CyberShip Software Suite

Architecture, Operation, and Configuration Manual

Marine Cybernetics Laboratory (MCLab)
NTNU

March 2026

Contents

1 Introduction

The **CyberShip Software Suite** is a collection of ROS 2 packages developed at the Marine Cybernetics Laboratory (MCLab) at NTNU. Together they provide a complete software stack for simulating, visualizing, and controlling autonomous maritime vessels. The suite covers everything from digital-twin URDF models and real-time sensor integration (IMU, motion capture) to advanced control algorithms (dynamic positioning, thrust allocation).

The suite is published on GitHub at `https://github.com/NTNU-MCS/cybership_software_suite` and archived on Zenodo (DOI: `10.5281/zenodo.17233653`).

1.1 Supported Vessels

Three vessel models are currently supported:

Voyager Compact vessel used for general control experiments.

Enterprise Medium-size vessel; simulation supported.

Drillship Large vessel with dedicated launch configuration.

2 Package Overview

The repository is organized as a collection of focused ROS 2 packages. Table 1 summarizes each package and its primary responsibility.

Package	Description
`cybership_bringup`	Launch files that orchestrate the complete startup of a vessel (hardware drivers, localization, sensor nodes).
`cybership_simulator`	Lightweight Python simulation scripts based on simplified vessel physics.
`cybership_viz`	RViz configurations and launch files for real-time vessel visualization.
`cybership_description`	URDF models, meshes, and coordinate-frame definitions for each vessel.
`cybership_config`	Centralized YAML configuration files shared across the suite.
`cybership_interfaces`	Custom ROS 2 message and service definitions for maritime operations.
`cybership_controller`	Base package for vessel controllers.
`cybership_dp`	Dynamic positioning controllers (force and velocity).
`cybership_thrusters`	Low-level thruster drivers supporting VSP, fixed, and azimuth types.

Package	Description
`cybership_mocap`	ROS 2 bridge to the Qualisys motion capture system.
`cybership_utilities`	Shared helper functions, launch argument definitions, and GUI tools.
`cybership_external`	Third-party submodule dependencies (IMU driver, Dynamixel, PCA9685, etc.).
`cybership_tests`	Integration and unit tests for the suite.

Table 1: CyberShip Software Suite packages.

3 System Architecture

3.1 High-Level Architecture

The system follows a layered ROS 2 architecture. Data flows from physical sensors upward through localization and state estimation to high-level control, then back down through thrust allocation to the actuators.

1. **Sensor layer** – IMU (BNO055), motion capture (Qualisys).
2. **Localization layer** – `robot_localization` EKF fuses sensor data into a consistent odometry estimate.
3. **Control layer** – Dynamic positioning controllers compute force and torque demands.
4. **Actuation layer** – The thruster package converts force demands to actuator signals (PWM, RPM, servo angle).
5. **Hardware layer** – PCA9685 PWM board, Dynamixel azimuth servos.

3.2 Distributed ROS 2 Network

The system runs as a distributed ROS 2 network across multiple machines. The **cybership-master** Raspberry Pi (`192.168.1.102`) acts as the **Fast DDS discovery server** for all participants. Vessels run their own bringup as systemd user services and connect back to the discovery server automatically.

Host	Role / Address
`cybership-master.local`	Discovery server, UI host – `192.168.1.102`
`cybership-voyager.local`	Vessel node
`cybership-drillship.local`	Vessel node
`192.168.1.10`	Qualisys motion capture system (read-only)

3.3 Coordinate Frames

The vessels use two main coordinate frames defined in `cybership_description`:

`base_link` East-North-Up (ENU) frame. All sensors are defined here.

`base_link_ned` North-East-Down (NED) frame. Used for navigation and control.

The NED frame is obtained by a static transform applied to the ENU frame.

3.4 Network Topology

Figure 1 shows the physical network layout of the CyberShip lab. All machines communicate over a shared `192.168.1.1/24` Wi-Fi LAN. The **cybership-master** Raspberry Pi acts as the central hub: it runs the Fast DDS discovery server (port 11811), serves the web UI (port 8001), and hosts any lab-side ROS 2 nodes. Vessel Raspberry Pis register with the discovery server automatically at boot. Student workstations join by exporting `ROS_DISCOVERY_SERVER` in their shell environment.

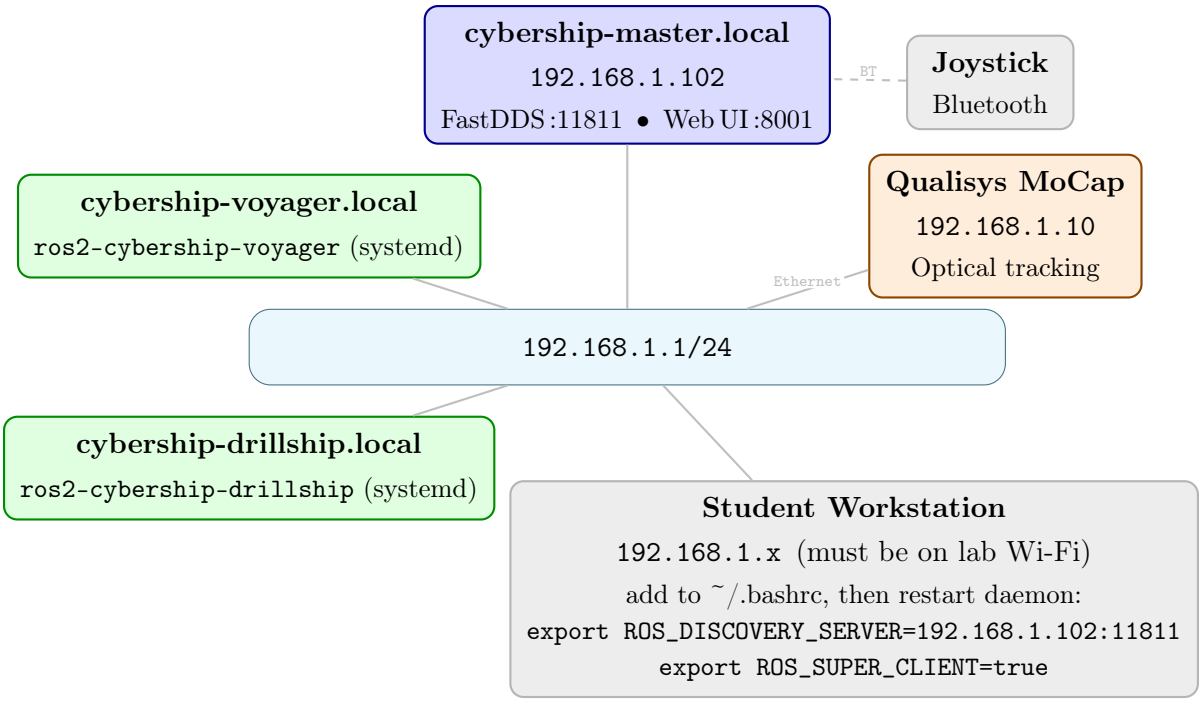

Figure 1: Physical network topology of the CyberShip lab. Vessels and workstations communicate over the `192.168.1.1/24` Wi-Fi network. **cybership-master** is connected to the Qualisys system via a dedicated Ethernet cable and acts as the Fast DDS discovery server for all ROS 2 participants. The Bluetooth joystick pairs with the master and its events are forwarded as ROS 2 topics across the network.

4 Installation

Installation instructions are maintained in the repository `README.md` and kept up to date with each release. Refer to:

https://github.com/NTNU-MCS/cybership_software_suite#installation

5 Network and Discovery Server Configuration

Note: The IP addresses listed throughout this document reflect the current lab configuration and are subject to change. If an address does not respond, verify the current assignment with the lab supervisor before attempting further troubleshooting.

The CyberShip lab uses a dedicated `192.168.1.1/24` subnet. All machines (vessels, lab computers, and the master node) must be on this subnet.

5.1 Connecting a Workstation

Add the following lines to `~/.bashrc` on any machine that needs to join the ROS 2 network:

```
export ROS_DISCOVERY_SERVER="192.168.1.102:11811"
export ROS_SUPER_CLIENT=true
```

After editing, restart the ROS 2 daemon:

```
ros2 daemon stop ; ros2 daemon start
```

5.2 Virtual Machine Setup

Students using a virtual machine must set their VM network adapter to **Bridged** mode and select the Wi-Fi adapter as the bridged interface. Verify the assigned IP is in the `192.168.1.1/24` range with `ip addr`.

5.3 SSH Access

```
ssh mclab@cybership-master.local
ssh mclab@cybership-voyager.local
ssh mclab@cybership-drillship.local
```

Tip: Always use `tmux` when working on `cybership-master` so sessions persist across disconnections.

6 Bringing Up a Vessel

6.1 Automatic Startup

Each vessel automatically starts its bringup at boot via a **systemd user service** named `ros2-cybership-<vessel_model>`. The service runs:

```
ros2 launch cybership_bringup all.launch.xml
```

You do not normally need to SSH into the vessel; simply powering it on (connecting the battery) is sufficient.

6.2 Manual Bringup

To manually launch bringup for a vessel, use the corresponding launch file from `cybership_bringup`:

```
# Voyager
ros2 launch cybership_bringup voyager.launch.py \
    vessel_name:=voyager vessel_model:=voyager

# Enterprise
ros2 launch cybership_bringup enterprise.launch.py

# Drillship
ros2 launch cybership_bringup drillship.launch.py
```

6.3 Bringup Include Files

The bringup package composes several include launch files, each starting a specific subsystem:

`azimuth_controller.launch.py`
: Starts the Dynamixel azimuth servo controller node.

`imu_bno055.launch.py`
: Starts the BNO055 IMU driver node.

`motion_capture_system_connector.launch.py`
: Connects to the Qualisys motion capture server and retrieves rigid-body poses.

`motion_capture_system_transformer.launch.py`
: Transforms raw mocap data into the vessel's coordinate frame.

`robot_localization.launch.py`
: Runs the `robot_localization` EKF node to fuse IMU and mocap data.

`servo_driver.launch.py`
: Starts the servo driver node (PCA9685-based).

`thruster_control.launch.py`
: Starts the thruster controller nodes.

`urdf_description.launch.py`
: Publishes the URDF model via `robot_state_publisher`.

`localization.launch.py`
: Top-level localization launch combining the above components.

7 Localization

Vessel localization fuses two complementary sensor streams:

1. **BNO055 IMU** – Provides 9-DOF inertial measurements (accelerometer, gyroscope, magnetometer). Integrated via the `bno055` ROS 2 driver in `cybership_external`.
2. **Qualisys motion capture** – Provides absolute 6-DOF pose measurements at `192.168.1.10` using the `mocap4ros2_qualisys` driver and `cybership_mocap` bridge.

The fused estimate is produced by the `robot_localization` Extended Kalman Filter (EKF) node. Convergence of the EKF can be monitored by inspecting the covariance matrix on the `/<vessel_name>/measurement/odom` topic or by watching the vessel in RViz.

7.1 Motion Capture Node Details

The `cybership_mocap` package subscribes to the `rigid_bodies` topic (`mocap4r2_msgs/msg/RigidBodie` and republishes pose data as `geometry_msgs/msg/PoseWithCovarianceStamped`. Key configuration parameters are shown in Table 2.

Parameter	Type	Description	Default
`world_frame`	string	Global reference frame name	`mocap`
`base_frame`	string	Robot base frame	`cybership/base_link`
`rigid_body_name`	string	Name in mocap system	`cybership`
`topic_name`	string	Output pose topic	`cybership/pose`
`publish_tf`	bool	Broadcast TF transform	`true`

Table 2: Motion capture node configuration parameters.

Important: The order in which rigid bodies are registered in the Qualisys software determines their index. Ensure the order matches the vessel being launched.

8 Thruster System

The `cybership_thrusters` package implements low-level thruster drivers in C++. It supports three actuator types.

8.1 Thruster Types

VSP *Voith Schneider Propeller* – Outputs differential arm positions (`arm_x`, `arm_y`) and a fixed RPM command. Suitable for omnidirectional propulsion.

Fixed A single-axis thruster outputting a normalized force signal. Force is linearly interpolated between `force_min` and `force_max`.

Azimuth Computes both a rotation angle (via `atan2`) and an RPM from the magnitude of the input force vector. Used for azimuth-steerable pods.

8.2 Safety Watchdog

All thruster controllers include a watchdog timer. If no force command is received on the configured `force_topic` within **2.0 seconds** (configurable per thruster), the driver automatically publishes zero on all output topics. The watchdog runs at 10 Hz.

8.3 Enabling / Disabling Thrusters

Thrusters are **disabled by default**. They must be explicitly enabled via ROS 2 service call:

```
# Enable
ros2 service call <vessel_name>/thruster/enable std_srvs/srv/Empty {}

# Disable
ros2 service call <vessel_name>/thruster/disable std_srvs/srv/Empty {}
```

Operational note: Before handing control to students, always navigate to the thrust allocation web UI at `http://192.168.1.102:8001` and **deactivate the thrust allocation** until the student controller is ready.

8.4 Thruster Configuration (YAML)

Thrusters are configured under the `thrusters` namespace in a YAML parameter file. Each thruster has a unique name and a type-specific set of parameters:

```
thrusters:
  bow_thruster:
    type: "fixed"
    force_topic: "/voyager/bow_thruster/force"
    signal_topic: "/voyager/bow_thruster/signal"
    force_max: 5.0
    force_min: -5.0
    signal_inverted: false
    safety_timeout: 2.0

  port_azimuth:
    type: "azimuth"
    force_topic: "/voyager/port_azimuth/force"
    angle_topic: "/voyager/port_azimuth/angle"
    rpm_topic: "/voyager/port_azimuth/rpm"
    force_max: 20.0
    force_min: -20.0
    angle_inverted: false
    rpm_inverted: false

  vsp_main:
    type: "vsp"
    force_topic: "/voyager/vsp/force"
    arm_x_topic: "/voyager/vsp/arm_x"
    arm_y_topic: "/voyager/vsp/arm_y"
    rpm_topic: "/voyager/vsp/rpm"
    force_max: 10.0
    force_min: -10.0
    rpm_cmd: 1000.0
    arm_x_inverted: false
    arm_y_inverted: false
```

9 Dynamic Positioning

The `cybership_dp` package implements two layered controllers for dynamic positioning.

9.1 Velocity Controller

The velocity controller receives a desired velocity setpoint and computes force/torque demands using independent PID loops for surge, sway, and yaw. Gains are specified in the package's `config/` YAML files. Computed demands are published to the force controller.

9.2 Force Controller

The force controller uses the **Skadi** library (https://github.com/incebellipipo/skadipy) for optimal thrust allocation. It receives 3-DOF force/torque demands (surge, sway, yaw) and distributes them across the available thrusters.

9.3 Launch Files

```
# Full DP system (velocity + force controller)
ros2 launch cybership_dp dp.launch.py \
    vessel_name:=voyager vessel_model:=voyager

# Force controller only
ros2 launch cybership_dp force_controller.launch.py \
    vessel_name:=voyager vessel_model:=voyager

# Velocity controller only
ros2 launch cybership_dp velocity_controller.launch.py \
    vessel_name:=voyager vessel_model:=voyager
```

9.4 Vessel Model Support

The `voyager` model is fully supported. Support for `enterprise` and `drillship` is planned but not yet complete.

10 Simulation

The `cybership_simulator` package provides lightweight Python scripts that simulate vessel dynamics using simplified physics. Each vessel has its own dedicated script, currently including:

- CyberShip Enterprise 1 (3-DOF planar model)

The simulator publishes the same topics as the physical bringup stack (odometry, transforms) so that all higher-level control packages are compatible with both modes.

Pass `use_sim_time:=true` to any launch file when running in simulation:

```
ros2 launch cybership_bringup voyager.launch.py \
    vessel_name:=voyager vessel_model:=voyager \
```

```
    use_sim_time:=true
```

11 Visualization

11.1 RViz

The `cybership_viz` package provides preconfigured RViz scenes showing:

- The vessel URDF model with live TF transforms.
- Sensor data overlays (IMU, pose estimates).
- Trajectory history.

11.2 URDF / Robot State Publisher

The vessel's 3D model is published by the `robot_state_publisher` node. URDF files and mesh assets live in `cybership_description/urdf/` and `cybership_description/meshes/` respectively. To launch standalone visualization:

```
ros2 launch cybership_description description.launch.py
```

11.3 Utilities Web UI

A lightweight web interface runs on the master node and is accessible at:

http://192.168.1.102:8001

It can be started with:

```
ros2 launch cybership_utilities ui.launch.py
```

11.4 Force Mux GUI

The `cybership_utilities` package also includes a PyQt GUI (`nodes/force_mux_gui.py`) for inspecting and switching the active force-mux input topic for a selected vessel namespace. The namespace can be supplied via the `VEHICLE_NS` environment variable or typed directly in the GUI.

12 Joystick / Teleoperation

Bluetooth joysticks are connected to the master node using `bluetoothctl`. Once connected, the teleop stack is launched with:

```
# Drillship on joystick 0
ros2 launch cybership_teleop allocation.launch.py \
    vessel_name:=drillship joy_id:=0

# Voyager on joystick 1
ros2 launch cybership_teleop allocation.launch.py \
```

```
    vessel_name:=voyager joy_id:=1
```

Raw joystick events are published by the standard `joy` node:

```
ros2 run joy joy_node
```

13 ROS 2 Interfaces

The `cybership_interfaces` package defines the custom ROS 2 interfaces used across the suite. Currently defined interfaces are listed in Table 3.

Type	Name	Description
Service	`ResetSimulator`	Resets the simulator to a given 2D pose. Request: `geometry_msgs/Pose2D`. Response: `bool success`.

Table 3: Custom ROS 2 interfaces.

To depend on these interfaces from another package:

```
<!-- package.xml -->
<depend>cybership_interfaces</depend>
```

```
# Python usage
from cybership_interfaces.srv import ResetSimulator
```

14 External Dependencies

Third-party packages are included as Git submodules under `cybership_external/`. Table 4 lists each dependency.

Package	Description
`mocap4ros2_qualisys`	ROS 2 driver for the Qualisys optical motion capture system.
`dynamixel_azimuth_driver`	ROS 2 driver for Dynamixel motors used as azimuth actuators.
`ros2_pca9685`	ROS 2 driver for the PCA9685 I^2C PWM controller board.
`mocap4r2_msgs`	ROS 2 message definitions for motion capture data.
`bno055`	ROS 2 driver for the Bosch BNO055 9-axis IMU.

Table 4: External submodule dependencies.

To update all submodules to their latest upstream versions:

```
git submodule update --remote
```

15 Configuration Reference

15.1 Centralized Configuration

The `cybership_config` package stores YAML configuration files that are installed and shared across the suite. Vessel-specific overrides are placed in per-vessel subdirectories within `config/`.

15.2 Common Launch Arguments

The `cybership_utilities` package defines a set of standardized launch arguments used by all launch files:

`vessel_model` Model identifier string (e.g., `voyager`, `enterprise`, `drillship`).

`vessel_name` ROS 2 namespace for the vessel (typically matches `vessel_model`).

`param_file` Path to an optional override parameter file.

`use_sim_time` Boolean flag to use simulation time clock.

15.3 ROS Environment

The vessel's ROS setup script is located at `~/.rosrc`. Source it before running ROS 2 commands on the vessel:

```
source ~/.rosrc
```

16 Systemd Service Management

Vessel bringup and the Fast DDS discovery server are managed as **systemd user services**.

`ros2-cybership-<model>` Per-vessel bringup service. Starts automatically at boot on the corresponding Raspberry Pi.

FastDDS service Runs on `cybership-master` as a user service, providing the DDS discovery server on port 11811.

Useful service management commands:

```
# Check status
systemctl --user status ros2-cybership-voyager

# Restart bringup
systemctl --user restart ros2-cybership-voyager

# View logs
journalctl --user -u ros2-cybership-voyager -f
```

17 Operational Procedure

17.1 Standard Startup Sequence

1. Power the vessel by connecting the battery. The bringup service starts automatically.
2. On the control workstation, ensure the ROS discovery server variables are set in `~/.bashrc` (Section 5) and restart the daemon.
3. Open the web UI at `http://192.168.1.102:8001` and **deactivate the thrust allocation**.
4. Wait for the EKF to converge (monitor covariance on `/<vessel_name>/measurement/odom` or watch RViz).
5. Once the student controller is ready, re-enable thrust allocation from the web UI or via the service call.
6. Keep the vessel within the motion-capture calibrated area during experiments.

17.2 Troubleshooting

Topics not visible Restart the ROS 2 daemon:

```
ros2 daemon stop ; ros2 daemon start
```

EKF diverging Check the covariance on `/<vessel_name>/measurement/odom`. Ensure the vessel is within the mocap calibrated area and that the Qualisys system has acquired all markers.

Rigid body order mismatch Verify the rigid body order in the Qualisys software matches the expected vessel index. Order matters for index-based assignment.

Thrusters unresponsive Confirm they are enabled:

```
ros2 service call <vessel_name>/thruster/enable std_srvs/srv
    ↪ /Empty {}
```

Also verify no safety watchdog timeout has silenced the commands (check that force commands arrive within 2 s).

Discovery issues Confirm the FastDDS service is running on `cybership-master` and that your workstation is on the `192.168.1.1/24` subnet.

18 Developer Notes

18.1 Adding a New Vessel Model

1. Add URDF and mesh files to `cybership_description/urdf/` and `cybership_description/meshe`
2. Create a vessel-specific configuration directory under `cybership_config/`.

3. Add a new `<model>.launch.py` in `cybership_bringup/launch/`.
4. Register the model string in `cybership_utilities/utilities.py`.
5. Implement force and velocity controllers in `cybership_dp/` for the new model.

18.2 Node Naming Convention

All nodes and topics are namespaced by the vessel name to allow multiple vessels on the same ROS 2 network without collisions:

`/<vessel_name>/<subsystem>/<topic>`

18.3 Extending Interfaces

Custom messages and services should be added to `cybership_interfaces`. After adding `.msg` or `.srv` files, update the `CMakeLists.txt` accordingly and rebuild the package.

19 Quick Operation Manual

This section is a concise step-by-step guide for the most common lab scenarios. For full explanations refer to the relevant sections of this document.

19.1 First-Time Workstation Setup

Do this **once** on any computer that will interact with the vessels.

1. Connect to the lab Wi-Fi and confirm your IP is in `192.168.1.1/24`:

```
ip addr
```

2. Add the following lines to `~/.bashrc`:

```
export ROS_DISCOVERY_SERVER="192.168.1.102:11811"
export ROS_SUPER_CLIENT=true
```

3. Apply the changes and restart the ROS 2 daemon:

```
source ~/.bashrc
ros2 daemon stop ; ros2 daemon start
```

4. Verify you can see ROS 2 topics from the network:

```
ros2 topic list
```

19.2 Starting an Experiment

1. **Power the vessel.** Connect the battery. The vessel boots and starts the ROS 2 bringup automatically via systemd. No SSH needed.
2. **Open the web UI** at `http://192.168.1.102:8001` and configure the vessel accoridng to your needs.

3. **Wait for the EKF to converge.** Watch the vessel in RViz or check the covariance on the odometry topic:

```
ros2 topic echo /voyager/measurement/odom
```

   The diagonal elements of the covariance matrix should drop to small, stable values.

4. **Ensure the vessel is inside the motion-capture calibrated area** before running any controller.

5. **Launch the visualization** on your workstation (optional but recommended):

```
ros2 launch cybership_utilities ui.launch.py
```

19.3 Running a Student Controller

1. Build and source the student package on the workstation:

```
cd ~/ros_ws
colcon build --packages-select <student_package>
source install/setup.bash
```

2. Launch the student controller:

```
ros2 launch <student_package> <launch_file>.launch.py \
    vessel_name:=voyager vessel_model:=voyager
```

3. Verify the controller is publishing on the expected force topic, then **activate** thrust allocation from the web UI at `http://192.168.1.102:8001`.

4. Monitor the vessel. If behaviour is unexpected, immediately **deactivate** thrust allocation from the web UI.

19.4 Joystick Teleoperation

1. On `cybership-master`, connect the joystick via Bluetooth:

```
bluetoothctl
# Inside bluetoothctl:
scan on
pair <MAC>
connect <MAC>
```

2. Launch the joy node and teleop allocator (on master or your workstation):

```
ros2 run joy joy_node &
ros2 launch cybership_teleop allocation.launch.py \
    vessel_name:=voyager joy_id:=1
```

3. Activate thrust allocation from the web UI and operate the vessel with the joystick.

19.5 Shutting Down

1. Stop the student controller process (`Ctrl+C` in its terminal).
2. Disconnect the vessel battery to power it off. The systemd service will stop cleanly.
3. No further action is needed on `cybership-master`; it can be left running between sessions.

19.6 Restarting the Bringup on a Vessel

If the vessel bringup needs to be restarted (e.g. after a configuration change):

```
ssh mclab@cybership-voyager.local
systemctl --user restart ros2-cybership-voyager
# Check it came back up:
systemctl --user status ros2-cybership-voyager
```

A Quick-Reference Command Cheatsheet

```
## Environment setup (add to ~/.bashrc)
export ROS_DISCOVERY_SERVER="192.168.1.102:11811"
export ROS_SUPER_CLIENT=true

## SSH
ssh mclab@cybership-master.local
ssh mclab@cybership-voyager.local
ssh mclab@cybership-drillship.local

## Vessel bringup (manual)
ros2 launch cybership_bringup voyager.launch.py \
    vessel_name:=voyager vessel_model:=voyager

## Utilities UI
ros2 launch cybership_utilities ui.launch.py
# Web UI: http://192.168.1.102:8001

## Dynamic positioning
ros2 launch cybership_dp dp.launch.py \
    vessel_name:=voyager vessel_model:=voyager

## Teleop (joysticks)
ros2 run joy joy_node
ros2 launch cybership_teleop allocation.launch.py \
    vessel_name:=voyager joy_id:=1

## Thrusters
ros2 service call voyager/thruster/enable std_srvs/srv/Empty {}
ros2 service call voyager/thruster/disable std_srvs/srv/Empty {}

## Restart ROS daemon
```

```
ros2 daemon stop ; ros2 daemon start

## Monitor EKF
ros2 topic echo /voyager/measurement/odom
```