The Journal of Navigation (2021), **74**:4 738–749
doi:10.1017/S0373463321000205

RESEARCH ARTICLE

Oceanus: a context-aware low-cost navigation aid for yacht racing

Ivan Scagnetto,[1]* Giorgio Brajnik,[1] Peter Gus,[1] and Francesco Trevisan[2]

[1]Department of Mathematics, Computer Science and Physics, University of Udine, Udine, Italy
[2]Polytechnic Department of Engineering and Architecture, University of Udine, Udine, Italy.
*Corresponding author. E-mail: ivan.scagnetto@uniud.it

Received: 30 December 2019; **Accepted:** 20 February 2021; **First published online:** 26 March 2021

Keywords: sail racing, navigation aid, internet of things, context awareness, usability, user experience

Abstract
Oceanus is a hardware and software platform designed and developed to provide useful information to the crew of a racing yacht. The key features of the proposed solution are its reliability, the possibility to extend and customise it, and its context-awareness, which simplifies its usage in an intelligent way. The target users of the system are both beginners who want a navigation aid, but cannot afford the expensive and often overly complicated commercial systems available on the market, and more experienced sailors who can benefit from an open and customisable instrument to study and fine-tune the setup and performance of their sailing boats. Furthermore, Oceanus strives to be as much as possible a low-cost architecture, both in software and hardware requirements.

1. Introduction

Modern racing yachts are equipped with sensors and monitors providing a wealth of real-time information to their crews: kinematic data (latitude, longitude, speed, geographic course, position on a map), boat-related data (magnetic heading, speed over water, angular velocities, and yaw, pitch and roll angles), and wind/water data (direction and speed of apparent wind, water depth and temperature). Consequently, sail racing, even for club and amateur races, has become more and more technology-oriented and technical. Most crews use digital devices providing information on wind and boat speed and direction, in some cases also computing the optimal speed and angle to the wind that the boat should achieve. In many cases crew members also use printed sheets of paper taped on deck that provide target speeds and angles. Such data is used to fine-tune boat steering and sail trimming.

Yacht racing is a suitable application domain for the Internet of Things and ubiquitous, context-aware computing. However, developing affordable and effective software solutions in this scenario is challenging. First, from the architectural viewpoint, there is the need to interface a computer with the onboard bus, to read and process in real time data coming from sensors (e.g., GPS, compass, barometer etc.), while coping with strict limitations about electric power consumption. Second, when racing, crew members are under intense cognitive stress to make the right decision at the right time, to perform corresponding actions with precision and in strict coordination with other members. This can be complicated by challenging weather or marine conditions and the aggressive behaviour of competitors. Thus, the user interface (UI) of an application needs to be crafted with attention. The first law of usability, 'Don't make me think', by Krug (2000), applies well to this context; users cannot spend time and attention on figuring out how to use the UI, even more when they are not professionally trained.

Figure 1. William B*, the yacht used by the UniUD Sailing Lab, and typical conditions on-board during a race.*

The UI should present the minimal information for the task at hand to avoid information overloading. Manual navigation between screens should be kept at the minimum to avoid distractions.

Beside professional solutions, which are often very expensive and hardware-specific (see, e.g., OS5 in Section 7), there are several mobile applications to choose from, but these present several limitations. They require installation of software on each crew member's device, that sometimes might not be compatible; furthermore, such devices might not be appropriate for the context (e.g., too small/low-quality screens, not enough memory or lacking some important sensors). The UIs are often quite complex, as the applications are not targeted to specific usages.

This paper presents Oceanus, a platform designed to explore and develop intelligent features aimed at supporting yacht racing crews. Oceanus encompasses a backend system (Argos) and a frontend system (Neptune). From the user point of view, it is a web mobile application based on low-cost hardware (Raspberry Pi for the server-side and any mobile device with a browser for the client-side. Amazon's Kindle was chosen in order to have an affordable screen readable in direct sunlight) and provides a UI that is tailored to a racing yacht crew. This paper shows how and why Oceanus satisfies most of the crucial requirements suggested above. In particular, Oceanus can 'sense' the environment (using the data coming from sensors on the boat) and 'interpret' the current situation (i.e., the context), in order to simplify the UI, providing only the relevant information to the user. This key feature is known as context-awareness and it will be discussed in Section 3.1. Oceanus has been developed by the UniUD Sailing Lab (Figure 1) and has been used in training and during club sailing races and more important championships (namely, the 2017 Italian Offshore Sail Race and the international 2017 ORC Worlds Trieste).

The contribution of this paper lies in: (i) identifying effective context-aware features that improve usability for a racing crew; (ii) defining a simple but effective and low-cost architecture that can be deployed easily on most yachts; (iii) showing an example of a UI that caters for racing crews.

2. Oceanus system architecture

Figure 2 represents the setup of Oceanus on-board the research team's sailing boat. The software system follows a client/server architecture encompassing the Argos server-side process (which reads

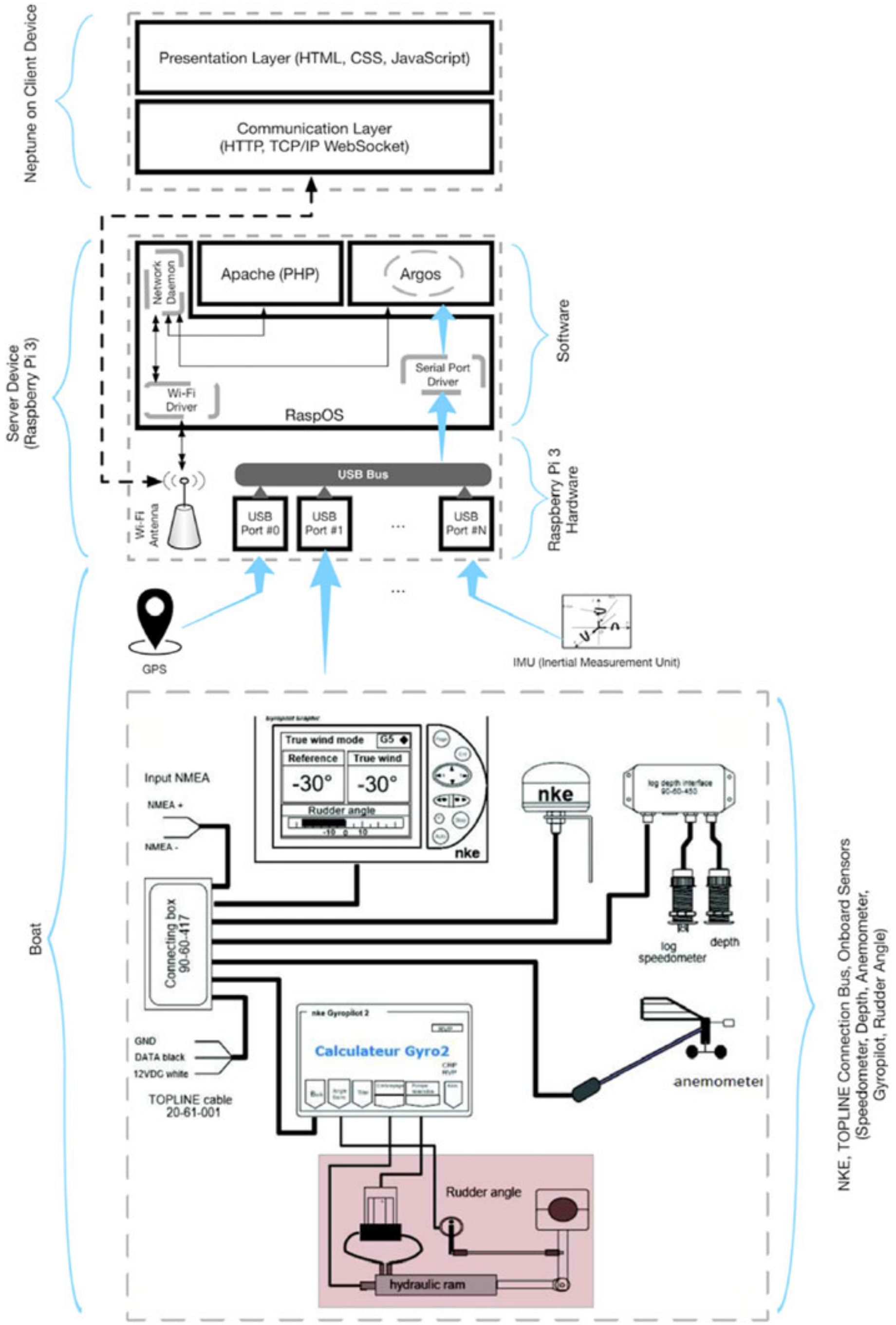

Figure 2. *System architecture of a typical setup of Oceanus (Argos + Neptune). The autohelm part of the figure is shaded, because we cannot control it; indeed, the Oceanus platform can only read the rudder angle.*

and processes data from the on-board sensors) and a client-side web interface named Neptune. The latter is a web application hosted and distributed to clients using the Apache web server. From the hardware perspective, the core element is a Raspberry Pi 3, which:

1. harvests raw data from sensors;

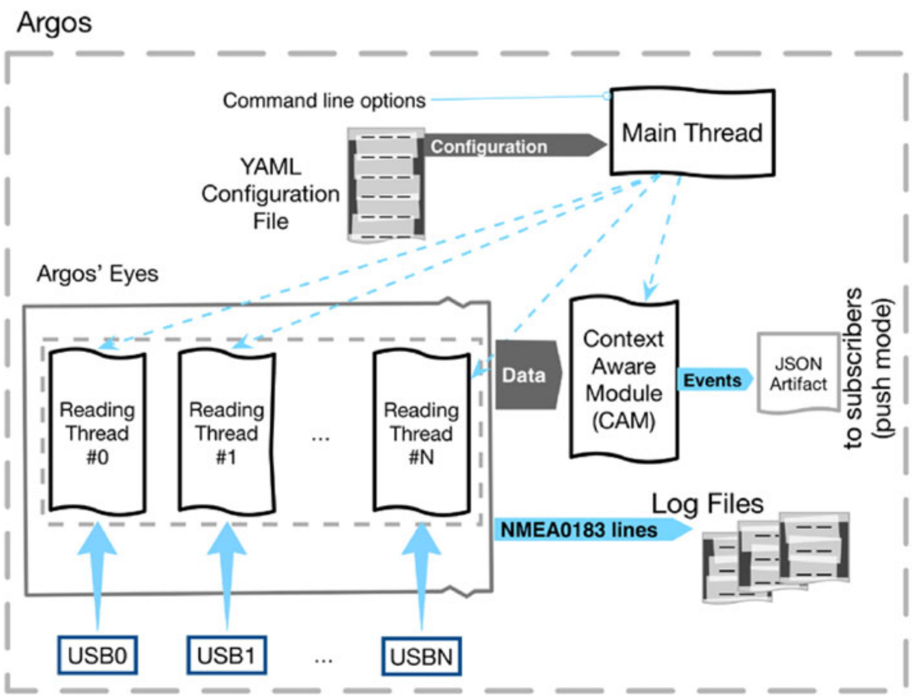

***Figure 3.** The Argos process.*

2. computes useful information from raw data;
3. uses the computed information to coordinate context-awareness, generating events (e.g., signalling if the boat is ‘downwind’, ‘upwind’, ‘before/after the start of a race’, etc.);
4. provides a Wi-Fi local area network (LAN);
5. publishes the computed information and the related events through a publish/subscribe service and a web application in the LAN.

Figure 2 denotes the multi-tiered architecture: on top of the boat hardware (a standard factory installation featuring the NKE TOPLINE bus), the operating system (RaspOS) provides drivers to manage USB ports (connected to sensors and the on-board equipment) and the Wi-Fi antenna, and the network daemon to establish the wireless LAN and to manage connections with clients.

The Argos process (see Figure 3) performs the following tasks:

1. after launch, it reads the configuration file in YAML format (https://yaml.org/), learning which are the sensors to use and how to setup the Wi-Fi LAN;
2. it reads data in real time, storing data in log files in NMEA format (https://www.nmea.org);
3. it computes secondary data from raw data and keeps it in main memory (for example, estimation of marine current);
4. it uses the context-aware module (CAM) to apply the built-in rules to computed data, in order to infer new events (see Section 5);
5. it notifies such information, in JSON format (https://www.json.org/), to subscribers (clients) using websockets.

An Apache web server provides the dynamic pages of the web interface (Neptune) to the client devices, allowing the crew to access the relevant information during normal on-board activity. Finally, the client layer consists of the Neptune frontend, which includes the communication layer (talking with Argos and Apache) and the presentation layer (i.e., the GUI, driven by the events published by the Argos service).

Calibration of sensors is an important aspect of the reliability and accuracy of Oceanus. For example, boat velocity is measured by means of a GPS (velocity with regard to the ground) and the velocity measure with regard to the water; usually (in the region where the research was conducted) their difference is within the range of 1 knot, according to the intensity of the water current. When this difference is above the specified threshold, Oceanus sends a calibration warning, indicating a measure error usually affecting the speed with regard to the water. It is envisaged that users will perform calibration procedures, before the start of a race, to automatically measure the current and set a corrective factor. Similarly, the magnetic compass and the GPS course differ by magnetic deviation within the range of 3 degrees. If this threshold is passed, then Oceanus provides a warning and, moreover, stores the corrections for different courses in the magnetic deviation table. Finally, for the detection of GPS outages, two GPS devices are placed aboard at a known distance (10 m) and, if the computed distance between the two differs above a specified tolerance, Oceanus provides a warning which gives an extent of the error affecting the GPS system.

3. UI for racing crews

The crew of a small racing yacht (9–12 m) is constituted by individuals with different roles; those involved with Neptune are:

- helmsman, who steers the boat and whose main responsibility is to maximise its performance in terms of speed and course;
- sail trimmers, who tune the sails and whose main responsibility is to adapt sail shapes to continuously changing wind, sea and boat handling conditions;
- tactician, who gives directions regarding where the boat should be going and whose responsibility it is to make strategic and tactical decisions, by exploiting wind and sea conditions and reacting to the behaviour of other competitors.

During a race, individuals concentrate on their tasks, observing the race field, competitors, sea and weather conditions. This requires high levels of attention and quick reaction times as in some cases situations may be safety-critical, leading to vessel damage, injuries, or to race-related issues (losing position or incurring penalties). Cognitive stress and physical challenges may arise from bad weather, strong wind and long duration of a race. The consequences are poor equilibrium on-board, inability to move around freely, or the need to move to other places (e.g., to counterbalance excessive heeling caused by wind). Hands are busy operating boat equipment; crew members wear sailing gloves; hands may be wet. Races may cover an entire day or more. Fatigue easily ensues. In general, as well, during the day there is strong direct sunlight, and at night there is no illumination, except for controlled light sources (such as red light to preserve natural night vision). This kind of physical environment, and the user roles and conditions, constrained how Neptune was built. The researchers opted for a low energy consumption user device, with a high visibility screen in direct sunlight, with a large view angle, and one that can be easily handheld or attached to the body (of the tactician) or placed in the boat cockpit for the benefit of the helmsman and trimmers.

3.1. Context and context-awareness

Context is an overloaded word, with many different interpretations. Dey (2001) provides one of the most general and often used definitions of context, namely, ‘any information that can be used to characterize the situation of an entity’. Entities can be people, places or objects relevant to the interaction between a user and an application, including the user and the application.

Nardi (1996) links context to distributed cognition, which is a cognitive discipline studying the representation and the propagation of knowledge among individuals and in the world, and the transformations that individuals and artefacts cause on external structures, according to Flor and Hutchins (1991). Distributed cognition focuses on a cognitive system composed of individuals and the artefacts they use; for example, Hutchins (1995) describes the activity of flying a plane and focuses on the cockpit system. The cockpit, with its pilots and instruments forms a single cognitive system, which can be understood only when we understand, as a unity, the contributions of the individual agents in the system and the coordination necessary among the agents to enact the goal.

Neptune is a component in a distributed cognition system. Some of its features are 'cognitive prosthesis' aimed at enhancing the crew capabilities; the boat's cockpit that also includes Neptune forms a single cognitive system. In this case it is the crew, rather than a single individual, that is part of this system. In fact, context-awareness is often determined based on relevance of data with respect to the entire crew, rather than to individual crew members. In Neptune, context is related to physical and time data, only; no data related to the user or the computing infrastructure are used.

Dey (2001) defines a system as context-aware if 'it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task'. The purpose of context-aware features in a system is to minimise user effort, enhancing usability and enabling a better user experience. This is also the purpose of Neptune. The attention and concentration required by the primary tasks (boat handling, sail trimming, awareness of competitors and sea/wind conditions) leave little cognitive capability to be spent for thinking how to interpret what a UI displays, or how to navigate through its screens; in addition, the physical situation leaves little ability to act physically on the UI and monitor its feedback.

Context-awareness features support the user in performing the primary tasks, by automatically switching to relevant screens, by displaying and highlighting the most relevant information. Most of the data that could be displayed on a sailing boat are contextual, e.g., data on the boat (location, speed, orientation, course) or the wind (direction and speed, as measured from an on-board wind station). Context-awareness though requires that data are task-relevant, and this is much more complex to achieve with accuracy. Since Neptune is used in a demanding environment, context-awareness must be based on reliable decisions about what is the current task of the user. If such decisions are data-driven, they require appropriately filtered data, otherwise chattering data are likely to lead to instability of the system.

In Neptune, task relevance is addressed in two ways. For the helmsman and trimmers, who cannot easily reach the UI, Neptune runs on a device that is located in the cockpit and supports the crew during the start of the race or by monitoring the boat performance in terms of speed and course. Context-awareness is achieved by automatic switching of screens based on gathered data: after the start time the display switches from the starting screen to the appropriate target speed/angle monitor; after tacking, visual cues are changed accordingly; different points of sail drive changes in the information shown.

Before or after the start, what is displayed and how it is displayed depends on the data. Just before the start of a race, if the boat position is beyond the starting line, a very salient display is used to alert all the crew. During sailing, according to the point of sail and to the tack, different data are displayed based on polar curves of the boat to let the crew optimise speed and course. These changes are driven by different combinations of point of sail and tack, which are automatically sensed.

For the tactician, Neptune runs on a device that is supposed to be handheld. Because the task of the tactician is much more varied and they can use their hands to operate the device, the user can decide what screen to display when. In fact, the start-assistance features are relevant also to the tactician before the start, but during the whole race the tactician might need repeatedly to check wind and marine current conditions.

Neptune performs a type of interaction defined as 'active execution' by Alegre et al. (2016); i.e., the system acts autonomously depending on the context. No configuration is done, either by the user or automatically.

Context-awareness in Oceanus was approached by aggressively restricting the range of activities to be supported and carefully selecting what to display, in order to ensure reliability. Indeed, displaying inaccurate information or irrelevant screens could completely void the value that the crew attaches to the system. A consequence of this point of view is that context-awareness in Neptune is determined based solely on data of the boat and of the environment, rather than assumptions about what task the user is performing.

In a system like Neptune there are many data that can be potentially relevant, and it is difficult to understand which ones are critical in which situation. Interviewing crew members and asking opinions or performing user tests on low-fidelity prototypes are usability techniques (Rubin and Chisnell, 2008) that were deemed not to be sufficiently effective. Instead, testing a live prototype with real data and possibly on-board, by observing real users using the prototype, was thought to be a more effective strategy. Therefore, a lean UX approach was adopted. As defined by Gothelf (2013), lean UX design 'implements functionality in minimum viable increments and determines success by measuring results against a benefit hypothesis'. Emphasis was put on the expected outcomes of design choices made on Neptune, on how features are used and what their benefits are. The development process was driven by usability evaluations, and it relied on iterative building of prototypes that were empirically evaluated to gather insights and improve the solutions. Moreover, exploratory testing and usability testing were carried out in the laboratory using data recorded during prior training sessions or races. All the context-aware features of Neptune were tested in this way.

4. User-level requirements and examples

Requirements for Neptune were formulated mostly as 'user stories' emphasising the value that they bring to the sailing crew or to how they help better understanding of the problem or the technologies that are used (Cohn, 2009). User stories were also associated with acceptance criteria: conditions that can be tested and used to determine if the user story is done.

Some of the focal user stories follow (available space for this paper prevents us from presenting all the relevant ones):

- As a helmsman, I can monitor the target data for the current point of sail so that I can react quickly to improve boat performance as conditions change. Target data depend on the actual point of sail and include error of the wind angle and of the boat's speed.
- As a trimmer, I can monitor the target data of the boat so that I can improve sail trimming.

The screens in Figure 4 help the helmsman to correct steering by suggesting that a change of course is needed to diminish the error in boat speed and pointing. The same data can be used by the sail trimmer to understand that better handling of sails is called for. With a cross-wind the notion of target speed changes. Rather than being defined as the maximum achievable speed at which the boat can reach a buoy that is perfectly upwind or downwind, with wind-abeam the target speed is defined as the maximum achievable speed along the boat's course.

To reduce human interaction, Neptune takes advantage of changes in the points of sail and tacks detected by Argos, and it switches mode as needed.

Other user stories deal with the starting phase:

- As a tactician, I need to:

o Get GPS fixes for both endpoints of the starting line or its bearing to prepare a plan for the start.
o Get more than one fix for each endpoint to average out small fixing errors.
o Visualise which is the favoured endpoint of the line and quantify the gain.

- As a helmsman and as a tactician, I need to:

o Estimate how far is the starting line (as a distance or time) so that I can change boat speed and course to cross it timely and fast.

Figure 5(a) and (b) show the start screen, with data concerning the starting line (position of endpoints, length, direction), the boat (distance and time to each of the endpoints, distance from the starting line, gains, if the boat is beyond the line) and the timings (countdown in seconds, projected time to reach the line). Neptune automatically changes the mode of the start screen depending on boat position and timing. It also automatically switches from the start to the target screen after the start time and 15 s after the line has been crossed.

Other user stories are related to strategy:

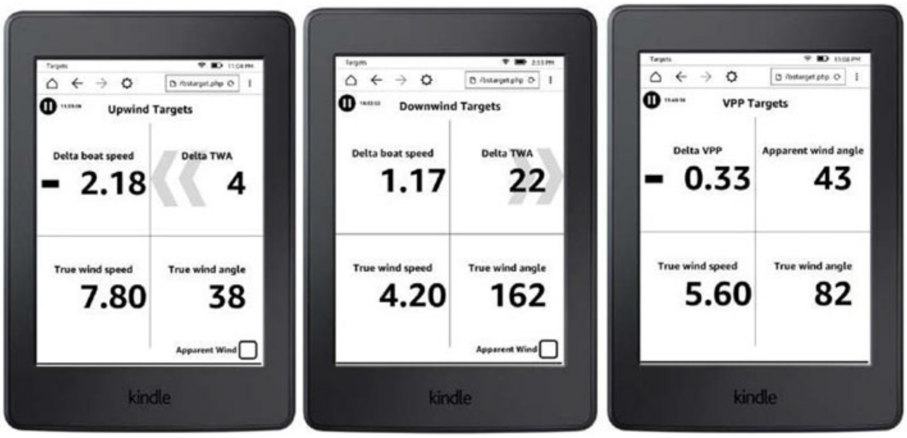

Figure 4. *(Left) The upwind target screen shows the error of the boat speed (too slow by 2·18 knots), the error of the true wind angle (4 degrees too close to the wind), the actual speed and angle of the true wind. The cues (background arrows) suggest steering to the left, away from the wind. (Centre) Similar data, but when sailing downwind. (Right) The wind-abeam target screen, that shows the speed error (too slow by 0·33 knots) along the current course, not in the direction of the wind as before.*

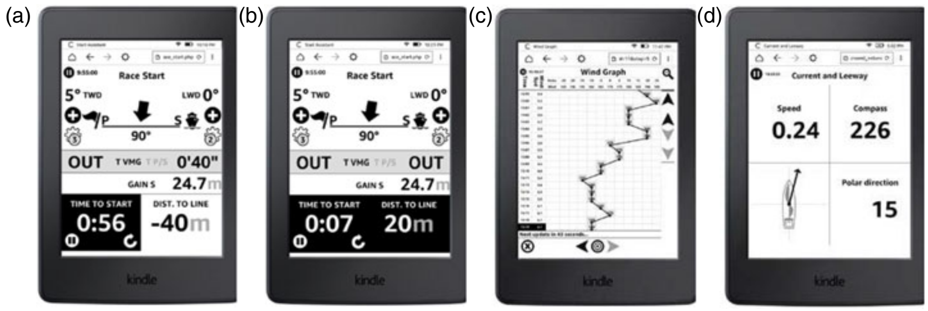

Figure 5. *(a) The start screen, showing that there are three fixes for the left end of the starting line, two for the right end, and that the wind is tilted 5 degrees to the right with respect to the normal of the starting line. This creates a strategic advantage for competitors starting close to the right end, who gain 24·7 m. The current shortest distance to the line is 40 m, and in 40 s by sailing at the target speed the boat will cross the line, i.e. 16 s ahead of time. (b) Similar data as in (a), but here the boat is 20 m beyond the line just 7 s before the start. The bottom part is highlighted and blinks to emphasise that the boat crossed the line too early. (c) The wind graph screen shows the trend in wind changes, sampled every minute. The two leftmost columns show the time and wind speed, and the corresponding cells of the grid show the wind angle. In the last 11 min the wind rotated counterclockwise by 35 degrees and increased from 3·3 to 6·1 knots. The next update of the screen will occur in 43 s. (d) The current and leeway screen, showing speed and direction of the combined action of the marine current and lateral push of the wind; it is estimated to be 0·24 Kt 15 degrees to the right.*

- As a tactician, I need to:

o Monitor trends in speed and direction of the wind every minute to predict whether the wind is oscillating/rotating and increasing/decreasing.
o Know how long it will be to the next update of the screen, to avoid staring at it continuously.
o Scroll back to previous data and jump directly to the most recent data, without spending time operating the UI.
o Estimate direction and speed of marine current to decide when to tack to reach a buoy.

Wind changes and marine currents are major factors considered in a race strategy. Figure 5(c) shows how wind data is displayed every minute in a visually salient way. Data is continuously drawn on the grid, the viewport is automatically scrolled to include the most recent data, the user can scroll the data up and down, and can zoom in/out.

Figure 5(d) also shows the screen with estimates of current and leeway. Such data are computed by comparing the magnetic heading over the last GPS fixes, and therefore estimating the cumulative effect of marine current and the boat leeway.

During the project, which lasted for about 12 months, several design and development iterations were executed, each resulting in a new prototype being built. Initially, static sketches were evaluated during informal user testing sessions (using the 'Wizard of Oz' technique). Later, executable prototypes and the actual system were used.

The feedback collected from three crews was rich. Iteration after iteration, each of the main screens of Neptune was revised and improved. The main purpose was to improve the ergonomics of the controls and quick understanding of data, and to make sure that only relevant information was shown. For example, on the Target Data screen directional cues were initially present also for the velocity error, not only wind angle. After two user testing sessions the researchers learned that only the wind-angle cues were necessary. Furthermore, the target speed was initially shown only for clause-hauled or running point of sails. After a user testing session it became obvious that the same could be useful also for broad and beam reaches. Several iterations helped optimise the Wind Graph screen. For example, the researchers learned that no manual scrolling was needed since a simple auto-centring was sufficient to keep the most relevant data visible. Likewise, the researchers learned that the most effective time resolution was 1 min. For the Start Assistant screen, the researchers learned that manual inputting of the angle of the starting line was not the most preferred interaction, and therefore it was made a second-level choice. Several improvements were made also on the Current & Leeway screen. For example, an initial screen included animation of the boat diagram: unfortunately, the flickering that ensued when on-board the boat made it quite annoying, and it was removed. Likewise, animation of the leeway and current vector was also flickering too much and was removed.

5. CAM

This section illustrates the algorithm implemented in the CAM component of Argos (Figure 3) and the way it triggers changes in the UI, according to the current context. CAM is capable of recognising a number of different contexts and of generating events which are pushed to Neptune. Such events carry all the information needed to trigger appropriate actions by the frontend UI. At the present stage of development, the recognisable context elements are:

1. type-of-navigation: which can be one of starting-a-race/normal-racing/normal-training/calibration; starting-a-race can be before-the-start/after-the-start;
2. point-of-sail: upwind/downwind/beam-reach combined with tack (port/starboard);
3. manoeuvre: tacking/gybing;
4. role: tactician/helmsman/trimmer;
5. environment: sudden-change-of-wind (speed and/or angle), proximity to either endpoint of the starting line.

CAM recognises such context elements by analysing data from sensors and using derived information, according to the following rules:

1. point-of-sail is detected by classifying standard navigation variables: Apparent Wind Angle (AWA), Speed Over Water (SOW), Apparent Wind Speed (AWS);
2. type-of-navigation is triggered by the user via the UI for starting-a-race/before-the-start, and by a countdown after-the-start;

3. manoeuvre (tacking/gybing) is detected by classifying the following navigation variables: Apparent Wind Angle (AWA), Speed Over Water (SOW), Course Over Ground (COG), Magnetic Heading (MH), and Apparent Wind Speed (AWS);
4. role is triggered by user input;
5. environment (sudden-change-of-wind) is detected when at least one of True Wind Speed (TWS)/True Wind Angle (TWA)/Magnetic Heading (MH) changes significantly with respect to the last trend. Proximity of endpoints is based on classifying GPS fixes.

The following combinations of context conditions then trigger some actions on the frontend UI:

1. point-of-sail and normal-racing/normal-training trigger the display of target screens;
2. before-the-start and input by the tactician trigger the display of the starting screen UI;
3. after-the-start switches to normal-race/training after a countdown;
4. starting-a-race and no tactician input trigger the display of the target screen;
5. training and tacking/gybing trigger the display of the manoeuvre-performance screen;
6. calibration: not-racing and ‘at least 3/4 of a circle has been followed’ start the marine current calibration procedure.

The last two rules are used in the currently developed enhanced version of Neptune.

In conclusion of this section, note that CAM has been designed to be completely parametric, i.e., the definition of context elements, rules and events can be changed easily, in order to extend the use of Argos in new scenarios.

6. Technological aspects and challenges

The hardware used for Oceanus is a Raspberry Pi for the server part and Amazon’s Kindle e-readers as mobile clients. However, any equivalent Linux-based single-board computer could be used as the backend and any mobile device with the following features is a good candidate for the frontend:

1. display readable in direct sunlight;
2. long-lasting battery;
3. a JavaScript-capable web browser.

Moreover, almost all devices can indeed be used as clients because the UI automatically adapts itself to the target client screen. Smartwatches were also considered, but it turned out that they are not very suitable in a yacht racing scenario, for the following reasons:

1. displays are definitely too small, missing important UI elements (there is no effective way to fit all the needed information);
2. to read data, people are forced to gaze at little screens, becoming distracted from the race and in precarious balance conditions;
3. a smartwatch is definitely a single user device, whereas a single screen in the cockpit can be simultaneously viewed by several crew members.

Another important remark is that using a web application does not yield noticeable latency delays. As an example, tacking and gybing require a time of the order of tens of seconds, while data are updated and communicated asynchronously every 200 ms. This means that, even if E-Ink displays perform less well both in refreshing the screen and in handling gestures, these limitations do not lead to shortcomings of Neptune: updates are provided in time to crew members, and gestures other than taps would not work in that environment (hands with gloves, wet fingers, postural instability).

As for power consumption, the Kindle battery lasts for about 5 h, and it can be easily recharged with a power bank. Powering the Raspberry Pi with a battery pack with a capacity of 8 Ah/5 V, gives an operational autonomy of about 26 h. Hence, power consumption is not a problem for the average duration of a race (less than 8 h). For offshore races lasting more than 24 h a backup battery pack or the on-board boat batteries can be used through a USB adapter.

7. Related work

Neptune works well with Kindle, but it does not require ad-hoc devices and it relies on a low-cost client-server web architecture. Its UI is tailored for racing, providing distraction-free displays. In contrast, many similar systems are based on native apps (iRegatta by Zifigo, ESA Regatta by ASTRA Yacht, SailRacer by UAB SailRacer). That approach leads to the advantage of providing a better performing and device-tailored UI, but there are drawbacks:

1. both data harvesting and computation are carried out on each mobile device available to the crew, requiring more complex configurations with possible incompatibilities, which may lead to inconsistencies;
2. in general, native apps are much more complicated to develop: hence, they are not well suited for an experimental framework, where flexibility is the key, in order to implement different variants, new features, etc.;
3. native apps are tightly coupled with specific devices: hence, to cover a wide range of solutions, it is necessary to develop and maintain many versions of the app (e.g., for Android, iOS, Windows);
4. not every device allows development of native apps.

OS5 (by Ockam) is a solution that includes advanced screens for many aspects of a sail race. It supports crews with advanced ways of using the polar diagrams; it supports cost/benefit analysis of tacking; it provides suggestions on crossing the starting line at maximum speed; it lets the crew get the starting line fixes without having to pass by; marine current is detected automatically and included in the calculation. However, its architecture requires a traditional Windows-based PC to be running on-board and connected to a Wi-Fi router; OS5 runs within a web server and users can use their own mobile devices. The UI of OS5 is not optimised for high visibility and no context-aware features seem to be made available. Energy consumption of OS5 (PC, router and screens) is also likely to be an issue on-board small yachts.

In contrast, Neptune has the advantages of a centralised approach. Sensing and computation are carried out server-side: clients are not overloaded by complicated computations. Hence, mobile devices can be simpler and cheaper. Moreover, adding new sensors is easy since they can be configured as additional serial ports of the server, without requiring substantial changes to the client software. Being a web application, Neptune can be used without user installation. It runs on any mobile browser; however, when used with a Kindle Paperwhite, it exploits the E-Ink display technology and backlight and a significant battery life. Extra data sources can be easily added to the Neptune server as well. Compared with OS5, Neptune relies on a cheaper and a more energy-efficient infrastructure.

Regarding the UI, Neptune shares some features with SailRacer (visual cues for correcting target errors) and with iRegatta (detecting when the starting line is crossed). Other features are unique to Neptune. The targets screen can switch between the appropriate readout sets, depending on the context, offering ways for monitoring marine current and leeway. Moreover, the wind screen in Neptune provides better readability by plotting wind direction on a grid along with speed data, rather than more conventional time-series plots. Furthermore, the start screen is very flexible, as it allows the tactician to get multiple fixes for both ends of the starting line and compute the coordinates of the average point. It supports partial detection of the line, by taking the line bearing and fixes for a single end, or just the line bearing.

Furthermore, Neptune is based on a context-aware logic that proactively proposes significant information and visual hints to the crew, reducing the cognitive load and simplifying the interaction. Such features rarely show up in commercial generic products, where users must go through a customisation of the information shown in the displays, if they do not want to be overwhelmed by too many details. Finally, Neptune is built on top of, and will be released as, open software which runs on a low-cost hardware infrastructure; hence both the amateur and the professional sailor can modify and extend it according to their needs, by interfacing new hardware and providing new features.

8. Conclusion

Based on the results of several tests and feedback collected from different crews, the Oceanus platform represents an interesting low-cost solution, providing a simple yet effective context-aware navigation aid integrating several sensors, suitable for both sailing race crews and amateur sailors.

In the future, the authors plan to enrich the UI of Neptune with new screens, e.g., for monitoring tacking performance and for automatically switching to them when a tack manoeuvre is entered. In addition, besides the ability to integrate other sensors, the authors plan to work on the CAM component of Argos. Currently, it is customised for a very specific scenario, but it could be made more general and easier to reconfigure so that it can support different kinds of UIs for different scenarios, like cruising. Finally, other research openings are concerned with installing appropriate sensors on buoys and making Oceanus capable of detecting them. Then, by deploying Oceanus on all the boats of a race fleet, it would be possible to manage a distributed system of sensors, providing a dynamic map of the race field with respect to boats, winds and sea conditions.

References

Alegre, U., Augusto, J. C. and Clark, T. (2016). Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, **117**, 55–83.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Boston, MA: Addison-Wesley Professional.

Dey, A. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, **5**(1), 4–7.

Flor, N. and Hutchins, E. (1991). Analysing distributed cognition in software teams: A case study of team programming during adaptive software maintenance. In: Koenemann-Belliveau, J., Glenn Moher, T. and Robertson, S. P. (eds.). *Reading in Groupware and Computer Supported Cooperative Work*. San Mateo, CA: Morgan-Kaufman.

Gothelf, J. (2013). *Lean UX: Applying Lean Principles to Improve User Experience*. Sebastopol, CA: O'Reilly Media, Inc.

Hutchins, E. (1995). *Cognition in the Wild*. Cambridge, MA: MIT Press.

Krug, S. (2000). *Don't Make me Think*. Berkeley, CA: New Riders.

Nardi, B. (1996). Studying context: A comparison of activity theory, situated action models, and distributed cognition. In: Nardi, B. A. (ed.). *Context and Consciousness: Activity Theory and Human-Computer Interaction*. vol. 69102, Cambridge, MA: MIT Press.

Rubin, J. and Chisnell, D. (2008). *Handbook of Usability Testing*. (2nd ed). Indianapolis, IN: Wiley.

Cite this article: Scagnetto I, Brajnik G, Gus P, Trevisan F (2021). Oceanus: a context-aware low-cost navigation aid for yacht racing. *The Journal of Navigation* **74**: 4, 738–749. https://doi.org/10.1017/S0373463321000205