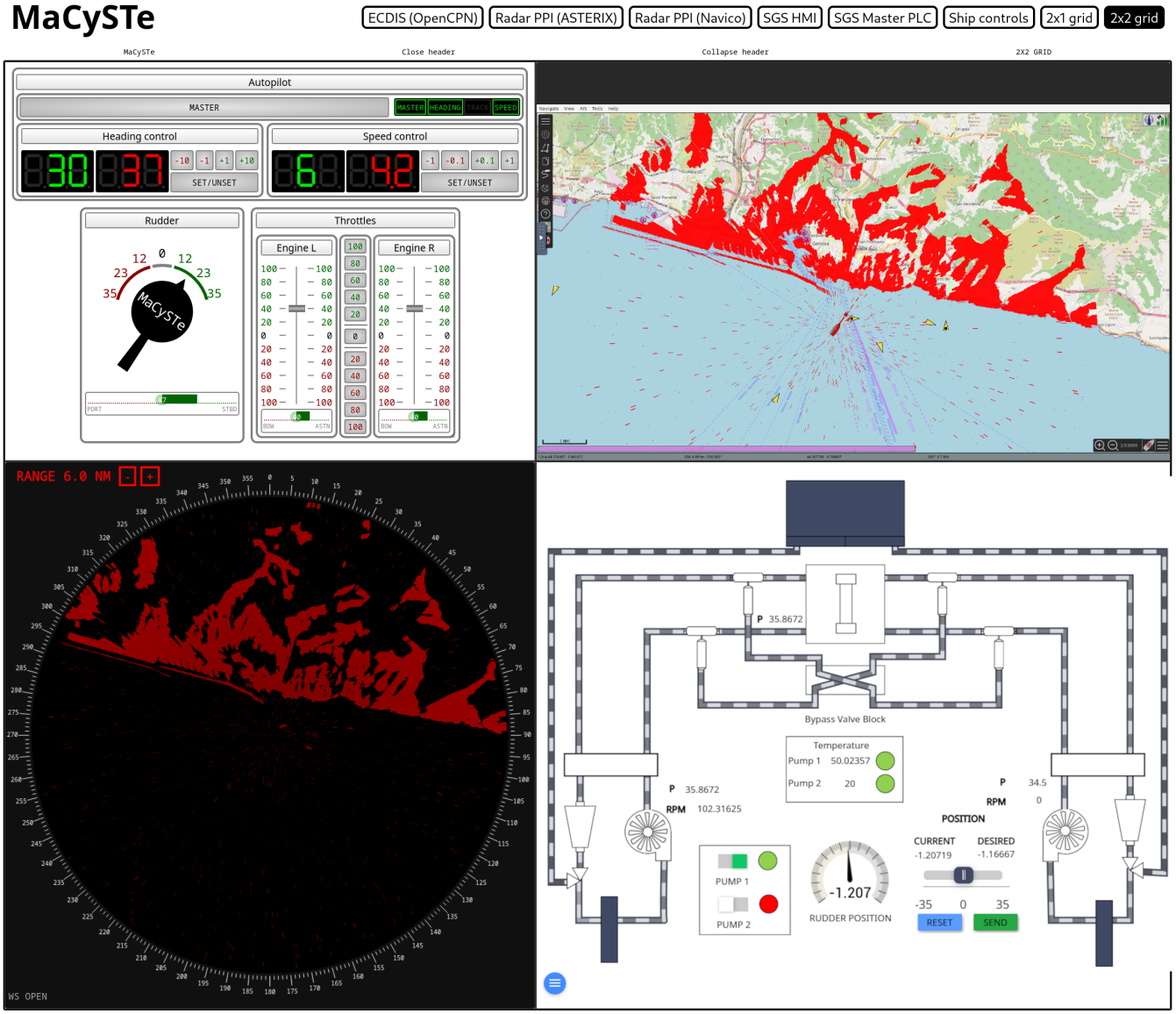
Introduction

Welcome to the documentation of the Maritime Cyber Security TEstbed (MaCySTe), a toolkit for cyber security research on ships.



Reference paper

IN PEER REVIEW - MaCySTe: a virtual testbed for maritime cybersecurity - G. Longo, A. Orlich, S. Musante, A. Merlo, E. Russo

Authors

MaCySTe has been developed in the University of Genova by Stefano Musante as his master thesis project.

Such work has then been extended by Giacomo Longo, Alessandro Orlich, and Enrico Russo in order to be a usable tool for the scientific community.

Contact details

Giacomo, Alessandro, and Enrico: *name.surname*@dibris.unige.it

Prerequisites

Before trying to run MaCySTe, please ensure the machine you are running on meets the following prerequisites.

Operating system

Any reasonably recent (2022+) Linux distribution should suffice.

For instance, MaCySTe initial development was done on Ubuntu 22.04 LTS and Fedora 37.

Programs

MaCySTe requires the following programs to function:

- `cat`
- `envsubst`
- `flatpak-builder`
  - A repository configured to provide `org.freedesktop.Sdk` and `org.freedesktop.Platform` (for instance, FlatHub)
- `flatpak`
- `ip`
- `make` , in particular its GNU implementation
- `podman` version `4.3+`
- `python` version `3.11+`
- `sysctl`
- `tee`
- `xdg-open`

Automatically checking prerequisites

All of these prerequisites can be checked by running `make check` from the repository root

```
$ make check
Found command cat
Found command envsubst
Found command flatpak-builder
Found command flatpak
Found command ip
Found command podman
Found command python3
Found command sysctl
Found command tee
Found command xdg-open
Podman version 4.3.1 is ok
Python version 3.11.1 (main, Dec  7 2022, 00:00:00) [GCC 12.2.1 20221121 (Red
Hat 12.2.1-4)] is ok
```

Requirements

Core scenario

With `SCENARIO_NAME=core` see here

- CPU: at least 4 cores
- RAM: at least 10GB
- GPU: OpenGL API or software rendering with `BC_HEADLESS=1`
- Storage: 50GB available

Attacker + SIEM scenario

With `SCENARIO_NAME=attacker_siem` see here

- CPU: at least 8 cores
- RAM: at least 12GB
- GPU: OpenGL API or software rendering with `BC_HEADLESS=1`
- Storage: 50GB available

Running

MaCySTe allows you to leverage every feature provided by the framework by issuing `make` commands.

Quickstart

Run `make check pull up run-bc`

Checking is your machine meets prerequisites

Run `make check`

```
$ make check
Found command cat
Found command envsubst
Found command flatpak-builder
Found command flatpak
Found command ip
Found command podman
Found command python3
Found command sysctl
Found command tee
Found command xdg-open
Podman version 4.3.1 is ok
Python version 3.11.1 (main, Dec  7 2022, 00:00:00) [GCC 12.2.1 20221121 (Red
Hat 12.2.1-4)] is ok
```

Starting

After you built the images and flatpaks

Run `make up` and then start the simulator

Starting the simulator

Run `make run-bc`

Opening the GUI

Run `make open-home`

Stopping

Run `make down`

Restart a single service

Run `make restart-service SERVICE=<service_name>` where `<service_name>` is the name of the folder inside of the scenarios base directory.

Building

Everything

Run `make build`

Equivalent to building containers and flatpaks.

Containers

Run `make build-containers`

Flatpaks

Run `make build-flatpaks`

Pulling pre-made images

If you want to use our pre-made images run `make pull`, to use pre-made flatpak, the file `src/flatpaks/bridgecommand/it.csec.Bridgecommand.flatpak`

Available scenarios

MaCySTe by default shipping with the `core` scenario but it also bundles 3 additional scenarios which can be selected by altering the `SCENARIO_NAME` variable.

Name	Description
`core`	The base scenario
`attacker`	The base scenario, augmented with the attacker addon
`siem`	The base scenario, augmented with the SIEM addon
`attacker_siem`	The base scenario, augmented with the attacker addon and SIEM addon

Changing options

The options file can be found in `src/settings.Makefile` settings can be either changed in the file or overriden during the make invocation like so: `<make command> <setting key>=<setting value>`

Available settings

Name	Description	Default
`SCENARIO_NAME`	Scenario to instantiate	`base`
`BC_HEADLESS`	Whenever to enable the render-less mode for BridgeCommand	*empty*
`BC_SCENARIO`	Scenario to load in BridgeCommand	`Genoa`

As a dataset generator

MaCySTe can be used for generating believable data streams of protocols specific to the maritime context.

For instance it can generate:

- NMEA sentences coming from the simulated Integrated Navigation System
- RADAR image data coming from the simulated radar antennas
- Industrial Control System data coming from the simulated PLCs

INS NMEA data

On the INS network, multiple instruments and the ECDIS stream and receive NMEA data by leveraging multicast UDP datagrams.

For the default MaCySTe configuration, such traffic flows to and from the IPv4 multicast address `239.0.1.1` on port `10110`.

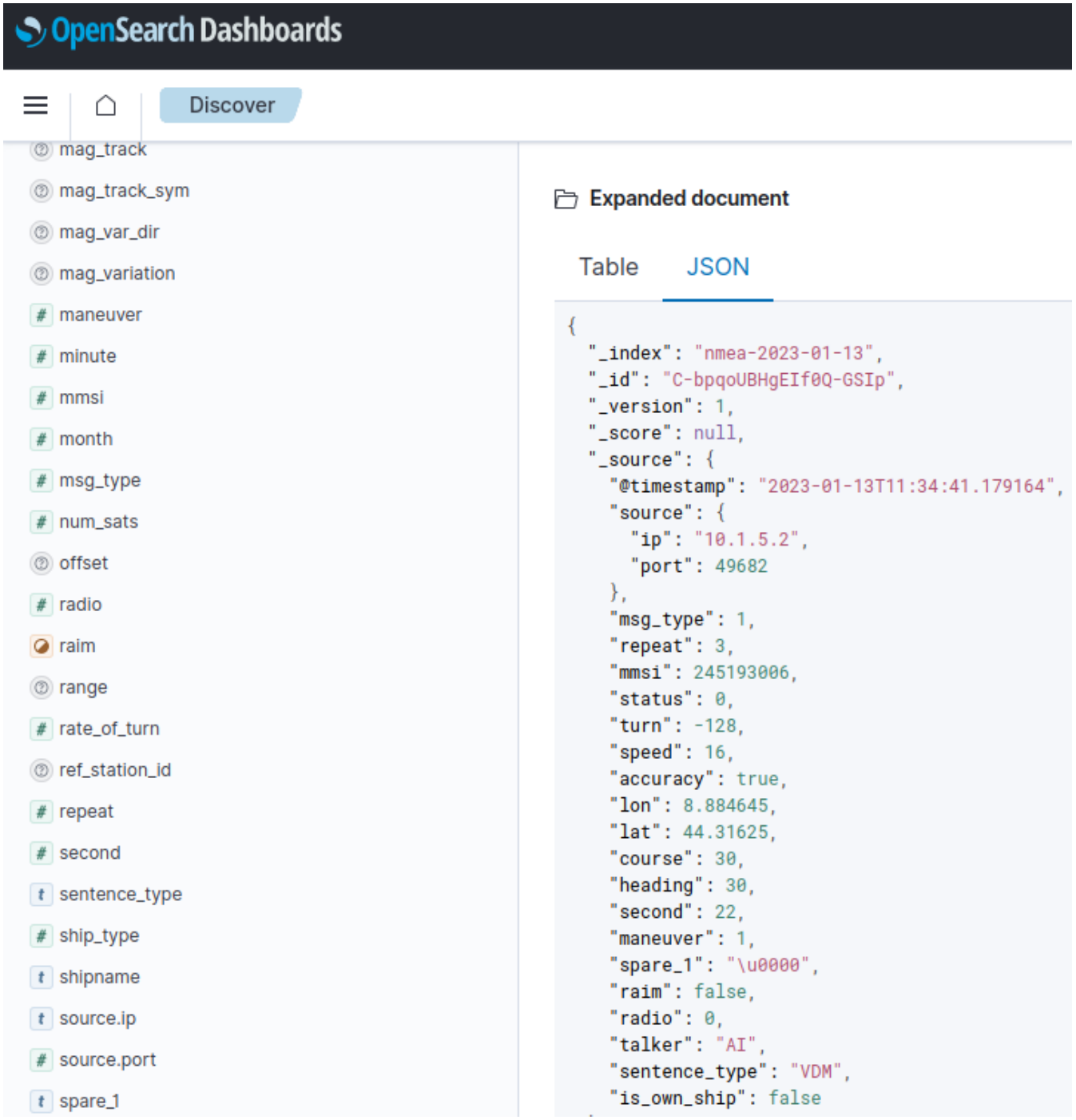
To acquire this traffic you can either:

1. Leverage the SIEM addon NMEA probe component to directly save every NMEA sentence in a structured format and query it
2. Leverage Wireshark on the host machine to inspect the network traffic
3. Add your own probe to the system by extending MaCySTe

Structured output to OpenSearch

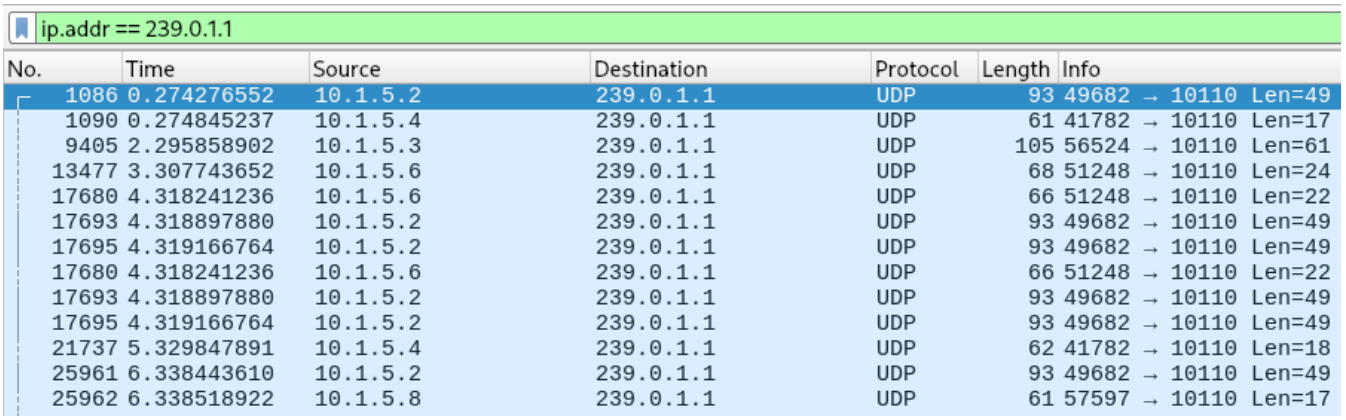
Ensure you have run MaCySTe with a scenario integrating OpenSearch (such as the built-in `siem` one).

Then, you will be able to access every NMEA sentence in structured format by opening the MaCySTe GUI, clicking on *SIEM*, and taking a look from the *Discover* page at messages present inside of the `nmea-*` OpenSearch indexes



Leveraging Wireshark

All of the flowing traffic can be inspected on the host by running Wireshark and listening to the special `any` interface


ip.addr == 239.0.1.1
No. Time Source Destination Protocol Length Info
1086 0.274276552 10.1.5.2 239.0.1.1 UDP 93 49682 → 10110 Len=49
1090 0.274845237 10.1.5.4 239.0.1.1 UDP 61 41782 → 10110 Len=17
9405 2.295858902 10.1.5.3 239.0.1.1 UDP 105 56524 → 10110 Len=61
13477 3.307743652 10.1.5.6 239.0.1.1 UDP 68 51248 → 10110 Len=24
17680 4.318241236 10.1.5.6 239.0.1.1 UDP 66 51248 → 10110 Len=22
17693 4.318897880 10.1.5.2 239.0.1.1 UDP 93 49682 → 10110 Len=49
17695 4.319166764 10.1.5.2 239.0.1.1 UDP 93 49682 → 10110 Len=49
17680 4.318241236 10.1.5.6 239.0.1.1 UDP 66 51248 → 10110 Len=22
17693 4.318897880 10.1.5.2 239.0.1.1 UDP 93 49682 → 10110 Len=49
17695 4.319166764 10.1.5.2 239.0.1.1 UDP 93 49682 → 10110 Len=49
21737 5.329847891 10.1.5.4 239.0.1.1 UDP 62 41782 → 10110 Len=18
25961 6.338443610 10.1.5.2 239.0.1.1 UDP 93 49682 → 10110 Len=49
25962 6.338518922 10.1.5.8 239.0.1.1 UDP 61 57597 → 10110 Len=17

RADAR data

Radar image data in MaCySTe can be generated with two protocols:

- Standard ASTERIX category 240
- Proprietary Navico BR24

For both of these protocols use 4096 spokes in a revolution, with each spoke being comprised of 512 8-bit cells.

ASTERIX data

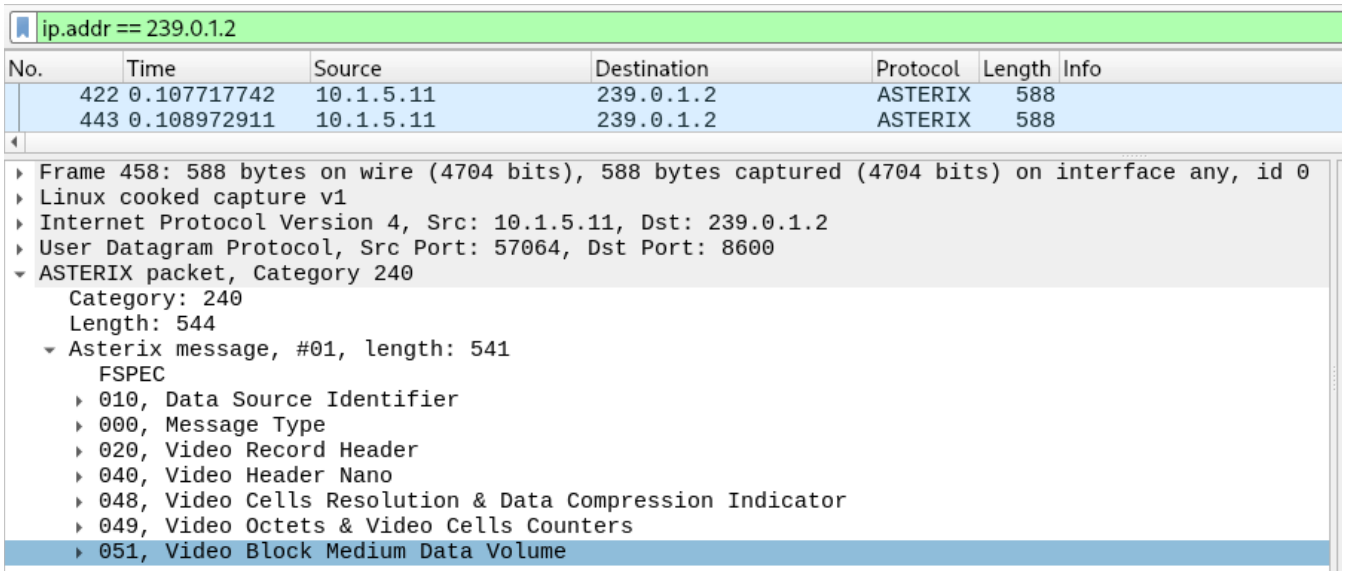
ASTERIX is a protocol for radar data exchange, MaCySTe can generate it according to its standard profile (UAP).

In the default configuration MaCySTe will send ASTERIX video to address `239.0.1.2` on port `8600`

For more details on the protocol structure we refer the interested reader to section **2.E** of Attacking (and defending) the Maritime Radar System

```
@misc{https://doi.org/10.48550/arxiv.2207.05623,
  doi = {10.48550/ARXIV.2207.05623},
  url = {https://arxiv.org/abs/2207.05623},
  author = {Longo,  G. and Russo,  E. and Armando,  A. and Merlo,  A.},
  keywords = {Cryptography and Security (cs.CR),  FOS: Computer and
information sciences,  FOS: Computer and information sciences},
  title = {Attacking (and defending) the Maritime Radar System},
  publisher = {arXiv},
  year = {2022},
  copyright = {arXiv.org perpetual,  non-exclusive license}
}
```

As with NMEA, ASTERIX can be listened to and dissected with Wireshark.



You can visualize the ASTERIX data flow inside of the PPI.

NAVICO data

MaCySTe can generate radar data for a NAVICO BR24 radar system.

Like its real counterpart (a system coming in a ready-to-use pre-configured state) the IP addresses and ports used by Navico are fixed.

The protocol is proprietary and our implementation was inspired by the following research article:

```
@incollection{dabrowski2011digital,
  title={A digital interface for imagery and control of a Navico/Lowrance
broadband radar},
  author={Dabrowski, Adrian and Busch, Sebastian and Stelzer, Roland},
  booktitle={Robotic Sailing},
  pages={169--181},
  year={2011},
  publisher={Springer}
}
```

You can visualize the NAVICO data flow inside of the PPI.

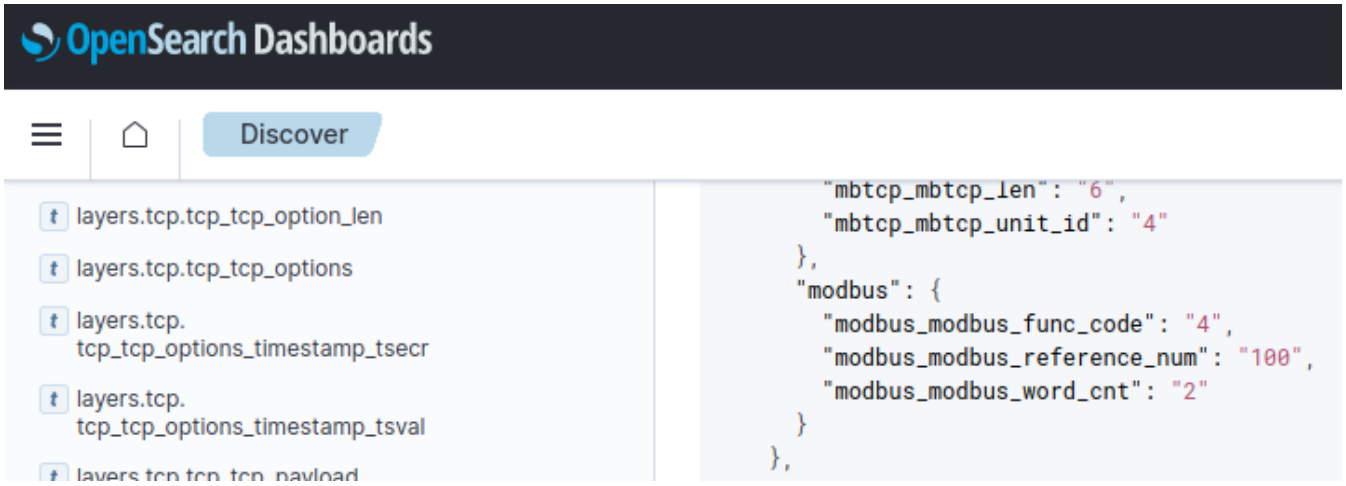
Industrial Control System data

Industrial Control System data in MaCySTe uses the ModBus protocol over TCP on the standard port 502.

For the core scenario, the helm, engine telegraphs, and internal Steering Gear Control System elements can be instrumented by leveraging a custom ModBus probe.

To look at the data, ensure you have run MaCySTe with a scenario integrating OpenSearch and the probe (such as the built-in `siem` one).

Then, you will be able to access every ModBus packet in structured format by opening the MaCySTe GUI, clicking on *SIEM*, and taking a look from the *Discover* page at messages present inside of the `modbus-*` OpenSearch indexes



As a target for attacks

MaCySTe reproduces the network infrastructure of a real bridge.

As a result, attacks can overhear and inject data as if they were acting in a real environment.

We provide two sample attacks:

1. Targeted at the Integrated Navigation System
2. Targeted at the RADAR

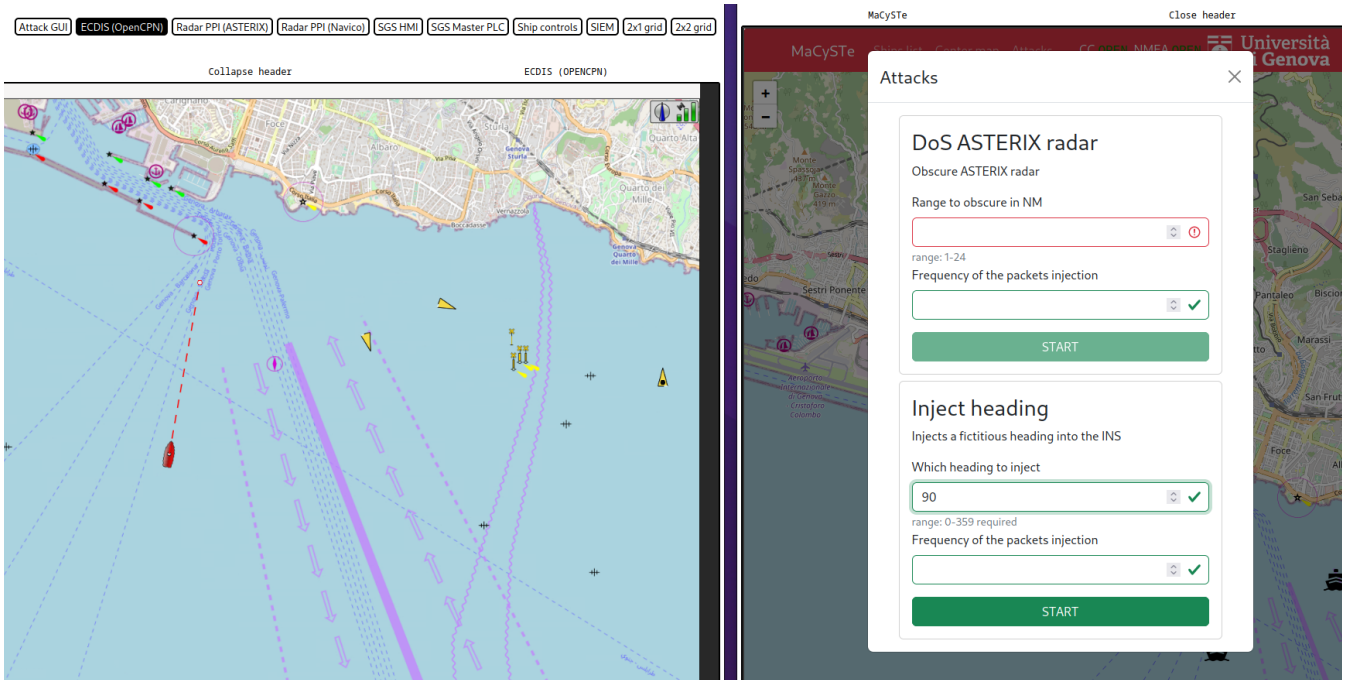
Attacking the Integrated Navigation System

MaCySTe can perform attacks targeted at the INS instruments.

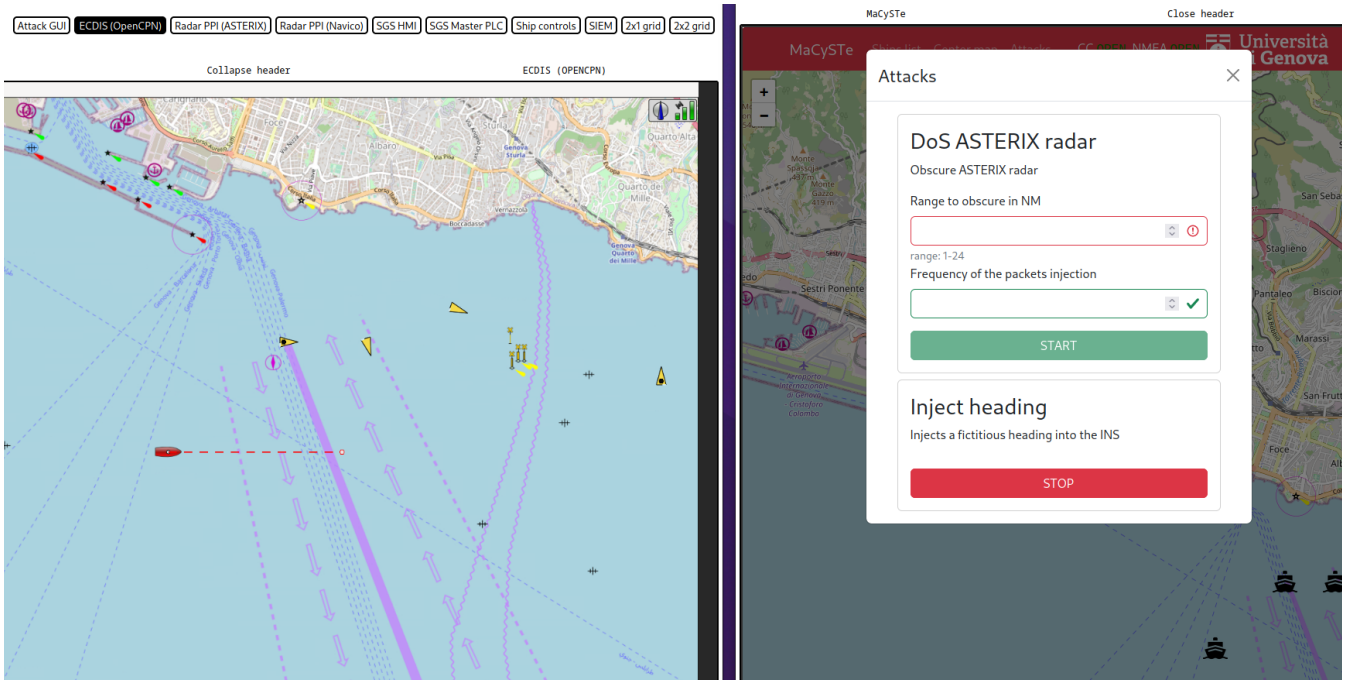
Bundled with MaCySTe there is a simple attack injecting a fictitious heading value into the navigation network.

To use it, ensure to have deployed a scenario containing the malware (such as `attacker_siem`) and access the attacker GUI from the MaCySTE GUI.

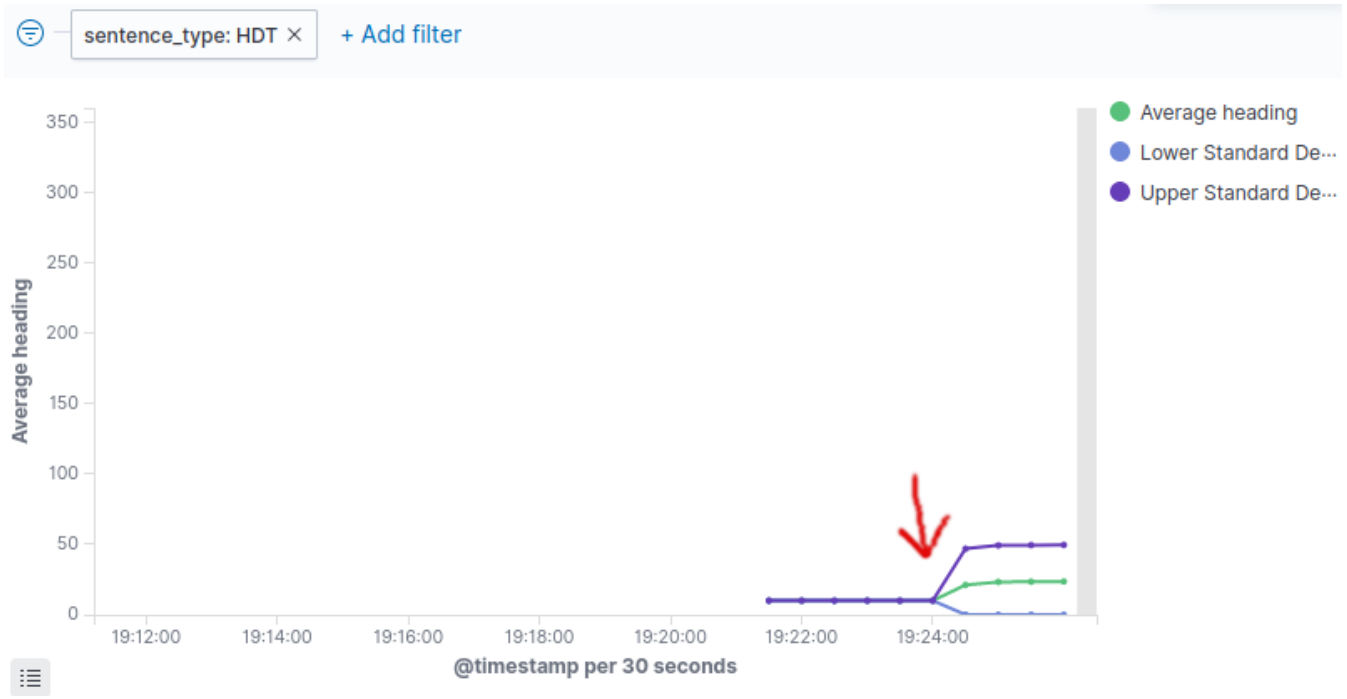
Once there, click on attacks and input the heading you wish to inject.



After starting the attack, you will see on other instruments such as the ECDIS a modified heading.



You will also be able to detect it within the SIEM.



Attacking the radar

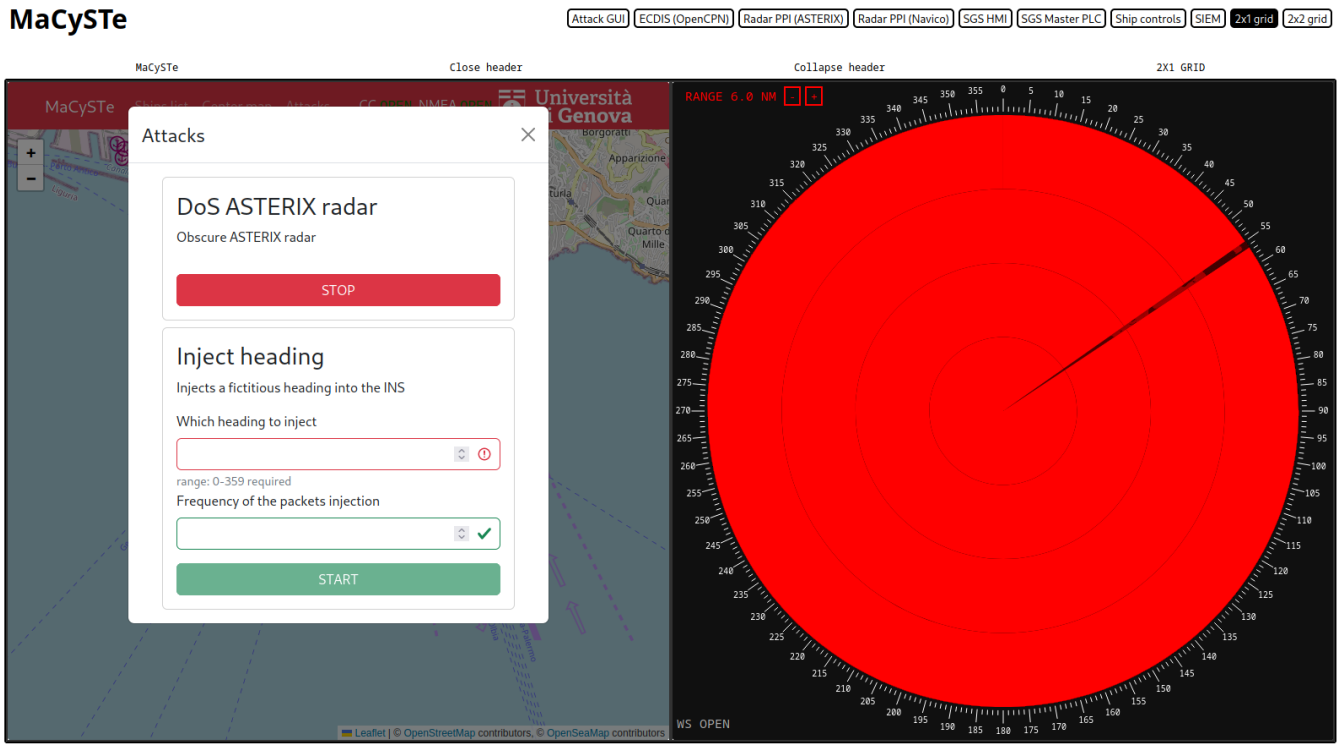
In MaCySTe we provide a sample implementation of the ASTERIX DoS attack on radar systems described in:

```
@misc{https://doi.org/10.48550/arxiv.2207.05623,
  doi = {10.48550/ARXIV.2207.05623},
  url = {https://arxiv.org/abs/2207.05623},
  author = {Longo,  G. and Russo,  E. and Armando,  A. and Merlo,  A.},
  keywords = {Cryptography and Security (cs.CR),  FOS: Computer and
information sciences,  FOS: Computer and information sciences},
  title = {Attacking (and defending) the Maritime Radar System},
  publisher = {arXiv},
  year = {2022},
  copyright = {arXiv.org perpetual,  non-exclusive license}
}
```

To use it, ensure to have deployed a scenario containing the malware (such as `attacker_siem`) and access the attacker GUI from the MaCySTE GUI.

Once there, click on attacks, select a range and start the DoS.

The result, as you can see below will be a packet obscuring the entire PPI.



Ship features

In this section, we describe which ship elements can be found and interacted with in MaCySTe.

Check in the sidebar which elements are available.

Network

MaCySTe simulates the networks onboard of the ship with two technologies:

- MACVLAN networks attached to dummies which are not reachable by the host and act as completely isolated L2 domains
- Bridge networks reachable by the host

MACVLAN networks

These MACVLAN networks act as virtual cables, completely segregated from the host and allowing a pristine network environment.

- **Bridge** (`10.1.5.0/24`), the network containing the ship Integrated Navigation System
- **Control** (`10.1.3.0/24`), the network containing the ship control systems and engineering workstation
- **Serial** (`10.1.2.0/24`), a network emulating direct connections (*this is a non-scenario network with unrepresentative traffic*)
- **NATS** (`10.1.4.0/24`), a network allowing communication with the message queue (*this is a non-scenario network with unrepresentative traffic*)

Bridge networks

These networks are host-reachable and are used for interacting with the scenario components

- **Management** (`192.168.249.0/24`), the network allowing the host to reach tools such as the GUI (*this is a non-scenario network with unrepresentative traffic*)

Attacker addon

These additional networks will be deployed as part of the attacker addon

- **Simulated internet** (MACVLAN `198.51.0.0/16`), a network simulating a public internet

SIEM addon

These additional networks will be deployed as part of the SIEM addon

- **SIEM** (MACVLAN `10.1.6.0/24`), a network joining the probes and the SIEM

Scenarios

MaCySTe allows the user to add new scenarios in the simulation environment.
This essentially requires two steps to be performed:

- Scenario and World generation for the ship simulator (BridgeCommand)
- Download of Navigation Charts for the ECIDS (opencpn)

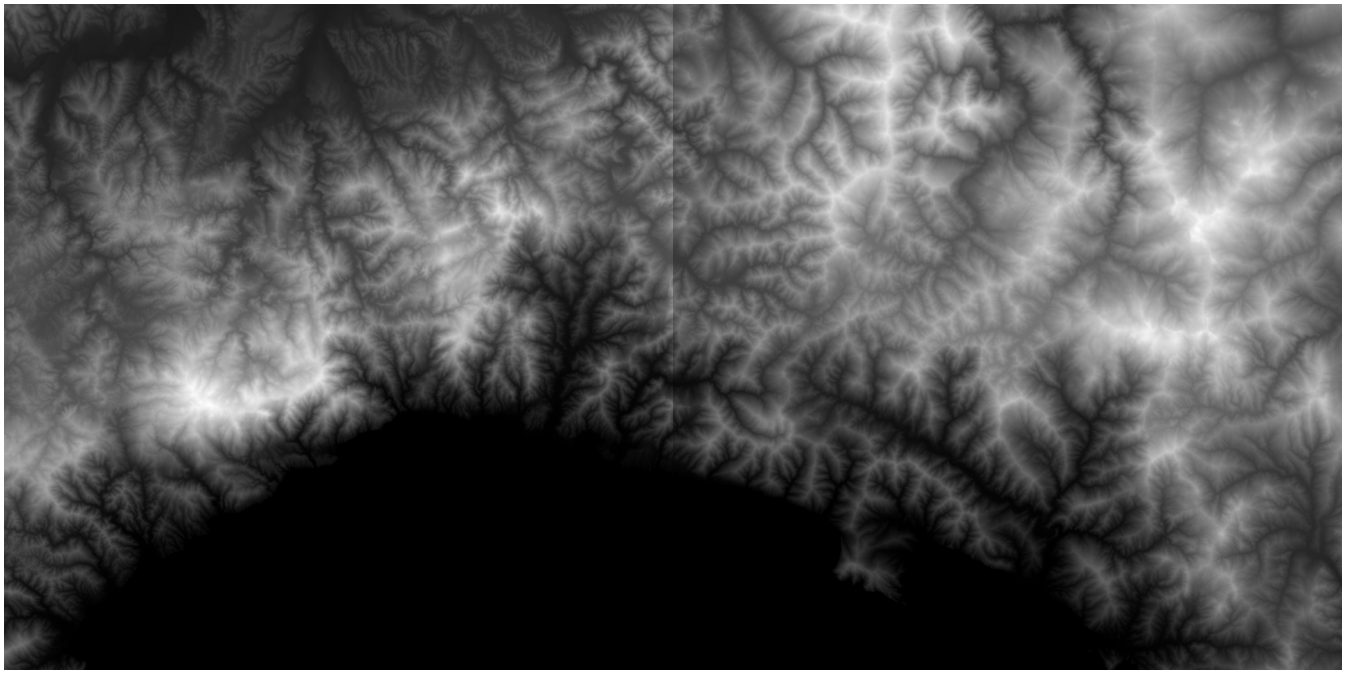
The former can be achieved by following the BridgeCommand documentation ([1] [2]).
The files have to be placed in the corresponding directories under `flatpaks/bridgecommand`.

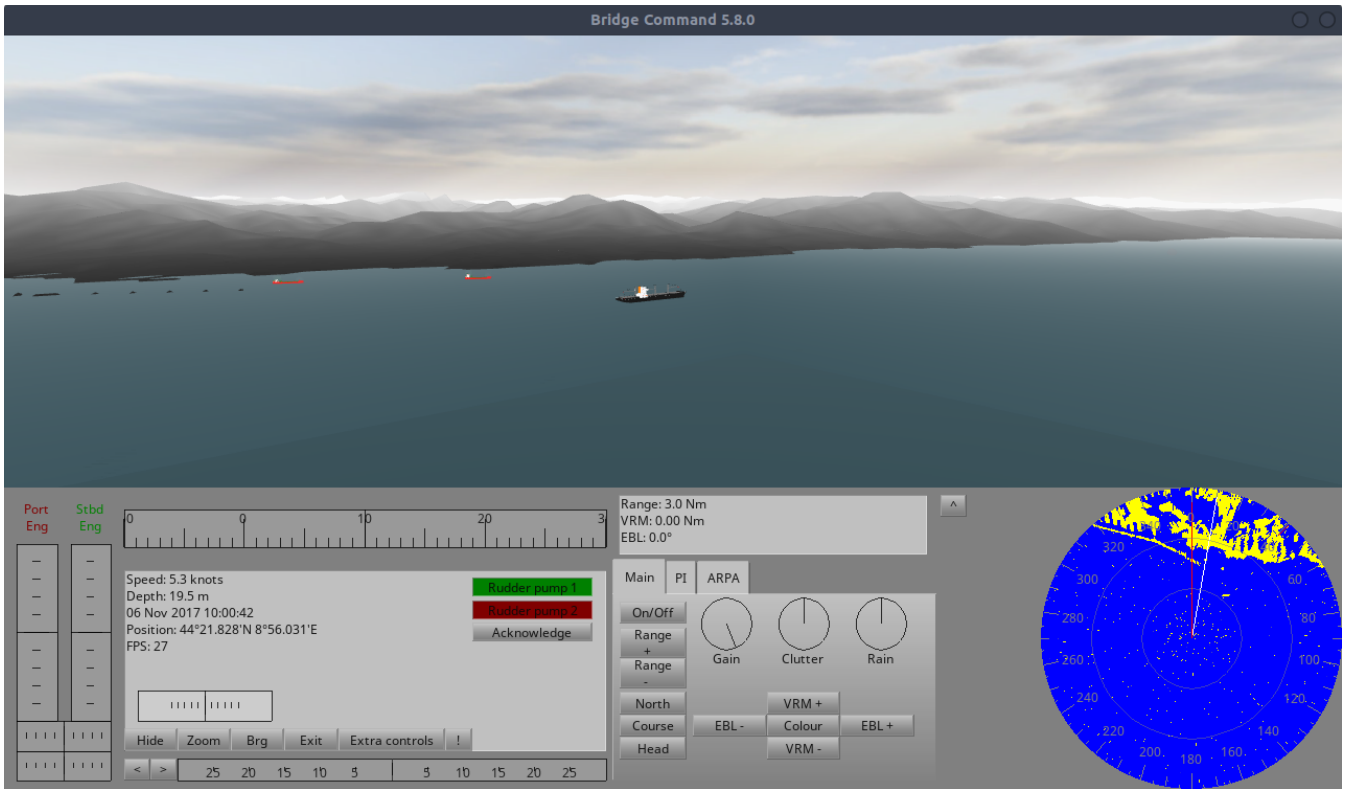
For the latter we suggest using OpenSeaMap ([3]).
The charts must be placed in `${CONFIG_DIR}/opencpn/charts`.

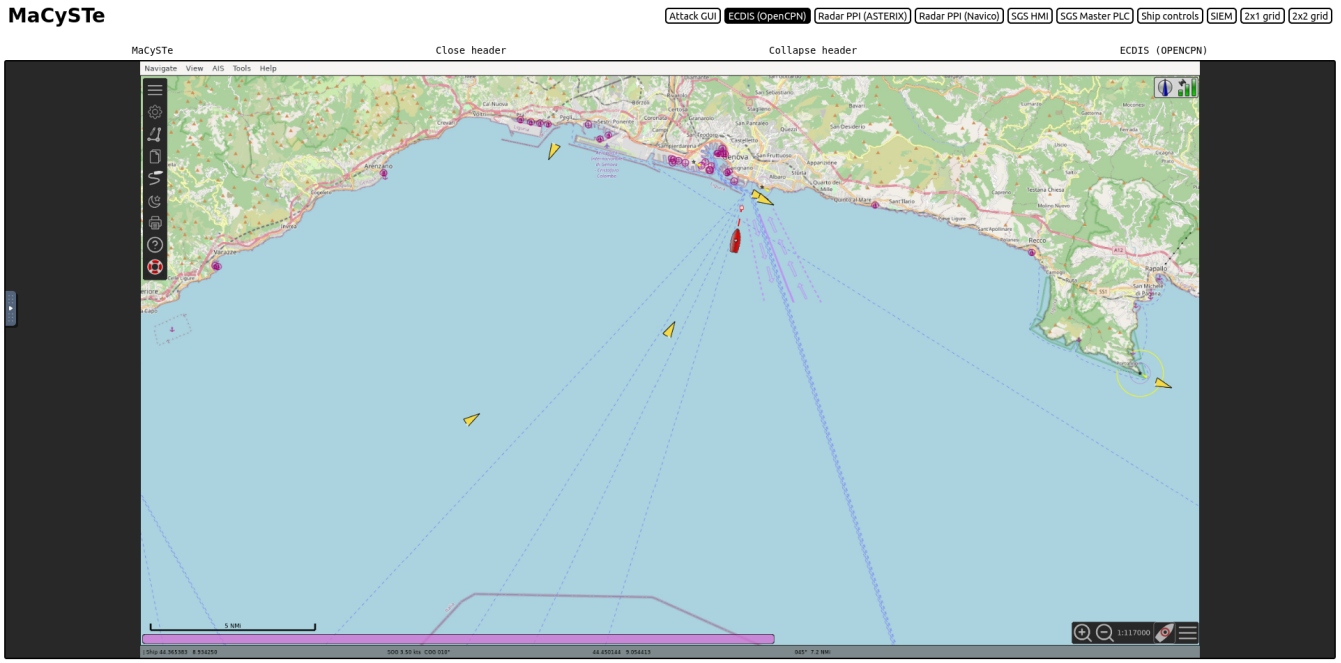
MaCySTe ships with a custom scenario of the Port of Genoa described in the following section.

Genova harbor

We added to the ship simulator a scenario set on the *Ligurian Sea* and the *Port of Genoa* (Coordinates: 44◦ 24' 10'' N, 8◦ 55' 0'' E). To create the 3D model we used the heightmap tiles made available by *the Italian National Institute of Geophysics and Volcanology* [1].





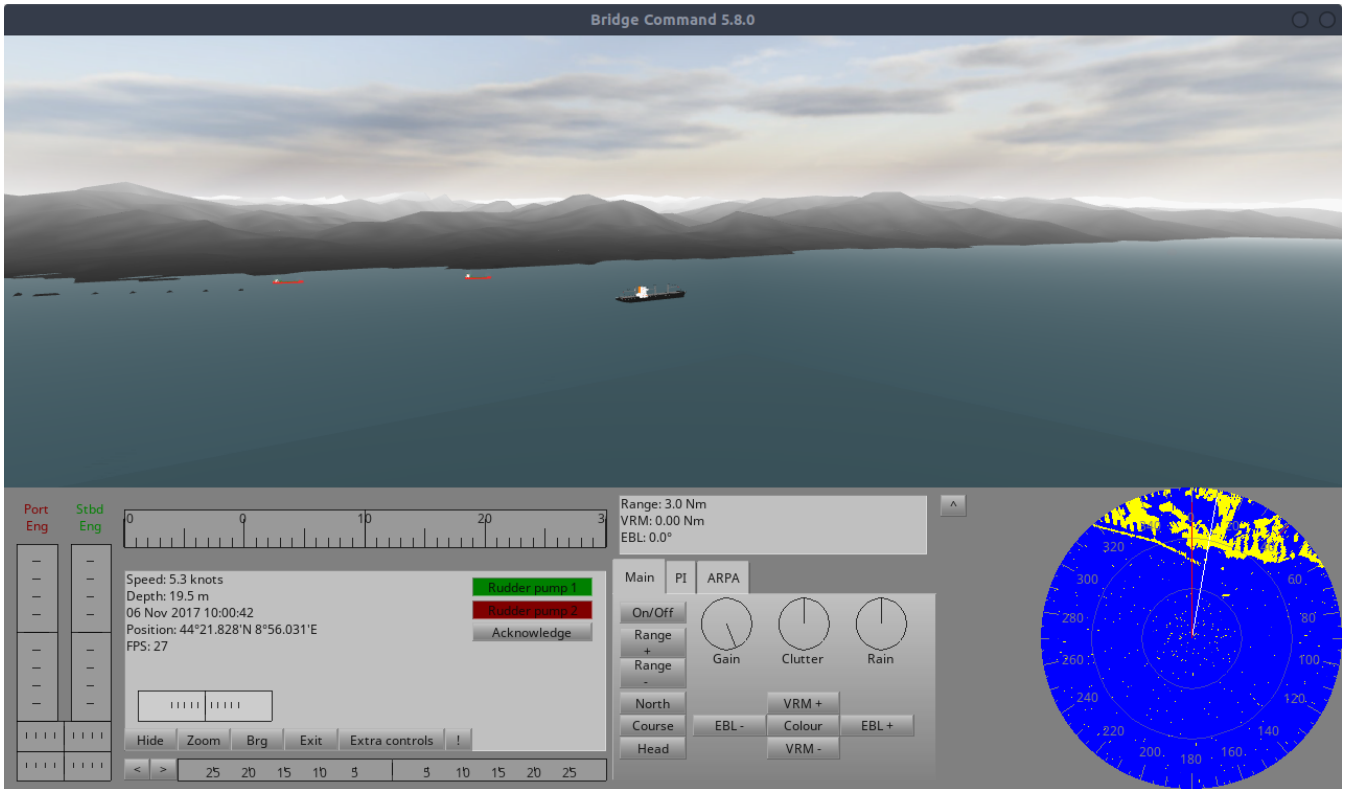

MaCySTe
Attack GUI
ECDIS (OpenCPN)
Radar PPI (ASTERIX)
Radar PPI (Navico)
SGS HMI
SGS Master PLC
Ship controls
SIEM
2x1 grid
2x2 grid
MaCySTe
Close header
Collapse header
ECDIS (OPENCPN)
Navigate
View
AIS
Tools
Help

Bridge Command

Inside of MaCySTe, the bridge simulator is another project: *Bridge Command*.

Bridge Command is an open source and liberally licensed ship simulator authored by James Packer.

Please refer to its documentation for more information.



Modifications

MaCySTe heavily modified Bridge Command to integrate it (patches are in `src/flatpacks/bridgecommand/bc-patches` relative to repo root) adding the following alterations:

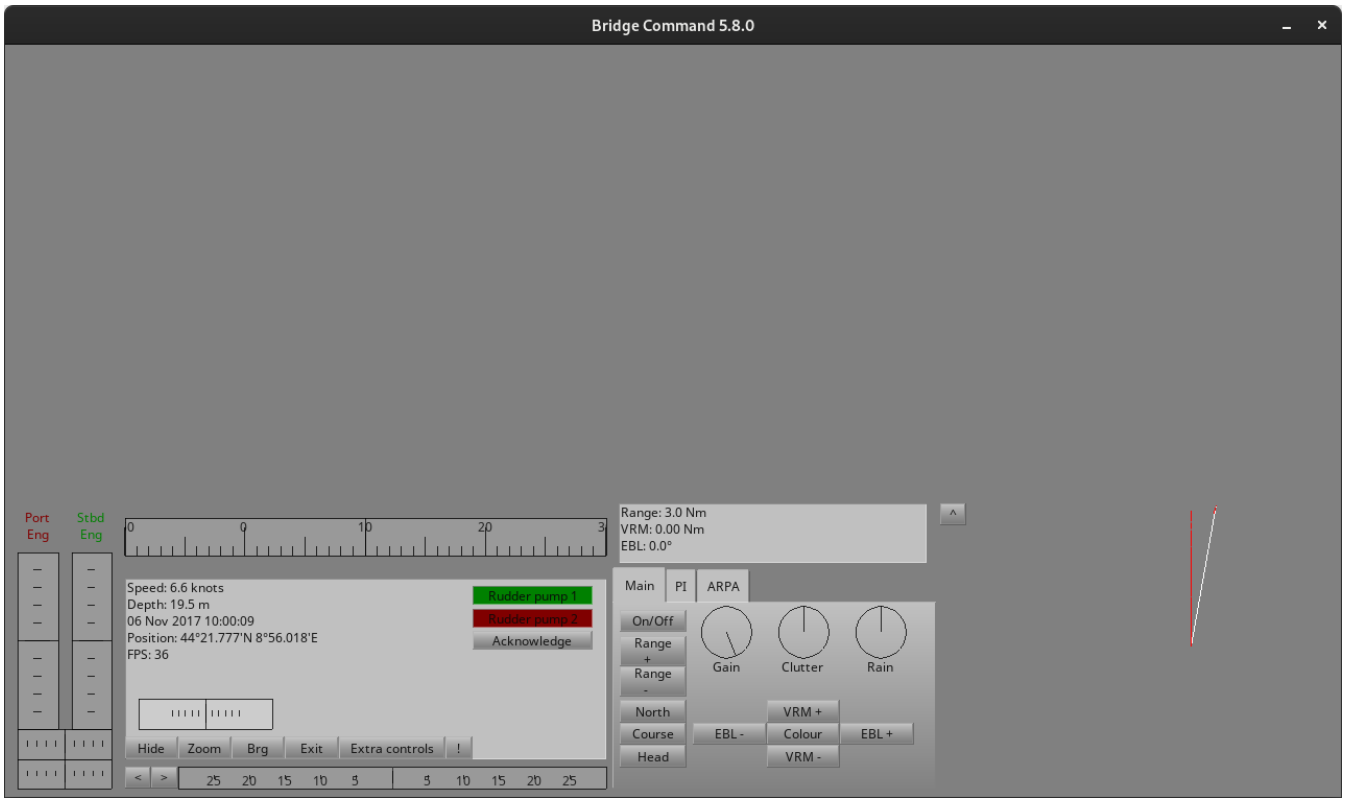
- We added AIS NMEA sentences (VDM, VDO) output
- We added the VTG NMEA sentence output
- We added the VHW NMEA sentence output
- We allow to automatically skip the loading pause and preseed the scenario by setting the `SKIP_PAUSE` and `SCENARIO_NAME` variables
- We added integration with the NATS.c library
- We added a variable `NMEA_TO_NATS` which once set will send every NMEA sentence to NATS
- We added a variable `SKIP_MENU` to skip the initial Bridge Command pause

- We added an exporter for raw radar image data to NATS KV store
- We added an integration for getting the rudder position from NATS
- We added an integration for getting the engine telegraph positions from NATS
- We added a way for disabling all rendering code in Bridge Command by setting the `HEADLESS` environment variable

Headless mode

Once the headless mode is enabled, Bridge Command will run skipping all rendering code, this can greatly improve the performance of MaCySTe whenever no rendering device is available.

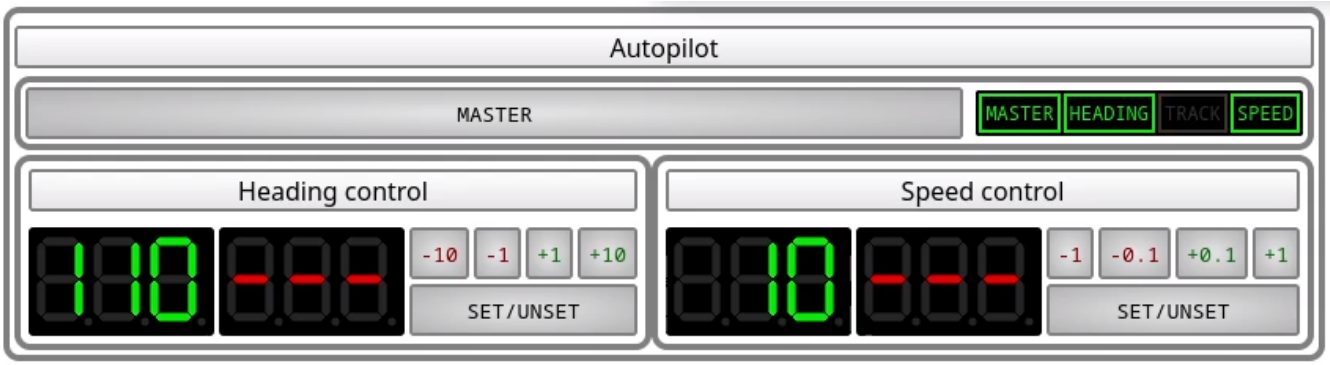
On Linux, you can take a look at Bridge Command outputs in the command line, if you see *llvmpipe* being mentioned then Bridge Command is using software rendering and you should use its headless mode.



Autopilot

MaCySTe bundles a tri-function autopilot allowing the automatic control of a ship.

Its interface can be accessed from the instruments GUI.



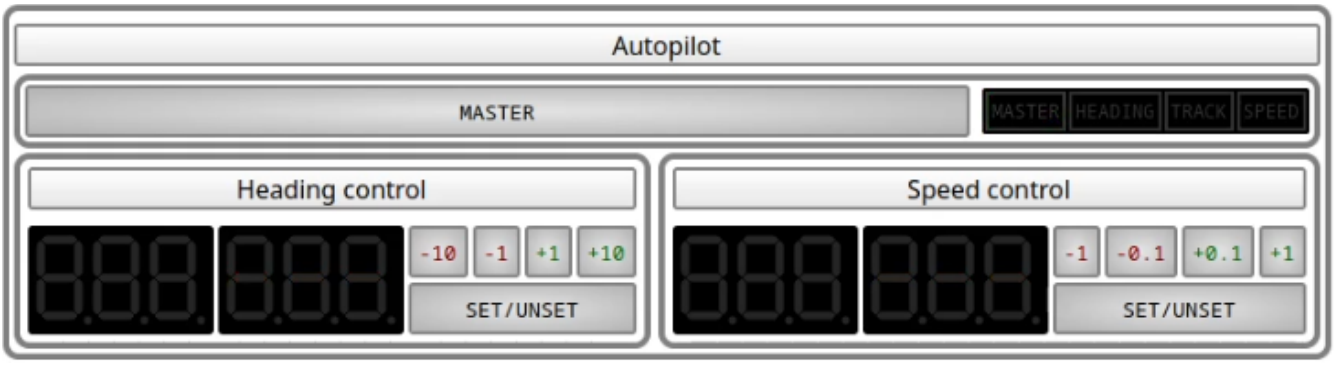
These three functions are

- Heading control
- Track control
- Speed control

Turning on the autopilot

The autopilot has a master switch, allowing you to completely disable it.

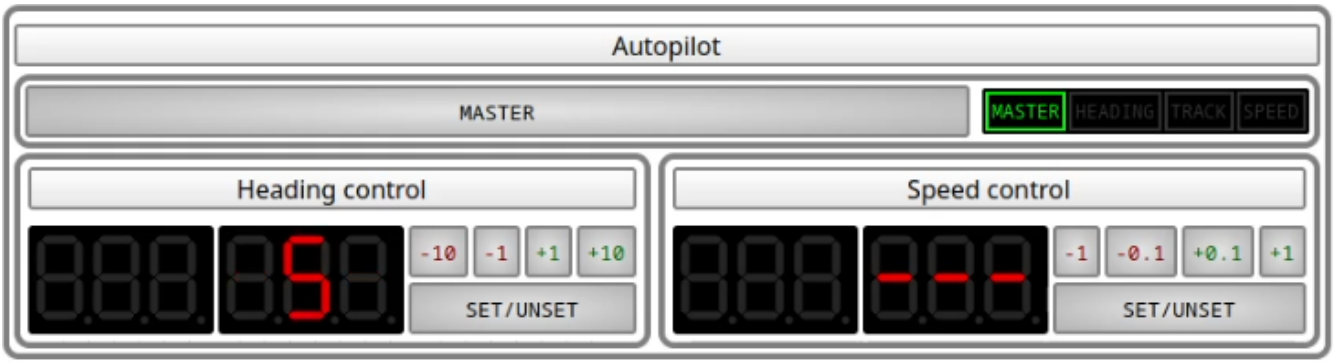
Whenever the autopilot is disabled, its master light does not illuminate and the bottom displays stay dark.



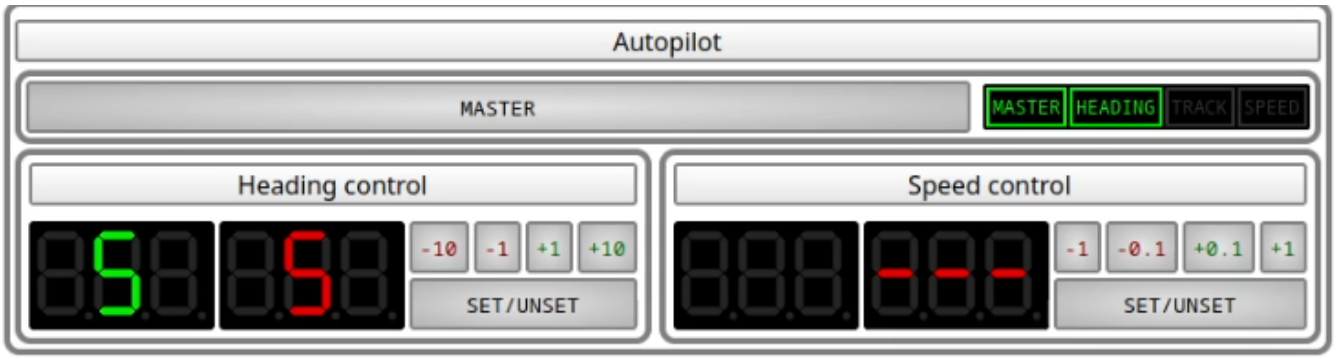
Heading control

Heading control will keep the boat pointing at a particular heading.

To use it, prepare with the buttons the heading you wish to hold.

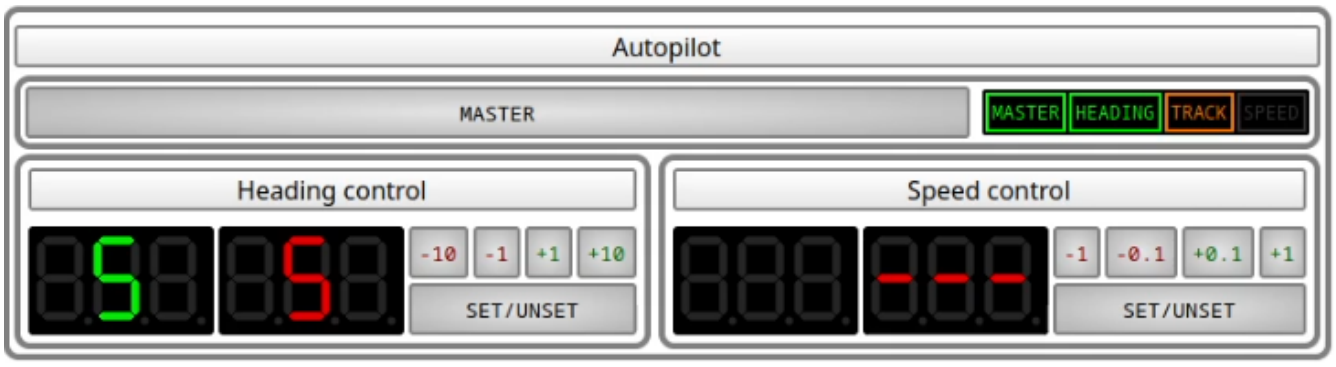


then, press the *set/unset* button



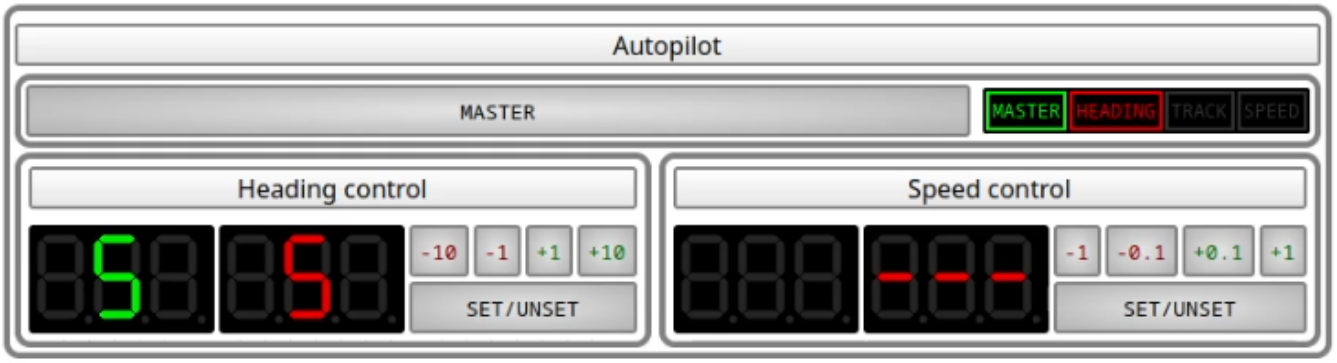
Warnings

If heading control is enabled but some waypoint source is offering a route suitable for track control applications, the track control light will flash in orange indicating that a switchover is possible



Errors

If heading control is enabled but the autopilot cannot determine the current heading, the heading control light will flash in red indicating a failure

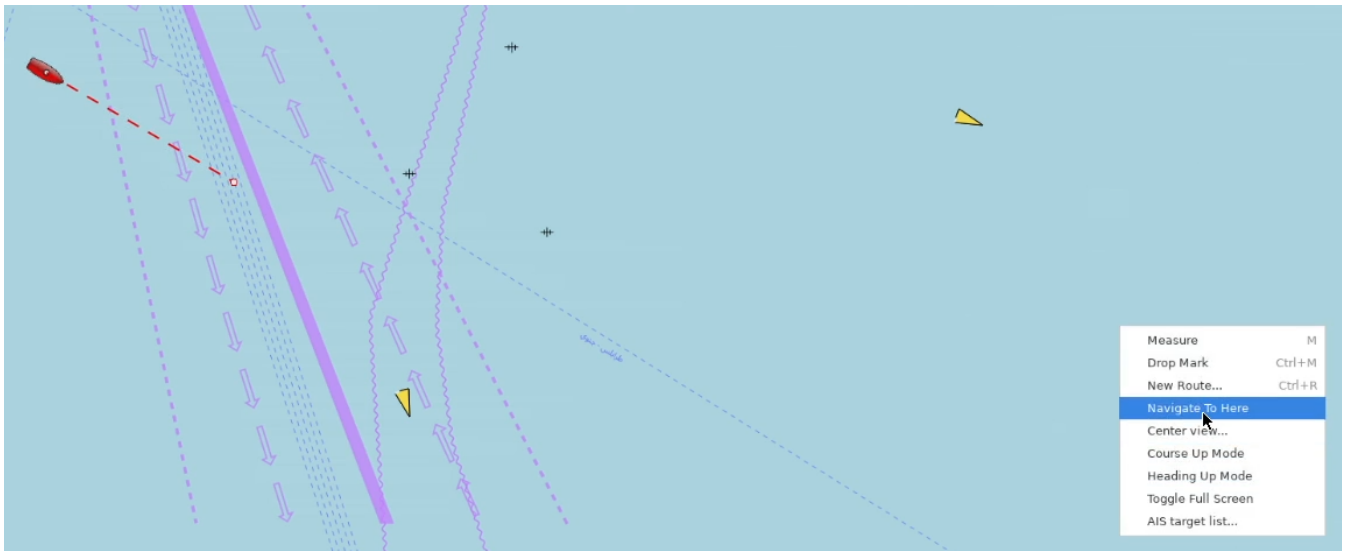


Track control

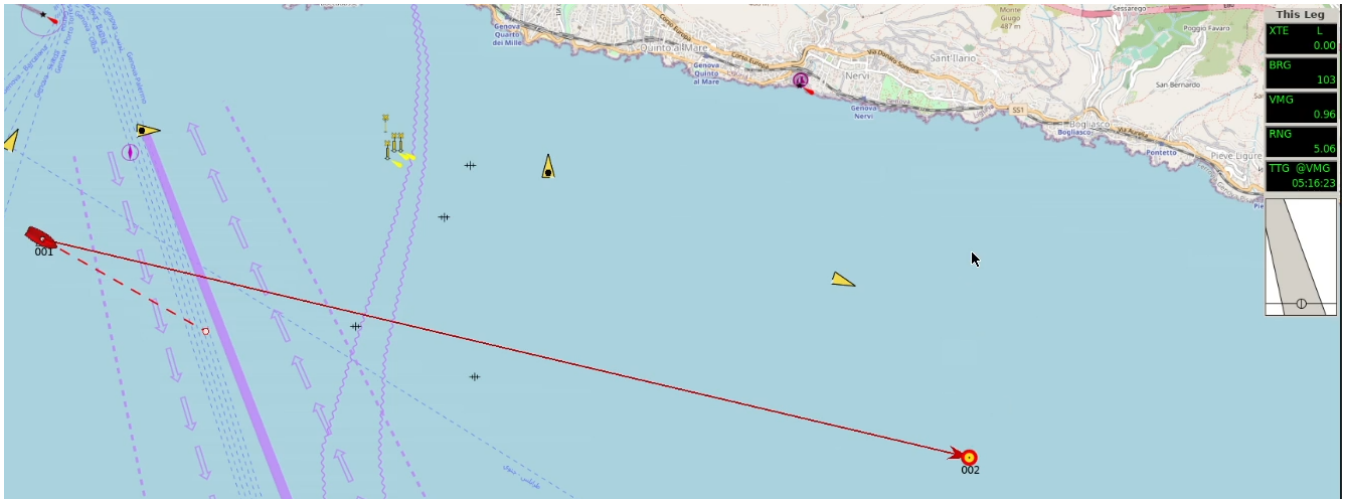
Track control is similar to heading control but instead of holding an heading set by the autopilot, it will hold a trajectory sent by some other instrument (typically an ECDIS).

In MaCySTe track control allows to automatically follow a given route without human intervention.

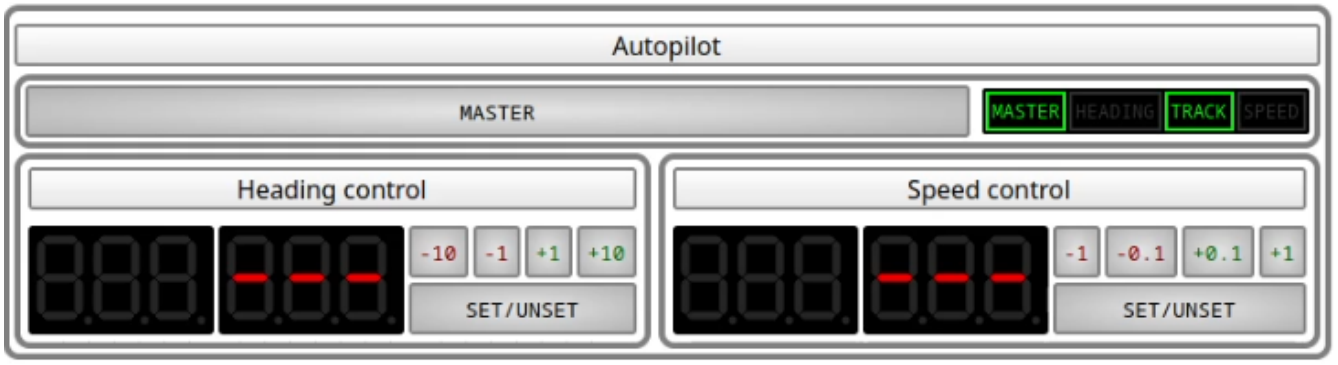
To setup a route, open the ECDIS and find a spot you desire to reach, then right click and press *Navigate To Here*



If you did everything correctly, a new window should appear indicating that OpenCPN has now set the route and is navigating towards the waypoint



In the autopilot window you will now see the track light illuminated, indicating the track control mode activation



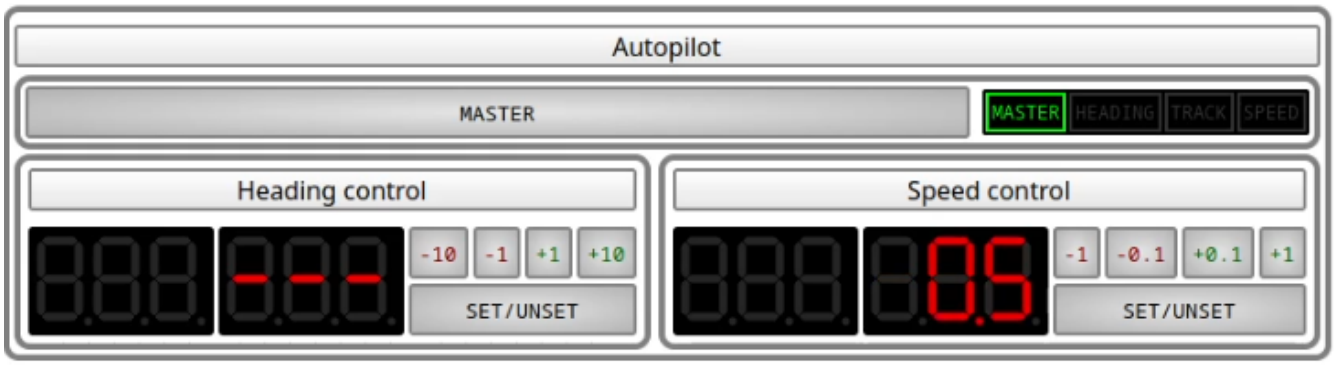
Track mode can be disabled, even when a track is available by:

- Shutting down the autopilot
- Enabling heading control

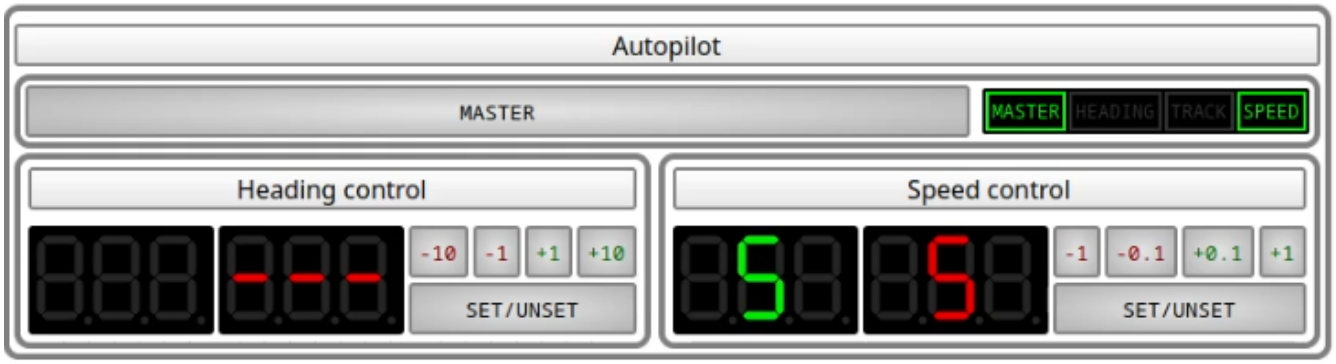
Speed control

Speed control will keep the boat at a set speed.

To use it, prepare with the buttons the speed you wish to hold.

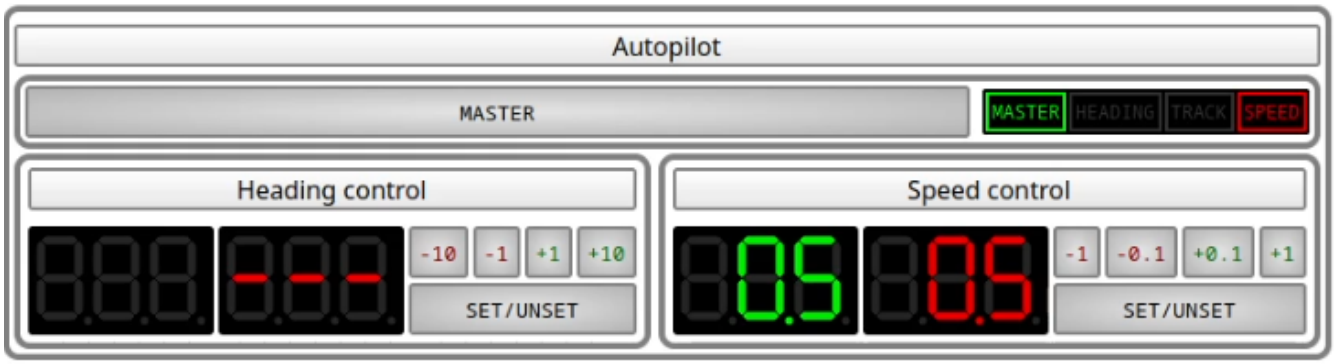


then, press the *set/unset* button



Errors

If speed control is enabled but the autopilot cannot determine the current speed, the speed control light will flash in red indicating a failure



Technical details

MaCySTe autopilot implements an extremely simple control logic known as a progressive-integral controller.

For a given setpoint x^* and error with respect to the current state $e = x^* - x$ the control action is calculated as

$$K_p \cdot e + K_i \cdot \int_t e$$

where K_p and K_i are coefficients.

This autopilot also implements two anti-windup techniques:

- The integral component is enabled only when e is small
- The integral component is reset on overshoot

The autopilot listens on the bridge network to reconstruct the ship state and its taken decisions are then actuated in the network by commanding via ModBus the master PLC of the steering gear system or the engine PLCs.

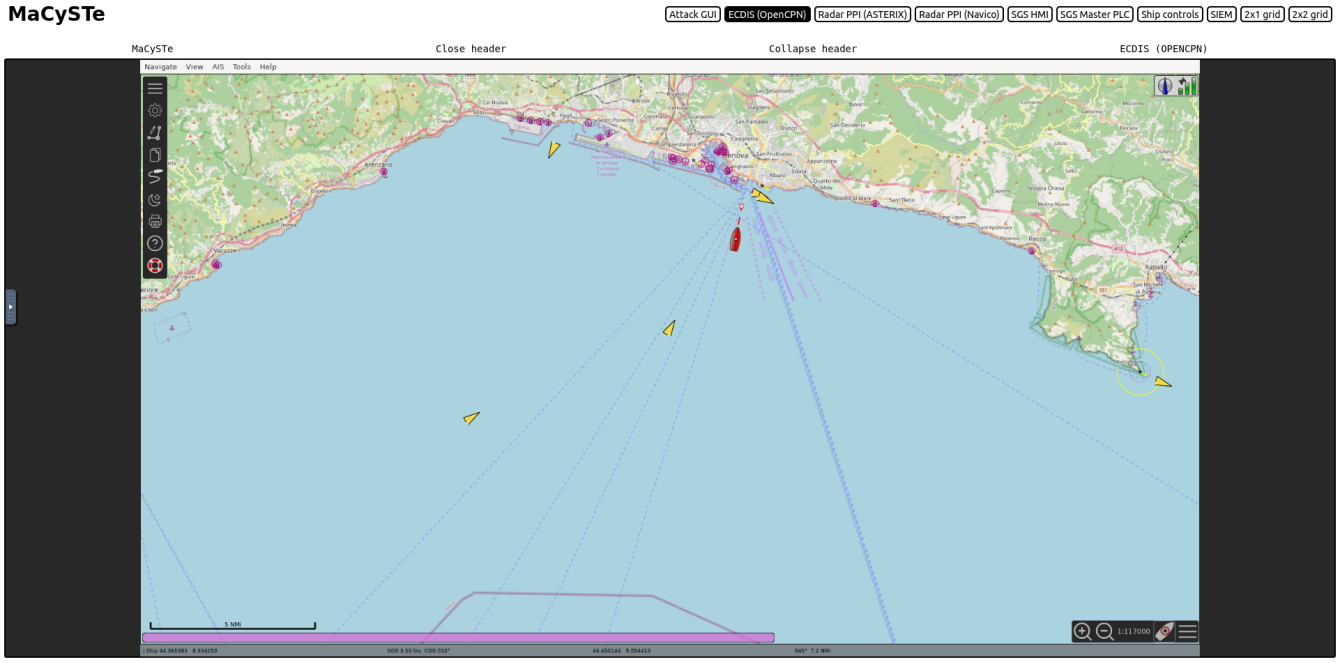
ECDIS

The Electronic Chart Display and Information System used by MaCySTe is the well-known OpenCPN project.

We invite users to refer to its documentation for more information on its usage and powerful features.

Please refer to the autopilot track control section for more informations on how to use it for waypoint to waypoint automatic navigation.

We leverage it as-is with the only addition of nautical charts for the Genoa scenario.



GUI home

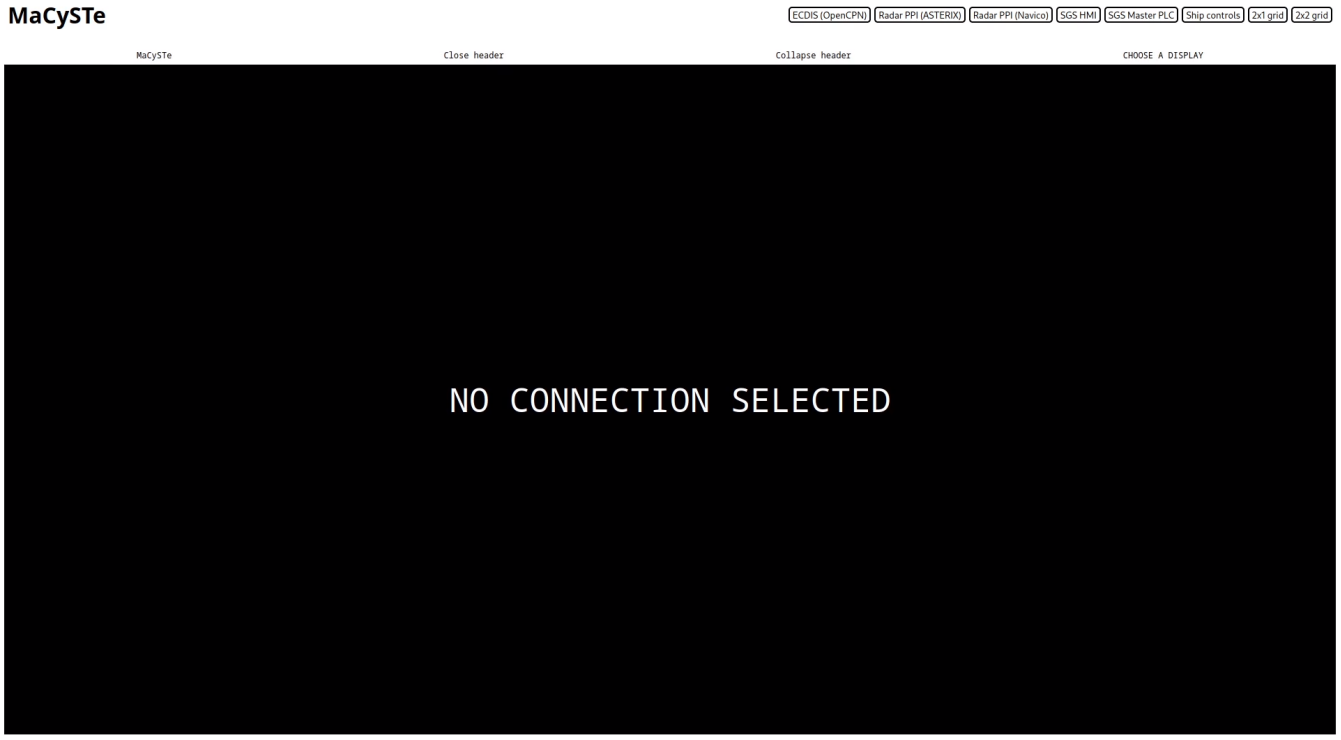
MaCySTe bundles inside of it a lot of graphical components accessible by the browser.

Given the dynamic nature of both the scenarios and their IP allocations, keeping stable identities for each of the components would have been a cumbersome process.

To streamline it, we provide a centralized hub which allows access to every interactible component in a simplified fashion.

Usage

Once opened, the GUI will download from the server a list of available visualizations to display and wait for user input



The user then can select which display to show by clicking on it.

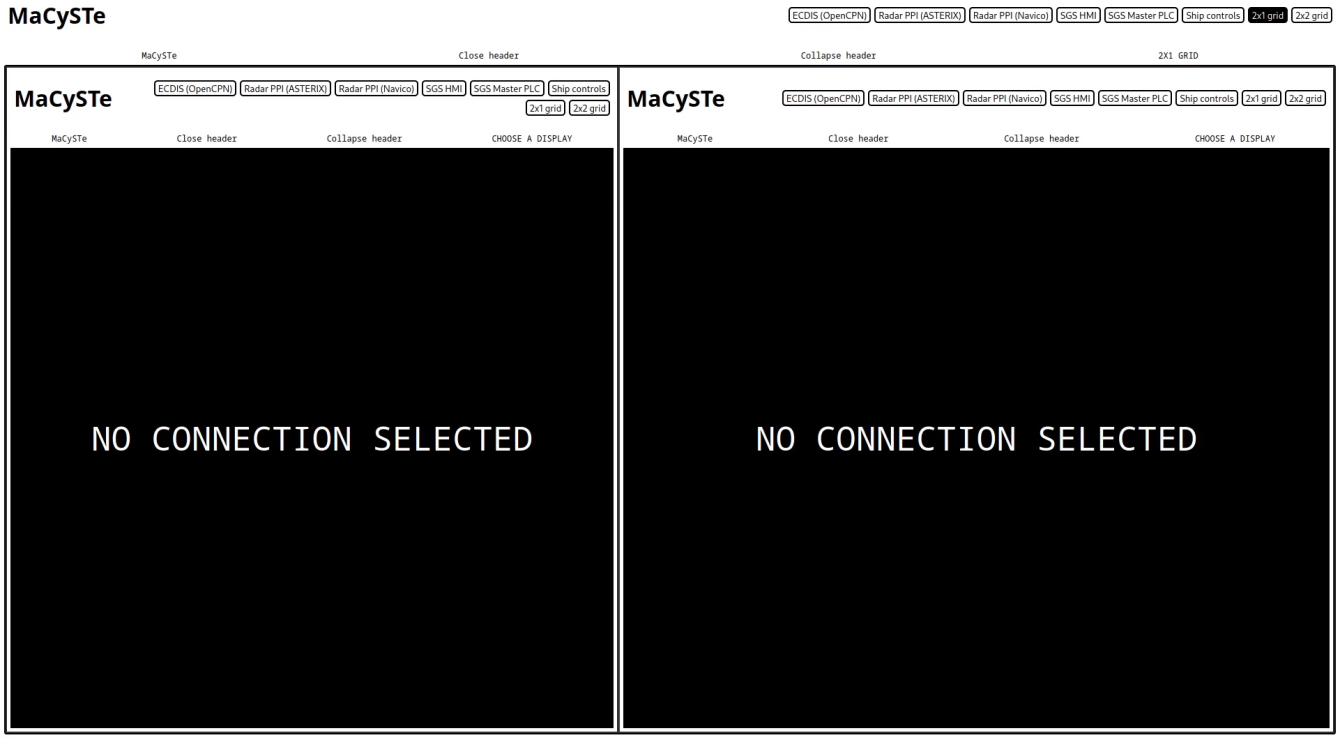
Once opened, the GUI header can be either reduced in size by clicking on *Collapse header* or entirely removed by clicking on *Close header*

2x1 grid

By clicking on the **2x1 grid** on the top of the screen, the MaCySTe GUI can be split into two resizable quadrants.

Each of the diplays inside of this side by side view can be operated as an indipendent GUI instance.

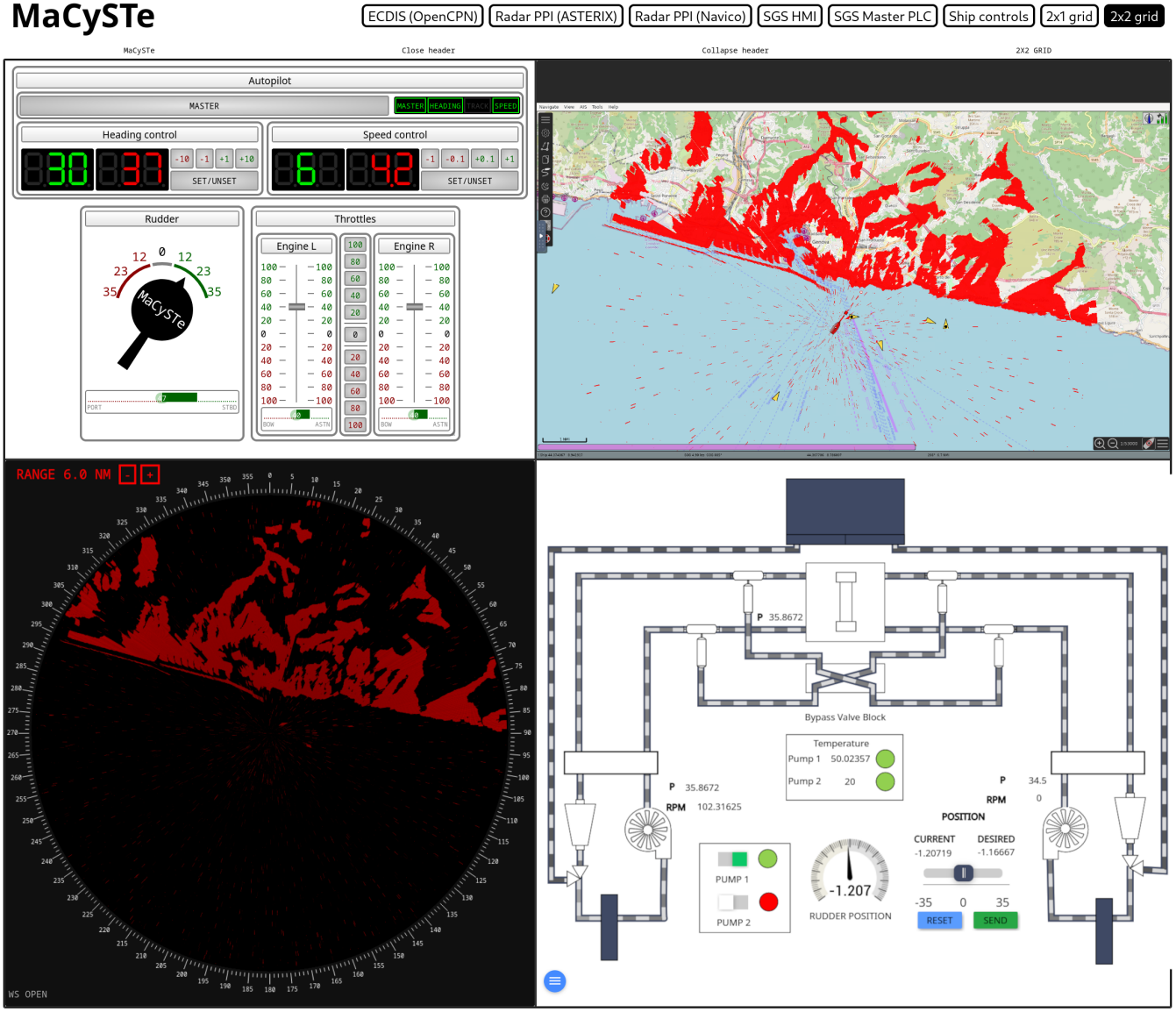
To move the center divisor click and drag it.



2x2 grid

By clicking on the **2x2 grid** button on the top of the screen, the MaCySTe GUI can be split into four resizable quadrants.

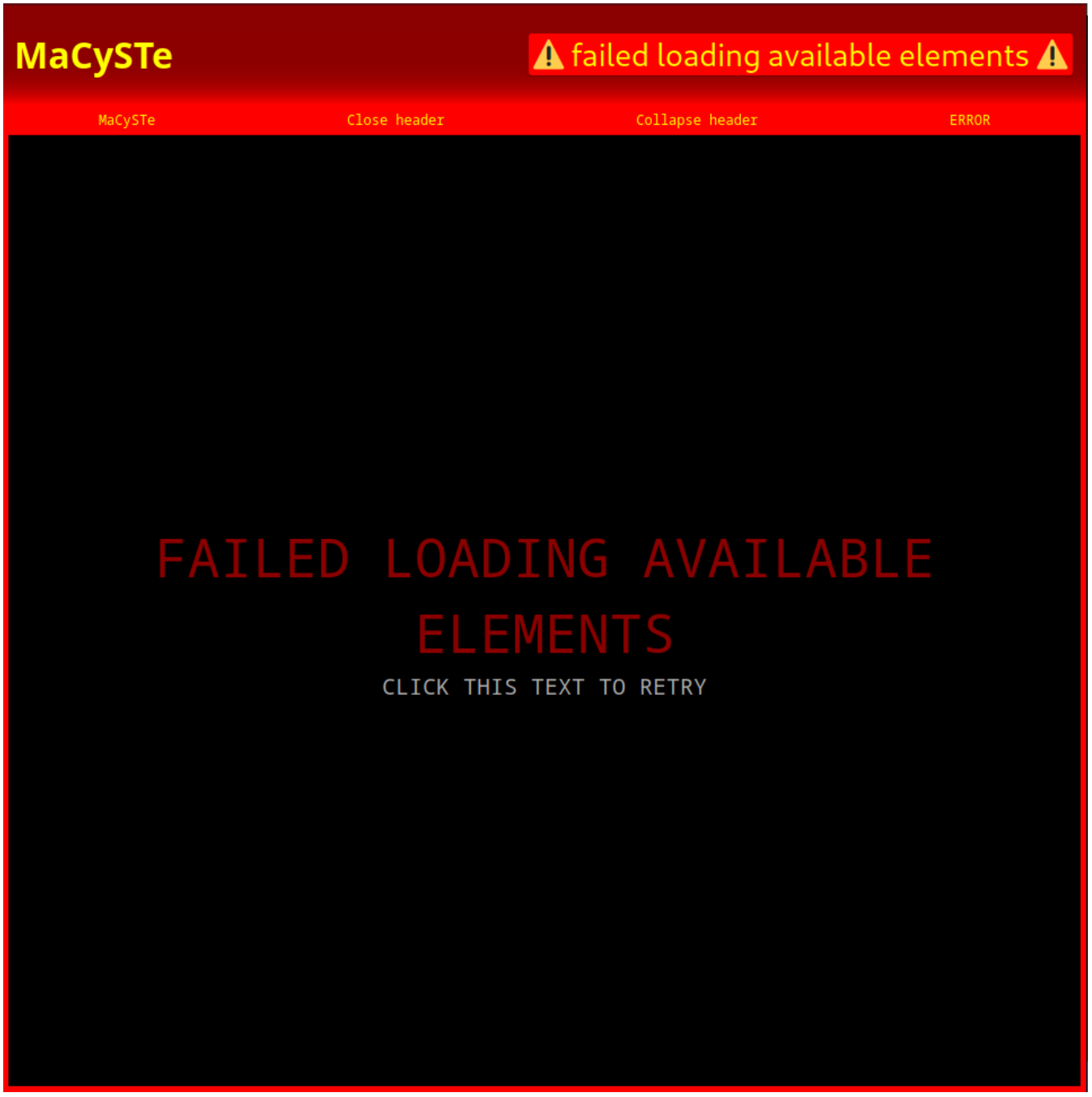
Each of the quadrants is a new MaCySTe GUI instance, allowing also deeply nested layouts.



To resize the quadrants, click and drag the frame separator.

Errors

In case of errors while loading the available GUI elements, an error interface such as the one below will be shown.



Download of available visualizations

In order to discover the available elements, the GUI will request the list of server environment variables at `/config/env`.

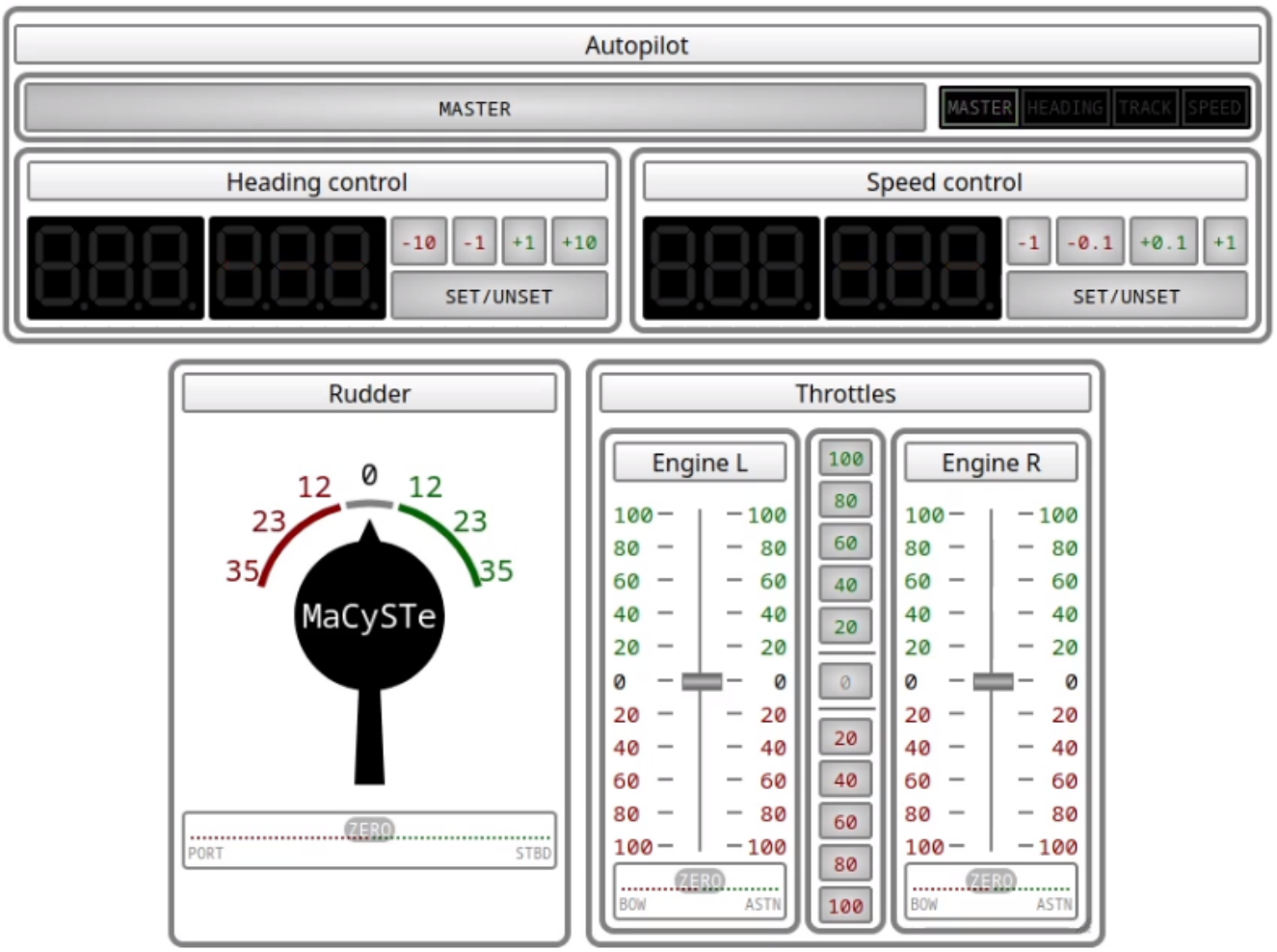
Then, for each variable named `MENU_<x>_TEXT` it will request from the server:

- The text to display on the button at `/config/env/MENU_<x>_TEXT`
- The tip to display in the interface at `/config/env/MENU_<x>_TIP`
    - The tip has to be a space separated `key=value` list which will be rendered as columns containing `key: value`
- The URL of the element at `/config/env/MENU_<x>_URL`

GUI instruments

MaCySTe bundles a in its GUI an instrument panel, allowing you to control the ship and its systems.

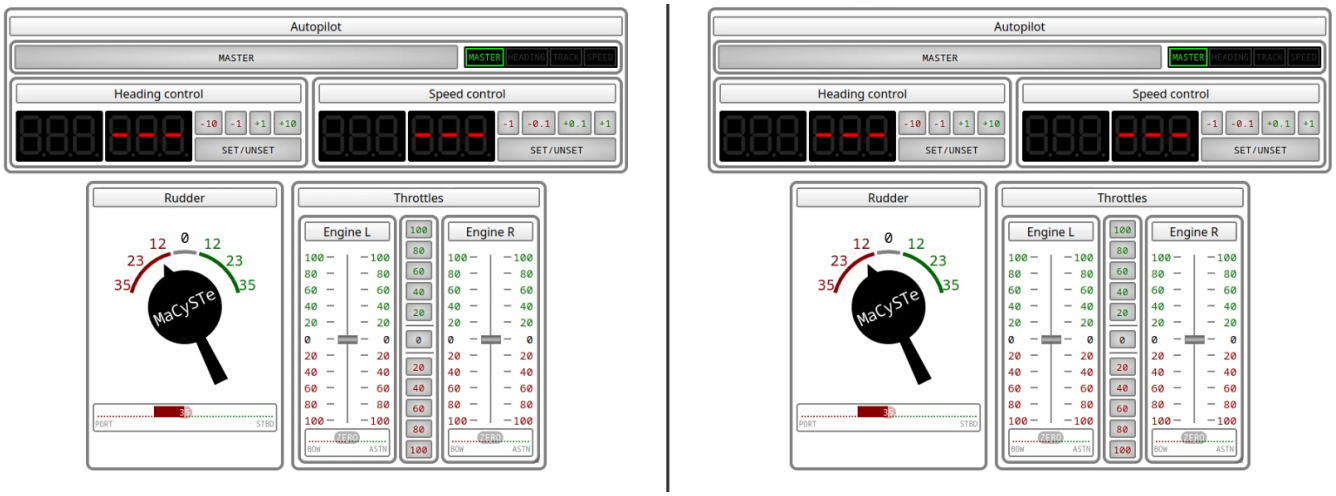
Such instrument panel can be accessed from the MaCySTe GUI home.



It bundles together three different control panels:

- An autopilot control panel
- An helm
- Two engine order telegraphs

Whenever more than one instance is open (e.g. in two separate browser windows) they will get synced together in real time, allowing you to control MaCySTe from multiple displays.



Technically, each and every move made inside of these simulated instruments is reflected inside of MaCySTe by interacting with the message queue over a WebSocket connection.

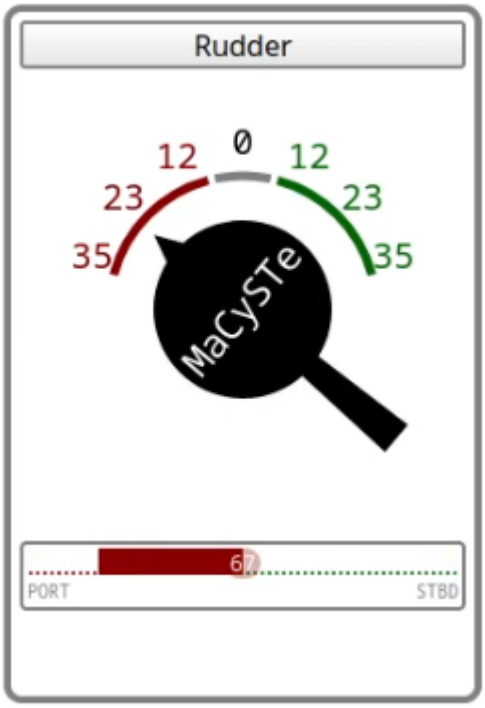
Autopilot

See its dedicated section

Helm

Within the instruments GUI of MaCySTE, there is an helm which can be acted on by either dragging it or clicking directly on the dial numbers.

Below, a colored bar will indicate the indicated rudder desired attitude.



Telegraphs

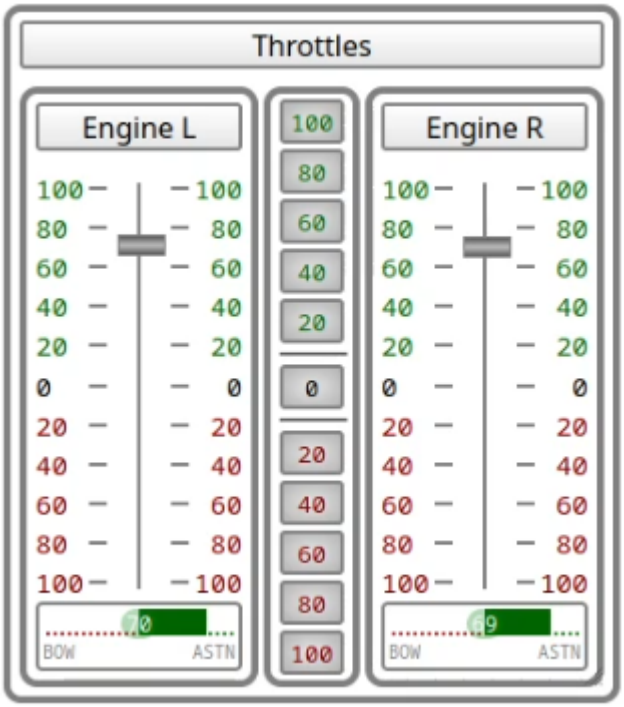
The telegraphs within the instruments GUI allow to control the engine power in MaCySTe.

Engines can either be governed indipendently or synced together.

To act on them indipendently, drag the silver handle upwards for ahead and downwards for astern. Alternatively, click on the numbers to the side of the handle to instantly set the desired output (e.g. click on the green 100 to perform a slam start).

Whenever you wish to sync the throttles, click on the center buttons to set both telegraphs at the same time.

A colored bar below each engine will indicate the current telegraph position.



Propulsion System

In MaCySTe, the propulsion system is simulated by Bridge Command.

Still, the engines appear inside of the control network by leveraging the ModBus to message queue component.

On the control network, engine telegraphs, following the motions of the GUI telegraph sets via ModBus the engine desired power values to two ModBus slaves emulating the Master PLCs of the respective engines.

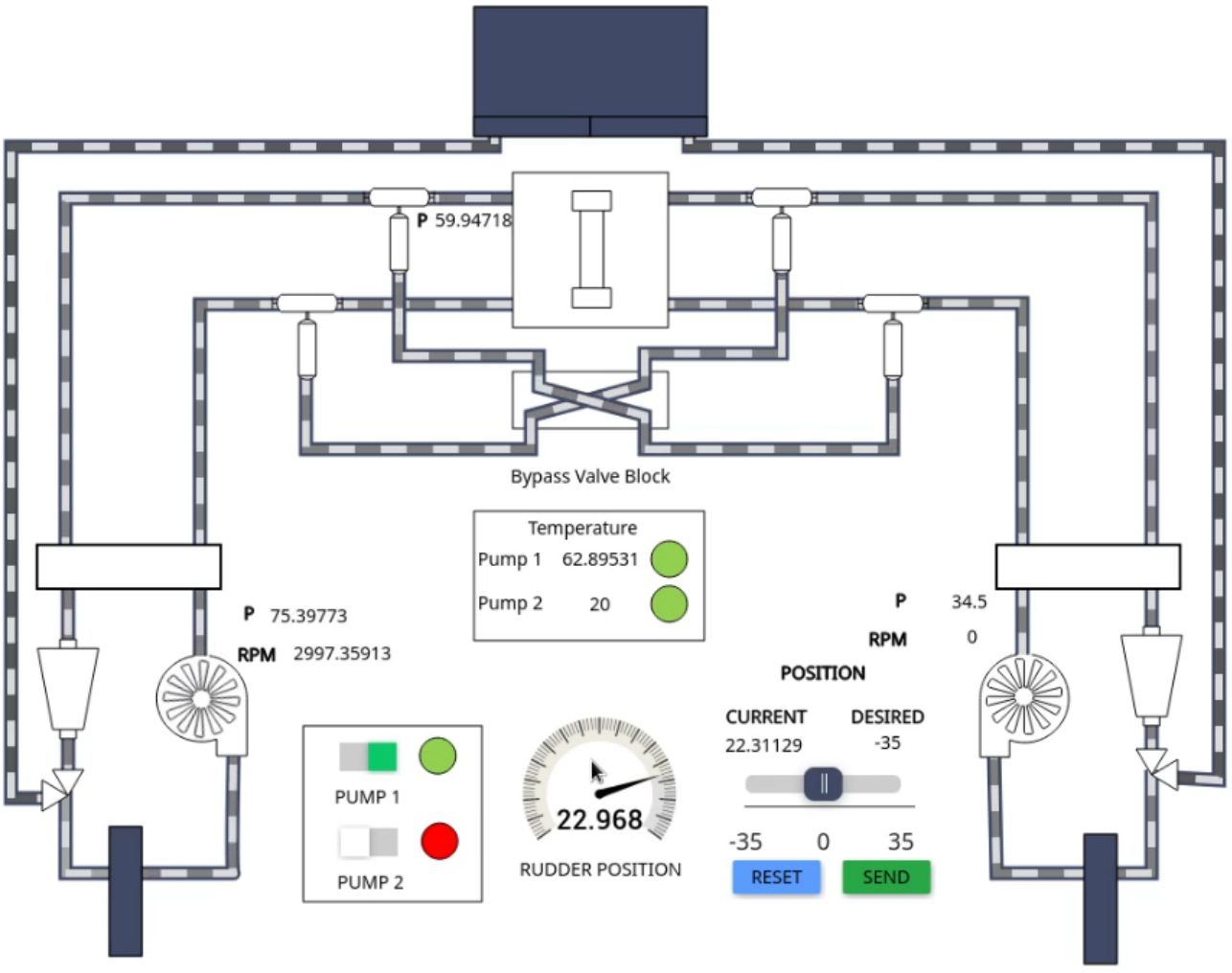
The commanded power output setpoint is then propagated via the NATS network to Bridge Command which runs the power plant simulation.

Steering Gear System

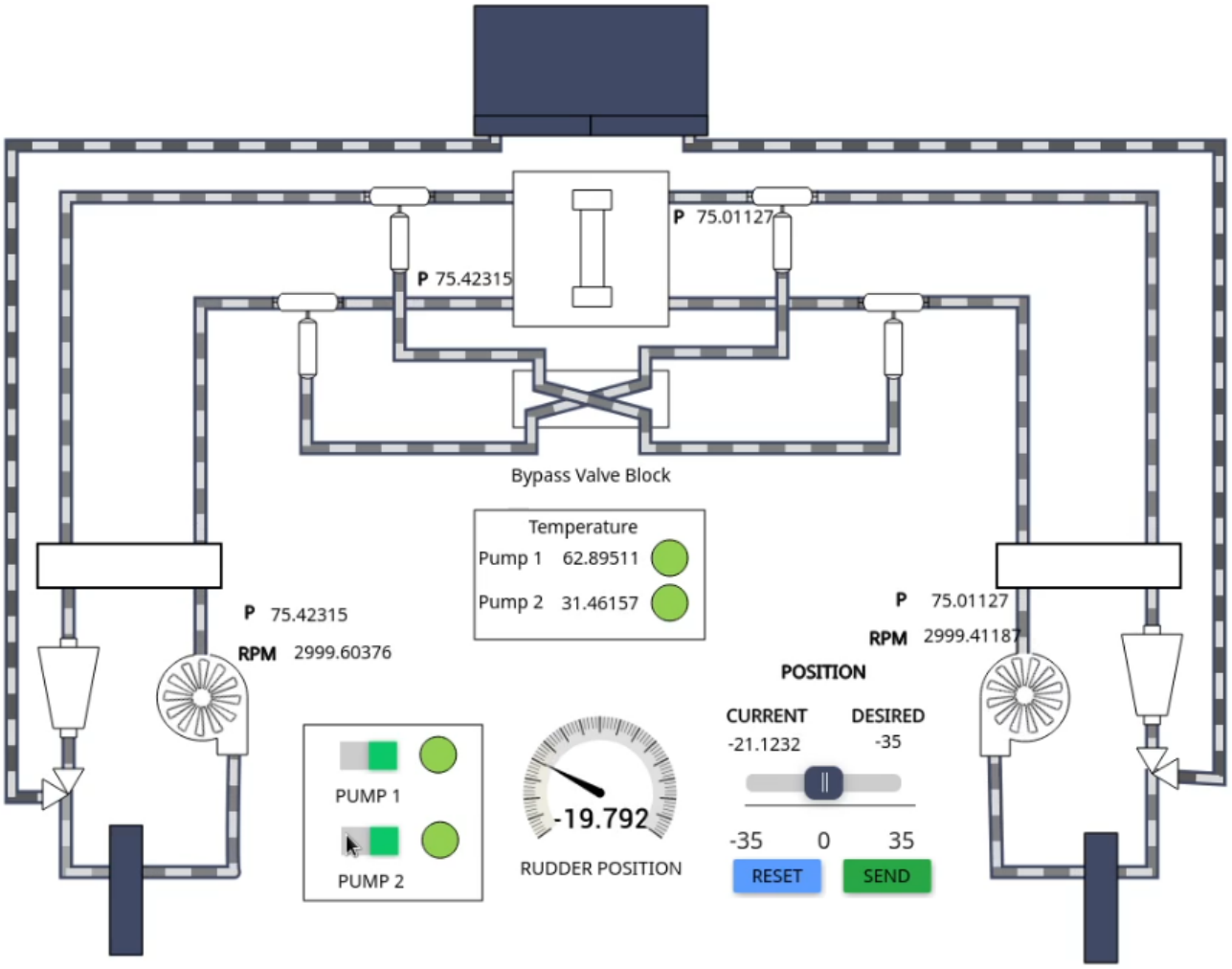
MaCySTe integrates a simulated quad ram steering gear system comprised of 6 PLCs and an external physics simulator.

Such steering gear system integrates two separate indipendent hydraulic systems operating each 1 pump, 1 oil tank, and two counteracting rams. Its structure and performance characteristics meet the standards set in the SOLAS regulations for steering gear systems.

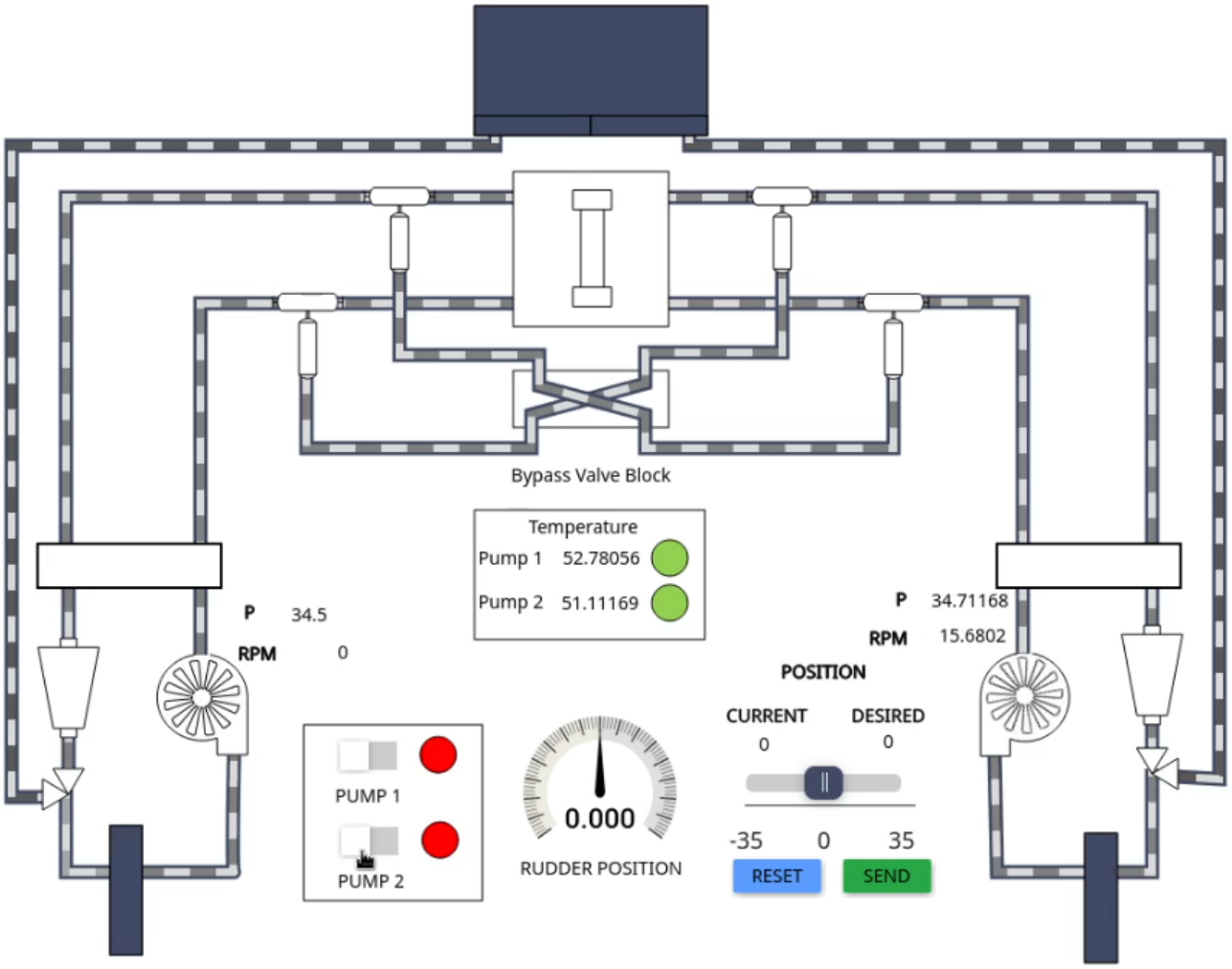
In the default configuration, the SGS runs in a single pump configuration with the master PLC receiving the desired rudder angle from the helm via ModBus and actuating the PLCs under it to actuate the steering rams.



If desired, from the *SGS HMI* of the GUI it is possible to manually dial in a new rudder position or to turn on a secondary pump



If both pumps are turned off, the rudder will be left in a neutral position, unable to move



The master PLC will also automatically start (unless it has been forced into manual mode) the secondary power unit whenever the expansion tank of the respective side activates its low level alarm in order to avoid loss of steering capability.

Please be aware that like in the real system, repeated full deflection of the rudder will overheat the hydraulic oil in the circuit, triggering alarms. To reduce overheating it might be needed to turn on both pumps during these high stress events.

RADAR Plan Position Indicators

Currently, MaCySTe supports the display of radar image for two radar protocols:

- Navico BR24
- ASTERIX CAT240

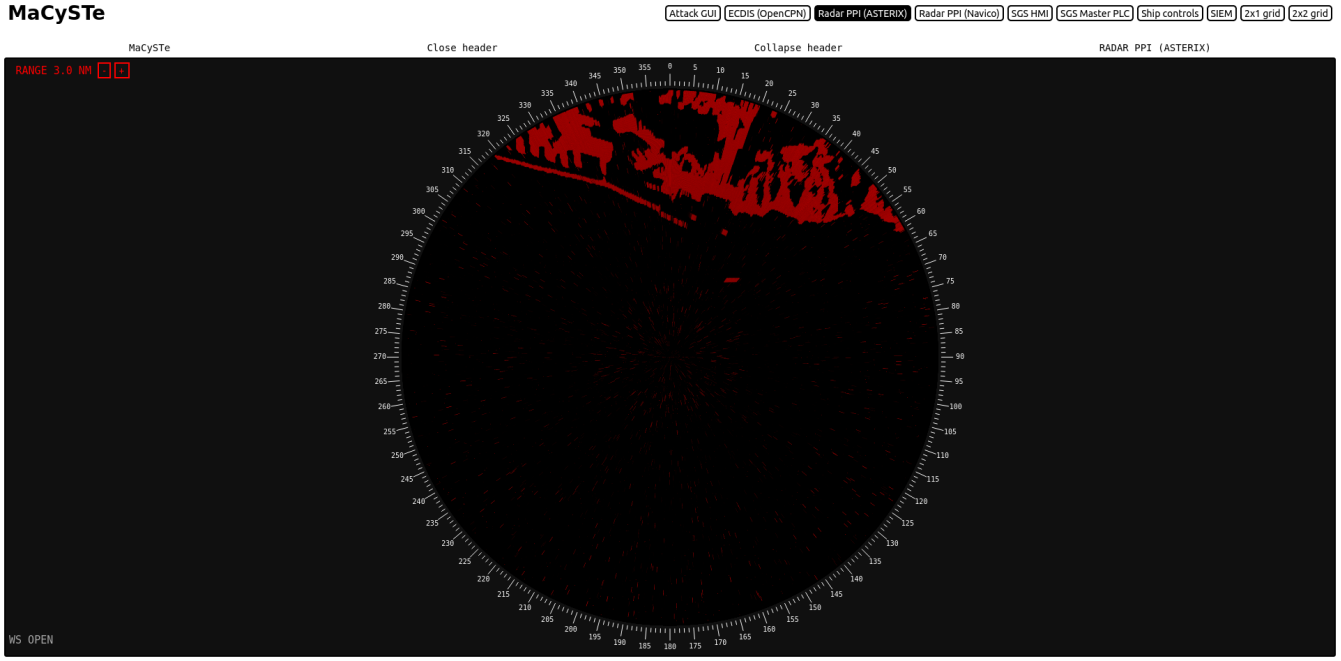
ASTERIX PPI

MaCySTe includes an asterix PPI (plan position indicator) under the form of web application.

The application listens on one websocket used to receive radar image under the form of ASTERIX CAT240 packets, and leverages NATS to publish commands to change the display radar range.

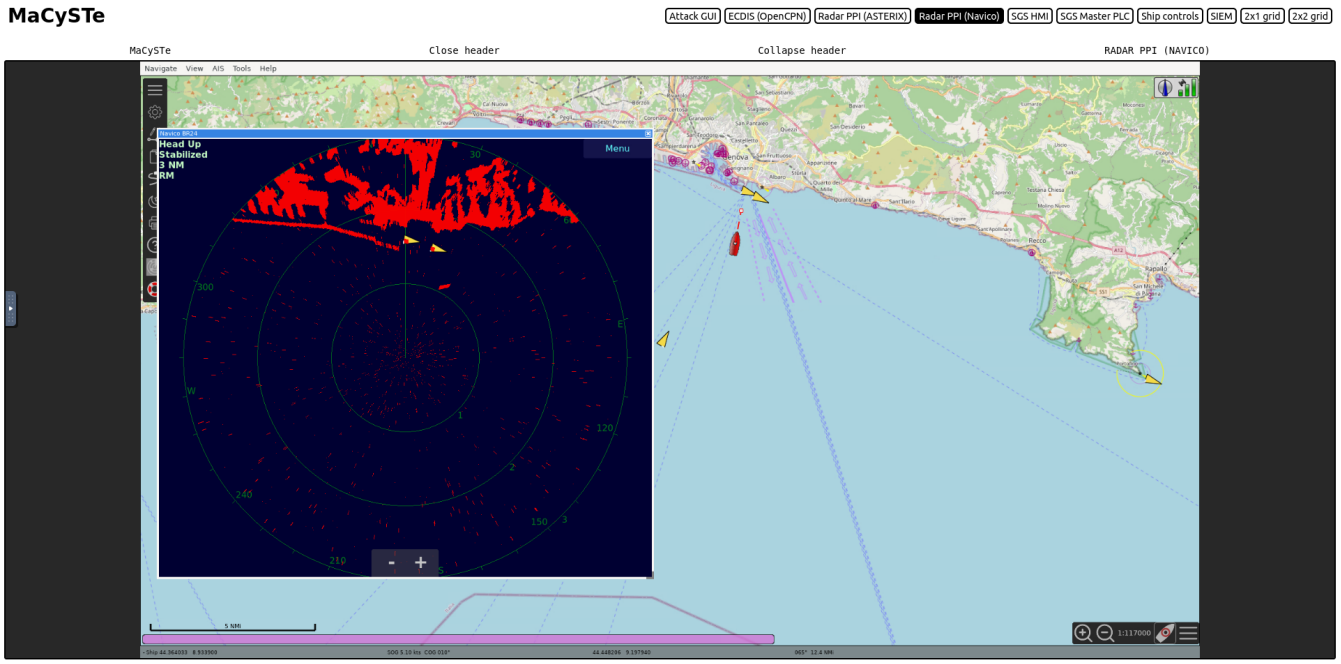
These two can be configured as environmental variables of the `spa_hoster`, namely `ASTERIX_WS_URL` and `NATS_WS_URL`.

The radar picture is displayed in north up orientation.

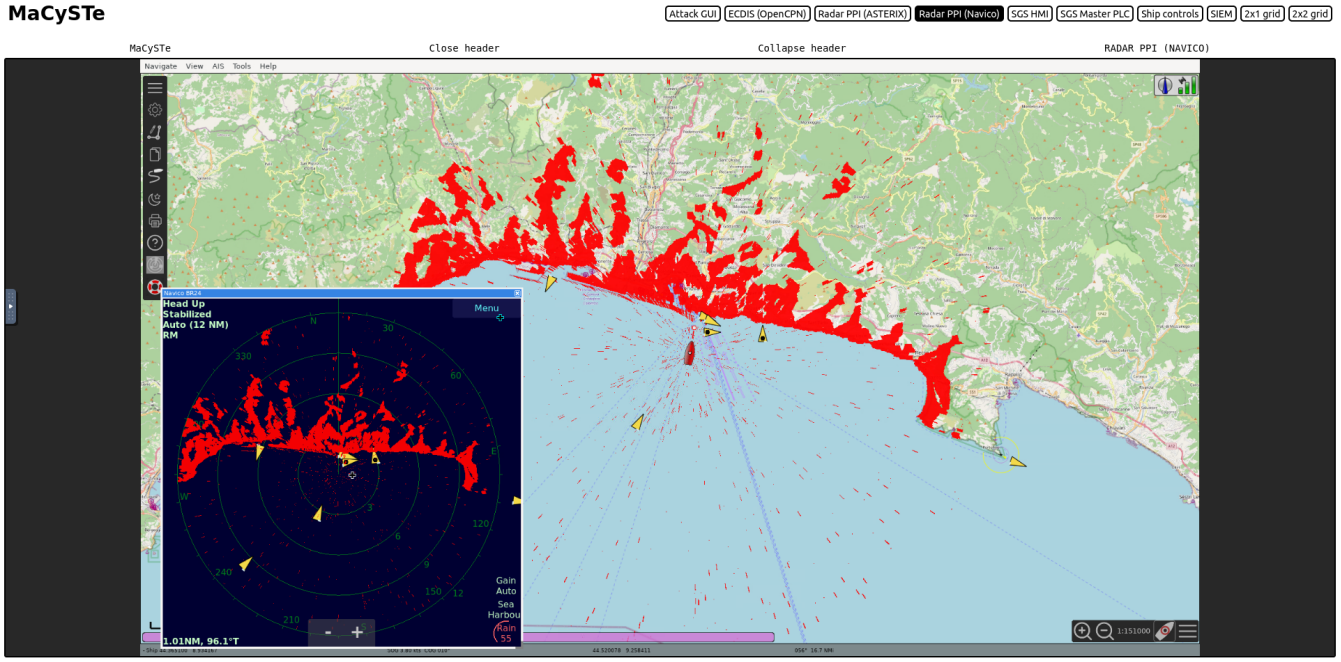


OpenCPN PPI

MaCySTe includes a PPI for Navico BR24, thanks to radar_pi - Radar Plugin for OpenCPN. The PPI supports both Head up and North up orientation and changes of the display ranges.



The plugin supports the radar overlay onto the map, to activate it press `Menu > Window > Radar Overlay On`



Toolkit elements

In this section, we describe which elements of MaCySTe are provided to assemble the scenario.

Check in the sidebar which elements are available.

Message queue

MaCySTe leverages NATS as the connecting glue for all publish subscribe traffic.

In addition, NATS is also used as a persistent Key-Value store to exchange variables and setpoints between programs.

All of the traffic to and from the message queue flows inside of a dedicated network in order not to contaminate with framework-specific data the simulated IT/OT systems' networks.

ModBus to message queue bridge

To implement some components such as the helm and the engine telegraphs a system for interfacing the message queue with the ModBus slaves is necessary and vice-versa.

To that purpose, MaCySTe implements two systems:

- A client which receives data from a message queue and subsequently writes the received setpoint to a connected ModBus slave
- A server which acts as a ModBus slave where every register write corresponds to a publication on the message queue

These two systems allow to introduce the simulator connecting glue (the message queue) commands as realistic industrial control system traffic.

In this way the client allows commands sent via the message queue to be reflected as ModBus holding register writes.

Similarly, the server allows ModBus commands to be reflected in the message queue.

INS data multiplexers and demultiplexers

The simulator outputs a stream of NMEA packets with a single network identity (i.e. IP address and source port pairing).

This situation does not accurately represent the situation on the INS network where multiple device interact together each broadcasting and receiving a specific piece of information.

To reproduce this effect in MaCySTe, we introduce some demultiplexers which spread the incoming NMEA data flow across multiple devices, simulating a distributed bridge.

They are not spread out at random, instead, each demultiplexer takes the role of a specific sensor, forwarding only messages coherent with its identity.

For instance, a gyrocompass will only forward sentences indicating `HE` as the talker.

Also included (but not used) in MaCySTe, we provide the reverse multiplexer system, allowing to receive NMEA data from a single endpoint and spread it to multiple demultiplexers.

Such a structure is achieved by forwarding the NMEA sentences over to the message queue and leveraging its topics system for distributing the messages to the correct demultiplexers.

RADAR signal generator

MaCySTe ships with a component named `radar-converter` that is in charge of simulating the behaviour of a RADAR antenna.

Currently, two types of antennas are supported, a Navico Broadband Radar (BR24) or an antenna trasmitting using ASTERIX CAT240 protocol.

The components takes as input the image matrix generated by Bridge Command and exported to a NATS Key-Value store.
Then, according to the type of antenna, it encodes the image into network packets compliant to the protocol, and sends them through a multicast udp socket.

Single page application hoster

Every web application of MaCySTe is hosted via the `spa-hoster` component. This component serves static files under a directory specified with the environment variable `STATIC_FILES_PATH`.
It can be binded on a specific port thanks to `BIND_PORT` environment variable.
`spa-hoster` exposes environment variables at path `/config/env/<ENV_NAME>` **BEWARE** that this may not be safe because it exposes all the env, but you can specify which variables to serve via a space separated list in another environment variable named `ENV_WHITELIST`.

Software PLCs

MaCySTe leverages the OpenPLC runtime for its PLCs, they communicate via ModBus and run programs written in ladder logic which are subsequently transformed into IEC 61131-3 structured text instructions.

We invite the users to visit the OpenPLC documentation to learn more.

Our OpenPLC container allows the user to preseed the program and slave device configuration, and to replace environment variables present inside of these configuration files.

Implementation of 32-bit floating point values

Given that ModBus supports holding registers keeping at most 16-bit values, the floating point numbers exchanged in MaCySTe are written and read by splitting a 32-bit float into two successive 16-bit registers.

UDP to websocket connector

This component has the task of reading traffic from an UDP socket and transferring it to a WebSocket.

Usage:

- Set in environment `PROXIES` to a space separated list of `path:host:port` places to listen on.
- Optionally specify `BIND_PORT` (default is `9090`)

The connector will then send traffic received on `host:port` to either `ws://<own_address>/<path>/BINARY` or `ws://<own_address>/<path>/TEXT` dependending on the type of data.

Attacker addon

This addon is available only in the attacker and attacker_siem scenarios

MaCySTe allows to experiment with the cyber attacks against the ship INS.

To that purpose we introduce the attacker addon bundling some components to allow for this kind of trials.

In particular, we introduce a scenario in which an attacker has installed a malware inside of the INS network.

Since the ship firewall would probably not allow any connection from the outside, the installed malware tries to reach outside on its own. This technique is called *reverse shell*. The endpoint which has to be reached is a command and control (C&C) server located on a simulated internet (which has as its CIDR the IANA reserved block for documentation and examples `198.51.0.0/16`)

The malware leverages the WebSocket protocol in order to:

- Keep a long lived connection open to the server
- Appear as a connection to the normal HTTP port to external observers
- Allow real-time streaming of data

Once connected to the C&C server, the attacker can then connect his control panel to the malware by using the C&C as reverse proxy to the malware.

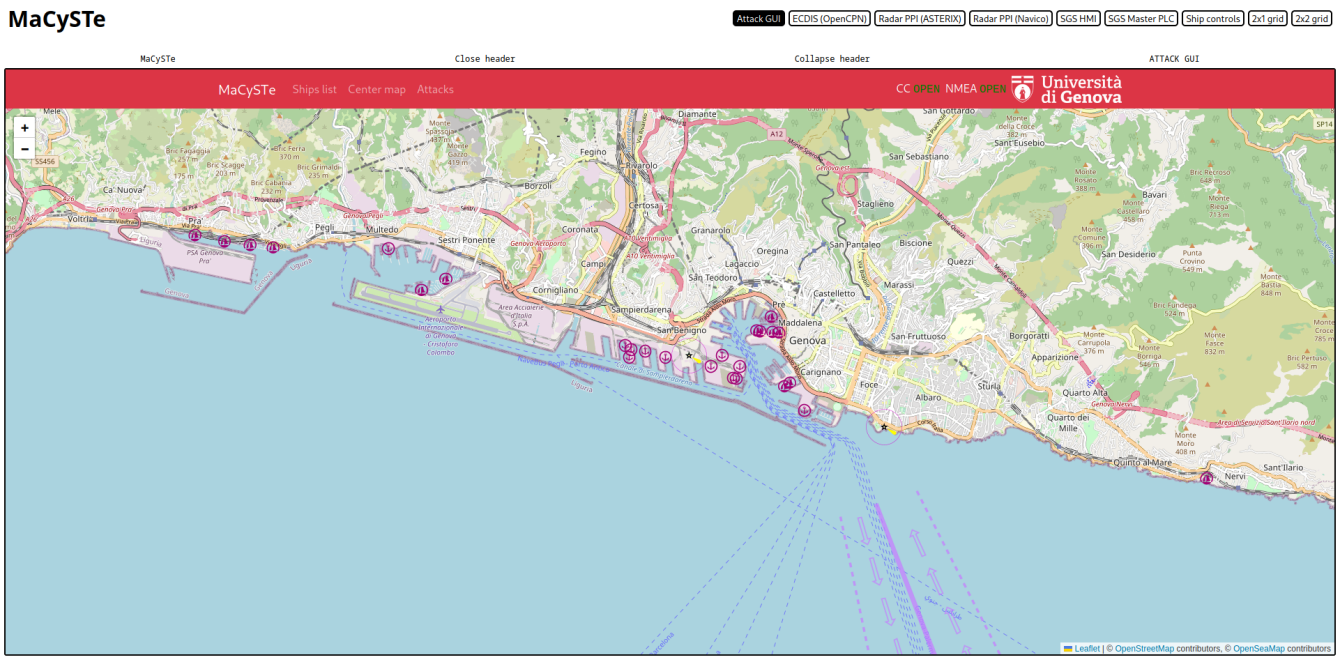
This addon is comprised of the following components:

- An attacker graphical user interface to launch and coordinate attacks
- A persistent malware installed on the bridge
- A satellite router allowing the ship to reach external resources within a simulated internet
- A command and control server that allows the attacker to rendezvous with the malware

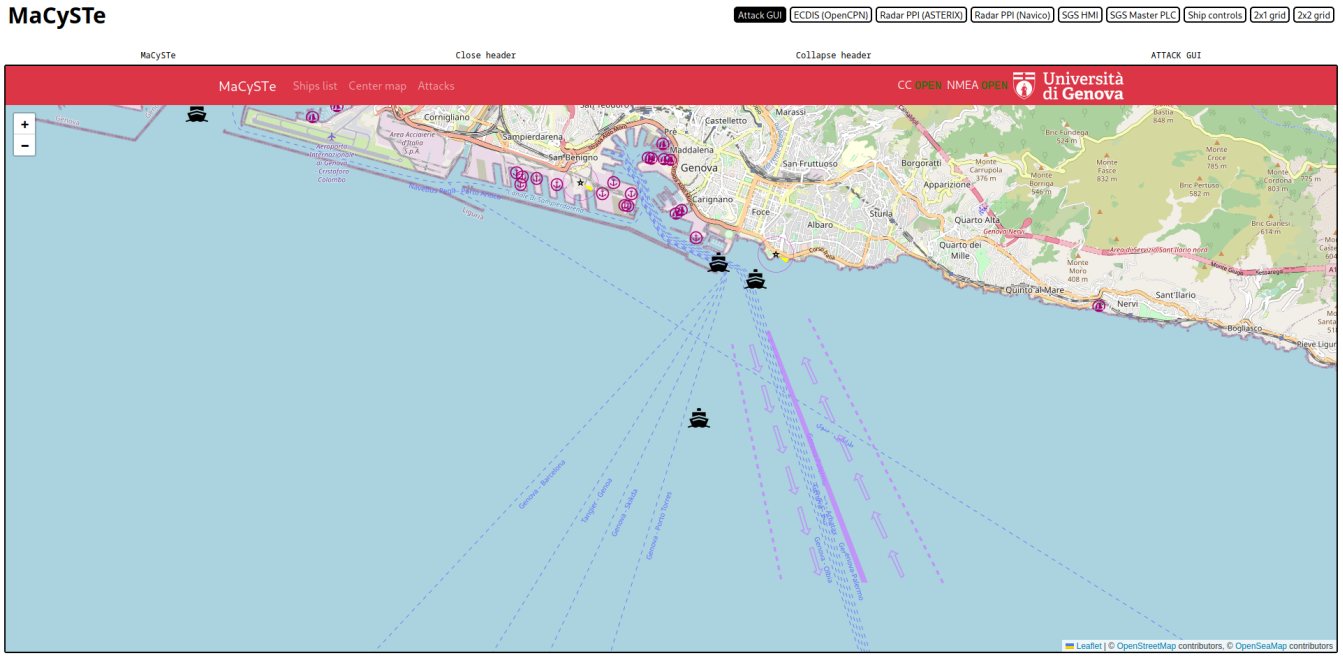
Attack GUI

MaCySTe includes a powerful and modular attack GUI, allowing the user to perform attacks in a point and click fashion.

To access it, from the GUI home click on *Attack GUI* and you will be presented with the interface

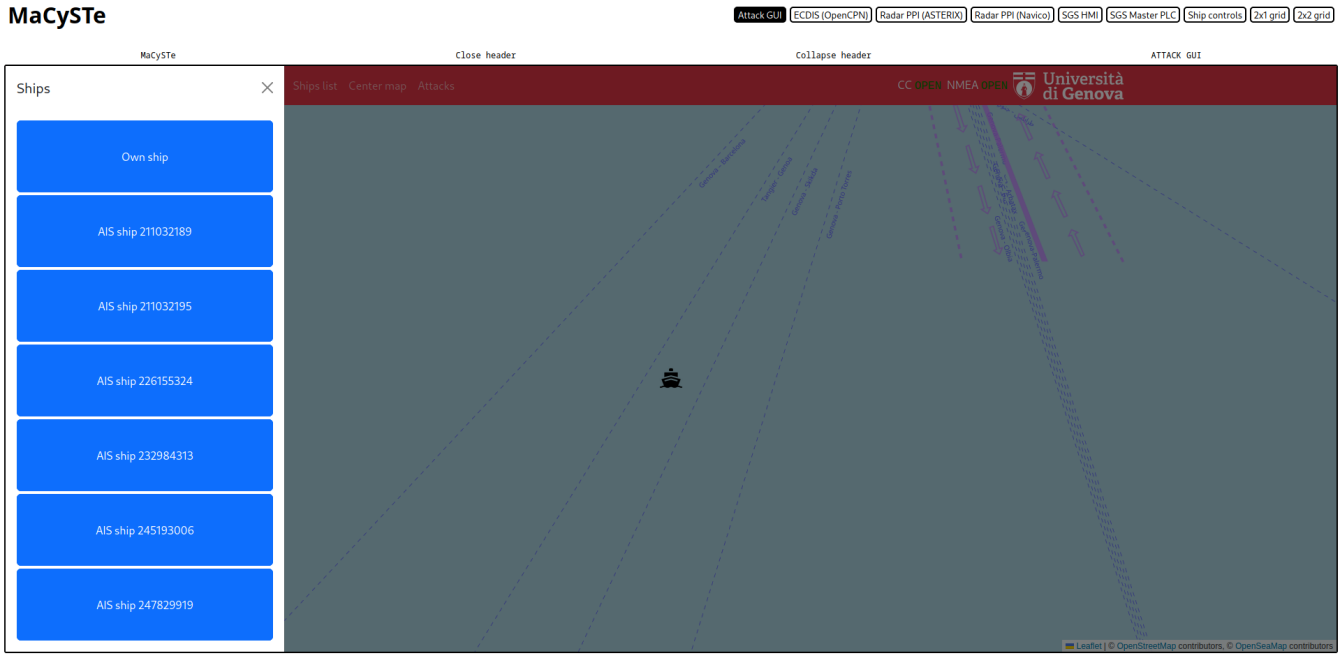


If the connection to the command and control server is established (indicated by the two green indicators in the top-right corner) you will soon see ship shapes starting to appear on the map

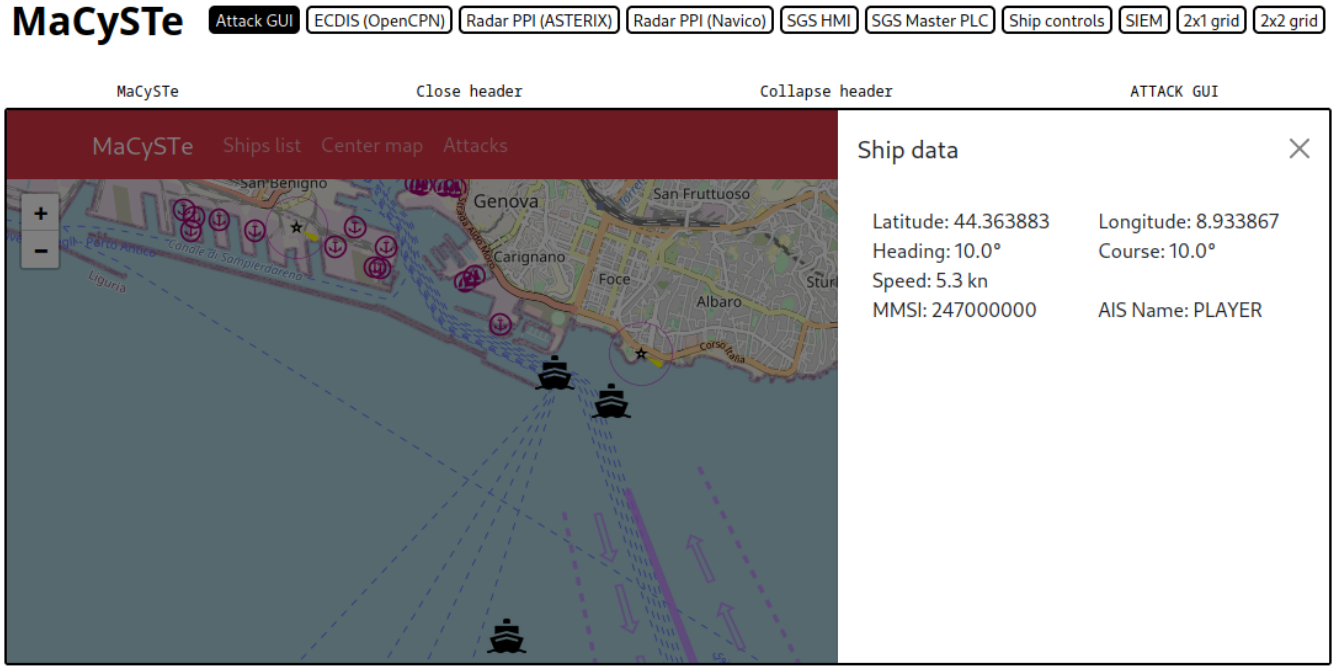


From this view, you can click on *Ships list* to see which ships have been found, click on the

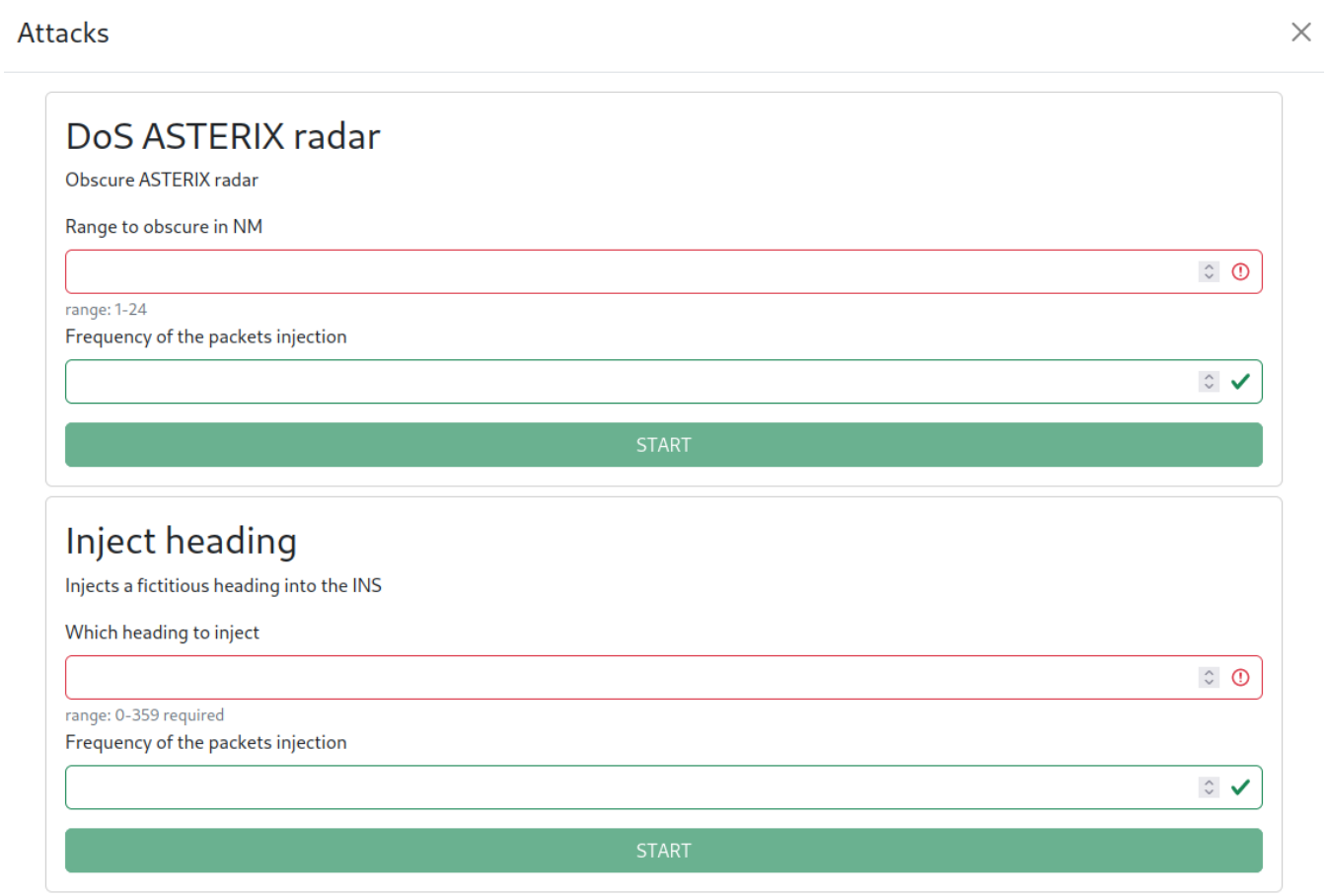
button to center the map on it



Click on a ship icon on the map to see its reported information



Click on *Attacks* to bring up the attacks list and run them



Automatic GUI generation

The attacks interface is actually dynamically generated by asking the malware via RPC for its available attacks.

The `attack_inventory` call will generate a response containing elements like

```
{
    "name": "dos_radar",
    "ui_name": "DoS ASTERIX radar",
    "description": "Obscure ASTERIX radar",
    "parameters": [
        {
            "name": "range_nm",
            "description": "Range to obscure in NM",
            "required": false,
            "type": "number",
            "default": 12,
            "min": 1,
            "max": 24
        },
        {
            "name": "injection_hz",
            "description": "Frequency of the packets injection",
            "required": false,
            "type": "number",
            "default": 1
        }
    ]
}
```

Each of the parameters shown will be automatically converted into GUI elements and client-side validated, allowing you to easily add your own attacks simply by adding them to the malware with the GUI taken care of automatically.

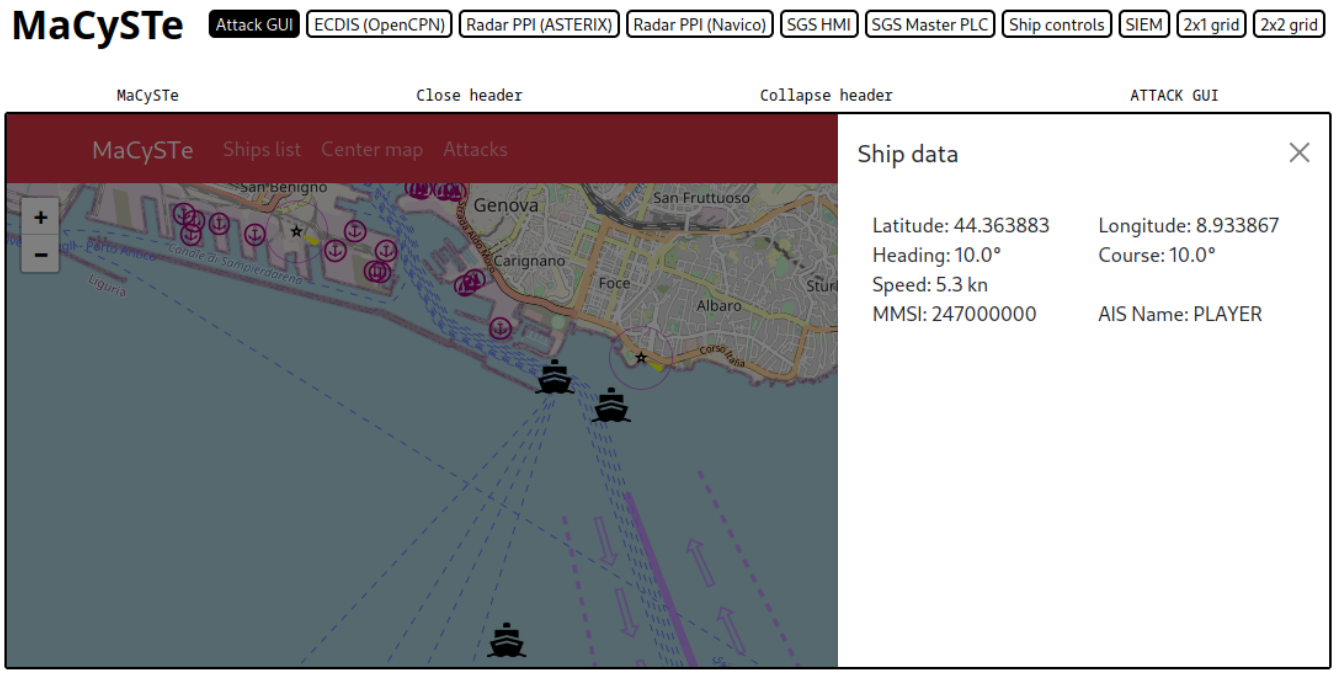
Currently only a numeric widget is made available but in future releases of MaCySTe we will add more specific panel elements.

Malware

MaCySTe integrates as part of the attacker addon a persistent malware installed inside of the INS Network.

This malware will try to connect to his preset command and control server via WebSocket and then:

- Overhear, parse, and send in structured format every received NMEA sentence allowing the attacker to reconstruct the ship state



- Expose a JSON-RPC based interface for starting and stopping attacks

The malware is structured as an extensible platform, allowing users to extend it to add their own attacks with automatic GUI integration.

Attack JSON-RPC API

The malware will listen for JSON-RPC messages of the following format

```
{
  "method": "<method-name>",
  "params": [ "<param-1>", "<param-2>", "<param-n>" ],
  "id": "<request-id>"
}
```

This example is not exactly spec compliant (it's missing the jsonrpc field)

attack_inventory method

This method allows to dynamically gather a list of available attacks

Sample request

```
{
  "method": "attack_inventory"
}
```

Sample response

```
{
    "jsonrpc": "2.0",
    "result": [
        {
            "name": "inject_heading",
            "ui_name": "Inject heading",
            "description": "Injects a fictitious heading into the INS",
            "parameters": [
                {
                    "name": "heading_to_inject",
                    "description": "Which heading to inject",
                    "required": true,
                    "type": "number",
                    "min": 0,
                    "max": 359
                },
                {
                    "name": "injection_hz",
                    "description": "Frequency of the packets injection",
                    "required": false,
                    "type": "number",
                    "default": 1
                }
            ]
        },
        {
            "name": "dos_radar",
            "ui_name": "DoS ASTERIX radar",
            "description": "Obscure ASTERIX radar",
            "parameters": [
                {
                    "name": "range_nm",
                    "description": "Range to obscure in NM",
                    "required": false,
                    "type": "number",
                    "default": 12,
                    "min": 1,
                    "max": 24
                },
                {
                    "name": "injection_hz",
                    "description": "Frequency of the packets injection",
                    "required": false,
                    "type": "number",
                    "default": 1
                }
            ]
        }
    ],
    "id": ":r0:"
}
```


attack_start method

This method allows to start an attack

Example request

```
{
    "id": ":r1:",
    "method": "attack_start",
    "params": [
        "dos_radar",
        6
    ]
}
```

Example response

```
{
    "jsonrpc": "2.0",
    "result": {
        "name": "dos_radar",
        "params": [
            6
        ],
        "running": true
    },
    "id": ":r1:"
}
```

attack_state method

This method allows to check the running state of an attack

Example request

```
{
    "method": "attack_state",
    "params": [
        "dos_radar"
    ],
    "id": ":r1:"
}
```

Example response

```
{
    "jsonrpc": "2.0",
    "result": {
        "name": "dos_radar",
        "params": [],
        "running": false
    },
    "id": ":r1:"
}
```


attack_stop method

This method allows to stop an attack

Sample request


```
{
    "id": ":r1:",
    "method": "attack_stop",
    "params": [
        "dos_radar"
    ]
}
```


Sample response


```
{
    "jsonrpc": "2.0",
    "result": {
        "name": "dos_radar",
        "params": [],
        "running": false
    },
    "id": ":r1:"
}
```


Adding your attacks

Your attacks can be easily be implemented by extending the `Attack` class specifying which elements have to be set in the GUI and adding them to the `available_attacks` array inside of the source code.

Each attack added in such a way will be automatically rendered inside of the attack GUI with automated form validation and status reports, see the dedicated section for more

details.

Mini router

Mini router implements one of two functionalities:

- In `router` mode it sets up a source network address translation (SNAT) allowing traffic coming from one network to reach out by reusing the router IP (as in the case of most satellite internet terminals used by boats)
- In `init-container` mode it allows to customize a container default gateway, overriding the preset default

Websocket to Websocket proxy

Websocket to Websocket proxy acts as the rendezvous server for the attacker and the malware, its usage is simple: each client can open a WebSocket by connecting to the server at a given path `/<path>`.

Every client connected to the same `<path>` will receive every message sent for the same `<path>`.

Every client connected to the same `<path>` can also send a message to everyone else subscribed to the same `<path>` simply by sending the message to the websocket to websocket proxy.

SIEM addon

This addon is available only in the siem and attacker_siem scenarios

MaCySTe integrates provisions for generating and studying the artifacts generated from a running ship bridge.

In particular, the MaCySTe SIEM addon augments the core MaCySTe functions with probes for acquiring informations and a SIEM for displaying and processing them.

The addon includes the following components:

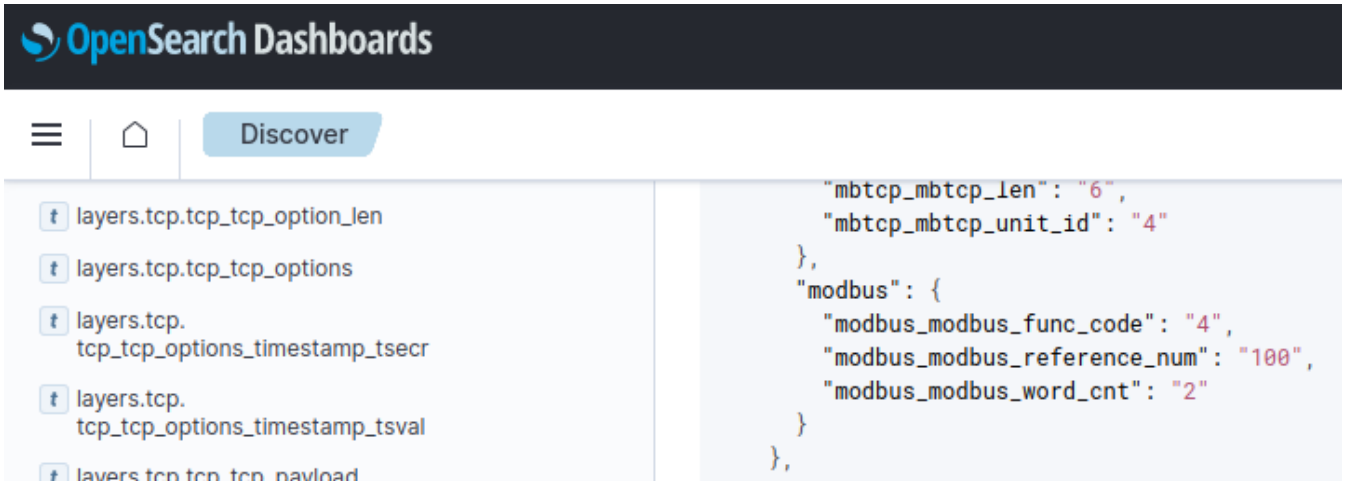
- A ModBus probe
- A NMEA probe
- A full text search database

ModBus probe

The ModBus probe installed by the SIEM addon takes place as a container colocated with the PLC listening to the network interface traffic.

This architecture resembles traditional Intrusion Detection Systems way of installation in which the IDS listens for every packet belonging to a particular network card.

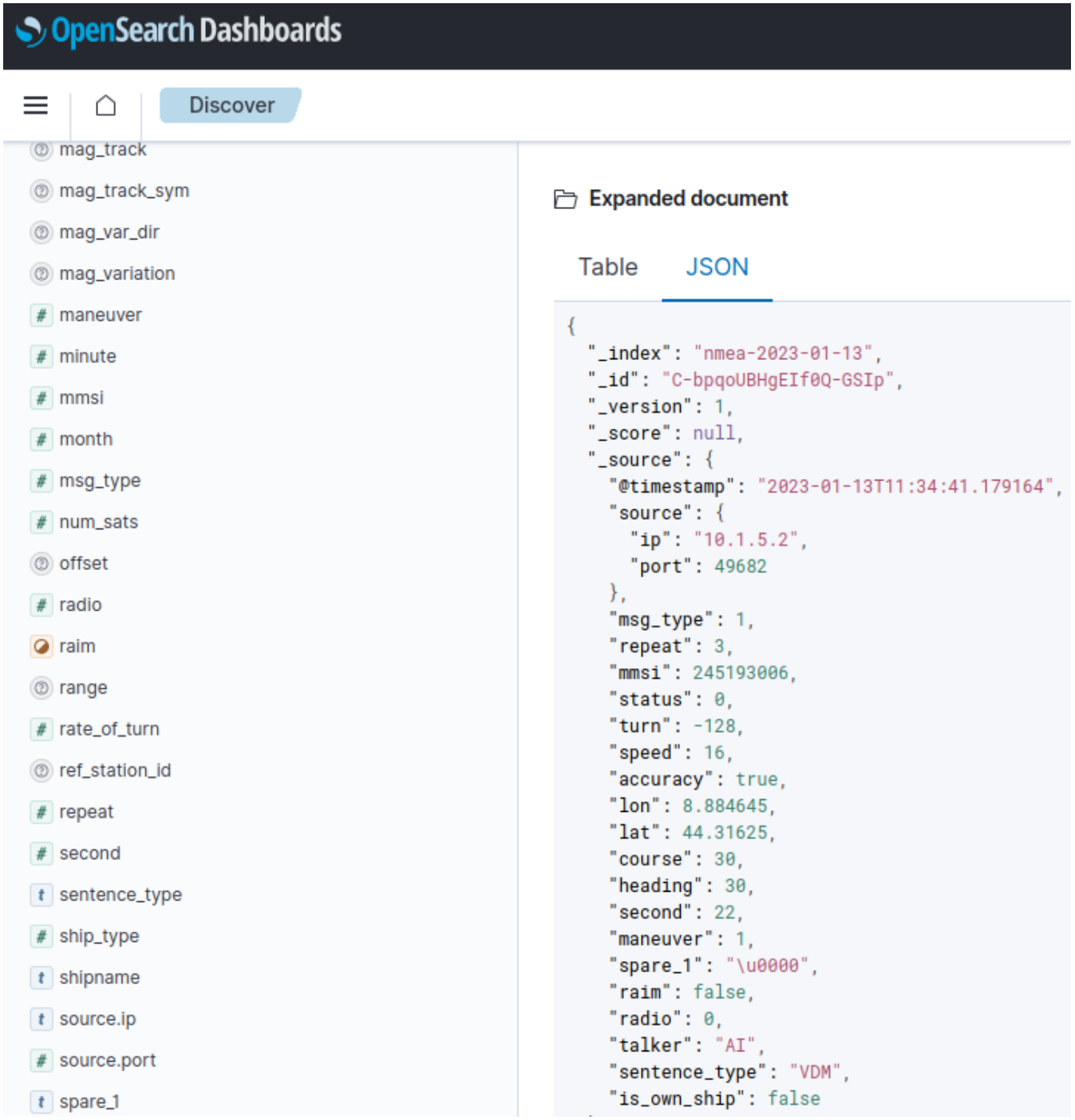
Each ModBus packet will get sent to OpenSearch.



NMEA probe

The NMEA probe will listen for the NMEA sentences being broadcasted in the INS network.

Each NMEA sentence will be parsed, converted into a structured format and sent to OpenSearch for storage.



Configuration

The probe takes the following parameters from its environment

Name	Description
`OPENSEARCH_URL`	URL of the OpenSearch server
`NMEA_HOST`	Address on which NMEA is multicasted
`NMEA_PORT`	Port on which NMEA is multicasted

OpenSearch

MaCySTe integrates with OpenSearch, automatically preseeding its dashboards server to display the data gathered from the probes.

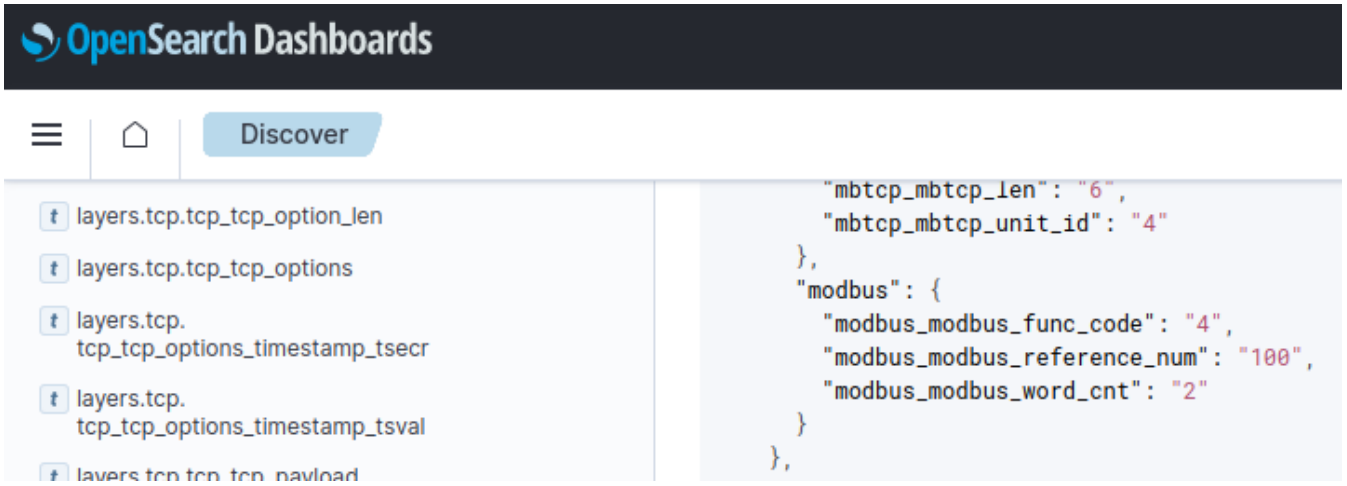
Once deployed, the dashboard can be accessed from the GUI and clicking on the `SIEM` button.

The default user and password are `admin`

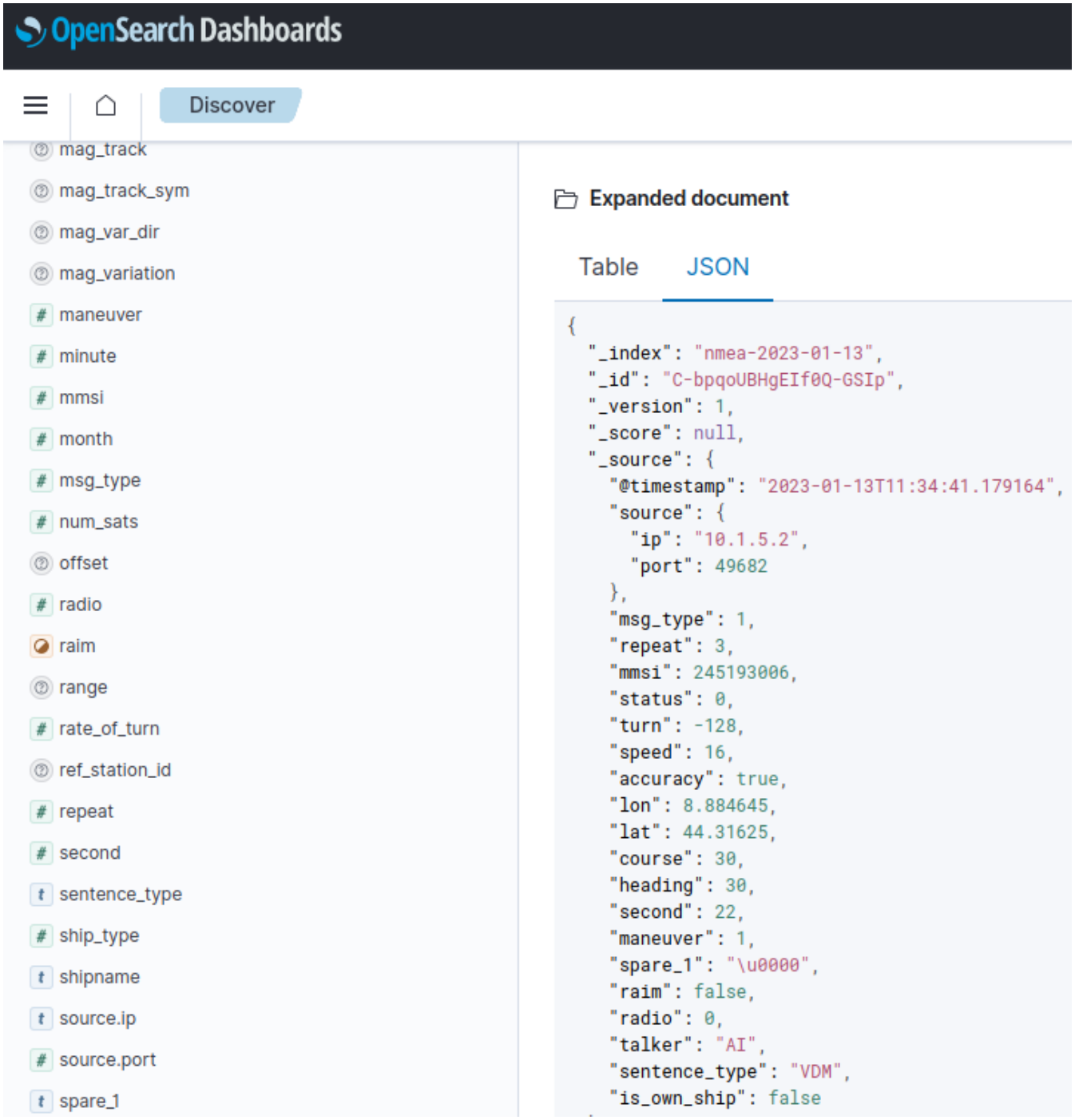
Do not expose the MaCySTe database to the outside without changing these credentials first

In order to explore the data, select *Discover* from the sidebar and select the index pattern in the top-left corner.

By default you will be able to inspect ModBus traffic



And NMEA traffic


OpenSearch Dashboards
Discover
mag_track
mag_track_sym
mag_var_dir
mag_variation
maneuver
minute
mmsi
month
msg_type
num_sats
offset
radio
raim
range
rate_of_turn
ref_station_id
repeat
second
sentence_type
ship_type
shipname
source.ip
source.port
spare_1
Expanded document
Table
JSON
{
"_index": "nmea-2023-01-13",
"_id": "C-bpqoUBHgEIf0Q-GSIp",
"_version": 1,
"_score": null,
"_source": {
"@timestamp": "2023-01-13T11:34:41.179164",
"source": {
"ip": "10.1.5.2",
"port": 49682
},
"msg_type": 1,
"repeat": 3,
"mmsi": 245193006,
"status": 0,
"turn": -128,
"speed": 16,
"accuracy": true,
"lon": 8.884645,
"lat": 44.31625,
"course": 30,
"heading": 30,
"second": 22,
"maneuver": 1,
"spare_1": "\u0000",
"raim": false,
"radio": 0,
"talker": "AI",
"sentence_type": "VDM",
"is_own_ship": false

Contributing

Feel free to contribute to MaCySTe, add new features, fix some bugs, enable new usecases, go wild!

You can also contribute by asking for new usecases and improvements.

We suggest you to:

1. start by creating an issue to gather community feedback on your ideas
2. submit your code via a pull request (WIP pull requests are fine to gather early feedback)
3. engage in the review process

Most likely, Giacomo and Alessandro will review your PR and merge it after a brief discussion. If you are ignored for a long time, please remind us by contacting at our email addresses indicated in the introduction

As a guideline try to respect the following suggestions:

- integrate with each of your pull request also a corresponding documentation change so that this manual is always kept up-to-date with your updates

Repository structure

MaCySTe follows a specific repository structure.

At the root level there are two directories `docs` and `src` where docs contains this reference manual and src the components.

docs

The documentation folder, leverages mdbook which must be installed.

Documentation can be either be read in its markdown format inside of the `src` subdirectory or seen in the browser by running `make open` (be aware that either `mdbook` or the Rust package manager `cargo` should be available in the repository)

src

Inside of the source code folder `src` , you will find the elements of MaCySTe.

In general, each top level folder indicates a related set of elements and the elements appear as direct childs of the folders.

configs

Each subdirectory of `configs` allows to store configuration files to be subsequentely mounted inside of containers and so on.

containers

Each subdirectory of `containers` will automatically be built as a container called `macyste_<subdir-name>` , to do so, ensure that a `Containerfile` is present inside.

Each container can leverage the following build args to make updating base images easier:

name	description
`FEDORA_IMAGE`	A base Fedora image
`JAVASCRIPT_IMAGE`	A base Node.js image

name	description
`PYTHON_IMAGE`	A base Python image
`RUST_IMAGE`	A base Rust image

flatpaks

Each subdirectory of `flatpaks` contains a custom Makefile with subtarget that can be invoked, this ad-hoc implementation is motivated by the fact that MaCySTe bundles only a single Flatpak Bridge Command

pods

Each subdirectory of `pods` contains a pod template which can be instantiated

pods/x/pod.yaml

The `pod.yaml` file is a mandatory file which should be present in every pod, it must be a file that once passed through an `envsubst` execution will yield an output compatible with the command `podman play kube`

pods/x/config.Makefile

Whenever some default parameters are required inside of a pod template, they can be included by specifying a `config.Makefile` file containing the variables to be defaulted in the following format:

```
variable_name ?= variable_value
```

scenarios

The scenarios directory includes the components of MaCySTe and different preset list of components. Each of these presets can be selected from the settings file.

scenarios/00-base

Each directory inside of the `00-base` represents an instantiation of a component which can be the instantiation of one or more pods and/or a network definition.

Please be advised that being a scenario the minimal unit of deploy in MaCySTe, you

should always create minimal scenario pieces to maximize their reusability.

scenarios/00-base/x/config.Makefile

The config makefile allows to define a scenario deployment information, it can include multiple directives depending on the desired usage.

Directives that appear inside of this file must be prefixed by the `x` path component to ensure uniqueness.

In order to deploy a pod, a comprehensive makefile looks like so:

```
# This indicates that this component will instantiate a pod contained in
src/pods/<pod_name_inside_of_pods_directory>
<x>_pods += <pod_name_inside_of_pods_directory>

# Each entry in this variable represent the name of a network interface card,
if undecided, use `eth0`
#   MaCySTe supports multi-nic, to do so simply add more than one interface
<x>_pod_<pod_name_inside_of_pods_directory>_network_interfaces += <if_name>
# MaCySTe will automatically allocate the IP to the pod
<x>_pod_<pod_name_inside_of_pods_directory>_<if_name>_network =
<network_name>

# Whenever some persistent state is required you can ask MaCySTe to create
these directories for you
<x>_pod_<pod_name_inside_of_pods_directory>_state_dirs += <state_dir>

# You can even augment other scenario elements' pod.yaml files with your own
additions
#   the variable $(SCENARIO_DIR) will be automatically be replaced by MaCySTe
with the path of src/scenarios/00-base
<y>_pod_<other_pod_name>_manifest_extensions += $(SCENARIO_DIR)/00-base/x/my-
snippet.yaml
```

For each network interface you define, MaCySTe will allocate an IP address which can be referenced from the vars.Makefile file by putting `$(<x>_pod_<pod_name_inside_of_pods_directory>_<if_name>_ip)`.

Similarly each state dir path will be available at `$(<x>_pod_<pod_name_inside_of_pods_directory>_state_dir_<state_dir>)`

scenarios/00-base/x/ipam.Makefile

This IPAM file allows to define new networks to be automatically allocated by MaCySTe.

To define a new network write:

```
NETWORK_NAMES += <uppercase_network_unique_name>

<uppercase_network_unique_name>_NAME = macyste_<network_name>
<uppercase_network_unique_name>_CIDR = <ip cidr>
<uppercase_network_unique_name>_DRIVER = <bridge / macvlan>
```

so if we wanted to create an isolated (macvlan) network with IPs from the `10.1.42.0/24` range called `mynet` you would write:

```
NETWORK_NAMES += MYNET

MYNET_NAME = macyste_mynet
MYNET_CIDR = 10.1.42.0/24
MYNET_DRIVER = macvlan
```

See the network page to see the difference between the different drivers.

Multiple network declarations can be bundled together if it makes logical sense to do it

scenarios/00-base/x/vars.Makefile

The vars file allow you to define additional substitutions to be performed inside of the pod.yaml file

To do so write a file like so:

```
<x>_pod_<pod_name_inside_of_pods_directory> += <var_name>

#   the variable $(CONFIG_DIR) will be automatically be replaced by MaCySTe
with the path of src/configs
#   the variable $(SCENARIO_DIR) will be automatically be replaced by MaCySTe
with the path of src/scenarios/00-base
<x>_pod_<pod_name_inside_of_pods_directory>_<var_name> = <var_value>

# If you want to define a default for a variable do like so
<var_name> ?= <var_value>
```

You can also passthrough variables that will be created as a result of the config.Makefile instantiation.

scenarios/x

Every other scenario not called `00-base` contains a specification for a selectable scenario

scenarios/x/alloc.Makefile

The alloc.Makefile file contains an automatically generated IP allocation for every

component belonging to the scenario. We advise you not to modify this file manually and let MaCySTe generate it.

If you wish to alter an IP allocation:

- Run your scenario with `make up SCENARIO_NAME=<name>`
- Stop running with `make down SCENARIO_NAME=<name>`
- Modify the `alloc.Makefile` file

MaCySTe will not modify your manually set values

scenarios/x/config.Makefile

This file allows you to specify which elements of `00-base` belong to a scenario

Define them like so

```
MODULES += <module_1_name>
MODULES += <module_2_name>
MODULES += <module_n_name>
```

Remember that they are used in order so please ensure that all dependencies are correct *for instance, a network should appear before modules using it for their pods*

scripts

The `scripts` directory contains scripts which are used by the makefile to perform functions not available inside of the restricted GNU Make language.

In MaCySTe, the principal script is `allocate_ip.py` a script that takes a space separated list of `alloc_file_name cidr allocation_name` from its standard input and will generate automatically an allocation file

state

`state` acts as a mutable counterpart to the config directory, allowing to store mutable data for containers such as database contents and so on.

Leveraging the state directory correctly allows to restore a MaCySTe instance from a backup more easily.

settings.Makefile

Please see the dedicated page

Makefile

Please see the dedicated page

Its internal structure is not for the faint of heart and requires deep knowledge of Makefile syntax. It leverages meta-programming to easily integrate the various modules into a comprehensive and cohesive experience.

Licensing

MaCySTe is licensed according to the GNU Affero General Public License v3 and its intellectual property belongs to its original authors (as listed in the introduction) and its contributors.

Contact us for additional licensing options.

Giving credit

The best way to give credit to MaCySTe is by citing its reference paper:

```
IN PEER REVIEW - MaCySTe: a virtual testbed for maritime cybersecurity - G.
Longo, A. Orlich, S. Musante, A. Merlo, E. Russo
```