Oceanus: a context-aware low-cost platform for yacht racing

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

[1]Dept. of Mathematics, Computer Science and Physics, University of Udine, Italy
[2]Polytechnic Dept. of Engineering and Architecture, University of Udine, Italy

February 1, 2019

Abstract

We present Oceanus: 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 and context-awareness, in order to simplify in an intelligent way the usage of the user interface. Furthermore, Oceanus strives to be as much as possible a low cost architecture, both in software and hardware requirements.

Keywords: Internet of Things, Sail Racing, Context Awareness, Usability, User Experience

1 Introduction

Modern racing yachts are equipped with several sensors and monitors that provide a wealth of information to their crews. Sensed or computed data range from GPS-related information (position, speed, geographic course, position on a map), to boat-related information (magnetic heading, speed over water, heeling angle), and to wind and water information (direction and speed of apparent wind, water depth). Sail racing, even for club and amateur racing, has become more and more technological: for example, most of the crew members use polar diagrams of the boat which indicate, for several points of sailing and wind speeds, the optimal speed and heading that the boat is capable of. Such data is used to continuously fine tuning boat steering and sails trimming, to achieve performances that go beyond the level that the average crew can reach based only on the "seats-of-the-pants" experience[1].

[1]See more details on polar diagrams for example on `https://www.yachtingworld.com/features/5-tips-developing-polar-diagrams-optimise-speed-71464`.

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

While large sailing teams (such as America's Cup or Volvo Ocean Race teams) can rely on high tech solutions, either on-board or on-the-ground, this is not the case for teams with much smaller budgets. Less sophisticated IT applications to be used on board of yachts do exist, but software developers face a number of challenges because of the particular context in which these applications are used, namely sail racing.

The context is characterized by the fact that crew members, while racing, are under intense cognitive stress due to the need of making the right decision at the right time, performing corresponding actions with precision, and in strict coordination with other members. This can be complicated further by challenging weather or marine conditions, and the aggressive behavior of competitors (see Fig. 1).

For these reasons the user interface (UI) of such applications needs to be crafted with attention. The first law of usability of Steve Krug, "Don't make me think" [Kru00], applies very well to this context: users, which often are non professional sailors and hence not professionally trained, cannot spend time and attention on figuring out how to use the UI. They have to be fully concentrated on their sailing tasks and do not have extra cognitive bandwidth to spare. Therefore screens of the UI should present the minimal information that suffice to the task at hand in order to avoid information overloading. Manual navigation between screens should be kept at the minimum to avoid distractions.

Furthermore, this kind of applications needs to provide support to different crew roles: the helmsman needs data to optimize boat steering, sail trimmers need data to fine tune sail shapes, and the tactician needs support for figuring

out strategic opportunities.

Development of suitable applications is made even more complex when we consider that environmental conditions under which the applications are used are generally quite demanding: in addition to running on devices being reasonably waterproof, other requirements come from the need to cope with strong solar illumination at daytime, nighttime use that preserves night vision, low energy consumption especially for mobile devices.

In the market there are mobile applications that can be used in these situations. However, they have limitations. They require installation of software on one own's devices, that sometimes might not be compatible. Different crew members need to install it on their own devices. Such target devices might not be appropriate for the context because they do not meet the requirements mentioned above. Furthermore, they are neither extensible nor customizable, in order to adapt to new or specific scenarios. As a consequence, the UI is often quite complex, as the application is not targeted to specific usages. Because of the generality of the application, context awareness is seldom implemented, in spite of the important role it can play in simplifying the UI. For example, context awareness can be exploited to limit the amount of shown information to only what is relevant to the specific context; context awareness can help in reducing controls made available in the UI to only what is needed at a given time.

In this paper we present Oceanus, a hardware and software platform designed to explore and develop intelligent features aimed at supporting yacht racing crews. Oceanus encompasses a frontend system (Neptune) and a backend system (Argos); it is a web mobile application based on low cost hardware (Raspberry Pi and Amazon's Kindle) and provides a UI that is tailored to a racing yacht crew. We show how and why Oceanus was developed in a certain way, how it satisfies most of the crucial requirements suggested above, how it features context awareness as a key to simplify the UI, and how adoption of a "Lean UX" and agile approach [Got13, Coh10] was beneficial to obtain a successful result.

The application has been developed by the Uniud Sailing Lab, and has been used in training and during club sailing races and more important championships (namely, the 2017 Italian Offshore Sail Race and the 2017 ORC Worlds Trieste[2]).

The contribution of this work lies in showing how such an application can be developed as a client-server architecture, which has not been done before on low cost hardware. We show how this decoupling leads to benefits such as ability to use multiple clients for different crew members, ability to use different UI platforms and extendability of the platform. The frontend is a rich Internet application that can be modified independently from the server. The server can be easily extended with modules that exploit new sensors (such as Inertial Motion Units, IMUs), and can be developed in such a way that each sensor is monitored in a concurrent way, to avoid low latency issues due to the presence

[2]See `https://www.campionatoitalianoaltura2017.it` and `http://www.orcworlds2017.com`

of sensors with differing response rate. We show how context awareness can be conceived as an intelligent way to simplify usage of the UI that becomes thus adaptive, based on inferences made on data provided by sensors. By design, we adopted the stance that context-aware features should be as reliable and useful as possible to maximize acceptability of Oceanus by the crews. For this reason their design has been particularly conservative.

At this stage, Neptune is the first frontend included in Oceanus and it relies on a context-aware module which has been implemented server-side, in order to support clients that are as thin as possible. Finally we explain how the software development process was organized, especially in relation to the inability to frequently test the application in the actual situation for which it was conceived, namely during races.

2 Neptune: Presenting Data to Racing Crews

This section describes the problem and the constraints related to the development of the front-end system, Neptune. Then, we describe and motivate the decisions we made in terms of context awareness and development approach.

2.1 Target Environment and Users

Neptune is used on board of a racing yacht (in the specific case it has been used in a 35 footer), by a racing crew during races and training. The crew is typically constituted by 7 individuals. The crew roles that are mostly involved with Neptune are:

- the helmsman: the person who steers the boat and whose main responsibility is to maximize the sailboat performance (in terms of speed and course);
- the sail trimmers: the persons who tune the sails (jib, genoa, mainsail, spinnaker) and whose main responsibility is to adapt sail shapes to continuously changing wind, sea, boat handling conditions;
- the tactician: the person that gives directions to the crew in terms of where the boat should be going and whose responsibility is making both strategic decisions by exploiting wind and sea conditions and tactical decisions related to the behavior of other competitors.

During a race, individuals are very concentrated on their tasks, by handling the boat and sails, by observing the race field, competitors, sea and weather conditions. This requires extremely high levels of attention, and quick reaction times as in some cases situations may be challenging. There could be non trivial human safety issues, risks for damages and issues related with the race (losing positions or penalties).

Not only cognitive stress occurs, but also physical challenges abound as well: they may derive from bad weather, strong wind, long duration. The consequences are poor equilibrium on board, inability to freely move around, in some cases the need to move to other places (for example to counterbalance excessive heeling caused by wind). Hands are usually busy operating boat equipments and cannot easily be used to operate mobile devices; crew members may wear sailing gloves; hands may be wet. Races may last or cover entire days, and for distance races they extend over more than 24 hours. Fatigue easily ensues.

In general, during days there is a strong direct sun light, and during nights there is no illumination, except for controlled light sources (such as red light to preserve natural night vision).

This kind of physical environment, users and conditions constrained how Neptune was built and its structure. In particular we opted for a low energy consumption device, with a screen with high visibility in direct sun light, with a relatively large view angle, and such that it can easily be hand-held or attached to the body (of the tactician) or placed in the boat cockpit for the benefit of the helmsman/trimmers.

2.2 Context and Context Awareness

Context is an overloaded word, with many different interpretations. What follows is a brief review of the major aspects that characterize it so that we can then better understand how context is exploited in Neptune.

Dourish [Dou04] explores context from a technical and a social perspective, and links the former to positivist theories and the latter to phenomenological theories. He says: "positivist theories seek objective, independent descriptions of social phenomena, abstracting from the detail of particular occasions or settings, often in favor of broad statistical trends and idealized models". And phenomenological theories "... turn analytic attention away from the idea of a stable external world that is unproblematically recognised by all, and towards the idea of that the world, as we perceive it, is essentially a consensus of interpretation".

Nardi [Nar96] links context to distributed cognition, which is a "branch of cognitive science devoted to the study of: the representation of knowledge both inside the heads of individuals and in the world ...; the propagation of knowledge between different individuals and artifacts ...; and the transformations which external structures undergo when operated on by individuals and artifacts ... [FH91]".

Distributed cognition is concerned with representations inside the head of users and external artifacts that users exploit, and the transformations these structures undergo. It tries to understand coordination between individuals, the system and external artifacts. It focuses on a cognitive system composed of individuals and the artifacts they use; for example, Hutchins [Hut95] describes the activity of flying a plane, focusing on the cockpit system. Systems have goals; in the cockpit, for example, the goal is the successful completion of a flight. The cockpit, with its pilots and instruments forming a single cognitive system, 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.

Dey [Dey01] provides one of the most general and often used definition of context, namely "Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves".

Further, context is categorized as (A) context related to computing infrastructure (network, communication, technical resources), (B) user context (user profile, location, social situation), (C) physical context (lighting, noise, traffic condition, temperature), and (D) time context (time of day, week, month; season of the year).

Context awareness is a property of the behavior of a system. Dey [Dey01] 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 users task". In general, the purpose of context-aware features in a system is to minimize user effort enhancing usability and enabling a better user experience.

Alegre at al. [AAC16] identified four types of interaction of users with context-aware systems: (A) active execution, where the system acts autonomously depending on the context; (B) passive execution, where the system presents options or suggestions of actions and the user is involved in actions; (C) active configuration, where the system is able to learn from the user preferences in order to autonomously evolve his rules for future behavior; (D) passive configuration, where the user is involved in the manual personalization of his/her preferences, likes, and expectations of the system, after its implementation. The authors also identified three major tasks that developers of context-aware systems may find difficult: to enumerate the set of contextual states that may exist; to know what information could accurately determine a contextual state within that set; and to determine what appropriate action should be taken in a particular state.

2.3 Context awareness for Neptune

Context awareness in Neptune is driven by usability. In fact, it is seen as a way to cope with cognitive and ergonomic demands: the attention and concentration required by the primary tasks that users are involved in (boat handling, sails trimming, awareness of competitors and sea/wind conditions) leave little cognitive capability to be spent on thinking on how to interpret what a user interface displays, or on how to navigate through its screens; in addition, the physical situation leaves little ability to physically act on the user interface and monitor its feedback.

Context awareness features support the user in performing the primary tasks, by switching to relevant screens, by displaying relevant information, and by highlighting the most important and relevant information.

Most of the data that could be displayed on a sailing boat are contextual, according to Dey's definition: they show data of the boat (location, speed, orientation, course) or of 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. For an application like Neptune, to be used in a demanding environment, context awareness has to be based on reliable decisions regarding what is the current task that the user is involved in. And to the extent that such decisions are based on data, they require data that are appropriately filtered, otherwise data that change quickly are likely to lead to instability of the context aware feature.

In Neptune, task relevance is addressed in two ways. Firstly, for the helmsman and trimmers, who cannot easily reach the UI, Neptune runs on a device that is in the cockpit and the task that it has to address is either supporting the crew during start or monitoring the boat performance in terms of speed and course. Context awareness is achieved in part 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. Furthermore, during each of these phases (start or after start sailing), what is displayed and how it is displayed depends on the context. During start, if the boat position is beyond the starting line, a very salient display is used to alert all the crew of this potentially risky situation (a penalty may ensue). During sailing after the start, according to the point of sail (upwind, downwind, reaching, irons) and to the tack (starboard or port) different data is displayed based on polar curves of the boat to let the crew optimize speed and possibly course. These changes are driven by different combinations of point of sailing and tack, which are automatically sensed and determined.

Secondly, for the tactician, Neptune runs on a device that is hand-held or which is attached to the body. Because the task of the tactician is much more varied and because in general the tactician can use his/her hands to operate the device, we opted for letting the user decide what screen to display when. In fact, the start-assistance features are relevant to the tactician before the start, but during all the time the tactician might need to repeatedly check wind and current conditions, as well as to assess the starting situation.

In terms of the theories and notions related with "context" that were mentioned in § 2.2, we approached context awareness as a positivist theory, by aggressively restricting the range of human activities to be supported. In Neptune we opted for a very careful selection of what to display based on very carefully designed decision rules. Inaccurate displayed information or irrelevant screen navigation choices that are automatically made could completely void the value that the crew associates to the system. One consequence is that context awareness is determined on the basis of data of the boat and of the environment, rather than assumptions on what task the user is performing.

Neptune can be seen as an example of distributed cognition system. Some of its features can be thought as "cognitive protheses" aimed at enhancing the crew capabilities, and the boat cockpit with its instances of running Neptune form a single cognitive system. In this case it is the entire crew, rather than a single

individual, who is part of this system. In fact, context awareness is determined on the basis of relevance of data that is not specific to a single individual. In Neptune, context is related to physical and time data, only; no data related to the user or the computing infrastructure are used.

In terms of the classification of system behavior suggested in [AAC16] we adopted the stance that active execution is performed by the system, and no configuration is done, neither by the user nor automatically.

2.4 Development Approach

During the project, we observed that while there are many data that can be potentially relevant, it is difficult to understand which ones are critical in which situation. Interviewing crew members and asking opinions or doing user testing on low-fidelity prototypes [RC08] was not deemed to be sufficiently effective. Testing a live prototype with real data and possibly on board, by observing real users that use the prototype was thought to be a more effective strategy.

Therefore we opted for a Lean UX approach [Got13]. Lean UX design is "a mindset, culture, and a process that embraces Lean-Agile methods. It implements functionality in minimum viable increments and determines success by measuring results against a benefit hypothesis". In particular, emphasis was put on the expected outcomes of design choices we made on Neptune, on how users actually took advantage of the features and what are the benefits that derive from them.

Lean UX can be seen as a way to combine agile development (which rests upon an iterative and incremental process that yields high quality software increments - see the agile manifesto[3]), lean software development with user-centered design. User-centered design[4] is an approach to design and development of an interactive system based on an explicit understanding of users and their needs, and their involvement in all the development phases; the process is driven by usability evaluations, and it relies on iterative building of prototypes that are empirically evaluated to gather insights that let improve the solutions.

During the course of the project, which lasted for about 12 months, several iterations were executed, each resulting with a new prototype being built. Initially, prototypes were static sketches that were evaluated during informal user testing sessions (using the "Wizard of Oz" technique). Later on, executable versions of Neptune were developed that were first usability tested in the lab with pre-recorded real data, and then tested on board during sailing training sessions. Some of the prototypes were used also during actual races.

[3] Agile Manifesto: `https://www.agilealliance.org/agile101/the-agile-manifesto` and [Coh10].

[4] see `https://www.iso.org/standard/52075.html`.

3 User-level Requirements and Examples

Requirements for Neptune were formulated mostly as "user stories", in the commonly used template "As a ⟨user⟩ I need to ⟨activity⟩ so that ⟨benefit⟩", and in such a way to minimize their mutual dependencies and to emphasize the value that they can bring once they are deployed. Some user stories bring value to the sailing crew, other stories are valuable because that help understanding better the problem or the technologies that are used. Most of the user stories were also associated to acceptance criteria, that is conditions that can be tested and used to determine if the user story is completed.

Some of the focal user stories follow:

- As a crew member, I need to continuously read the data about wind and boat speed and direction so that I can react quickly to changes in the environment. Data need to be refreshed at 1Hz.
- 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 the conditions change. Target data depend on the actual point of sail, and include error of the True Wind Angle and error of the Velocity Made Good (error is defined as the $current - target$).
- As a trimmer, I can monitor the target data of the boat for the current point of sail so that I can decide that a correction is needed in how sails are trimmed, rather than in how the boat is steered.

See Figure 2. The screen on the left can be used by the helmsman to correct steering by understanding that a wider angle to the wind needs to be held in order to reduce the boat speed error. The arrows in the foreground are visual cues that reduce the cognitive load of the helmsman, telling him or her directly what action to take, i.e. steer left or right. If the angle error is below 4 degrees then no arrows are shown as it can be impossible to reduce such an error in some sea and wind conditions.

The same data can be used by the sail trimmer to understand that the error in boat speed can be reduced also by better handling of sails. If the helmsman keeps the error of the wind angle under control, then it is the job of the sail trimmers to improve the boat performance.

The dynamics when sailing downwind is similar (central screenshot): the only change is related to the condition triggering the foreground arrows. The situation changes with wind-abeam, where the notion of target speed changes. Rather than being defined as the maximum speed at which the boat can reach a buoy that is perfectly upwind or downwind, with a cross wind the target speed is defined as the maximum speed along the chosen course. As shown in Figure 2 (right), the target data now displayed include only the error in boat speed and wind conditions. No changes in steering are needed to reduce such an error, whereas better sail trimming might be needed.

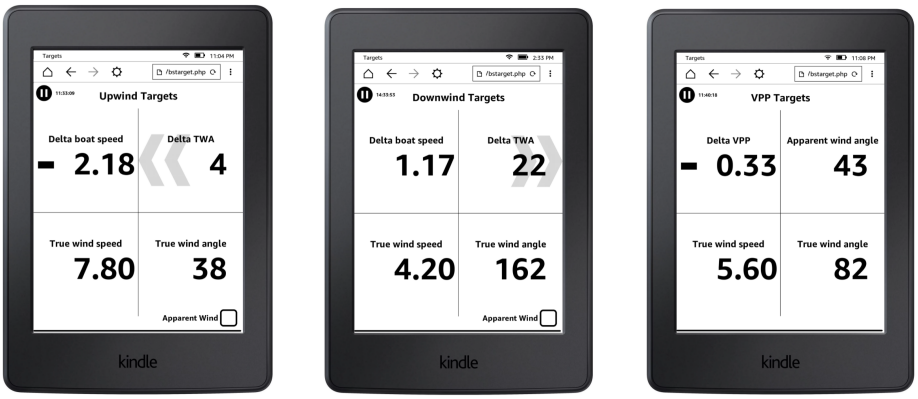

Figure 2: (Left) Screenshot of the upwind target screen, that shows the error of the boat speed (too slow by 2.18 kt), the error of the True Wind Angle (4 Deg too close to the wind), the actual speed and angle of the True Wind. The foreground arrows suggest the helmsman to steer to the left, away from the wind. (Center) Same data, but when sailing downwind. (Right) Screenshot of the wind-abeam target screen, that shows the error of the boat speed (too slow by 0.33 kt) along the current course, not in the direction of the wind as before.

In order to reduce human interaction, Neptune is capable of detecting changes in the points of sail and switching mode as needed.

Other user stories are concerned with the starting phase:

- As a tactician, I can get fixes for both ends of the starting line in order to plan the start.
- As a tactician, I can get more than one fix for each point in order to average out small fixing errors.
- As a tactician, I can get a bearing of the line direction, in case I cannot manage to have the fixes for the endpoints.
- As a tactician, I need to visualize which is the favored starting end if the line is not orthogonal to the wind. I need also to estimate what is the quantitative gain.
- As a helmsman or as a tactician, I need to estimate how far is the starting line, as distance and as sailing time, so that I can decide if I need to change boat speed and course.
- As a crew member, I need to see if the boat is over the line so that I can move away from there or check that all the data are correct.
- As a crew member, I can replay the start data during a debriefing session to learn what we should have done during the start.

Figure 3: (Left) Screenshot of start assistance screen, that shows that there are 3 fixes for the left end of the starting line, two for the right end, 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 24.7m. The current shortest distance to the starting line is 40m, and in 40s by sailing at the target speed the boat will cross the starting line, i.e. 16s ahead of time. (Right) Similar data as before, but here the boat has already crossed the starting line by 20m just 7s before the start.

Figure 3 shows the screen to be used at starts, with data concerning the starting line (position of end points, length, direction), the boat (distance to each of the end points, time needed to reach each end point, 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).

The screenshot on the right shows that the bottom part is highlighted and blinks to emphasize that the boat is on course side too early, and this may lead to penalties.

Neptune automatically changes mode of the start assistance screen depending on boat position and timing. It also automatically switches from the start assistance to the target screen after 15 seconds since the boat crossed the start time after the start.

Other user stories capture needs that are specific to the tactician:

- As a tactician, I need to monitor trends in speed and direction of the wind in the last several hours in order to predict if the wind is oscillating or rotating, and increasing or decreasing in speed. I need new data every minute.
- As a tactician, I need to know how long it will take since the next update of the trend, in order to avoid staring continuously at the screen.
- As a tactician, I can rely on the system to continue collecting data about the wind even if I switch to other screens.

- As a tactician, I scroll back to previous data and jump directly to the most recent data, without spending time operating on the user interface.
- As a crew member, I can replay today's wind changes during a debriefing session to learn what we should have done during the race.
- As a tactician, I need to estimate the direction and strength of marine current in order to decide when to tack to reach a buoy or cross a competitor.
- As a tactician, I need to estimate the direction and strength of marine current in order to plan a better start and avoid sailing beyond the laylines.

Wind changes and marine currents are major factors considered by a sail race strategy. Figure 4 shows a screenshot of the wind-graph screen that shows how wind data is collected (and averaged) every minute and used to display in a visually salient way the changes in wind direction. Data is continuously drawn on the grid, the viewport is automatically scrolled horizontally to include the most recent data, the user can scroll the data up and down, and can zoom-in/out the data in order to get a general overview.

Figure 5 shows the screen that presents estimates of current and leeway. Such a data is computed by comparing the magnetic heading and boat speed over water with the last set of GPS fixes, and therefore estimating the cumulative effect of marine currents and the boat leeway. At the moment, Neptune does not use inertial measurement units, which could be exploited to separate the leeway effect from the current. This requires accurate models of leeway for a specific boat, under different points of sailing, different speeds, different wind and sea conditions.

3.1 Testing

Testing a system that is normally used in races is challenging, because the normal usage situation is not suitable for testers (they are not part of the crew, the crew has no time to participate in testing, malfunctions of the tools would jeopardize trust). And yet, especially for usability aspects, testing the UI in such situations is the most effective way to improve it.

Another challenge is that sometimes testing a feature requires sensors producing certain kinds of data that are difficult to replicate in the lab, from scratch. This is the case of computations of current and leeway information. Sailing conditions that differ because of varying wind speed and direction, boat speed over water, waves, boat magnetic heading, beat heeling angle, boat pitch motion affect measurements and calculations. Therefore, this kind of testing can only be performed with accuracy during navigation.

For this reason we opted for a multi-strategy testing approach.

- The programmer developing a feature would implement and run unit tests.

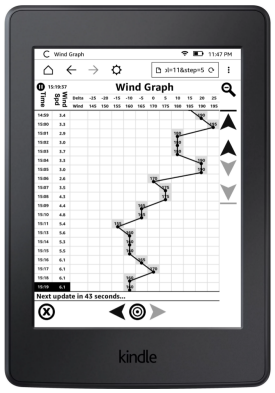

Figure 4: Screenshot of the wind graph screen that shows the trend in wind changes, sampled every minute. The two left-most columns show the time and wind speed, and the corresponding cells of the table show the wind angle. In the last 11 minutes the wind has rotated counter-clockwise by 35 degrees and increased from 3.3 to 6.1kt. The next update will occur in 43 seconds.

Figure 5: Screenshot of the current and leeway screen, that shows the speed and direction of the combined action of the marine current and lateral push of the wind. At the moment it is estimated to be 0.24kt 15 degrees to the right.

- When a user story would complete then its acceptance criteria are used to perform exploratory testing using canned data, configured specifically for that user story. For example, acceptance criteria for the user story that deals with estimating which is the favorite end of the starting line include:
  - If fixes for both ends of the line are available, then the gain (in meters) is updated every second in the UI, close to the favored end.
  - If only one end was fixed, or none, but a bearing is available, then the gain (in percentage) is updated every second in the UI, close to the favored end.
- Exploratory testing is carried out also using data that was recorded during prior training sessions or races. This is done within the lab. All the context aware features of the system were tested in this way.
- Some usability tests with crew members are carried out in the lab using pre-recorded data.
- Other functional tests are carried out during training sessions. This was the case especially for the current and leeway screen, subject to specific weather and sea conditions.
- Finally, other usability tests were run during training sessions. These were invaluable to collect feedback from crew members. Such a feedback ranged from issues with icons or numbers that were difficult to understand or use, to new ideas about features to include or new calculations to perform, to understanding what kinds of context awareness could be implemented to achieve what effect. For example, switching from upwind/downwind targets to VPP targets was the result of a brief brainstorming session held during a training session.
- During races only minimal feedback was collected, based on written notes that some crew members (2 of the authors) brought back on shore at the end of the race and later discussed with developers. As mentioned in the Introduction, the application was used during club sailing races and more important championships (namely, the 2017 Italian Offshore Sail Race and the 2017 ORC Worlds Trieste), by different crews.

During test sessions with actual crew members, several opportunities for improvement were brought to light. For example, it was discovered that the level of flickering of initial implementations of the current and leeway screens were intolerable, and reduced usability of the screen. Thus a refactoring took place, assuming a constant heading of the boat and the vector of the polar angle of the current and leeway slowly rotating as needed.

Other examples of collected feedback are related to the wind trend screen; a tactician suggested that there should be two modes of operation, one used for on-board use and one for debriefings. The UI for racing should include fewer

buttons and features. Notice that this would be another context-aware feature, dependent on the user task.

During testing of the starting screen, errors in fixes were detected of the order of 1-3 meters. Initially they were thought to be due to the use of planar spatial coordinates instead of coordinates related to the earth surface. Later on it was discovered that these errors were due to latency of the GPS sensors and the 1Hz refresh rate of the UI. Other suggestions included displaying the time needed to reach the starting line, not just its distance.

Some of these suggestions will be included in the next version of Neptune.

4 System Architecture: the Big Picture

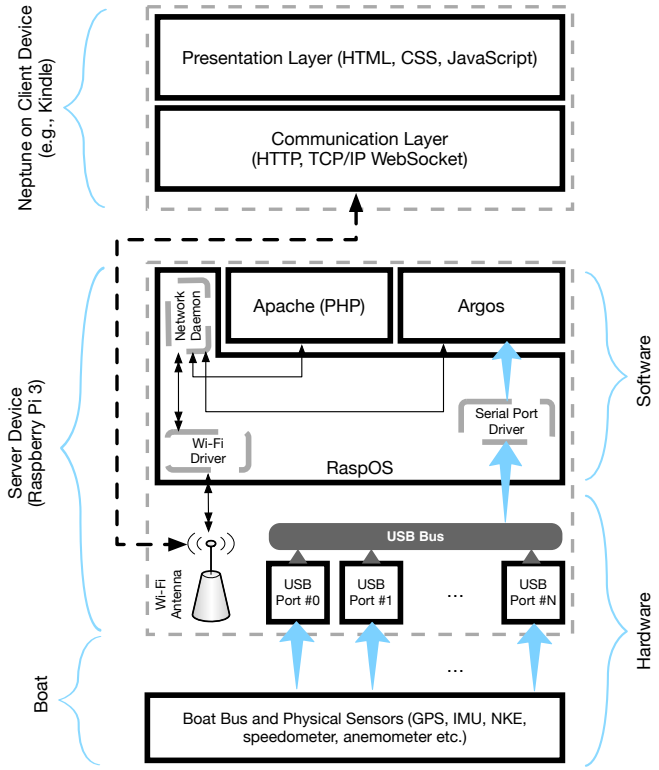

Figure 6: An overall look at the system architecture.

If we look at the hardware infrastructure of our system (see Fig. 6), at the lowest level we have the sensors, the communication bus, and some electrical units which are rather common in sailing boats. Then, the core element of Oceanus is a Raspberry Pi 3 (model B+), which allows us to:

1. harvest raw data from sensors (connected through USB ports);
2. compute meaningful and useful information from raw data;
3. use the computed information to coordinate context-awareness through the generation and handling of events;
4. provide a WiFi local area network (LAN);
5. publish the computed information and the related events through a socket service and a web application in the LAN.

Finally, e-ink devices (like, e.g., Amazon Kindle) are used by the crew to access the relevant information during the normal on-board activity.

From the software point of view, the whole system has a multi-tiered architecture, in order to make it robust, easy to maintain, and extensible. Immediately above the bus and the sensors there is the server component of Oceanus. Its first software layer is constituted by the operating system of the Raspberry Pi 3 (RaspOS). It provides the drivers to manage USB ports and the WiFi antenna, and the network daemon to manage connections with clients and to establish the wireless LAN. Then, we have the Argos process which (i) reads data in real-time, and stores them in log files; (ii) computes secondary data from raw data and keeps them in a suitable structure in main memory; (iii) applies contex-aware rules to computed data to infer new events; (iv) sends such information to subscribers (clients) via a publish/subscribe service. At this level, another important process is the local Apache web server which provides the dynamic pages of the web interface to client devices.

Finally, the client layer consists of the Neptune web application. It is organized in two modules, namely, (i) the communication layer (i.e., responsible for the communication with the Argos and Apache processes) and (ii) the presentation layer (i.e., the graphical UI driven by the events published through the Argos service).

5 Technological Issues and Limits

In small ($\leq$ 40 ft) leisure yachts, the management of energy on board is always a critical issue and constrains the use of different electrical and electronic devices especially during a long ($\geq$ 200 M) sail navigation. A typical installation relies on three 12 V lead batteries of about 80 Ah each; one is exclusively dedicated to the engine start and one to the auto pilot mechanical actuator and its governing computer. Thus, only one battery supplies services, like navigation lights, cabin lights, electronic instrumentation and bus, water pumps, refrigerator.

Therefore it is crucial to minimize energy consumption and throughout the whole project development, many hardware related choices were affected by such limitations of the electric power supply available on the boat. A typical example is the selection of the Raspberry Pi 3 as the low-power central unit of

our architecture. In case of need an external 5 V power bank serves as a backup power supply for the Raspberry.

Moreover, in order to exploit the hardware as much as possible without introducing too many latencies, we favored runtime speed and memory optimization against development ease, writing all the code of Argos (see Section 7) in C, avoiding interpreted and object-oriented languages like Python and Java.

Since the communication between the Raspberry Pi 3 and other visualization devices occurs via WiFi (typically on channel 6 at 2.4 GHz), we had to carefully evaluate the electromagnetic compatibility between such computer and the existing electronic devices, particularly the electronic compass. Moreover, also local external radiated disturbances must be monitored in order to avoid loss of communication. As an example, in the gulf of Trieste, where we performed the experiments, a radiated disturbance exactly at the channel 6 of the WiFi is present; the solution was to switch to a different WiFi channel the Raspberry Pi 3.

6 Sensors and Buses

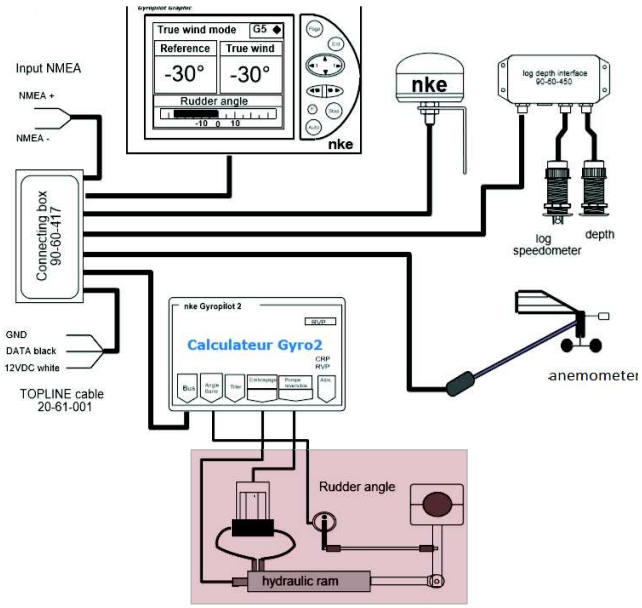

Figure 7: Nke TOPLINE bus used to interconnect on board sensors from a standard factory installation.

Most the sailing boats are equipped by the manufacturer with a standard instrumentation pack from a specific brand (e. g. B&G, Garmin, Nke, Rayma-

rine are instances of widely used commercial brands); according to the brand, data from different sensors of the same manufacturer dialogue each other via a *proprietary* bus. As an example, Nke interconnects its sensors, actuators, multifunction displays and the processing unit, named "Gyropilot calculator", by means of the so called TOPLINE bus, Fig. 7.

The graphic display is the user interface of this instrumentation; it allows the user to control the autopilot and adjust the settings of the sensors. The Gyropilot calculator is the controller of the autopilot; together with the hydraulic unit and the electric unit, it controls and actuates the boat's rudder on the base of a prescribed control variable such as course or wind direction.

Depending on the level of equipment a typical installation consists of: a fluxgate compass used to specify the Magnetic Heading (MH) of the longitudinal boat axis with respect to the magnetic North, a speedometer sensing the Speed Over Water (SOW) along the longitudinal boat axis based on a paddle-wheel, a rudder angle sensor, a gyro sensor (integrated in the calculator), a wind station on the top of the mast, sensing amplitude of the wind, Apparent Wind Speed (AWS), and its direction with respect to the longitudinal boat axis, Apparent Wind Angle, (AWA). Moreover, depth below the keel, air and water temperature, voltage and power consumption, are measurements available on the bus. We will denote such quantities as *primary* quantities since they are directly measured by a dedicated sensor.

From a subset of primary quantities, a number of *secondary* quantities are usually computed and are available on the bus together with the primary ones. As an example, from the triple {SOW, AWS, AWA} the so called True Wind Speed (TWS) sensed by an observer fixed with respect to the water surface is computed. Similarly for the corresponding True Wind Angle (TWA) with respect to the longitudinal boat axis; both are secondary quantities. In addition, from the pair {MH, TWA}, the Magnetic Wind Direction (MWD) is computed with respect to the magnetic North. In the case of Nke installations, the Gyropilot calculator performs such computations as a black box, making the quantities {TWS, TWA, MWD} available on the TOPLINE bus.

Such installation represents a *closed* hardware and software infrastructure; thus the input/output interaction with other devices or sensors is provided by a dedicated input/output interface which allows the bidirectional conversion of the data present on proprietary bus to a public data bus. The NMEA 0183 open standard is a specification for communication between items of marine and GPS equipments. It is defined and controlled by the National Marine Electronics Association [nme18]. The 0183 standard uses a simple serial protocol to transmit a sentence to one or more listening units. A NMEA frame uses every ASCII character. As an example, the Nke interface dialogues through the NMEA 0183 V2.30 standard, yielding sentences like

Listing 1: Example NMEA sentence

```
$IIVWR,053.,L,04.5,N,02.3,M,008.3,K*6A,
```

where $\$IIVWR$ is the label specifying computed true wind data, TWA=53 Deg to the left side of the longitudinal axis (label L), TWS=2.3 knots (label N) or

equivalently TWS=8.3 m/s (label M) according to the unit. The format of frames is: 4800 baud rate, 8 bits with bit 7 at 0 / 1 start bit and 1 stop bit, with checksum. The frames are continuously transmitted in an asynchronous way and every frame is transmitted during each cycle.

The NMEA interface allows the conversion of the data present on the proprietary bus onto the NMEA data bus, so as to allow the interfacing between the proprietary network of devices and external devices from a different manufacturer.

A typical external device, interfaced with the closed hardware and software infrastructure, is a GPS; it provides as output in NMEA 0183 format at least the latitude, longitude, Speed Over Ground (SOG), Course Over Ground (COG), time and date; in the case of a cartographic GPS, way points setting and the corresponding cross track distance and course are sent in addition.

6.1 Raspberry Pi 3

The rationale of the Oceanus project is to set up an independent acquisition unit, with in house developed software, capable to record and process primary and secondary quantities generated not only from the existent closed infrastructure but also from a number of different independent devices.

To this purpose, we used an RS232 serial port to interface the closed infrastructure to a Raspberry Pi 3 computer. In addition, we considered an Inertial Motion Unit (IMU) and an Automatic Identification System (AIS) unit.

The IMU is equipped with a 3-axis accelerometer, magnetometer (compass) and gyroscope and works as a USB inertial measurement unit. It can measure 9 degrees of freedom and computes quaternions, linear acceleration, gravity vector as well as independent heading, roll and pitch angles. It is a complete attitude and heading reference system.

The class B AIS transponder allows exchange of AIS messages with other AIS equipped vessels within VHF range, the essential way to improve collision avoidance on the water; it is designed to provide NMEA 0183 data via a waterproof USB port enabling a simple integration with the Raspberry Pi 3; for safety reason it is equipped with a dedicated GPS unit.

6.1.1 Timing

Since we deliberately avoid the use of local clocks, the absolute date and time is provided by a further GPS unit; it is connected to the Raspberry Pi 3 as an external device, yielding, every second, a NMEA sentence at 9600 baud rate.

Listing 2: Example NMEA sentence

```
$GNRMC,163221.000,A,4604.8468,N,01316.0955,E
    ,4.73,233.38,281017,,,A*7C,
```

Listing 2 shows an example, where 163221.000 encodes the UTC time instant in the format HHMMSS, latitude 4604.8468 North and longitude 01316.0955 East

in sexagesimal degrees, SOG 4.73 in knots and COG 233.38 in Deg, date 281017 in the format DDMMYY.

In between a pair of successive GPS sentences like Listing 2 a number of NMEA sentences are acquired and processed; all data encoded in such sentences are attributed to the time instant encoded in the preceding GPS sentence; thus a time jitter up to 1 sec affects such data.

7 Argos: Harvesting Data in Real Time

The first software layer of our system above the OS is named Argos[5] and it is responsible for harvesting the raw data produced by sensors and fed to the USB ports of the Raspberry Pi 3. Indeed, as we saw in § 6, all the sensors ultimately provide a stream of NMEA 0183 sentences over a serial communication interface. The only differences are in the baud rate of the different kinds of devices. However, as we will see in detail in § 7.1, a multithreaded solution suffices to read the data concurrently, in real-time, and in a lossless way (see Figure 8).

7.1 Reading Data Safely and Efficiently

In order to cope with the different features (e.g. speed, parity and stop bits etc.) of the devices attached to the USB ports, the main execution thread can read the configuration of the devices either from command line options or from a configuration file in YAML[6] format. Then, it spawns a dedicated reading thread for each detected serial device (in Fig. 8 they are represented by *Reading Thread #0*, *Reading Thread #1*, ...): to keep with our metaphor about the giant Argos, these threads are called *Argos' Eyes*, since they represent the capability of the system to acquire data from the environment.

Raw data consist of lines of text in NMEA 0183 format; hence, each Eye stores such lines in a queue (for logging purposes as we will see later on in this section) and it uses a very simple parser to extract the single data fields contained in those lines. Such fields are manipulated by a computation library which provides several functions allowing the system to synthesize more structured and meaningful information for the crew (see § 7.2). This information is stored (and continuously updated as sensors provide new raw data) in a suitable structure in main memory shared among all threads. Since each Eye can compute new information and update the shared structure, the latter must be protected from simultaneous accesses, otherwise it might become corrupted. For such purpose we use a mutex, paying attention to avoid deadlocks, of course.

As we anticipated before in this section, the NMEA 0183 lines are logged in the filesystem of the Raspberry Pi 3. In order to avoid an excessive load for the SD card, we do not keep open the log files during all the lifetime of the

[5] The name refers to Argos (a.k.a. Argus Panoptes), i.e., the *all-seeing* primordial giant of the Greek mythology.

[6] YAML stands for *YAML Ain't a Markup Language* and it is essentially a human readable data serialization language. See the official website for further details: `http://yaml.org/`.

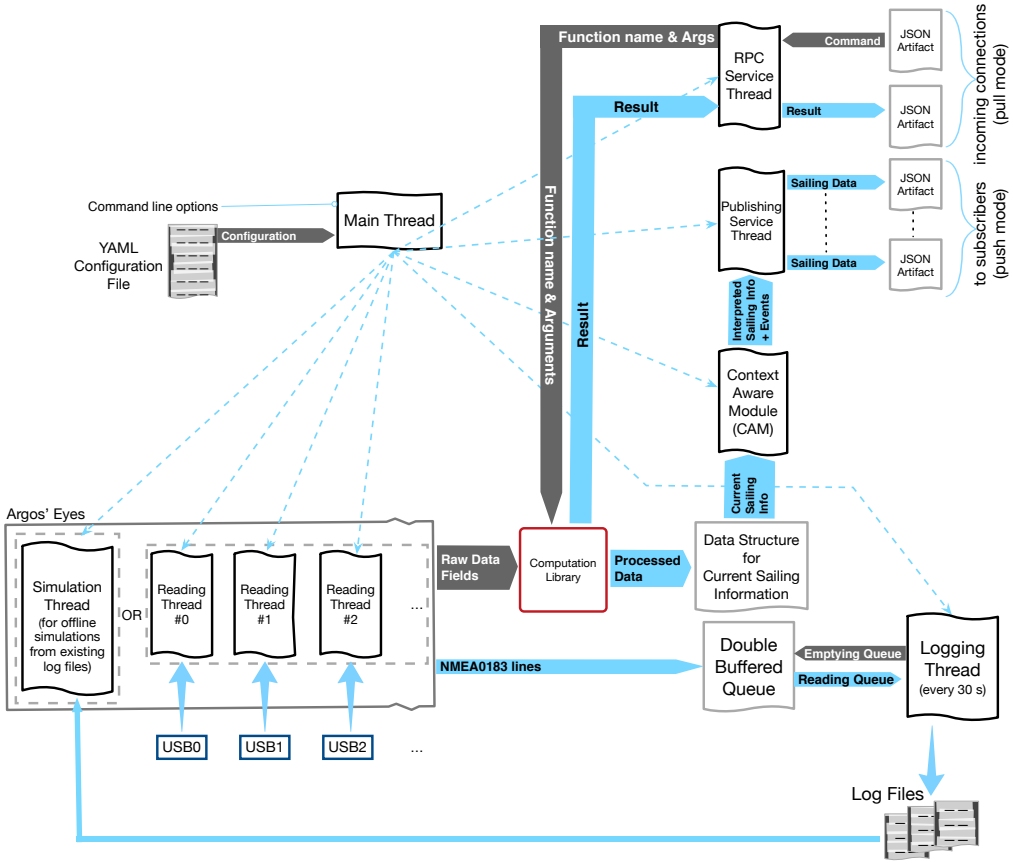

Figure 8: Argos architecture.

Argos process. Instead, we schedule a suitable logging thread (see Fig. 8) to intervene every 30 seconds (but the user can change this time interval modifying the configuration file) saving the current queue to the filesystem and emptying it for subsequent use by the Eye threads. Of course, this is another crucial process since saving the queue to disk and emptying it should neither interfere nor slow down the reading threads. Hence, we opted for a double buffered queue, where: (i) one buffer contains the current queue of NMEA 0183 lines and the second buffer stays empty; (ii) when the logging thread must save it to disk, the buffers are switched: the one full of lines is given to the logging thread, while the empty one is assigned to the reading threads. Thus, the logging and reading threads are not affected by the relative execution speeds. Of course, the act of switching the buffers must be atomic: such constraint is granted by the use of another mutex.

We conclude this section illustrating another interesting feature of Argos. Indeed, in the diagram of Fig. 8 we see that, instead of spawning the classic

reading threads, the system can instantiate a "Simulation Eye", namely, a reading thread acquiring NMEA 0183 data lines from existing log files instead of USB ports. By doing so, we can use Argos to review and analyze sailing data off-line (i.e., after a race), possibly interfacing our software with numerical data analysis tools or with machine learning algorithms.

7.2 Computing Data

As we saw in § 7.1, the acquisition system on the Raspberry Pi 3 consists of a C program capable to access in real time the serial ports connected to it, recording on dedicated log files a user-selected subset of NMEA sentences to be used for the off-line a posteriori analysis. Moreover, the selected NMEA sentences are processed in real time on the basis of the initial label and they are tokenized into fields.

Thanks to the developed acquisition system, primary quantities are extracted and conditioned as functions of time; moreover, they are combined in order to produce new secondary quantities, independent from those already computed by the closed infrastructure.

7.2.1 True wind computation

From the triple { (V_l, V_t), AWS, AWA} we generate the new pair {TWS_V, TWA_V}, where the pair (V_l, V_t) represents the longitudinal and transversal components, respectively, of SOW[7] or SOG according to the case; the advantage is that true wind data can be computed also in the case SOW is not available due to a malfunctioning of the sensor in contact with water. Of course a different notion of true wind is computed in this case (true wind with respect to the ground), which cannot be used for target speeds of the boat. From elementary geometry, we obtain

$$u = \text{AWS}\cos(\text{AWA}\frac{\pi}{180}) - V_l,\ v = \text{AWS}\sin(\text{AWA}\frac{\pi}{180}) - V_t \tag{1}$$

$$\text{TWS}_V = \sqrt{u^2 + v^2},\ \text{TWA}_V = \frac{180}{\pi}\text{atan2}(v, u). \tag{2}$$

By adding to the TWA signal the magnetic heading MH, with modulo 360, the Magnetic Wind Direction (MWD) which refers to the magnetic North con be deduced.

7.2.2 Geodetics computation

We considered standard approximate geodetics formulas on the local tangent plane at the actual latitude, [top18, Kar13]; calculations on the basis of a spherical earth (ignoring ellipsoidal effects) are accurate enough for our purposes.

[7] In the case of SOW, $V_t = 0$ holds, because the boat speed sensor is aligned with the longitudinal axis of the boat.

In order to plot the trace on the local tangent plane from actual values of latitude LAT and longitude LON in sexadecimal degrees, we performed the Mercatore conformal projection

$$x = R\,LON\frac{\pi}{180},\ y = R\log(\tan\pi(0.25+\frac{LAT}{360})), \tag{3}$$

where x, y are the resulting Cartesian coordinates in meters referred to an origin located at the intersection between the Greenwich meridian and the equator; R is the local earth radius. As an example, in Fig. 9 the trajectory of a typical jibe is drawn with green circles on a local plane; the TWS and TWA are also represented with arrows. Moreover, a label shows in the selected point (the small square) the time instant in seconds, COG and SOG, SOW, MH, AWA, TWS; finally the roll (ROL) and pitch (PIT) signals from IMU are also shown.

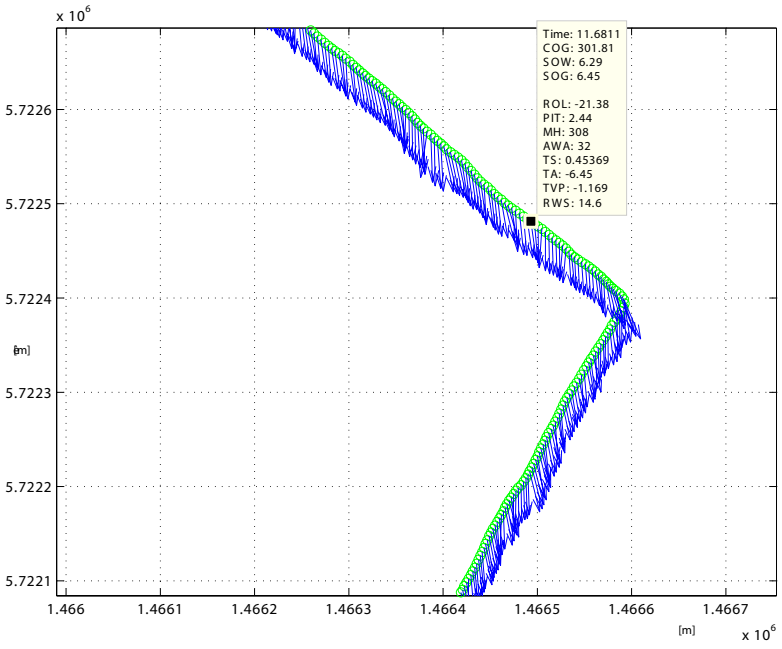

Figure 9: A trajectory of a boat during a jibe on the local plane (the boat is moving downward, with a Northerly wind); North is up. Green circles represent boat positions, and blue arrows represent wind direction.

7.2.3 Target data

On the basis of the actual geometry of the sails, of the boat and of a model of the geometry of the hull, a software program known as Velocity Prediction Program

(VPP) of the Offshore Racing Congress [orc18], computes the best velocity and true wind angle (they are often referred to as Target Speed and Target Angle respectively) the boat should have at a specified true wind speed whose value in knots belongs to the canonical set {6, 8, 10, 12, 14, 16, 20}. As an example for the Uniud Sailing Lab boat, the simulated best true wind angles in Deg during upwind navigation are {42.4, 40.9, 40.0, 38.4, 38.0, 37.9, 37.6}. These numbers represent the polar curves of the boat. Argos performs in real time the comparison between the actual SOW and TWA with the simulated best velocity and best true wind angle during upwind or downwind navigation, showing the discrepancy from interpolated target values. In Fig. 9 such a discrepancy is denoted with TA for the angle and with TS for the speed; a negative value indicates an underperforming boat.

7.3 Publishing Interpreted Data and Events

The data structure representing the current sailing information in Fig. 8 is continuously kept up to date by the system through the use of the *computation library*. Then, such data structure is used by the Context-Aware Module (CAM, see Fig. 8) to *interpret* the current situation. For instance, from the data values of TWA=38, SOW=5.6, and MH=225, CAM will infer the interpretations "port-tack" and "close-hauled". Those interpreted data are then routed to subscribers (in our case the Neptune web application) by the *Publishing Service Thread* in JSON format (see Fig. 10 for an example related to this case). Furthermore, CAM will generate *events* like "tacking" which will be added to the interpreted data and sent by the Publishing Service Thread to the registered clients (again, Neptune in our case). Correspondingly, if Neptune is showing the targets screen and it knows to be in the state of "port-tack", then, at the moment of receiving the event of "tacking", it will change the screen accordingly to begin monitoring the maneuver.

```
{
  "twa" : 38,
  "sog" : 5.6,
  "mh" : 225,
  ...
  "mode" : "live_race",
  "intepretations" : ["port-tack", "close-hauled"],
  "events" : ["tacking"]
}
```

Figure 10: A sample of sailing information published in JSON format: dots (...) represent omitted (non-relevant) data.

In order to ensure that the system is also able to respond to *pull requests*, we implemented a simple *Remote Procedure Call (RPC)* mechanism. Indeed,

thanks to the latter, computation tasks can also be requested to Argos from external software through a JSON artifact sent to a dedicated *RPC Command Thread* (look at the upper right part of Fig. 8). Such artifact plays the role of a remote procedure call, specifying both the name of the function to invoke and the related arguments: once it has been received, a dedicated thread will execute the function call and send back the result of the computation to the remote caller, always in JSON format.

As an example, we added the possibility for ask Argos to compute the maximum speed towards each of two possible destinations (conventionally called `s`, the starboard end of the starting line, and `p`, the port end of the line), given as input arguments the current true wind speed (`tws`) and the true wind angles with respect to the given destinations (resp. `twas` and `twap`) in JSON format as follows

```
{ "cmd": "polar", "tws": "9.7", "twap": "124.0",
  "twas": "-72.0" }
```

The results (resp. `speedp` and `speeds`) are computed in real time and returned to the caller in JSON format:

```
{ "speedp": "7.273267", "speeds": "7.167200"}
```

This kind of requests is very useful before the start of a race, in order to help the crew to avoid crossing the starting line (between the buoy and the committee boat) before the official start time. Requests of this kind can be performed in several ways, e.g., through a PHP dynamic page, which is the case of Neptune.

8 Related Work

In this section we highlight a selection of commercial software solutions currently available on the market for supporting racing crews. In the last decade, smartphones and tablets have become accessible to a large spectrum of users, so this kind of applications is no longer intended just to a narrow group of professionals, but can be adopted by amateurs as well. Here we analyze the following three applications, by focusing on their main functionalities, describing their pros and cons, and key differences with Neptune:

1. iRegatta (by Zifigo)
2. Esa Regatta (by ASTRA Yacht)
3. SailRacer (by UAB SailRacer)

8.1 iRegatta

Available on iOS and Android platforms and developed by the danish company Zifigo, iRegatta has been on the market since 2009. It is an application intended for both racing and cruise navigation and it comes with a rich set of different features.

The customizable Race view allows the crew to monitor up to a maximum of four different readouts at the same time. Moreover, it provides the crew with a horizontal performance bar, based on a percentage value, which is obtained by comparing the current boat speed to the polar predicted one. This value can be actually higher than 100%, when the boat beats the predicted speed. The screen shows, at the bottom, speed and VMG history graphs.

The Course view can be accessed by sliding down from the Navigation View. It provides the tactician with a sketch of a course with three buoys and a starting/finishing line. A buoy location can be saved by approaching the waypoint and tapping the related button.

The Layline view provides information about the current point of sail, true wind direction and target true wind angle, along with a graphical view of the boat and the current waypoint laylines. The Start view allows the tactician to ping the starting buoy and vessel positions, retrieve distance/time to line information in real time, and check the count down to race start. Moreover, it provides him with a graphical view of the boat and its dead zone, along with the starting line and laylines. The burn-or-gain bar is visible on the left, a component that indicates whether the boat needs to speed up or slow down, in order to reach the starting line on time. Once the boat crosses the starting line, the Race view is automatically activated.

Several extra views depend on availability of specific NMEA sensors, such as sonar, radar or thermometer. Other features allow users to record navigation data for further external analysis, view polar diagrams and import/export polar data through CSV files.

8.2 Esa Regatta

Developed by the Italian company ASTRA Yacht Srl, Esa Regatta is currently available on iOS platform only.

As far as the Navigation view is concerned, this solution shares several common aspects with other apps, such as the ability to view a list of up to four different quantities, which can be customized by tapping the related buttons in the two side columns. Moreover, it also provides a performance bar, as previously seen on iRegatta.

An interesting feature of this app is its ability to automatically update target data and polars through patented algorithms. Smart polar mode can be enabled by pressing the ESA polar button on the bottom. Moreover, ESA polar data can be exported and analyzed by Esa Data & Polar Analyzer, a professional desktop application developed by the same company.

The Start screen does not provide a vector view of the boat and the starting line, as seen in iRegatta. Instead, the UI shows a static graphical representation of the line, along with an arrow indicating how wind direction relates to it. Other numeric values describe wind status, distances and times to line and its ends. Favored end and related gain are indicated bottom right. A count down indicator is visible on the left.

8.3 SailRacer

SailRacer has entered the market in 2013 and is currently available on both iOS and Android platforms. It offers most of the main features previously mentioned about the other apps, plus some extra add-on, such as the option to view the UI on a dedicated remote device.

The Navigation screen allows the helmsman to follow the route to one or more waypoints by monitoring a compass-like widget, which highlights the difference between current TWA and its target value. A polar curve diagram, related to current wind speed, is integrated in the widget itself. A percentage indicates the boat performance, as described in other solutions. In the right column, numeric values refer to TWA, heading and distance/time to layline and current waypoint. Next to TWA angle, arrow-shaped indicators help the user to decide whether to luff or bear-away.

The Start screen partially resembles the Navigation screen, but adds some important components, such as the count down and time-to-burn indicators, as previously seen in iRegatta. The compass widget shows the distance between the boat, the starting line and its ends, which can be marked by tapping the related side icons. The numeric values in the right column include VMG.

Sailracer app provides the user with the ability to view the UI on an external device. The company currently offers a modified version of the inkBOOK Classic 2, an Android-based e-book reader with an E Ink display built-in. The device comes with a waterproof protection and integrates a high-capacity battery, but has no backlight and cannot be used at night or in poor lighting condition.

8.4 Neptune and third-party applications: a comparison

The three applications just illustrated are examples of standalone applications, that must be manually installed and updated on each single device. They all support WiFi or Bluetooth connections to NMEA-compatible data sources. If no NMEA device is available on board, the user can still retrieve boat position, speed and direction information through the sensors available on the mobile device itself.

Oceanus, on the other hand, is based on a client-server architecture, which allows users to access the latest version of the frontend at any time through local WiFi network. Users can use Neptune on a Kindle device or any other mobile device. As for the commercial apps mentioned, extra NMEA sources can be easily added and accessed to Oceanus as well. Moreover, third-party apps are designed for smartphone and tablet displays, which generally suffer for low-readability under strong sunlight. Neptune, instead, is optimized for Kindle Paperwhite and takes advantage of its E Ink display technology and backlight, which makes it accessible in any lighting condition. Furthermore, the significant battery life is also a benefit that should not be ignored. Currently, SailRacer is one of the few commercial apps that supports a dedicated device with such peculiarities.

As far as the UI is concerned, Neptune has several aspects in common with

the three commercial apps. It shares some specific context-aware feature, such as luff and bear-away indicators, as seen in SailRacer. Moreover, it detects the moment when the boat crosses the starting line and provides a visual feedback about it and switches to another screen. A similar behavior was also noticed in iRegatta, where this specific event triggers the Race view, instead.

On the other hand, the Targets screen in Neptune is able to switch between the appropriate readout sets, depending on the current point of sail: it switches between upwind, downwind and wind abeam presenting and highlighting different data. Neptune also provides a special screen for current and leeway monitoring. Such feature was not found in the commercial apps, although some of them do integrate COG and SOG based current calculation into specific screens, such as iRegatta in its Race map view. Moreover, the Wind screen in Neptune allows better readability by drawing the graph overlay straight on top of true wind direction numeric data table, while other solutions generally place the numbers on the axes and require from the tactician an extra effort to interpret the graph. Furthermore, Neptune's Start screen is more flexible, compared to other applications. First of all, it allows the tactician to take multiple fixes for both the ends of the starting line, to obtain the coordinates of the average position and manage the two lists of fixes. Besides, it supports partial detection of the line, by taking the line bearing and fixes for a single end, or just the line bearing.

8.5 Oceanus vs. Commercial Sailing Instruments Products

For the sake of completeness, in this section we will consider also "complete" systems, i.e., solutions providing both the low-level communication layers with the boat hardware (i.e., the role played by Argos in our case) and the user graphical interface (i.e., our Neptune web application). Such solutions are commercial products which usually operate and are integrated with ad-hoc hardware and can be expensive.

Worth noting is OS5, a software solution offered by Ockam (`http://www.ockam.com/`)[8] to racing crews. This highly sophisticated solution includes advanced screens for many different aspects of a sail race. For example, a particular screen supports crews with the Wally technique, an advanced way to use the polar curves of the boat; another screen provides the BET Diagram, supporting cost/benefit analysis of tacking vs Wallying. Several features are made available for supporting the starting phase of a race: the DogLeg Start technique supports the crew by suggesting how to cross the line with a high boat speed; a technique based on triangulation allows the crew to get the fixes for the line endpoints without having to pass by; marine current is detected automatically and included in the calculation.

Compared to Oceanus, the architecture of the system is quite demanding, as it requires a normal PC to be running on board and connected to a WiFi router.

[8] Visited in August 2018, but pages were last updated in 2014.

The PC runs Ockam's application within a web server. Users can adopt their own mobile devices. The UI does not seem to be optimized for high visibility and no context-aware features seem to be made available. Furthermore it seems it require crews that are trained to use the system. Energy consumption of OS5 can also be an issue on board of small yachts.

Another system, Triton^2 of B&G (`https://www.bandg.com/bg/series/triton/`), uses a dedicated display for direct sunlight viewing: its features, besides a completely customizable displaying of wind, speed, depth, distance to target, and heading, include advanced functions of autopiloting like SailSteer which allows, together with a dedicated keypad hardware, to automatically perform steering maneuvers, taking into account wind changes, laylines, tide and current information. However, such wealth comes for a price: the cost of the central unit of the autopilot computer ranges from 840 USD to 1,499 USD (depending on the particular configuration). Whence, a complete system (with all the sensors, actuators and displays) can easily reach the price of several thousands of dollars.

The same applies for other well-known products by other manufacturers (featuring more or less the same characteristics, including the steering assistant), e.g., the Simrad (`https://www.simrad-yachting.com/`) AP70 autopilot, whose price for a complete configuration ranges from 3,300 USD to 4,350 USD. Other renowned producers of similar solutions are Garmin with Nexus Marine Instruments (`https://www.garmin.com/uk/nexus`), ComNav (`http://www.comnavmarine.com/`), Sailmon (`https://sailmon.com/`).

Of course, our proposal is not meant to be a competitor of the above mentioned solutions. However, it is interesting to see that many professional features can indeed be provided in an effective way, even with low cost hardware. Furthermore, we implemented a context-aware intelligence which proactively proposes significant information and visual hints to the crew, without requiring neither an excessive cognitive load nor an intensive interaction with the graphical interface. Such features rarely show up even in very expensive products, where users have to go through a preliminary customization of the information shown in the displays, if they do not want to be overwhelmed by too many details.

Finally, our system is built on top of open source software; 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.

9 Conclusion

After several tests and feedback collected from different crews (see § 3.1), we believe that Oceanus represents an interesting low-cost solution, providing a simple, yet effective, context-aware navigation aid both for sailing race crews and for amateur yachtsmen willing to improve their sailing skills.

In addition to extendability, one of the most useful features provided by our platform is the ability to use multiple clients for different crew members,

with different user interfaces according to the different user roles, with different user devices. For instance, the device being used by the tactician can display data that differ from those appearing on the helmsman's device, since they have different information needs. According to our knowledge, this feature cannot be found even on much more expensive solutions available in the market.

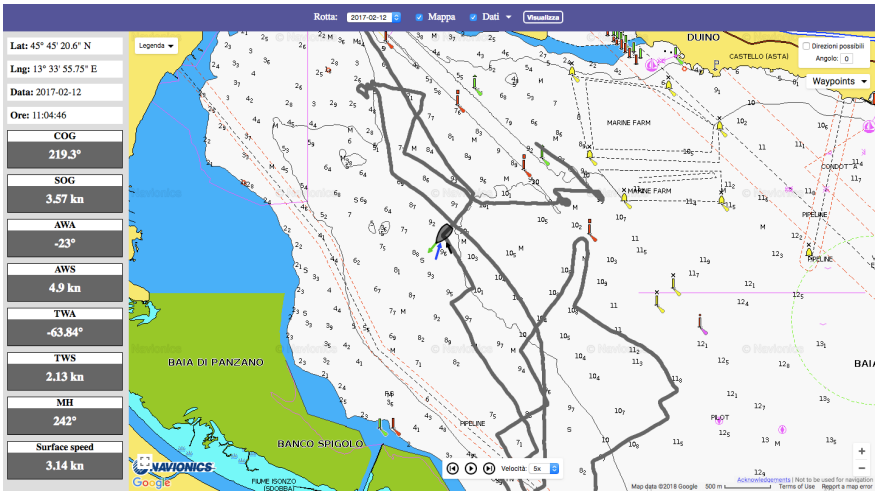

Figure 11: A screenshot of a web application allowing to analyze ex-post sailing sessions. The path of the sailing boat is displayed as a gray line: the user can zoom in and out, he can choose the speed of the movement of the boat (1x, 2x, 5x, 10x), whose current position is represented by a thumbnail with its current direction (green arrow) and the true (black arrow) and apparent (blue arrow) wind directions. On the left detailed data (Lat, Lon, Date, Time, COG, SOG, AWA, AWS, TWA, TWS, MH, SOW) about the current position are displayed.

Moreover, after two years of development and use of Argos, we have a wealth of log files which, besides being a useful resource for debriefing sessions (in Fig. 11 there is a screenshot of a web desktop application we developed and we commonly use to review sailing performance), represent a precious source of data for machine learning purposes. We plan to apply such techniques in order to learn a faithful model of the boat which, in turn, could lead to more accurate and timely navigation aids during races. For instance, we could infer better polar diagrams and support accurate leeway predictions. These can be exploited for obtaining accurate estimates of marine current and therefore for providing corrections to laylines, wind speeds and angles, and boat targets.

Other research openings are concerned with installing appropriate sensors on

buoys and making Oceanus capable of detecting them, and deploying Oceanus on all the boats of a race fleet, and manage therefore a distributed system of sensors, which will provide a dynamic map of the race field with respect to boats, winds, sea conditions.

A network of distributed systems of sensors can be used also for other purposes than supporting racing crews. In fact, sensors of chemical properties of the water (like those measuring salinity) could be easily integrated within Oceanus and in this way an entire water area could be monitored continuously by a fleet of leisure yachts. Argos in each of the yacht would collect this information and when the yacht is docked it would upload the data to the cloud.

Furthermore, the context-aware logic of Neptune will be enriched with new screens, for example for monitoring tacking performance and for automatically switching to it. In addition, besides the ability to integrate other sensors, we plan to work on the Context-Aware Layer of Neptune. Currently it is customized for a very specific scenario (namely, assisting the crew of a sailing race), but we are considering to make it more general and easy to replace so that it can support different kinds of UIs for different scenarios.

In conclusion, the best added value of Oceanus relies on its modularity and openness to extensions. We believe that it could help in increasing the level of support that electronic equipments can provide to sailors. In the next months the software of Oceanus will be published as open source software.

Acknowledgments

We would like to thank the crews of William B for their useful suggestions in making and fine tuning the system presented in this paper, and the students who contributed to implement the hardware and software solution.

We also thank the University of Udine for sponsoring this project within the Uniud Sailing Lab.

References

[AAC16] U. Alegre, J. C. Augusto, and T. Clark. Engineering context-aware systems and applications: A survey. *Journal of Systems and Software*, 117:55–83, 2016.

[Coh10] M. Cohn. *Succeeding with agile: software development using Scrum*. Pearson Education, 2010.

[Dey01] A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[Dou04] P. Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.

[FH91] N.V. Flor and E.L. Hutchins. Analysing distributed cognition in software teams: A case study of team programming during adaptive software maintenance. *Reading in Groupware and Computer supported Cooperative Work. San Mateo, CA: Morgan-Kaufman*, 1991.

[Got13] J. Gothelf. *Lean UX: Applying lean principles to improve user experience.* " O'Reilly Media, Inc.", 2013.

[Hut95] E. Hutchins. *Cognition in the Wild.* MIT press, 1995.

[Kar13] Charles F. F. Karney. Algorithms for geodesics. *Journal of Geodesy*, 87(1):43–55, Jan 2013.

[Kru00] S. Krug. *Don't make me think.* New Riders, 2000.

[Nar96] B.A. Nardi. Studying context: A comparison of activity theory, situated action models, and distributed cognition. *Context and consciousness: Activity theory and human-computer interaction*, 69102, 1996.

[nme18] National Marine Electronics Association. `https://www.nmea.org`, 2018.

[orc18] ORC Organization. `http://www.orc.org/`, 2018.

[RC08] J. Rubin and D. Chisnell. *Handbook of Usability Testing.* Wiley, second edition, 2008.

[top18] Movable Type Scripts. `https://www.movable-type.co.uk/scripts/latlong.html`, 2018.