CyberShip Enterprise I

User Manual

2021.11.22

NTNU – Trondheim
Norwegian University of
Science and Technology

Faculty of Engineering Science and Technology

Department of Marine Technology

Contents

Part I

Technical description

Chapter 1

Hardware

1.1 Introduction to CSE1

The CS Enterprise I was initially bought in 2009, as a fully configured model boat named "Aziz" built by Model Slipway. Due to requirements for master and PhD experiments, the model was refitted by Skåtun (2011). The work performed on CS Enterprise I include, but is not limited to, dynamic positioning systems, maneuvering systems and path following, and navigation with virtual reality.

The vessel is a 1:50 scale model of a tug boat, and is fitted with two Voith Schneider propellers(VSP) astern and one bow thruster(BT). The main dimensions of the vessel are:

Table 1.1: Main dimensions of CSE1

LOA	1.105[m]
B	0.248 [m]
Δ	14.11 [kg]

1.1.1 Literature

The development of CSE1 is a product of much research from several theses, which contain complementary information on the theory applied to the system.

Journals and conferences

- LOS guidance for towing an iceberg along a straight-line path (Orsten et al., 2014)

Specialization projects and master theses

- Development of a DP system for CS Enterprise I with Voith Schneider thrusters. (Skåtun, 2011)
- Development of a modularized control architecture for CS Enterprise I for path-following based on LOS and maneuvering theory (Tran, 2013)
- Automatic Reliability-based Control of Iceberg Towing in Open Waters (Orsten, 2014)
- Line-Of-Sight-based maneuvering control design, implementation, and experimental testing for the model ship C/S Enterprise I.(Tran, 2014)
- Remote Control and Automatic Path-following for C/S Enterprise I and ROV Neptunus (Sandved, 2015)
- Marine Telepresence System (Valle, 2015)
- Nonlinear Adaptive Motion Control and Model-Error Analysis for Ships-Simulations and MCLab experiments (Bjørne, 2016)
- Low-Cost Observer and Path-Following Adaptive Autopilot for Ships (Mykland, 2017)

Other

- YouTube video (Skåtun, 2014)

1.2 Actuators

Figure 1.1 illustrate the position of the actuators, and their distance from CO is given in Table 1.2. The BT and VSP motor speeds are controlled by an Electronic Speed Control(ESC). The ESC receive their setpoints as pulse-width modulated (PWM) signals from the cRIO digital output

module. The VSP blade pitches are controlled by servos. The servos also receive their setpoint as PWM signals.

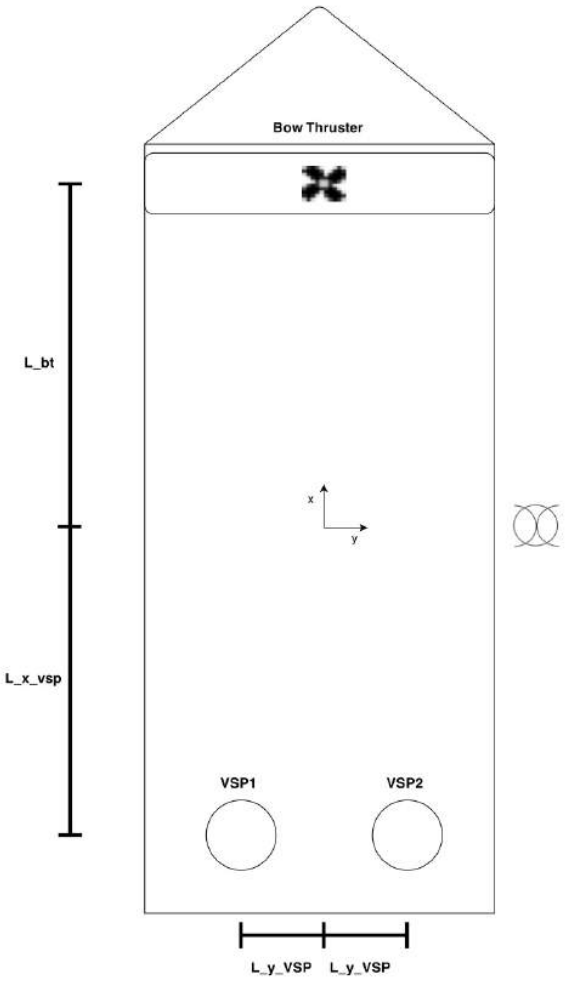


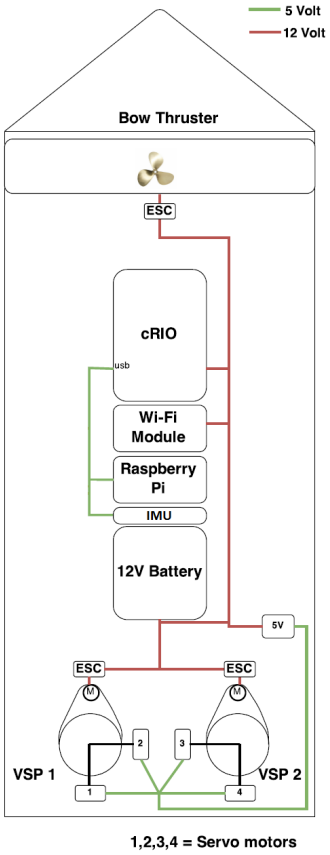
Figure 1.1: Position of actuators. Adapted from Valle (2015)

Parameter	**Symbol**	**Value**[m]
x length to VSP	$L_{x,VSP}$	-0.4574
x length to BT	L_{BT}	0.3875
y length to VSP	$L_{y,VSP}$	0.055

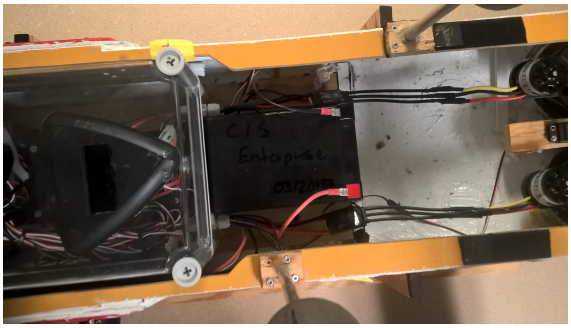
Table 1.2: Position of actuators

1.3 Power system

CSE1 is powered with one 12V 12Ah battery on-board. Some of the components require different voltage, and thus some voltage converters are mounted. However, the setup works as it is, and by connecting the battery to the wires, the whole system is powered. A schematic of the power grid is illustrated in Figure 1.2a, and Figure 1.2b show a photo of the battery mounted and connected.



(a) CSE1 power system



(b) Battery mounted and connected

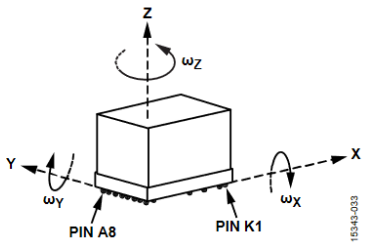
Figure 1.2: Battery system

1.4 IMU

CSE1 is equipped with one Inertial Measurement Unit (IMU) from Analog Devices. The sensor mounted on-board is the ADIS16470 and includes a triaxis gyroscope and triaxis accelerometer. The sensor has built-in compensation for bias, alignment and sensitivity, and thus provides accurate measurements over a temperature range of -10 to +75 degrees Celsius. The sampling rate is set to 100 Hz. The most relevant data is presented in Table 1.3, and for supplementary information the reader is referred to the data sheet Analog Devices (2019). The coordinate frame of the sensor is illustrated in Figure 1.3a, with positive directions illustrated by arrows. THe the sensor is mounted with a different orientation than the body frame, as can be seen in Figure 1.3b. Using the *zyx*-convention, the sensor frame has an orientation relative body frame: $(\phi, \theta, \psi) = (\pi, 0, 0)$. Hence, by using the rotation matrix with these values, the measured accelerations and angular rates can be rotated to the body-frame.

Table 1.3: IMU specifications

	Parameter	Typical value	Unit
Gyroscopes	Dynamic range	±2000	°/sec
	Sensitivity	10	LSB/°/sec
	Bias stability, σ	8	°/hr
	Angular random walk	0.34	$°/\sqrt{hr}$
	Output noise	0.17	°/sec rms
Accelerometers	Dynamic range	±40	g
	Sensitivity	800	LSB/g
	Bias stability, σ	0.013	mg
	Velocity random walk	0.037	$m/sec/\sqrt{hr}$
	Output noise	2.3	mg rms
Power supply	Operating voltage	3.3 ± 0.3	V



(a) IMU reference frame from manufacturer

(b) IMU mounted in the vessel

Figure 1.3: Inertial Measurement Unit in CSE1

1.5 Control system

The on-board control system consists of the following parts:

- a Raspberry Pi 4 (RPi) single-board computer
- three electronic speed controllers (ESC)

- four servos

1.5.1 RPi

The Raspberry Pi provides communication with the DS4 controller, the servos and the electronic speed controllers.. It works as an embedded system, and once powered it will start searching for the wireless controller. When connection is established, it will continuously send signals to the servos and ESC based on operation mode. To successfully connect the sixaxis controller to the RPi, wait for the Bluetooth dongle to start blinking before pressing the PS-button on the controller.

1.5.2 ESC

The ESC's are controlled with PWM signals, based on PWM tick signals. Table 1.4 describe the setup for all ESC's on-board CSE1, and Table 1.5 gives the pwm signal range for each ESC.

Table 1.4: PWM specification for ESC

Initial value	**Scaling**	**Offset**	**PWM period** [Ticks]
0	100	0	800.000

Table 1.5: PWM ranges for ESC

	ESC_BT[%]	**ESC_VSP1**[%]	**ESC_VSP2**[%]
min	7.00	3.12	3.12
neutral	7.55	5.01	5.01
max	8.10	6.90	6.90

Chapter 2

Software

2.1 Introduction

The control system of CSE1 is built around a framework called Robot Operating System. ROS consists of tools, libraries, and conventions that help in building robot applications. This chapter gives a description of vessels software hierarchy. Note that most of this software is ready to use, and alterations in the software described here should not be necessary, unless specified.

2.2 Driver setup

2.2.1 Actuator drivers

To activate the actuators on the vessel, first SSH to the onboard RPi 4 and launch nodelet manager and the CSE actuator driver:

```
$ rosrun nodelet nodelet manager __name:=nodelet_manager
$ rosrun nodelet nodelet load cse_actuator_driver/
    cse_actuator_driver_nodelet
nodelet_manager __name:=cse_actuator_driver
```

You should hear a beeping sound if the nodes are succesfully run.

2.2.2 Qualisys drivers

Next is to launch the Qualisys drivers. These are written in ROS2 rather than ROS1 and must be activated from the ROS2 workspace it is installed to, either from the onboard RPi, or from another computer connected to the MCLab network:

```
$ ros2 run ros2_qualisys_driver qualisys_driver_exe
--ros-args --params-file src/ros2_qualisys_driver/params/params.yaml
$ ros2 lifecycle set /qualisys_driver configure
$ ros2 lifecycle set /qualisys_driver activate
```

To source ROS2 or ROS1 in the workspace, use:

```
$ . /opt/ros/$ROS_DISTRO/setup.zsh
```

[1] Then you have to run the ROS2 bridge that allows the qualisys driver to properly communicate with the ROS1 drivers.

```
$ ros2 run ros1_bridge dynamic_bridge --bridge-all-2to1-topics
```

Since the **roscore** is likely running on the raspberry, you will also have to export the IP of the Pi and Rosmaster:

```
$ export ROS_MASTER_URI=http://192.168.0.171:11311
$ export ROS_IP=192.168.0.104
```

2.2.3 IMU-drivers

To read the data from the IMU, their drivers first have to be read. These are written in C++, but are compatible with a ROS-system written in Python or Matlab. Put the imu_driver package inside your workspace **src**-directory and run catkin_make. After the package is built, first source the bash file and then simply run the package.

```
$ source devel/setup.bash
$ rosrun imu_driver imu_node
```

The driver publishes the measurements to the topic **/imu**.

[1] $ROS_DISTRO may vary, but is most likely **foxy** for ROS2 and **noetic** for ROS1

2.3 ROS with Python

The python based software for CSEI consists of 5 premade packages; **controller**, **observer**, **messages**, **gain_server**, **simulator** and **tf_publisher**. This section will document each package.

2.3.1 Prerequisites and recommended software

Required:

- Python 3.8.3 +

Recommended:

- Visual Studio Code with the **Remote SSH** extension. This enables you to SSH directly in to raspberry via VS Code, allowing you to edit files on the pi in the text-editor.
- A computer running linux. This is not required as most software will run on the Raspberry Pi, but can be beneficial for testing and visualizing.

2.3.2 Sourcing your ROS-distribution

Before you do anything with ROS, you need to source the installation. On the RP this is done automatically, but in case you are working on your own computer the following steps are necessary:

```
$ source /opt/ros/DISTRIBUTION/setup.bash
```

2.3.3 Building a catkin workspace

To run ROS-packages we must first define a worspace for them on the Pi. SSH onto the pi and run

```
$ mkdir —p workspace/src; cd workspace
$ catkin_make
```

This should build your catkin workspace, and leave you with three sub-directories; **build**, **devel** and **src**. All ROS-packeges are placed in the **src**-directory. Everytime you add a package to the workspace, rebuild it using catkin_make.

2.3.4 Running packages

To run your packages, first make sure you have sourced the setup.bash file:

```
$ source devel/setup.bash
```

Then, to run the package use

```
$ rosrun <package_name> <node_script>.py
```

where <package-name> is the name of the package you want to run, e.g "controller", and the <node-script>.py is the python script that initialisez the rosnode, e.g "ctrl_node.py".

2.3.5 The controller package

The first ROS-package is the *controller*-package. It contains relevant tools and scripts to build guidance systems, regulators and thrust-allocation. In total, the package contains six scripts.

math_tools.py

As the name implies, math_tools.py contains functions related to math-operations. For example computing Rotation-matrices, converting quaternions, etc. The user should add to this file according to their needs, and import the functions in other relevant files.

lib.py

lib.py contains the tools related to extracting and publishing the data from the different ROS-nodes in the system, as well as the functions for initializing and destroying the controller node. The tools are class-based, with one object for every relevant parameter in the control system. Each object contains methods for publishing new information, retrieving information from other nodes. In total there are five classes:

- **PS4**: This object subscribes to the topic **/joy** and is automatically updated with inputs from the DualShock4 controller. Each button input is mapped to a corresponding variable, which can be called by in the pattern of <object.variable>. I.e ps4.R2 returns the current signal from the R2 button.

- **UVector**: The UVector object publishes the inputs to the vessel actuators. After computing u, call the function UVector.publish(u) to publish the array.

- **Observer_Converser**: The Observer_Converser subscribes to the topic **/CSEI/observer** and automatically updates every time a observer estimations are published. The estimates can be called by

```
eta_hat, nu_hat, bias_hat = observer.get_observer_data()
```

- **Reference_Converser**: Similar to the observer_converser, except that it subscribes to the /CSEI/ref topic and contains methods for publishing as well. References are extracted and published by

```
eta_d, nu_d, nu_d_dot = reference.get_ref() # Call reference
refrence.publish_ref(eta_d, nu_d, nu_d_dot) # Publish new reference
```

- **Controller_Gains**: The Controller_Gains object is updated every time the user re-configures the controller-gains. To extract the new gains, call:

```
Kp, Kd, Ki, mu, Uref = gains.get_data()
```

All objects are declared in the script, and can be imported in custom scripts at the users behest:

```
from lib import ps4, Uvector, observer, reference, Gains
```

ctrl_joy.py

ctrl_joy.py is the default control-software for the CSE1. It enables manual control of the vessel thrusters using the DS4 controller.

ctrl_stud.py

This is a skeleton file, where custom control systems are to be implemented. It contains a loop() function, which operates similar to a main function. All function-calls should be handled inside this function.

ctrl_node.py/ctrl_joy_node.py

These are the scripts that initialize and launch the ROS nodes, and are generally not meant to be reconfigured. They initialize the nodes at 100 Hz and call the loop()-function from either ctrl_joy or ctrl_stud.

To manually run the package using either ctrl_joy or ctrl_stud use:

```
rosrun controller ctrl_joy_node.py # For ctrl_joy
rosrun controller ctrl_node        # For ctrl_stud
```

2.3.6 The observer package

The observer package handles everything related to observer implementation, and is very similar in structure to the controller package. It has a total of four scripts, one being a math_tools.py identical to the one in controller.

lib.py

lib.py contains the tools related extracting and publishing the data from the different ROS-nodes in the system, as well as the functions for initializing and destroying the observer node. Like with the controller package, The tools are class-based, with one object for every relevant parameter in the control system. Each object contains methods for publishing new information, retrieving information from other nodes. The observer package utilizes the same UVector-class as the controller, along with a couple new.

- qualisys: This object subscribes to the **/qualisys** topic, and retrieves the position the system captures of the vessel. To retrive the data call on the method:

  ```
  eta_hat = qualisys.getQualisysOdometry()
  ```

- Observer_Converser: Identical to the object described in the controller package. Also contains methods for publishing the observer data:

  ```
  eta_hat, nu_hat, bias_hat = observer.get_data()          # Return
     newest estimate
  ```

```
observer.publish_observer_data(eta_hat, nu_hat, bias_hat) # Publish
    computed estimate
```

- Observer_Gains Retrieves the injection gains for the observer in a similar manner as Controller_Gains.

```
L1, L2, L3 = Gains.get_observer_gains()          # NB! These are np
    .arrays with n=3
```

obs_stud.py

Skeleton file for implementation of observer. All user-defined functions must be called in the right order in the loop() function.

obs_node.py

Initializes the observer node

2.3.7 The gain_server package

gain_server utilizes another ROS-package called dynamic reconfiguration. This enables the user to reconfigure parameters in a ROS-node during runtime. For the Enterprise, this allows for dynamic tuning of the control-system gains. To run the package:

```
$ rosrun gain_server server.py
```

To tune parameters from the command-line use:

```
$ rosrun dynamic_reconfigure dynparam set gain_server parameter_name value
```

Alternatively, it is possible to create a .yaml file with all parameters which can be edited and subsequently uploaded to the node. To create the file, and load the new parameters run:

```
$ rosrun dynamic_reconfigure dynparam dump gain_server gains.yaml
$ rosrun dynamic_reconfigure dynparam load gain_server gains.yaml
```

Parameters

Currently, the package supports a fixed set of parameters. These are observer injection gains $L1, L2, L3$ as well as the controller gains Kp, Kd, Ki, μ and $Uref$. If other parameters are needed these must be added to the **gains.cfg** file in the cfg-directory.

GUI-Setup

If using the GUI-laptop in the MC-Lab, the gains can be tuned via a GUI. To open it, run the following command from the terminal:

```
$ rosrun rqt_gui rqt_gui —s reconfigure
```

2.3.8 Launch files

Instead of manually activating each node, you can initialize several at the same time using a launch file. These can be found in the folder **launch** in each package.

```
$ roslaunch package_name launch_file
```

2.3.9 Usefull ROS-commands

The following command-line tools are usefull when running a ros system:

1. `$ rostopic list`

   - Shows a list of all ROS-topics

2. `$ rosnode list`

   - Shows a list of all active ROS-nodes

3. `$ rostopic echo topic`

   - Prints the messages and data sent to each topic to the command-line window

Bibliography

Analog Devices, 2019. ADIS16364 Data Sheet. Technical Report. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ADIS16470.pdf.

Bjørne, E.S., 2016. Nonlinear Adaptive Motion Control and Model-Error Analysis for Ships-Simulations and MCLab experiments. Master's thesis. NTNU.

Mykland, A., 2017. Low-Cost Observer and Path-Following Adaptive Autopilot for Ships. Master's thesis. NTNU.

Orsten, A., 2014. Automatic Reliability-based Control of Iceberg Towing in Open Waters. Master's thesis. Norwegian University of Science and Technology.

Orsten, A., Norgren, P., Skjetne, R., 2014. LOS guidance for towing an iceberg along a straight-line path, in: Proceedings of the 22nd IAHR International Symposium on ICE 2014 (IAHR-ICE 2014).

Sandved, F., 2015. Remote Control and Automatic Path-following for C/S Enterprise I and ROV Neptunus. Master's thesis. Norwegian University of Science and Technology.

Skåtun, H.N., 2011. Development of a DP system for CS Enterprise I with Voith Schneider thrusters. Master's thesis. Norwegian University of Science and Technology.

Skåtun, H.N., 2014. CS Enterprise I. Video. Http://www.youtube.com/watch?v=MiESJsIZO04.

Tran, N.D., 2013. Development of a modularized control architecture for CS Enterprise I for path-following based on LOS and maneuvering theory. Specialization project.

Tran, N.D., 2014. Line-Of-Sight-based maneuvering control design, implementation, and experimental testing for the model ship C/S Enterprise I. Master's thesis. Norwegian University of Science and Technology.

Valle, E., 2015. Marine Telepresence System. Master's thesis. Norwegian University of Science and Technology.