# Simulation Framework and Software Environment for Evaluating Automatic Ship Collision Avoidance Algorithms*

Trym Tengesdal[1] and Tor A. Johansen[1].

***Abstract*— Research on maritime automatic collision avoidance algorithms is nowadays abundant, and becoming more mature. Moreover, methods for the evaluation of such algorithms are emerging. Therefore, we propose a software environment for simulation and an evaluation framework for maritime collision avoidance, used for testing and evaluating control algorithms. The system comes with a set of standard scenarios that can be augmented with random variations and a database of scenarios based on historical Automatic Identification System (AIS) data that can be used in the simulation. In this framework, the user can set up scenarios with pre-programmed target ship routes, or with target ships that exhibit proactive and reactive collision avoidance maneuvers. Moreover, the user can specify uncertainty in the tracking of target ships and situational awareness, and include the presence of grounding hazards from Electronic Navigational Charts (ENCs). The evaluation tool computes penalties and scores on algorithm performance according to the maritime traffic rules (COLREGS), safety, and other performance-related metrics. The framework can be used for verification and assurance, and further enables automatic adjustment of collision avoidance planners online or offline.**

## I. Introduction

In numerous studies on automatic collision avoidance (COLAV) for ships, a rich set of algorithms have been developed. These range from rather reactive methods such as Artificial Potential Field (APFs) methods [28], Velocity Obstacle (VO) methods [11, 12] and interval programming [3], to deliberate higher level planning algorithms such as Rapidly-exploring Random Trees (RRTs) [7], A*-search [17] and Scenario-based Model Predictive Control [20]. Hybrid combinations of proposed algorithms have also been studied and developed [13, 6], as well as algorithms that consider both anti-collision and anti-grounding, [21, 5]. Many of these studies show partial compliance with the International Regulations for Preventing Collisions at Sea (COLREGS) [1] in simulation or experiments.

Despite many of these having been tested extensively in experiments, there is limited focus on systematic and extensive testing and validation also in a simulation environment. This is paramount for regulatory bodies and the public to gain trust in autonomous navigation systems and will make it easier to deploy such systems as decision support in existing vessels and as part of the autonomy suite on emerging autonomous ship technology. Most of the cited studies have created their own simulators, selected some situations for testing, and evaluated the performance qualitatively or on non-standard metrics. This type of simulation and evaluation suffers from several limitations: First, it makes the comparison of algorithms hard and sometimes impossible due to the discrepancy between scenarios, initial conditions, and metrics used. Second, studies reporting scarce amounts of evidence might easily miss negative evidence regarding the validity of the proposed COLAV algorithm. Third, the completeness and robustness of algorithms are not being gauged properly.

There are a few studies that have systematically considered simulation-based testing and/or evaluation of collision avoidance algorithms. Woerner et al. [26] developed a framework for evaluating control performance with respect to the COLREGS, using the geometric configurations of the vessels at different stages to score safety, individual traffic rules, and other important aspects. This was further extended in [9] to provide more details on the individual penalties and scoring functions used in the evaluation, including the impact of grounding hazards. It was also tested on simulated and historic Automatic Identification System (AIS) scenarios. A fuzzy-logic-based method for COLREGS evaluation was introduced in [2], covering rules 2 to 19. These studies were mostly focused on the evaluation part of ship collision avoidance, and do not consider the full pipeline with how to generate scenario definitions, simulate the vessel behavior for each scenario, and evaluate vessel performance.

The authors in [18] proposed an evaluation method that considers scores related to the ship mission, in addition to safety and COLREGS similar to [26]. Adaptive scenario generation and unsupervised clustering are here used to identify low performance scenarios. For the simulation part, generated vessels all follow constant velocity, which will not always properly reflect real behaviours in hazardous encounters. An automatic collision avoidance testing suite was developed using the CyberSea simulator of Det Norske Veritas (DNV) [14], with a focus on the COLREGS evaluation proposed by Woerner [25]. The work considers the generation of scenarios with dynamic obstacles adhering to constant speed throughout, on which two COLAV algorithms were compared. An Autonomous Navigation System (ANS) test environment was introduced in [16], where the Robotic Operating System (ROS) is used for simulation, with usage of Woerner‘s work [26] for evaluation.

Vagale [23] proposed an evaluation and simulator platform for testing algorithms with consideration of collision risk.

*This work was supported by the Research Council of Norway through the Autoship Centre for Research-based Innovation, project number 309230, and by Kongsberg Maritime through the University Technology Center on Ship Performance and Cyber-Physical Systems.

[1]The authors are with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway (e-mail: {trym.tengesdal, tor.arne.johansen}@ntnu.no)

Fuzzy inference based on an expert rule base is used to take static, dynamic and historic risk factors into account for each time step along a vessel path. A root mean square function is applied afterward in order to obtain a total extended collision risk assessment (ECRA) score. However, the evaluation does not consider COLREGS. Furthermore, the Autoferry Gemini simulator [24] is used, which is tailor-made for testing computer vision and sensor fusion systems, not COLAV planning algorithms.

The authors in [22] utilized Signal Temporal Logic (STL) as a means to evaluate robustness scores for COLAV algorithms with respect to safety and the COLREGS, using a Gaussian Process (GP) to estimate how the scores depend on scenario parameters. The GP was further used to guide the selection of scenarios to test the COLAV algorithm, based on its confidence level on having covered the simulation parameter space, i.e. the set of considered scenarios. Static obstacles and dynamic obstacle uncertainty are not considered in the simulation and evaluation. How the approach scales with increasing complexity in the scenarios and parameter space is not discussed.

In [27], an AIS-based scenario generation method was proposed. AIS data was analyzed and used to estimate Probability Density Functions (PDFs) describing the parameters of an encounter, such as distances between vessels, their speeds and bearings. The PDFs were then used to generate a large number of scenarios for testing COLAV algorithms. The goal was to increase the test coverage for such systems, over that which is possible with only expert-designed and real AIS data scenarios. Again, generated vessels all follow constant velocity, which does not always reflect true vessel behaviour in hazardous encounters. Furthermore, grounding hazards are not considered.

In this article, we propose a new framework for visualizing and evaluating the performance of COLAV planning algorithms. A mixture of historic AIS scenarios and randomly generated variations of standard scenarios can be used, with usage of Electronic Navigational Chart (ENC) data. The framework can include grounding hazards, uncertain situational awareness, and intelligent or non-intelligent behavior for any of the ships in the simulation. A scenario generator takes in configuration files containing relevant input data for creating or loading a scenario. The scenario can be partially or fully specified by historical AIS data, or alternatively with hard-coded and randomly generated components. The simulator in the framework then goes through all scenario definitions created by the scenario generator and generates simulation data. The evaluation tool in [9] is used to measure the performance of vessels involved in a scenario with respect to the COLREGS, safety, and other metrics. The reader is referred to [1, 26] for more details on the COLREGS protocol.

The framework is flexible in that the vessels being simulated in a scenario can be configured with different combinations of subsystems such as sensors, ship models, target trackers, controllers, etc. COLAV planning algorithms can be tested by providing them externally as input to the simulator at run-time, or internally through configuration files by developing a wrapper around the algorithm inside the framework. Thus, the framework is agnostic to specific subsystem components, as long as its interface is adhered to. A key feature is that ship maneuvering can be considered during simulation, as opposed to the restrictive constant velocity assumption used by most studies. Created scenarios can be saved, and the results from simulation and evaluation can be analyzed, visualized, and stored as per the users' needs. Furthermore, the framework lays the groundwork for developing new search algorithms for finding interesting scenarios to test a COLAV algorithm, on which its performance has not been tested properly yet. It also provides an environment for developing learning-based or data-driven COLAV algorithms that adjust their behaviour based on the evaluation of their performance over scenarios.

The rest of the article is structured as follows. An overview of the simulation and evaluation framework is given in Section II. Here, the components of therein are detailed sequentially. Section III then showcase example usage of the full scenario generation, simulation, and evaluation pipeline. Lastly, Section IV wraps up the main content of the article.

## II. COLAV Simulation and Evaluation Framework

The proposed framework is illustrated in Fig 1, where the flow of information between the scenario generator, simulator, and evaluator is shown. The framework allows for feedback in the simulation and evaluation process, where performance assessments from the evaluation can be used to guide the creation of scenarios depending on where the own-ship COLAV system performs the worst, similar to work in [22] and discussed in [16]. The framework is implemented in Python 3.10, with each component in the framework being a class object with its own variables and methods. Abstract classes are used to enforce interfaces between the objects. An executable is used locally to run the generation, simulation and evaluation pipeline.

All ships, including the own-ship, which are involved in the simulation can be configured with specific subsystems. This allows for the consideration of various ship types with different COLAV algorithms, target trackers, etc. The Evaluator module can evaluate the performance of all the ships involved in a scenario, although the own-ship running a specific COLAV system is most likely to be considered. We note that the simulator is not required to be of high fidelity such as those e.g. considered by [16] or in the Open Simulation Platform (OSP) [15], as the goal of this framework is to provide an easy to use simulation and evaluation framework for validating COLAV planning algorithms on a large number of scenarios. In the following subsections, the components of the framework are detailed.

### A. The Scenario Generator

The scenario generator creates a set of configured scenarios for the simulator to run through, based on input data and

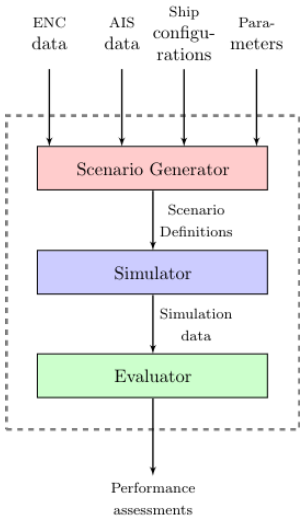


Fig. 1: Simulation and evaluation framework.

generation parameters. The main interface function for the `ScenarioGenerator` module is the following function: `generate(configFile, **kwargs) -> Tuple`: The function generates a maritime scenario definition based on the input configuration file `configFile`, which corresponds to the inputs in Fig. 1. It can also be called with an already created scenario configuration object, also with options for changing generation parameters at run-time, hence the **kwargs argument. Any number of $n_{episodes}$ can be generated for a given scenario, thus enabling Monte Carlo simulation. A tuple consisting of a list of fully configured `Ship` objects for each episode, an ENC object for the map information, and other scenario metadata is returned from the function.

The generator is flexible in that both historical AIS data and randomly generated data can be used for specifying initial ship poses and trajectories in any given scenario. The behavior of a ship can be fully specified by a historical AIS trajectory, or through simulation with a configured ship object. If AIS trajectories are used, they are interpolated to the timespan and timescale used in the simulation. The parameter input shown in Fig. 1 among others specifies the scenario simulation timespan and time step. Imazu [10] developed a set of standard scenarios for testing ship collision avoidance maneuvering, which are included as a standard benchmarking scenario suite in the framework, and can also be used.

The ship object can be initialized to a position, course and speed over ground (CSOG) state $\boldsymbol{x}_{csog} = [x, y, U, \chi]^T$ with planar coordinates $x$ and $y$, speed $U$ and course $\chi$ from AIS data or be randomly generated. The list of input ship configurations is used to specify the initial state and plan, subsystems, and identifiers such as its Maritime Mobile Service Identity (MMSI). Because the ship object is programmed with interfaces for each subsystem, it makes it easy to test different combinations of these in simulation. More importantly, it allows for testing third-party or internally developed COLAV planning algorithms, as long as the interface is adhered to. The COLAV algorithm can be provided through the scenario configuration file, or as an external input to the simulator at run-time. Any of the target ships can be configured as intelligent or cooperative, i.e. with an onboard COLAV planning algorithm, or non-intelligent/non-cooperative in the sense that it will blindly follow a pre-planned trajectory. Note that the ships do not need to be fully configured in the input configuration file, as the scenario generator will fill in missing information through random generation. Fig. 2 shows an example scenario configuration file, which was used to generate the first example case in Section III.

```yaml
name: "example1"
save_scenario: True # If true the scenario is saved after all ships are configured and initial plans/trajectories are generated.
t_start: 0.0 # Start time of scenario.
t_end: 500.0 # End time of scenario.
dt_sim: 0.1 # Time step used in the simulation.
scenario_type: HO # Whether it is a single-ship (SS), head-on (HO), crossing (CR), overtaking (OT), multi-ship (MS) or random type (R) of scenario.
utm_zone: 33 # Universal Transverse Mercator zone used for specifying the considered coordinate frame.
map_origin: [40000.0, 6960450.0] # East-north coordinate origin for the map.
map_size: [20000.0, 20000.0] # East-north size components of the map.
map_data_files: [More_og_Romsdal.gdb"] # ENC data file to consider.
new_load_of_map_data: True # Whether or not to process the .gdb data files into shapefiles initially.
n_random_do_ships: 1 # Number of dynamic obstacle ships to generate randomly.
n_episodes: 1 # Number of episodes to generate for the scenario.
ship_list: # List of configurations for the ships one wants to set specific details for. This includes both the own-ship and dynamic obstacle ships.
  - id: 0 # Ship identifier. Zero always correspond to the own-ship ID.
    mmsi: 100 # Maritime Mobile Service Identity
    random_generated: True # Whether or not the initial ship plan is partially/fully randomly generated.
    csog_state: [6966500.0, 48500.0, 7.0, 0.0] # Initial course and speed over ground state. Specifying it prevents it from being randomly generated.
    colav: None # Can here specify any COLAV planning algorithm. Specifying it as None makes it a non-intelligent vessel that will blindly follow its pre-defined or randomly generated plan.
    tracker:
        kf: # Linear Kalman Filter
            P_0: [49.1, 49.1, 0.11, 0.11]
            q: 0.15
    sensors:
      - ais:
            max_range: 3000.0
            ais_class: "A" # AIS transponder type
            R: [26.3, 26.3, 0.1, 0.1] # Tuneable sensor noise covariance
            R_true: [26.3, 26.3, 0.1, 0.1] # True sensor noise covariance
    guidance:
        los:
            pass_angle_threshold: 80.0 # Threshold angle for having passed a waypoint segment, if one is far away.
            R_a: 40.0 # Acceptance radius
            K_p: 0.015 # Proportional gain
            K_i: 0.0 # Integral gain
            e_int_max: 200.0 # Maximum cross-track error considered in the integral action
    model:
        csog: # Kinematic ship model considering course and speed over ground
            draft: 3.0
            length: 10.0
            width: 3.0
            T_chi: 3.0 # Time constant for the course dynamics
            T_U: 5.0 # Time constant for the speed dynamics
            r_max: 4.0 # Maximum rate of turn
            U_min: 0.0 # Minimum speed
            U_max: 15.0 # Maximum speed
```

Fig. 2: The scenario configuration file used to generate the example case shown in Fig 7.

The Sea Charts Application Programming Interface (API) Python package [4] is used for visualizing and utilizing Electronic Navigational Charts (ENCs) data. The map data shown in this article can be obtained from the Norwegian Mapping Authority (NMA): `https://www.geonorge.no/`. A generated classic head-on COLREGS scenario is shown in Fig. 7 for the own-ship (OS) and a nearby dynamic obstacle

(DO). As per Line 6 in Fig. 2, different COLREGS scenarios can be randomly generated by specifying the scenario type, which will influence the random generation of CSOG states, waypoints, and speed plans. For scenarios with more than two ships, the scenario type is configured to a *multi-ship* setting, where the specific COLREGS situation for ship pairs is randomized. The `ScenarioGenerator` has distribution parameters that can be specified by the user, which will influence the variation ranges in initial ship CSOG states and plans. The initial state distributions are conditioned on the type of scenario being generated. In e.g. a head-on scenario, the own-ship can have an initial CSOG state generated from a uniform distribution

$$\boldsymbol{x}_{os} \sim \mathcal{U}\left(\begin{bmatrix} x_{min} \\ y_{min} \\ U_{os,min} \\ 0.0^\circ \end{bmatrix}, \begin{bmatrix} x_{max} \\ y_{max} \\ U_{os,max} \\ 360.0^\circ \end{bmatrix}\right) \tag{1}$$

where $x_{min}, y_{min}$ and $x_{max}, y_{max}$ are found from the ENC data bounding box, and the minimum and maximum speeds $U_{os,min}$ and $U_{os,max}$ from the configured ship model. Then, the opposing dynamic obstacle can have a CSOG state generated from

$$\boldsymbol{x}_{do} = \begin{bmatrix} x_{os} + d_{os,do} * cos(\chi_{os} + \beta_{os,do}) \\ y_{os} + d_{os,do} * sin(\chi_{os} + \beta_{os,do}) \\ U_{do} \\ \chi_{os} + 180.0^\circ + \psi_{do,range} \end{bmatrix} \tag{2}$$

with $U_{do} \sim \mathcal{U}(U_{do,min}, U_{do,max})$, e.g. $\psi_{do,range} \sim \mathcal{U}(-10.0^\circ, 10.0^\circ)$, and with distance $d_{os,do}$ and bearing $\beta_{os,do}$ between the ships drawn from e.g. $d_{os,do} \sim \mathcal{U}(1000.0m, 2000.0m)$ and $\beta_{os,do} \sim \mathcal{U}(-15.0^\circ, 15.0^\circ)$, respectively. The limits of the uniform distributions are here chosen arbitrarily and will vary depending on the COLREGS situation and the variation ranges specified by the user. The generation of random waypoints and speed plans also uses uniform distributions, incrementally sampling waypoints with distances $d_{wp} \sim \mathcal{U}(d_{wp,min}, d_{wp,max})$ and bearings $\beta_{wp} \sim \mathcal{U}(\chi + \beta_{wp,min}, \chi + \beta_{wp,max})$, with corresponding desired speeds along each waypoint segment drawn from $U_d \sim \mathcal{U}(U + U_{d,min}, U + U_{d,max})$ until $n_{wps}$ waypoints are drawn, starting at the ship initial position and with the nominal direction along its initial course $\chi$.

As the initial ship CSOG states and plans must be feasible with respect to grounding hazards (ENC data) and not be on top of other ships, sampling from the distributions is repeated until collision and grounding-free configurations and waypoint segments are created.

Two scenario examples are shown in Figs. 3 - 4. The first depicts a case with a randomized OS CSOG state and plan, and historical AIS data for specifying trajectories of multiple DOs of different types. The second case shows a fully random generated scenario, with the generated waypoint plans for all vessels shown.

### *B. The Simulator*

The simulator has one main interface function `run() -> dict`, which runs through a list `configFile`

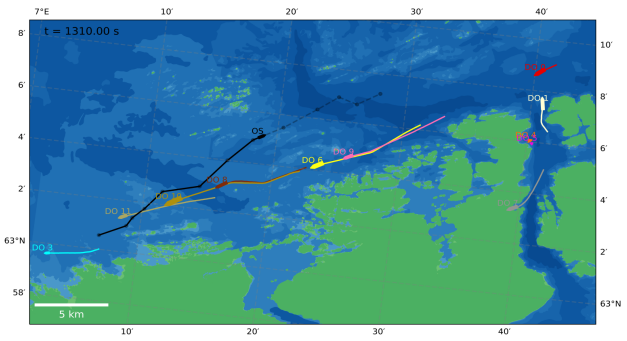


Fig. 3: Screenshot from simulation of a scenario with usage of historical AIS data.

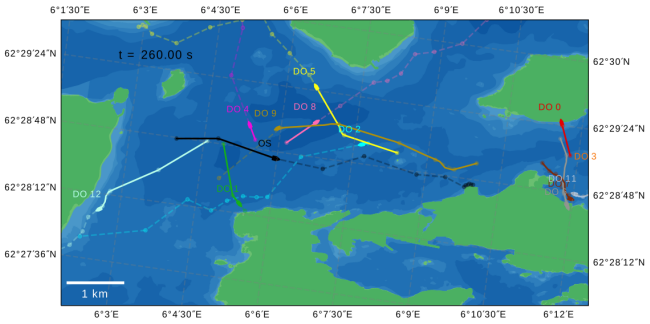


Fig. 4: Screenshot from simulation of a scenario with random generated data outside Ålesund, Norway.

for the specified scenarios. This list is either provided at the construction of the `Simulator` object or afterward through a setter function. The output dictionary stores simulation data, ship information, and scenario definitions which can be used for further result inspection by the user, or in the evaluator described in the next section.

For each scenario being run through, the simulator provides the scenario generator with a configuration file and gets a full scenario definition in return. As mentioned briefly in the previous section, a scenario definition consists of

- A list of fully configured `Ship` objects, with specified subsystems. This also includes an initial CSOG state, a set of waypoints and a speed plan to follow, or usage of a historical trajectory if this is specified. Furthermore, it includes the timespan for when the ships enter and exit the simulation.
- Scenario metadata. This among others includes loaded and preprocessed ENC data into polygons, coordinate system and transformations used, and the simulation timespan and time step.

The simulator then sets up the map environment through the preprocessed ENC data and uses the configured `Ship` objects through its interface. A flow chart representing the simulation process is shown in Fig. 5, and shows the following main ship interface functions:

`track(t, dt, trueDOstates) -> Tuple`: This function uses the onboard configured ship sensors and target

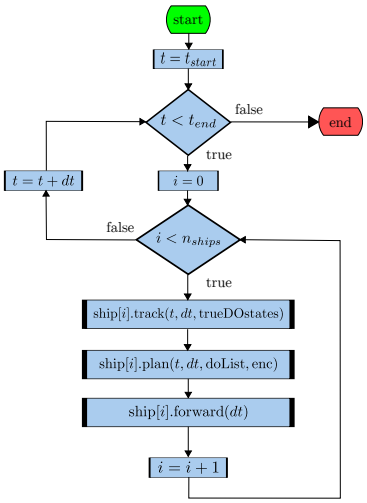


Fig. 5: COLAV simulator flow chart illustrating the internal of its `run() -> dict` function. The input scenario definition data, output simulation data, and other miscellaneous details are not shown for simplicity.

tracker to estimate the states and error covariances for target ships or dynamic obstacles (DOs) in its vicinity. The parameter $t$ is the current simulation time, $dt$ is the time between the current and previous track update, and the true DO states are used to generate sensor measurements. A tuple consisting of a tuple list of nearby DO information (ID, state estimates, and corresponding covariances) and sensor measurements used (if any) is returned.

`plan(t, dt, doList, enc) -> np.array`: Main planning function for the own-ship and other dynamic obstacle ships. The parameter $dt$ is here the time between the current and previous planning iteration. It utilizes the input list of dynamic obstacles and available ENC data. A COLAV planning algorithm is used to facilitate intelligent behaviour if such an algorithm is configured or provided. The utilized algorithm computes a reference trajectory (one-step or multi-step depending on the approach) $\boldsymbol{r}_d$. Otherwise, a nominal guidance system is used to compute references representing non-intelligent behavior.

`forward(dt) -> Tuple`: Simulates the vessel $dt$ seconds forward in time, using its ship model, controller, guidance, and/or COLAV planning algorithm. If the ship is configured to use historical AIS data, it will use its historical trajectory at a possibly interpolated time step. This function returns a tuple consisting of the new ship state, inputs, and references to get there. The simulator, in general, considers 3DOF motions, meaning a ship state $\boldsymbol{x} \in \mathbb{R}^6$ and an input vector $\boldsymbol{u} \in \mathbb{R}^3$ as is commonly used in [8]. The references computed by the ship COLAV system or guidance system are in general a trajectory $\boldsymbol{r}_d \in \mathbb{R}^{n_r \times N}$, where $n_r = 9$ consists of the desired poses, velocities and accelerations for the ship over a horizon of $N$ samples. Note again that the framework is agnostic to specific subsystems, meaning that it is possible to use other models, as long as the input/output interfaces are adhered to. The 3DOF kinetic model in [19] and kinematic model in [21] are implemented in the current framework version.

As the sensors in the `Ship` class can be configured with parameters for different maximal ranges, measurement sample rates, and accuracies, it allows for emulating sensor data under varying conditions. The target tracker is in the simplest case a Kalman Filter as was chosen in the configuration file shown in Fig. 2, but can in general be any type as long as its tracking interface is adhered to. This is also the case for the sensor types, where implementations of a `Radar` sensor class and an `AIS` transponder class are included in the current version. Future work involves developing more realistic sensor models that e.g. consider the effect of occlusion by objects such as land and other ships when detecting dynamic obstacles.

Information from the simulation can be visualized live, as specified by the user through a simulator configuration file. The previously shown Figs. 3 - 4 are all screenshots from the live visualization in the simulator. Fig. 6 shows a screenshot from a live simulation where sensor measurements, DO estimates, and error covariances are also shown. Results can alternatively be shown after a simulation run, as shown in Fig. 7. When simulating and evaluating thousands of scenarios, the simulations can also be run automatically in the background.

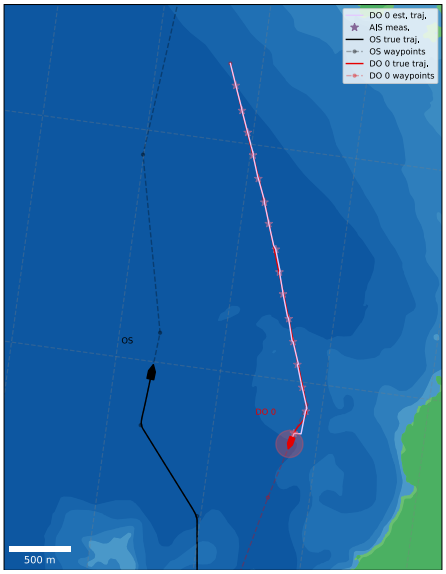


Fig. 6: Screenshot from simulation with target tracking uncertainty information also visualized, when using an AIS transponder as the own-ship sensor. The $3\sigma$ probability ellipse for the dynamic obstacle position tracking is also shown at the current time in red color.

### C. The Evaluator

After going through all considered scenarios with the simulator, one can use the `Evaluator` module to assess the performance of the vessels involved in all the scenarios. The simulation data from the simulator must first be converted into `VesselData` objects, which store all relevant ship

information to be used by the `Evaluator`. Alternatively, one can also use the evaluation class stand-alone by loading e.g. AIS data directly for use in the class. After the data is loaded, one can interface with the following main class function:

`evaluate() -> EvaluatorResults`: This function is responsible for scoring the behaviour of all vessels with respect to all other vessels, using the approaches outlined in [9]. The main behavioural rules 8, 13-17 from the COLREGS [1] are considered, which are relevant for power-driven vessels. The return type `EvaluatorResults` contains scores, penalties, and other information for all vessels, with respect to all metrics considered in the `Evaluator`.

The method outlined in [9] has a core functionality based on [26], and computes scores and penalties for vessel performances in COLREGS situations. This is based on metrics such as having a sufficient range to the other ship at the Closest Point of Approach (CPA) and whether or not a sufficiently large course and/or speed change during the situation has been undertaken in the case of the give-way vessel. The scores and penalties are compensated for the presence of grounding hazards by analysis of ENCs, see [9]. The module thus serves a good purpose for evaluating the performance of ships configured with COLAV planning algorithms. Furthermore, the module also enables the adjustment of COLAV planning algorithms, by using the performance assessment as feedback.

## III. Examples

We here illustrate usage of the full pipeline in two example cases. The first is a randomly generated head-on scenario, shown in Fig. 7. To generate the scenario, the configuration file from Fig. 2 was used. It is specified with map data and coordinate system information for the considered Ålesund area. Then, the scenario type parameter is set to specify a two-ship head-on case. The simulation timespan and time step are also provided. The own-ship is configured with specific subsystems, whereas the dynamic obstacle was configured to default subsystem settings as an entry for it was not provided in the ship configuration list. The default ship configuration settings entail the usage of a simple kinematic ship model with Line-of-Sight (LOS) guidance as described in [21]. The `Evaluator` is tuned similarly to what is found in [9]. All examples were generated and run on a Macbook Pro with an Apple Silicon M1 chip, for a scenario episode timespan of $500s$ and time step of $0.1s$. Then, a crude run-time estimate based on $50$ script executions for the entire scenario generation (excluding map data loading), simulation, and evaluation pipeline were $4.54 \pm 0.04s$ and $10.56 \pm 0.15s$ for the first and second case, respectively.

The results from evaluating the OS and DO performances are given in Table I, for which full definitions of the variables shown can be found in [9]. It shows that the OS attains a reasonably good total score for COLREGS rule 14 ($\mathcal{S}_{14}$), with a slight penalty due to a delayed give-way maneuver. This is evident through a non-zero $\mathcal{P}_{delay}$. The OS gets a good safety score $\mathcal{S}_{safety}$, which is given as a conditional weighted sum over the scores $\mathcal{S}_r$ and $\mathcal{S}_\Theta$ for the OS range and pose at the CPA, respectively. Furthermore, no penalty for non-apparent maneuvering ($\mathcal{P}_\Delta^{\neg ap}$) is given. The DO gets a lower score, due to making a maneuver at an even later stage than the OS. The reader is referred to [9] for more information on the calculation of the scores and penalties, and/or [26] for more details on COLREGS evaluation.

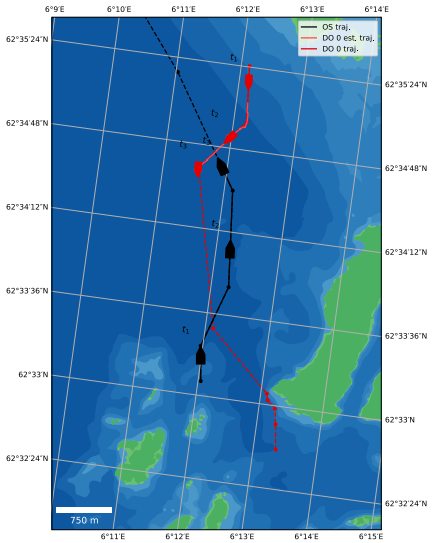


Fig. 7: A randomly generated head-on scenario near Ålesund, Norway. The vessels are shown at three different times over the simulation timespan.

TABLE I: Head-on scenario evaluation for OS. Variables $\mathcal{S}$ denote scores whereas variables starting with $\mathcal{P}$ are penalties.

(a) OS evaluation.

| Score/penalty | Value |
|---|---|
| $\mathcal{S}_{14}$ | 0.83 |
| $\mathcal{S}_{16}$ | 1.0 |
| $\mathcal{S}_{safety}$ | 1.0 |
| $\mathcal{S}_\Theta$ | 0.97 |
| $\mathcal{S}_r$ | 1.0 |
| $\mathcal{P}_{delay}$ | 0.16 |
| $\mathcal{P}_\Delta^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta U}^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta \chi}^{\neg ap}$ | 0.0 |

(b) DO evaluation.

| Score/penalty | Value |
|---|---|
| $\mathcal{S}_{14}$ | 0.54 |
| $\mathcal{S}_{16}$ | 1.0 |
| $\mathcal{S}_{safety}$ | 1.0 |
| $\mathcal{S}_\Theta$ | 0.99 |
| $\mathcal{S}_r$ | 1.0 |
| $\mathcal{P}_{delay}$ | 0.45 |
| $\mathcal{P}_\Delta^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta U}^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta \chi}^{\neg ap}$ | 0.0 |

The second case is a multi-ship scenario with four vessels, shown in Fig. 8 with resulting scores for the OS given in Table II. As the OS does not properly give way to DO 1 in the head-on situation, a zero score is given here. For the give-way crossing with DO 0, the OS performs a late maneuver and therefore gets a significant delayed penalty $\mathcal{P}_{delay}$, which leads to a low crossing score $\mathcal{S}_{15}$ and give-way score $\mathcal{S}_{16}$. The OS comes near collision with DO 2 as the obstacle does not give way properly. A near-perfect score is given for the OS here as it stands on correctly, with no penalty for port turns $\mathcal{P}_{pt}$ and no penalties for changing course $\mathcal{P}_{x,\Delta\chi}$

or speed up/down, $\mathcal{P}_{x,\Delta U\uparrow}$ and $\mathcal{P}_{x,\Delta U\downarrow}$. As the OS in the stand-on situation also had give-way obligations for DO 0 and 1, it received a compensation $\mathcal{C}_{x,gw}$ [9] for this. With respect to all DOs, the OS kept a sufficiently high safety margin to avoid a low safety score.

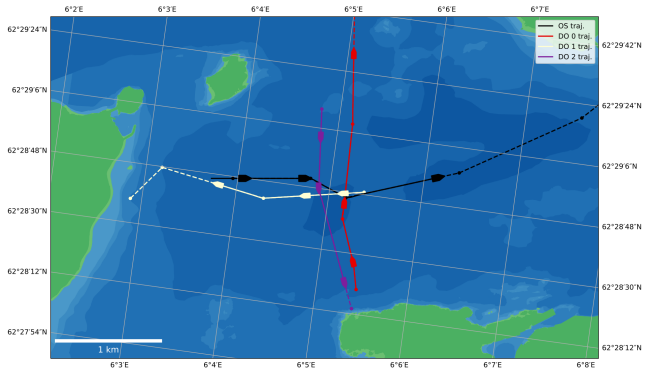


Fig. 8: A randomly generated multi-ship scenario near Ålesund, Norway. The vessels are shown at three different times over the simulation timespan, but without time annotation shown to simplify the figure.

## IV. Conclusion

This paper has introduced a simulation and evaluation framework for testing and validation of maritime automatic COLAV planning algorithms. It allows for testing any third-party or internally developed COLAV algorithm, in scenarios with intelligent and/or non-intelligent maneuvering obstacle ships. The framework is flexible in how scenarios used in validation are generated, where both randomly generated data and historical data can be used. As ENC data can also be included, it allows for grounding hazard visualization and consideration for the COLAV planning and evaluation. The framework simulator can run through a list of scenarios, from which the simulation data and results can be saved or used in an evaluator module for assessing vessel COLAV performance.

The random generation of states and plans when considering collisions with grounding hazards and other vessels can be computationally expensive. Thus, parts of this process should be implemented with multi-threading or parallelism, or in another language like Rust, C or C++, in order to speed up the pipeline.

In the future, the framework should be used as a pipeline for offline or online training/adjustment of the parameters in automatic COLAV algorithms, where the evaluation part will be a key component. Furthermore, as the COLAV performance assessment is highly dependent on the situation, better adjustment of the evaluation parameters based on the considered location, situation, and vessels involved must be properly investigated.

## V. Acknowledgements

The authors want to thank Magne Aune, Joachim Miller and Melih Akdag for their contribution to the preliminary simulator development.

TABLE II: Multi-ship scenario evaluation for OS. Variables $\mathcal{S}$ denote scores whereas variables starting with $\mathcal{P}$ are penalties.

(a) OS scores with respect to the give-way crossing with DO 0.

| Score/penalty | Value |
|---|---|
| $\mathcal{S}_{15}$ | 0.25 |
| $\mathcal{S}_{16}$ | 0.25 |
| $\mathcal{S}_{safety}$ | 1.0 |
| $\mathcal{S}_{\Theta}$ | 0.56 |
| $\mathcal{S}_{r}$ | 1.0 |
| $\mathcal{P}_{delay}$ | 0.75 |
| $\mathcal{P}_{\Delta}^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta U}^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta \chi}^{\neg ap}$ | 0.0 |

(b) OS scores with respect to the head-on with DO 1.

| Score/penalty | Value |
|---|---|
| $\mathcal{S}_{14}$ | 0.0 |
| $\mathcal{S}_{16}$ | 0.0 |
| $\mathcal{P}_{sts}$ | 1.0 |
| $\mathcal{P}_{nsb}$ | 0.0 |
| $\mathcal{S}_{safety}$ | 1.0 |
| $\mathcal{S}_{\Theta}$ | 0.97 |
| $\mathcal{S}_{r}$ | 0.94 |
| $\mathcal{P}_{delay}$ | 1.0 |
| $\mathcal{P}_{\Delta}^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta U}^{\neg ap}$ | 0.0 |
| $\mathcal{P}_{\Delta \chi}^{\neg ap}$ | 0.0 |

(c) OS scores with respect to the stand-on crossing with DO 2. Stage 2 and 3 correspond to the different parts of a COLREGS situation [9].

| Score/penalty | Value | |
|---|---|---|
| $\mathcal{S}_{15}$ | 0.95 | |
| $\mathcal{S}_{safety}$ | 1.0 | |
| $\mathcal{S}_{\Theta}$ | 0.99 | |
| $\mathcal{S}_{r}$ | 1.0 | |
| $\mathcal{S}_{17}$ | 1.0 | |
| $\mathcal{P}_{pt}$ | 0.0 | |
| | Stage 2 | Stage 3 |
| $\mathcal{P}_{x}$ | 0.0 | 0.0 |
| $\mathcal{P}_{x,\Delta \chi}$ | 0.0 | 0.0 |
| $\mathcal{P}_{x,\Delta U\uparrow}$ | 0.0 | 0.0 |
| $\mathcal{P}_{x,\Delta U\downarrow}$ | 0.0 | 0.0 |
| $\mathcal{C}_{x,gw}$ | 0.2 | 0.2 |

## References

[1] International Maritime Organization (IMO). "COLREGS - International Regulations for Preventing Collisions at Sea". In: *Convention on the International Regulations for Preventing Collisions at Sea, 1972* (1972).

[2] Azzeddine Bakdi and Erik Vanem. "Fullest COLREGs Evaluation Using Fuzzy Logic for Collaborative Decision-Making Analysis of Autonomous Ships in Complex Situations". In: *IEEE Transactions on Intelligent Transportation Systems* 23.10 (2022), pp. 18433–18445. DOI: 10.1109/TITS.2022.3151826.

[3] Michael R. Benjamin et al. "A method for protocol-based collision avoidance between autonomous marine surface craft". In: *Journal of Field Robotics* 23.5 (2006), pp. 333–346. DOI: https://doi.org/10.1002/rob.20121.

[4] Simon Blindheim and Tor Arne Johansen. "Electronic Navigational Charts for Visualization, Simulation, and Autonomous Ship Control". In: *IEEE Access* 10 (2022), pp. 3716–3737. DOI: 10.1109/ACCESS.2021.3139767.

[5] Mauro Candeloro, Anastasios M. Lekkas, and Asgeir J. Sørensen. "A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels". In: *Control Engineering Practice* 61 (2017), pp. 41–54. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2017.01.007.

[6] Giuseppe Casalino, Alessio Turetta, and Enrico Simetti. "A three-layered architecture for real time path planning and obstacle avoidance for surveillance USVs operating in harbour fields". In: *OCEANS 2009-EUROPE*. 2009, pp. 1–8. DOI: 10.1109/OCEANSE.2009.5278104.

[7] H. L. Chiang and L. Tapia. "COLREG-RRT: An RRT-Based COLREGS-Compliant Motion Planner for Surface Vehicle Navigation". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 2024–2031. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2801881.

[8] Thor Inge Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[9] Inger Berge Hagen. "Topics on Marine Collision Avoidance". PhD thesis. NTNU, 2022.

[10] H. Imazu. "Research on Collision Avoidance Manoeuvre (in Japanese)". PhD thesis. University of Tokyo, Japan, 1987.

[11] Boris Kluge and Erwin Prassler. "Recursive Probabilistic Velocity Obstacles for Reflective Navigation". In: vol. 24. Jan. 2003, pp. 71–79. ISBN: 3-540-32801-7. DOI: 10.1007/10991459_8.

[12] Y. Kuwata et al. "Safe Maritime Autonomous Navigation With COLREGS, Using Velocity Obstacles". In: *IEEE Journal of Oceanic Engineering* 39.1 (Jan. 2014), pp. 110–119. ISSN: 0364-9059. DOI: 10.1109/JOE.2013.2254214.

[13] Ø. A. G. Loe. "Collision Avoidance for Unmanned Surface Vehicles". MA thesis. NTNU, 2008.

[14] P. K. E. Minne. "Automatic testing of maritime collision avoidance algorithms". MA thesis. NTNU, 2017.

[15] OSP. *Open Simulation Platform*. Visited on 19.01.2023. 2023. URL: https://opensimulationplatform.com/.

[16] Tom Arne Pedersen et al. "Evolution of Safety in Marine Systems: From System-Theoretic Process Analysis to Automated Test Scenario Generation". In: *Journal of Physics: Conference Series* 2311.1 (July 2022), p. 012016. DOI: 10.1088/1742-6596/2311/1/012016.

[17] Michael Schuster, Michael Blaich, and Johannes Reuter. "Collision avoidance for vessels using a low-cost radar sensor". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 9673–9678.

[18] Paul G. Stankiewicz and Galen E. Mullins. "Improving Evaluation Methodology for Autonomous Surface Vessel COLREGS Compliance". In: *OCEANS 2019 - Marseille*. 2019, pp. 1–7. DOI: 10.1109/OCEANSE.2019.8867549.

[19] Trym Tengesdal, Edmund F. Brekke, and Tor A. Johansen. "On Collision Risk Assessment for Autonomous Ships Using Scenario-Based MPC". In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 14509–14516. ISSN: 2405-8963.

[20] Trym Tengesdal, Tor A. Johansen, and Edmund F. Brekke. "Ship Collision Avoidance Utilizing the Cross-Entropy Method for Collision Risk Assessment". In: *IEEE Transactions on Intelligent Transportation Systems* 23.8 (2022), pp. 11148–11161.

[21] Trym Tengesdal et al. "Ship Collision Avoidance and Anti Grounding Using Parallelized Cost Evaluation in Probabilistic Scenario-Based Model Predictive Control". In: *IEEE Access* 10 (2022), pp. 111650–111664. DOI: 10.1109/ACCESS.2022.3215654.

[22] Tobias Rye Torben et al. "Automatic simulation-based testing of autonomous ships using Gaussian processes and temporal logic". In: *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 0.0 (2022). DOI: 10.1177/1748006X211069277.

[23] Anete Vagale. "Evaluation Simulator Platform for Extended Collision Risk of Autonomous Surface Vehicles". In: *Journal of Marine Science and Engineering* 10.5 (2022). ISSN: 2077-1312. DOI: 10.3390/jmse10050705.

[24] K Vasstein et al. "Autoferry Gemini: a real-time simulation platform for electromagnetic radiation sensors on autonomous ships". In: *IOP Conference Series: Materials Science and Engineering* 929.1 (Nov. 2020), p. 012032. DOI: 10.1088/1757-899X/929/1/012032.

[25] K. Woerner. "Multi-Contact Protocol-Constrained Collision Avoidance for Autonomous Marine Vehicles". PhD thesis. Massachusetts Institute of Technology, 2016.

[26] Kyle Woerner et al. "Quantifying protocol evaluation for autonomous collision avoidance". In: *Autonomous Robots* 43.4 (Apr. 2019), pp. 967–991. ISSN: 1573-7527. DOI: 10.1007/s10514-018-9765-y.

[27] Feixiang Zhu, Zhengyu Zhou, and Hongrui Lu. "Randomly Testing an Autonomous Collision Avoidance System with Real-World Ship Encounter Scenario from AIS Data". In: *Journal of Marine Science and Engineering* 10.11 (2022). ISSN: 2077-1312. DOI: 10.3390/jmse10111588.

[28] Zhongxian Zhu et al. "An Efficient Ship Automatic Collision Avoidance Method Based on Modified Artificial Potential Field". In: *Journal of Marine Science and Engineering* 10.1 (2022). ISSN: 2077-1312. DOI: 10.3390/jmse10010003.