Mission Planning for Deep Sea Robots

June 2019

Gijs Koning, Thijmen Langendam, Dennis Mouwen and Jochem Raat

TU Delft Allseas LOBSTER

Final Report

Mission Planning for Deep Sea Robots

by

Gijs Koning
Thijmen Langendam
Dennis Mouwen
Jochem Raat

in partial fulfilment of the requirements for the degree of

Bachelor of Science
in Computer Science and Engineering

at the Delft University of Technology,
to be defended publicly on 2 July 2019 at 16:00

Client:	Jeroen Breukels
Coach:	Mark Neerincx
Bachelor Project Coordinators:	Huijuan Wang & Otto Visser

An electronic version of this thesis is available at `https://repository.tudelft.nl/`

TU Delft

Acknowledgements

We would like to express our gratitude to Mark Neerincx, our coach, for his guidance, encouragement and useful critiques of this bachelor end project. We would also like to thank Jeroen Breukels for his advice, insights and assistance as our client at Allseas. We would also like to express our great appreciation towards Jelle Vos for his assistance and critiques throughout the course of this project. Finally we would also like to extend our thanks to the employees of Allseas for their thoughts on our product and their explanation of the company.

Summary

The LOBSTER student team in Delft is building an Autonomous Underwater Vehicle, that aims to be a low-cost method to reach the deep sea. This project is sponsored by Allseas. Since the robot is completely autonomous, all missions need to be planned in advance. Before we started this project, there was no way to create these mission plans except for writing a program to run each mission manually. Therefore, a mission planning application was needed that was easy and intuitive to use, and would not require a lot of training to use.

During a period of ten weeks we researched, designed, and developed a product to solve this challenge. We started by analysing the problem and existing mission planning software. We coordinated with various stakeholders, like the LOBSTER team and Allseas, and formulated the initial requirements for our product, which consisted of 16 must-haves, 14 should-haves and 10 could-haves.

Based on our research, we concluded that the application needed to be completely usable to create complex real-world missions. Furthermore, the product had to be usable offline, since planning often takes place on a ship where internet access is scarce. Most importantly, the product must be able to export the mission plan in a standardised format, such that it can be used by the robot.

In the following weeks, we built the product based on these requirements, while staying in contact with the various stakeholders. From the start of the project we made use of both unit and integration tests to ensure the correctness of our code base. These tests helped us find bugs and prevented old bugs from re-occurring. We worked using the Scrum methodology and used code reviews for every change in our code.

We implemented several sanity checks, which include not allowing certain actions to precede each other, or by showing a warning when filming in the deep sea with the lights off, and many more. Together these help prevent mistakes made in planning the mission and supporting the user in mentally modelling the mission.

Once we had a minimal viable product, we started user testing to find new improvements and issues. From these tests, we found various valuable improvements and we were able to improve the usability of our application. In the last weeks of the project we performed a final summative usability study which allowed us to evaluate the final usability of our product.

The final usability study with 9 participants, who had experience with underwater robots, showed that our product can be used successfully to plan a mission from start to finish without the users needing any additional training for the product. Furthermore, our product fulfils all must-haves from the requirements and almost all should-haves. The product can easily be extended to work for other robots by simply adding the properties of the new robot. In this way, the product is a great start to solve the original challenge not only for the current Autonomous Underwater Vehicles of the LOBSTER project, but also for any future robots.

Contents

Chapter 1

Introduction

Foreword

For the past 10 weeks we have been working on our Bachelor End Project. As this project is the final stage of our bachelors degree, we were able to apply all the skills and knowledge we acquired during our study to one project. We used our project management skills to design the product from scratch and we used our technical skills to build a working and maintainable product.

This report will describe the choices we made during the project and the challenges encountered. We will describe the final product and how we evaluated its effectiveness. Finally, we will recommend which further improvements can be made to the product.

The Goal

The goal of our project is to create a *mission planner* for the LOBSTER deep sea robots. The main robot of LOBSTER is the LOBSTER Explorer. This is an Autonomous Underwater Vehicle (AUV), designed to dive up to a depth of approximately 4 kilometres into the ocean. The Explorer's biggest selling point is that it is much cheaper than other AUVs currently on the market.

The downside of this however, is that the Explorer does not have all the features other AUVs have. For example, the Explorer is not able to determine its exact latitude and longitude while underwater, which creates an interesting set of challenges. Furthermore, the robot needs to act completely autonomously while underwater, since it is unable to communicate with the base station.

Because of this lack of contact with the base ship while underwater, the LOBSTER robots require a mission planner to specify the entire mission in advance. This mission planner needs to allow the owner of the robot to plan the missions they want to perform, and upload this mission plan to the robot in a pre-defined format. Furthermore, the application should assist the user in creating the mission they want, and try to prevent mistakes from being made. This is especially important since a mission might take hours or even days, so a mistake in the mission plan could waste a lot of time.

The original proposal (see Appendix B) also stated that we would design software for high level mission control, e.g. interpreting the output of our mission plan to low level robot controls such as 'Move to this depth' or 'Move in this direction for 10 seconds'. However, the low level control interface of the LOBSTER AUVs was still in active development during our project and not yet ready for us to work with.

Therefore, we decided during the research phase of the project, in coordination with the stakeholders, that we would not be working on mission control. We also did not include any mission control features in our requirements for this reason.

The Team

We worked on this project in a group four of Computer Science Bachelor students: Gijs Koning, Thijmen Langendam, Dennis Mouwen and Jochem Raat.

Our TU Delft coach is Mark Neerincx, professor of Human-Centred Computing. Our client is Jeroen Breukels, Unit Head Innovations at Allseas[1]. We also worked together with people of the LOBSTER team, specifically Jelle Vos, who helped us better understand the technical aspects of the robot.

[1] `https://allseas.com/`

Chapter 2

Problem

In this section we will describe the problem we set out to solve during this Bachelor End Project. We will explain why this is a problem, what makes this problem unique and why a solution is necessary. Furthermore, we will look into current solutions for similar problems and explain how they relate to our problem.

2.1 Background

The LOBSTER student team is developing a deep sea robot which aims to be low cost, fast and lightweight: the LOBSTER Explorer. To achieve these goals, various design decisions were taken which make the LOBSTER Explorer robot a unique project. First of all, the robot is completely autonomous once it is powered on and it cannot receive instructions from the user during its mission. Furthermore, it does not have any mechanism to determine its exact latitude, longitude and the speed it is travelling at while submerged. It does however know its exact depth using pressure sensors and its direction using a magnetometer and gyroscope.

From these properties of the LOBSTER robot follows an important problem which our project addresses: If the robot is autonomous once underwater, the user needs to be able to specify a comprehensive mission plan up front. Additionally, this specification needs to be usable by the robot itself to carry out the plan and thus the plan should be specified in a documented and unambiguous format.

The problem is to define a format for this mission plan, and develop an application which allows the users of the robot to create a mission plan effectively. This application should assist the user in the process of creating a mission plan as much as possible. For this reason, it should not allow the creation of invalid or erroneous mission plans. Moreover, it should provide the user with feedback on the created plan, for example with estimations of the total mission duration and battery usage.

The group most directly affected by this problem is the LOBSTER student team themselves. They are developing the robot and will be the first ones to use any mission planning solution to run the first test missions. As long as this problem is not solved they are not able to easily create or modify missions and instead need to program each mission separately.

As a result of this impact on the LOBSTER team our client Allseas is also affected by this problem. They currently do not use AUV's on a regular basis and this means that for them this is also a pilot on how things might work in the future.

2.2 Current Solutions

We looked for similar products primarily by using search engines. We used both Google and Google Scholar to search for terms like "Mission planning", "AUV software", "AUV planning", etc. Then we selected any mission planning systems for Autonomous Underwater Vehicles (AUVs) for which we could find sufficient

information. Furthermore we also included some interesting products aimed at other types, such as aerial, land and even space vehicles. The key differences between the similar products we found are summarised in Table 2.1, which we will elaborate on in the rest of this section.

As can be seen in the summary, almost all existing products are desktop applications, with the notable exception of the NASA Open MCT web application. Furthermore seven of these systems also include a mission control interface which allows interaction with the vehicle during the mission. However this is not applicable in our case as the LOBSTER vehicle is not able to communicate while underwater. In the vehicles column we indicate which specific robots the software is aimed at or a dash if it is applicable across a wider range of vehicles.

Finally we looked at the main User Interface approach taken by each of these products, which is summarised in the last column. Most planners for underwater vehicles used a 2D top-down map of the seabed at the mission location. On this map the coordinates of the mission can then be drawn. The exception is the Marius mission planning system which is based on Petri nets, which can be specified graphically. These Petri nets essentially capture the sequence of steps based on conditions in the form of a graph. [13]

Name	Platform	Type	Planning	Control	Vehicles	UI Approach
VectorMAP [12]	Desktop	Underwater	✓	✗	Iver AUVs	2D map
MIMOSA [7]	Desktop	Underwater	✓	✓	Ifremer AUVs	2D map
Triton [8]	Desktop	Underwater	✓	✓	-	2D map
Marius [13]	Desktop	Underwater	✓	✓	Marius AUV	Petri nets
UgCS [16]	Desktop	Aerial	✓	✓	-	3D satellite
ArduPilot [3]	Desktop	Aerial	✓	✓	-	2D satellite
Open MCT [10]	Web	Space	✗	✓	-	Multi timeline
Mindstorms EV3 [9]	Desktop	Land	✓	✓	LEGO robots	Block timeline

Table 2.1: Comparison of various mission planning and/or control systems

On the other hand we have two products aimed at aerial vehicles which make use of satellite images. Of these UgCS displays the area in 3D, which allows the user to intuitively see the altitude at various points of the mission. ArduPilot however simply displays the 2D top-down satellite view, which does not easily display the altitudes.

Then we have Open MCT which uses various UI elements of which the most prominent is a multi-track timeline. On this timeline various operations can be scheduled in time, which also allows simultaneous actions. Another timeline based approach is used by Mindstorms EV3, which lays out the operations as blocks which can be put into the right place. This approach complicates simultaneous actions, but is more visual and intuitive.

Overall we found that most underwater mission planners use a top-down map view, which is not well-suited to our product since the control of the LOBSTER is not location-based. Furthermore the Marius system used Petri nets, which allow flexibility with user-specified conditions. On the other hand we also found some programs (Open MCT and Mindstorms EV3) for other fields which use a timeline-based approach which seems more applicable to our product.

2.3 Conclusions

As far as mission planning goes, there currently are no solutions that fit our project needs. Most mission planning software are made for Remotely Operated underwater Vehicles (ROVs), thus making them unsuitable for our robot. The few applications we could find for AUV's worked pretty similar to aerial drone

mission planning software. They provide a top down map (and sometimes a 3D view) that lets the user select locations on the map which the robot must follow one by one. As stated before, our robot does not have a precise location underwater, making this approach unfit for our project.

On the other hand, Open MCT and Mindstorms EV3 use an approach based on actions, which is more applicable to the LOBSTER robots. However, both of these are not aimed at underwater exploration, where the possibility of movement is 3D instead of 2D. Therefore, they do not provide the user with useful visualisations for this purpose, which our solution will aim to do.

Overall, none of the found similar products are able to solve the problem for LOBSTER. None of them offer the right approach for an AUV that does not know its location. Furthermore, all of the existing products aimed at underwater robots do not provide a way to support new robots. Since these programs are all proprietary, it is not possible to take one of these products and adapt it to work with the LOBSTER robot.

Chapter 3

Product Design

In this section, we will discuss the design of our product and describe our view of our solution to the problem described in Chapter 2. Following from the problem and these assumptions, we will describe the design choices we made. Starting with the initial design choices from the research phase followed by the further decisions made through the course of the rest of the project.

3.1 Solution Design

In Chapter 2 we have defined and analysed the core problem, however just analysing *the problem* is not enough. We also need to design our solution to this problem. Luckily with the problem analysed we have a solid foundation on the general idea behind the solution: Creating a product which can easily be used to plan missions. The specifics of this product and how we plan it to be used are not yet clear, so we will describe these in this section.

3.1.1 Product Usage

An important thing for us to decide is the context and environment in which our product will be used. We decided this based on discussions with our client, coach and other stakeholders such as the LOBSTER team and possible users like researchers.

In our discussions we discovered that missions sometimes need to be planned offshore, instead of on land. However, internet access on the sea is often slow and sometimes limited to certain areas of the ship only.

Hence, we have to consider the fact that our product will sometimes be used on devices which have (almost) no internet connection. Therefore we concluded that our solution should be usable for the complete mission planning process whilst offline. By that we mean that the application can be used to complete the entire mission planning process, from start to exporting the mission plan.

However, we encountered some features that would provide value to users, but which do require internet access, such as the weather forecast or an interactive map. Adding these features to our product would be valuable for those times when the program is used in a situation where internet access *is* available. We therefore decided that implementing non-vital features which depend on internet access is fine for our solution, as long as the application still works without these features when there is no internet access.

3.1.2 Robot Location

Since the robot does not guarantee a precise location, all data we show the user regarding location is an estimation. The precision of this estimation depends on the robot. We also need to ensure it is clear to the

user that everything is an estimation, so that they don't put take the provided estimations as precise facts. A simple solution is to just tell the user when information is an estimation. We applied this principle in all locations were this was possible.

3.1.3 Sanity Checks

Before you send the robot underwater to perform its mission, you want to make sure everything is set up correctly and that there are no errors in the mission. But what if what you are trying to do is not an error, but simply a mistake? For those occasions, we decided to add sanity checks: Simple checks that verify you are not doing anything you did not mean to do. One of these checks might warn the user to "turn on your lights when filming deep underwater, or else you will have dark footage". These checks guide the user in creating their mission and make sure the robot comes back with valid data. We want to add these sanity checks wherever possible.

3.2 Initial Design Choices

Listed below are the design choices we made during the research phase, taken from our research report. You can find the entire research report in Appendix D.

3.2.1 Target Platform

The target platform of our application is an important decision to be made, since it influences the possible choices of programming languages and libraries. We discerned two main feasible choices of target platforms for this project, targeting the web or a native desktop platform.

Web Application

The web allows developers to create one application which can be used across a variety of systems, since web browsers are available on all desktop systems. This is one of the main advantages of developing for the web, since there is no need to develop separate applications for separate systems. Furthermore, web applications can be used without installation, which simplifies the usage.

On the other hand, web applications also have some disadvantages, such as only being able to use JavaScript-based languages. Additionally, web applications are not available offline by default, so if this is desired it needs to be specifically implemented.

Native Desktop Application

Desktop applications in general come with various advantages. Since desktop applications do not need a run-time security layer, unlike web applications, they can respond faster. Additionally, native desktop frameworks have access to more system functionalities, such as advanced 3D graphics. Finally, any desktop application is available offline without additional effort.

On the other hand, there are also some drawbacks inherent in the approach of desktop applications. Desktop applications require an installation step. Furthermore, although the user interface framework often provides portability of the graphics, it does not handle other tasks such as accessing the file systems. Therefore, additional effort is required to ensure correct portability among operating systems.

Chosen Target Platform

In the end, we decided that the best fit for our project was to develop a web application. We noticed that most advantages of desktop applications are not applicable to our case, since we don't require advanced 3D graphics or ultimate efficiency.

However, the advantages of web applications, such as ease of use and portability are relevant to our goals. Another factor in the decision making process of our platform was the fact that LOBSTER already works with different UIs that are also web-based, which would lead to an unnecessary increase of complexity when using different platforms for the same product. Finally although it does require some amount of extra effort, we can still make our web application available offline if necessary.

3.2.2 Web Framework or Library

After we chose to target a web application our next decision was which framework (if any) to use. We considered the following possibilities: no framework, Vue.js[1], Polymer[2] and React[3].

One of the approaches to web development is not using any additional framework at all. An advantage of this approach is being independent of additional libraries, and therefore not being limited by the possibilities or requirements of a certain framework or library. However, the downside is additional work on features which could be provided by a library.

Another approach is using a flexible and minimal framework such as Vue.js, which can be used 'incrementally'. This allows developers to choose to which level they use the framework, which provides more flexibility. However, a drawback of this approach is that the framework is less comprehensive and does not provide a syntactic sugar to build components.

React is another option, which is aimed at building interactive user interfaces. It allows developers to build components which manage their own state, from which the application is built. Furthermore, React provides an efficient syntax to describe these components. It also enforces the data hierarchy which can be passed on from parents down to children but not the other way around.

Finally, we also considered Polymer which allows developers to use Web Components. Using polymer, a large library of existing Web Components which can be used in other projects is available[4]. These components can then be customised and incorporated. However, for our purposes one of the downsides of Polymer lies in its complexity. Polymer does not enforce a rigid structure of data hierarchy, instead allowing data flow in both directions, which can result in unnecessary complexity.

Ultimately we decided to use React for our web application, primarily because of its structured approach to components. Using React we will easily be able to create the custom interactive components needed

[1] `https://vuejs.org/`
[2] `https://www.polymer-project.org/`
[3] `https://reactjs.org/`
[4] See `https://www.webcomponents.org/` for a large collection of Web Components

for our interface. Furthermore, the syntax of React will allow us to specify the logic and content of our components close to each other in an elegant way.

3.2.3 TypeScript

Another decision we made was to use TypeScript instead of plain JavaScript. This language adds the option to add types to parts or all of your program, which can help reduce problems at run-time. Furthermore TypeScript is a strict superset of JavaScript, meaning anything that can be written in JavaScript can also be written in TypeScript. The only downside of using TypeScript is the extra required step of compiling the code to JavaScript. However this process can easily be automated so that it is done automatically upon code changes. Therefore we have chosen to use TypeScript to allow ourselves the additional possibility of type checking our code.

3.2.4 Mission Routing Visualisation

The visualisation of mission planning was another important decision. It is the direct link between the idea behind the mission itself, and how the robot will interpret the commands and thus act during a mission. The three different approaches we considered for this were in 2D and 3D Maps and finally a modular timeline. All discussed methods have their benefits and drawbacks which we will describe in the following sections.

Top and side-view map

The first idea that we came up with when we were discussing the visualisation of the route was using two maps, one from above looking onto the sea surface, and another map looking from the side.

The main problem of this implementation was the unknown angle of the side view as the robot can move in three dimensions. Additionally, a two dimensional map would not be able to show movement in the z-direction without clear indication and could even lead to overlapping points of interest.

3D Map

A straightforward solution to the aforementioned problems is to make use of a single three dimensional map. Coordinates would then be shown in this map at places where the robot would execute certain actions, easily visualising the entirety of the mission it would then execute.

However a clear drawback of this method is the fact that a three dimensional map requires a significant amount of processing power compared to two dimensional alternatives. Next to this, another problem would arise when users would click on the map to add points in this three dimensional space, as a flat screen cannot show the clicking depth.

Modular Timeline

A modular timeline works with different categories of actions, each uniquely identifiable by their colour. These actions can be added by the user to the timeline in a block or chain building fashion by dragging

the different actions onto the timeline in their desired order. Additionally, it allows an easier calculation and visualisation of the power and time usage of each action, and can thus help the user in the mission planning.

A comparable type of software that also uses a modular timeline for programming robots is the LEGO Mindstorms EV3 software (discussed in Section D.3). This software is used to control the behaviour of the robots both in movement as well as all related actions. This software was also created using a modular timeline to allow children to understand and build their own robots.

Chosen visualisation method

The modular timeline was chosen because it is an easy way for all users to understand the flow of the mission, and can also adapt to the user's choices dynamically. An example of this dynamic adaptability is that once a user adds a retrieval action, it no longer allows actions that can only be executed deep below the surface of the water.

Sketching the Modular Timeline

During the design choices of how this map would look like we did some sketching and came up with two main designs, as shown in Figure 3.1 and Figure 3.2 . Figure 3.3 is an extension of the first design as this design used graphics or icons for each of the actions.

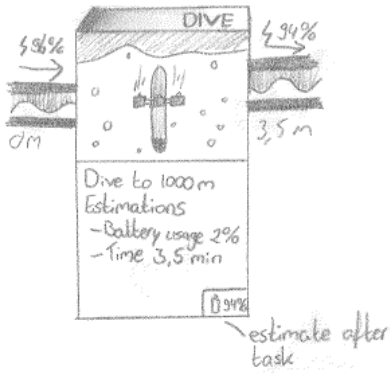


Figure 3.1: The first design, including an icon of the specific task and a thicker timeline line.

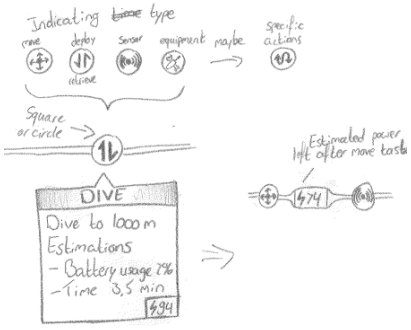


Figure 3.2: The second design, having task specific blobs and a thinner timeline with integrated charge estimation.

Next to the way we wanted the timeline to look like we also had to think of the entire user interface besides the timeline. Since a user needs to grab all these different tasks, we added side-tabs that house the tasks by category. Furthermore a button to export the current mission was added. This all was drawn together in the first UI sketch (Figure 3.4)

After looking over these sketches with the entire team we were missing some features, and thought of some quality of life features that would not be too complicated to add. These were for example an estimation of the battery percentage and time spent on this mission, as well as a zoomed out mini-map of the timeline. (Figure 3.5)

Added to these were two slightly larger features that helped the user in the creation and maintenance of their missions, these being an option to select some tasks and "saving" this sequence for later replica-

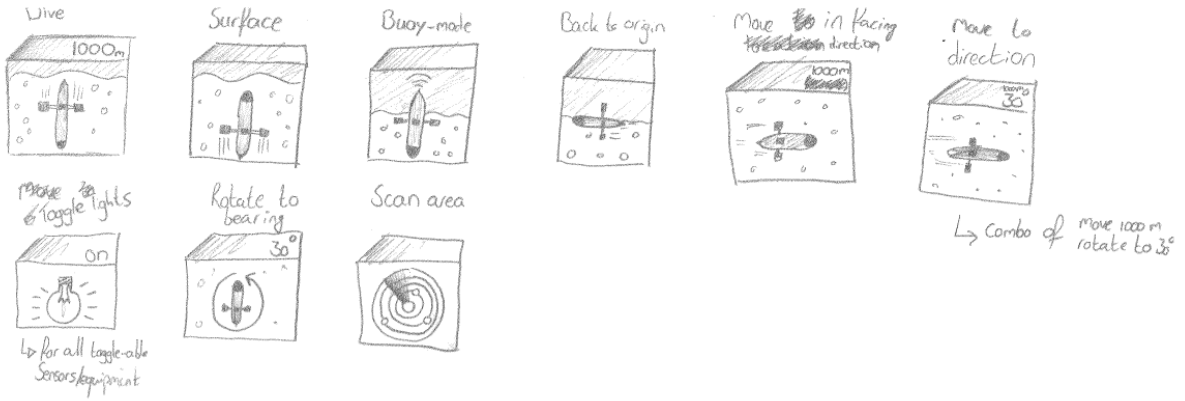


Figure 3.3: Sketches of some of the actions that a user could use to create a mission for their robot. We extended these icons with extra information below as seen in Figure D.1 to add clearer information such as how far the robot would dive, or to where the robot should rotate in degrees.

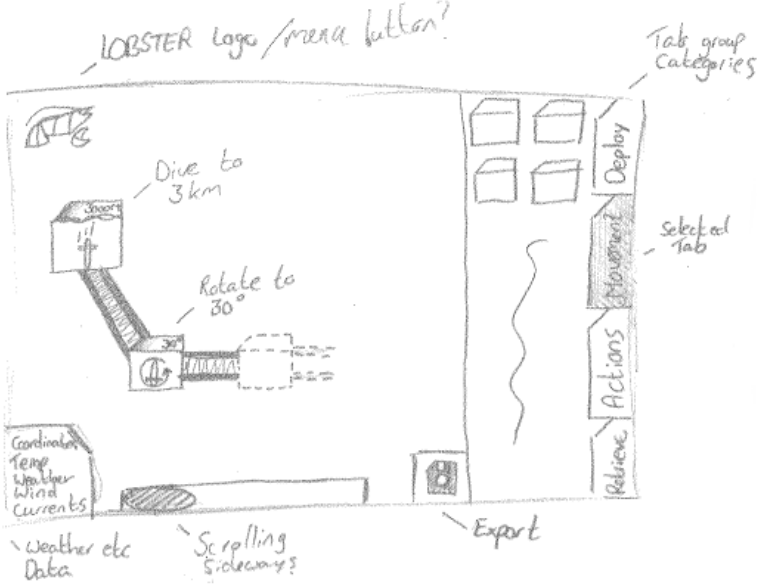


Figure 3.4: A sketch of the first iteration of the entire UI

tion (Figure D.6), but also being able to select tasks and grouping them as a "Phase", allowing for better organisation of the mission planning.

3.2.5 Interface architecture

To further clarify our understanding of the intended user interactions we created a diagram which can be found in Figure . This diagram maps out the possible transitions between actions and screens by the user within our application.

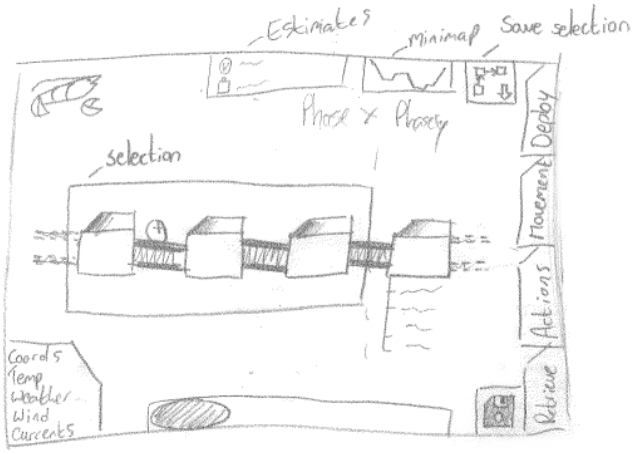


Figure 3.5: A sketch of the second iteration of the entire UI

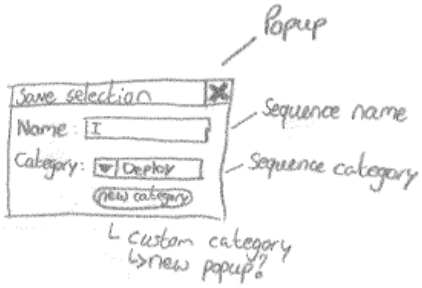


Figure 3.6: Popup that appears when saving a selection of tasks

3.2.6 Exported file formatting

To export the missions created by the users, we decided to use the JSON file format. We preferred this format over others because of multiple reasons, the first being the fact that JSON is a widely used file format for web-based data storage. As mentioned in Section D.4.1 we chose for a web-based application, and we will thus use JavaScript. Additionally every popular language has interpreters for JSON files and therefore this file format is an easy file to read for the robot interpreters to convert to actual commands. Last but not least, JSON is a simple format to use and read both for humans and machines.

After we decided that the JSON file format was the best choice for this project, we also had to decide on how these files would be structured, as JSON files are incredibly flexible in their style. Luckily, JSON Schema [14] comes to the rescue. JSON Schema is used to annotate and validate JSON documents. This way, we can define a set of rules to which the output file must cohere. We can also use this to test the output of our program. An example of such a schema file can be found in Appendix H.

3.2.7 Multiple robot compatibility

Initially, the idea was to create this software just for the LOBSTER robot. However during our first meeting it came to our attention that it was possible that the LOBSTER robot was not the only robot in need of this software. For example, second generation LOBSTER or other robots would need entirely new and specific software.

We decided to add the option for adding custom robot specifications and saving these for later use, and also including pre-made robot templates for the user to work with. These settings would then change the options available to the planning of the mission depending on the selected robot.

3.2.8 Testing

React comes together with the testing framework Jest. Jest is made for both unit and integration tests, which means we can use it to create all of our tests. It even supports UI snapshots, so allowing us to verify

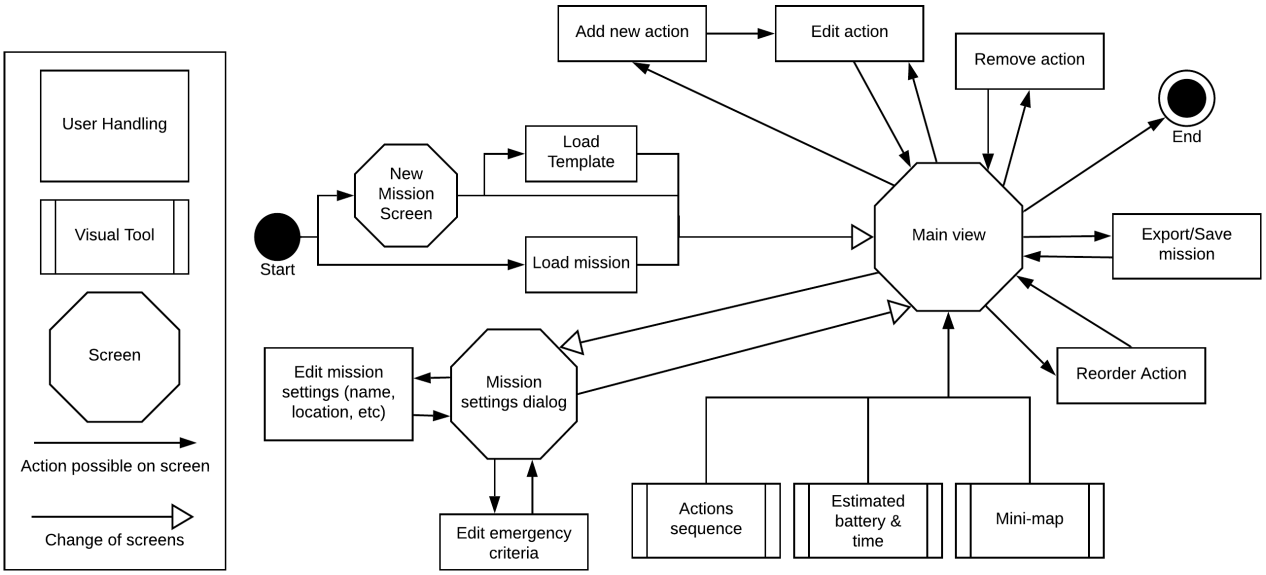


Figure 3.7: Diagram of user interactions with the application

UI changes.

3.2.9 Logging

To improve debugging and data extraction, we decided to add logging points to the timeline. These logging points determine what kind of data should be logged, when it should be logged, and how often it should be logged. The implementation of how the robot should log this is out of the scope of this project. We will only tell the robot when it should log.

3.3 Additional Design Choices

During the course of our project, we had to make additional design choices as we came across new problems, and clients change their needs. In this section we discuss the choices we had to make during the later phases of our project.

3.3.1 Mission Visualisation

When most of our must-haves were implemented, it was already possible plan a mission and work with the actions. As it turned out, it was pretty hard to visualise where the robot was during each part of the mission. Looking at the actions, the user would have an idea of what it was doing, but not where it was and how long certain actions took. It became clear that we needed to put some work in visualising the mission.

We started off with some basic designs, which you can see in G. These mock-ups already provided some insights in how the mission could be visualised and what information is useful to the user. We chose to implement the depth-over-time map and the top view map. They turned out to give the most valuable information, as well as not giving duplicate information.

3.3.2 Payload

During the project we came to understand that Autonomous Underwater Vehicles often support payloads, which can easily be switched for different missions. This allows the owner of the robot to add additional instrumentation for specific missions. Since the usage of these payloads needs to be specified in the mission plan, our application needs to support them for the user to be able to use them.

For this reason we decided to add the ability to add specify the payload items of each robot. These payloads can then be activated or deactivated for each action so that the user can control them easily.

3.3.3 User Interface Simplification

In the last few weeks of the project we noticed that the interface had become quite cluttered. A lot of colours were used throughout the interface, which made it hard to determine at a glance which information was important. Therefore we decided to limit the usage of large blocks of colours to the important aspects of the application. In this way the crucial warnings, such as when the estimated battery percentage is too low, stand out more. See Figure 3.8 for a comparison.

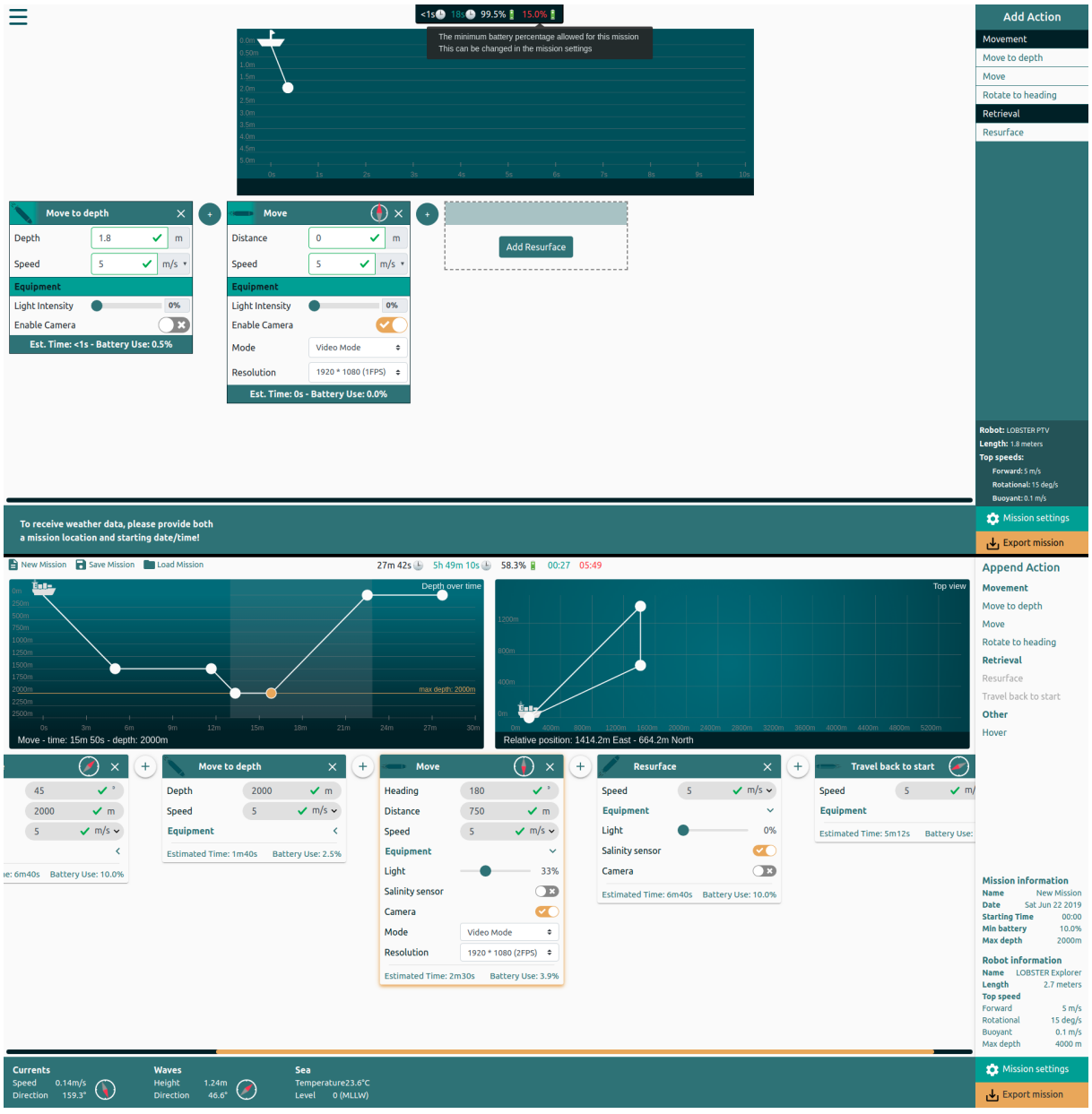


Figure 3.8: The application as it looked in week 7 (top) and in week 9 (bottom). In the second screenshot less coloured planes are used, to reduce clutter.

3.4 Conclusions

This design leads to a unique product in various aspects. First of all, it is the first application which will use a actions based timeline for Autonomous Underwater Vehicles. Other solutions for underwater robots focus on location based commands, which is not applicable to the low-cost LOBSTER robot.

Another distinguishing aspect of this design is its independence from any specific AUV. While the design is most suited to robots that can not determine their exact location, it can easily be adapted to any other AUV thanks to the robot specifications (as described in Section 3.2.7).

The application is also easier to install and use than other existing products. Most other products are native desktop applications, which means that they need to be installed by an administrator before use. Our product can be run without installing anything, simply by opening the program in a browser. Furthermore, our application can also be used offline by downloading it in advance, in contrast to other web based mission planning products such as Open MCT [10].

Finally, our application is distinguished by our focus on interaction design and usability. We aimed to make the interface easy to use and understand without any training. We tried to achieve this by showing information to the user only when need it. This is of course a balance between too much information and too little at the same time.

Chapter 4

Process

In this chapter we will discuss the process of our project. This consists of various aspects, such as the cooperation within our team, but also our interactions with external parties. We will describe the way we approached these aspects during the project and what we learned from them.

4.1 Software Methodology

During the course of this project, we have worked according to the Scrum [4] framework. Every week, we rotated the Scrum Master, to make sure we all had to learn this role. On Mondays, we reviewed last week's sprint, as well as planning our new sprint. We managed our Scrum board using a GitHub Projects board, where we wrote down all our issues and gave them labels according to their functionality and priority. We also wrote down our requirements, so all progress was immediately available to see.

Our Git repository was hosted on GitHub, which allowed us to use GitHub pull requests. These pull requests allowed us to review all code before it enters our code base. We configured our repository so that each pull request required at least two approving reviews. In this way we ensured that all code in our repository had been sufficiently checked.

We used Travis CI[1] to automatically test each code change. Additionally, we used Codecov[2], which listed the changes in test coverage for each change as well. Thanks to this setup we could easily see whether each change passed the tests and if it kept the test coverage sufficiently high. This helped prevent problems in our main branch and kept our code well tested, you can read more about our use of automated testing in Section 5.5.

4.2 Team

Within our team we tried to keep the cooperation efficient and healthy. We did this by fostering an open team culture in which each team member felt free to voice their thoughts, concerns, and frustrations. This openness was useful in cases where team members had conflicting ideas of the right way to do things and it helped prevent frustrations building up which could impede cooperation and communication.

To facilitate this openness we started the project by telling each other what we hoped to get out of the project and how we wanted it to go. We made some agreements on our work hours (9 AM to 5 PM each workday) and on the type of software methodology we wanted to use (Scrum, as discussed in Section 4.1).

At the end of each sprint we evaluated the last sprint with each other, which also allowed the opportunity to voice any concerns or problems. Halfway through the project, we held a group meeting to evaluate our cooperation up to that point as a group.

[1] `https://travis-ci.com/`
[2] `https://codecov.io/`

4.3 Skills Acquired and Lessons Learned

Working in a group during our end project provided useful learning opportunities as it was a different project compared to the ones we faced before. Usually the time spent on a project was around a third of the quarter, whereas during this project it was an entire quarter.

For one, we learned that with every group, there are always some people who are more skilled at different tasks than others. So it is always valuable to learn from their experience for future projects. In this way we we learned various skills from each other, simply by working together.

Another thing we learned was the importance of re-prioritising the focus of your work at the right times. As can be read in Section 4.5.1, we had a lot of open issues in week 6 for which it wasn't clear to everyone in the team how important they were. This caused us to work on some features that were not actually needed at that point, which slowed us down. In the end we solved this by having a meeting in which we talked about the importance of each issue as a team. So from this we learned that it is important to adjust your priorities during the project and make sure that all team members are in agreement about these priorities.

We also found out that the seating configuration of the team can be very influential on communication and cooperation. For the first few weeks of the process we were seated in such a way that we could not all see each other. We noticed that cooperation went much smoother once we moved to a bigger room where we could all face each other. From then on we were able to easily ask question to each other, which improved the communication a lot.

4.4 Ethics

During the process we continuously made sure that everything we were doing was ethical. Since ethics are such a broad concept, it relates to various aspects. For example, at the beginning of the project we considered the impact of our project on society. We concluded that this impact will be positive, since it allows people to use underwater robots more efficiently. This in turn will help with underwater research, which we think will have a positive impact on us all.

Another ethical aspect we considered during this project is our user tests. When working with other people you always have to be careful to make sure that you treat them correctly. To ensure that our tests were not dangerous for the participants we worked with we filled in the ethics checklist from TU Delft from which we found that our tests would be of minimal risk. You can read more about this in Appendix C.

4.5 Challenges

Throughout the project we had to continuously adapt to and overcome various challenges. This is a natural part of any project and these challenges provided great learning opportunities for our team. In this section we will describe some of the more notable challenges we encountered, how we handled them, and what we learned from them.

4.5.1 User Study Re-prioritisation

When we started with user studies, we received a huge amount of feedback. After the first two user tests we conducted, we already received around 30 to 40 comments on things that didn't work as expected. This resulted in a major challenge as we realistically could not implement all of these suggestions simultaneously and we did not know where to start.

We overcame this challenge by having a meeting with the entire team to re-evaluate our priorities. In this meeting we assessed all open issues and prioritised what we wanted to do next. Based on this prioritisation, we were able to make consistent and steady progress in the following weeks.

4.5.2 Software Complexity

Throughout the development process of our product one of the main challenges was managing the complexity of the code base. While a software products grows, it slowly becomes more and more complex, making it harder to change things. Therefore, keeping the complexity of the software manageable was one of our main priorities.

When we noticed things were getting too complex, we would often fix this issue by refactoring our code. This means we rewrote some of the code to reduce the complexity based on new insights.

Another lesson we learned about reducing complexity was not to create low priority features too early in the project. During the project we noticed that some features not necessary for the core product were complicating the rest of the code. We would have been able to develop our application more efficiently if we had implemented these features at a later point.

4.5.3 Working on the Report

All four of us liked working on the software of the product a lot, which made for a great atmosphere as everyone liked what we were doing. However, this also had a downside, since we liked working on the software more than working on the report. This led to us sometimes working too much on the software and too little on the report.

We solved this problem by scheduling times at which we all would work on the report and not code at all. We would encourage each other to work on the report to efficiently improve this part of our project.

4.5.4 Seabed Elevation Data

One of the sanity checks we wanted to implement was the ability to get an accurate estimation of the depth, or elevation, of the seabed at the starting location provided by the user.

To get this data, we had to use an external Application Programming Interface (API), of which a few were available. Unfortunately all but one of these APIs were either non-global data, paid services, no longer maintained or inaccurate. The only option was the Google Maps Elevation API, which we tried to fully implement into our product.

However, this API required us to add billing information as this is required by Google to use their API, even if you would not actually exceed the uncharged quota. This was a problem as none of us had access to a

credit card. We could not implement this feature due to this limitation.

4.6 Interactions with Stakeholders

For our project we had various stakeholders who all have different goals, requirements and perspectives. Throughout the project we tried to explore the requirements of these various stakeholders continuously. We used this information to inform our design choices in all aspects of the project. In this section we will explain how we interacted with each of the stakeholders and how that influenced our process.

4.6.1 Our Coach

Our coach was a stakeholder in our project as a representative for the academic aspect of the project, on behalf of TU Delft. His main interest was therefore to help us succeed in this project and ensure that it meets academic standards.

We met with our coach almost every week and during these meetings we would explain our current progress and challenges. He asked critical questions and gave valuable feedback. Especially valuable were the tips and feedback our coach gave us about usability testing, which helped ensure that we got very valuable information from these tests. More information about these tests can be found in Chapter 6.

4.6.2 Our Client

On the other hand, our client represented the interests of Allseas, who are mostly interested in a useful resulting product from the project. We also met with our client almost every week to discuss our progress. We showed a small demo each time and our client would ask questions and provide feedback. This feedback helped us make sure that we kept our focus on the features that were important to our client. For example in these meetings we discussed that we should first focus on having a working version of all necessary features and only after that we would focus on polishing the existing ones.

Additionally, our client helped bring us into contact with Pat Quakernaat of the Survey department of Allseas. We scheduled a meeting with him in the fourth week of the project. This meeting was very interesting and it greatly improved our understanding of the usage of underwater robots at Allseas.

The main takeaway was that Allseas needs very precise location data in most robots they use since they often need to measure things very precisely. Therefore the LOBSTER robot won't be immediately useful to Allseas, since it can not determine its exact location. However, the contribution our product makes to the LOBSTER project is still indirectly useful to Allseas since they want this project to succeed.

4.6.3 LOBSTER Team

An obvious stakeholder is the LOBSTER team, who will be using our product to plan the missions for their robot. Therefore, we also regularly met with a member of the LOBSTER team, Jelle Vos, to discuss their requirements and needs. Especially in the early stages of the project this was very useful to help us work out what direction we wanted to go in with our product.

Throughout the project these meetings helped us to keep our project on track towards a useful product for the LOBSTER team. In the user tests we also tested our product with members of the LOBSTER team to check that it was also usable by them, more about this can be found in Section 6.2.

4.6.4 Marine Researchers

Our product is mostly meant for Allseas and LOBSTER, but we did not want to create a product that is only usable for a small amount of robots. For this reason, we also had contact with the Royal Netherlands Institute for Sea Research (NIOZ). Especially in the beginning of the project we received a lot of feedback on how their systems worked right now, and what they would like to see in our product. Some feedback that they gave was: The program should be able to work without internet, since they often work off-shore. They would also like to be able to couple the sensor data with the navigation, for example to find a ship-wreck.

4.7 Product development

To give a perspective of how our program evolved during the project, we explain in short for each 2 weeks which new features and changes were made.

Week 1 & 2

In the first two weeks we spent most of our time on research. During the second week we also began implementing the first features. In the beginning we tried to prototype and test the features on their behaviour before fully implementing them. We created the basic layout of our program including a timeline containing the actions of the mission and an action list where actions can be added to the timeline. A dialog was also made to put in the information for the mission.

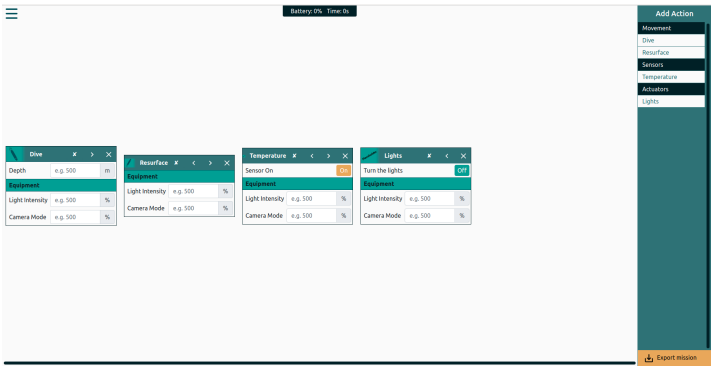


Figure 4.1: The first basic layout of our application

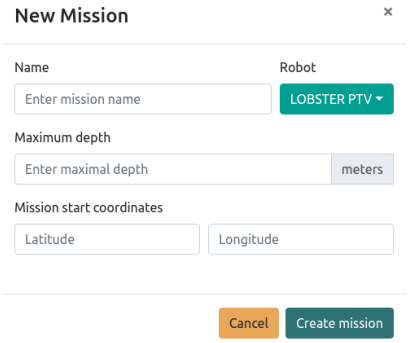


Figure 4.2: The mission settings form

Week 3 & 4

During week 3 and 4 the actions were extended with estimations for time and battery (See Figure 4.3), though they were not actually computed yet. The camera option was extended with a drop-down. A file upload dialog was added that allows users to upload a JSON file with an exported mission (See Figure 4.4). A critical condition dialog was made to make sure the robot would not exceed these limits during the planning of the mission (See Figure 4.5). The last feature was adding an action between two actions with the plus button.

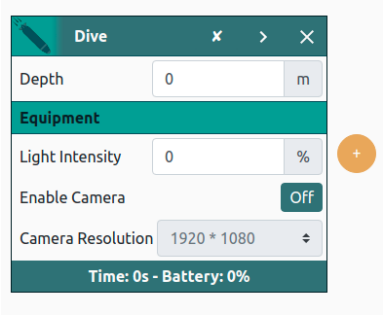


Figure 4.3: The improved action block

Figure 4.4: The screen for uploading and loading a mission

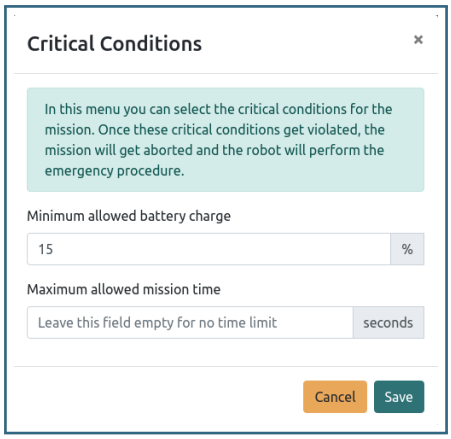


Figure 4.5: The critical conditions dialog

Week 5 & 6

The most important feature of this period was the depth vs time graph (See Figure 4.6). This enables to user to see how the robot moves during the mission. A world map for selecting a start location was added to the mission dialog. Estimations are computed as can be seen in the top info-bar. Finally, the camera inputs were extended to have different camera modes.

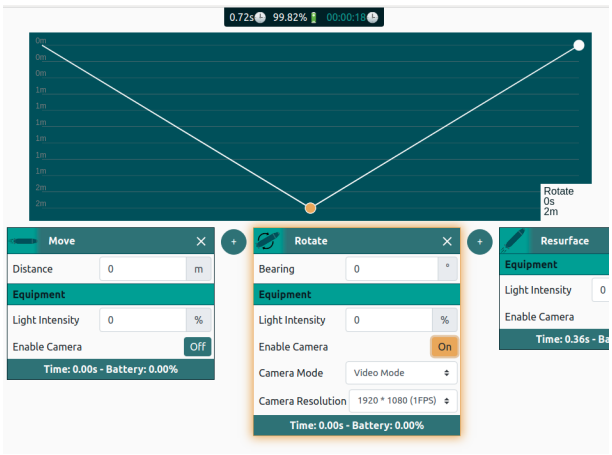


Figure 4.6: The depth over time graph

Week 7 & 8

A lot of different features where added and merged during these two weeks. Because most of these features are unchanged in the final product you can find the images in Chapter 5. To give more information about the robot during the planning of the mission, robot information is shown at the right of the screen. Mission information is also given in the same window.

Next to this a speed option was given to all actions that move, which the user can use to limit battery usage. We also changed the way actions could be added between other actions: a drop-down containing the action list, appears when clicking on the plus button. Weather data has also been added, showing information about the sea and the currents. It can be retrieved when the user has selected a time and location for the mission.

In week 8 we started to change the look of the application. This week the right bar and top bar were cleaned up. Some sanity checks were also added, like a warning when the user creates a new mission when the old mission is not yet saved, and a warning when the user enables the camera but does not have the lights on when diving in deep waters. The robot templates were extended to have different equipment, for example a salinity sensor or a temperature sensor.

The last big feature is the top-view map. It shows the user the position relative to the starting point of the mission during each step (See Figure 4.7).

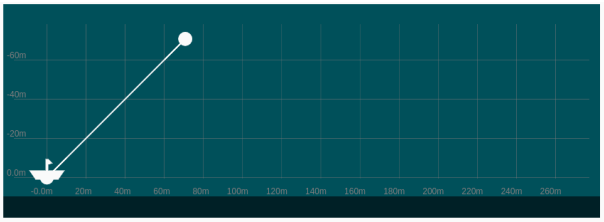


Figure 4.7: The top-view map of week 8

Week 9 & 10

During the final weeks we further improved the look of application. The action blocks look less cluttered and we added a gradient to the two maps. The battery estimations were improved to work more realistically. At last, some actions were added to improve the usability of the program. The changes of the final weeks can be seen in the next chapter.

Chapter 5

Final Product

This chapter will explain and evaluate the final product, giving more insight on how well the product solves the problem given in Chapter 2.

To obtain our final product you can send an e-mail to the contact person listed in Appendix A. Then we can give you access to the source code repository and send you the latest application build, which you can try on your own computer.

5.1 Main Features

In this section we will describe all main features that we have implemented in our application, why they are important for the user and why we have implemented them.

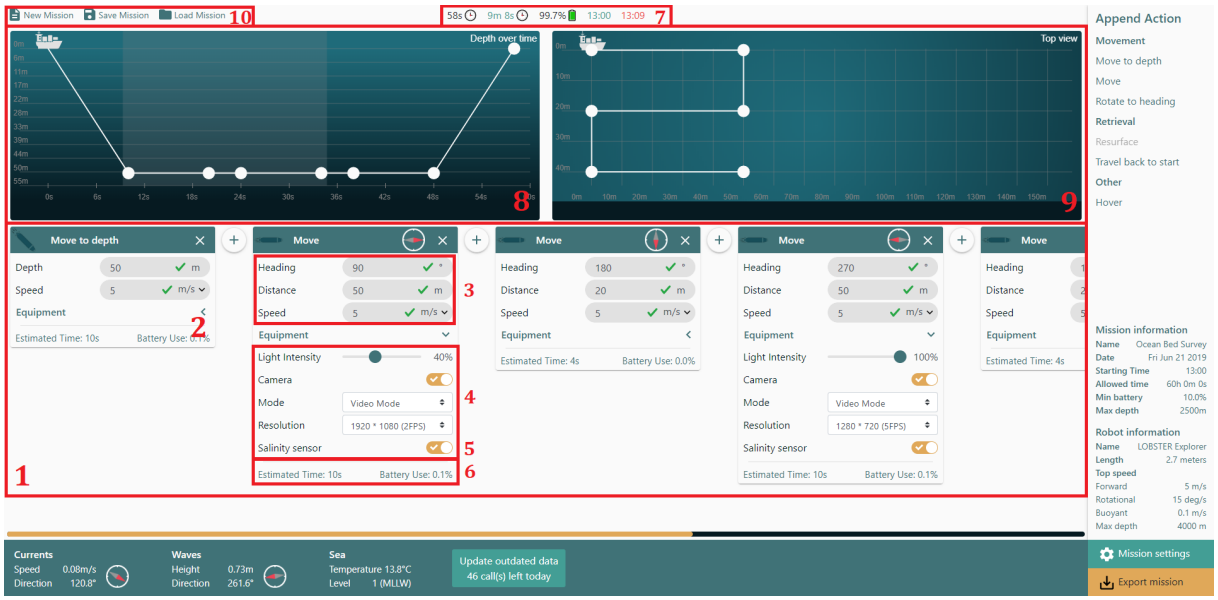


Figure 5.1: A screenshot of the final application

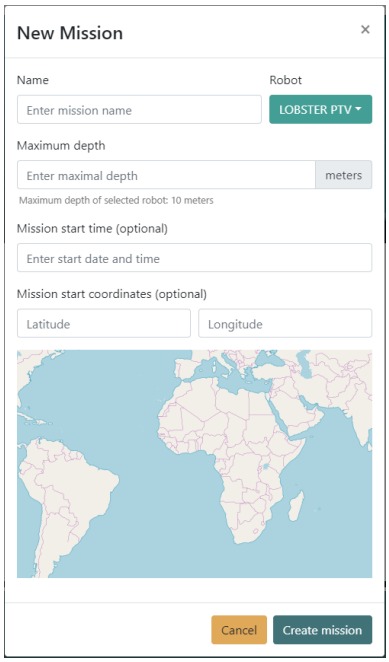


Figure 5.2: The new mission dialog

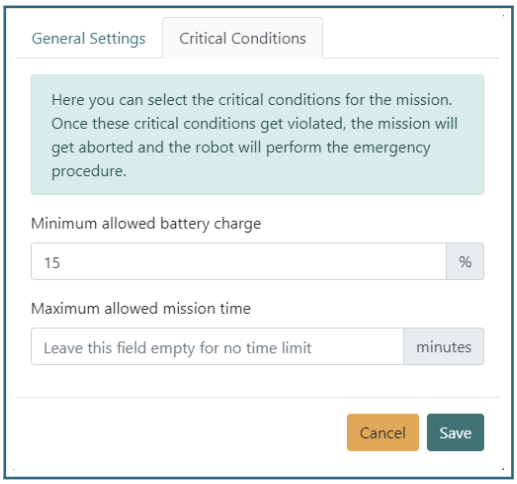


Figure 5.3: The critical conditions screen

Timeline

A timeline was added as one of the first features to allow the user to create and edit missions (Figure 5.1 No. 1). This timeline is a block-building feature that allows the user to sequence multiple actions together to create a mission. With this timeline, we gave the user a visual toolkit to create missions rather than the standard way of creating missions manually every time.

The timeline is the main feature of our application as our application would not work without it. All actions on our timeline are able to be dragged around, and the user has the ability to add more actions to the end or in between actions, or delete actions the user does not need.

Actions

Each of the actions a robot could take, think about moving, diving, rotating etc., has its own action block that can be added to the timeline (Figure 5.1 No. 2).

Each of these blocks have a few changeable features such as the action-specific settings, like a depth when diving (Figure 5.1 No. 3), but also general equipment settings such as enabling or disabling the camera (Figure 5.1 No. 4). Lastly there are payload-specific settings that depend on which robot the user is currently using to plan the mission, as each robot can have different payloads installed (Figure 5.1 No. 5).

Mission simulations and estimations

To help the user visualise how their created mission could play out, we decided to implement a multitude of ways to simulate and estimate possible behaviour. One of the first features we implemented was the estimation of actions. As each action is an individual process, they all use a specific amount of time and percentage of the robot's battery. Thus, for each action we estimate these and display them (Figure 5.1 No. 6).

At the top of our application we decided to add an info-bar that displayed the estimations of the entirety of the mission, but also an estimation of the time a mission could take if the robot were to shut down or run out of battery at the most unfortunate point of the mission (Figure 5.1 No. 7).

We continued to show more information to the user by simulations by adding two types of maps to our application: one that shows the user the depth of the mission over time (Figure 5.1 No. 8), and a top-down view of the mission to visualise the sideways movement (Figure 5.1 No. 9).

Mission management

To help users managing a mission, we implemented a multitude of small features to help the user gain control of all aspects of the mission.

As a start, we created a very in-depth mission creation dialog that asks the user all the necessary information needed for a mission to be created, such as a title, robot and a maximum mission depth. It also asks for optional settings, such as a mission date and location, which the user can input by either clicking on a real world map or manually inputting latitude and longitudes. (Figure 5.2)

Of course we do not expect all missions to be created instantly and thus gave the user the ability to save the current mission, accompanied with the ability to load any mission that was previously saved. (Figure 5.1 No. 10.) The missions get stored in the local storage of the browser.

Finally, we added critical conditions the user can manually input. When any of these critical conditions is met, the robot will start the emergency procedure. Examples of these critical conditions are a maximum allowed mission time, and a minimum battery percentage the robot is not allowed to exceed. (Figure 5.3)

Compatibility with Multiple Robots

Our application makes use of *robot templates* to allow it to easily be adapted to other robots. These templates allow robot owners or producers to easily specify the parameters of the robot such as its speed, battery usage and other features. This template can then easily be included into our application after which it can be used to plan missions for that robot.

We have specified the format of these robot template files using a JSON Schema [14]. This schema defines exactly which parameters this file should include and in what format. This schema can be found in I.

Sanity checks

We decided to also implement a fair amount of sanity checks, helping the user create a mission that brings back valuable data. Some of these sanity checks are very simple, whilst others verify more complex situ-

ations.

One of the more basic checks we added was the validation of all the small user inputs. An example of this is that when asking the user for a depth to dive to, only allowing positive numbers between the length of the vehicle and the maximum depth, as seen in Figure 5.4.

More complicated sanity checks occurred when we had to verify multiple inputs to be valid together. Due to the sea starting to rapidly lose sunlight below a certain amount of meters, we wanted to make sure that if the user wanted to use the camera below this point, a warning would appear if the user did not have any lights enabled to illuminate the recorded scene, shown in Figure 5.5.

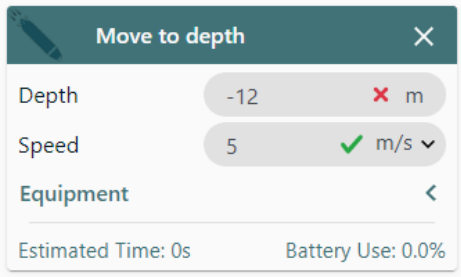


Figure 5.4: A simple sanity check for user input

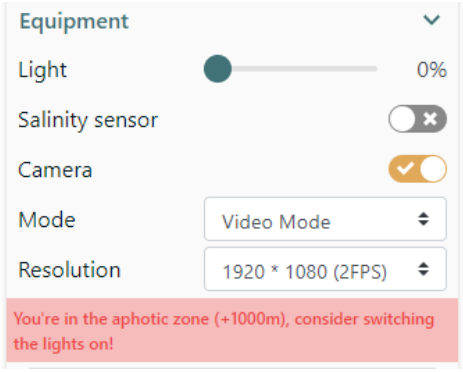


Figure 5.5: Warning when using the camera in the dark depth zones without lights.

We implemented sanity checks based on the value of the user, which would be valid, but possibly incorrect. One of these was a little compass that would show the heading the user inputted for the current action, as shown in Figure 5.6. This is not a check that would disallow the user to use these values, but rather for the user to check if the values they provided were correct.

Another sanity check that was not exactly related to an invalid input value, was the mission map, located in the new mission creation screen. This map was provided to allow the user to see the exact location of the inputted longitude and latitude on a map of the world, as can be seen in Figure 5.7. This allows the user to double check that the correct location has been set.

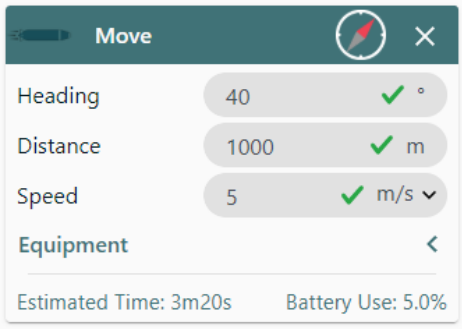


Figure 5.6: Small compass indicating the current heading

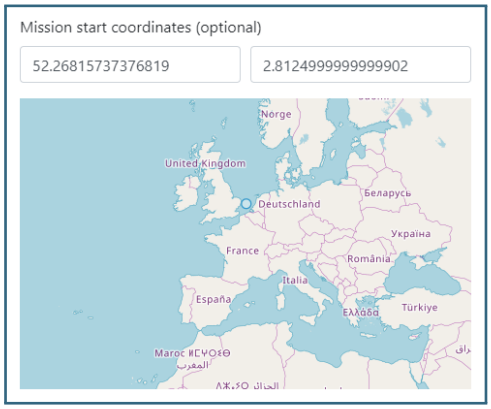


Figure 5.7: Map of the world in the new mission screen

5.2 Satisfaction of requirements

We implemented almost all requirements that we came up with in the research report (see Appendix D). We also added additional features that came up through user testing or the course of the project.

We implemented all must-haves (see Section F.1), except default mission templates, but we decided at first that this was an optional feature anyway. We are quite satisfied with the fact that we implemented all of these and most of them were already implemented in the starting weeks of development due to their priority and impact on the application as a whole.

Looking at our should-haves (see Section F.2), we did not have sufficient time to implement all of them. For example, we did not implement the ability to group actions up into phases, or allowing actions to be selected together and saved for later use. One feature we also did not implement from our should-haves was the ability to select actions for logging, but the reason we did not add this is because we argued that in reality all actions should be logged anyway. In total, we implemented 85% of the should-haves. Next to that, we also implemented many additional features we discovered through the user tests, which we found out to have higher priority than some of the should-haves.

Lastly our could-haves (see Section F.3), of which we only implemented the features we were able to implement quickly and easily. Of our pre-created could-haves, we only implemented 33% of the features, due to them being a lower priority.

All of our non-implemented features could be future work to further improve our application and make it more functional. None of these were essential to a working product though.

5.3 Interaction Design

To address the usability problems, our main goals were making the product intuitive and quick to use. We used a combination of usability heuristics and user tests to improve the usability of our application. In Chapter 6 we describe our approach to usability in detail.

5.4 Code Quality

During the development of our product we had to upload our code to Software Improvement Group (SIG) for a general quality check. This was done twice: First to guide us in the right direction, and the second time to see if we have improved.

Midterm Check

The first code quality review we received from SIG scored 3.6 stars on their scoring system, this meant that our code was market-average maintainable ("marktgemiddeld onderhoudbaar" in Dutch) and they pointed out that our module coupling could see some improvement.

Next to that, the software SIG uses had not been able to identify our test code. This was due to the fact that we use the TypeScript programming language with the React framework. We sent an e-mail to SIG

about this and they informed us that it was indeed a problem on their side, and that it will be fixed for the next review.

The reason they would like to see improvement in our module coupling was due to the fact that frequent calls to specific parts of code that could lead to the code being less maintainable. We agreed on this to some extent, although SIG themselves also acknowledges that in some cases coupling is inevitable.

Improvements

Based on this feedback from SIG, we improved our code to reduce this coupling. We split functionality from the most coupled classes (such as `AbstractAction`) into multiple different classes. This way code in these classes can be changed more easily without breaking other parts of the product.

We submitted our code for the final code check of SIG in week 9. The final feedback of the SIG was quite positive. They observed that while the code volume had grown, the maintainability had also increased. The module coupling has been significantly and structurally improved. Finally, they concluded that we had largely taken the recommendations from the previous feedback into account in the development process (in Dutch: "Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie grotendeels zijn meegenomen in het ontwikkeltraject").

5.5 Automated Testing

We have made extensive use of automated testing to ensure the maintainability and correctness of our product. We used a mix of various testing techniques to test the various parts of our product. We used unit tests to verify the logic of our application in various edge cases and to ensure that accidental changes would not lead to incorrect calculations. Additionally, we used integration tests to check that the various components worked correctly together.

For these integration tests we also made use of the Enzyme [2] framework to test the UI components in detail. This allowed us to create tests which encapsulate the steps an actual user might take. For example, we have a test which creates a new mission, fills in the various parameters, adds some actions and checks that the expected mission is exported. This entire test is written with simulated button clicks and was able to show us if anything in this sequence broke unintentionally.

Since the first week after the research phase we consistently held a branch test coverage of at least 85%, as can be seen in Figure 5.8. In our final product the branch test coverage is 90% of the entire code base. This means that of all decisions that are made in the program (for example an "if statement") 90% of the possible choices are tested.

These were valuable throughout the process because they prevented various types of bugs from entering our main branch, because the tests were automatically run before any code could be accepted. Whenever we found a bug, we would also write a new tests to prevent it from ever re-occurring.

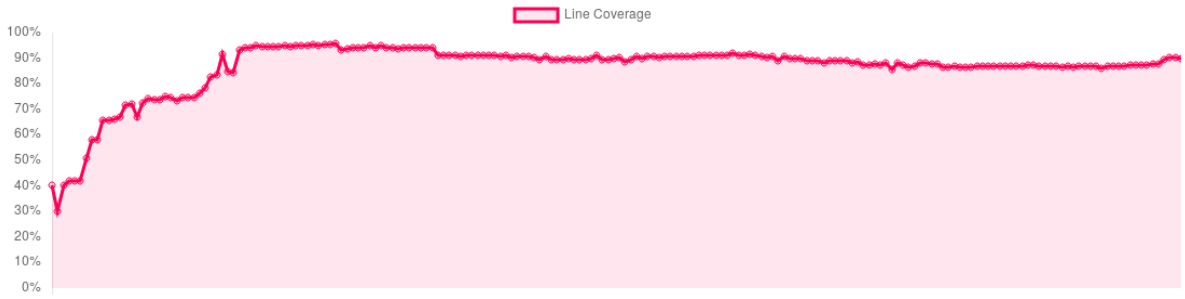


Figure 5.8: The code coverage of our automated tests over the course of our project.

Chapter 6

Usability

The most important goal of any user interface is that it can be effectively used by the target users, and this is no different for our project. Our goal is to create a tool which can effectively be used to plan a mission for an AUV, and for this to be achieved our product has to be usable. In this chapter, we will discuss the usability of our product and how we measured and improved it.

In the international standard ISO 9241-210 [1], and for our purposes, usability is defined using three characteristics: effectiveness, efficiency and satisfaction. Effectiveness means the degree to which users can achieve their goals accurately, which in our case means planning a mission. The efficiency aspect is defined by the amount of effort required by users to achieve the goals. Finally, the satisfaction is defined as "freedom from discomfort and positive attitudes towards the use of the product". [1]

A further useful distinction that can be made within this concept of usability is between two different levels: the task and communication level. The task level refers to the functionalities of the system and which information is presented during what part of the process. On top of that, you have the communication level which refers to the way these functionalities and information are presented, this can also be called the "look-and-feel" aspect. [11]

Throughout our project, we aimed to design and build our product to improve the usability on both of these levels. We achieved this through two main approaches: heuristics and user tests. We will explain how we used both in the rest of this chapter.

6.1 Heuristics

Heuristics are an efficient way to develop for usability from the very beginning of the product and they provide ways to find problems and solutions without having to set up user tests. There are multiple useful sets of usability heuristics available, but in the end most describe the same set of underlying principles. In this section we will discuss how one of these sets of heuristics, Eight Golden Rules of Interface Design as described by Shneiderman and Plaisant [15], applies to the usability of our product:

1. Strive for Consistency

We strove for consistency both at the task and the communication level. For example, on the task level we made sure that all modal dialogues, such as the new mission dialog, the load mission dialog and the critical conditions dialog, provided the same options for cancelling or accepting the changes.

On the communication level we also used consistent styles and colours throughout the interface, like making sure all cancel buttons have the same colour. Furthermore, we ensured consistency with other common standards, for example by using default icons for saving and loading.

2. Seek Universal Usability

This principle states that an interface should be usable both by experts and novices. To facilitate the novice users we added explanations of terms throughout the interface. However, to prevent clutter hindering the expert users we made most of these explanations visible only when the user hovers the mouse over the item. In this way, the novices can receive explanations when they need it, but the experts are not overloaded with irrelevant information.

3. Offer Informative Feedback

Giving feedback on the mission plan the user is working on is something we found very important. We implemented various ways in which information is given about the current mission, for example through the battery estimations and the visualisations of the mission. All of these are updated each time anything about the mission is changed to give immediate feedback.

4. Design Dialogues to Yield Closure

It means that the process which the user goes through has a clear start, middle and end. This helps the user understand where they are in the process. To achieve this, we implemented a separate new mission dialog for the beginning of the process. The middle stage is then visually signified by this dialog being resolved. Finally, the end is achieved when the user is shown that the mission has been exported.

5. Prevent Errors

We designed our entire interface to prevent errors while planning the mission as much as possible. For example, it should be impossible to add a resurface action when the robot is already at the surface. Furthermore, the interface does not allow the user to input invalid values for the various parameters of actions, such as depth and distance.

6. Permit Easy Reversal of Actions

We supported easy reversal in changing settings by providing a "cancel" button which would reset the settings to the previous values. However, a possible future improvement on this front is the addition of an "undo" button which can reverse any changes to the actions or settings.

7. Keep Users in Control

This rule basically states that the users should be able to achieve their goals in their preferred way. Therefore, we attempted to keep our interface open to various approaches to the mission planning problem. If someone wants to start adding actions at the end of the mission, then that is possible, but it is also possible to add actions at the beginning or in between actions.

8. Reduce Short-term Memory Load

We tried to reduce the short-term memory load by keeping all relevant information visible within the same main screen. Additionally, the visualisations help give a summary of the current mission so that the progress can easily be reviewed. Because all this information is always available, the user needs to remember less information on their own.

6.2 Tests

The second tool we used to improve the usability of our product was usability testing, which is the testing of our User Interface (UI) with actual users to evaluate its usability. This lead to more detailed feedback, but it required more effort to organise compared to using heuristics.

We have used usability testing throughout the project to improve and measure our usability. We conducted two different types of user tests during our project. The first type we used is formative tests, which are aimed at finding defects and possible improvements in our interface. The second type is summative tests, which are aimed at evaluating the usability of our interface and finding which problems still exist.

We used formative tests to find new improvements, primarily during the middle stage of the project. In contrast, the summative usability study was done in the last two weeks of the project to evaluate the usability of our final product. In the rest of this section, you will find more information about both these types of tests and what the results of those were.

6.2.1 Formative User Tests

Starting in week 5, we performed our first formative user tests with the first prototype aimed at finding new improvements. We chose to start user testing in this week because at this point we had a reasonable basic product which could actually be used to perform some tasks.

Since these user tests focused on finding improvements, and did not aim to evaluate the usability formally, we tried to keep them open and broad. We asked the users to perform some set tasks and looked at what went wrong and what went right. Furthermore, we asked some questions to better understand the user needs for visualisations of the mission.

The exact details of how we handled these formative user tests are left out of this section to ensure briefness. If you want to know more about how exactly we handled the user tests, which tasks we asked the participants to attempt and what questions we asked, you can read Appendix C, which explains this in detail.

Results

First of all, an important result was that all participants were able to complete all tasks within a few minutes, although it did sometimes take some looking around to find the correct button or information. Another significant positive result was that the participants found it very intuitive to drag the actions, which showed that this mechanism was sufficiently usable.

However, we also found some issues: For example, one participant expected the critical conditions to be

reachable by clicking on the information bar at the top of the application, which was not possible. Since we found this a reasonable expectation, we added this feature in the following week. Another deficiency we found was that users did not find the action addition mechanism very intuitive. They expected the "+" buttons next to the actions to allow them to add new actions directly (eg. through a drop-down menu), instead of having to use the actions bar on the right. We also addressed this in the following weeks.

Countless other minor improvements were identified during the tests, from obscure word choices in the interface to the absence of an icon for the new mission button. In fact, we found so many possible improvements that we had to re-prioritise our issues. You can read more about this in Section 4.5.1

Furthermore, we got a lot of interesting feedback on the possible visualisations from the answers to the questions in Section C.1, which we will summarise. We got some feedback that the depth over time visualisation was confusing at first glance. Some participants stated that it was not clear what the visualisation represented when they first looked at it. Based on this feedback we added more context clues to the visualisation, such as a title and a ship icon at the start of the line.

On the possibility of displaying the battery percentage left during the mission using a colour gradient, all participants agreed that it would be a nice to have, but not a vital feature. Therefore, we decided not to implement this as it would require a lot of work and we had more important improvements in mind.

All participants immediately found it clear what the interruption in the line of Figure C.2 was representing. One noted that the axis of Figure C.3 were not clear due to no labels being present.

6.2.2 Final Summative Study

In the last two weeks of the study, we performed a summative user project, which was aimed at evaluating the overall usability, and strengths and weaknesses of our product. To evaluate the usability in this study, we asked eight participants to complete a set of tasks. These participants all had a background in underwater robots, either because they were part of the LOBSTER team or because they worked with underwater robots at their job. We noted down what went wrong and what went right. Finally, we asked each of the participants to fill out the System Usability Scale [6] and asked some open questions. More detail on the exact plan and procedure for these tests can be found in Appendix C.

Results

During the study we asked each of the users to fill out the System Usability Scale [6] questionnaire, which resulted in an average SUS score in the range of 0-100. On average, the participants in our study resulted in a score of 78.13. This score is generally assigned a letter grade of B, and were quite close to receiving an A. Additionally, this score corresponds to right in between Good and Excellent on an adjective scale (where Good is about 71 and Excellent about 85). [5]

We also asked every participant the following three open questions:

- Can you name three things you liked about the product?
- Can you name three things you disliked about the product?
- Do you have any ideas for future improvements or additions to the product?

We categorised and clustered all the answers, giving us the following results. The most liked aspects of the program were:

Participant	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Score
1	1	4	3	3	2	3	3	3	3	4	72.5
2	3	1	1	4	2	3	3	2	1	4	60
3	3	3	3	4	3	4	3	4	4	3	85
4	4	3	3	3	3	3	4	4	3	3	82.5
5	4	4	3	3	3	4	4	0	4	2	77.5
6	1	4	2	4	2	3	3	2	3	3	67.5
7	3	4	3	4	4	3	4	4	4	4	92.5
8	3	4	4	4	3	3	4	4	3	3	87.5
Average											78.13

Table 6.1: Table representing the results of the SUS

- helpful info on the screen such as sanity checks, battery info, total mission time, etc.
- visualisations of the mission, e.g. the depth over time map and top view map
- swapping of actions by dragging

This again shows the usability of all the sanity checks, as well as the visualisations. They really help the user in making the mission they want. The most disliked aspects were:

- unclear visualisation details, like axis titles, colours used, etc.
- no branching or conditional actions
- the information bar in the top does not display why it is giving a warning

The most requested future improvement was more complex actions, such as moving in an area using a set pattern, or grouping actions together into one action for re-usability.

- moving in an area using a pre-configured pattern
- move until a sensor reaches a pre-configured value (for example, move until the salt level is 3%)
- start filming when the camera sees something interesting

Other suggested future improvements included:

- copying and pasting actions
- branching or conditional actions
- show the seabed or depth levels in the visualisations

6.3 Researchers

In the last week, we showed our product to two researchers of NIOZ who often work with AUVs and ROVs. We wanted to know their opinion since they will eventually be the end-users of our product. They thought our product would be very usable for their purposes, if it would be possible to add an altimeter or Doppler Velocity Log (DVL). Luckily, with the way robot templates work, it is very easy to add different sensors to the robot.

6.4 Conclusions

Overall we got a lot of useful information out of the user tests, both to guide it in new directions, and to verify our solution. The System Usability Scale score of 78.13 shows that our product is easy to use, and proves the validity of our solution. We think this is a splendid score for a first prototype.

The open questions of our user tests provided some nice insights and helped us to see what can be done regarding future work (see Chapter 8). We already had most of the negative comments in mind for future work on the product, but the usability study strengthens our case for these recommendations. We describe these possible future improvements in more detail in Chapter 8.

Chapter 7

Conclusion

We have built a working prototype to successfully solve the problem stated in Chapter 2. We started by researching the problem and existing solutions, based on which we constructed our initial requirements in collaboration with the client.

We added sanity checks to give the user more feedback on the current mission and detect mistakes, as discussed in Chapter 5. Throughout the implementation phase we stayed in contact with the various stakeholders to adjust our goals to their needs.

Another distinguishing feature of our product is its usability. From the start of the project we aimed to make the application easy to use. We did this by working on usability at both the task and communication level. [11] At the task level we implemented various sanity checks to provide the user with dynamic feedback on the current mission to prevent mistakes. At the communication level we focused on using colours and layout to ensure that the most important information is noticed by the user at the correct time.

Starting from the fifth week of the project we performed formative user tests to further guide the development of our user interface. These proved immensely valuable to discover new possibilities and opportunities. Finally, we performed a final formative usability study in the last weeks of the project. As discussed in Chapter 6, we can conclude that our final product is successfully usable to create missions for Autonomous Underwater Vehicles from this study.

We also evaluated our project in comparison to the the initial requirements we formulated in the research phase of the product in Chapter 5. We were able to implement all must-haves and most should-haves. Additionally, we have implemented several features that we initially did not set as required, such as the ability to drag actions and showing a world map in the new mission dialog.

We think the usability study and the requirements analysis together show that overall our project was a success. We succeeded in solving the problem as specified in Chapter 2 and in the process we improved our technical and project management skills. We have also researched what future steps could be taken to improve the product even further and we will discuss these in the next and final chapter.

Chapter 8

Future Work

Although the product is already usable to create missions, we have thought of various possible improvements that could be made to improve the application. This chapter we will discuss the features that could be implemented in the future. There are a lot of features to talk about but we will only talk about the most important ones. Some of these features were briefly touched upon in Section 5.1, but we will go into more detail in this chapter.

Selection Saving & Phasing

The first one that we did not implement, was the ability of saving and phasing selections. This feature allows the user to select an arbitrary amount of neighbouring actions. This selection can then either be identified as a phase, and thus giving these actions some identifying feature like a different colour or a surrounding square. The user could also save this selection of actions, adding it as a single button to the list of addable actions. Clicking on this button will then add the entire selection to the timeline wherever the user wants to put it.

In the final usability tests it became clear that this feature is one of the most requested features by prospective users. They feel that this feature could save them a lot of time by reducing the need to add repetitive actions. Therefore, we would recommend this feature to be one of the main priorities to implement in the future.

Estimated Location Deviation

The current top view visualisation we implemented in our application heavily relied on the accuracy of our estimations, and assumed that the robot always perfectly knew where it was. In reality this is not possible, as currents can affect the sideways location of the robot, as well as making the robot over- or undershoot the desired distance travelled. To allow the user to visualise this, we thought of a way to display the possible deviation of the robot, which increases the longer it is submerged, as can be seen in Figure 8.1. The highlighted area shows the location the robot can be in at that point.

Actions Based on Payload

Some robots have a payload section that can house different equipment for different missions. An example of such a payload is a specialised sensor like a salinity sensor. A feature we wanted to implement was the ability to allow actions to start or end when for example a certain value of a payload sensor was reached, like diving until the salinity of the water is 25 ppt (grams of salts per kilogram of water).

This, however, did bring some possible problems to the table, one problematic scenario being an action that was set to stop when a certain sensor reading corresponds to some number, and if the sensor were to be faulty, or the value that was inputted was incorrect, it would never actually reach the end of this action.

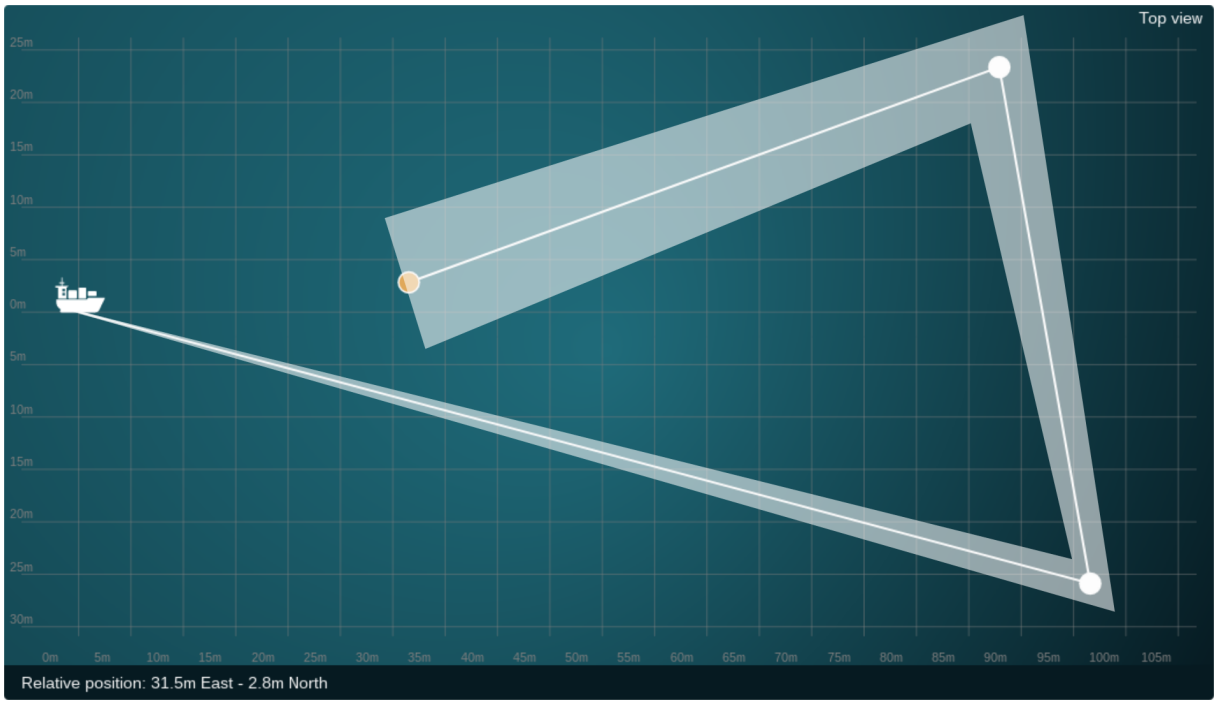


Figure 8.1: Mock-up of showing the estimated deviation on the top map

Conditional Branching

Another one of the bigger features we originally wanted to implement was conditional branching. This feature allowed the user to act on the possible different outcomes of an action. As mentioned above, we wanted some actions to act on the state of the payload, thus a possible conditional branching example is to execute action A if the state is reached, and otherwise move to action B if it was not reached, as can be seen in Figure 8.2.

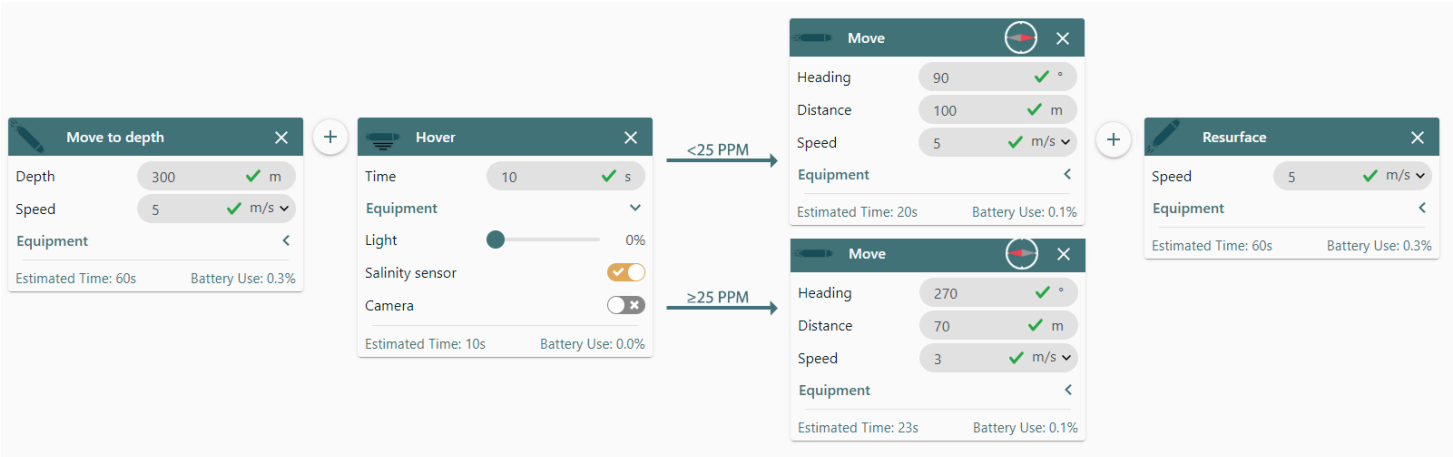


Figure 8.2: Possible visualisation of conditional branching

Additionally, we wanted to add branching in a state of emergency. If the robot was performing an action, and something went wrong, then instead of completely shutting down and cancelling the mission, some

other action could be performed that would fix or counteract the problem that was discovered.

More External Data

Our application in the current state allows the user to receive information on the conditions of the sea at the time and location the user specified. This data includes the severity and direction of currents, waves and wind. Our application also allows the user to manually set the maximum depth of the mission.

One of the features we wanted to add was more external data, for example data on the depth of the seabed at the user specified location, and feeding this back to the user to prevent them from setting a maximum depth that was below the seabed.

Another data-point that we thought of was the values of the earth's magnetic field at the starting location of the mission, allowing the user to see information that they can use to calibrate their magnetometer.

Custom Robot Uploading

Currently, the application only works with the two robots of the LOBSTER team, however, if this product were to be put on the market for other companies to work with, it would not be of much use if their robot was not exactly the same.

Consequently, one of the features we wanted to add, was to allow the user to create a completely new robot profile, of which the user can add the values and other information themselves. This robot profile can then be saved locally or to some online database. The reason this was thought of was the fact that some companies have their own custom robots, whilst others use robots that are available to buy by everyone. Already having a robot profile of the exact same robot that was set by someone else would be a useful and time-saving feature.

Direct Uploading to Robot

The last bigger feature was the ability to directly upload the created mission to the robot. Currently, our application exports an easy to read and specified JSON file in which all information required for the robot to execute the mission is located. An easier and faster way would be, instead of using another program to convert this file into a file the robot directly can read, to integrate this into our application, rolling out a ready to use instructional file.

This can then be further improved by allowing the robot to be connected to the device on which our application is running, either with our without a wire. This can be followed by, instead of downloading a file, directly uploading it to the robot.

Glossary

API	Application Programming Interface	24
AUV	Autonomous Underwater Vehicle	2, 6, 8, 10, 19, 21, 37, 41, 43, 52
AUV's	Autonomous Underwater Vehicles	8, 9
Branch	This is a mechanism through which developers can maintain multiple versions of the software artefact in a Git repository. Each branch represents a different version of the product	22, 35, 47, 52
DVL	Doppler Velocity Log	41
Git repository	This is a tool to store the development history of a software artefact. It allows us to store all previous versions and variants of the product and be able to review them	22, 47, 52
LOBSTER	This is a student startup team in Delft which is developing a low-cost autonomous deep-sea robot. The goal of this robot is to dive into the deep sea and return safely. They currently have two robot prototypes: the LOBSTER Explorer, which will actually be able to go to the deep sea, and the LOBSTER PTV, which is used for testing the software in shallow waters. Our product is primarily aimed at creating a mission planner for these two robots at least	6–8, 10, 11, 13, 17, 21, 25, 26, 46, 52, 53, 57, 58, 60–63, 68
NIOZ	Royal Netherlands Institute for Sea Research	26, 41
Pull request	This is a request to add code changes to another branch in a Git repository. It allows team members to review the changes before they are added	22
ROV	Remotely Operated underwater Vehicle	9, 41
SUS	System Usability Scale	40, 42, 59
Test coverage	This is a metric that counts the percentage of code which is covered by the automatic tests. Test coverage is commonly used in software development to track how well-tested a software artefact is, but good test coverage on its own does not guarantee useful tests	22, 35
UI	User Interface	9, 13, 15–18, 35, 39, 63, 65, 67, 68

Bibliography

[1] ISO 9241-210:2010. *Human-centred design for interactive systems*. Standard. Geneva, CH: International Organization for Standardization, Mar. 2010.

[2] Airbnb. *Enzyme*. 2019. URL: `https://airbnb.io/enzyme/` (visited on 06/25/2019).

[3] ArduPilot. *ArduPilot Open Source Autopilot*. 2019. URL: `http://ardupilot.org/` (visited on 05/28/2019).

[4] Atlassian. *Scrum - what it is, how it works, and why it's awesome*. 2019. URL: `https://www.atlassian.com/agile/scrum` (visited on 05/28/2019).

[5] Aaron Bangor, Philip Kortum, and James Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale". In: *Journal of usability studies* 4.3 (2009), pp. 114–123.

[6] John Brooke et al. "SUS-A quick and dirty usability scale". In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[7] Ifremer. *MIMOSA - Ifremer Fleet*. 2019. URL: `http://flotte.ifremer.fr/fleet/Presentation-of-the-fleet/Logiciels-embarques/MIMOSA`.

[8] Triton Imaging. *Triton Software Compatability*. 2019. URL: `http://www.tritonimaginginc.com/site/content/software/compatibility/oldindex.htm` (visited on 05/28/2019).

[9] LEGO. *About EV3 - Mindstorms LEGO.com*. 2019. URL: `https://www.lego.com/en-us/mindstorms/about-ev3` (visited on 05/28/2019).

[10] NASA. *Open MCT - Open Source Mission Control Software — Open MCT*. 2019. URL: `https://nasa.github.io/openmct/` (visited on 05/28/2019).

[11] Mark A Neerincx. "Situated cognitive engineering for crew support in space". In: *Personal and Ubiquitous Computing* 15.5 (2011), pp. 445–456.

[12] OceanServer. *Vectormap mission planning - I3 oceanserver - autonomous underwater vehicles*. 2019. URL: `https://ocean-server.com/vectormap/` (visited on 05/28/2019).

[13] P Oliveira et al. "Mission control of the MARIUS autonomous underwater vehicle: system design, implementation and sea trials". In: *International journal of systems science* 29.10 (1998), pp. 1065–1080.

[14] Felipe Pezoa et al. "Foundations of JSON schema". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 263–273.

[15] Ben Shneiderman and Catherine Plaisant. *Designing the user interface: strategies for effective human-computer interaction*. 6th ed. Pearson Education India, 2016.

[16] UgCS. *Ground Station Software | UgCS PC Mission Planning*. 2019. URL: `https://www.ugcs.com/` (visited on 05/28/2019).

Appendix A

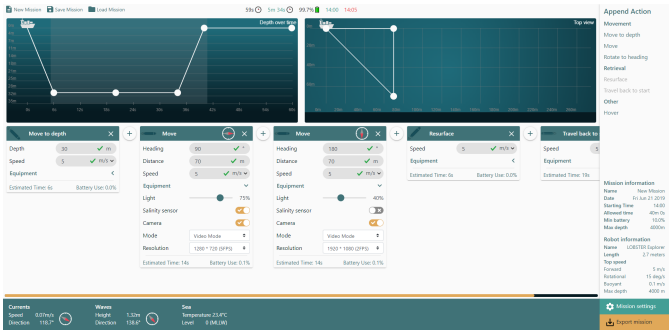
Infosheet

On the next page you will find the infosheet of our project, which provides a short summary on one page.

Mission Planning for Deep Sea Robot
Client organization: **Allseas**
Presentation on the **2nd of July, 2019**

Our challenge was to define a format for a mission plan, and develop an application which allows the users of the robot to create a mission plan effectively. This application should assist the user in the process of creating a mission plan as much as possible. Our client was Allseas who was already the sponsor of the LOBSTER project.

During the research phase we learned that our application would be best suited for an offline environment, as the user would not necessarily have an internet connection due to being at sea. Furthermore, we learned that the application had to be easy to use and not too complex.



A possible mission on our application

Throughout the course of this project we worked with Scrum, our repository was hosted on GitHub, with Travis CI integrated to test our code. Using this we could see our coverage and errors could not enter our final product. In our team we cooperated with an open culture to prevent frustrations. We made agreements on work hours and software methodologies. Still, we faced challenges which ranged from underestimating time needed for certain features to an increasing complexity of code and data gathering.

The final product is a web application which can be accessed offline. This application can be successfully used to create missions for Autonomous Underwater Vehicles, which we showed in our final usability evaluation. The LOBSTER team will be using our product to plan their missions in the future. However, before it can be fully used, the low-level control software of the LOBSTER robots needs to be finished by the LOBSTER team.

- **Gijs Koning:** Interested in Software Engineering, Robotics
  Main contributions: Project Management, Code Quality, Estimation Algorithms
- **Thijmen Langendam:** Interested in UI Design, Back End Development, Mathematical Logic
  Main contributions: Interface Design, Sanity Checks and External Data Sources
- **Dennis Mouwen:** Interested in Software Engineering, UX/UI Design
  Main contributions: Visualisation Tools, Interface Architecture and Output Schema
- **Jochem Raat:** Interested in Programming Languages and Computation Theory
  Main contributions: Input Forms, Visualisation Tools, Usability Testing

All team members contributed to front end development and the final report.

Client: **Jeroen Breukels, Allseas**
Coach: **Mark Neerincx, Delft University of Technology**
Contact person: **Gijs Koning,** `gijs-koning@live.nl`

The final report for this project can be found at: `http://repository.tudelft.nl`

Appendix B

Original Project Proposal

Do you want to be a part of a project that creates an autonomous deep sea exploration robot?

Allseas is developing a robot that operates under high pressure, around four kilometres deep in the ocean. The robot needs to navigate to certain locations on the bottom of the ocean and do it safely. The robot is already created but there are still a lot of software challenges to overcome. One of these is that a customer needs to be able to create a mission for the robot through a User Interface and the robot needs to execute this mission. The user will be able to define a path within the capabilities of the robot. For example if the robot needs to go really fast it also drains its battery faster and thus the mission duration becomes shorter. At the end of the project the group is able to execute their code in real life on the robot.

Objectives:

- Build a User Interface where a user can create and edit a mission
- Create a high level mission control
- Integrate the mission control with our software
- The mission should be logged and monitored for errors
- The user should be able to setup an emergency plan
- The mission planner should be able to handle different robot configurations

Appendix C

Usability Testing Procedures

In this appendix we will explain the exact details of how we conducted our various user tests. These are kept out of the main body of the text in the interest of briefness, but we do think that it is interesting to read about this here for those who want to take a closer look at how we arrived at our results. More background information on what goals we had in mind for these tests and the results from the tests can be found in Chapter 6

C.1 Formative tests

For these user tests we worked with members of the LOBSTER team who were not part of our software project. We chose them as participants because they are part of the group who will eventually be using our product to develop actual missions for the LOBSTER Explorer. Furthermore, they work in the same building and are therefore easily accessible for us to conduct user tests with.

For the user tests we used a slightly modified version of our application. For example, it makes sure that the application does not start with a default mission set, but instead requires a mission to be created at the start. In our Git repository we made the `usability-study` branch for this purpose, which contains these changes.

C.1.1 Test Procedure

To test our product with each of these users we used a standard procedure, which specifies what tasks we ask the participant to attempt, and what data we record from the test. In particular, we used the following procedure for each participant:

1. Before the user arrives, we prepare the computer with the correct software and ensure that it is set up for all the tests. This includes resetting the program and saving a sample mission called "Underwater Explorer".
2. We thank the participant for helping us test our product and make it clear that we are interested in any feedback they have on our product. We also explain that we would like them to try to vocalize their thoughts while they are using our product, so that we can understand where things go right or wrong.
3. We give the following short introduction about the context of our product:

   > *"We have been working on a Mission Planning application and we look forward to discovering your experience with it. This application is meant to be used to design mission plans for Autonomous Underwater Vehicles (AUVs) and specifically for the AUVs developed as part of the LOBSTER project. This AUV does not know its exact latitude and longitude while underwater and therefore navigation is based on estimations of its location.*

4. We ask them to perform each of the tasks listed in Section C.1.2 one by one. For each of these we record whether they succeeded in the end or not and any relevant problems or comments on how the process went.
5. After all tasks have been attempted we will ask some questions to find out more about the users view on possible additional visualisations,:
   (a) Do you think the visualisation tools that are in the product were helpful and/or clear?
   (b) Do you think it would be helpful if these also displayed the battery percentage left at each point by color, as demonstrated in Figure C.1?
   (c) Do you think it is clear what the visualisations in Figures C.2 and C.3 are representing?
   (d) Can you think of any other visualisations that you think would be helpful?

C.1.2 User Tasks

These are the tasks we asked the particpants to complete:

Creating a new mission

Create a new mission for the LOBSTER Explorer robot, for a region where the sea floor is at 1000 meters, and add a fitting date and location to this mission.

Planning your mission

Plan the mission such that the robot will do the following things:

1. Dive to a depth of 500m. While the lights are at half power
2. Move 100m south
3. Move 250m east
4. Resurface such that the robot can be retrieved

Once you have done this, please answer the following questions based on the information available to you on the application:

- How long do you think this mission will take if everything goes according to plan?
- How much battery do you think will be left at the end?
- *While pointing at a random point of interest in one of the two map visualizations:* What action is this point related to?

Enabling the camera

Modify the mission plan you've made so that the robot starts filming at the highest possible frame-rate while it is moving east.

Editing critical conditions

Make sure that the robot starts the emergency resurfacing procedure only when the battery percentage left is lower than 5%.

Saving and loading

Save the current mission and then load the "Underwater Explorer" mission.

Exporting

Export your current mission in a format which can be uploaded onto the robot.

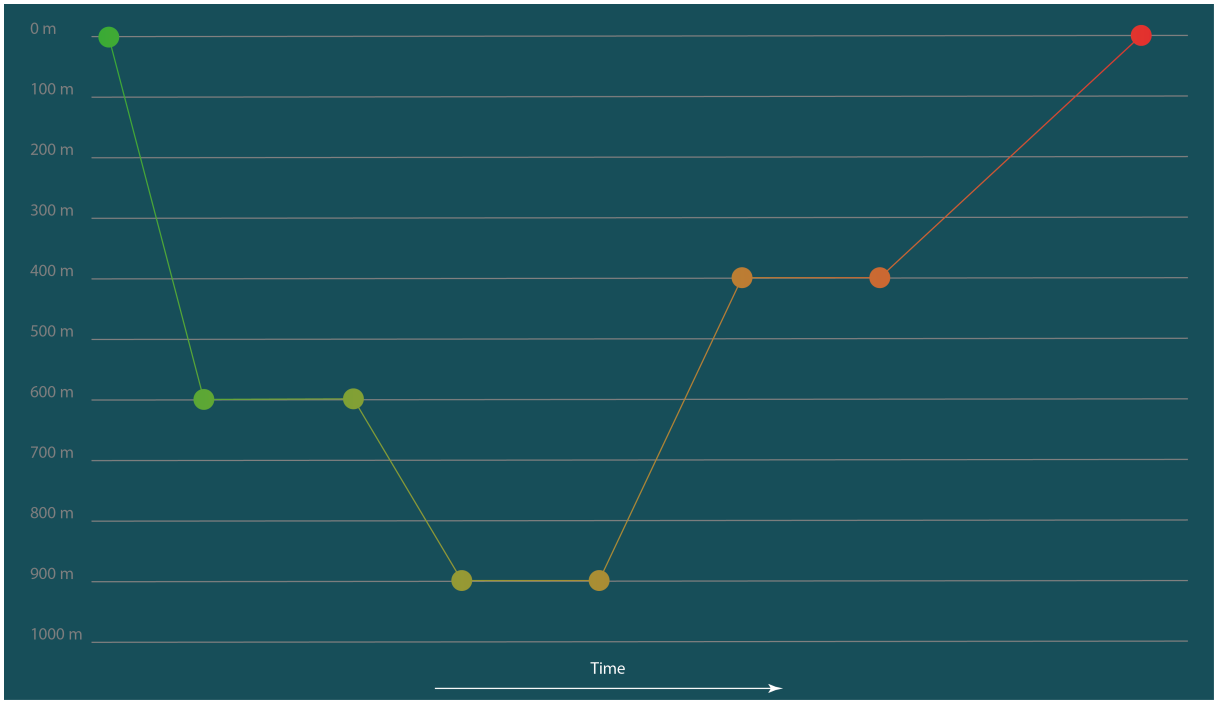


Figure C.1: Mockup of a depth timeline which visualizes the estimated battery left as well

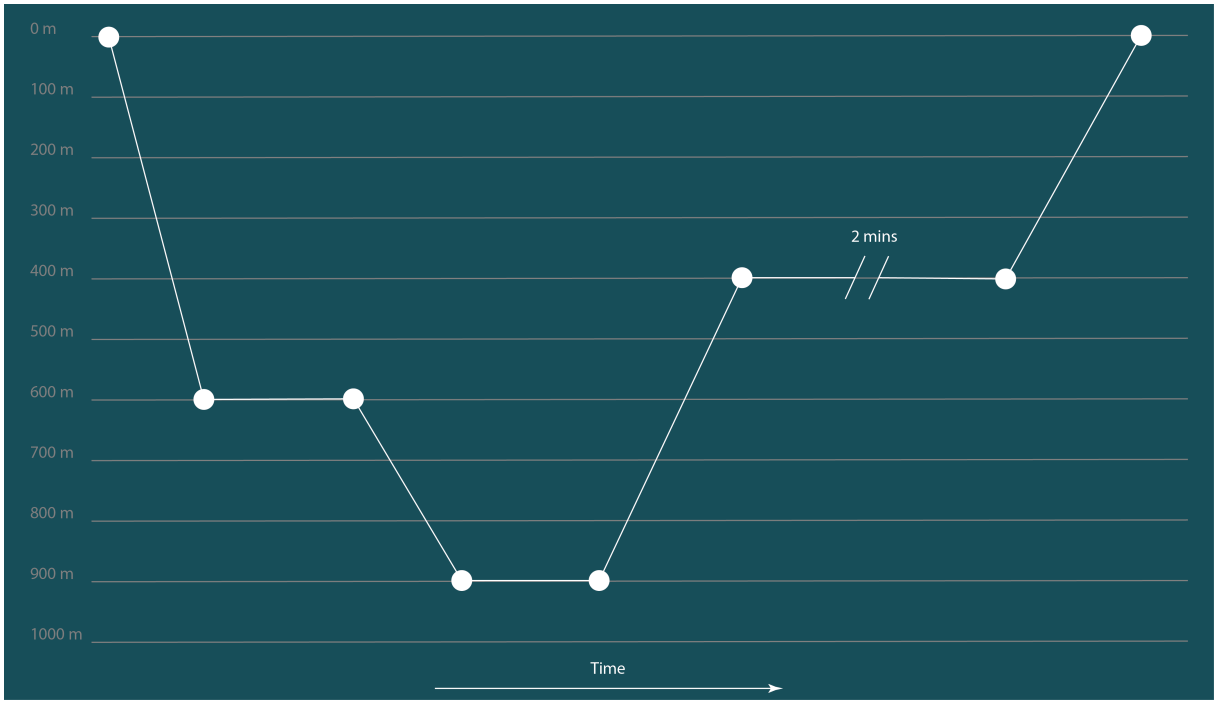


Figure C.2: Mockup of a depth timeline which includes a gap.

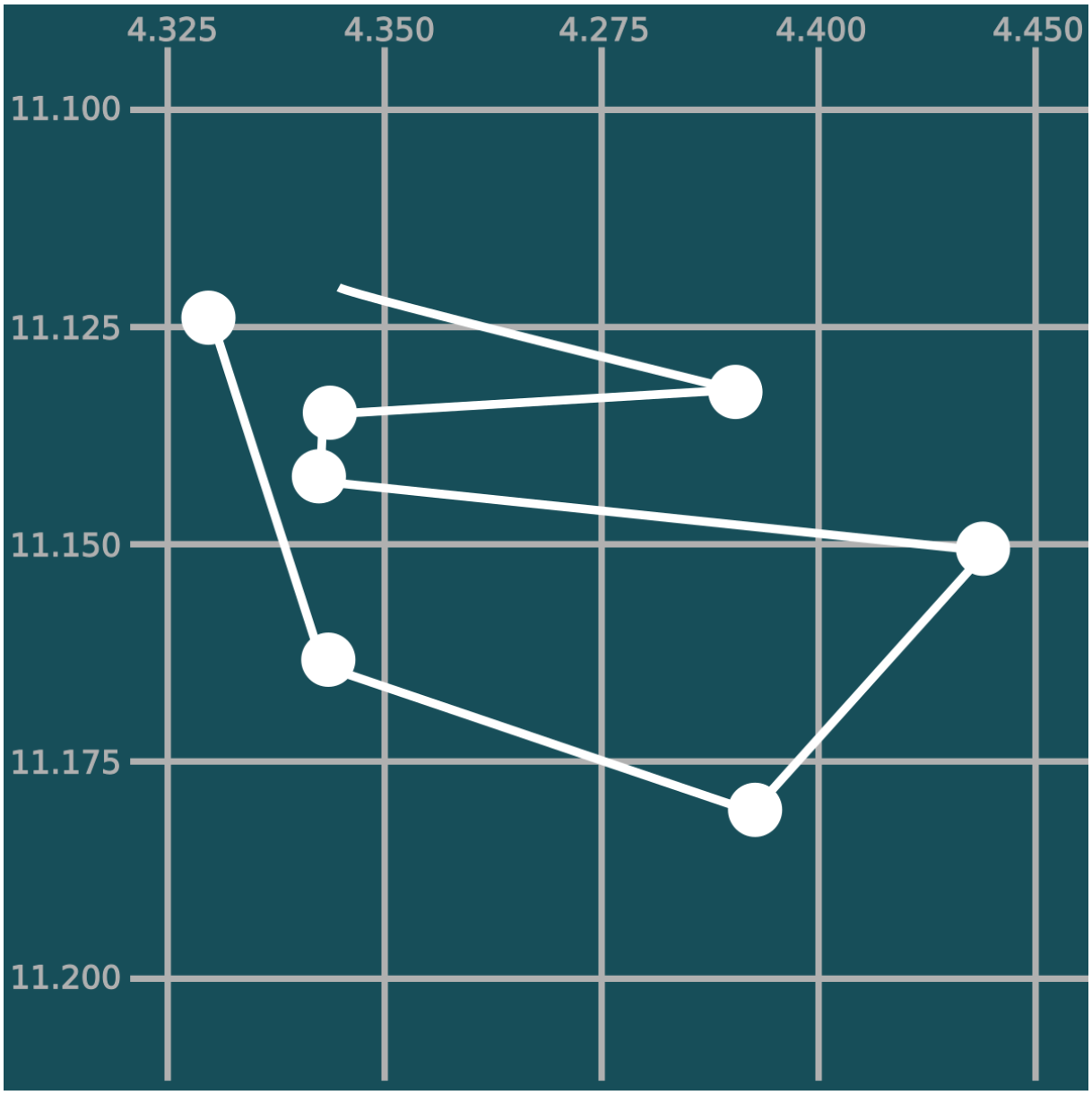


Figure C.3: Mockup of a map visualising the movement of the robot.

C.2 Final Summative Study

This section contains the details on the summative final usability study we conducted during the last two weeks of our project. In this section we will mostly be focusing on the procedural aspects of the study. For background on the purpose and goals of the usability study you can read more in Chapter 6.

C.2.1 Ethics Checklist

To ensure that our study was suitable to try on human subjects without any risks, we filled out the ethics checklist for minimal risk studies (see Appendix K. We were able to answer all questions with "No", so our study could be assessed as minimal risk.

C.2.2 Subjects

We aim to perform our study on between 6 and 10 subjects. These subjects will all have experience working with underwater robots. For example, some of them will have experience working in the LOBSTER team while others will have experience working with underwater robots at their workplace.

C.2.3 Test Procedure

The procedure for the the summative study is largely the same as for the formative tests. However, there are some differences in the preparation and in the metrics we collect. The exact procedure per user for this study is described below:

1. We reset the application to the beginning state and enable the screen recording software. This screen recording can then later be retrieved to analyse exactly how the interface was used. Furthermore, we make sure that the correct missions are already saved which will be used in the tasks, as described in Section C.2.4.
2. We thank the participant for helping us test our product and make it clear that we are interested in any feedback they have on our product. We also explain that we would like them to try to vocalise their thoughts while they are using our product, so that we can understand where things go right or wrong.
3. We give a short introduction to the context of our product, the full text of which can be found in Section 6.2.1.
4. We ask the participant to perform the tasks listed in Section C.2.5 one by one in the order that they are listed. For each of these we record whether the participant was able to complete it and how long it took. Furthermore, we record any problems or frustrations and also any positive comments by the participants.
5. Finally, we ask the participants some questions about their experience using our product. These questions can be found in Section C.2.7.

C.2.4 Prepared Missions

We prepare a single mission before each user test:

North Sea Survey

This mission simulates a survey in the North Sea. It consists of diving to a depth of 50 meters and moving around while recording. Then the robot moves 10km north, then 10km east, then 10km south, and finally 10km west and where robot resurfaces.

C.2.5 User Tasks

These are the tasks we asked the users to perform.

Mediterranean Sea Survey

You are tasked with finding a shipwreck in the Mediterranean Sea. You know that the wreck is located somewhere within a few hundred meters of latitude 41.93 and longitude 6.26 at a depth of about 1100 meters. For this you will use the LOBSTER Explorer robot. Survey the area by travelling 1km east and 1km north, and export the mission.

Modifying a Mission

You want to continue working on a mission called "North Sea Survey", so you load it. Currently the robot moves clockwise in a square, but you want it to go the other way around. Your task is to edit the mission so that the robot moves over the same route, but instead counterclockwise.

Emergency Procedures

You want to reduce the risk of your robot not returning due to a malfunction. Make sure that in the current mission the robot activates the emergency procedures when half of its battery power has been used or if the mission takes longer than an hour.

C.2.6 Maximum Task Times

These are the maximum times we defined for each of the user tasks in Section C.2.5. These are defined in this separate table because we do not want the participant to know in advance how hard or easy we think the task is, since this might influence their behaviour.

Task	Maximum time
Find the Shipwreck	8 minutes
Modifying a Mission	5 minutes
Emergency Procedures	3 minutes

C.2.7 Questions

In this section we describe the questions we asked the participants after they attempted to complete all tasks. Our questioning process consists of two parts, we start by asking the participant to fill out the System Usability Scale (SUS)[6] on paper. Followed by asking some open questions to get more details on the experience of the participant.

System Usability Scale

We asked each participant to fill in the System Usability Scale on a form, which can be found in Appendix J. We chose the SUS because it is a reliable indicator of usability and has been used successfully for multiple decades.

Open Questions

Finally we asked the following three questions to get a more specific idea of the experience of the participant.

- Can you name three things you liked about the product?
- Can you name three things you disliked about the product?
- Do you have any ideas for future improvements or additions to the product?

Appendix D

Research Report

This research report was written as the first deliverable in the first two weeks. Parts of it have been included in other parts of this final report, but we have included the full research report in this appendix as well.

Introduction

When researchers and off-shore companies use an Autonomous Underwater Vehicle (AUV) to survey, measure or inspect the sea, they need to give actions to the robot about what it should be doing. With a user interface (UI) for mission planning, they are able to easily create a mission themselves without having to require knowledge on any of the internal systems of the robot. LOBSTER, a research project of Allseas, is developing an AUV and is still missing a mission planner to create missions. This paper will explain in detail the research phase for creating this software, first by defining what a mission planner is, then exploring similar products, followed by the design choices made for implementing the concept and finally the requirements of the mission planner.

D.1 Mission Planner

The mission planner allows a user to set up a mission in different steps each containing an action. An action can for example be moving the robot or activate a sensor. Each step can have information about the duration of the action or battery level after that step or other. Afterwards, the mission can be saved and exported to the specific robot where it can be executed by mission control.

D.1.1 Verifying Plans

When a user is composing a mission, the system should check if the proposed steps can be executed by the robot. Different types of checks can be made which we will explain in this section.

The first and most obvious one is that the mission should have a start and end phase. These phases can have different steps that have to be done. For example the user can choose that the robot will sail back to the ship with GPS or just resurface from its last position.
Furthermore the robot has a limited battery capacity, which is why the mission planner should check if the robot can reach the end phase without an empty battery.
Some actions can't be adjacent. For example it is not possible to resurface twice or go the same position, orientation or depth. The program can also give a warning when two steps can be combined like doing a dive action diving for 10 meters twice could be combined in diving 20 meters.

D.2 Stakeholders

Since this project is part of our bachelor thesis, we have multiple stakeholders, all with different preferences and needs:

- Allseas: Our client is an offshore contractor, specialised in pipelay, heavy lift and subsea construction. They will function as our primary customer. Allseas uses underwater vehicles to explorer underwater construction sites, and to perform research in specific underwater areas.
- LOBSTER: Our first stakeholder is LOBSTER. Their robot will be our primary target, and as such will have the most impact on our design choices and assumptions regarding robot specifications.
- Researchers: Since this product will primarily be used by companies or researchers, our user experience design choices will mostly focus on their knowledge and skills. Our product should be easy to learn and use for this group.

D.3 Similar Products

To inform ourselves on the state of the art, we researched which similar products already exist. We looked for similar products primarily by using search engines. We used both regular Google and Google Scholar to search for terms like "Mission planning", "AUV software", "AUV planning", etc. We then selected any mission planning systems for AUVs for which we could find sufficient information. Furthermore we also included some interesting products aimed at other types, such as aerial, land and even space vehicles. The key differences between the similar products we found are summarised in Table D.1, which we will elaborate on in the following section.

As can be seen in the summary, almost all existing products are desktop applications, with the notable exception of the NASA Open MCT web application. Furthermore seven of these systems also include a mission control interface which allows interaction with the vehicle during the mission. However this is not applicable in our case as the LOBSTER vehicle is not able to communicate while underwater. In the vehicles column we indicate which specific robots the software is aimed at or a dash if it is applicable across a wider range of vehicles.

Finally we looked at the main User Interface approach taken by each of these products, which is summarised in the last column. Most planners for underwater vehicles used a 2D top-down map of the seabed at the mission location. On this map the coordinates of the mission can then be drawn. The exception is the Marius mission planning system which is based on Petri nets, which can be specified graphically. These Petri nets essentially capture the sequence of steps based on conditions in the form of a graph.

On the other hand we have two products aimed at aerial vehicles which make use of satellite images. Of these UgCS displays the area in 3D, which allows the user to intuitively see the altitude at various points

[1] https://ocean-server.com/vectormap/
[2] http://flotte.ifremer.fr/fleet/Presentation-of-the-fleet/Logiciels-embarques/MIMOSA
[3] http://www.tritonimaginginc.com/site/content/software/compatibility/oldindex.htm
[4] https://www.ugcs.com/
[5] http://ardupilot.org/
[6] https://nasa.github.io/openmct/
[7] https://www.lego.com/en-us/mindstorms/about-ev3

Name	Platform	Type	Planning	Control	Vehicles	UI Approach
VectorMAP[1]	Desktop	Underwater	✓	✗	Iver AUVs	2D map
MIMOSA[2]	Desktop	Underwater	✓	✓	Ifremer AUVs	2D map
Triton[3]	Desktop	Underwater	✓	✓	-	2D map
Marius[13]	Desktop	Underwater	✓	✓	Marius AUV	Petri nets
UgCS[4]	Desktop	Aerial	✓	✓	-	3D satellite
ArduPilot[5]	Desktop	Aerial	✓	✓	-	2D satellite
Open MCT[6]	Web	Space	✗	✓	-	Multi timeline
Mindstorms EV3[7]	Desktop	Land	✓	✓	LEGO robots	Block timeline

Table D.1: Comparison of various mission planning and/or control systems

of the mission. ArduPilot however simply displays the 2D top-down satellite view, which does not easily display the altitudes.

Then we have OpenMCT which uses various UI elements of which the most prominent is a multi-track timeline. On this timeline various operations can be scheduled in time, which also allows simultaneous actions. Another timeline based approach is used by Mindstorms EV3, which lays out the operations as blocks which can be put into the right place. This approach complicates simultaneous actions, but is more visual and intuitive.

Overall we found that most underwater mission planners use a top-down map view, which is not well-suited to our product since the control of the LOBSTER is not location-based. Furthermore the Marius system used Petri nets, which allow flexibility with user-specified conditions. On the other hand we also found some programs (Open MCT and Mindstorms EV3) for other fields which use a timeline-based approach which seems more applicable to our product.

D.4 Design Choices

In this section we describe various design choices we made during the research phase of this project. For each of those we justify our choice by listing the alternatives and the choice with their respective advantages and disadvantages.

D.4.1 Target Platform

The target platform of our application is an important decision to be made, since it influences the possible choices of programming languages and libraries. We discerned two main feasible choices of target platforms for this project, targeting the web or a native desktop platform.

Web Application

The web allows developers to create one application which can be used across a variety of systems, since web browsers are available on all desktop systems. This is one of the main advantages of developing for

the web, since there is no need to develop separate applications for separate systems. Furthermore, web applications can be used without installation, which simplifies the usage.

On the other hand, web applications also have some disadvantages, such as only being able to use JavaScript-based languages. Additionally, web applications are not available offline by default, so if this is desired it needs to be specifically implemented.

Native Desktop Application

Desktop applications in general come with various advantages. Since desktop applications do not need a run-time security layer, unlike web applications, they can respond faster. Additionally, native desktop frameworks have access to more system functionalities, such as advanced 3D graphics. Finally, any desktop application is available offline without additional effort.

On the other hand, there are also some drawbacks inherent in the approach of desktop applications. Desktop applications require an installation step. Furthermore, although the user interface framework often provides portability of the graphics, it does not handle other tasks such as accessing the file systems. Therefore, additional effort is required to ensure correct portability among operating systems.

Chosen Target Platform

In the end, we decided that the best fit for our project was to develop a web application. We noticed that most advantages of desktop applications are not applicable to our case, since we don't require advanced 3D graphics or ultimate efficiency.

However, the advantages of web applications, such as ease of use and portability are relevant to our goals. Another factor in the decision making process of our platform was the fact that LOBSTER already works with different UIs that are also web-based, which would lead to an unnecessary increase of complexity when using different platforms for the same product. Finally although it does require some amount of extra effort, we can still make our web application available offline if necessary.

D.4.2 Web Framework or Library

After we chose to target a web application our next decision was which framework (if any) to use. We considered the following possibilities: no framework, Vue.js[8], Polymer[9] and React[10].

One of the approaches to web development is not using any additional framework at all. An advantage of this approach is being independent of additional libraries, and therefore not being limited by the possibilities or requirements of a certain framework or library. However, the downside is additional work on features which could be provided by a library.

Another approach is using a flexible and minimal framework such as Vue.js, which can be used 'incrementally'. This allows developers to choose to which level they use the framework, which provides more flexibility. However, a drawback of this approach is that the framework is less comprehensive and does not provide a syntactic sugar to build components.

[8] `https://vuejs.org/`
[9] `https://www.polymer-project.org/`
[10] `https://reactjs.org/`

React is another option, which is aimed at building interactive user interfaces. It allows developers to build components which manage their own state, from which the application is built. Furthermore, React provides an efficient syntax to describe these components. It also enforces the data hierarchy which can be passed on from parents down to children but not the other way around.

Finally, we also considered Polymer which allows developers to use Web Components. Using polymer, a large library of existing Web Components which can be used in other projects is available[11]. These components can then be customised and incorporated. However, for our purposes one of the downsides of Polymer lies in its complexity. Polymer does not enforce a rigid structure of data hierarchy, instead allowing data flow in both directions, which can result in unnecessary complexity.

Ultimately we decided to use React for our web application, primarily because of its structured approach to components. Using React we will easily be able to create the custom interactive components needed for our interface. Furthermore, the syntax of React will allow us to specify the logic and content of our components close to each other in an elegant way.

D.4.3 TypeScript

Another decision we made was to use TypeScript instead of plain JavaScript. This language adds the option to add types to parts or all of your program, which can help reduce problems at run-time. Furthermore TypeScript is a strict superset of JavaScript, meaning anything that can be written in JavaScript can also be written in TypeScript. The only downside of using TypeScript is the extra required step of compiling the code to JavaScript. However this process can easily be automated so that it is done automatically upon code changes. Therefore we have chosen to use TypeScript to allow ourselves the additional possibility of type checking our code.

D.4.4 Mission Routing Visualisation

The visualisation of mission planning was another important decision. It is the direct link between the idea behind the mission itself, and how the robot will interpret the commands and thus act during a mission. The three different approaches we considered for this were in 2D and 3D Maps and finally a modular timeline. All discussed methods have their benefits and drawbacks which we will describe in the following sections.

Top and side-view map

The first idea that we came up with when we were discussing the visualisation of the route was using two maps, one from above looking onto the sea surface, and another map looking from the side.

The main problem of this implementation was the unknown angle of the side view as the robot can move in three dimensions. Additionally, a two dimensional map would not be able to show movement in the z-direction without clear indication and could even lead to overlapping points of interest.

[11] See `https://www.webcomponents.org/` for a large collection of Web Components

3D Map

A straightforward solution to the aforementioned problems is to make use of a single three dimensional map. Coordinates would then be shown in this map at places where the robot would execute certain actions, easily visualising the entirety of the mission it would then execute.

However a clear drawback of this method is the fact that a three dimensional map requires a significant amount of processing power compared to two dimensional alternatives. Next to this, another problem would arise when users would click on the map to add points in this three dimensional space, as a flat screen cannot show the clicking depth.

Modular Timeline

A modular timeline works with different categories of actions, each uniquely identifiable by their colour. These actions can be added by the user to the timeline in a block or chain building fashion by dragging the different actions onto the timeline in their desired order. Additionally, it allows an easier calculation and visualisation of the power and time usage of each action, and can thus help the user in the mission planning.

A comparable type of software that also uses a modular timeline for programming robots is the LEGO Mindstorms EV3 software (discussed in Section D.3). This software is used to control the behaviour of the robots both in movement as well as all related actions. This software was also created using a modular timeline to allow children to understand and build their own robots.

Chosen visualisation method

The modular timeline was chosen because it is an easy way for all users to understand the flow of the mission, and can also adapt to the user's choices dynamically. An example of this dynamic adaptability is that once a user adds a retrieval action, it no longer allows actions that can only be executed deep below the surface of the water.

Sketching the Modular Timeline

During the design choices of how this map would look like we did some sketching and came up with two main designs, as shown in Figure D.1 and Figure D.2 . Figure D.3 is an extension of the first design as this design used graphics or icons for each of the actions.

Next to the way we wanted the timeline to look like we also had to think of the entire user interface besides the timeline. Since a user needs to grab all these different tasks, we added side-tabs that house the tasks by category. Furthermore a button to export the current mission was added. This all was drawn together in the first UI sketch (Figure D.4)

After looking over these sketches with the entire team we were missing some features, and thought of some quality of life features that would not be too complicated to add. These were for example an estimation of the battery percentage and time spent on this mission, as well as a zoomed out mini-map of the timeline. (Figure D.5)

Added to these were two slightly larger features that helped the user in the creation and maintenance of

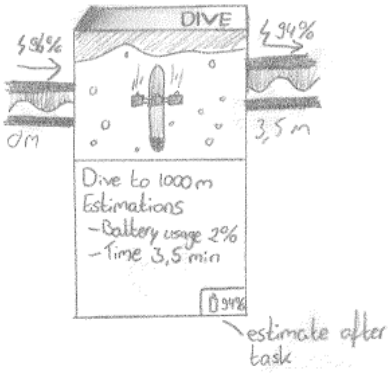


Figure D.1: The first design, including an icon of the specific task and a thicker timeline line.

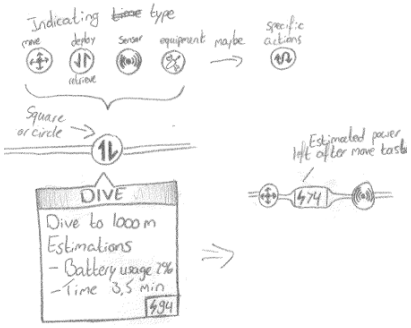


Figure D.2: The second design, having task specific blobs and a thinner timeline with integrated charge estimation.

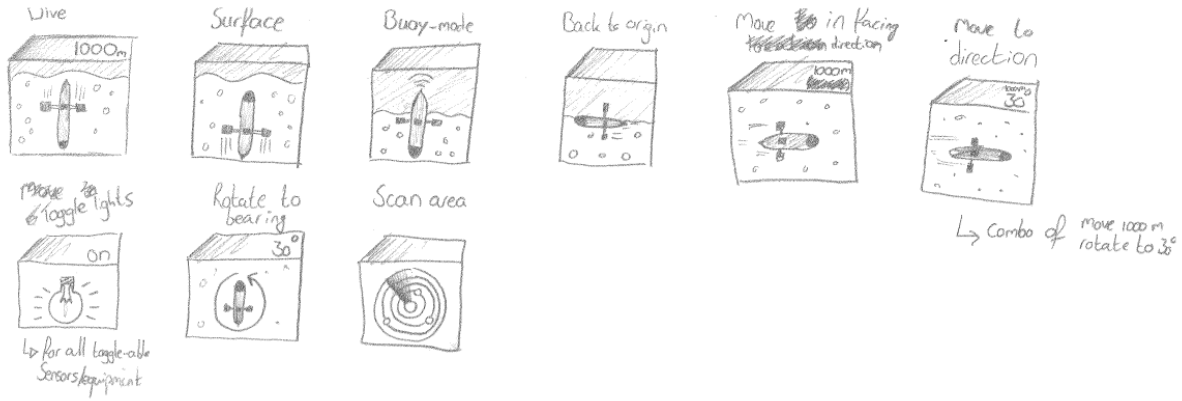


Figure D.3: Sketches of some of the actions that a user could use to create a mission for their robot. We extended these icons with extra information below as seen in Figure D.1 to add clearer information such as how far the robot would dive, or to where the robot should rotate in degrees.

their missions, these being an option to select some tasks and "saving" this sequence for later replication (Figure D.6), but also being able to select tasks and grouping them as a "Phase", allowing for better organisation of the mission planning.

D.4.5 Interface architecture

To further clarify our understanding of the intended user interactions we created a diagram which can be found in Figure . This diagram maps out the possible transitions between actions and screens by the user within our application.

D.4.6 Exported file formatting

To export the missions created by the users, we decided to use the JSON file format. We preferred this format over others because of multiple reasons, the first being the fact that JSON is a widely used file

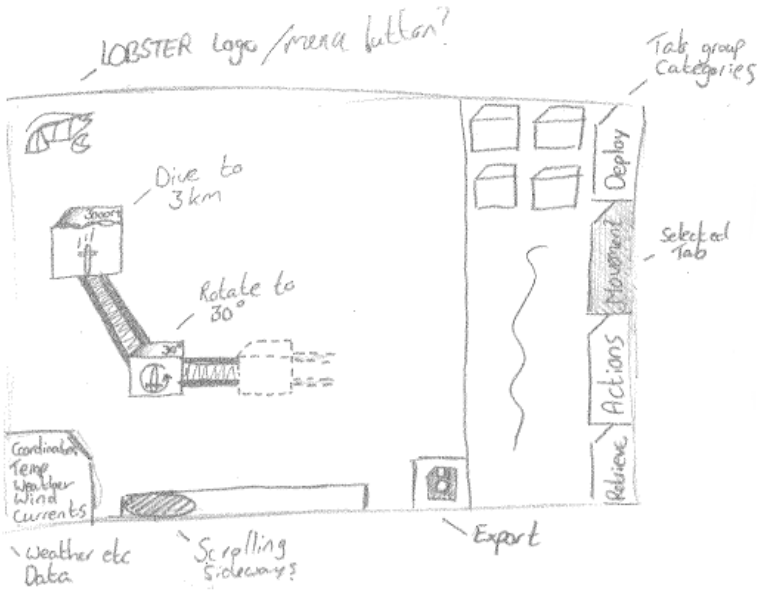


Figure D.4: A sketch of the first iteration of the entire UI

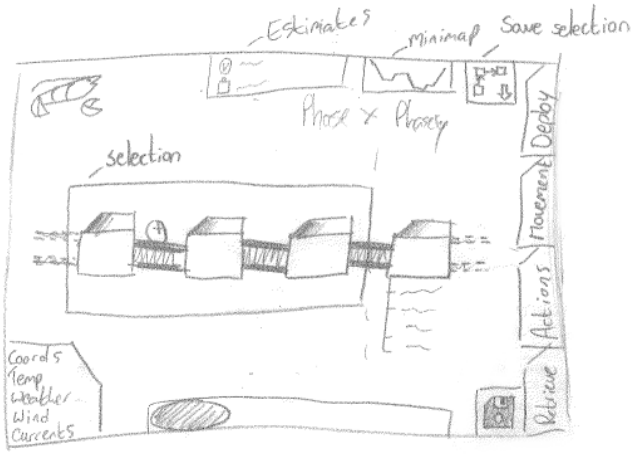


Figure D.5: A sketch of the second iteration of the entire UI

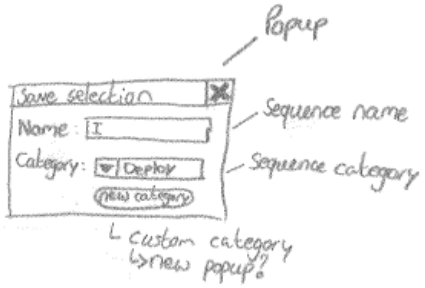


Figure D.6: Popup that appears when saving a selection of tasks

format for web-based data storage. As mentioned in Section D.4.1 we chose for a web-based application, and we will thus use JavaScript. Additionally every popular language has interpreters for JSON files and therefore this file format is an easy file to read for the robot interpreters to convert to actual commands. Last but not least, JSON is a simple format to use and read both for humans and machines.

After we decided that the JSON file format was the best choice for this project, we also had to decide on how these files would be structured, as JSON files are incredibly flexible in their style. Luckily, JSON Schema [14] comes to the rescue. JSON Schema is used to annotate and validate JSON documents. This way, we can define a set of rules to which the output file must cohere. We can also use this to test the output of our program. An example of such a schema file can be found in Appendix H.

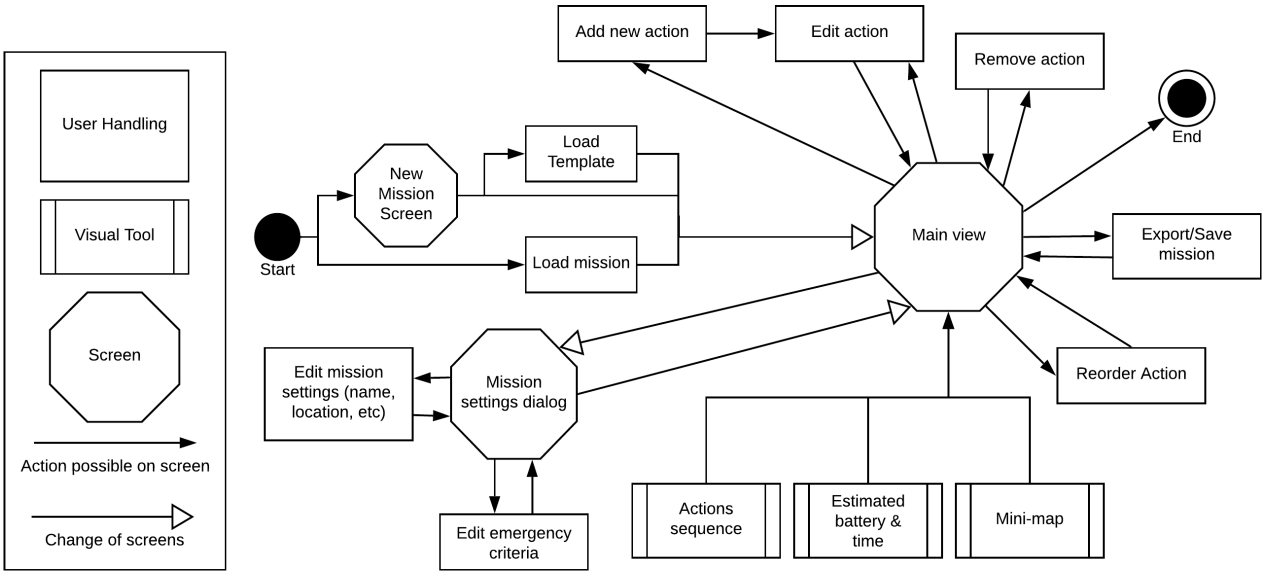


Figure D.7: Diagram of user interactions with the application

D.4.7 Multiple robot compatibility

Initially, the idea was to create this software just for the LOBSTER robot. However during our first meeting it came to our attention that it was possible that the LOBSTER robot was not the only robot in need of this software. For example, second generation LOBSTER or other robots would need entirely new and specific software.

We decided to add the option for adding custom robot specifications and saving these for later use, and also including pre-made robot templates for the user to work with. These settings would then change the options available to the planning of the mission depending on the selected robot.

D.4.8 Testing

React comes together with the testing framework Jest. Jest is made for both unit and integration tests, which means we can use it to create all of our tests. It even supports UI snapshots, so allowing us to verify UI changes.

D.4.9 Logging

To improve debugging and data extraction, we decided to add logging points to the timeline. These logging points determine what kind of data should be logged, when it should be logged, and how often it should be logged. The implementation of how the robot should log this is out of the scope of this project. We will only tell the robot when it should log.

D.5 User Study and Tests

We will be performing two types of user tests, namely formative and summative. The formative tests are aimed at exploring the feasibility of our interface and directing our development towards most needed features. On the other hand the summative user study aims to evaluate the quality, problems and advantages of our final interface.

D.5.1 Target Users

The target users of our application are operators working at organizations which use underwater robots. These organizations could be non-commercial such as research institutes or commercial companies. These operators are trained on and aware of the capabilities and limitations of underwater robots and the goals of their mission. Therefore we can assume that they understand the basic concepts of underwater exploration.

D.5.2 User input

Mission planning software is not new and is used a lot by our target users. We will survey different companies and researchers so we don't invent the wheel again. They will be asked: Why they used it, how they use it and what features they miss. During the 10 weeks we will try to continue having conversations with them to get a better understanding of what the user wants.

D.5.3 Formative Tests

We plan to start conducting the first formative user tests from week 5 onwards, since we expect to have a sufficiently complete minimal viable product at that point. However if we achieve this faster, we could start the formative tests earlier.

The formative tests will be primarily aimed at determining the main deficiencies and omissions in the interface during development, to guide the shaping of the application. Therefore we will be collecting both explicit feedback from the test subjects and the completion rates and average times of the tested tasks.

We aim to conduct our user tests on members of the target group of the application as far as possible. However if during some weeks not enough members of the target group are available, we may also make use of other subjects who are not familiar with application already.

D.5.4 Summative Study

Towards the end of the Bachelor End Project we also intend to perform a summative user study to ascertain the overall quality of our developed product. We will let the user perform basic instructions that will use all the basic features of our product, and evaluate how easy-to-use our application is. This will be done by specifying appropriate tasks beforehand, along with success criteria for these tasks. These

criteria specify amongst others the maximum provided time to complete the task and at what point hints should be given.

Additionally we will specify which metrics we will be focusing on as results of the study. These could be things like average completion time per task, average completion rate or things like results from a questionnaire which asks how the user perceived the tool.

Appendix E

Example Scenarios

A few example scenario's will be shown here, varying from the simplest behaviour to more specific abilities the final product may be able to provide. These are meant to clarify the actual nature of the intended user interactions arising from our requirements. These scenarios are based on all requirements, including the should-haves and could-haves, so it is possible that the final product will not support all these scenarios.

Basic mission creation

A user is able to create a new mission in the menu, add their desired actions the robot is supposed to perform through the categorized tabs, and then press the export button to download this mission for the user to upload to their robot.

Tweaking of missions

A user is able to upload a previously created mission to the program through the menu. The user is then able to edit the individual actions of the mission, for example changing the order of actions or action parameters on the action-blocks itself or adding/removing of actions, and is then able to export this new mission.

Mission estimates and critical conditions

A user is able to see the estimated battery charge and time required for this mission to complete without any anomalies at the top of the screen. Additionally, the user is able to include critical conditions in the mission creation screen, such as the battery not being allowed to drop below a certain value, on which the robot will enter a certain state the user provides.

Illegal action

When a user tries to add illegal actions to the mission sequence, such as surfacing twice in direct succession, the program will not allow this and show that it is not possible.

Robot and mission templates

Upon creating a new mission, the user is able to select from some mission templates in the creation screen, additionally, in the same screen, the user is able to select a robot template to make the program adapt to the capabilities of their robot.

Phasing and logging

The user is able to select actions that require logging on the action block. The user is also able to select multiple actions and group them into a phase, a mission could for example be split up into multiple phases which include but are not limited to diving, searching, scanning and resurfacing.

Custom robot specifications

The user is able to create their own robot template through the menu, this template can be shared online for other users that own the same robot to use, or can be privately uploaded for private use.

Worst-case time estimation and battery warnings

The user is able to see an estimation of the maximum amount of time a mission can take, taking possible emergency shutdown scenarios into account on the worst places possible, which is shown by the program. Using this the user can understand what the estimated maximum amount of time they have to wait before the robot will resurface is. Additionally the user will be warned by the program if adding or modifying an action leads to the estimated battery level dropping below some user-configured threshold.

Selection saving

The user is able to select a set of consecutive actions and use a button to make the program save this specific sequence. The user can then reinsert this sequence later at any other point.

Top-down map and small timeline view

The user is able to see two different visual representations of the estimated mission locations timeline. Firstly the user can look at a timeline which shows the estimated depth of the robot throughout the mission in the form of a depth graph. Secondly the user can look at a mini-map which shows the estimated locations of the robot in a top-down view.

Conditional branching and emergency planning

The user is able to add conditions to certain actions, this is added to allow the vehicle to perform pre-programmed behavior in the case of a failure, an example could be trying to find an object, and doing some other behaviour in the case it is not found. Additionally the user can modify the global emergency criteria to decide the minimal allowed battery charge.

Location, depth, wind, currents and waves

The user is able to view a forecast and estimation of various characteristics of the water at the specified mission start location. These might include characteristics such as the depth of the sea, the expected wind direction and speed, expected currents and expected wave height.

Appendix F

Requirements

In this section we describe the requirements of our product using the MoSCoW method. Requirements are ordered from most important ("must-have") to lower priorities ("should-have", "could-have") and finally deliberate omissions ("won't-have"). The different colours of bullet points indicate the categories of the requirements:

- General - Features of the entire program
- Mission planning - Features related to global mission planning functionalities
- Timeline - Features related to the modular timeline of the Mission Planner

F.1 Must-haves

- Ability to export the mission to a JSON file
- "New Mission" setup procedure including:
  - Mission name
  - Mission coordinates
  - Choice of mission template (optional)

- Saving and loading of mission files

- Give an estimation of the total mission-cost in terms of battery percentage and total length
- Possibility to add critical conditions, as the mission might fail in some conditions. An example would be a low battery, for which the robot must shut down and resurface.

- Each action has a possible set of parameters to be set by the user
- Give an estimate of battery usage and length (time) per action
- Timeline with actions that show the general flow of the mission, categories of actions have different identifying colours, actions include:
  - Movement actions
  - Enabling/disabling of sensors and hardware (camera, lights etc.)
  - Deploy/retrieval specific actions
- Disable conflicting action combinations
- Editing of actions by insertion, removal or adding

F.2 Should-haves

- Have default mission templates for users to use and tweak
- Robot specification templates that allow the program to handle different types of robots

- Allow the user to define phases in the timeline which allows for displaying information about that group of actions
- Allow the user to add logging steps that tell the robot when to log, what data to log, and how often it should log that data.

- Only the actions that are possible with the selected robot configuration should be possible to be used
- Allow the user to provide a maximum depth for the mission manually
- Display a warning if the estimated battery usage exceeds the capabilities of the robot
- Allow user to upload new robot specifications
- Display worst-case time estimate of the mission, this mostly happens when a emergency power-down occurs
- Add buoy- and travel-mode options at retrieval to either keep the robot in-place or make it sail back to the starting position after it has finished its mission
- Display time estimate of the current mission

- A top-down view of the water, including the estimated locations of the robot during the mission
- A user can select a group of actions that can be saved for future usage
- A small view of the entire timeline indicating the depth differences of the mission

F.3 Could-haves

- Conditional branching and emergency planning in case of failure or other anomalies
- Display the depth of the body of water at the inputted GPS-location either automatically or manually, and give a warning if the given actions exceed or come close to this depth
- Animations for the User Interface
- Directly send output to the robot from the mission planner application
- Calculate and store the appropriate calibration of the magnetometer to be used at the mission location
- Ability to upload self-made robot specification templates for others to use, with possible flag for non-shared uploading.

- Allow mission branching based on conditions
- Visuals for the wind, waves and currents of the inputted GPS-location, generated or given manual
- Allow the user to upload a log file to visualise the passed mission

F.4 Won't-have

- A 3D map of the mission area

Appendix G

Mission Visualizations

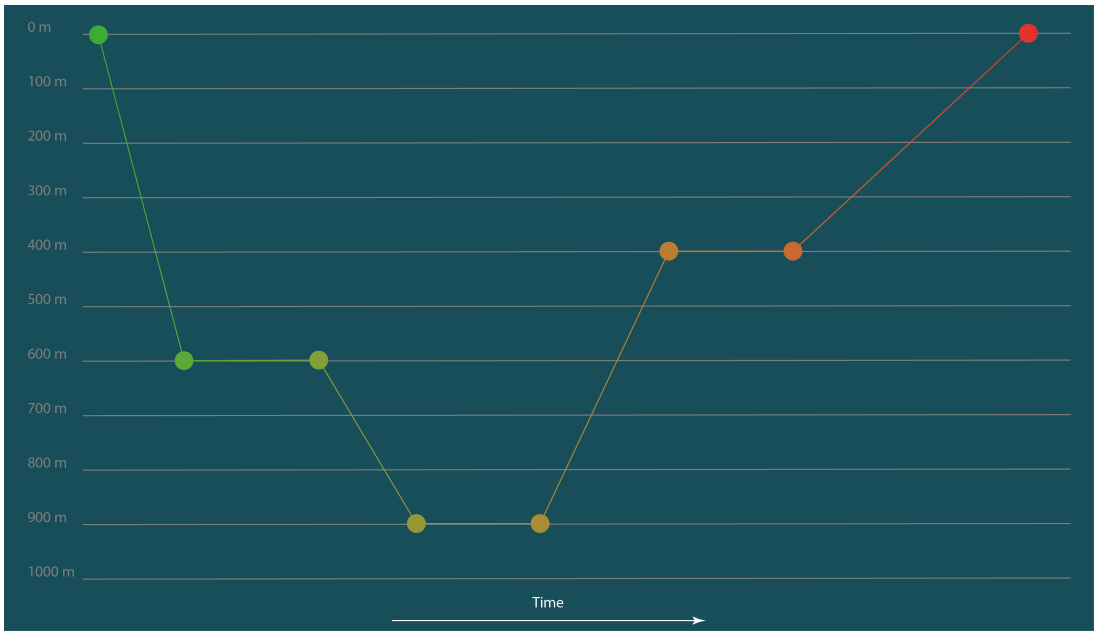


Figure G.1: Battery over Time

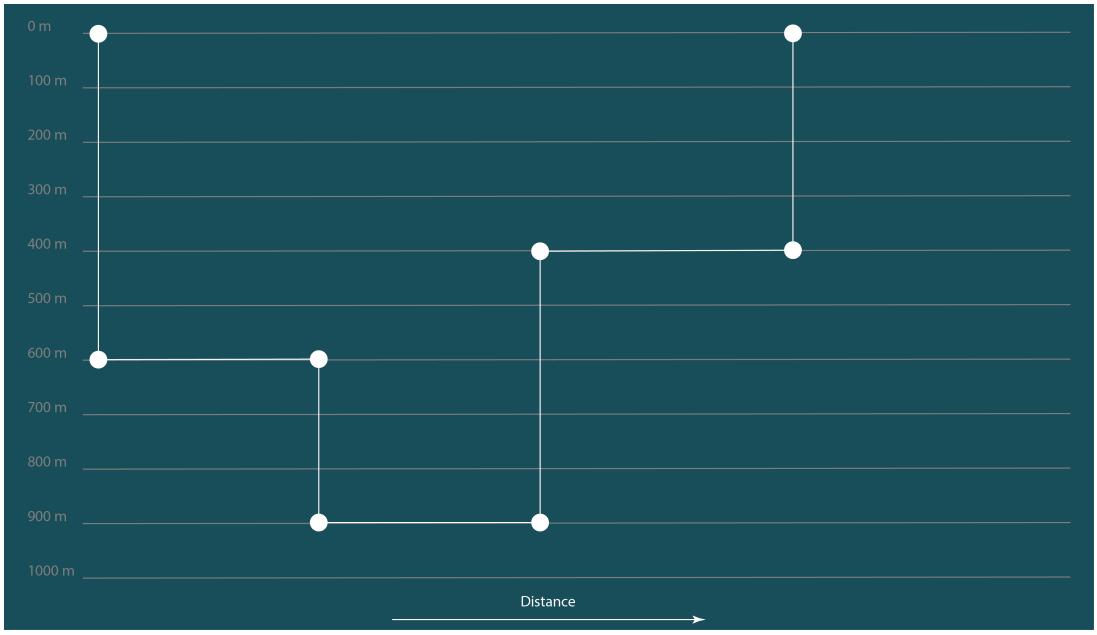


Figure G.2: Battery over Time

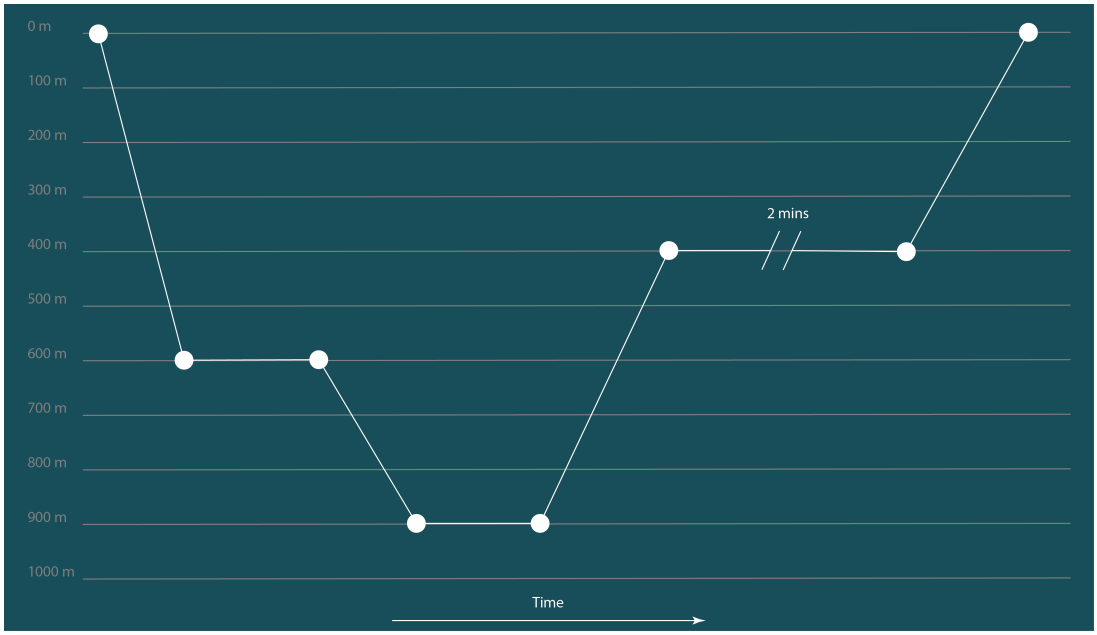


Figure G.3: Battery over Time

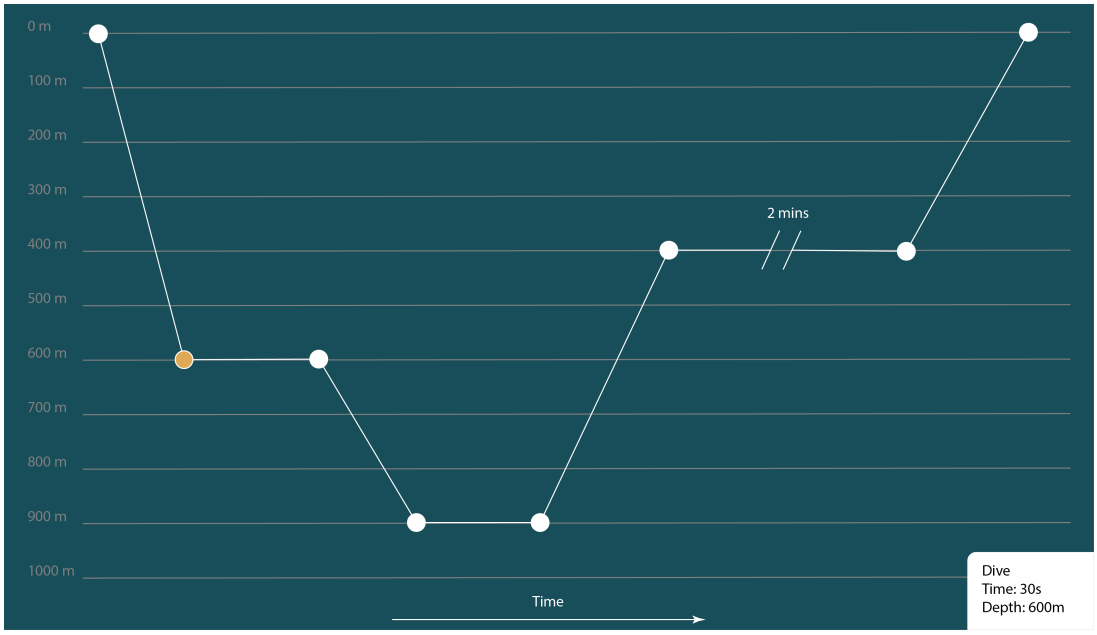


Figure G.4: Battery over Time

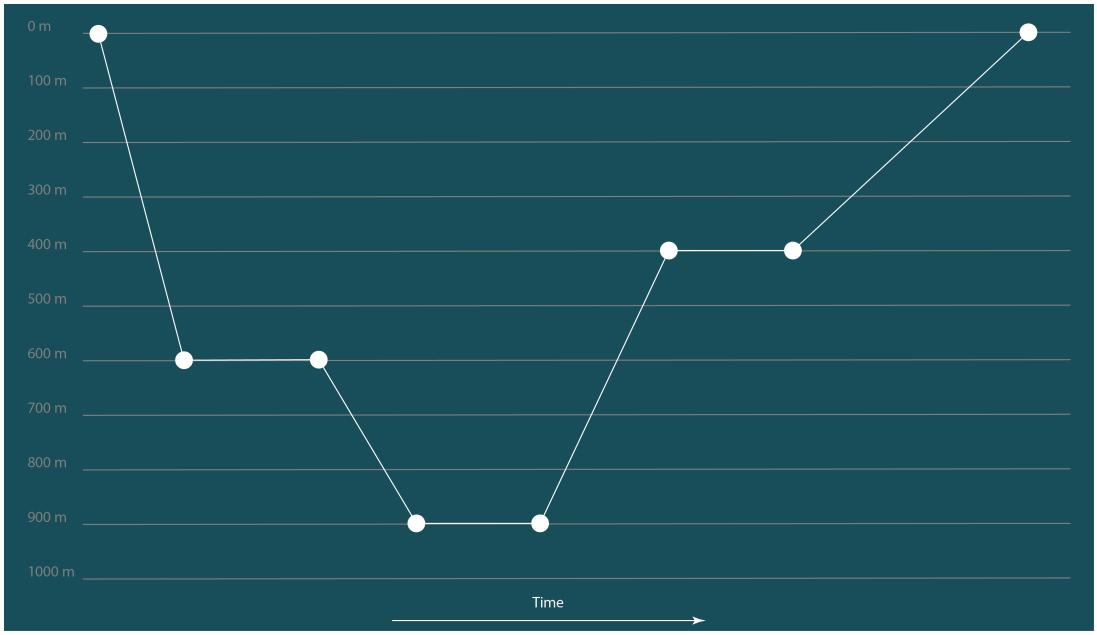


Figure G.5: Battery over Time

Appendix H

Output JSON Schema

This appendix contains the initial example of a JSON Schema for the mission plan format. This JSON Schema describes the format of the mission plan in a way which can be verified easily by software. This makes it easier to check that the software creates the plan correctly and makes sure that all edge cases have been thought of. We made this schema as part of the research phase and as such it does not reflect the final product.

The final output schema can be found in our Git repository on GitHub. Since this repository is private you will have to request access, you can do this by sending an e-mail to the contact person listed in Appendix A.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://lobster.com/mission.schema.json",
  "title": "lobster mission",
  "type": "object",
  "required": [
    "name",
    "configuration",
    "steps"
  ],
  "properties": {
    "name": {
      "description": "The unique identifier of a mission",
      "type": "string"
    },
    "configuration": {
      "description": "describes all mission configurations",
      "type": "object",
      "required": [
        "start location"
      ],
      "properties": {
        "start location": {
          "$ref": "#/definitions/coordinate"
        }
      }
    },
    "steps": {
      "description": "describes all mission steps",
      "type": "array",
      "items": {
        "$ref": "#/definitions/step"
      }
    }
  },
  "definitions": {
    "step": {
      "type": "object",
      "required": ["name"],
      "properties": {
        "name": {
          "type": "string"
        },
        "parameters": {
          "type": "object",
          "patternProperties": {
            "^.*$": {}
          }
        }
      }
    },
    "coordinate": {
```

```
      "type": "object",
      "required": [
        "latitude",
        "longitude",
        "elevation"
      ],
      "properties": {
        "latitude": {
          "type": "number",
          "minimum": -90,
          "maximum": 90
        },
        "longitude": {
          "type": "number",
          "minimum": -180,
          "maximum": 180
        },
        "elevation": {
          "type": "number",
          "maximum": 0
        }
      }
    }
  }
}
```

Appendix I

Robot Template

This appendix lists the JSON Schema[14] for the robot templates. This schema specifies the parameters which each robot should define in order to be used with our application.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://lobster.com/schemas/robot-template.json",
  "definitions": {
    "robot-template": {
      "type": "object",
      "required": [
        "name",
        "speed",
        "batteryUsage",
        "maxDepth",
        "cameraVideoConfigurations",
        "equipment",
        "defaultCriticalConditionsSettings"
      ],
      "properties": {
        "name": {
          "type": "string"
        },
        "length": {
          "type": "number",
          "minimum": 0,
          "description": "The length of the robot in meters, which is used to determine the minimum dive depth"
        },
        "speed": {
          "$ref": "#/definitions/speed",
          "description": "defines the top speeds of the robot"
        },
        "maxDepth": {
          "type": "number",
          "minimum": 0,
          "description": "The maximum depth this robot can physically reach"
        },
        "batteryUsage": {
          "$ref": "#/definitions/batteryUsage",
          "description": "defines the battery usages of the robot, all in percentage of the total battery capacity per second"
        },
        "cameraVideoConfigurations": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/cameraVideoConfiguration"
          }
        },
        "equipment": {
          "type": "object",
          "description": "List of all equipment. The key of each equipment represents the title displayed.",
          "items": {
            "$ref": "#/definitions/equipment"
          }
        },
        "defaultCriticalConditionsSettings": {
          "$ref": "critical-conditions.json#/definitions/critical-conditions"
        }
      }
    },
    "speed": {
      "type": "object",
      "required": [
        "translation",
```

```
        "rotation",
        "surfacing"
      ],
      "properties": {
        "translation": {
          "type": "number",
          "exclusiveMinimum": 0,
          "description": "top movement speed in meters per second"
        },
        "rotation": {
          "type": "number",
          "exclusiveMinimum": 0,
          "description": "top rotational speed in degrees per second"
        },
        "surfacing": {
          "type": "number",
          "exclusiveMinimum": 0,
          "description": "speed the robot travels to the surface in case of power outage"
        }
      }
    },
    "batteryUsage": {
      "type": "object",
      "required": [
        "idle",
        "movement"
      ],
      "properties": {
        "idle": {
          "type": "number",
          "minimum": 0,
          "description": "battery usage in percentage per second when idle"
        },
        "movement": {
          "type": "object",
          "required": [
            "translation",
            "rotation"
          ],
          "properties": {
            "translation": {
              "type": "number",
              "exclusiveMinimum": 0,
              "description": "battery usage of thrusters moving forwards at full power in percentage
                  per second"
            },
            "rotation": {
              "type": "number",
              "exclusiveMinimum": 0,
              "description": "battery usage of thrusters when rotating at full power in percentage per
                   second"
            }
          }
        },
        "lights": {
          "type": "number",
          "minimum": 0,
          "description": "battery usage of lights at full power in percentage per second"
        }
      }
    },
    "cameraVideoConfiguration": {
      "type": "object",
      "required": [
        "width",
        "height",
        "fps"
      ],
      "properties": {
        "width": {
          "type": "number",
          "exclusiveMinimum": 0,
          "description": "width of the camera recording in pixels"
        },
        "height": {
          "type": "number",
          "exclusiveMinimum": 0,
          "description": "height of the camera recording in pixels"
        },
```

```
        "fps": {
          "type": "number",
          "exclusiveMinimum": 0,
          "description": "number of pictures taken by camera per second"
        }
      }
    },
    "equipment": {
      "type": "object",
      "required": [
        "type",
        "batteryUsage"
      ],
      "properties": {
        "type": {
          "type": "string",
          "description": "Type defines how the equipment is used",
          "enum": [
            "toggle",
            "camera",
            "percentage"
          ]
        }
      }
    }
  }
}
```

Appendix J

System Usability Scale Questionnaire

On the next page you can find the System Usability Scale questionnaire as used during our final summative usability study. This page was printed out and given to the participants to fill out after all tasks had been attempted.

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

Appendix K

Ethics Checklist for Usability Study

On the next pages you can read the ethics checklist we filled in for our usability study, which classifies our study as 'minimal risk'.

Delft University of Technology
Ethics Review checklist for human research as part of TI2806 Context project

(Adapted from HREC checklist version 10.10.2017)

This checklist should be completed for research that involves human participants and should be submitted before potential participants are approached to take part in your research study.

In this checklist we will ask for additional information if need be. Please attach this as an Annex to the application.

Please have this form signed by your supervisor for approval.
Please upload the signed documents unto bright space

I. Basic Data

Project title:	Mission planning and Control for a deep sea robot
Name and student ID group members	Gijs Koning, 4582381 Thijmen Langendam, 4592646 Dennis Mouwen, 4452070 Jochem Raat, 4582381
E-mail contact person	j.j.raat@student.tudelft.nl
Name of project supervisor	Mark Neerincx

II. A) Summary Research

(Please very briefly (100-200 words) summarise your research, start with giving a short description of the project, next a shortly description of the activity that involves human participation (here referred to as the study), explain the question for the research of this study, who will participate, the number of participants to be involved and the methods/devices to be used. Please avoid jargon and abbreviations).

As part of our Bachelor End Project for our Bachelor in Computer Science and Engineering at the TU Delft, we are developing a Mission Planning application for underwater robots. This application allows users to plan a mission using a laptop or desktop, which can then be uploaded to a robot.

As part of our project we plan to do a usability study. This usability study aims to asses the qualities and problems of our product by having participants complete simple tasks on a laptop. These tasks ask the participant to plan or modify a mission. The participant will then try to complete this task by using the application on the laptop and interacting with our interface.

Afterwards we will ask the participants some questions on their experience using our application. In total it will not take longer than 30 minutes for each participant.

B) Risk assessment

Please indicate if you expect any potential risks for the participants as a result of your study and, if so, explain how you will try to minimize these.

We do not expect any potential risks for the participants in our study.

III. Checklist

Question	Yes	No
1. Does the study involve participants who are particularly vulnerable or unable to give informed consent? (e.g., children, people with learning difficulties, patients, people receiving counselling, people living in care or nursing homes, people recruited through self-help groups).		X
2. Are the participants, outside the context of the research, in a dependent or subordinate position to the investigator (such as own children or own students)?[1]		X
3. Will it be necessary for participants to take part in the study without their knowledge and consent at the time? (e.g., covert observation of people in non-public places).		X
4. Will the study involve actively deceiving the participants? (e.g., will participants be deliberately falsely informed, will information be withheld from them or will they be misled in such a way that they are likely to object or show unease when debriefed about the study).		X
5. Will the study involve discussion or collection of information on sensitive topics? (e.g., sexual activity, drug use, mental health).		X
6. Will drugs, placebos, or other substances (e.g., drinks, foods, food or drink constituents, dietary supplements) be administered to the study participants?		X
7. Will blood or tissue samples be obtained from participants?		X
8. Is pain or more than mild discomfort likely to result from the study?		X
9. Does the study risk causing psychological stress or anxiety or other harm or negative consequences beyond that normally encountered by the participants in their life outside research?		X
10. Will financial inducement (other than reasonable expenses and compensation for time) be offered to participants?		X
Important: if you answered 'yes' to any of the questions mentioned above, reconsider your study set up, as you will have to submit a full application to Human Research Ethics Committee (HREC) of university with the support of your supervisor. Note that HREC meets once a month. (see: HREC TUDelft website for more information).		
11. Will the experiment collect and store videos, pictures, or other identifiable data of human subjects? [2] If "yes", please fill in Annex 1 and make you sure you follow all requirements of the applicable data protection legislation. In addition, please provide proof by sending us a copy of the informed consent form.		X
12. Will the experiment involve the use of devices that are not 'CE' certified?		X

1 **Important note concerning questions 1 and 2.** Some intended studies involve research subjects who are particularly vulnerable or unable to give informed consent .Research involving participants who are in a dependent or unequal relationship with the researcher or research supervisor (e.g., the researcher's or research supervisor's students or staff) may also be regarded as a vulnerable group . If your study involves such participants, it is essential that you safeguard against possible adverse consequences of this situation (e.g., allowing a student's failure to complete their participation to your satisfaction to affect your evaluation of their coursework). This can be achieved by ensuring that participants remain anonymous to the individuals concerned (e.g., you do not seek names of students taking part in your study). If such safeguards are in place, or the research does not involve other potentially vulnerable groups or individuals unable to give informed consent, it is appropriate to check the NO box for questions 1 and 2. Please describe corresponding safeguards in the summary field.

2 Note: you have to ensure that collected data is safeguarded physically and will not be accessible to anyone outside the study. Furthermore, the data has to be de-identified if possible and has to be destroyed after a scientifically appropriate period of time. Also ask explicitly for consent if anonymised data will be published as open data.

Question	Yes	No
Only, if 'yes': continue with the following questions:		
➢ Was the device built in-house?		
➢ Was it inspected by a safety expert at TU Delft? (*Please provide device report, see: HREC website*)		
➢ If it was not built in house and not CE-certified, was it inspected by some other, qualified authority in safety and approved? *(Please provide records of the inspection).*		
13. Has or will this research be submitted to a research ethics committee other than this one? (*if so, please provide details and a copy of the approval or submission).*		X

IV. Enclosures (tick if applicable)

- o Full proposal (if 'yes' to any of the questions 1 until 10)
- o Informed consent form (if 'yes' to question 11)
- o Device report (if 'yes' to question 12)
- o Approval other HREC-committee (if 'yes' to question 13)
- o Any other information which might be relevant for decision making by HREC

V. Signature(s)

Students

Names and Signature(s) of students(s)

Gijs Koning

Thijmen Langendam

Dennis Mouwen

Jochem Raat

Date: ***June 14th 2019***

Supervisor

(*Only sign this form if questions 1 to 10 have been answered by the students with a NO*)

As supervisor I agree that questions 1 to 10 should be answered with a "NO"

Name supervisor

Mark Neerincx

Signature:

Date: