WP2 Autonomous Platform for Research and Education

Experimental Automatic Steering Control in a real-scale boat

https://projectfast.nl/

This project is made possible by the European Regional Development Fund within the framework of REACT-EU

Het project F.A.S.T. is mede mogelijk gemaakt door het Europees Fonds voor Regionale Ontwikkeling in het kader van REACT-EU

Contents

1 Introduction 3

2 Examples of ship control platforms found in the literature 4

3 Use Case description and experimental steps 5

4 System Architecture 6
4.1 Software setup . . . 6
4.2 Multiprocessing Python code . . . 7
4.3 Reference Track Initialization . . . 7
4.4 GPS (serial) . . . 8
4.5 Wind sensor (serial) . . . 8
4.6 Real-Time Control algorithm . . . 8
4.6.1 Heading deviation from reference track . . . 8
4.6.2 Calculating heading setpoint: . . . 8
4.7 Steering actuator (serial or Modbus) . . . 9
4.8 Speed control (serial) . . . 10
4.9 HMI - Human Machine Interface (TCP/UDP) . . . 10
4.10 Logging and Plotting . . . 10

5 First system test in Blik op Water 11

6 Conclusions and Future work 13

A Code extract: Main steering control loop 15

B Bearing and distance between two points 16

C Finding the shortest distance between a point and a line segment 16

D Control architecture used in *Cogge* 18

E Communication framework used in *Maverick* 19

F Real-time monitor during sailing 20

Experimental Automatic Steering Control in a real-scale boat

A. Caballero-Rosas[1], M. Brederveld[1], and R. van de Pijpekamp[2]

[1]HZ University of Applied Sciences
[2]TMC Mechatronics BV

November, 2023

Summary

The main goal of Work Package 2 of the Field Lab Autonomous Sailing Technology (FAST) project is to realize an autonomous shipping platform to be used for research and education purposes. To attain this goal, the HZ University of Applied Sciences teamed with its main project partner in this journey, the Research lab. Autonomous Shipping (RAS) based at the Delft University of Technology. Several streams of work were defined in the early stages of the project tackling different aspects of autonomous shipping: 1) Upgrading lab autonomous fleets used in the Master curriculums or Maritime technology and Mechatronics; 2) Applied research on machine object recognition (aka, situational awareness); and 3) creation by students or staff of small and real scale boats with track pilot capabilities and automatic control systems. This report describes the key design aspects of a real-scale automatic steering boat and the results of the first field experiments. It also sets the next steps for the continuation of this track within the HZ University of Applied Sciences.

1 Introduction

Autonomous shipping is expected to deliver multiple benefits to the domain of shipping. It is expected that not only will it be key in realizing a decrease in crew costs, but also in generating fuel savings due to improved aerodynamics, increasing crew and cargo safety due to reduced human errors, and enabling flexibility and optimization opportunities in logistics chains (Akbar et al., 2021). Even though the expected benefits of this technology are recognized in the literature, and there is a shared consensus among experts, it is also known that the wide adoption by industry is slow. Experts have demanded more evidence that this innovation can generate economic benefits and that it will provide an equal or even better level of safety when compared to conventional shipping.

The expected role of knowledge institutions in advancing this technology is to continue conducting research and educational projects in real-scale platforms. Not only would this new technology be further improved and adopted, but also future professionals can interact with it. With this motivation, this study group developed an automatic control of speed and direction which will be installed in a 4-person boat, called Blik op Water, property of the Water Management program at the HZ University of Applied Sciences. This platform will be the basis of subsequent autonomy development assignments and experiments conducted by staff or students of this institution.

Based on the overall architecture of an autonomous ship in Figure 1, the controller of the autonomous vessel uses sensors to obtain information on the position, speed, heading, and environmental information such as wind speed. Moreover, another subsystem will find information on obstacles in the surroundings of the ship. Based on the collected information, optimal paths to follow will be determined. Finally, commands are transmitted accordingly to actuators for autonomous navigation (Chen et al., 2016). The work presented in this article refers in particular to the yellow sections of Figure 1, namely, sensors, actuation, and motion control.

In addition, the platform aims at enabling full transparency when assessing system performance. For this reason, a log file and monitoring system for tracking relevant variables will be proposed. Enabling transparency can become a key input to demonstrate that a system design complies with regulations, and in the case of non-compliance, it should indicate where the gaps are.

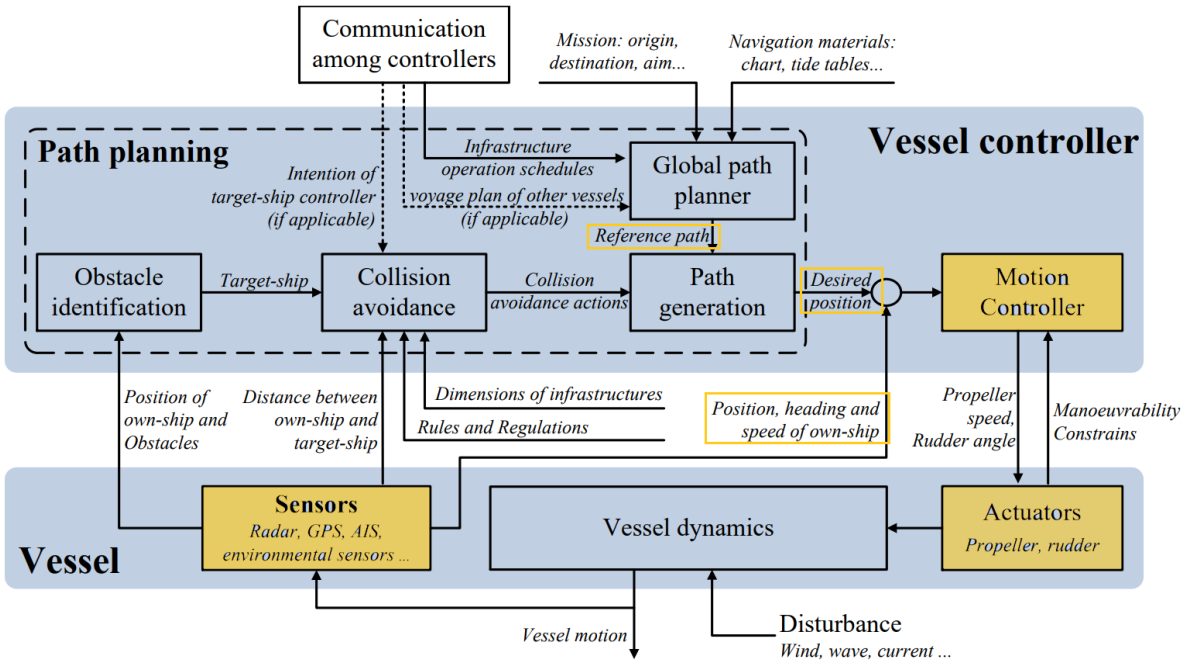


Figure 1: The overall architecture of an autonomous vessel. Adapted from Chen et al., 2016

This document continues as follows: Section 2 makes a brief overview of similar approaches used in other research projects with similar objectives. Section 3 describes the main use case for this platform. Section 4 explains the software architecture and its control algorithm. Results and conclusions are elaborated in Sections 5 and 6 respectively.

2 Examples of ship control platforms found in the literature

Several open-source implementations of track pilots and remote-operated vessels are found in the literature. Peeters et al. (2020) chose the open-source cross-platform software suite named Mission Oriented Operations Suite (MOOS) for the development of their Cogge autonomous vessel. This MOOS software provides internal asynchronous publish-subscribe communication between MOOS Applications (MOOS-Apps). Peeters et al. (2020) used a marine-oriented expansion of MOOS called MOOS Interval Programming, or MOOS-IvP. MOOS-IvP separates the vehicle navigation and control parts from its autonomy system, enabling independence across all these subsystems. In this system, the user can define a set of behaviors that the vessel should follow, for example: follow waypoints and maintain a certain speed. Next, the system requires information such as the current heading and speed (fed by GNSS and IMU sensors) to determine the desired heading and speed for the ship to reach the following waypoint. Peeters et al. (2020) implemented a Proportional Integral Derivative (PID) controller to manage the Cogge vessels in two degrees of freedom on the water plane. Figure 10 in Appendix D displays the main components of the control hierarchy in the work from Peeters et al. (2020).

Brushane et al. (2021) implemented a communication and control system for an autonomous boat platform using the OpenDLV software ecosystem and its microservice-based architecture. This architecture is characterized by small, lightweight processes and decentralized control, resulting in services that are integrated into the system in a way that changes to one service do not require changes to another [1].

The Research Autonomous Shipping (RAS) lab in the TU Delft deployed the Robotic Operating System (ROS) in their Grey Seabax and Tito Neri scale models. These ships use Model Predictive Control (MPC) and neural networks-based adaptive control, and their node communication is enabled via a router using WiFi (Haseltalab et al., 2020). The purpose of ROS is to allow the ships to share their state with the other nodes in the network (e.g. to the monitoring control station and/or the other ships in the same

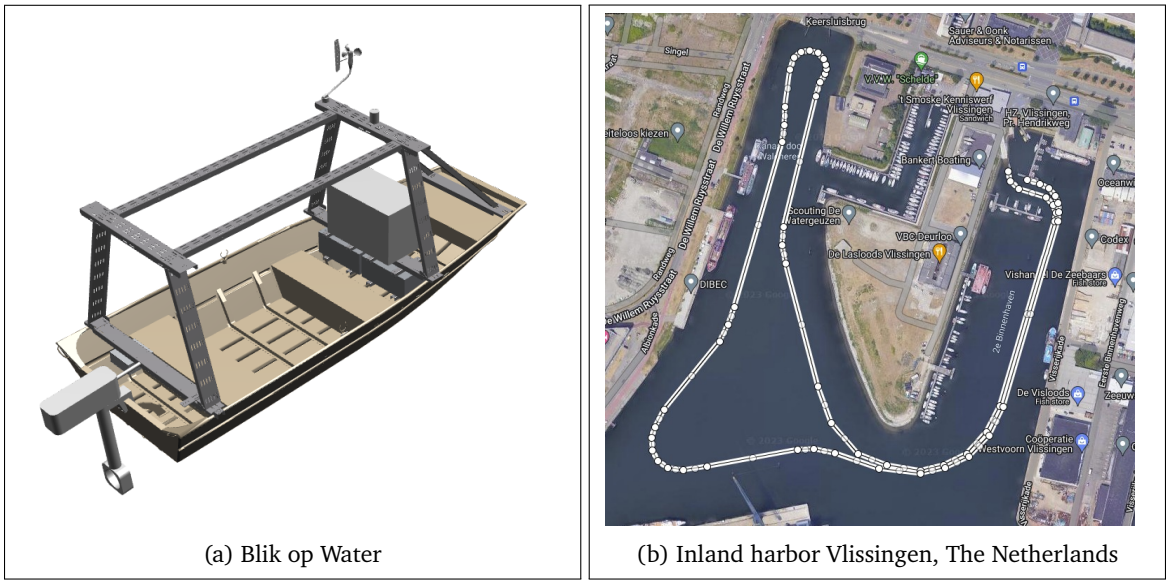
[1] https://opendlv.org/index.html

network). ROS is a platform that allows the connection of different algorithms and sensors on a local network, meaning that different devices can send data while being recognized by all the agents involved in the network. In ROS, every node can publish or subscribe to a topic that contains data and a device can be designed to have multiple nodes that subscribe and publish to different or the same topics. A key element in this network is the ROS master, which facilitates communication in the ROS network by keeping track of all active ROS entities. Every node needs to register with the ROS master to be able to communicate with the rest of the network[2].

The implementation approach in our study relates mostly to the work of Zhang et al. (2023). They developed a test bed to evaluate technology for autonomous sailing. There are some differences between our prototype and the one of Zhang et al. (2023): they developed a catamaran cargo vessel with azimuth thrusters, whereas we experimented with a 4-person boat with rudder-propeller actuation. The hardware architecture in Zhang et al. (2023) has a modular design, which allows for versatility in different sensor setups. The central component is the message-oriented middleware: Neural Autonomic Transport System (NATS). All communication between the various components takes place via the NATS. For example, the NATS translates input from the sensors into information on the current vessel status and passes this information on to the controller. The controller then determines the desired control actions and passes them back to the NATS. The NATS communicates these control actions to the actuation control system. Through a publish/subscribe middleware, the test bed of Zhang et al. (2023) treats sensors as external devices and therefore does not rely on integrated onboard sensors. The interactive communication framework of Zhang et al. (2023) is displayed in Figure 11 in Appendix E.

3 Use Case description and experimental steps

The Blik op Water is a boat with an aluminum hull and with 5 meters in length. This boat will be equipped with an automatic control system. The boat's performance, the system's ability to correct its course, and its overall navigational capabilities will be assessed. Blik op Water is originally suitable for navigating with a crew of a maximum of 4 people and it will include an electric engine with a battery pack of 48V/40 Amp. The system will have the possibility to easily switch between automatic control mode and manual control mode, and it will include an emergency button to be activated in special circumstances.



(a) Blik op Water

(b) Inland harbor Vlissingen, The Netherlands

Figure 2: Experiment scenario: boat type and test route

Once the control system is developed, the following activities will take place during the experimentation phase.

[2] https://www.mathworks.com/help/ros/ug/connect-to-a-ros-network.html

1. Before departure, the crew will input the predetermined waypoints into the control system. The waypoints will be defined by their coordinates in latitude and longitude using the WSG84 standard.
2. The crew will start the motor, and activate the automatic control system.
3. The boat will then start sailing towards the first waypoint at a predetermined speed. The control system will constantly monitor the boat's position using the GPS and make adjustments to the heading as needed to keep the boat on track.
4. The crew will continuously monitor the boat's progress and ensure that the system is functioning correctly. The crew will also be responsible for navigating the boat in case of any unexpected circumstances or obstacles, such as other boats or buoys.
5. If the crew detects any obstacles along the way, they will manually override the automatic control system and take control of the boat to avoid the obstacle.
6. Once the boat reaches the final waypoint, the crew will deactivate the automatic control system and steer the boat to the desired destination. After the system is shut off, a collection of the logs can be made for further analysis.

4 System Architecture

The control system at the heart of the autopilot, is responsible for processing the positional data received from the RTK GPS, calculating the necessary course adjustments, and sending the corresponding commands to the actuation system. The control logic is designed to minimize lateral deviation from the predefined path while taking into account the dynamic characteristics of the boat and the environmental conditions encountered.

The actuation system consists of the components responsible for executing the commands sent by the control system. It includes the boat's rudder, the throttle, and the electric motor. The actuators will receive commands from the control system and adjust the rudder and throttle positions accordingly to correct the boat's course and speed. Each subsystem of the control and actuation systems will be explained in the next subsections.

4.1 Software setup

Python is chosen as the development language for this system due to its accessibility, which makes it ideal for educational purposes. Because it's a user-friendly language, students of all levels can participate in enhancing this system. The following factors explain how the chosen setup is found most beneficial for educational purposes:

- Real-Time Control: The software's real-time control algorithm allows students to experiment with control systems, gaining hands-on experience and practical insights.
- Logviewer: An intuitive Logviewer component lets users visualize and play back-logged data, sensors, and calculations, aiding in data analysis.
- Data Replay: Users can replay logged sensor data, facilitating precise control algorithm tuning and enhancing students' understanding of system behavior.
- Cross-Platform Compatibility: The software runs on both Linux and Windows, ensuring accessibility to a wide range of users.
- Command Line Efficiency: A command line parser simplifies user interaction by allowing options to be set when starting the program from the command line.
- Customization: Users can enable/disable individual components and sensors, tailoring their learning experience.
- Hardware Independence: All sensors interface via Ethernet (TCP/IP or TCP/UDP) or USB (serial RS232 and Modbus), with automatic device detection for seamless hardware integration.

The expectation is that the developed platform will equip the students of HZ University of Applied Sciences with a versatile platform for educational exploration and real-time control, making it a valuable tool for learning and experimenting.

4.2 Multiprocessing Python code

In the Python-interpreted language, it is known that the Real-Time performance can be poor compared to other more efficient programming languages. To boost the performance of a Python program, Multi-threading or Multiprocessing is used to speed up the code execution time.

Multithreading allows slower processes to be interrupted by processes requiring more CPU time. As such, data sharing is relatively easy because there is still only one interpreter actively sharing the same memory space. Because of this single interpreter, the code can still be slow compared to multiprocessing. Multiprocessing starts a new interpreter for each process which is similar to starting multiple Python programs from within the code. This makes it relatively fast and efficient. Data sharing can only be done via shared memory because there is no other direct connection between all individual processes.

Figure 3 displays this concept and the interacting processes running in the multiprocessing environment, which are also listed below:

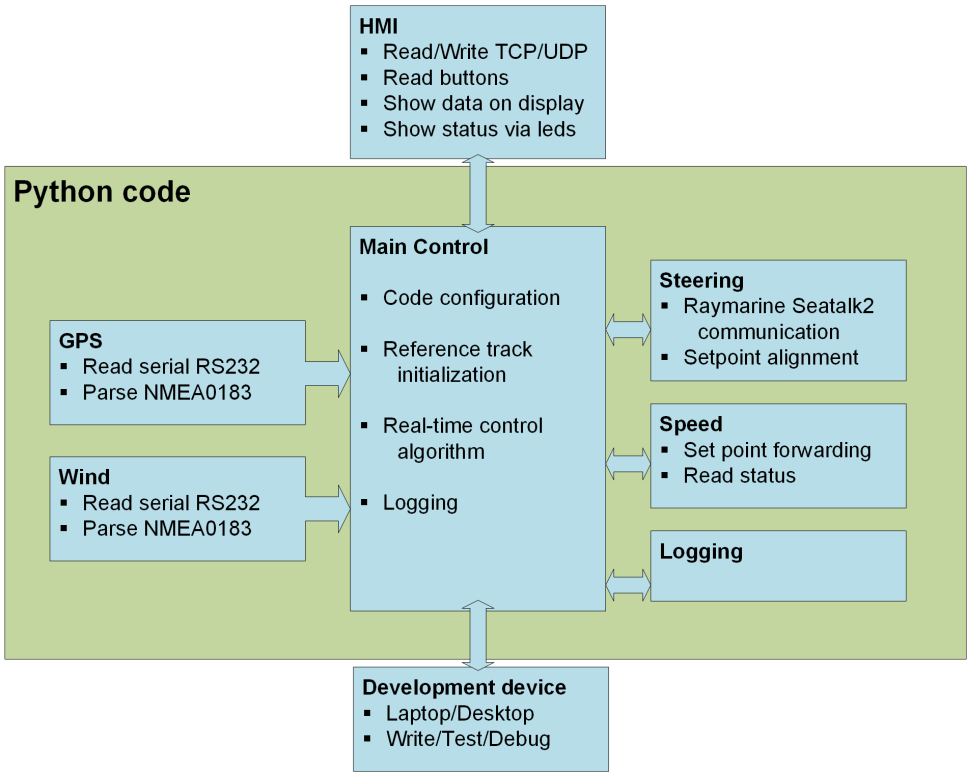


Figure 3: Software architecture

4.3 Reference Track Initialization

The waypoint records consist of WSG84 coordinates and the desired speed at each setpoint, i.e. $[Long, Lat, Speed]$. Waypoints can be generated using Google Maps, or exported from any other application that can export WSG84 coordinates. Speed setpoints have to be manually added to the coordinates table.

The reference track is also converted to X-Y coordinates in meters, using the first waypoint as 0.0m, 0.0m. The reference heading between all waypoints is calculated at the beginning of the code.

4.4 GPS (serial)

The GPS used is a dual antenna Real-Time-Kinetics (RTK) GPS, based on the Ublox ZED-F9P chipset.

Having two antennas with RTK accuracy of +/- 2 cm makes it possible to have a direct heading measurement of the vessel at all times, even at low speeds or zero speed. RTK corrections are received via a subscription service NTRIP received via 4G on board. This results in a GPS accuracy of 2 to 4 cm. Update rate of the GPS is set to 5 Hz. This is also the main clock cycle of the real-time control algorithm. The GPS Python process parses the incoming NMEA0183 and UBX messages and converts them to global variables for the other Python processes.

4.5 Wind sensor (serial)

Wind speed and direction (relative to ship heading) are available as sensor input data for the real-time control algorithm. It can be used as a feedforward to the heading/steer controller to make it more robust for cross-wind conditions. As such, sensor data is converted from incoming NMEA0183 messages and made available to the rest of the code as global variables.

Note: Signals of this sensor are captured, however, embedding it in the control algorithm is part of future projects.

4.6 Real-Time Control algorithm

The GPS data is used as primary input and clock-tick (when there is valid GPS with RTK precision). Several computations are made based on the reference track:

- Waypoint segment, the line between the waypoint passed and the next waypoint.
- Segment heading, heading between waypoint passed and next heading. See Appendix B for further explanation of this calculation.
- Next segment heading: heading between the next waypoint, and the waypoint after that (if it is not the last point on the reference track)
- Lateral deviation from reference track = nearest point perpendicular to the current segment. See Appendix C for further explanation on this calculation.

Note: It would be better to fit a spline or curve between the points, but for this first iteration, this linear approach works.

4.6.1 Heading deviation from reference track

The smoothing of the heading reference is proportional to the current segment heading and the heading of the next segment. When approaching the next segment the heading reference will be proportional to the distance remaining to the next segment averaged between the current and next segment heading. This will result in a continuous heading reference between segments.

Advancing the waypoints: When the minimum calculated perpendicular lateral deviation from the next segment is smaller than the lateral deviation to the current segment, it is assumed the waypoint is passed, and the next segment is entered. As such, the waypoints and segments will be advanced in this case.

4.6.2 Calculating heading setpoint:

Setpoint heading is relative to the vessel's current heading, with a maximum of +/- 30 degrees to the left or right of the vessel.

- Lateral deviation results in a setpoint heading that will steer the vessel toward the reference track.

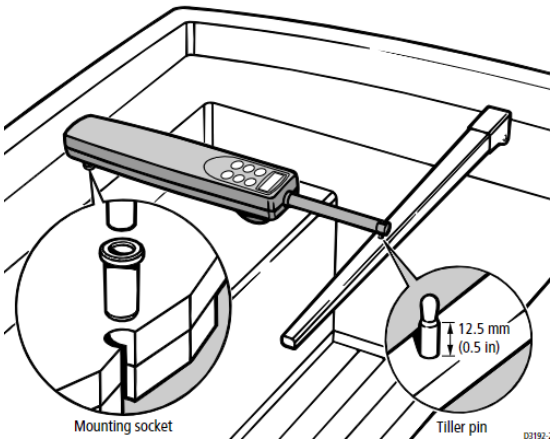
- Heading deviation will result in a setpoint heading to steer towards the correct heading.

The lateral deviation and heading deviation both have their gains. The combined setpoint for steering is an addition of the setpoints for lateral deviation and heading. Tuning these gains will result in a compromise between small lateral deviation, but more nervous steering, or larger lateral deviation but better reference heading following. As for the speed setpoint, this is taken from the current segment and sent to the speed controller of the vessel.

Note: Many other control strategies might be tested and implemented, and one of the goals of this project is to allow students to enhance the system with better algorithms and test them in real life using this platform.

4.7 Steering actuator (serial or Modbus)

A Raymarine ST2000+ tiller autopilot is used as the steering actuator (Figure 4a). This actuator is designed for marine purposes and includes an internal compass, a yaw rate sensor, and a heading controller.



(a) Raymarine ST2000+ (from user manual)

(b) Rim-drive engine, by Insumo BV

Figure 4: Steering and Propulsion actuators

The actuator can receive commands from an external system using the NMEA0183 messages or (propriety) Seatalk2 messages. The actuator is capable of following the waypoint when the NMEA0183 protocol is used. However, the lateral control accuracy is +/- 0.05 nm (+/- 93m). This is not accurate enough for the desired control behavior of the vessel (i.e., ideally +/- 1.5 m controlled lateral deviation). The direct control of the steer-actuator is more or less possible using the Seatalk2 messages, which has been greatly reverse-engineered by Dipl.-Ing. Thomas Knauf[3]. A small converter is inserted to translate to/from the physical Seatalk2 protocol to a 9-bit TTL serial protocol based on the design of AK-Homberger[4], but using Mosfets instead of the 74LS07.

When the Raymarine is in heading control mode, the internal compass is used as reference and a setpoint heading is set by the user with buttons on the device. The setpoint can also be set via the Seatalk2 messages. The internal compass is reasonably accurate but it was found during testing that it can be influenced by large metallic objects and might drift over time. The compass can be reset using Seatalk2 messages to keep it aligned with any external heading measuring system available. In our case, the RTK GPS provides an accurate heading. When sending a new compass calibration to the Raymarine ST2000+, it will adjust the internal compass but also does not accept any other messages for approximately 10 seconds.

[3] http://www.thomasknauf.de/rap/seatalk2.htm
[4] https://github.com/AK-Homberger/Seatalk-Autopilot-Remote-Control

Note: It would be easiest to send a heading setpoint to the Raymarine ST2000+ and keep the compass aligned with the measured heading coming from the GPS. Experiments have shown that losing control of heading for more than 10 seconds results in an undesirable large deviation from the reference trajectory.

New heading setpoints are accepted by the Raymarine at rates of up to 5 Hz. A new control strategy is adapted, where the control algorithm reads the current measured compass heading by the Raymarine internal compass, and adjusts the desired setpoint to be corrected relative to the internal compass. If the desired setpoint from the RT algorithm is 10 deg left, the steer-actuator Python process reads the current measured compass heading from the Raymarine and sends back a heading setpoint 10 degrees left of this measured compass heading. In this way, the error in the heading of the internal compass is compensated and results in an accurate following of the desired real-time control algorithm.

The Raymarine ST2000+ internal control algorithm seems to be a combination of measured heading and yaw rate. The measured yaw rate is used for fast correction, and the measured heading is used for the overall control target. The actuator does not have any sensor on the steering actuator to measure the extension/retraction of the actuator. When first engaged, if the vessel is heading straight with zero to little rate, the actuator position is assumed to be neutral.

Note: Combined yaw rate and heading control might be interesting to explore when a different type of actuator is used for steering. The yaw rate is available in the control algorithm based on GPS-heading measurement.

4.8 Speed control (serial)

A rim-drive electric thruster motor is provided by Insumo B.V. in the Netherlands. The controller is a SLS60-240 whose logic can be found in the Sinus Leistungssteller website[5]. This motor has an RS232 serial protocol to override the physical throttle input, so it is possible to control the speed from an external device. The controller is in speed control mode. A setpoint is therefore referenced as an RPM setpoint (indirectly) controlling the speed of the vessel.

Note: For actual speed control, an outer control loop might be added, measuring the speed via GPS and adjusting the RPM setpoint. For now, it uses an open loop speed control based on the setpoint stored in the reference track.

4.9 HMI - Human Machine Interface (TCP/UDP)

To use the system without the need for any connected laptop, an HMI box is added. This box has a small display and some LEDs showing status information regarding the sensors and control algorithm and has a couple of buttons, to enable/disable automatic track following.

The HMI box communicates via Wi-Fi to the main controller. The HMI CPU is an ESP32-based controller running micro-python. Micro-python is very similar to regular Python and therefore fits within the architecture of the complete control system, enabling students to alter the main code and the HMI using Python as the main programming language.

4.10 Logging and Plotting

When a laptop is connected, real-time data can be visualized via the main code. It will plot the most relevant sensor and control data and show some numerical data. It has more information than what is shown on the HMI display and is very useful for developing and debugging the control algorithm. An example screenshot is shown in Appendix F, Figure F.

[5] https://www.sinusleistungssteller.de/

5 First system test in Blik op Water

Section 3 described the main test scenario and steps to be executed during testing. This section provides a detailed analysis of the system's performance and results during the experiments.

Figure 5: Example planned and executed route

The results suggest the system's ability to correct the boat's course in real-time (see Figure 5). During the experiment session, multiple rounds of the same track were performed. Some of them could not be fully terminated due to oncoming traffic, which led to the suspension of the system. The results shared further below, belong to one test carried out almost completely in automatic mode. Measurements displayed in Figure 6, reveal that even if the boat maintains its course, deviations up to +/- 7 meters are observed.

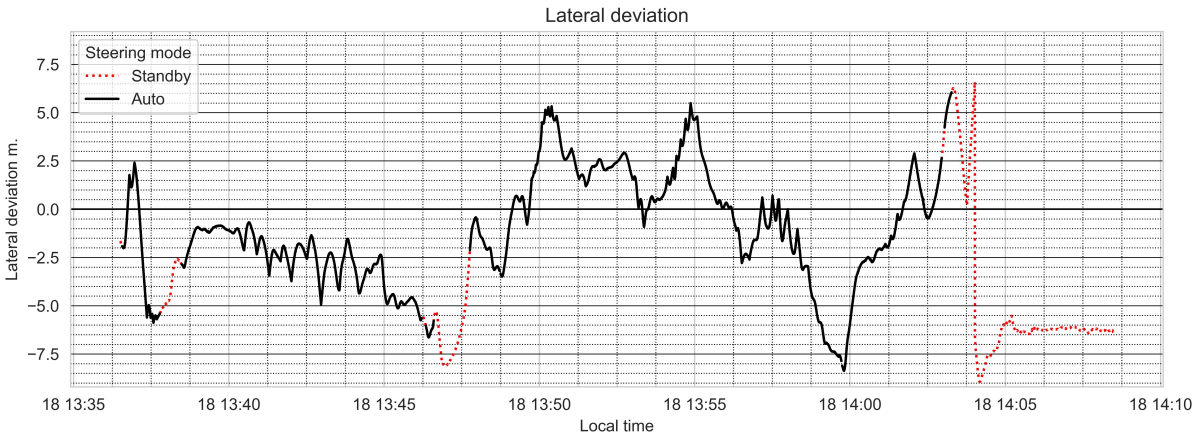


Figure 6: Lateral deviation over time

Later it was found that the control algorithm of the Raymarine ST2000+ might become unresponsive when running for a longer period and receiving setpoints via Seatalk2. A quick cycle to standby and

automatic mode will re-set the Raymarine actuator and resume accurate heading control operation. This might be automated from within the Python code (which remains a future student project). This behavior is shown in Figure 7 where the thicker line reflects a larger lateral deviation. After pressing the standby button of the Raymarine ST2000+ (which is flagged by the *Change mode* text in Figure 7), an immediate correction is observed in a reduced lateral deviation.

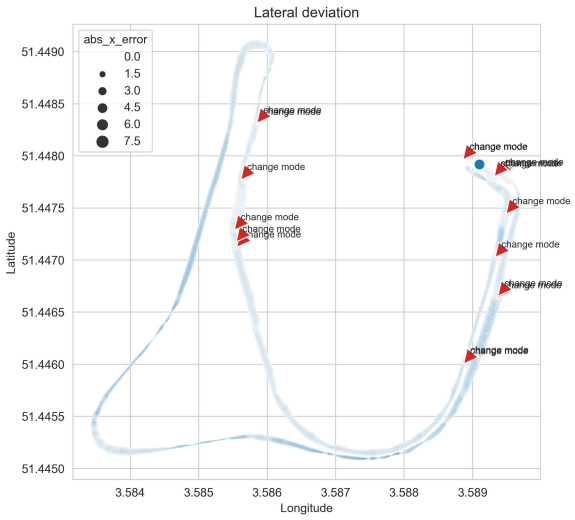


Figure 7: Observed systematic deviations corrected with Standby button

The control system displayed a very good performance when taking the 180-degree bend, close to the Keersluisbrug bridge. A video demonstration of this particular situation is available here: https://youtu.be/pYhZuHU9UoU.

Another observation relates to the ability to safely berth the boat in the last waypoints, where larger lateral deviations were observed. The reference speed set points that the team set for mooring were much lower than the speed set points for normal sailing. Given the good performance observed during normal sailing, it is believed that larger deviations are the results of more difficult manoeuvering the boat at such lower speeds. It remains a gap to be addressed, understood, and corrected in future experiments.

Beyond the ability of the system to maintain the course and heading at acceptable levels, the team experienced a very good performance of the powering system (i.e. battery). After a couple of hours of constantly sailing and testing, a minor percentage of the battery power was consumed, which speaks of the good efficiency of the motor. It must be said though that weather conditions were good and moderate wind was present. Also, the superstructure built for mounting control boxes, sensors, and batteries allowed for comfortable sailing and does not conflict with the other uses that Blik op Water has with other departments.

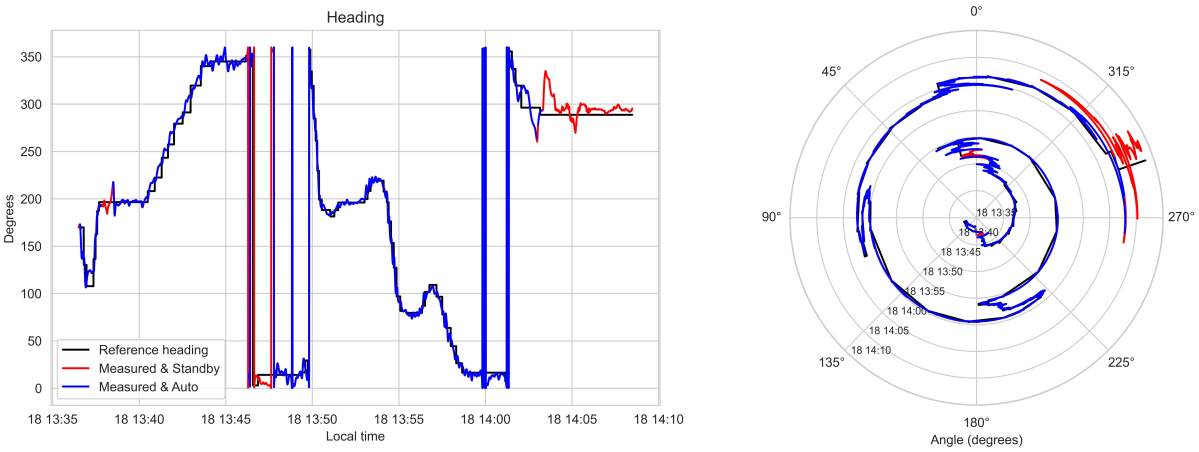


Figure 8: Heading deviation over time

Last but not least, the logfiling system lends itself to further improvement, but the amount of data that is present after sailing enables deep exploration of the system. A set of Python scripts is already available to plot the log files in the more common graphic outputs to evaluate in detail a specific experimental session.

6 Conclusions and Future work

The developed autopilot system successfully demonstrated its capability to navigate a small boat along a predefined path while automatically correcting its course based on lateral deviations detected by the RTK GPS. Despite some challenges observed with the actuator and systematic deviations, the platform does fulfill its main goal, which is to provide a solid foundation for future experimentation with different control logic and algorithms and the development of situational awareness capabilities with computer vision.

The intention here, beyond building a robust platform with acceptable performance, is to equip HZ University of Applied Sciences with more platforms to enable our study programs and lectorates to further explore and learn about automatic controls in a real-scale environment.

Future work will focus on augmenting the system's capabilities by incorporating additional sensors, such as IMU and wind sensors, to anticipate and correct for strong deviations caused by environmental factors. An idea to incorporate an affordable lidar has been also proposed to allow the system to better handle trajects under bridges, where it is known that GPS drops its accuracy.

More importantly, the intention of leveraging the platform so other groups in HZ can embed it in their education programs and lectorates is the next challenge of this project. For the moment, the first idea on the table is about the integration of the ongoing work on computer vision technology conducted by the Data Science lectorate. This will enable the development of situational awareness capabilities, enhancing the boat's ability to detect and avoid obstacles autonomously (which refers back to Figure 1 at the beginning of this document, where *object detection* and *collision avoidance* are key subsystems that make part of an autonomous boat.) Ultimately, these enhancements will contribute to the development of a more comprehensive and robust autonomous navigation system for small boats and better education capabilities in our institute.

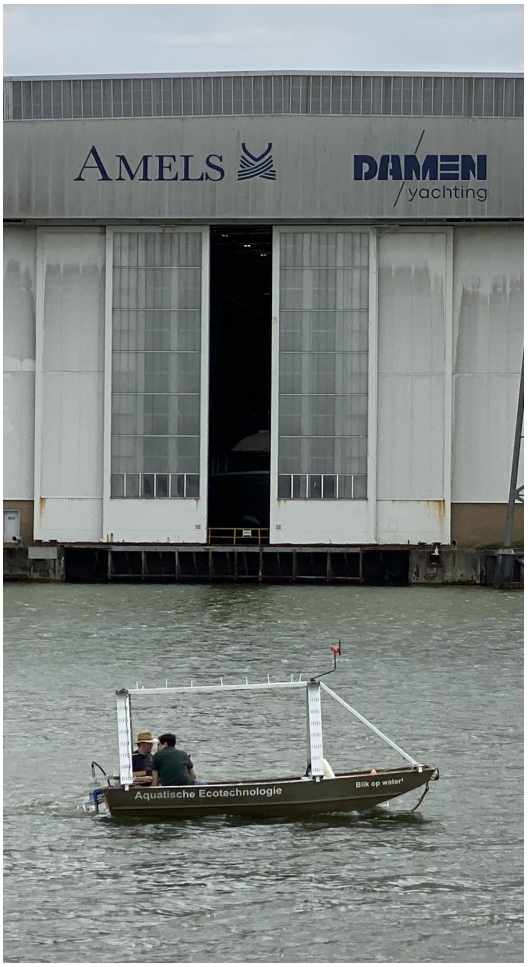


Figure 9: Man in the loop experiment

Acknowledgements

The work presented in this article is part of the Field Lab Autonomous Shipping Technology PROJ-04119 funded by the European Regional Development Fund within the framework of REACT-EU.

Special thanks to colleagues of Insumo BV, Provincie Zeeland, Art en Tech, TMC Mechatronics, CE Industries, and Maarten van Oeveren who made relevant connections to line up all these stakeholders. Special gratitude to the Water Management team who facilitated their Blik op Water for this study and accommodated it some of their courses agreeing with our development schedule.

References

Akbar, A., Aasen, A. K., Msakni, M. K., Fagerholt, K., Lindstad, E., & Meisel, F. (2021). An economic analysis of introducing autonomous ships in a short-sea liner shipping network. *Int T Oper Res*, *28*, 1740–1764. https://doi.org/10.1111/itor.12788

Brushane, F., Jämsä, K., Lafond, S., & Lilius, J. (2021). A experimental research platform for maritime automation and autonomous surface ship applications [13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2021]. *IFAC-PapersOnLine*, *54*(16), 390–394. https://doi.org/https://doi.org/10.1016/j.ifacol.2021.10.121

Chen, L., Negenborn, R., & Lodewijks, G. (2016). Path planning for autonomous inland vessels using a*bg. *9855*, 65–79. https://doi.org/10.1007/978-3-319-44896-1_5

Haseltalab, A., Garofano, V., Afzal, M. R., Faggioni, N., Li, S., Liu, J., Ma, F., Martelli, M., Singh, Y., Slaets, P., et al. (2020). The collaborative autonomous shipping experiment (case): Motivations, theory, infrastructure, and experimental challenges. *Conference Proceedings of iSCSS*.

Peeters, G., Kotzé, M., Afzal, M. R., Catoor, T., Van Baelen, S., Geenen, P., Vanierschot, M., Boonen, R., & Slaets, P. (2020). An unmanned inland cargo vessel: Design, build, and experiments. *Ocean Engineering*, *201*, 107056.

Zhang, Y.-Y., Shuai, J., Billet, J., & Slaets, P. (2023). Design and build of an autonomous catamaran urban cargo vessel. *Journal of Physics: Conference Series*, *2618*(1), 012002.

A Code extract: Main steering control loop

Algorithm 1: `Main steering control loop`

Input: $(lon, lat, speed)$, for each $waypoint_i$ where $i \in [1 \dots n]$
foreach $waypoint_i$ **do**
  Convert lon and lat to X, Y in meters, where $X_1, Y_1 = [0, 0]$;
  Compute ref. heading H_i from $waypoint_i$ to $waypoint_{i+1}$ `// based on B`
end
Initialize;
$closestPoint = 0, lateralDeviation = 0, lateralDirection = 0, segmentLength = 0$;
$distanceToSegment = 0, segmentHeading = 0$;
$rudderSetpoint = 0, yawRate = 0$;
Set $B = 30$, as blending distance `// based on B`;
Set $i = 1$, as $waypoint_i$ counter;
while *program is running* **do**
  $actualPosition$ = GPS position;
  Find segment $\overline{\text{AB}}$ such that $A = waypoint_i$ and $B = waypoint_{i+1}$;
  Find segment $\overline{\text{BC}}$ such that $B = waypoint_{i+1}$ and $C = waypoint_{i+2}$;
  Find Distance D of $actualPosition$ to nearest point of $\overline{\text{AB}}$;
  Find Distance F of $actualPosition$ to nearest point of $\overline{\text{BC}}$;
  if $F > D$ **then**
    $i = i + 1$;
    Update;
    $closestPoint, lateralDeviation, lateralDirection, segmentLength$;
    $distanceToSegment, segmentHeading$
  end
  $Delta_{heading} = H_i - H_{i+1}$;
  Normalize $Delta_{heading}$ such that $Delta_{heading} \in [-180° \dots 180°]$;
  Calculate $lateralDeviation$ of $actualPosition$ to $waypoint_i$;
  if $lateralDeviation < B$ **then**
    $relativeCovered = 1 - (lateralDeviation/B)$;
    if $relativeCovered > 1$ **then**
      $relativeCovered = 1$
    end
    if $relativeCovered < 0$ **then**
      $relativeCovered = 0$
    end
  end
  $H_{new} = H_i + (relativeCovered * Delta_{heading})$;
  Normalize H_{new} such that $H_{new} \in [0° \dots 360°]$;
  Compensate for 0 crossing;
  Calculate if the error is to the left or to the right;
  Calculate $rudderSetpoint$, correct to $[-30° \cdots + 30°]$;
  Send setpoint to steer actuator;
  Send setpoint to speed controller;
end

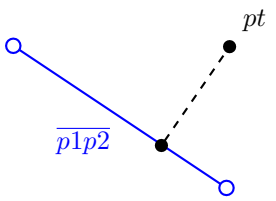
B Bearing and distance between two points

The bearing or heading angle between two points on the Earth's surface is calculated by the following expressions[6]:

- Input values are the latitude and longitude in the WGS84 decimal format of the two points.
- The function first defines the Earth's radius in meters as $R = 6377918$.
- It then converts the input latitude and longitude coordinates from degrees to radians using the $\frac{\pi}{180}$ conversion factor.
- Next, it calculates the X and Y components of the distance between the two points using the Haversine formula.
  - $X = R * \cos(lat_2) \sin(lon_2 - lon_1)$,
  - $Y = R * (\cos(lat_1) \sin(lat_2) - \sin(lat_1) \cos(lat_2) \cos(lon_2 - lon_1))$
- Finally, the function calculates the bearing (i.e. heading) between the two points using the arctan function: bearing (degrees) $= \arctan(X, Y) * \frac{180}{\pi}$.
- The bearing is returned in degrees and is adjusted to be within the range of 0 to 360 degrees: $$\text{bearing (degrees)} = \begin{cases} \text{bearing (degrees)} + 360, & \text{if } < 0 \\ \text{bearing (degrees)}, & \text{otherwise} \end{cases}$$
- The distance between the two points uses the Pythagorean theorem, dist $= \sqrt{X^2 + Y^2}$

C Finding the shortest distance between a point and a line segment

The deviation of a current point pt to a line segment defined by two points $p1$ and $p2$ is calculated by finding the closest point on the line segment, the lateral deviation, the lateral direction, the length of the segment, the distance from pt to the end of the segment, and the direction from pt to the closest point on the segment[7].



1. **Calculate Distances and Directions**
   - Calculate the length of the segment $p1p2$:
   $$\text{dst\_segment} = \sqrt{(p1_x - p2_x)^2 + (p1_y - p2_y)^2}$$
   - Calculate the distance from pt to $p2$:
   $$\text{dst\_end} = \sqrt{(pt_x - p2_x)^2 + (pt_y - p2_y)^2}$$
   - Calculate the changes in x and y from $p1$ to $p2$:
   $$\Delta x = p2_x - p1_x$$
   $$\Delta y = p2_y - p1_y$$
   - Calculate the heading of the segment:
   $$\text{dst\_heading} = \arctan 2(\Delta x, \Delta y) \times \frac{180}{\pi}$$
   Adjust dst_heading to be in the range $[0, 360]$ degrees.

[6] based on https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/
[7] based on http://csharphelper.com/howtos/howto_point_segment_distance.html

2. **Handle Case where $p1$ and $p2$ are the Same Point**

   - If $p1$ and $p2$ are the same point, then the closest point is $p1$, and the lateral deviation is the distance from pt to $p1$:

$$\text{lat\_deviation} = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

3. **Calculate the Closest Point on the Line Segment**

   - Calculate the parameter t that minimizes the distance from pt to the line defined by $p1$ and $p2$:

$$t = \frac{(pt_x - p1_x)\Delta x + (pt_y - p1_y)\Delta y}{(\Delta x)^2 + (\Delta y)^2}$$

   - If $t < 0$, then the closest point on the line segment is $p1$.
   - If $t > 1$, then the closest point on the line segment is $p2$.
   - Otherwise, the closest point on the line segment is:

$$\text{closest}_x = p1_x + t\Delta x$$

$$\text{closest}_y = p1_y + t\Delta y$$

4. **Calculate the Lateral Deviation and Direction**

   - Calculate the changes in x and y from pt to the closest point:

$$\Delta x = pt_x - \text{closest}_x$$

$$\Delta y = pt_y - \text{closest}_y$$

   - Calculate the lateral deviation:

$$\text{lat\_deviation} = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

   - Calculate the lateral direction:

$$\text{lat\_direction} = \arctan 2(-\Delta x, -\Delta y) \times \frac{180}{\pi}$$

   Adjust lat_direction to be in the range $[0, 360]$ degrees.

D Control architecture used in *Cogge*

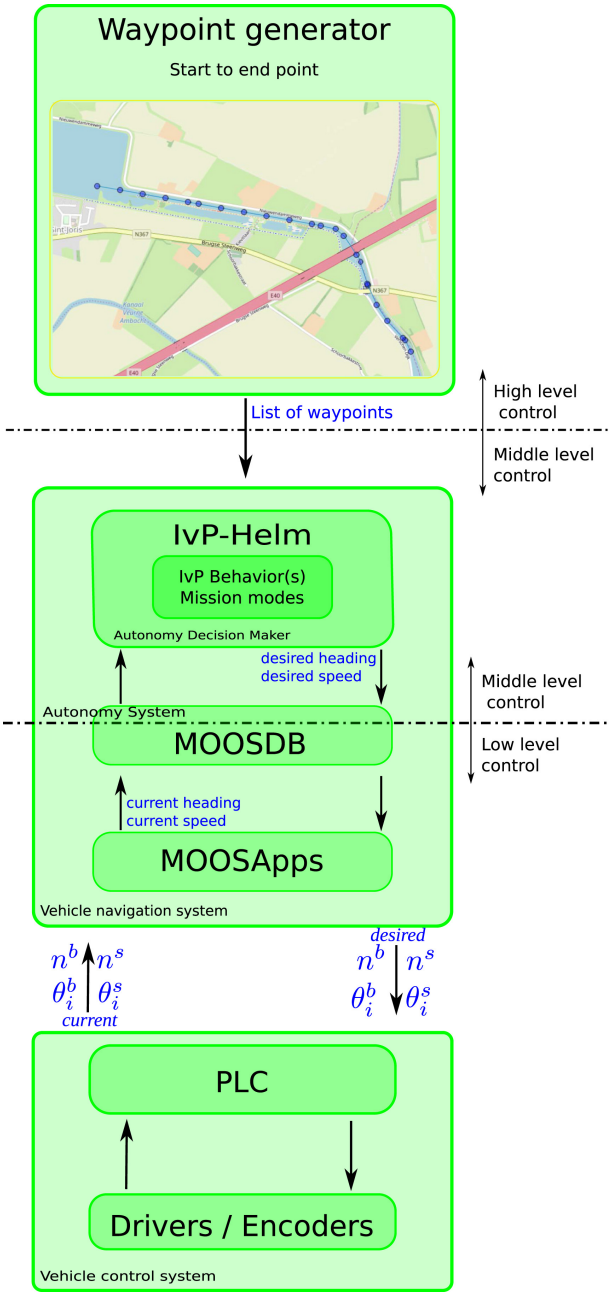


Figure 10: Control hierarchy implemented in the autonomous vessel Cogge. Taken from Peeters et al., 2020

E Communication framework used in *Maverick*

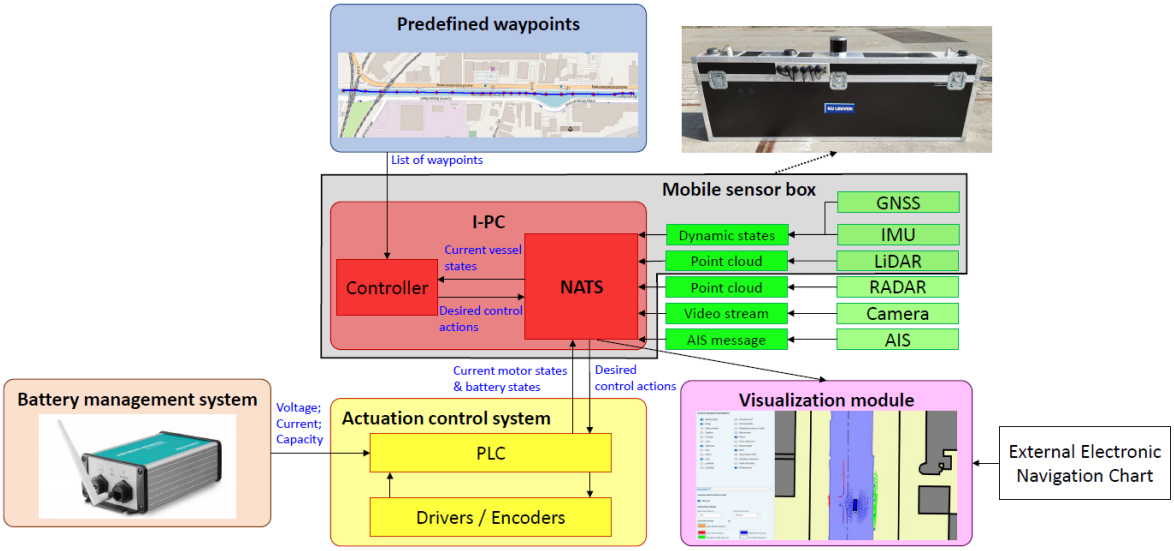


Figure 11: Interactive communication framework of Maverick. Source: Zhang et al., 2023

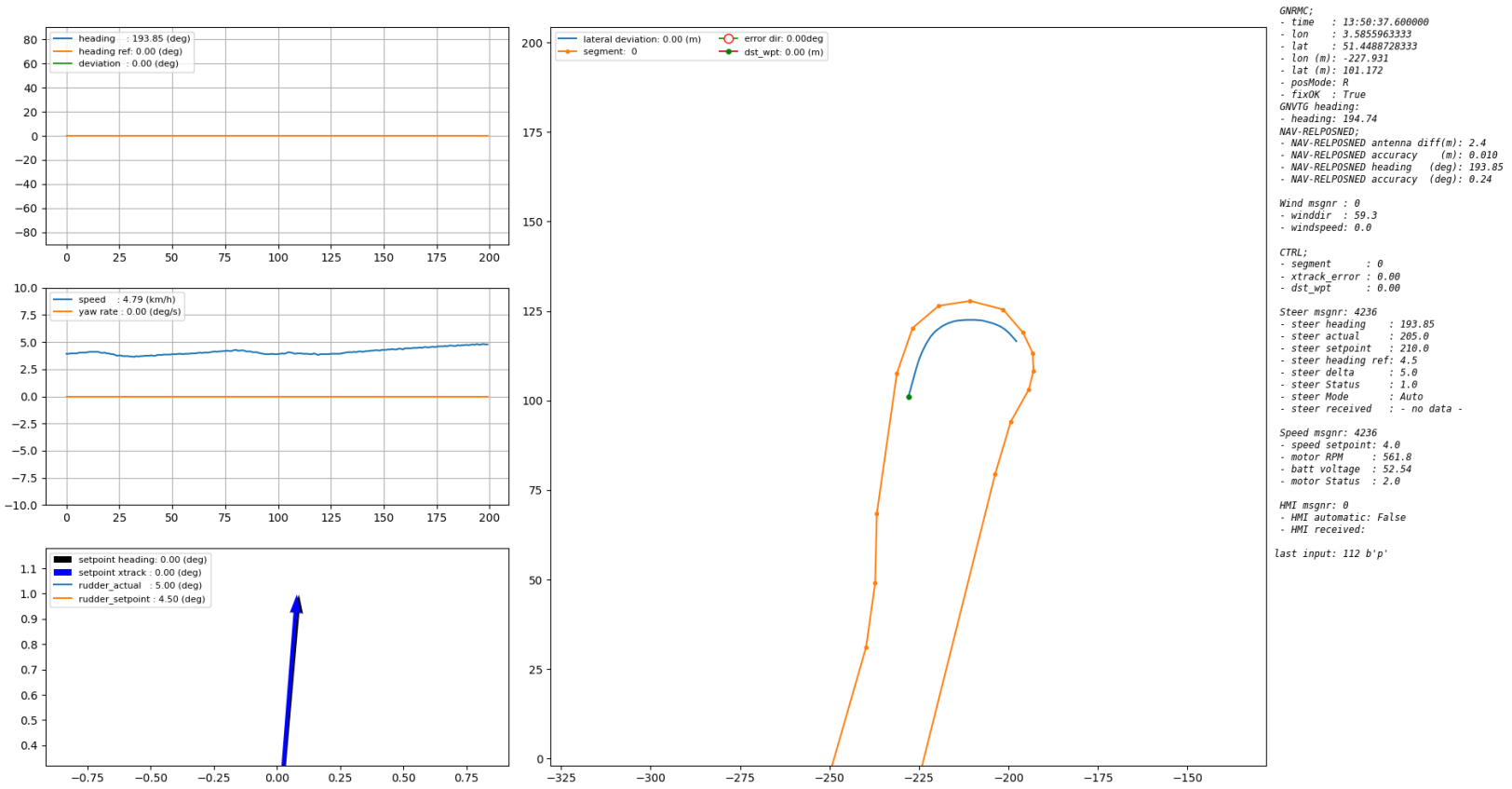


Figure 12: Live monitor during sailing

F Real-time monitor during sailing