple user interface for doing so. This still does not solve all the problems proposed in research question RQ1, as the data is entered as text and numbers, and there is no way to visually see the data in relation to the wider context. Another application that is currently under development is the CVT output visualizer. The output visualizer is designed to view CVT output data, which contains ship trajectories over a time period, on a map background.

When creating our solution we want to make sure that it is compatible with CVT GUI and the CVT output visualizer. This is also part of the research question RQ2.

The gathered requirements seek to answer research questions RQ1 and RQ2.

3.1 Functional requirements

Requirements related to research question RQ1:

1. The user should be able to create CVT objects interactively on a map.
   - The CVT objects we want to be able to create are ships, obstacles and leeway areas. The map of the real world should be used as a base map.
2. The user should be able to modify created CVT objects.
   - It should be possible to change the shape and position of objects on the map, and delete them. It should be possible to do these actions interactively on the map.
3. The user should be able to set and change the parameters of all created CVT objects.

   - Parameters are extra data of objects that do not have to be shown visually on the map.
4. The user should have an option to undo changes made to CVT objects.
5. The user should be able to specify the waypoints of the planned paths of ships.
   - It should be possible to mark waypoints as mandatory or optional.
6. The user should be able to select a canvas area for CVT on a map.
   - Specifying the spatially bounded canvas area is required by CVT, and it should also keep the user from selecting an area that is too large.
7. The user should be able to specify the coordinate system and its origin point for CVT.
8. It should be possible to automatically make obstacles convex.
9. It should be possible to automatically merge overlapping obstacles.
   - It is important to not merge obstacles with different parameters unless it makes sense to do so. This requirement and the previous requirement can also apply to leeway and restricted areas.
10. The user should be able to choose the measurement units used to represent data.
    - We will be providing a list of options to choose from. The units should be taken into account when displaying data as well as when entering data.
11. The user should be able to save and load the created CVT objects together with their map background.
    - Saving to and loading from a file on the user's device is acceptable.
12. The user should be able to combine CVT objects as different map layers from multiple saved sources together.
13. The user should be able to export created CVT objects in the format required by CVT.
    - Only the objects inside the selected canvas area should be exported. Canvas area must be selected before exporting can be done. Exporting to a file on the user's device is acceptable.
14. The application should check data for validity before exporting data for CVT.
15. The user should be able to work with real world data from different sources directly in the application.
    - Data should be loaded on demand as the user requests it and automatically converted into CVT objects, after which it should be possible to edit the objects as required. We want to have different types of data, most importantly shoreline and water depth data. Data should be sourced from reliable external sources.

The most important functional requirements are requirements number 1, 3 and 13. These cover the most basic functionalities that our application should have in order to be able to use it for creating CVT input data.

Requirement number 15 describes functionality that would add a significant amount of value to our application, as currently it would be very difficult to work with large amounts of external data. This is likely the most complex requirement to fulfill, but it should be prioritized.

Requirements related to research question RQ2:

1. The user should be able to export created CVT objects in the format required by CVT.
2. The application should provide necessary information for the CVT Visualiser to visualize the CVT output data.
3. It should be possible to bring the CVT objects created in the application over to CVT GUI and vice versa.
    - CVT GUI works with the same CVT objects, but we may have a situation where CVT GUI has access to some parameters or extra data that our application does not, or vice versa.
4. It should be possible to save the created CVT objects to the CVT GUI database.
    - If we are adding new parameters to CVT objects then we need to consider how the database can support those. It should also be possible to load CVT objects from the CVT GUI database.

3.2 Non-functional requirements

The non-functional requirements are not related to any research question in particular, but concern the general usability of the application.

1. The application should be usable without having to install multiple programs first.
2. The application should provide instructions on how to use its functionality.
3. The settings applied in the application should be persistent.
4. The colors used in the user interface should meet the color contrast standards.
    - We used the web browser's developer tools to determine the suitability of colors used.
5. The user data should not be accessible to unauthorised people.
6. Operations inside the application should not take longer than two seconds.
    - This requirement is important for more complex geometric calculations and the data export itself.
7. Loading external data into the application should take no longer than two seconds.
    - This requirement suggests that we should do the necessary data preprocessing

only once so that it does not have to be done every time the user wants to use the data.

4. Analysis of existing solutions and data sources

4.1 Existing solutions

Before creating our application we analyzed already existing applications that focus on showing and working with specialized data on a map. We had to consider the option that the problems with CVT input data creation could be solved by extending an existing application, instead of creating a new one. We found that many freely usable applications exist for exploring data, but less for editing data. We looked more closely at applications in the maritime domain, which is the domain of our work. Two applications we found that specialized on displaying specific kinds of data are MarineTraffic and Nutimeri.

4.1.1 MarineTraffic

MarineTraffic is a website that shows real-time information about vessels [14]. One of its example use cases is for companies to track the vessels that they own. The site is operated by trade intelligence company Kpler [15] and operates a large worldwide Automatic Identification System (AIS) network for tracking vessels [16]. The site shows vessel current positions and headings on a map of the world, as can be seen in Figure 3, and additional vessel data can be seen by selecting a vessel on the map. MarineTraffic is a commercial product and as such requires a paid subscription to access most of its more advanced features, including more detailed vessel data and vessel past track. Exporting data, either manually or through an API, also requires a subscription, with the amount of exports limited to a specific number per month depending on the subscription tier.

4.1.2 Nutimeri

Nutimeri is a web application created by the Estonian Transport Administration (*Transpordiamet*) for viewing data maintained by the administration, displayed on an orthophoto of Estonia (see Figure 4) [17]. The application can show navigational charts released by the transport administration and data from Hydrographic Information System (*Hüdrograafia Infosüsteem*) and Navigation Marks Database (*Navigatsioonimärkide Andmekogu*) which are also maintained by the transport administration. Nutimeri can also show real-time positions of vessels similar to MarineTraffic, but only in areas close to Estonia. Different types of data are separated into different layers which can be turned on or off. This makes it possible to reduce visual noise by hiding the unnecessary data points.

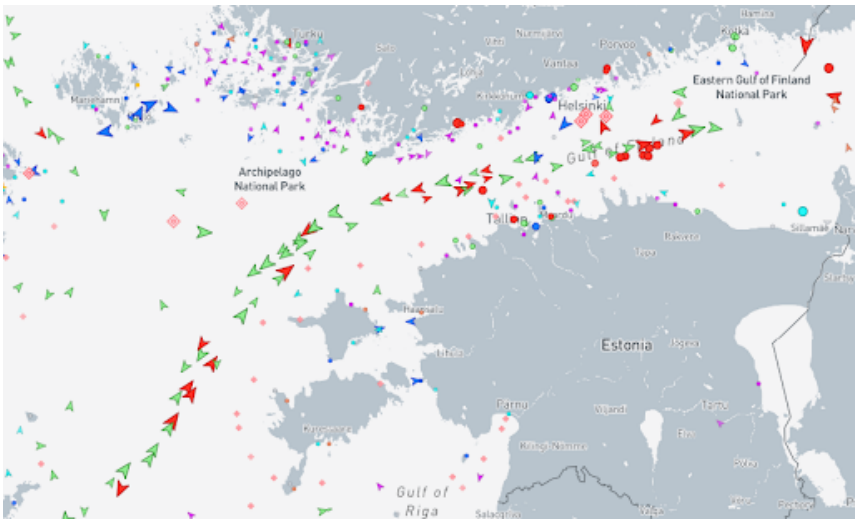

Figure 3. *MarineTraffic [14] showing current vessel traffic near Estonia.*

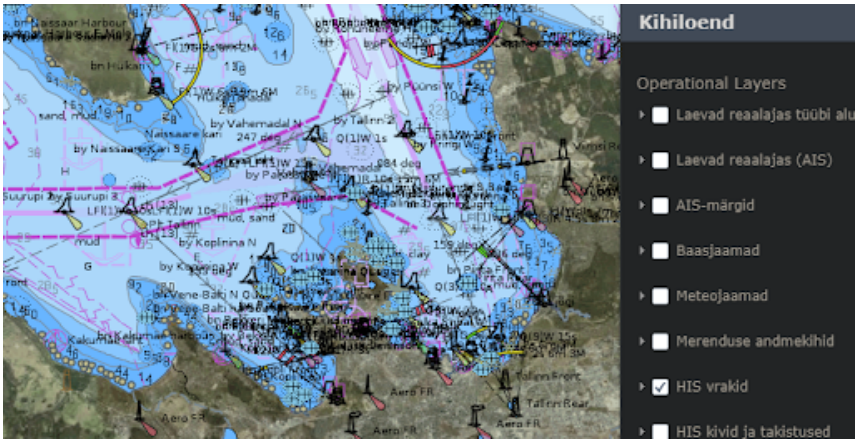

Figure 4. *Nutimeri [17] showing navigational chart data near Tallinn, with its layer selection menu on the right.*

4.1.3 GIS systems

For working with geographical data, a large selection of different Geographic information system (GIS) software has been created over many decades [18]. The most basic feature of GIS tools is editing and viewing geospatial data, but many tools are specialized for certain types of data analysis and visualization. As such, functionalities for creating and modifying map features are well integrated into most GIS tools. These tools are usually complex, offering a wide range of features and supporting many different data formats. One of the benefits of using well established GIS software is that they are more likely to be stable and reliable as software, and have more learning material available. Additionally, if a data provider offers access to its data in standardized Web Map Service (WMS) or Web Feature Service (WFS) format, loading the data can usually be done automatically through a URL. Some examples of GIS software are ArcGIS [19] and QGIS [20] (see Figure 5).

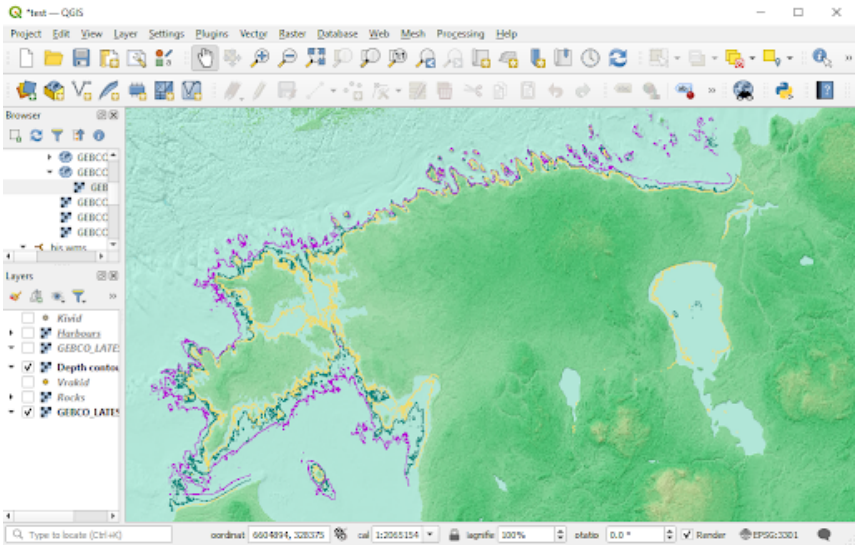

Figure 5. *QGIS [20] application showing data from GEBCO and HIS.*

4.1.4 Electronic navigational charts

Nautical charts are maps that contain important information for marine navigation, such as coastline, water depth, hazards and sea marks. Traditionally nautical charts were released on paper, but in the modern day electronic navigational charts are widely used and integrated into navigational systems for vessels [21]. Charts are released and maintained by hydrographic offices of countries and municipalities, or sometimes by private companies. Electronic navigational charts are stored in specific data formats, like S-57 or S-101, that are not used in other domains, and require specialized software to view [22, 23].

Some web applications have been created to view electronic navigational charts online, for example OpenSeaMap (see Figure 6) [24] and C-MAP [25]. They also have a number of additional features such as route planning and weather data integration. Online chart viewers usually come with a predefined set of navigational charts, and users are not able to load their own charts. An example of software that can load charts provided by the user is OpenCPN [26], which also has features like AIS support and is more similar to electronic chart systems used in larger ships.

4.1.5 Summary of analysis

Many of the previously mentioned applications are proprietary and closed-source, which would make extending them to support our required functionality not possible. Additionally, one important requirement for our solution is to have tools for modifying and creating new geospatial features. If an existing solution does not already support that, then it is very likely that it is created in a way such that adding this functionality would be more

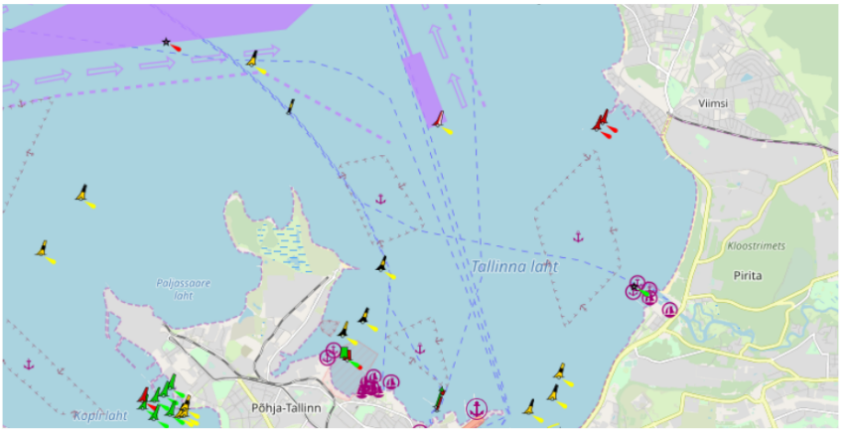

Figure 6. *OpenSeaMap [24] shows important navigational data like sea marks and traffic lanes, but does not show water depth.*

complicated than implementing it from scratch together with data displaying functionality.

GIS tools usually already have map feature editing functionality similar to what we need built in, but their complexity from both the user and developer point of view make them not ideal for our purpose. Also it is not clear how CVT specific data creation would be integrated into their already existing toolset, as users should not be required to have experience with other tools to create CVT input data. As such, we concluded that extending an existing application would not be the best approach to solve CVT input data creation problems, especially when software libraries exist that can make creating our own application easier.

When creating our application we took inspiration from the applications we analyzed to increase the user experience of our application. When displaying ships we adopted an idea similar to one used in MarineTraffic, where ships' headings are clearly shown. Having data grouped into different layers and the ability to toggle between them similar to what Nutimeri has was also one of the features we decided to add, and we used some data sources that Nutimeri also uses.

4.2 Data sources

One of the desired functionalities of our solution is the ability to use data from external sources easily within our application. To achieve this we must first find data that we are allowed to use, which is reliable, and has acceptable quality. For this we analysed different data providers and available datasets. We focused our analysis on freely available data.

The types of data we want to integrate with our application is the data related to maritime

navigation. Most important would be shoreline and water depth (bathymetry) data, but we are also interested in sea marks, obstacles and ships data. We wanted to have data that is mostly consistent when used multiple times. So we chose not to include weather data because it has significant changes every day. Integrating weather data could be done in a future version of the application.

4.2.1 Shoreline data

There are many organizations that have created numerous maps of specific places or even the entire world. Maps usually show all kinds of useful data, and the border between land and water is displayed on most maps. The problem here is that maps are almost always provided as simple raster images, often at multiple zoom levels. What we need in our application is vector data, since this is what CVT requires as its input. In theory it would be possible to extract vector data from a map raster image, however different zoom levels, coordinate systems, colors used to represent map features, and markers and text on the map all make it very complicated to do so reliably. For this reason we preferred using data from sources that already provide vector data that is based on geographic coordinates.

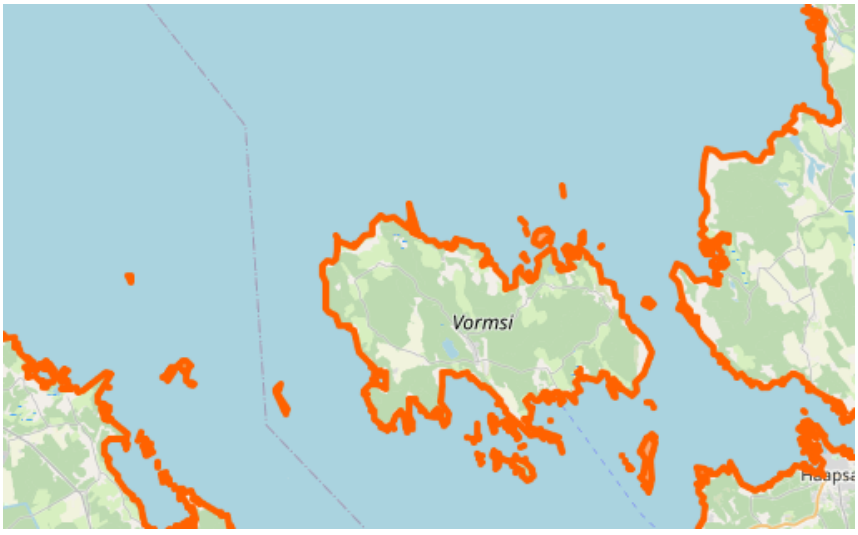

Figure 7. *OpenStreetMap [27] shoreline highlighted as orange. Entire country shoreline is available as a single multipolygon feature.*

One map provider that allows accessing the underlying data used to create the maps is OpenStreetMap. OpenStreetMap [27] is a community driven project that maintains an open-source map of the world. With a large user base, a large number of contributors and much of its data being sourced from national mapping agencies, OpenStreetMap can be considered a reliable source of data. OpenStreetMap is primarily a land-based map so besides shoreline data and commercial ship routes it does not have data that we would be interested in using. Figure 7 shows a shoreline boundary selected in OpenStreetMap,

which can be exported as polygons.

Besides maps, there is also some vector data available on the internet. Natural Earth [28] is a website that hosts geographical data in the public domain, including land polygons for identifying shoreline. At a reported scale of 1 to 10 million it is quite inaccurate and is missing many smaller islands. Global Self-consistent, Hierarchical, High-resolution Geography Database (GSHHG) [29] offers vector data at a noticeably better accuracy, as can be seen in Figure 8. GSHHG compiles its data from multiple sources, basing the shoreline data on data provided by the National Oceanic and Atmospheric Administration of the United States of America. Both GSHHG and Natural Earth offer data as vector polygons, which is the same representation used by CVT.

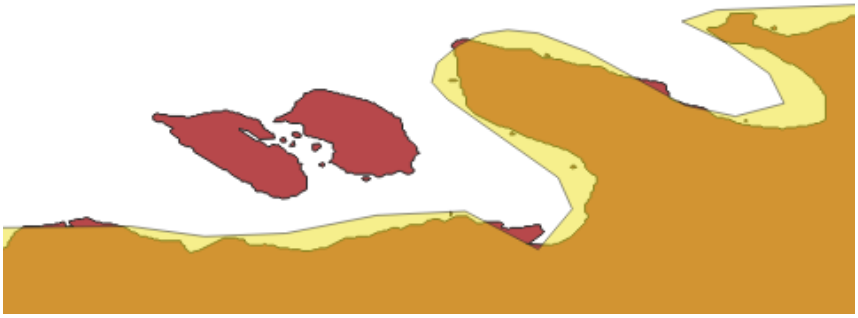

Figure 8. *Natural Earth [28] (yellow) and GSHHG [29] (red) shoreline data. The islands are missing in Natural Earth data.*

Another place to obtain trustworthy shoreline data is from official agencies of governments directly. This data will only cover a specific area and might not be available for certain parts of the world, and the data formats might be different between different agencies, making it more difficult to integrate data this way. As an example the Estonian Land Board makes much of its gathered geographical data freely available in many different data formats as vector features [30].

One aspect to consider when integrating with external data is the size of the data. Higher resolution and better quality data means larger datasets. Larger datasets like GSHHG take longer to load into our application, especially if we need to do some extra processing every time the data is loaded. When we work with data we mostly do so in only a small region defined by a selected canvas area, so in case of a large global dataset it would be beneficial to split it into smaller regions.

4.2.2 Water depth data

Vessels refrain from entering areas where water depth is too shallow for safe navigation. The safe water depth depends on vessel type, as larger vessels need deeper water. For this reason we cannot determine a single safe area for all ships and need to take different water levels into account.

To acquire water depth data we have to look at more specialized data sources. Bathymetry analysis is often done by local authorities and released as part of navigational charts. It is possible to extract data from electronic navigational charts. The drawback of this method of acquiring data is that electronic navigational charts often need to be purchased. Sometimes the data is available directly, for example the Estonian Transportation Administration releases bathymetry data as depth lines and shaded relief map images [31].

GEBCO (General Bathymetric Chart of the Oceans) is an organization backed by International Hydrographic Organization (IHO) and Intergovernmental Oceanographic Commission (IOC) of UNESCO. One of its main goals is to provide freely accessible bathymetry data of the world's oceans and seas [32]. A lot of this data is gathered by different organizations throughout the world and may not be publicly accessible [33], as such GEBCO is a good choice for sourcing bathymetry data if data from any location in the world is desired.

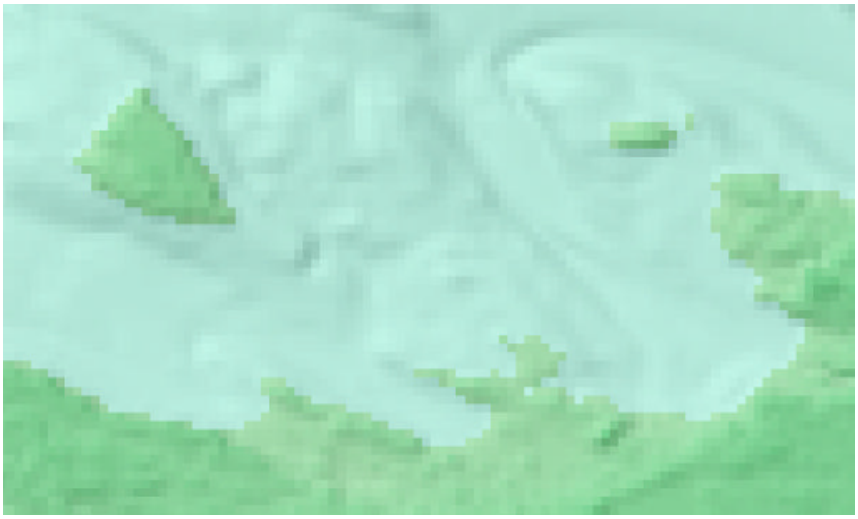

Figure 9. *Shaded relief of GEBCO data near Tallinn coastline. Individual pixels can be seen as this is the maximum resolution of data available.*

GEBCO is constantly seeking to improve their dataset - under their Seabed 2030 Project new data has been published yearly since 2019, with the latest release currently in July 2024 [34]. The dataset uses satellite measurements as base data [35], both for land height and seafloor depth. The satellite measurements as well as the final bathymetry dataset are given as a grid with a sampling interval of 15 arc-seconds, which means one data point for

about every 0.46 km (see Figure 9) with longitudinal resolution increasing further from the equator due to the projection used.

This data resolution is great for open water areas like seas and oceans, but has issues with shallower areas near coastlines. For example, there may be small shallow areas near ports that are unsafe to enter, but are not visible when averaged out with the surrounding 0.5 km^2. As such, GEBCO data can be used for a good estimation and is not suitable for actual navigational purposes. This warning is also issued by GEBCO in their dataset description.

Natural Earth also has bathymetry data available as vector polygons. The polygons are based on depth intervals, where the first interval is between 0 and 200 meters of water depth. This makes this data not useful for our purposes, as all vessels can enter areas where water depth is at least 200 meters.

4.2.3 Sea marks and other obstacles

Sea marks are important for real navigation situations as in addition to marking dangerous areas they can also be used to regulate maritime traffic. As sea marks are only relevant for maritime navigation, information about them is not found in most datasets. One aspect of sea marks that is not common for shoreline or water depth is that sea marks can be easily changed. We would also benefit from including data about other features found in water such as dangerous underwater rocks.

Navigational charts show sea marks as well as all other features that are important for navigation. We have already discussed potential problems with using electronic navigational charts as data sources, such as their availability and data formats. When sea marks change the charts are updated and they would need to be updated in our application as well. If we were to use a global chart aggregator like OpenSeaMap [24] we would additionally have to rely on OpenSeaMap to maintain and update their chart.

Similar to bathymetry data, sea marks data can also be published directly by agencies responsible for maintaining them. The Navigation Marks Database (NMA) [36] of Estonian Transport Administration maintains an up to date list of sea marks in Estonia. Another database managed by the Estonian Transport Administration is the Hydrographic Information System (HIS) [31], which contains data about other dangerous obstacles like underwater rocks and shipwrecks.

5. Implemented application

As previously stated, we decided that the best approach to solve the current problems with CVT input data generation is to create an application that makes it easy and productive to do so. This chapter describes the application that we developed, its technical details and the functionality implemented, and the reasoning behind certain choices.

The application itself consists of a front-end user interface part and a small back-end proxy server. When developing the functionality we primarily followed the requirements described in Chapter 3, but also added extra functionality where deemed necessary.

In this chapter we frequently use the term *feature*. In GIS and mapping terminology *feature* is generally used to refer to geospatial map features – these are all the point, line and polygon based objects that exist on a map. We use this term here in a similar way, to refer to all the singular objects that the user can create on the map – that is the ships, obstacles and leeway areas used for CVT input data. In Chapter 3 we referred to these as *CVT objects*.

When creating the application we also used various software packages and software libraries, which will simply be referred to as libraries in this chapter.

5.1 Overall architecture of the maritime safety analysis system

The current maritime safety analysis system is composed of multiple applications that work together to verify the safety of maritime situations. Figure 10 shows an overview of the different components and how they integrate with each other. CVT is at the core of the system and is responsible for turning situation descriptions into safe trajectories. CVT works together with OSP to generate more realistic trajectories, described in more detail in Chapter 2.3. CVT also has a database where generated trajectories are stored.

The user interface part of the system contains multiple applications for different purposes. Our application exists alongside the CVT GUI [13]. These applications are where CVT input data is created. The input data consists of ship and obstacle definitions, and will be passed to CVT for trajectory calculation. The CVT GUI also contains a database for storing created input data. CVT GUI only allows creating data through text inputs, whereas our application has the ability to create and edit the data on a map directly. Our application

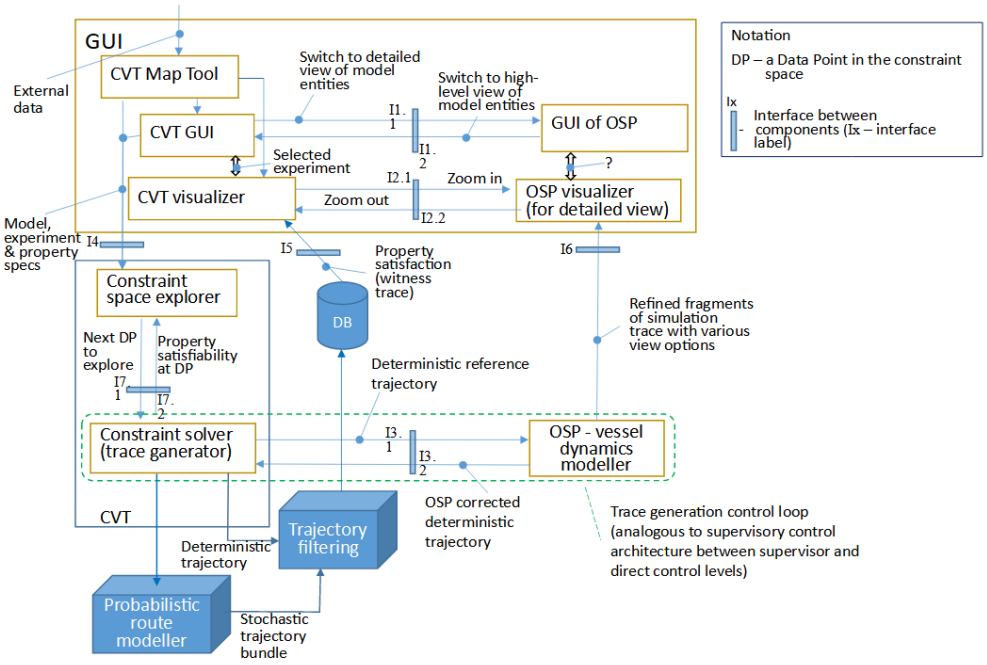

Figure 10. *Model of the current maritime safety analysis system. Our application, CVT Map Tool, can be seen in the top-left.*

also has integrations with various data sources, which is something that the current CVT GUI does not have.

The CVT visualizer is also part of the user interface of the system, and the final part of the data flow. CVT visualizer displays the trajectories generated by CVT on a map. It also has the ability to step through the different timesteps of the CVT output data to see how the ships would move, and some extra functionalities for identifying potentially unsafe situations. The new version of CVT visualizer is currently under development as a part of two students' bachelor's thesis, as the old version of visualizer only had very basic capabilities.

5.2 Application architecture

5.2.1 Front-end

The front-end is the user facing part of the application that contains the main user interface. As improving the process of creating data for CVT is one of our main goals, creating a well made user interface with sufficient functionality is important for achieving that goal. The user interface contains all the tools for viewing and editing CVT input data and the integrations with external data sources are accessible through this part of the application.

As we deemed it important for users to be able to create and modify data on a real map, a way to display a map to the user was required. Although it is possible to achieve this without a software library by just drawing images on a canvas, many libraries already exist created for this purpose. Such libraries are most commonly created for the web environment (viewable through a web browser) and are usually referred to as Tiled Maps or Slippy Maps [37]. Examples of such libraries that are freely usable are: Leaflet [38], OpenLayers [39], Mapbox GL JS [40], Google Maps JavaScript API [41].

Leaflet and OpenLayers are two of the most used tiled map libraries that are also open-source. Leaflet is the more popular library, evidenced by having more downloads on npm, more stars on GitHub, and being used on sites such as Wikipedia [42], OpenStreetMap [27] and MarineTraffic [14]. However, the ability to draw and edit geometric features on Leaflet maps is not supported natively and requires the use of plugins [43]. OpenLayers has built-in support for drawing and editing map features, and based on our testing it provides a better user experience for this purpose.

As drawing and editing map features is one of the primary functionalities of our application, we chose to use OpenLayers as the primary map rendering and geometric features drawing library. OpenLayers makes it easy to use OpenStreetMap as a base map and also has built-in capabilities for reading and writing GeoJSON and converting between different coordinate systems. One example of a website that uses OpenLayers is ADS-B Exchange [44].

As our application can be considered a type of user interface application, developing it for the web browser is appropriate. Doing so also has the benefit of not requiring the user to install or run a separate program on their device. It will also match with the existing CVT user interface and the CVT output visualiser that is currently under development, both of which are also developed to be used through the web browser. One of the drawbacks of this approach is that it makes it more difficult to work with files, as every file read operation must be initiated by the user and every write operation must be confirmed by the user.

For creating the user interface part we chose to use Vue as the front-end framework. Vue is a JavaScript framework designed to make it easier and more efficient to build user interfaces [45]. According to surveys, Vue is currently in the top 3 most used front-end frameworks along with React and Angular [46]. All three are quite similar in terms of capabilities and as such choosing a framework usually comes down to personal preference and previous experiences. We opted to use Vue's composition API over the options API for its benefits of better code layout, better integration with TypeScript, and better performance [47].

For the programming language we chose TypeScript over JavaScript. Creating a web browser based application (and our previous choice of libraries) restricts the language choice, as JavaScript is the only programming language natively supported by all current browsers. As TypeScript compiles to JavaScript it is a possible alternative. The main benefit of TypeScript is its type system, which allows developers to take advantage of the benefits of static typing [48]. Vue and OpenLayers both have support for TypeScript.

5.2.2 Back-end

Because the majority of the required functionality is implemented in the front-end part of the application, there is no direct need for a complex back-end. As we decided that data should be stored in the CVT GUI database, a separate database is not needed for our application.

When attempting to retrieve data from HIS WFS we found that it is not possible to request data through a web browser due to CORS policies. To overcome this problem we decided to create a simple proxy to forward the requests. For this task we chose the Go programming language [49], as it has built-in support for easily creating web servers and has good performance and low resource usage [50].

5.3 Integrating external data

To streamline working with real world data we integrated data from multiple external data sources into our application. Using external data is done through the user interface, where the user has to select an area on the map and the type of data, after which the data will be loaded automatically.

The types of external data we made available in our application are: shoreline data, seafloor depth data, sea marks, dangerous shipwrecks and underwater obstacles data. All of these will be added to the current working set as obstacle features when external data is loaded, with the obstacle type parameter used to distinguish between the different types.

Currently our application only has external data available in waters around the coastline of Estonia. This is partially due to the fact that multiple data sources we used only provide data about Estonia. Since our application is still in the experimental phase and the significance of different kinds of external data should be measured over a longer period of usage, we reasoned that having external data for only a specific area would be acceptable. Instead of integrating external data for different areas of the world, we focused on adding

multiple different kinds of data for a specific region. Estonia was chosen because this is the area the the people who participated in the functionality analysis process were most familiar with.

We obtained shoreline data from OpenStreetMap. The main benefit of using shoreline data from OpenStreetMap is that we also used it as the base map. This means that when shoreline data is loaded it matches with the coastline that is shown on the interactive map.

The shoreline data is not updated dynamically, it was manually exported from the OpenStreetMap database using its Overpass API [51]. Currently only shoreline data for Estonia is available in our application, but it is possible to add data for any region of the world. Since shoreline data should very rarely change, downloading the data once and reusing it should be acceptable for our use cases. OpenStreetMap exports its features in the GeoJSON format, which makes it easier for us to work with it since we also use this format for saving and loading our data.

Seafloor depth data comes from the GEBCO dataset. Although different local data sources could provide more accurate data, the benefit of using GEBCO is that a single source can provide data for any part of the world. Since we decided it was necessary to add data only for waters around Estonia, we chose to only include a small section of GEBCO data in our application, as the complete dataset is very large. We used the GEBCO data download application to only download data near Estonia. Since GEBCO data is updated infrequently and it does not have an API where data can be requested in a suitable format we concluded that requiring manual work to update and expand the data when necessary is acceptable.

The main problem with GEBCO data, as well as any other accurate seafloor depth data sources, is that the data is not represented in a way that CVT can understand. CVT requires obstacles to be polygons represented by the coordinates of their corner points. GEBCO data on the other hand is a pixel grid, similar to a raster image. From other data sources we can also find different data representations, for example depth lines. Some data sources do provide seafloor depth data as polygons, but this data is segmented at very large intervals and does not provide enough accuracy for maritime calculations.

To overcome this issue we decided that converting data from pixel grid to polygons was necessary. Instead of letting the user pick any arbitrary depth value we chose to have three depth intervals that should be relevant for most ships: 5m, 10m and 25m. This way the conversions could be done beforehand and importing the data into the application would be fast. For each of these intervals polygons would be created for all areas where water depth was less than or equal to the given threshold in the source data.

For converting the data automatically we created a script in the Python programming language. Python was chosen as it is a popular language that is often used for creating scripts in cases where program run time is not the most important. The script takes input data in the Esri ASCII format that GEBCO provides and outputs polygon coordinates for each depth interval in plaintext format. The script works by first grouping all connected cells in the input data where depth is less than the threshold, constructing borders around groups and then tracing the borders of each group to obtain coordinates for polygon corner points. When updating or adding new GEBCO data the script must be run on the data to make it usable by our application. The GEBCO data must be downloaded in the Esri ASCII format for the script.

The shipwrecks and submerged rocks data comes from HIS. This data is retrieved on demand via the HIS WFS API, through our proxy server. Only shipwrecks marked dangerous or above sea level are added. For importing underwater rocks the user is able to select a depth and rocks at the selected depth or closer to the surface are loaded. Because HIS data uses the EPSG:3301 (Estonian Coordinate System of 1997) coordinate system but our application uses EPSG:3857 (WGS 84 Web Mercator), converting coordinates between the coordinate systems is also necessary to allow us to import the data.

Sea marks such as buoys are retrieved from NMA. Here we also use our proxy server to download the data, which comes in the SOAP XML format. Sea marks are the most frequent to change out of all the types of external data our application can use, as their positions can be easily changed, and temporary and seasonal buoys are commonly used. Therefore, here we benefit the most from requesting data directly from the data source when the user wants to use it. Different types of sea marks can have different meanings for maritime navigation and some extra data is available in the NMA dataset, but our application does not currently use any of the extra data, treating sea marks as regular obstacles that should be avoided. This could be improved in the future, but currently CVT does not have support for different meanings of sea marks either, and adding support for it is not planned in the near future.

5.4 Implemented functionality

The implemented functionality is accessed by the users through the application's main user interface which consists of two parts: a map view in the center where feature editing is done, and a section with a menu bar and switchable tabs on the left, as can be seen in Figure 11. The map view in the center displays an interactive map of the world, provided by OpenStreetMap. Ships, obstacles and leeway areas can be created and edited on the map, which will later become the input data for CVT. The different types of features exist

on separate layers of the map and can be toggled and edited separately.

Figure 11. *View of the main user interface of the created application.*

The section on the left contains all other UI elements required for working with the application. The UI elements are separated into different tabs and the menu bar is used to switch between the different tabs. Tabs *Leeways*, *Ships* and *Obstacles* are for the three different types of features that the application supports. These tabs contain a feature list with all the features that have been created, and UI elements that enable editing the parameters of the currently selected feature. The *Canvas* tab contains options for area selection and external data importing. Data exporting and loading is done through the *File* tab and application settings can be changed from the *Settings* tab.

5.4.1 Drawing and editing map features

The application allows creating and working with three types of map features: ships, obstacles and leeways. The features are logically separated into separate map layers according to their type. Features of a type can only be created and edited when the tab corresponding to the feature type is selected. Features from other layers will still be visible but cannot be interacted with.

The geometries of the features are created by the user by selecting points on the map. To create a new feature the user must first click the “Add New” button in the feature list, after which the ability to select points on the map is enabled. After the user has selected the desired points, the selection can be finalised by selecting the last point again (or the starting point for polygon-type features). The created feature is then added and is visible from the list of features. Figure 12 shows what features look like in the application.

Existing features can be selected by clicking on them either on the map or in the feature

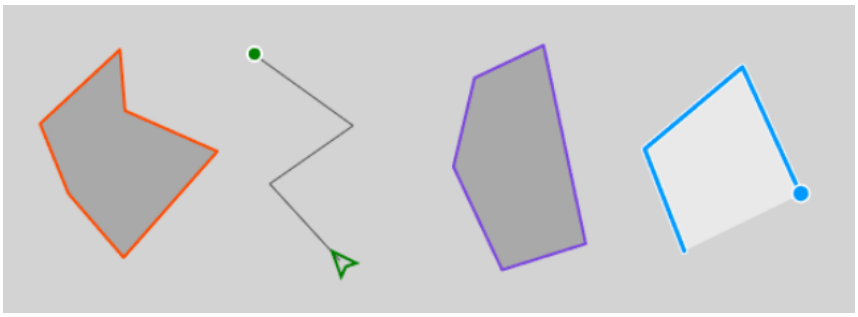

Figure 12. *Different features that can be created in our application. From left: obstacle feature, ship feature, leeway feature, and a feature in the process of being added.*

list. After being selected the feature will be highlighted in both places. Clicking on an already selected feature in the feature list will zoom and pan the map so that the selected feature is visible in the center of the map view. Features can be deleted by clicking the red "X" button corresponding to the feature in the feature list. It is also possible to remove all features of a single type at once by clicking the "Clear" button below the feature list. The feature list is displayed in Figure 13.

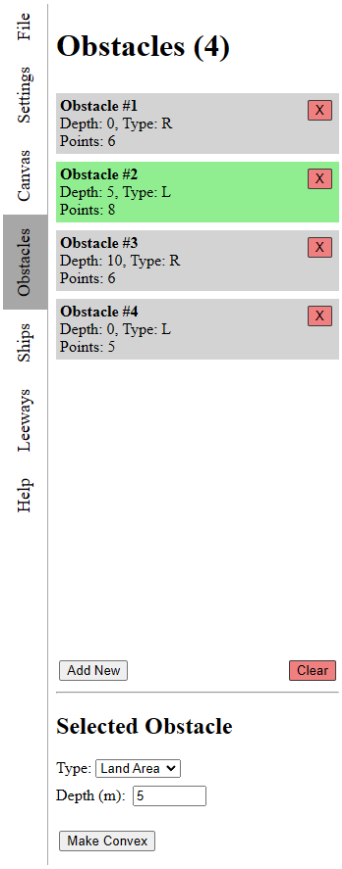

Figure 13. *Obstacles feature list with 4 features. The parameters of the selected feature are shown at the bottom.*

Once a feature has been selected its geometry can be edited. Points of the feature's geometry can be freely dragged around on the map to change the geometry. New points

can be added by clicking and optionally dragging on the polygon edge (for obstacles and leeways) or line (for ships) while holding Control (Ctrl). The requirement to hold Control was added to reduce the chance of accidentally adding new points, and can be disabled in the options menu. Points of the geometry can be deleted by holding Alt and clicking on a point. Deleting is not allowed if it would result in a polygon with less than three points or a line with less than two points.

All feature geometry editing options, including adding and deleting a feature, have support for undo and redo functionality. History is saved separately for each feature type and the currently selected tab determines for which feature group undo and redo actions are performed, similarly to feature editing and selecting. Certain actions, like importing saved features data from a file, will clear the edit history, and the user is warned about this beforehand.

After a feature is selected it is possible to configure its parameters. The parameters will be shown below the feature list and will take effect immediately after the user makes changes to them (see Figure 13). The feature parameters supported by our application correspond to the parameters accepted by CVT. During our discussions with the main creator of CVT, we agreed to add new parameters that previously did not exist, to make input data even more realistic. Newly added feature parameters are obstacle type and depth, and leeway parameters being defined as intervals. The new parameters must also be implemented in CVT, but that is not done as a part of this work. The ship waypoints system, which is implemented through map interactions and not parameters, is also new for CVT.

The obstacle feature represents multiple different types of obstacles that ships might need to avoid. Through setting the obstacle type parameter, we can choose if the obstacle is a land obstacle, a restricted area or a sea mark. The depth parameter defines how deep underwater the obstacle is. As such not all obstacles need to be avoided by all ships, for example when a land obstacle is deep enough underwater that a ship can safely cross over it, or when a ship has permission to enter an otherwise restricted area.

Directional parameters like the ship's initial heading can also be set by choosing a position on the map. After the user presses the button with the circle icon, a red line will be drawn between the feature's position and the user's cursor position to indicate direction selection mode. After clicking on the map, the parameter will be updated with the direction from the feature's position to the clicked position. The process is displayed in Figure 14. The ship's initial direction parameter is also displayed on the map as the direction the ship's icon is pointing towards.

Figure 14. *Selecting the ship's direction. The direction selection button is to the right of the heading input field.*

Ship features are shown as a line string, which connects the waypoints the ship should path through. The lines between the points are always drawn as direct straight lines with the purpose of making it easier to see the order of the waypoints. They do not show the actual path the ship is expected to take, which will be calculated by CVT and will likely differ from the direct lines that are drawn on the map. Waypoints will be highlighted as red or blue circles when a ship feature is selected, with red indicating required and blue indicating optional waypoints. All waypoints are set as optional when a ship feature is created and can be toggled between required and optional states by clicking on a point while holding Shift. The final target position is always treated as required and cannot be changed to optional. Figure 15 displays a ship feature with multiple waypoints set.

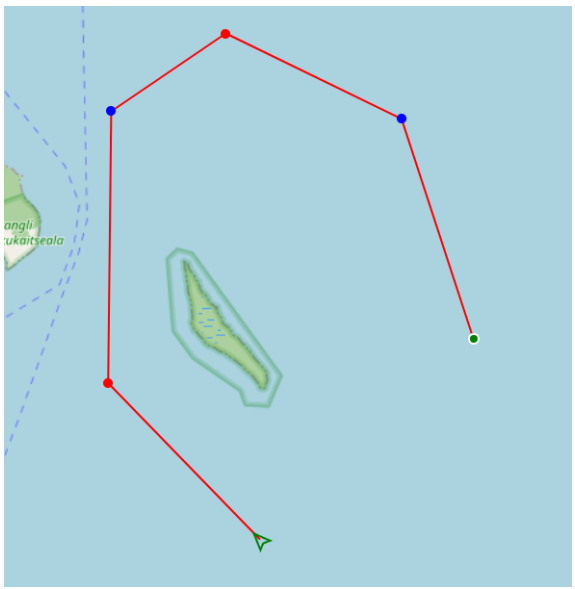

Figure 15. *A ship feature with both required (red) and optional (blue) waypoints.*

To help optimise obstacle polygons for better CVT performance, we added the ability to automatically make obstacles convex and merge overlapping obstacles together. Making obstacles convex can significantly reduce the vertex count of more detailed obstacles. Under normal circumstances ships would also not want to enter the concave areas of

obstacles, so making them convex ahead of time can further simplify the calculations CVT needs to do. To implement this functionality we adapted an algorithm based on the monotone chain algorithm [52]. An example of features being made convex can be seen in Figure 16.

Merging overlapping obstacles also has the benefit of simplifying calculations, and also makes it easier for users to work with obstacle features, as overlapping areas only add visual noise without contributing extra data to CVT. For merging we used an external library called polyclip-ts [53, 54]. An example of the merging process can be seen in Figure 16.

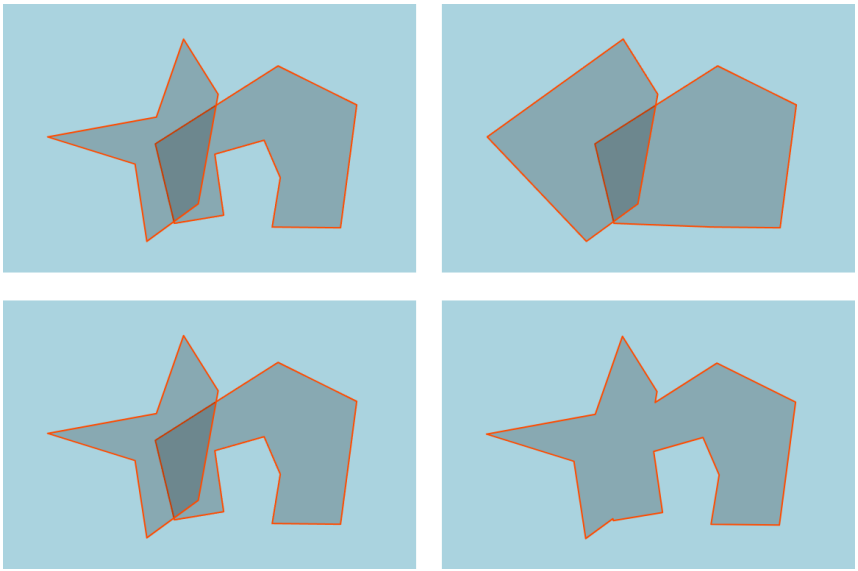

Figure 16. *Two overlapping obstacle features before and after being made convex (top), and being merged together (bottom).*

The controls for making obstacles convex and merging obstacles are found in the obstacles tab. These operations are only done at the user's request, as having concave or overlapping obstacles can often be useful.

5.4.2 Canvas area

For exporting data to CVT, importing data from other sources or using certain operations on map features, the user must first define the canvas area. The canvas area is a rectangular area of the map, and it is shown on the map after it has been set (see Figure 17 for an example). The user can set the canvas area by selecting "Set Canvas" from the canvas menu and then selecting two points on the map, which will be the diagonally opposing canvas corner points. Canvas width and height will be displayed in the canvas menu after the canvas area has been set. The recommended maximum canvas size is 40 by 40 nautical miles, and the user is warned if the selected canvas area is larger.

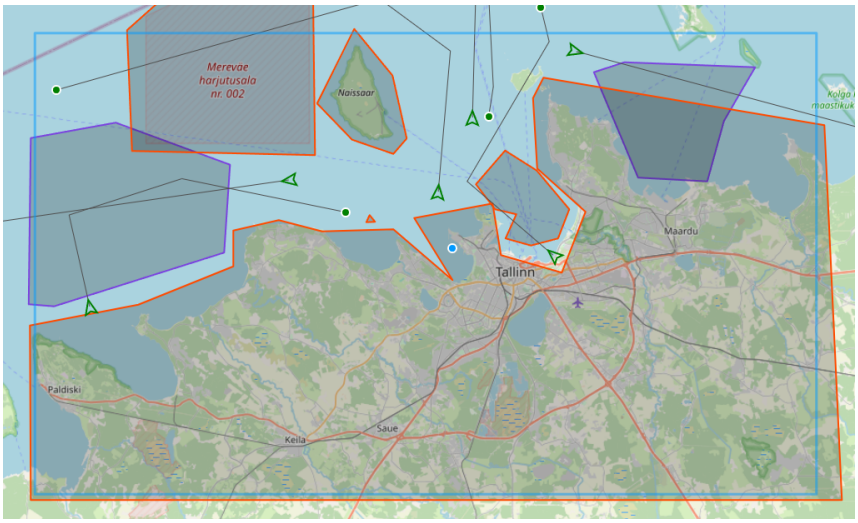

Figure 17. *Canvas with multiple features defined. Canvas area is marked with the blue rectangle. The blue circle in the center marks the canvas coordinate origin point.*

When spatial position data is exported for CVT it is exported relative to the canvas coordinates origin point. By default the canvas origin point is the most southwestern corner of the canvas area. The origin point can be changed by selecting “Custom origin” from the canvas menu, which adds a blue circle that denotes the origin point, as displayed in Figure 17. The origin point’s position can be changed by dragging it around on the map when the canvas tab is selected.

Another function of the canvas area is to filter the features to only keep those inside of the canvas area. We called this operation clipping, as polygonal features that are only partially inside of the canvas area will be clipped to have only the area of the feature that is inside of the canvas area bounds. Features outside of the canvas area will be removed completely. Clipping can be done separately for each feature type. An example of clipping can be seen in Figure 18.

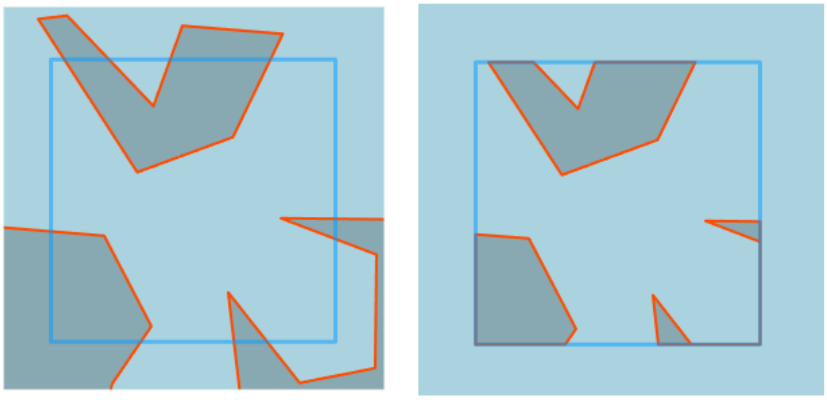

Figure 18. *Obstacle features before (left) and after (right) being clipped to the canvas bounds.*

Using data from external sources is also done with the help of the canvas area. When the canvas area is selected the user can choose to load data from one of the external sources, and features are added only for the data inside the canvas area. Clipping is done for area based features such as coastline and water depth. The restriction to only load data inside of the canvas area exists to prevent loading unnecessary data which would only slow down the application and make it more difficult for the user to manage it, as the total amount of real world data can be very large. The external data will be added as regular features with extra information stored in feature parameters, and work in the same way as features created by the user.

We based our clipping implementation on the Sutherland–Hodgman algorithm [55], optimized to only clip against a rectangular area. We also experimented with using the polyclip-ts library for clipping, but it was too slow for large amounts of data, likely because it is designed to have the ability to clip against polygons of arbitrary shapes. Our clipping solution needs to be fast, because it is used during the external data loading process to extract features that are inside the canvas area from the entire dataset.

We also created a basic measurement tool that can be used to measure distances on the map. Distances are shown in both nautical miles and meters. The measurement tool can also be used as a marker, as the measured line persists on the map until cleared. The map also has the ability to display geographical coordinates as the cursor is moved around on the map.

5.4.3 Conversion of measurement units

The application allows users to change the measurement units that are used for showing values throughout the application. Measurement units are used in places such as most feature parameters and canvas dimensions information. When exporting data for CVT it is also possible to export data in the currently selected measurement units. By default the measurement units used are knots for speed, nautical miles for distance, and degrees for direction. Additional units available for selection are meters per second and kilometers per hour for speed, meters and kilometers for distance, and radians for direction. Direction is represented by a clockwise angle from north.

When the user changes one of the selected measurement units, all shown values will immediately be converted to display the value as the newly selected unit. To avoid possible cumulative conversion errors, all values are stored as default units and converted to selected units for displaying. This means that small rounding errors can occasionally occur once when users enter values in units that are not the default ones - when loading the same

value from file or changing selected unit to something else and back again the value might be different by a decimal point. To prevent the application from showing the users unreasonably long floating point numbers, the converted numbers are rounded to two to four decimal points depending on the unit. This should still provide enough accuracy as the minimum safe distance between ships almost always exceeds multiple meters.

Measurement units can be changed from the settings tab. The settings tab also has options to toggle different map layers' visibility and other application settings. When a feature layer is set to not visible its features will not be shown unless the corresponding tab is selected. The base map layer can also be disabled from the settings tab. Changes made to settings other than layer visibility are saved to the browser's local storage so they persist between sessions.

5.4.4 Saving and loading data

Features created in the application can be saved to a file to continue working with them later. All features along with their parameters and the defined canvas area are written to a single file. The data is saved in JSON format, where features themselves also follow the GeoJSON standard. Default measurement units are always used for saved data values, regardless of which measurement units the user has selected for the application to display.

When loading data from a file it is possible to pick between different feature types and replace or merge features. The user is first prompted to choose a file to load data from, and then a small form is shown, as displayed in Figure 19. Here the user can select the types of features to import and how to import each type. Selecting Merge will add the imported features to the currently existing ones and selecting Replace will remove the currently existing features before importing. The number of features found in the file is shown next to the feature type names, and if features of a certain type do not exist in the file then import selection buttons are not shown for that feature type. If the file does not contain any features or a canvas area, or it is not a valid JSON file, then a warning is shown to the user and the data import form is not shown.

For the purposes of saving and loading data, the canvas area is treated as a regular feature of its own distinct type. Custom origin points for canvas areas are saved and loaded normally, similar to feature parameters. As only one canvas area can exist at a time, it is not possible to select Merge when loading canvas area data from a file.

Once features from the file were successfully loaded and the file contained a canvas area, the map is zoomed and panned so that the canvas area will be visible in the center of the

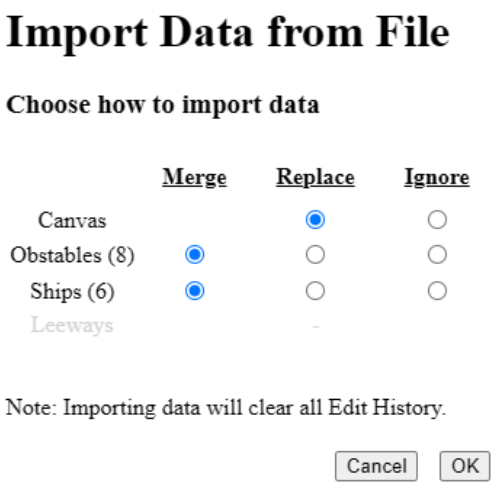

Figure 19. *Import form shown to the user when loading saved data from a file. The leeways line is disabled, as the file does not contain any leeway features.*

map view.

As previously discussed, the CVT GUI application can also be used for creating CVT input data and we want our solution to be compatible with it. After discussing with the creators of CVT GUI, we decided that a good approach would be to have the ability to exchange saved data in an agreed upon format. Since JSON is a frequently used standard, we agreed to use the same JSON and GeoJSON format that is currently used for storing data by our application. Having this functionality would allow users to start creating input data in either application and then switch as needed. To be able to easily store and retrieve data from the CVT GUI database directly, an API should be created. Since the database exists in the CVT GUI application and it also already has a back-end for communicating with the database, we agreed that the best approach would be to extend the CVT GUI back-end to include endpoints for storing and retrieving data in the JSON format that is also used by our application.

5.4.5 Exporting data for CVT

After the desired features have been specified, the data can be exported in the format that CVT expects. Exporting is only allowed if the canvas area has been set. Only the features within the canvas area are exported. In addition to feature data, the file generated by exporting also contains information about the canvas area and selected measurement units.

Data validity checks are performed before export file is generated to reduce the chances of

unsolvable scenarios being sent to CVT. Validity checks include checking that ships do not overlap with land obstacles, that multiple ships do not have the same identifier and that the canvas area is not too large. The user is warned when validity checks do not pass, but it is still possible to export data if the user decides to.

Exporting data creates a Prolog file which contains the data as Prolog facts in the format required by CVT. Detailed description of the export format is given in Appendix 2. Since CVT assumes that each canvas has a unique name, a random UUID is generated for every data export. This also helps to differentiate it in the CVT GUI database. When exporting the user can choose to use currently selected measurement units or default units for the exported data. When selected units are used, the information about the measurement units is included in the exported file, otherwise it is omitted.

To better interface with CVT Output Visualizer the generated file also contains the data in JSON format, wrapped in Prolog facts. This was added because CVT output does not contain all of its input data, and the Visualizer was built so that it understands the JSON format of our application. Since CVT output is also in JSON format, it can then pass the input along with the output to the Visualizer in a relatively simple way. This does create extra redundancy in the output file and causes the file size to be larger, so with future developments to the Output Visualizer or CVT its removal should be considered.

Positions of features are exported as distances from the canvas area coordinate origin point. Because CVT needs to calculate distances between objects during its simulation and verification process, we decided to add the conversion from geographical coordinates to distances in the data export step. Distances are given as two components: the distance of only the longitudinal difference (at the latitude of the origin point) and the distance of only the latitudinal difference of two points. Distances between geographical coordinates are calculated using the haversine formula to account for the curvature of the Earth.

6. Validation

The application created as a part of our solution is meant to enhance the existing CVT maritime safety analysis system. As such our application is only useful when used together with other parts of the system, such as CVT itself.

Real use cases of our application also depend on the real use cases of the entire system. One potential use case for the system is for researchers to discover shortcomings in COLREG when it comes to regulating MASS. The CVT system can help with finding quantitative restrictions to the COLREG rules that are needed for creating navigational algorithms and systems. Another example use case is incident analysis, where it would be possible to simulate the situation that happened and verify if the situation was resolved according to COLREG.

When only looking at our application, the main use case is generating input data for CVT. This use case is well covered with the different editing tools and data integrations that we implemented. The functionality needed for this use case was presented as functional requirements and the application covers all of them.

Our solution improves the current method of creating COLREG input data by providing multiple tools to make it more productive. By showing the data directly on a map the users get immediate visual feedback on the shape of the data in the surrounding context. Allowing the users then to directly edit the same data on the map makes editing intuitive. By using an underlying real world map as a base, the users can easily see where the created data would exist in the real world context. Our solution also links CVT input data with real coordinates, whereas before the coordinates were arbitrary positions in an imagined simulation space.

Our application's integration with different data sources enables creating more realistic simulation environments with little effort, something that would have required significant amounts of manual work before. The reusability of created CVT inputs also means that it is possible to pick different parts to use as a base for creating more simulation input data, or preparing a model situation environment once to use it later with multiple different ship configurations easily. The ability to change the measurement units that the application uses, allows users to work in a unit system that is most comfortable for them. All these are the main ways our solution improves the current workflow of generating CVT input data.

To further validate our solution we also presented it to expert users to get feedback, and implemented tests to ensure the correctness of functionality.

6.1 User feedback

To get feedback on our solution we had discussions with a couple of experts on maritime topics, including practitioning helmsman and the Chief Safety Officer of *Eesti Riigilaevastik*, a company run by the Estonian Ministry of Climate that maintains a fleet of vessels used for various utility purposes [56]. We presented the created application's main functionalities from the user perspective to find parts that are unclear and features that are missing. Throughout the development of our application we also had discussions with the main creator of CVT, who is also the supervisor of this thesis, to understand the current shortcomings with CVT input data creation and find the best ways to integrate our solution with the existing CVT ecosystem.

The feedback from a round of discussion earlier during the development phase was already taken into consideration in the current version of the application. The ability to control the ship paths by setting waypoints was one of the functionalities that was added as suggested by the experts.

The experts liked the application's ability to automatically load water depth data. This was also seen as a potential way of configuring how far away from the shoreline the vessels should keep during navigation. Having water depth data makes shoreline data somewhat redundant, as vessels would always have to account for water depth and cannot simply navigate according to the shoreline as areas near it are usually too shallow for sailing. The ability to get depth data for any depth value specified by the user was seen as something that would be beneficial to have.

The importance of sea marks for navigation was highlighted by the experts. Sea marks are often used to indicate the direction in which the vessels should pass them. In narrower areas with more dense maritime traffic, sea marks are used to create traffic lanes. Not following the rules set by sea marks can lead to serious threats to safety and greatly increase the risk of collisions. As mentioned earlier, CVT does not have support for taking the different meanings of sea marks into account at the moment.

Another important factor for navigation is the current water level. In addition to tides, water level also depends on wind speed and direction. When water level is very low, previously safe areas might become dangerous, but high water levels can also pose unique dangers. This is something that should be considered for future versions of CVT and our

application.

Overall the feedback was positive and highlighted the greater need for automatic processes and data integrations over manual data editing tools.

6.2 Application testing

The created application has been verified to work in Windows using Chromium based web browsers (e.g. Google Chrome, Brave). We also confirmed that the main functionality of the application works correctly in Windows and Linux using both Chromium and Firefox web browsers, but did not perform thorough testing. We observed minor visual layout issues in Linux when using Firefox to view our application.

We have been checking that the features implemented in our application work as expected throughout the development process. There are a couple of known issues, which we hope to resolve soon. To ensure the complex polygon operations of map features work as intended, we created a set of unit tests to verify their correctness in different situations.

7. Future work

The more data we have available to us, the more realistic the situations are that we can simulate and verify with CVT. Currently the application we created has integrations with only a limited amount of external data sources. It would be useful to have even more integrations with external data sources in the future. The usability of the application could also be increased by adding extra functionality.

Here we list some suggestions for how to develop our application further, based on user feedback and ideas gathered during development:

- Add integrations with weather and water level data. This data changes often, and it is possible that users do not always want real-time data, but data about extreme conditions instead.
- Add the ability to load real-time ship data with the help of AIS. Ship specifications could then be loaded from an online ship register.
- Load data from electronic navigational charts instead. The benefit of using navigational charts would be that they already contain all the important information required for navigation. The availability of charts and the complexity of their data format may cause problems.
- Load the data about the different meanings of sea marks and pass it to CVT. This would only be useful if CVT is also updated to make use of them.
- Improve the feature editing tools by adding the option to filter features by type for obstacle features, and potentially make them look different on the map. Currently the different kinds of obstacles all look the same which can cause confusion.
- Combine multiple applications in the current CVT ecosystem together. Currently our application, CVT GUI and CVT visualizer are all separate applications. It would be better for users if everything could be done from a single application.
- Use machine learning to generate synthetic data, or just randomly generate data. Existing data could be used as a basis. This may lead to invalid data, so more validity checks would be needed.

8. Summary

The main goal of our work was improving the existing CVT input data creation process. The current way of entering data manually through text is too cumbersome for entering large amounts of data. No easy way to visualize data while editing it also poses an issue, as the majority of the data is spatial. Another problem is that it is currently difficult to make use of existing data, as it has to be formatted to be accepted by CVT, and then further edited to add experiment specific details.

To solve the main problem we created an application that simplifies and increases the productivity of creating CVT input data. We first analysed other applications that work with geospatial data to determine what features our application should have and how it should look like. Based on the analysis and the expected shape of CVT input data, we then compiled a list of requirements our application should have in order to solve the main problem.

We decided that making use of existing data gathered from the real world would greatly help with generating more realistic CVT input data, which would then also produce more realistic simulation results. Since we did not have any data already available, we analysed potential external data sources where we could gather data from. We found multiple data sources and later integrated them with the created application, so that data could be loaded into the application automatically and be edited as needed before submitting to CVT.

The application we created works as a user interface to CVT input data creation. It allows users to edit data on a map directly, making use of the spatial nature of the data. After discussions with the creator of CVT we decided to add new parameters to data that would further increase the quality of the input data and make it more realistic. We also considered the other applications that currently exist in the broader CVT ecosystem, and made sure that our solution is compatible with them. To validate the results we had discussions and reviewed the created application with experts in maritime safety and navigation.

Our solution should make working with CVT and the maritime safety analysis system more approachable for new users and make generating input data a more productive and faster process. Additionally, with the external data integrations it is significantly easier to create more realistic situations for CVT, which can help it with finding more problematic situations with COLREG and MASS.

References

[1] ALLIANZ GLOBAL CORPORATE & SPECIALTY. *Safety and shipping review 2018*. [Accessed: 01-03-2025]. 2018. URL: https://commercial.allianz.com/content/dam/onemarketing/commercial/commercial/reports/AGCS-Safety-Shipping-Review-2018.pdf.

[2] Xiang-Yu Zhou et al. "A study of the application barriers to the use of autonomous ships posed by the good seamanship requirement of COLREGs". In: *The Journal of Navigation* 73.3 (2020), pp. 710–725.

[3] International Maritime Organization. *Autonomous shipping*. [Accessed: 15-05-2025]. URL: https://www.imo.org/en/MediaCentre/HotTopics/Pages/Autonomous-shipping.aspx.

[4] Roly McKie. *Maritime Autonomous Surface Ships (MASS) and SAR*. [Accessed: 15-05-2025]. URL: https://www.international-maritime-rescue.org/News/maritime-autonomous-surface-ships-mass-and-sar.

[5] Georgios Daniil and Michael Boviatsis. "Evaluation of Environmental Impact Assessment Factors in Maritime Industry". In: *Journal of Environmental Science and Engineering B* 11 (2022), pp. 18–23.

[6] SAFETY4SEA Editorial Team. *MASS Roadmap: Important dates to keep in mind*. [Accessed: 15-05-2025]. 2024. URL: https://safety4sea.com/mass-roadmap-important-dates-to-keep-in-mind/.

[7] International Maritime Organization. *Convention on the International Regulations for Preventing Collisions at Sea, 1972 (COLREGs)*. [Accessed: 10-05-2025]. URL: https://www.imo.org/en/About/Conventions/Pages/COLREG.aspx.

[8] International Maritime Organization. *Convention on the International Regulations for Preventing Collisions at Sea*. IMO Publication, 2003. ISBN: 9789280141672.

[9] International Maritime Organization. *Member States*. [Accessed: 10-05-2025]. URL: https://www.imo.org/en/OurWork/ERO/Pages/MemberStates.aspx.

[10] Hiba Ben Lahib, Mohamed Taha, and Jüri Vain. "Autonomous Vessels Collision Verification: Geometric optimization". [Forthcoming]. 2025.

[11] Hiba Ben Lahbib et al. “Two-phase Path Planning for Fuel-Efficient Safe Navigation”. [Forthcoming]. 2025.

[12] Open Simulation Platform. *The Open Simulation Platform*. [Accessed: 10-05-2025]. URL: https://opensimulationplatform.com.

[13] Hiba Ben Lahib. “Verification of Navigation Rules for Maritime Autonomous Systems”. MA thesis. University of Tunis El Manar, 2024.

[14] Marinetraffic. *Marinetraffic Home Page*. [Accessed: 14-05-2025]. URL: https://www.marinetraffic.com/en/ais/home.

[15] Kpler. *About Us: From Vision to Reality. A Decade of Innovation*. [Accessed: 14-05-2025]. URL: https://www.kpler.com/company/about-us.

[16] Kpler. *MarineTraffic Data Services*. [Accessed: 14-05-2025]. URL: https://www.kpler.com/product/maritime/data-services.

[17] Transpordiamet. *Nutimeri*. [Accessed: 14-05-2025]. URL: https://gis.transpordiamet.ee/nutimeri/.

[18] Esri. *History of GIS*. [Accessed: 14-05-2025]. URL: https://www.esri.com/en-us/what-is-gis/history-of-gis.

[19] Esri. *ArcGIS*. [Accessed: 14-05-2025]. URL: https://www.esri.com/en-us/arcgis/geospatial-platform/overview.

[20] QGIS. *Spatial without Compromise*. [Accessed: 14-05-2025]. URL: https://qgis.org/.

[21] International Maritime Organization. *Electronic Nautical Charts (ENC) and Electronic Chart Display and Information Systems (ECDIS)*. [Accessed: 14-05-2025]. URL: https://www.imo.org/en/OurWork/Safety/Pages/ElectronicCharts.aspx.

[22] UK Hydrographic Office. *S-57 to S-101: Explaining the IHO standards for ECDIS*. [Accessed: 14-05-2025]. 2023. URL: https://www.admiralty.co.uk/news/s-57-s-101-explaining-iho-standards-ecdis.

[23] Inland ENC Harmonization Group. *About IEHG*. [Accessed: 14-05-2025]. URL: https://ienc.openecdis.org/about-iehg.

[24] Olaf Hannemann. *OpenSeaMap - Online map*. [Accessed: 14-05-2025]. 2012. URL: https://map.openseamap.org/.

[25] Navico Group. *Accurate worldwide charts backed by C-MAP expertise*. [Accessed: 14-05-2025]. URL: https://www.c-map.com/.

[26] OpenCPN. *About OpenCPN*. [Accessed: 14-05-2025]. URL: https://opencpn.org/OpenCPN/info/about.html.

[27] OpenStreetMap Foundation. *OpenStreetMap provides map data for thousands of websites, mobile apps, and hardware devices*. [Accessed: 14-05-2025]. URL: https://www.openstreetmap.org/about.

[28] Natural Earth. *Natural Earth*. [Accessed: 14-05-2025]. URL: https://www.naturalearthdata.com/.

[29] Pål Wessel and Walter HF Smith. "A global, self-consistent, hierarchical, high-resolution shoreline database". In: *Journal of Geophysical Research: Solid Earth* 101.B4 (1996), pp. 8741–8743.

[30] Maa- ja Ruumiamet. *Eesti topograafia andmekogu*. [Accessed: 14-05-2025]. URL: https://geoportaal.maaamet.ee/est/ruumiandmed/eesti-topograafia-andmekogu-p79.html.

[31] Transpordiamet. *Meremõõdistamine*. [Accessed: 14-05-2025]. URL: https://transpordiamet.ee/meremoodistamine.

[32] GEBCO. *STRATEGY 2024-2030*. [Accessed: 14-05-2025]. 2024. URL: https://www.gebco.net/sites/default/files/documents/gebco_strategy2024_2030_dec_2024.pdf.

[33] Larry Mayer et al. "The Nippon Foundation—GEBCO seabed 2030 project: The quest to see the world's oceans completely mapped by 2030". In: *Geosciences* 8.2 (2018), p. 63.

[34] GEBCO. *The GEBCO_2024 Grid*. [Accessed: 14-05-2025]. URL: https://www.gebco.net/data-products-gridded-bathymetry-data/gebco2024-grid.

[35] Brook Tozer et al. "Global bathymetry and topography at 15 arc sec: SRTM15+". In: *Earth and space science* 6.10 (2019), pp. 1847–1864.

[36] Transpordiamet. *Navigatsioonimärkide andmekogu*. [Accessed: 14-05-2025]. URL: https://nma.vta.ee/.

[37] OpenStreetMap Wiki. *Slippy map*. [Accessed: 14-05-2025]. URL: https://wiki.openstreetmap.org/wiki/Slippy_map.

[38] Volodymyr Agafonkin. *Leaflet - an open-source JavaScript library for mobile-friendly interactive maps*. [Accessed: 14-05-2025]. URL: https://leafletjs.com/.

[39] OpenLayers. *OpenLayers Home Page*. [Accessed: 14-05-2025]. URL: https://openlayers.org/.

[40] Mapbox. *Mapbox Web Maps*. [Accessed: 14-05-2025]. URL: https://www.mapbox.com/mapbox-gljs.

[41] Google for Developers. *Maps JavaScript API*. [Accessed: 14-05-2025]. URL: https://developers.google.com/maps/documentation/javascript/overview.

[42] MediaWiki. *Extension:Kartographer*. [Accessed: 14-05-2025]. URL: https://www.mediawiki.org/wiki/Extension:Kartographer.

[43] Volodymyr Agafonkin. *Leaflet Plugins database*. [Accessed: 14-05-2025]. URL: https://leafletjs.com/plugins.html#user-interface.

[44] ADS-B Exchange. *ADS-B Exchange Home Page*. [Accessed: 14-05-2025]. URL: https://www.adsbexchange.com/.

[45] Evan You. *The Progressive JavaScript Framework*. [Accessed: 14-05-2025]. URL: https://vuejs.org/.

[46] Devographics. *State of JavaScript 2024: Front-end Frameworks*. [Accessed: 14-05-2025]. 2024. URL: https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/.

[47] Vue. *Composition API FAQ*. [Accessed: 14-05-2025]. URL: https://vuejs.org/guide/extras/composition-api-faq.html.

[48] Justus Bogner and Manuel Merkel. "To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github". In: *Proceedings of the 19th International Conference on Mining Software Repositories*. 2022, pp. 658–669.

[49] Google. *The Go Programming Language*. [Accessed: 14-05-2025]. URL: https://go.dev/.

[50] Google. *Go for Cloud & Network Services*. [Accessed: 14-05-2025]. 2019. URL: https://go.dev/solutions/cloud.

[51] OpenStreetMap Wiki. *Overpass API*. [Accessed: 14-05-2025]. URL: https://wiki.openstreetmap.org/wiki/Overpass_API.

[52] Alex M Andrew. "Another efficient algorithm for convex hulls in two dimensions". In: *Information processing letters* 9.5 (1979), pp. 216–219.

[53] Luiz F. M. Barboza et al. *polyclip-ts*. [Accessed: 17-05-2025]. URL: https://www.npmjs.com/package/polyclip-ts.

[54] Francisco Martinez et al. "A simple algorithm for Boolean operations on polygons". In: *Advances in Engineering Software* 64 (2013), pp. 11–19.

[55] Ivan E Sutherland and Gary W Hodgman. "Reentrant polygon clipping". In: *Communications of the ACM* 17.1 (1974), pp. 32–42.

[56] Riigilaevastik. *Riigilaevastik Home Page*. [Accessed: 17-05-2025]. URL: https://www.riigilaevastik.ee/.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Renee Kroon

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Application for Generating Input Data for COLREG Verifier", supervised by Jüri Vain
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

19.05.2025

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - CVT input data format description

```
canvas_json(PartID, Text).
   * PartID - number of the JSON fragment (number)
   * Text - JSON as text (string)

canvas(CanvasID, (Xmin, Xmax), (Ymin, Ymax),
   (Xorig, Yorig)).
   * CanvasID - unique id (string / number)
   * (Xmin, Xmax), (Ymin, Ymax) - canvas area coordinates
      (max 40 miles)
   * (Xorig, Yorig) - canvas origin point (coordinates)

canvas_units(_,((DistanceUnit,DistanceUnit),
   (DistanceUnit,DistanceUnit)),(coordinate,coordinate)).
ship0_units(_,_,_,(DistanceUnit,DistanceUnit),
   DirectionUnit,SpeedUnit,_).
route0_units(_,_,_,[(_,((DistanceUnit,DistanceUnit),
   (DistanceUnit,DistanceUnit)),_,_,_,_,_)]).
obstacle_units(_,[(DistanceUnit,DistanceUnit)],meters).
leeway_units(_,[(DistanceUnit,DistanceUnit)],
   (DirectionUnit,DirectionUnit),(SpeedUnit,SpeedUnit)).
   * DistanceUnit - one of: miles | meters | kilometers
      (default: miles)
   * DirectionUnit - one of: degrees | radians
      (default: degrees)
   * SpeedUnit - one of: knots | mps | kmph
      (default: knots)

ship_type_def(TypeName, (Length, Beam, Draft), Trusters,
   FuelConsumption, Comment).
   * TypeName - unique name
   * (Length, Beam, Draft) - ship dimensions
   * Thrusters - text
   * FuelConsumption - Function of speed gradient
      (function name)
```

```
   * Comment - text

ship0(ShipID, TypeName, Timestamp, (X, Y), COG, SOG,
   UnderControl).
   * ShipID - unique id (string / number)
   * TypeName - refers to ship_type_def
   * Timestamp - when ship enters simulation
   * (X, Y) - starting position relative to canvas origin
      (meters)
   * COG - starting course over ground (0-359.99 degrees)
   * SOG - starting speed over ground (0-40 knots)
   * UnderControl - laeva juhitavuse määr (0-1)

route0((ShipID, _), _, _, [(required,((X1,Y1),(X2,Y2))
   ,_,_,_,_,_)]).
   * ShipID - ship the route belongs to
   * required - red for required, blue for optional
      (red | blue)
   * (X1,Y1) - starting point for route segment,
      relative to canvas origin
   * (X2,Y2) - ending point for route segment,
      relative to canvas origin

obstacle(ObstacleID, [RegionPolygon], MinimalDepth).
   * ObstacleID - unique id (string / number)
   * [RegionPolygon] - obstacle area relative to canvas
      origin (counter-clockwise, meters)
   * MinimalDepth - minimum water depth in area
      (0 for ground)

leeway(LeewayID, [RegionPolygon], (DirectionMin,
   DirectionMax), (SpeedMin, SpeedMax)).
   * LeewayID - unique id (string / number)
   * [RegionPolygon] - leeway area relative to canvas
      origin (counter-clockwise, meters)
   * (Direction) - leeway COG (min - max)
   * (Speed) - leeway SOG (min - max)
```