

Freie Universität Berlin

Bachelor thesis at the Institute for Computer Science of the Freie Universität Berlin

Developing a Portable Wind Monitoring System for Sailing Events

Finn Böger

Matriculation Number: 5014827

finn.boeger@fu-berlin.de

First Assessor: Prof. Dr. Lutz Prechelt

Second Assessor: Prof. Dr. Jochen Schiller

December 20, 2022

Berlin

<i>Background</i>	In sailing the wind is essential for propulsion and racecourses need to be laid out perpendicular to it to create a fair racing environment.
<i>Objective</i>	Development of a portable wind monitoring system for the mark layer to easily and accurately measure and transmit wind data to the race committee officials.
<i>Methods</i>	A requirement analysis was performed, the general design was thought out and the different iterations of the implementation were tested to evaluate the system.
<i>Results</i>	The system is capable of accurately measuring and transmitting wind data at rest and during constant motion but struggles during periods of acceleration.
<i>Conclusions</i>	The system successfully helped the race committee and reduced the workload placed on the mark layer, allowing less experienced people to assist in that position.

Statuary Declaration

I declare that this paper has not been written by anyone other than myself, that all aids used, such as reports, books, Internet pages, or the like are listed in the bibliography and quotations from other works are marked as such. The work has not been submitted or published in the same or a similar form anywhere.

December 20, 2022


Finn Böger

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

20. Dezember 2022


Finn Böger

Contents

1	Introduction	7
1.1	Goals	9
1.2	State of the Art	9
2	Requirements Analysis	12
2.1	Existing Hardware	13
3	Architecture and Hardware Choices	14
3.1	Selecting the appropriate platform	14
3.2	Hardware selection	15
3.3	Hardware Testing	17
4	Development of the First Prototype	21
4.1	First Live Test	22
5	Development of the Second Prototype	27
5.1	IMU Experimentation	27
5.2	Hardware Upgrade	28
5.3	3D-Printed Enclosure	28
5.4	NMEA2000 Library	29
5.5	Code refactoring	30
5.6	Continued Testing	30
6	Result	32
6.1	Library contributions	32
6.1.1	ICM20948	32
6.1.2	NMEA2000	32
6.2	Wind monitoring system prototype	32
7	Conclusion	33
7.1	Evaluation	33
7.1.1	Usefulness	33
7.1.2	Process	33
7.2	Outlook	33
8	Appendix	35
8.1	Possible itemizations of homebrew alternative	35
8.1.1	Components of solution seen in Kiel	35
8.1.2	Cheapest possible option	35
	References	36

1 Introduction

Sailing is a sport that has numerous expressions, from relaxed cruising along on a lake after work to sailing competitively around the world on 60ft long carbon fiber racing machines. Whichever form your personal relationship with sailing takes, one thing is unavoidable. You are always reliant on the wind and use it to propel yourself forward.

For this thesis, we are taking a look at how regatta organizers use wind measurement data and how we can help them with an automated solution. First, we will have to define what a sailing regatta is:

A sailing regatta is a competition made up of one or more races, where a predefined course is sailed. The length and duration of a race can vary from globe-spanning and a year at sea[14] to short races that take only a quarter of an hour[17]. The course is always laid out using marks, however, these can take different shapes. For offshore races, geographic landmarks are typically used, whereas, for inshore races, the course is normally marked using buoys[40, p. J4]. This second category is what we are focussing on.

The courses used for an inshore race are laid out in such a way that there is one leg that is directly against the wind and one that is with the wind¹. This is done by the mark layer who is typically on a smaller motorboat, either with a rigid hull or a rigid inflatable boat (RIB). While there are many variations on this theme, the general layout always holds[40, p. J6 ff.].

Because sailboats can't sail straight into the wind, they always have to sail at an angle to the wind, to make their way to the windward mark, tacking between starboard and port tack as needed. This makes the upwind leg the leg where the field is most spread out until they coalesce again at the windward mark. The race committee (RC) aims to ensure that sailors spend roughly equal amounts of time on both tacks and that neither tack nor course-side offers an advantage or disadvantage, to allow for fair sailing and tactical freedom[40, p. L 13]. If one were to spend significantly more time on one tack than on the other or possibly even be able to reach the windward mark without having to tack at all, the tactical possibilities on the upwind leg and also the following downwind leg would be severely diminished.

To create a fair course, the RC not only needs to account for the direction but also the race distance to achieve the targeted duration. Accordingly, the committee needs to know the wind speed, the wind direction, and how quickly the class of boats can sail in these conditions. If the speed or direction changes significantly, the committee might have to act by changing the course or, in the worst case, abandoning the race entirely[40, p. L12 ff.]. Additionally, for some classes, depending on the wind speed, specific rules come into play and need to be signaled to the sailors[2, C1.1(a)(1)].

While the RC can measure the wind down at the starting vessel very easily, they can't see how the wind is at the top of the course. Instead, to get the full picture of how the wind behaves across the course, they need the mark layer to position themselves to windward and regularly update the principal race officer (PRO) who ultimately makes these calls.

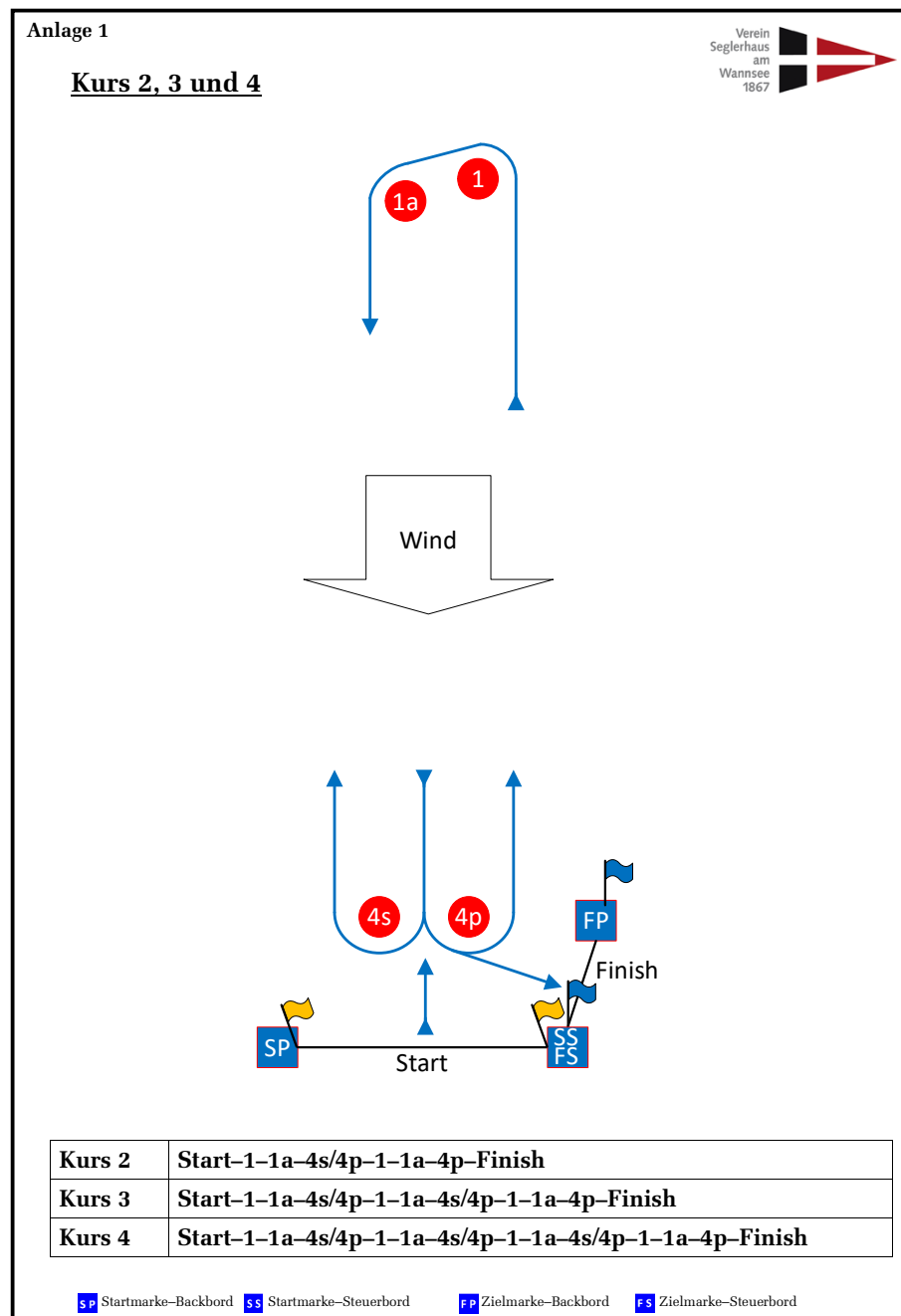


Figure 1: A typical windward-leeward course. ©VSaW [23]

1.1 Goals

My aim is to assist race committees with the laying of the course by giving them a tool to better and more easily monitor the wind and communicate the data.

To this end, I attempt to create an **affordable** solution that can be used to automatically measure the wind on the mark-laying vessel and transmit it to the starting vessel so that the PRO can react to wind shifts before they make their way down over the course.

The proposed solution should be capable of measuring the true wind while in motion. The true wind differs from the apparent wind, which can be directly measured, in that it does not contain the motion component of the measuring platform, thus describing the wind you would measure if you were to stand still at that position.

The automatic transmission of wind data to the starting vessel will reduce the need for radio communication and free up radio channels for other purposes. This could be e.g. communication with the secondary mark layer who is responsible for the leeward gate and the starting line pin end (opposite the starting vessel). This is relevant because most communication takes place when the course is initially laid out. Additionally, automating the data transmission and moving it to a cellular connection allows for clearer transmission in windy conditions as well as over distances longer than a nautical mile, as long as a cell carrier signal exists. While both of these conditions don't make radio communication impossible, they add a significant amount of noise, making relaying a larger amount of precise data very difficult.

A secondary goal is to make the final solution useful for coaching purposes as well, where more immediate wind measurements are required and the boat that the device is stationed on is in motion more often and also changing its speed and heading way more frequently.

1.2 State of the Art

Even without fancy automatic measurement systems the job still needs to get done which means that most race committees resort to manually measuring the wind, typically using a piece of string on a stick together with a hand bearing compass as a wind vane that works even in light (low wind) conditions and a handheld anemometer to get the wind speed. These tools cost around 75€ for a good anemometer and 60€ for the compass, making this the cheapest way to get wind data at a cost of around 135€. Unfortunately, this setup has the downside of only working when the boat is not in motion and requiring constant manual work to catch the trends of the wind shifts.

If one has the money to spend there are at least two commercially available options in the $\geq 10.000\text{€}$ range, specifically

- the **YachtBot WindBot** by **Igtimi** costing around 15.000€[9, p. 6] with the exact price only available on request ²,
- and the **WeatherFile OWS-5 Monitoring System** costing around 11.800€[35], depending on the GBP(£) exchange rate ³,



Figure 2: Yachtbot Windbot installed on a VSR RIB in Kiel.



Figure 3: WeatherFile OWS-5 installed on another boat.

which are both capable of calculating the true wind even while in motion and recording the data over time. The WindBot is completely self-contained and battery powered whereas the OWS-5 requires external power. Nevertheless, both can be quickly installed on a motorboat. Both of these solutions make use of the **Gill WindSonic 1** ultrasonic anemometer, itself available for around 1.400€. Another trait they share is that they send the data over the cellular network to the cloud service provided by their respective manufacturer and allow users to access that data using various integrations.

Another group of solutions that I've observed was the use of off-the-shelf marine hardware meant for much larger boats on RIBs. This took the form of a GPS Compass, wind sensor, and Chartplotter connected together to form an NMEA2000 network. The price for this solution can vary by a large margin and the systems I've seen would be in the 5.000€ neighborhood ⁴.

The cheapest option using this approach would cost around 1700€ to be broadly comparable and 2100€ using the same sensor. However, these systems would not have the transmission capability that the commercial solutions boast and are also missing the capability to counteract the swaying/rolling of the boat.



Figure 4: Homebrew solution spotted in Kiel.

2 Requirements Analysis

Now knowing the general goal it is time to check in with the potential future users.

The stakeholders I've identified are the race committee, with a focus on the principal race officer and the (windward) mark layer, the coaches in the end users role, the sailors as beneficiaries, and myself as the sole developer. For large events that hold e.g. World Cup status or are relevant for Olympic qualification, the supervising associations can also be considered to be stakeholders.

The PRO needs to be able to get the true wind speed and true wind direction on demand, preferably not only as it is right now but as it was on average over the last few minutes and with a 5° accuracy. Preferably the changes over time can be seen as well.

The mark layer can't constantly measure the wind manually and thus needs the measurement to be done automatically. They also don't want to have to stop the boat each time they take a measurement, thus the true wind needs to be computed from the measured apparent wind. The data transfer should be performed automatically as well. As a backup, the data should also be visible locally, preferably including any averaging and filtering. They are volunteers and can't be expected to be specially trained on the equipment, thus it should not be more complicated than plugging the device in.

The coaches have two distinct use cases. When coaching sailors at a regatta the interaction with them is limited to the breaks between the individual races. During the races themselves, they can only observe which means they typically follow the fleet to the windward mark. After making their way upwind on the side of the course, they wait there to note the mark rounding maneuver and then travel back down the course alongside the sailors.

While observing the fleet they need to gather data about the environment (e.g. wind shifts and where the gusts start) which they can then impart to the sailors for the next race. Therefore, the coaches need the true wind to be calculated while in motion and the data stored to be analyzed after the race or during waiting periods.

However, when they are instead coaching a single boat or a small group, it is common to act as a chase boat and follow the sailboat, keeping an eye on their actions and settings. In this scenario, instead of the wind over time, the current wind and the angle that is being sailed to the wind are of interest. As it is difficult to match the speed of a sailboat exactly the motorboat will also frequently accelerate or decelerate which can throw off the true wind calculation if only the GPS is used for movement data.

The sailors want a fair racing environment which is achieved by the race committee having accurate data and acting on it in accordance with the rules and regulations. The same is the case for the supervising associations.

My own requirement as the developer is to keep the scope to something that I can reasonably develop in 3 months without the result suffering quality-wise. I should also be able to gracefully reduce the scope without compromising on the core requirements if some parts turn out to not be possible in the given timeframe. Additionally, I wanted to



Figure 5: Existing wind transducer and mast (and buoys) on “Lottchen”.

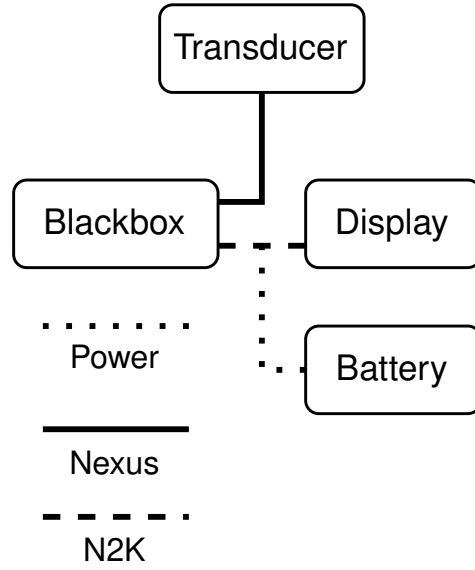


Figure 6: Simplified wiring diagram.

take advantage of the existing hardware and develop a solution that could be attached to arbitrary wind sensors, as long as they used the same protocol.

2.1 Existing Hardware

Garmin had supplied the club with a wind transducer package and a display that are mounted to a 2.8-meter tall carbon fiber mast. The transducer is the Garmin gWind[13] which was originally manufactured by Silva and thus uses the proprietary Nexus NX2 network. To facilitate communication with modern devices it comes in a package with the Garmin GND™ 10 Blackbox[12], which translates to NMEA2000. The display is the Garmin GMI™ 20 multi-function display[11] which also uses NMEA2000 but supports the older NMEA0183 as well. The wiring to connect the components and provide power was done by soldering each channel together, attaching a 120 Ω resistor to the data lines, and then encapsulating the solder joints in a non-permeable compound. The power line was simply open-ended and needed to be connected to a 12v power supply.

The mast can be mounted to either the coaching RIB “Wannsee I” or the mark-laying boat “Lottchen”⁵ using two different mechanisms. On the RIB, a metal plug is attached to the floor, that the mast can be positioned over, while on Lottchen the mast was simply tied down using ropes. The ability to use the mast on at least both of these boats should be maintained and ideally improved by removing the cumbersome process of connecting the bare wires to the onboard power.

3 Architecture and Hardware Choices

Looking at the requirements, I decided that the best way to have the data available on the starting vessel was by transmitting it to a server where it can be accessed in the shape of graphs and formatted current values. The server is also responsible for storing the data needed to create the graphs in the first place.

To transfer the data to the server, I selected wanted a protocol that added as little overhead as possible to my payloads while still being reliable, which made me take a page out of the IoT book and choose MQTT. MQTT is a machine-to-machine protocol that needs a broker to act as an in-between for the clients, which can act as subscribers or publishers[25].

Last but not least, I had a look at the part that has to perform the actual work. The client device has to collect the data from the NMEA2000 network, process this data, and send it to the server while also feeding the processed data back into the NMEA2000 network.

3.1 Selecting the appropriate platform

Approaching the work, my requirements for the platform I would build my solution on were as follows. I required a cellular data connection, a way to know my current speed and heading, my current bearing, and a way to talk to the existing wind sensor. Additionally, to improve the movement data, an Inertial Measurement Unit (IMU) was desirable. Taking these requirements leads me in two directions, either build my own bespoke solution using a microcontroller or a single-board-computer or take advantage of a common device which already fulfills nearly all my requirements: A smartphone. The only part of my requirements that was not built in was the ability to communicate with the existing sensor, however this could be taken care of by using a NMEA2000 to USB adapter, available for roughly 180€[47].

Unfortunately, upon closer examinations, a few unresolvable issues cropped up. The workload of the phone would be very similar if not even more demanding compared to using a smartphone for position tracking. In both cases the device will constantly monitor its position via GPS and transmit it using the cellular network. This is a use case that I gained some experience with in the context of the Sailing Champions League, where smartphones were used as trackers for boats and marks on the water. The primary issue encountered there was the devices overheating, shutting down as a protective measure and only restarting after the temperature dropped well below the threshold, which could take the device out of circulation until the rest of the day.

Investigating the viability of smartphones as a platform revealed further requirements to me that I assumed implicitly before. The platform should be stable and capable of easily running for a full 12 hours. It should start automatically when power is supplied and turn off when external power is removed. The second set of requirements would be difficult to achieve on a smartphone as well, as not all of them support `fastboot oem off-mode-charge 0` and if they do, this option can soft-brick the phone if the battery is fully discharged because the phone would try to boot as soon as power is connected,

however the trickle charge current would not be enough to actually power the phone, trapping it in a boot loop[6].

Having eliminated the smartphone option, I was left with the choice between a micro-controller like the Arduino platform or a single-board computer (SBC) like the Raspberry Pi, ultimately opting for the Raspberry Pi due to familiarity with the device. It also helped that I would be able to use Python on the Pi as I've found myself to work quicker with this language compared to the likes of C.

3.2 Hardware selection

The Raspberry Pi family has several options for SBCs that contain the iconic 40-pin GPIO header and have a broad ecosystem of add-on boards. I narrowed down my selection by focusing on the relatively small computational requirements and the fact that a small form factor would be advantageous. While, with a regular Pi the whole the device could still be mounted on the carbon mast, it would protrude a fair bit further, which I wanted to avoid. Thus, the smaller Pi Zero products were of particular interest to me. Due to the general chip shortages Pi Foundation products were hard to come by, so I purchased both a Pi Zero WH and a Pi Zero2 W ahead of time, taking advantage of them being in stock at that moment.

Having selected a platform to develop on I needed to extend it to satisfy our requirements. The Raspberry Pi by itself can't do any of the things I need, thus necessitating the purchase of additional hardware. The ecosystem of the Pi is large enough, that there were solutions available for all my problems in the pHAT form factor. HAT stands for "Hardware Attached on Top" and is the official specification for extension boards for Raspberry Pi family computers. They have the same dimensions as a Raspberry Pi B and can be easily stacked atop each other. pHAT conversely is not officially specified but is taken to mean the same for computers of the Raspberry Pi Zero family. This includes the stacking concept and dimensions that are adjusted to be the same as the Zero. While physical stacking is easily possible as long as the HAT passes the GPIO pins through, this does not hold for the electrical connections where one needs to take care that pins are not used by multiple boards at the same time, or if they are that it is in a fashion that supports such use, e.g. in an I2C interconnect.

Looking at the requirements I needed a GPS to get my movement vector. I had two options here, either purchase a marine GPS that would then directly integrate with the NMEA2000 network or get a GPS receiver that I could connect to the Pi. In the interest of keeping costs down, I went with the Pi-attached option as there was a larger range of cheaper solutions. Since I was satisfied with the positioning capabilities of my phone I thought that a GPS in the same capability class would suffice. I also wanted to use an IMU to improve the movement data and was not particularly interested in the location data. This distinction is relevant because modern GPS receivers use the Doppler effect to get velocity and heading instead of calculating the difference between two consecutive position fixes. Thus, the accuracy of the position data does not directly correlate with the accuracy of the movement data [36].

Desiring to minimize the depth of the assembly and thereby the distance it would

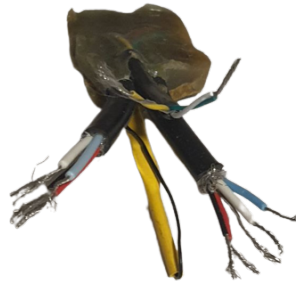


Figure 7: Soldered and overmoulded connection point.

protrude from the mast I opted to focus on modules that were available as pHATs and excluded USB modules from my search. A few expansion boards combined network connectivity with a GPS receiver on a single PCB. This was an attractive option to me, both for saving space and being generally less expensive. I had to ignore a few options as they only had 2G or 3G connectivity on board which is in the process of being decommissioned in Germany. This narrowed my search to multiple offerings by Waveshare and two modules caught my eye in particular. The “SIM7600G-H 4G HAT (B)” and the “SIM7000E NB-IoT / Cat-M / EDGE / GPRS HAT”. The SIM7600G version features full-fat 4G[42] whereas the other, as the name implies, only comes with NB-IoT and LTE Cat-M[41]. However since all carriers in Germany claim to support these standards[27][28][46], the module was significantly cheaper and the slower speed did not matter for my application, I went with the SIM7000E HAT.

Having already satisfied the cellular networking portion of the requirements the next step is to be able to determine the orientation. For this, there weren’t any alternatives in the pHAT form factor making my choice a simple one, so I bought the “Waveshare Sense HAT (B)” to get access to its Accelerometer, Gyroscope, and Magnetometer assembly.

Next, I needed to handle the communication with the sensor. NMEA2000 is based on SAE J1939 and thus uses the CAN-Bus. This meant that instead of having to purchase an expensive NMEA2000 to USB adapter I could go with a regular CAN adapter, which is an order of magnitude cheaper. The chosen form factor again limited my choices and I purchased the “Waveshare RS485 CAN HAT” which also included an RS485 adapter.

The existing wiring could not accommodate an easy expansion to connect the Raspberry Pi. This was due to the electrical connections being made by soldering the different wires together and then sealing them in an isolating hardened compound⁷.

To add new devices I elected to use a proper NMEA2000 backbone. While more expensive than simply soldering the different leads together it also allowed for easier debugging and development. If I had gone with the soldering option the addition of new devices would’ve been significantly more difficult, and I would have another part that would need to be manually waterproofed. Knowing that I needed to connect at least three devices (wind sensor, display, Raspberry Pi) plus power to the network I formulated

the requirements.

- Male and female end resistor
- 4x T splitter
- Power cable
- 3x short data cable

Several manufacturers had enticing options in the form of starter kits but all of them fell short, primarily in the amount of included cables, while some simply had too few connection points. To minimize the cost while still having a proper backbone I went with a non-certified option and bought the pieces from China. The electrical connection is made using a “DeviceNet Micro-C” connector, also known as “M12 5-Pin A-coded”.

Because the coaching RIB had an automobile auxiliary power outlet, also commonly known as a 12v cigarette lighter socket, I added a corresponding plug to the designated power cable and made a note to purchase another socket for installation on the mark-laying boat.

This powered the network with 12V which the Raspberry Pi could not use directly, so I included a simple step-down converter from 12V to 5V to supply the Pi with usable power.

3.3 Hardware Testing

After the equipment arrived I began testing the various components. Starting with the GPS I encountered the first problem. The Raspberry Pi has one serial port (UART) available over the GPIO pins. While the GPS worked over this integrated UART, the cell modem could not be used at the same time. To have both working at the same time required a USB connection between the Raspberry Pi and the HAT. This made my choice to forgo the USB-attached solutions meaningless, but now I was stuck with this hardware.

Testing the cellular connection was not possible at this moment as I didn’t have an NB-IoT specific SIM card and planned on equipping the device with an IoT cell contract granting 500MB of data volume over 10 years of use for 10€ plus tax.

Moving on to the Sense HAT, I was again glad for the example code provided by the manufacturer. Running it I could see that the Magnetometer experienced the same issues as the compass in your phone does. I am of course speaking of drift and the need to be frequently recalibrated by spinning the device in a figure-of-eight motion. This is predictably difficult if it is installed on a 3-meter-long mast. Knowing which direction we are facing is paramount when measuring the wind direction, otherwise, we can only tell which way the wind is coming from relative to ourselves. One option for solving this is using the heading provided by our GPS. This comes with the issue that it only works while in motion; turning on the spot would not update our GPS-determined heading while still changing it. Another problem is that this would require the assembly to be

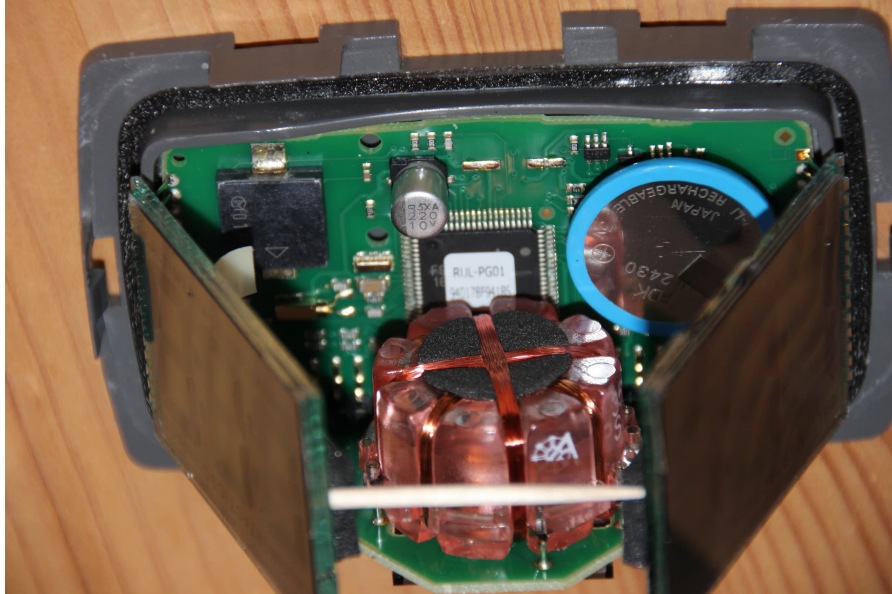


Figure 8: TacTic Micro Compass without its enclosure.

always mounted facing straight ahead which is difficult to enforce, especially given the different mounting positions on different types of boats.

When sailing myself I use an electronic compass that does not need to be recalibrated and does not even allow for user calibration, so I knew that there had to be a semi-affordable option. The compass I'm used to using is the "TacTik T060 Micro Compass" by Raymarine, with an RSP of 370€. Because I couldn't find out online what kind of technical solution was used inside, I decided to disassemble the compass.

Removing the gray outer shell revealed a clear acrylic housing. Prying away the retaining tabs of the housing allowed me to extract the internal assembly. Now I could bend the screens to the side and take a glance inside ⁸. Unfortunately, I could not immediately identify the used components. Luckily I had a friend who was able to tell me that I was looking at a fluxgate compass.

Armed with this new knowledge I went shopping again and found out that the installed component was manufactured by Autonnic. Autonnic is a British company producing both OEM components but also end-user devices using their own components. The compass is available in four variants, either gyro stabilized or not and either outputting as NMEA0183 or NMEA2000, and costs between 156£ and 215£ plus tax and shipping(15£) turning it into 230€ to 295€[3]. While searching, I also found a compass from NASA Marine which has an RSP of 108£[29] and was available for 132€ at that time. The creatively named "NASA Marine - NMEA Compass Sensor" uses NMEA0183 for communication. This is the predecessor to NMEA2000 but designed in a completely different way. Instead of facilitating communication using a bus it is used for point-to-point communication but could be daisy-chained. Electrically it uses either RS422 (v2.x+) or RS232 (v1.x) depending on the revision.

Being cheap I purchased the NASA compass and upon arrival noted that the NASA compass used the older version, which I determined by it having only one data wire in addition to power and ground. While the HAT only claims to support RS485 the used transceiver is the SP3485 which also supports RS422[24]. This was fortunate for me because this meant that it tolerated higher voltages and could handle the output of the compass.

The compass worked as advertised and matched the readings that I took with the TacTik Micro Compass.

Next up was the communication with the NMEA2000 devices. While still waiting for the cables to arrive I replicated the previous wiring solution using WAGO clamps, although without the resistor. The data was transmitted and received properly and could be read using the `candump` utility from the `can-utils` package

Having tested all parts individually I tried to perform an integration test and attach all HATs to the Raspberry Pi at the same time. This was not immediately possible due to the Sense HAT not repeating the GPIO header, instead letting the pins of the board below pass through itself. Because the pins of none of the boards were long enough to pass through and then connect with the next board, I needed to order some GPIO header risers.

After the risers arrived I could continue with the integration test. This immediately revealed a problem where the `can0` interface could not be opened. At the same time, the serial connection provided by the CAN HAT was not affected. To investigate this issue I removed the GPS HAT and noticed that the problem persisted. However, when I removed the Sense HAT and left the GPS HAT connected I could bring the `can0` interface online. Thus concluding that the Sense HAT was in some way interfering I attempted to narrow down which pin was causing the issue ⁹.

It turned out that the GPIO Pin 25 (BCM) was used by both the CAN interface and the air pressure sensor for interrupts. This was a fact that I missed when I checked the spec sheets before purchasing. As I didn't need the interrupt feature of the air pressure sensor I simply positioned the Sense HAT above the CAN HAT and removed the electrical connection by snipping that pin off at the GPIO riser that I used to connect both boards.

Another issue occurred with the serial console which was getting used by both the GPS and the compass. This was easily resolved by moving the GPS over to the USB connection which needed to be added anyway to use the cellular connection in the future.

The result was that the Sense HAT was connected using an I2C connection via pins 2 and 3, as well as pins 5 and 6 in use for interrupts. The RS485 CAN HAT used the serial connection on pins 15 and 16 for the RS485 transceiver and the SPI0 connection on pins 7 through 11 for the CAN transceiver. Finally, the assembly was capped off with the GPS and NB-IoT HAT which did not connect to the GPIO pins and instead had a USB cable leading to it.

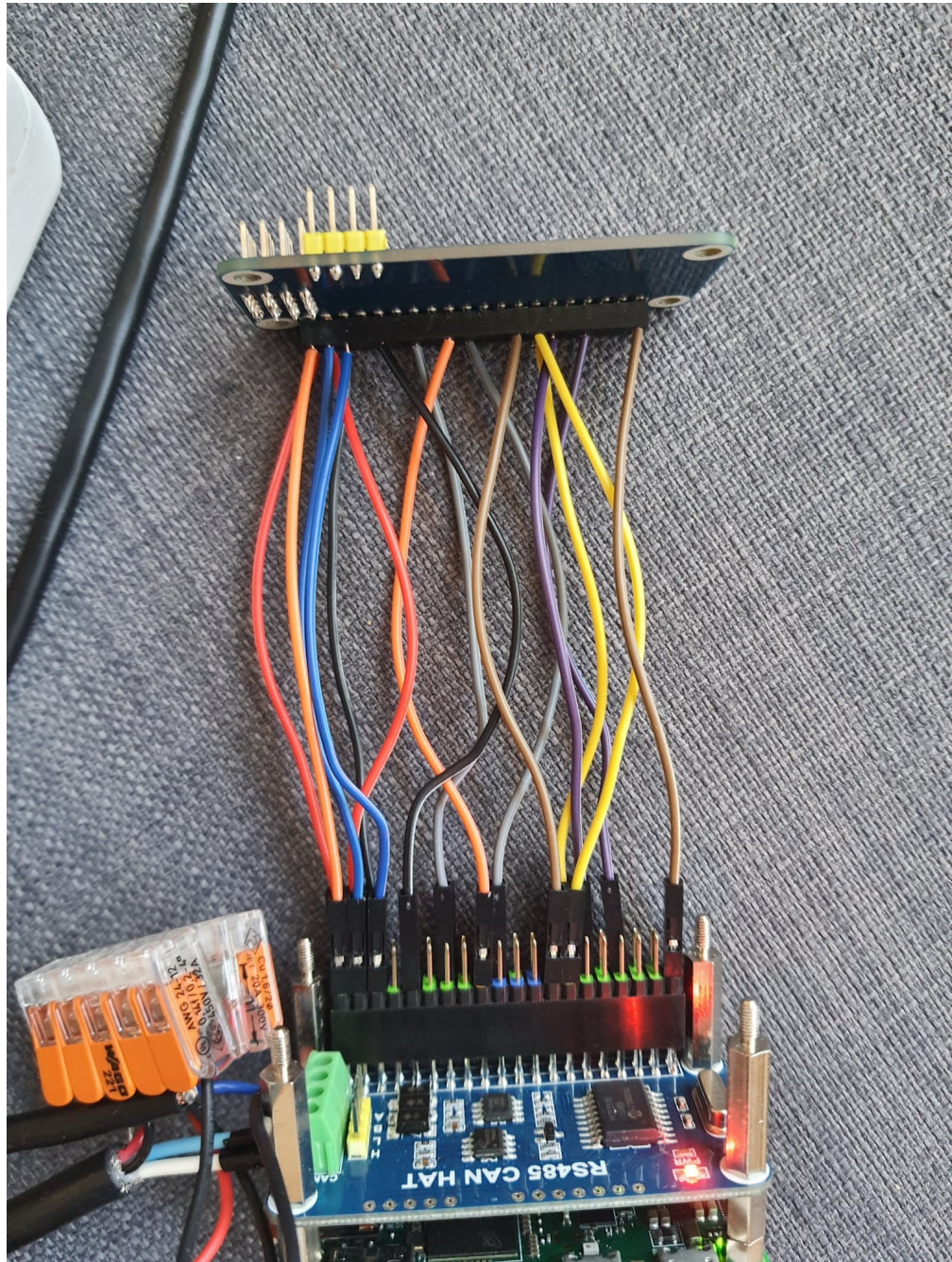


Figure 9: Debugging the pin collision by testing individual pins.

4 Development of the First Prototype

Now that the hardware was confirmed as working, I had a lot of data inputs that I needed to transform into readable data and then a sensible output. Given that NMEA2000 is a binary protocol I needed some help to parse the messages from the wind sensor. This came in the form of the `canboat`^[4] set of utilities, in addition to the already installed `can-utils`^[43]. The provided utilities include `candump2analyzer`, `analyzer` and `n2kd`, which are used in true Unix fashion by piping the output of one to the input of the next. The full pipeline then looked like this: `candump can0 | candump2analyzer | analyzer -json | n2kd`. While `candump` read the raw binary data from the interface and echoed it to `stdout`, encoded as hexadecimal¹⁰.

```
can0 1DEFFF01 [8] 80 0E E5 98 05 05 02 02
can0 1DEFFF01 [8] 81 FF 00 00 1F DF FA FF
can0 1DEFFF01 [8] 82 FF FF FF FF FF FF FF
can0 09FD0201 [8] FF FF FF FF FF FC FF FF
can0 09F11201 [8] FF FF FF FF 7F FF 7F FD
can0 09F11202 [8] 5B 94 EA 00 00 00 00 01
```

Figure 10: Output of `candump`

In the next step `candump2analyzer` added timestamps to the messages and decoded the message ID into priority, Parameter Group Number (PGN), source, and destination, while also changing the format to be comma separated¹¹.

```
2022-12-16-23:56:38.675,7,126720,1,255,8,80,0e,e5,98,05,05,02,02
2022-12-16-23:56:38.675,7,126720,1,255,8,81,ff,00,00,1f,df,fa,ff
2022-12-16-23:56:38.675,7,126720,1,255,8,82,ff,ff,ff,ff,ff,ff,ff,ff
2022-12-16-23:56:38.676,2,130306,1,255,8,ff,ff,ff,ff,ff,fc,ff,ff
2022-12-16-23:56:38.676,2,127250,1,255,8,ff,ff,ff,ff,7f,ff,7f,fd
2022-12-16-23:56:38.676,2,127250,2,255,8,5b,94,ea,00,00,00,00,01
```

Figure 11: Output of `candump2analyzer`

This is still not exactly readable, which is where the `analyzer` comes in and turns it into human-readable JSON¹².

The `n2kd` daemon then creates several TCP servers, one of which streams the JSON data and another that translates the PGNs to the old NMEA0183 format if possible, and serves these as an ASCII stream. This is not strictly necessary but is convenient because it allows me to treat it the same as the serial consoles that are streaming their data at me, also in NMEA0183 format^[26]. The parsing of all the incoming NMEA0183 messages was then offloaded to the `pynmea2`^[37] library.

When connecting power the Pi boots and immediately executes our code. First, we wait for the GPS receiver to determine its position, then we establish a connection with the MQTT server. Afterward, the main loop is entered where we read from the sensors and perform some basic vector addition to get the true wind. This is then transferred to the server via MQTT¹³.

```
{
  "timestamp": "2022-12-17-00:01:24.880",
  "prio": 2,
  "src": 2,
  "dst": 255,
  "pgn": 127250,
  "description": "Vessel Heading",
  "fields": {
    "SID": 91,
    "Heading": 344.1,
    "Deviation": 0,
    "Variation": 0,
    "Reference": "Magnetic"
  }
}
```

Figure 12: Output of `analyzer -json -empty` (formatted with `jq` and truncated to only show the last message)

```
init_gps()
connect_mqtt()
while True:
    gps, heading, wind = read_sensors()
    true_wind = combine_forces(wind.direction, wind.speed, gps.heading, -gps.speed)
    true_wind.direction += gps.magnetic_declination
    mqtt.publish("true-wind", format(true_wind))
```

Figure 13: A very simplified version of the code I used

Having pieced the code together, I had to figure out a way to mount the system on the mast and waterproof it for its first live test at Kiel Week. While I had searched for enclosures previously, I didn't find anything that exactly matched my desires. However, as this was still only an early prototype, looks weren't important, so I bought an oversized cable junction box from my local hardware store ¹⁴. Mounting this on the mast directly was not possible, leading to me 3D printing some small brackets that would allow me to use cable ties for mounting.

Having transferred the data to the server I still needed to make it understandable at a glance for the end user, while still including data that allowed me to verify the results. To do this I used the `justpy` framework for the server which allowed me to quickly provide automatically updating data and created a very simple layout that is supposed to be easily readable on a tablet ¹⁵.

4.1 First Live Test

When rigging the boats for the event and mounting the mast to the RIB that we were using, it became apparent that the case could not be fastened as tightly as I wanted. This threw a spanner in my plans to log some IMU data and determine how much the boat (and especially the mast) sways during operation. Another thing that I could not test was the cell reception as I did not have a SIM card yet and had to fall back on a

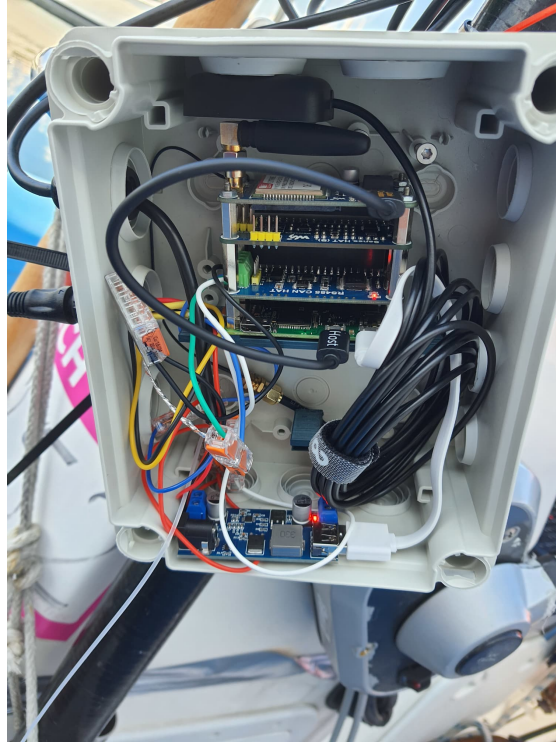


Figure 14: Raspberry Pi with HATs mounted inside a waterproof junction box.

mobile hotspot.

The first day revealed some bugs that are already fixed in [Figure 13] such as a missing sign in front of `gps.speed`, thus leading to incorrect true wind or typos in the MQTT topic. Much more problematic was, that the serial consoles provided by the GPS & NB-IoT HAT sometimes did not appear. To get it to reappear at least one and sometimes multiple restarts were necessary. However, once the connection with the console was successfully established, it stayed accessible. I could not solve this issue by replacing the cables, so I had to live with it for the time being. A problem that compounded to make this worse was that the power connector would not stay in its socket. Because the socket on our RIB was a smooth bore and the spring on the top of the connector was too powerful, each shock had the connector move further away until power was intermittent, and the device was frequently restarting. To mitigate the problem the solution was again to use tape to fix the connector in place. While this worked to an extent it only extended the amount of time that the device stayed powered on, and it still sometimes lost power. After increasing the uptime of the device it quickly became apparent that the data transmission cut out after a short while even though the code kept chugging along without any errors. I pinpointed the source as a rate limit imposed on the free public MQTT of HiveMQ[16] that I was using even though I couldn't find any mention of it on their website. Switching this to a free private instance provided by HiveMQ resolved this issue.

4 Development of the First Prototype

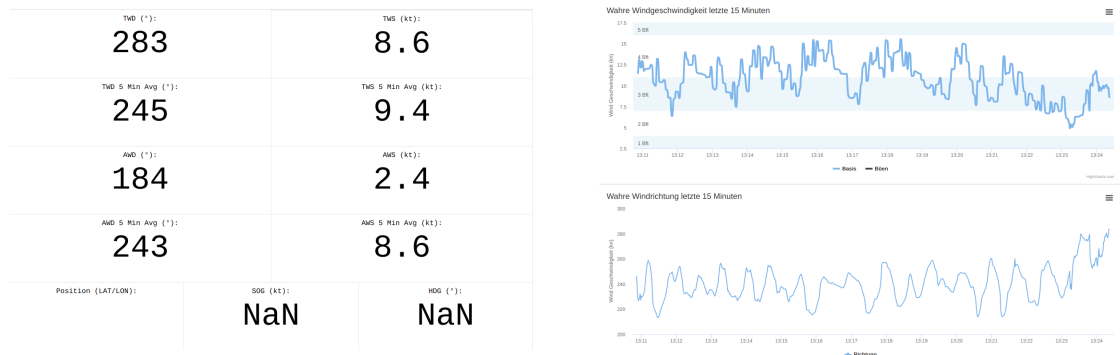


Figure 15: Raw data and graph view (for true wind) provided by the server.

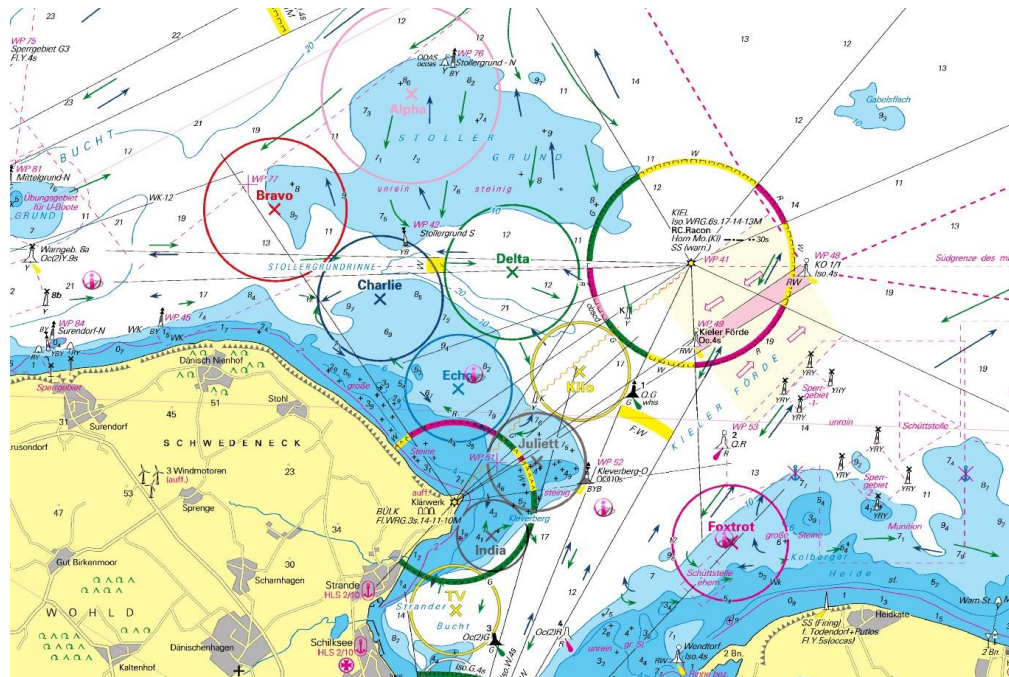


Figure 16: Nautical map of the racing areas in Kiel. ©NV Charts[22]

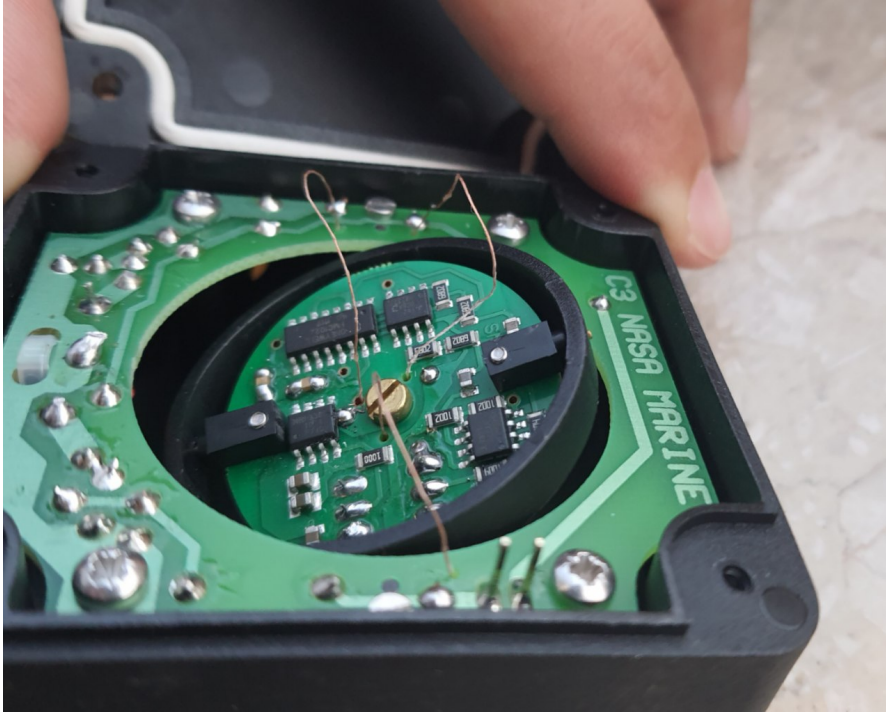


Figure 17: NASA Marine Compass opened to expose the gimbaled section.

Because we were working on the water for the whole day and beat afterward, I had to wait for the break between the two halves of Kiel Week to find some time to fix the software bugs. The second part of Kiel Week saw us offshore on race area Charlie ¹⁶ which presented an unexpected issue in the form of lacking cellular data coverage. At this point, the only way to read the calculated true wind was by connecting to the device using ssh and printing the values to the console instead of attempting to send the data via MQTT. Being further out to sea also meant a higher sea state in windy conditions. This exposed an issue with the gimbaling that the compass employed ¹⁷. Because it only used a small weight at the bottom of a PCB that was itself mounted with 2 axes of (limited) rotation it could start swinging. This then resulted in heading values that were off and oscillating, even when keeping course.

When debugging the issues in Kiel I noticed that the connection to the compass was suboptimal for easy disassembly. Because NMEA0183 does not come with standardized connectors, the cable was routed through a cable gland into the enclosure and then connected via some WAGO clamps. This and the fact that I would have to redo the cable ties meant that I could not easily remove the case from the mast and had to work on the boat. While the cable ties were only ever meant to be a temporary arrangement, I hadn't originally planned on attaching the compass differently. Unfortunately, I crossed the data and power lines when attaching the plug and thereby bricked the compass. As I already had the gimbaling issues in Kiel I decided to replace it with the alternative from Autonnica and went with the NMEA2000 version to have it all using one connection. The

4 Development of the First Prototype

Autonnic compass uses a dampening system and is also gimbaled to 45° instead of 30° .

5 Development of the Second Prototype

5.1 IMU Experimentation

The first test exposed a lot of issues but was still promising. I turned my attention to improving the GPS data next to make it more responsive to changes. My idea was to use a simple Kalman filter to synthesize measurements using the IMU. What I had at this moment could not be called an IMU yet, instead, I only had separate measurements by the accelerometer, gyroscope, and magnetometer, as well as the compass that could be used to compensate for yaw drift. To turn this into useful data I needed to feed the data into an inertial navigation system algorithm. But for that to happen I first had to get the data from the sensor.

Waveshare provided some demo code in python[44] which demonstrated how to get individual measurements but didn't provide a library. A third party had created an open-source library[19] for their breakout board using the same sensor with roughly the same scope as the demo code which was a good basis but still limited. Fortunately, the used sensor (icm20948) had several other libraries, one of which was by Wolfgang Ewald and included support for the FIFO functionality[20]. Unfortunately, it only supported getting the accelerometer and gyroscope values this way. Using the C++ code as a base I added FIFO functionality to the python library I had found. The direct port I started out with only read a single set from the queue at a time which was barely an improvement over querying the sensor for a fresh sample. This way it took 0.4 seconds to collect 100 samples. I improved the code by reading the maximum amount of bytes at a time over the I2C bus which was 32 bytes for this device. This dropped the time it took to transfer the same amount of data all the way down to 0.022 seconds. This meant that we could stop the queue, get the amount of waiting samples, restart the queue, and then extract the data without risking the queue overflowing in the meantime. The reduction in transfer time allows a theoretical maximum sample rate of 4.5kHz. Of course, adding the magnetometer to the queue in the future would increase the amount of data that needs to be transferred and thus decrease the sample rate, but it should still exceed the maximum of 1.125kHz that the sensors are capable of. The minimum amount of time the sensor needed to rest between complete reads of the FIFO seemed to be roughly 0.01 seconds as determined by testing. This meant that the queue would be populated by roughly 144 bytes of data before it could be emptied again which would be well under the maximum capacity of 4 kB. Again values with magnetometer data would differ but should still not exceed the maximum.

To test the feasibility of running an inertial navigation system (INS) algorithm on the Raspberry Pi Zero I ran an attitude and heading reference system (AHRS) algorithm that was already implemented in a python library, fittingly named `ahrs`[18]. I choose the Madgwick algorithm as it seemed to be the closest in terms of workload. The library makes no claims to be an efficient implementation and even claims the opposite, being in pure python, but still makes use of NumPy for many of its calculations. Running the algorithm while only sampling accelerometer and gyroscope data at a rate of 100Hz already kept the CPU load pegged at 100%. This convinced me to upgrade the platform

to the more powerful Raspberry Pi Zero2.

5.2 Hardware Upgrade

Making use of the increased computational power also necessitated a rewrite of the code to take advantage of the now 4 available processor cores. At the same time, I needed to tackle displaying the generated data on the display. The current solution of using `canboat`[4] worked well for reading, but it did not provide sufficient support for writing data. While it did have a utility to write to `socketcan` interfaces the input format was the same as what `candump2analyzer` generated¹¹. As this included raw binary data it would always require some effort on my side to encode my data in this proprietary format. Simultaneously, sending data to the network elicited messages from nearly all other devices, inquiring who I am, and depending on chosen source address, requesting that I move to a different address.

Seeing the issues with using `canboat` for sending I made up my mind to create an NMEA2000 library that would take care of all the automated messages and let me send my e.g. wind data without having to do any manual encoding.

Moving on to the hardware side of things I replaced the generic 12v plug on the power cable with a marine-specific version that includes a locking mechanism.

I also still needed to figure out a more permanent solution for the enclosure. Because I couldn't find appropriate options for sale I opted to design an enclosure myself and 3D print it.

5.3 3D-Printed Enclosure

One of the most used raw materials for 3D prints is PLA, which is neither suitable for outdoor use nor completely waterproof, but very easy to work with. There are however methods and materials that can yield a waterproof and weather-resistant 3D print[15].

For the first prototype of the enclosure, the drawbacks of PLA didn't matter as much because it was not intended to be permanent. The print would protect the internals against water spray and a simple layer of tape over the top of it protected against rainwater accumulating there and seeping in overnight.

The parts that needed to be designed were not only the enclosure but also the mounting brackets for the various pieces of hardware. To allow for easy changes in the dimensions I opted for a parametric design approach. My first attempt was using FreeCAD, but I quickly noticed that it was easy to use for parts that extrude from a single plane it fell apart when anchoring more planes to an object and then resizing the base object. Instead of spending more time learning to use the FreeCAD GUI, I pivoted to a code-driven approach and switched to OpenSCAD[34].

Using OpenSCAD allowed me to perform a dry fit of all my designed components to verify that they could be assembled before committing to the print and prevented me from printing at least one faulty part.

Seeking to minimize the depth of the enclosure I tried to find a micro USB to micro USB OTG cable with right-angle connectors on both ends but ended up having to manually

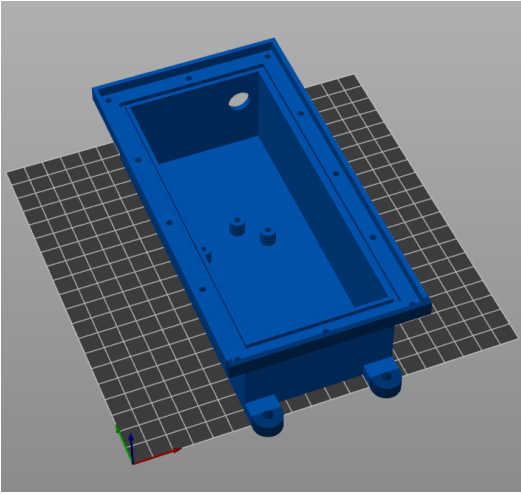


Figure 18: 3D model of enclosure body.



Figure 19: Enclosure filled with all components and sealant around the edges.

solder the cable. Interestingly this new cable stopped the issues I had been experiencing with the GPS console from happening.

5.4 NMEA2000 Library

NMEA2000 is based on SAE J1939 and both are Higher Level Protocols that use the CAN bus to provide the data link and physical layer[39][45]. Whereas the CAN messages only include message IDs with a length of either 11bit or 29bit and up to 8 bytes of data, NMEA2000 includes additional information by encoding it in the message ID[21]. Specifically, NMEA2000 uses only the extended CAN message with the 29bit ID and encodes the message priority, Parameter Group Number (PGN), source, and destination in it [5][30]. The PGN determines how the binary data is interpreted and is where the proprietary nature of the standard is very noticeable. The official documentation can be purchased but costs several thousand dollars and would not allow open-source software to be developed with that knowledge[31]. This means that all knowledge has to come from reverse engineering purchased devices. Luckily there are already two libraries that have performed this work. The first was already introduced as we used it for the first prototype and is `canboat`. The second is `ttlappalainen's NMEA2000 library`[32] which on the surface looked exactly like what I needed, but it came with the issue of being written in C++ and didn't have any python bindings available. As I had never before worked with C++ I couldn't even get it to compile and didn't want to be stuck with maintaining a library using an unfamiliar language. Being an otherwise good match for my needs I decided to port the library to python.

The library supported several modes which impacted how it behaved on the bus, from

passive listening to sending without announcing itself on the bus and finally a complete node that can listen and send and properly announces itself. The library implemented a listener that handled the system messages and then called user-provided listeners. If the forwarding flag had been set, the data was then also sent to a user-specified stream.

To make my work easier and because I disagreed with some design decisions I modified the architecture. First I dropped the code that manually handled the CAN connection and replaced it with the `python-can` library. Next, I dropped support for emulating multiple devices with a single instance and removed the Arduino-specific code that sought to reduce memory usage. Another victim of my cuts was the whole message forwarding code, as it could be easily implemented using a custom message listener if one needed it. Some things I only skipped for the first implementation, such as support for Transport Protocol (TP) messages, that enabled payloads up to 1785 bytes[38].

5.5 Code refactoring

The refactored client code splits the tasks across 3 processes. While the start stays largely the same, as soon as a GPS fix has been achieved the separate processes are started. Process 1 is supposed to deal with refining the GPS-determined movement using the inertial data but does nothing at this time. Process 2 handles communication with the MQTT broker instance. And most importantly the initial process 0 listens to the GPS serial console and runs the NMEA2000 node. The correction of the measured wind data is performed in a message handler that is registered with the node and performs the vector addition using the latest available data.

The corrected data is then sent back to the NMEA2000 network and labeled as “Theoretical Wind (ground referenced, referenced to True North)”. For hardware-specific reasons, the apparent wind is also transmitted again, as the Garmin display can only choose one wind source and thus otherwise wouldn’t be able to display both apparent and true wind at once.

The communication between the process is handled via `multiprocessing.Queue`s which handle thread and process safety for me.

5.6 Continued Testing

While I was porting the NMEA2000 library I had some more opportunities to test the current iteration. To recap, at this point I had mostly ironed out the bugs of the very basic implementation using only the GPS, and the corrected data should successfully arrive at the server. I had however not yet been able to confirm it working from end to end in real-world conditions as the lack of cellular coverage in the second part of Kiel Week thwarted any testing there.

The tests in Berlin were ill-fated and plagued with hardware issues in the parts of the sensor assembly that I didn’t modify. First, the field-installable connector on the cable connecting the wind transducer to the GND10 Blackbox suffered from water ingress. Removing the connector, drying the insides, and reattaching it seemed to solve this issue, but as I didn’t bring the relevant tools, I wasn’t able to identify the issue until the second



Figure 20: Bare wires extruding out of the bottom of the transducer, replaced with a M8 4-pin socket in the same spot.

day and could only solve it after the day's sailing had finished. The problem persisted at the next testing opportunity even though I had attempted to waterproof the plug with sealing tape however, armed with the knowledge of the water ingress I applied the same fix and got a connection while still in the harbor. Moving out to the racecourse the connection was lost again, and it appeared, that the inner cable had developed a breakage or short circuit.

Because I had to desolder the data cable from the transducer anyway I took up another member of the race committee on their offer to help me add a socket to the transducer assembly²⁰. This would act as both a strain relief for the delicate cables coming out of the transducer, and let the whole mast be transported more easily, as the sensor could be disconnected and protected during transport.

The aluminum plate prevented water from draining away through to the bottom of the mast forcing me to seal the top of the mast with marine sealant, thus creating another unexpected issue. Because the mast is black, water vapor would rise inside it in the morning and get into the solder connections. As they were isolated from each other this did not lead to any shorts just yet but needs to be addressed in the future by sealing the complete transducer assembly.

The issue of the broken cable was still not solved, and I had to wait for a new cable with a molded plug to arrive, which would also solve the water ingress issue of the connector permanently.

Another issue that I encountered was that the cellular modem failed to connect to the internet. While it claimed to have a network connection and did have an IP address, even ICMP packages did not make it back and timed out instead. Due to time constraints this issue was sidestepped and is not solved.

6 Result

The artifacts I've created as part of this thesis can be classified into two categories. First are contributions to libraries and second are all the components that combined make up the prototype wind monitoring system.

6.1 Library contributions

6.1.1 ICM20948

This is a library to communicate with the ICM20948 9DoF motion sensor that is used on the Sense HAT (B). My contribution to it consists of some code that allows the user to take advantage of the onboard FIFO queue to store sample data and read it back later instead of querying each sample individually. The pull request is open but not yet merged and can be viewed at <https://github.com/pimoroni/icm20948-python/pull/21> [10].

6.1.2 NMEA2000

This is a library that aims to enable easy communication with other NMEA2000 devices. It is a port of <https://github.com/ttlappalainen/NMEA2000> with a slightly reduced feature set. It can be found at <https://github.com/finnboeger/NMEA2000> [33].

As it is, the library can be used for most sensor interactions but has not been tested for anything other than wind data. It is also not yet available for installation via the Python Package Index.

6.2 Wind monitoring system prototype

The primary result of the thesis is the working prototype and instructions to recreate it. The capabilities of the system include accurately measuring the true wind speed and direction, and displaying them on the display and a website. To this end, it consists of

- the client code [7],
- the server code [8],
- the 3d design files [1],
- the bill of materials [7],
- and instructions to set up the system [7].

7 Conclusion

7.1 Evaluation

The last regatta of the year finally allowed me to test a working prototype without any unexpected issues. I was on the windward mark-laying boat with the measurement device again and could make some final observations.

7.1.1 Usefulness

The goal of reducing radio chatter has been achieved as the radio traffic asking for wind measurements dropped to almost, but not quite, nothing. The automated wind measurement has been very helpful to have a better basis for decisions than just occasional manual measurements and a general gut feeling. It also allowed for a more confident reading of the wind direction as the boat often rotates (or swings if anchored) making the reading of the wind bearing often a bit of guesswork. With the latest prototype, the corrected measurements were instead shown on the display and the values over time could be graphed as well.

The testing with the working prototype could only be performed on a lake which didn't make it possible to use the measurement at the windward mark as an early warning system. Instead, the wind was simply different due to the influence of the forested shoreline and the general shape of the lake.

The feedback from the PRO has been positive, and he could see and use the wind data to assist in making his decisions. Accordingly, the system will be especially helpful when less experienced people are on the mark-laying boat, who would not actively monitor the wind and communicate any changes themselves. Although the data could be understood on the website a more simplified interface was desired that would remove the apparent wind from being shown. Further data processing that showed trend lines for both the direction and speed graphs was also noted as potentially helpful. The short spikes occurring whenever the boat started moving were noticeable and should be removed to avoid confusion.

7.1.2 Process

A lot of time was lost to hardware problems. As I do not know how these could have been either avoided or resolved faster, I can't add too much to this.

7.2 Outlook

This thesis got the prototype to a working state but still left room for several improvements due to time constraints. As such some initially planned features had to be cut or are not yet working as intended.

This includes working on the inertial sensors and managing to extract the magnetometer data to create a working inertial navigation system. The INS can then be combined with the GPS as planned and improve the responsiveness of the calculation.

7 Conclusion

While the NMEA200 library is in a state that works for the prototype it is not yet feature-complete. Some of this is related to missing support for specific parts of the protocol, but mostly it is missing a lot of PGNs that still need to be implemented. My goal is to expand this library to a comprehensive framework that supports all known messages by leveraging the work of the canboat project and auto-generate the message code. Another aspect that is still missing is proper automated testing. While complete integration tests are only possible if one has access to equipment that understands those messages, it can at least be ensured that the library is consistent and can read its own messages.

The server code also holds room for improvement. Specifically, the layout can be updated to focus on the important information and the amount of data that is sent to visitors can be reduced by only sending the latest data point instead of re-sending the whole graph.

Lastly, the client can be enhanced by solving its connection issues with the cellular network. This most likely requires a hardware change to a different chipset series or at least to the global version of the same chipset. When cellular connectivity cannot be achieved, and the client is installed in a system without the benefit of a display, it would also be advantageous to have a local server running that could be accessed via Wi-Fi and provides the same data processing as the remote server. Another minor issue that could be fixed is the resilience against voltage drops. Currently, starting the motor causes a voltage drop below the necessary threshold of the step-down converter and the device loses power. When the user wants to turn off the device the only option is to turn off the power as well. While I want to keep the device as simple as possible and not introduce a power switch, adding a small battery would allow it to survive short power outages and otherwise a graceful shutdown.

8 Appendix

8.1 Possible itemizations of homebrew alternative

8.1.1 Components of solution seen in Kiel

- Simrad HS70 GPS Compass: 1500€ (or comparable)
- Gill WindSonic 1: 1400€
- Simrad NSS7 evo3 COMBO: 1700€ (or comparable)
- NMEA2000 Backbone: 130€
- Aluminum Spar: 60€

8.1.2 Cheapest possible option

- Compass & GPS Combination sensor: Garmin GPS24xd (290€)
- Wind sensor: NASA Marine Windsystem (300€)
- Offbrand NMEA2000 Backbone: 120€
- Aluminum spar, 3mx50mm, 3mm wall thickness: 60€
- Display: 600€

References

- [1] *3D Print files for the enclosure and mounting brackets*. URL: <https://github.com/finnboeger/mobile-wind-sensor-3d-parts/tree/c9e4d5a94cd3ed1cd90fbb7bd556dadb20d0657c>.
- [2] *470 Class Rules*. 2023-01. URL: https://web.archive.org/web/20221220133304/https://d7qh6ksdplczd.cloudfront.net/sailing/wp-content/uploads/2022/03/02082109/470_CR_2023-01Jan-01.pdf.
- [3] *Autonnic Compas Sensor 2000*. URL: <https://web.archive.org/web/20220523180447/https://www.autonnic.com/product-page/compass-sensor-nmea2000>.
- [4] *CANBOAT*. URL: <https://web.archive.org/web/20220820004632/https://github.com/canboat/canboat>.
- [5] *Canboat getISO11783BitsFromCanId*. URL: <https://web.archive.org/web/20221220152936/https://github.com/canboat/canboat/blob/master/common/common.c#L643>.
- [6] *cauchypotato.com. Set your Android Phone to Automatically Power on when USB Charger is plugged in*. 2016-01. URL: <https://www.youtube.com/watch?v=Zp9G6A6EF1A>.
- [7] *Code for prototype mobile wind monitoring system*. URL: <https://github.com/finnboeger/mobile-wind-sensor-client/tree/147987f229c6f17d4d72357d5a7a4e14872a424d>.
- [8] *Code for server to process and display wind data gathered by client*. URL: <https://github.com/finnboeger/mobile-wind-sensor-server/tree/7c3b6b5f9f838f699c28d274cfed28e295e6c789>.
- [9] *Donation requests denied by PubCharity*. 2018-06. URL: <https://web.archive.org/web/20220707170431/https://www.pubcharitylimited.org.nz/assets/Documents/PublishedDeclined-PubCharity-Jun-2018.pdf>.
- [10] *Feature branch for FIFO addition to IMU library*. URL: <https://github.com/finnboeger/icm20948-python/tree/1cd5e94a1e2f23c4c7106bc2cf70a1132b4163b0>.
- [11] *Garmin GMI20 Display*. URL: <https://web.archive.org/web/20221220135439/https://www.garmin.com/de-DE/p/126694>.
- [12] *Garmin GND10 Blackbox*. URL: <https://web.archive.org/web/20221220135435/https://www.garmin.com/de-DE/p/144123>.
- [13] *Garmin gWind Transducer*. URL: <https://web.archive.org/web/20221220135420/https://www.garmin.com/de-DE/p/144124>.
- [14] *Golden Globe Race*. 2022-09. URL: <https://web.archive.org/web/20221003163853/https://goldengloberace.com/the-race/>.

- [15] *Guide to 3D Printing Materials: Types, Applications, and Properties*. URL: <https://web.archive.org/web/20220807134415/https://formlabs.com/blog/3d-printing-materials/>.
- [16] *HiveMQ Public Broker*. URL: <https://web.archive.org/web/20221123013352/https://www.mqtt-dashboard.com/>.
- [17] *How Events Work - SailGP*. URL: <https://web.archive.org/web/20220922090755/https://sailgp.com/general/sailgp-how-events-work/>.
- [18] <https://github.com/Mayitzin/ahrs>. URL: <https://web.archive.org/web/20221108080125/https://github.com/Mayitzin/ahrs>.
- [19] *ICM20948 9-DOF Accelerometer, Magnetometer and Gyroscope*. URL: <http://web.archive.org/web/20200914072924/https://github.com/pimoroni/icm20948-python>.
- [20] *ICM20948_ WE*. URL: https://web.archive.org/web/20221220150051/https://github.com/wollewald/ICM20948_WE.
- [21] *Introduction to the Controller Area Network (CAN)*. URL: <https://web.archive.org/web/20221220152732/https://www.ti.com/lit/an/sloa101b/sloa101b.pdf?ts=1671343590536>.
- [22] *Kieler Woche Bahnkarte*. URL: <https://www.facebook.com/nvcharts.nvverlag/photos/die-kieler-woche-bahnkarte-kann-auf-unserer-seite-wwwnv-verlagde-f%C3%BCr-den-chart-n/892440287449283/>.
- [23] *Manage2sail digital notice board*. 2022-10. URL: <https://www.manage2sail.com/ch/event/jhgp22#!onb?tab=documents&classId=1beae80c-5b30-40db-8750-09f1f9235c45>.
- [24] *MaxLinear. SP3485 3.3V Low Power Half-Duplex RS-485 Transceiver with 10Mbps Data Rate*. Rev 2.0.2. URL: <https://web.archive.org/web/20221124170435/https://assets.maxlinear.com/web/documents/sp3485.pdf>.
- [25] *MQTT: The Standard for IoT Messaging*. URL: <https://web.archive.org/web/20221218203126/https://mqtt.org/>.
- [26] *n2kd - canboat*. URL: <http://web.archive.org/web/20201101152652/https://github.com/canboat/canboat/wiki/N2kd>.
- [27] *NarrowBand IoT / LTE-M: Die Maschinen- und Sensorennetze*. URL: <https://web.archive.org/web/20220813161916/https://iot.telekom.com/de/netze-tarife/narrowband-iot-lte-m>.
- [28] *Narrowband IoT & LTE-M: Ideal für Maschinen und Sensorennetze*. URL: <https://web.archive.org/web/20221213112355/https://www.vodafone.de/business/loesungen/narrowband-iot/>.
- [29] *NASA Marine NEMA Compass Sensor*. URL: <https://web.archive.org/web/20220701134837/https://www.nasamarine.com/product/nmea-compass-sensor/>.

- [30] *NMEA 2000 PGN's deciphered*. URL: <https://web.archive.org/web/20220524235728/https://endige.com/2050/nmea-2000-pgns-deciphered/>.
- [31] *NMEA 2000 Standards*. URL: <https://web.archive.org/web/20221220153056/https://www.nmea.org/nmea-2000.html>.
- [32] *NMEA2000 library for C++*. URL: <https://web.archive.org/web/20220510132536/https://github.com/ttlappalainen/NMEA2000>.
- [33] *NMEA2000 python library*. URL: <https://github.com/finnboeger/NMEA2000/tree/b37f1a09b0a00ae05ecbfc3ea33b4a6abf5fef>.
- [34] *OpenSCAD - The Programmers Solid 3D CAD Modeller*. URL: <https://web.archive.org/web/20221218180809/http://opencad.org/>.
- [35] *OWS-5 Monitoring System Shop Page*. 2022-12. URL: <https://web.archive.org/web/20221220134551/https://r-p-r.co.uk/weatherfile/ows-5.php>.
- [36] Mark Petovello and Salvatore Gaglione. "How does a GNSS receiver estimate velocity?" In: *InsideGNSS* (2015-03). URL: <https://web.archive.org/web/20221127002139/https://insidegnss.com/wp-content/uploads/2018/01/marapr15-SOLUTIONS.pdf>.
- [37] *pynmea2*. URL: <https://web.archive.org/web/20221219204100/https://github.com/Knio/pynmea2>.
- [38] *SAE J1939 Transport Protocol (TP) Functions*. URL: <https://web.archive.org/web/20210512020812/https://copperhilltech.com/blog/sae-j1939-transport-protocol-tp-functions/>.
- [39] *SAE J1939 vs. CAN Bus - What's the Difference?* URL: <https://web.archive.org/web/20210921163113/https://copperhilltech.com/blog/sae-j1939-vs-can-bus-whats-the-difference/>.
- [40] World Sailing. *Race Management Manual*. 2022-05. URL: <https://web.archive.org/web/20221214093509/https://d7qh6ksdplczd.cloudfront.net/sailing/wp-content/uploads/2022/05/18102900/RM-Manual-Draft-17May2022-Para-Sailing-Section-added-MR-Section-updated-Bookmarks-added.pdf>.
- [41] *SIM7000E NB-IoT / Cat-M / EDGE / GPRS / GNSS HAT*. URL: <https://web.archive.org/web/20220520184941/https://www.waveshare.com/SIM7000E-NB-IoT-HAT.htm>.
- [42] *SIM7600G-H 4G HAT (B)*. URL: <https://web.archive.org/web/20220702110040/https://www.waveshare.com/product/sim7600g-h-4g-hat-b.htm>.
- [43] *SocketCAN userspace utilities and tools*. URL: <https://web.archive.org/web/20221202120150/https://github.com/linux-can/can-utils/>.
- [44] *Waveshare Sense HAT (B) Demo Code*. URL: <https://www.waveshare.com/w/upload/6/6c/Sense-HAT-B-Demo.7z>.
- [45] *What is the Difference Between SAE J1939 and NMEA 2000?* URL: <https://web.archive.org/web/20210624034848/https://copperhilltech.com/blog/what-is-the-difference-between-sae-j1939-and-nmea-2000/>.

- [46] *Wir vernetzen das Internet der Dinge*. URL: <https://web.archive.org/web/20220810150335/https://iot.telefonica.de/iot-m2m-produkte/>.
- [47] *Yacht Devices NMEA2000 to USB Gateway*. URL: <https://web.archive.org/web/20220926041837/https://busse-yachtshop.de/s/Yacht-Devices-NMEA2000-zu-USB-Gateway-YDNU-02NM>.