CyberShip Enterprise I

User Manual

2022.09.04

NTNU – Trondheim
Norwegian University of
Science and Technology

Faculty of Engineering Science and Technology

Department of Marine Technology

Contents

I Technical description **2**

1 Hardware **3**

1.1 Introduction to CSE1 . . . 3

1.1.1 Literature . . . 3

1.2 Actuators . . . 4

1.3 Power system . . . 5

1.4 IMU . . . 6

1.5 Control system . . . 7

1.5.1 RPi . . . 8

1.5.2 ESC . . . 8

2 Software **9**

2.1 Introduction . . . 9

2.2 Driver setup . . . 9

2.2.1 Actuator drivers . . . 9

2.2.2 Qualisys drivers . . . 10

2.2.3 IMU-drivers . . . 10

2.3 ROS with Python . . . 10

2.3.1 Prerequisites and recommended software . . . 10

2.3.2 Sourcing your ROS-distribution . . . 11

2.3.3 Building a catkin workspace . . . 11

2.3.4 Running packages . . . 11

2.3.5 Template workspace . . . 12

2.3.6 Node-structure 12
2.3.7 Simulator 13
2.3.8 Common_tools package 13
2.3.9 Messages 14
2.3.10 The gain_server package 14
2.3.11 Launch files 16
2.3.12 Useful ROS-commands 16

Bibliography **18**

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.

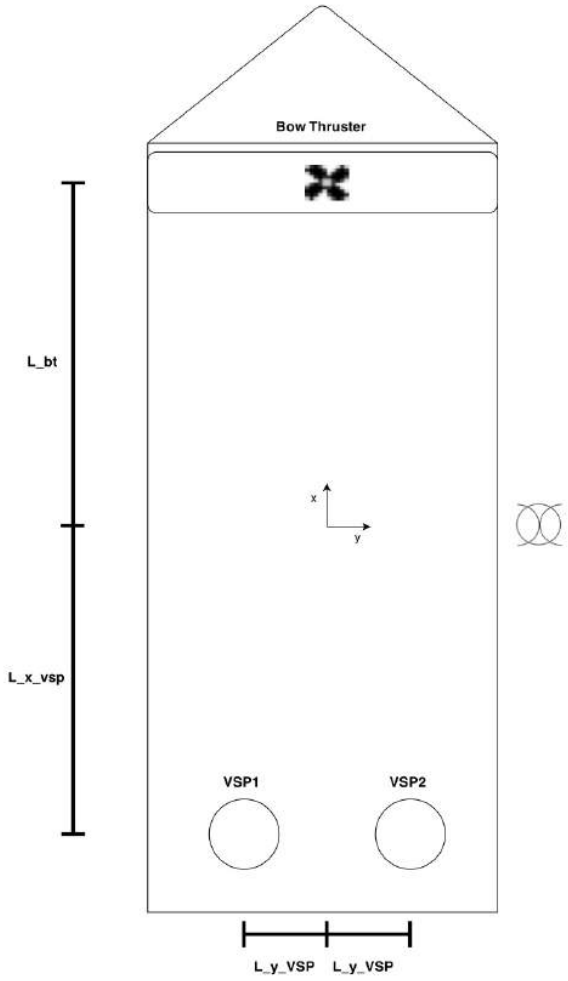


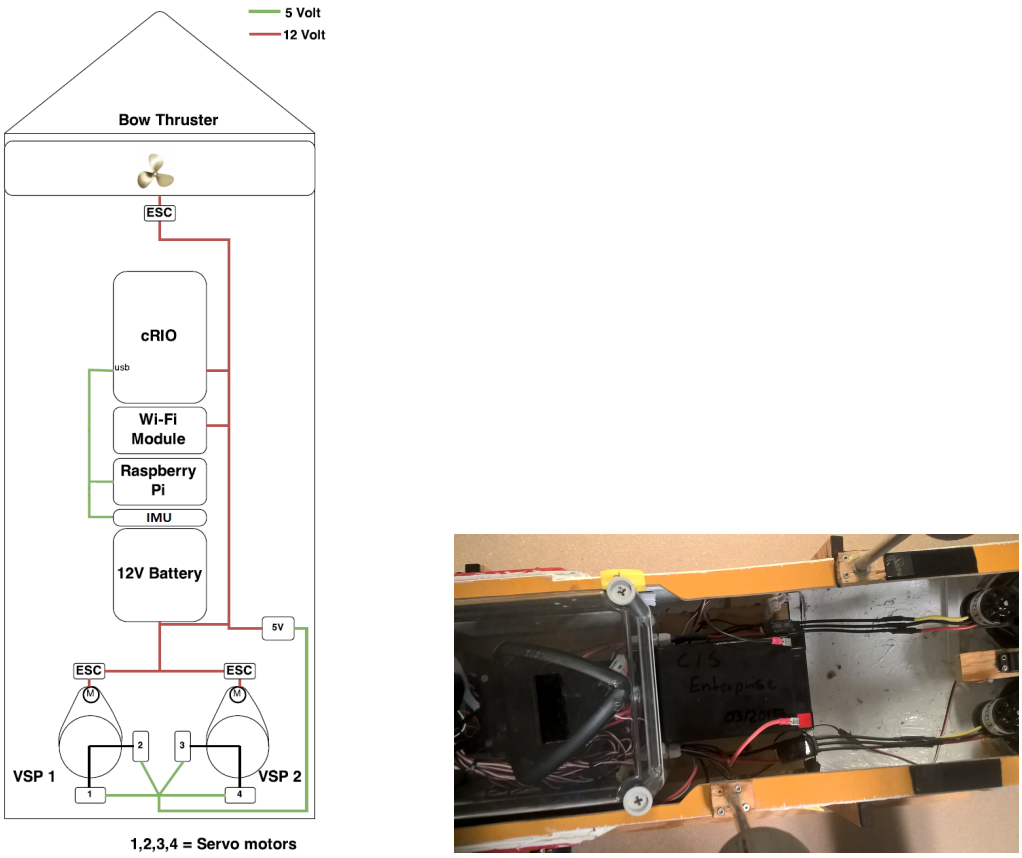
Figure 1.1: Position of actuators. Adapted from Valle (2015)

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

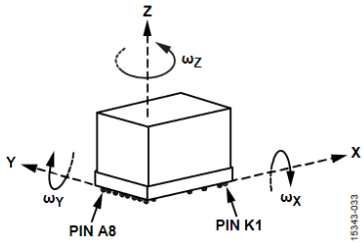
Figure 1.2: Battery system

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 can be found in the Github repository for the MC-Lab. After downloading the package, you can choose to run it from a external computer or on the onboard RPi. To activate it source your workspace, and run the command

```
$ roslaunch mocap_qualisys qualisys.launch
```

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**.

2.3 ROS with Python

A number of base python-packages have been made to help build control-system for the CSE1. These can all be found on the NTNU-MCS Github repository. Simply download the packages you needed for your own project. This section will provide an outline of different packages, the overall architecture of a ROS workspace as well as the basic knowledge required to run a ROS control system.

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 "controller.py".

2.3.5 Template workspace

A template workspace consisting of several packages is provided to make developing control systems simpler. The workspace, denoted **ws_templates** in the Git-repository, consists of a total of five ROS-packages corresponding two traditional modules in a maneuvering DP-system. These are named:

- controller
- guidance
- observer
- thrust_allocation
- simulator

Additionally, there are three packages that provide supporting tools and conventions:

- gain_server
- messages
- common_tools

2.3.6 Node-structure

The first four nodes; *controller*, *guidance*, *observer* and *thrust_allocation* are generic, skeleton packages. Their purpose is to provide a base to build a control-system on, so they only contain the necessary functionality to execute ROS-nodes. The main functionality is left to the user to implement. All the packages follow the exact same structure.

Within every package there is a *src*-directory where a single python script is placed. This is where you will write and call your code. Listing 1 shows an example of a ROS-node that maps the inputs from a PS4 controller to forces in each of CSE1 thrusters.

Quickly summarized, you write your functions above the

```
if__name__ == 'main'
```

statement. These functions should then be called in the while loop located inside. The two statements before the while loop are functions for initiating the node, and setting the refresh rate, or frequency, that we want to run the node at. This is predefined in all templates so that you only have to worry about writing the functions and calling them inside the while loop.

Fully fleshed out packages are also available on Github, if one just needs a working control system for experiments without any implementation. These can be found in the folders of *DP_Lab* and *DP_Nom* and consist of a backstepping manuevering regulator, a DP-observer, thrust allocation schemes as well as a path-parameterization guidance function.

2.3.7 Simulator

The simulator package contains a full Model-in-the-loop ROS implementation of the CSE1. It is based of the low-speed dynamics, and can be used for testing the implemented control systems outside of the MC-lab.

The simulator subscribes tp feedback from the topic **CSEI/u**, to model the signal flow on board the vessel itself. Upon receiving a signal from the topic, it will compute the generalized forces and new position of the vessel.

Note that the model is very simple, and does not take in to account environmental forces such as current or wind.

2.3.8 Common_tools package

As seen in Listing 1, the templates import from a module called **common_tools**. This is where you can find all the tools for publishing and subscribing to the relevant signals in your system, as well as the functions that initialize nodes. The tools are object-oriented, with every relevant signal pertaining to an object in the library. Each object contains methods called publish, and callback.

Every time a subscriber detects that a new signal is published to its topic, it calls the callback function in our object, and updates its class variables. You can then retrieve these by using a get-method, or just calling the variable directly through the object.

```
u = u_data.get_data() # Retrieve actuator commands
```

```
u = u_data.Udata        # Retrieve acutator commands
```

The publish method takes an input, and publishes it to the relevant topic.

```
u_data.publish(u) # Publish actuator commands
```

All objects are declared and imported into the relevant code-templates from the **lib** submodule . Subscriptions to the different topics are all handled inside these classes and the initialization nodes.

It is important that you at any time keep the common_tools package inside your workspace. The package also contains a submodule called **math_tools**. This contains some predefined mathematical functions that may be useful for the project. Rotation matrices and conversions between euler-angles and quaternions are examples of tool you can find here. Feel free to add more functions here as you please, and import them to your nodes. This makes for cleaner and more structured code.

2.3.9 Messages

While ROS provides many different messaging formats, some times these may be unsuitable to the needs of the system. One might, for example, wish to store the 9 estimated states of the DP observer in a better format than a generic array. That is where custom messages enter. Custom message templates are all stored in the message-package, and can be imported to any other module in the workspace.

2.3.10 The gain_server package

To make tuning your controller and observer more effective, tools for dynamic tuning are provided in the form of the ROS-package gain_server. This allows you to change the gains in real time, rather than having to hard-code the gains and then stopping and starting your simulation every time you want to change them. To activate the node, and GUI use the following steps:

- Make sure you have imported the Gains object from lib.py. It is built like most other objects and contais get-methods for extracting gains, along with callback-functions.

- In the **observerInitNode**() or **controllerInitNode** uncomment the line starting with *gain_client*.
- To run the node use the command:

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

  NOTE! If you use the dynamic tuning, you should always activate this node **first** or you may experience errors. This is because the observer and controller nodes will expect values to be there when they are not. When the node is activated, the initial gains should be printed to your terminal

- In another terminal, activate the GUI:

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

  This should open a program that looks like this:

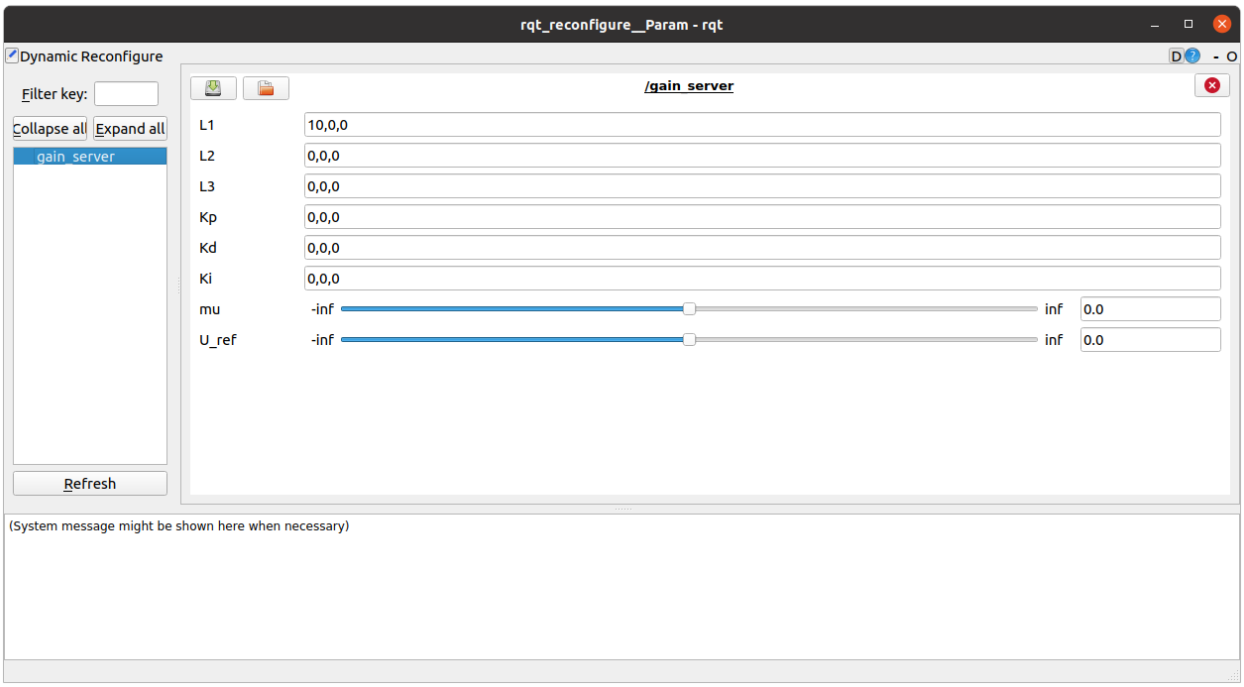


Figure 2.1: Tuning GUI

Each field or slider here corresponds to a gain in either the observer or controller. Not all the variables are relevant either. The last to are single float values, while the first six are meant to represent the elements on the diagonal of a 3×3-matrix. Change these to tune you controller.

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.

2.3.11 Launch files

As a project progresses, more and more nodes are typically added. The process of activating nodes can therefore become more tedious as the complexity increases. To avoid having to open new command-line windows for every node, we can instead use *launch-files* to activate multiple nodes simultaneously. A launch file is easy to create, and is placed in the *launch* directory of a package.

To run a launch file we use the following command:

```
$ roslaunch <package_name> my_launchfile.launch
```

Replace package_name with the name of the package you placed your launch file in. All launch files have the .launch ending (e.g DP-system.launch).

2.3.12 Useful 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

```python
#!/usr/bin/env python3
import rospy
import numpy as np
import math
from nav_msgs.msg import Odometry
from std_msgs.msg import Float64MultiArray
from common_tools.lib import ps4, tau, u_data, controllNodeInit, nodeEnd
from common_tools.math_tools import *

### Write your code here ###

# Example functions for manuevering with the DS4 controller
def saturate(u):
    """
    Saturate ensures that the input to the actuator remains bounded to the interval [-1,
    """
    if u > 1:
        u = 1
    elif u < -1:
        u = -1
    return u

def sixaxis2thruster(lStickX, lStickY, rStickX, rStickY, R2, L2):
    """
    sixaxis2thruster() directly maps the sixaxis playstation controller inputs
    to the vessel actuators.
    """
    ### Acutator commands ###
    u1 = -0.5*(L2 - R2)
    u2 = saturate(math.sqrt(lStickX ** 2  + lStickY ** 2))
    u3 = saturate(math.sqrt(rStickX ** 2 + rStickY ** 2))


    ### VSD angles as described in the handbook ###
    alpha1 = math.atan2(lStickX, lStickY)
    alpha2 = math.atan2(rStickX, rStickY)

    u = np.array([u1, u2, u3, alpha1, alpha2])
    return u



if __name__ == '__main__':

    node = controllNodeInit()
    r = rospy.Rate(100)

    while not rospy.is_shutdown():
            # Handle calls to methods or functions as below
        u = sixaxis2thruster(ps4.lStickX, ps4.lStickY, ps4.rStickX, ps4.rStickY, ps4.R2,
```

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.