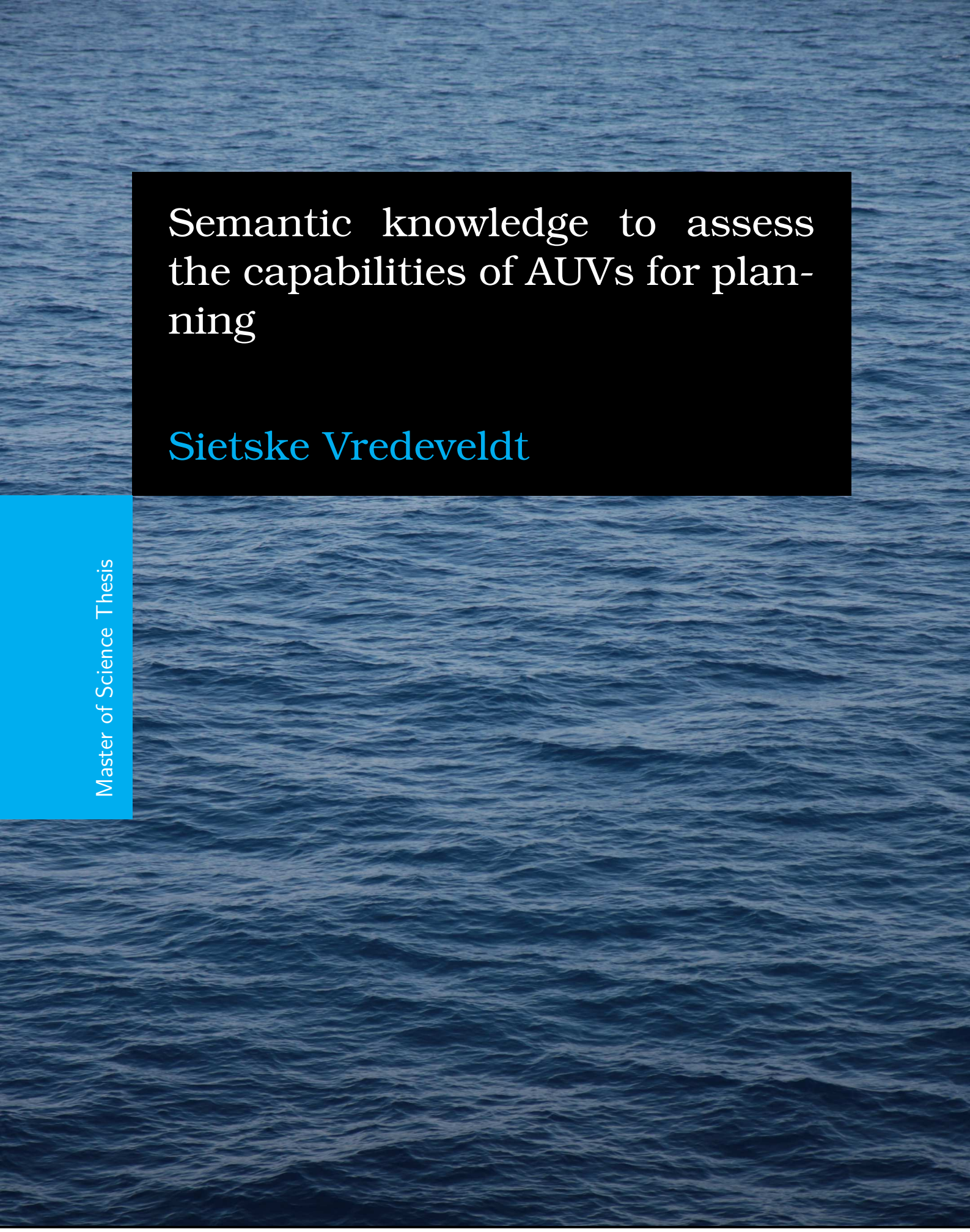


Semantic knowledge to assess the capabilities of AUVs for planning

Sietske Vredeveldt

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Semantic knowledge to assess the capabilities of AUVs for planning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Sietske Vredeveldt

August 31, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by TNO. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Semantic knowledge to assess the capabilities of AUVs for planning

by

Sietske Vredeveldt

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: August 31, 2019

Supervisor(s):

Dr.ir. J. Sijs

Prof.dr.ir. B. de Schutter

Reader(s):

Prof.dr. M. Neerincx

Dr.ir. C. Hernandez Corbato

Ir. J. Fransman

Abstract

Autonomous Underwater Vehicles (AUVs) are unmanned vehicles that are often used for searching an area of the seabed for objects, such as naval mines. For autonomous planning of such search operations, it is useful to be able to infer what tasks an AUV can perform and how well it can do so. The objective for this final thesis project therefore was to develop and implement a semantic knowledge model in the form of an ontology for assessing the capabilities of AUVs that perform mine search operations.

For developing an ontology, first the use case and existing ontologies for related topics were studied to obtain an overview of the concepts and relationships that are most relevant for AUVs that perform mine search operations. The ontology was then developed by formalising these concepts and relationships, and by identifying instances of these concepts that are specific to the use case. By implementing this ontology and developing a reasoning process for it, automatic inference of the capabilities of an AUV based on the components it is equipped with was realised. In addition, a data-driven methodology for performance assessment was developed and incorporated in the ontology to provide a basis for automatic inference of the expected performance of an AUV. This performance assessment methodology was tested by performing simulations with an AUV in a virtual underwater environment. These simulations showed that the developed methodology can successfully be used to predict performance under different conditions.

Table of Contents

List of Figures

List of Tables

Acknowledgements

This thesis is the result of a year of researching at the TNO Acoustics & Sonar department. The topic I have been working on initially was very unfamiliar to me, which made my research quite challenging, but therefore also instructive and exciting. Working on such a new subject in a professional environment has taught me a lot about the process of researching in general and more specifically about the challenges of combining different research areas. Therefore, I would like to thank my supervisor Dr.ir. J. Sijs for giving me this opportunity and for providing assistance where needed, while also letting me go my own way in this research where possible. Additionally, I would like to thank the colleagues at the A&S department for having me there for a year and for making me feel welcome. Special thanks to Lukas, who was my roommate at the office and helped me several times in my research, also by being available for discussing big or small struggles in my project. Finally, I would like to express my gratitude to the Royal Netherlands Navy for giving me the time and means to follow a master's programme at the TU Delft.

Delft, University of Technology
August 31, 2019

Sietske Vredeveldt

Chapter 1

Introduction

The development and use of unmanned and autonomous vehicles has received a great deal of attention over the past few decades. A lot of research on topics related to these vehicles is being conducted and the resulting scientific and technological developments are closely followed both in the scientific community and in society. It is important to make a distinction between unmanned and autonomous vehicles, because not all unmanned vehicles are autonomous, for example remotely operated vehicles, and not all autonomous vehicles are unmanned, for example autonomous cars. In this thesis, the focus is on vehicles that are both unmanned and autonomous and more specifically on Autonomous Underwater Vehicles (AUVs). These unmanned autonomous vehicles, also called autonomous mobile robots, are very useful for tasks that are considered to be dull, dirty or dangerous [1]. In the underwater environment, tasks are inherently dangerous and arguably also dirty for humans in some sense. Currently, AUVs are often deployed for underwater search and survey tasks, which can be seen as dull tasks and which would be costly and time consuming if done by divers [1, 2]. An example of such a task is searching the seabed for naval mines, as part of a Naval Mine CounterMeasures (NMCM) operation. This mine search task is taken as the use case in this thesis.

1-1 Motivation

The goal of an NMCM operation is to clear a certain area of the seabed of mines, which serves the higher purpose of guaranteeing safe use of the sea and of port entrances [3]. Traditionally, such operations are performed using manned minehunting or minesweeping vessels or by divers, but several countries are starting to deploy unmanned systems such as AUVs to decrease the risk for their personnel [4, 5].

Two crucial factors for an NMCM operation are its duration and its effectiveness, often measured by the percentage clearance. This percentage describes what fraction of the mines in the area in which the operation was executed is successfully removed [6, 7]. So, in most cases, the goal of an NMCM operation is to achieve a high percentage clearance in a short amount of time.

When AUVs are used in an NMCM operation, these AUVs are generally tasked with searching the area for mines, which can then be removed or destroyed by divers or with a Remotely Operated Vehicle (ROV). For a search operation performed by an AUV as part of an NMCM operation, the duration and the effectiveness are crucial as well. For an AUV to perform well on both these aspects, it needs to have a high level of autonomy. This is required because the environment in which a search task takes place is often uncertain and subject to change. Additionally, the possibilities for communication underwater are limited due to the properties of the medium, so relying on an operator for decision making in challenging situations is not an option. [5, 6, 1]

Important aspects of autonomy for autonomous vehicles are decision making, planning and fault tolerance [1]. A common denominator of these three aspects is that they require a certain level of understanding of the vehicle, its environment and the interaction between the vehicle and the environment. A prerequisite for understanding something is to have knowledge about it, so knowledge is a useful asset for an autonomous vehicle [8, 9]. The type of knowledge that is referred to here can be classified as semantic knowledge, which indicates that the focus is on meaning. This focus on meaning was chosen for this research because it is necessary to take meaning into account when the goal is to realise understanding of a domain.
To be able to use semantic knowledge in an autonomous vehicle, some sort of knowledge representation method is required. A commonly used method that will also be used in this thesis is the ontology format. An ontology is a semantic model of knowledge in the form of concepts and relationships that are relevant for a certain domain. An important advantage of ontologies is that they are both computer-interpretable and intuitively understable for humans. Furthermore, many tools are available for performing automatic reasoning with knowledge in the form of an ontology. Another advantage is that ontologies are widely used in the robotics community and that common robotics ontologies are being developed, which enables the exchange of knowledge between different organisations and different autonomous systems. [10, 11, 12, 13]
Also for the use case considered in this thesis, which is using AUVs in NMCM search operations, research on implementing knowledge in the form of ontologies has been done before. Examples of research on this topic are discussed in [8, 14] and more generally on ontologies for underwater survey mission with AUVs in [11, 12, 15]. Several of these articles concentrate on the planning aspect of AUV autonomy, which was also chosen as the topic to focus on in this thesis because it can affect both the duration and the effectiveness of a search operation performed by an AUV. Also, there is much to gain in this area, since currently some AUVs still use predefined mission plans instead of (re)planning autonomously. As will be discussed later, the decision was made to approach the planning aspect by looking at the capabilities that an AUV that performs NMCM search operations can or should have.

1-2 Research objective

Considering the motivation described in the previous section, the following objective was set for this final thesis project:

Develop a semantic knowledge model in the form of an ontology for assessing the capabilities of AUVs that perform NMCM search operations, to support the planning of these operations, and implement this ontology to show its relevance for the use case.

To achieve this objective, four sub-problems were defined, which will be addressed in this thesis. These sub-problems are the following:

1. Analyse the use case to obtain an overview of the concepts that should be included in the ontology.
2. Study ontology theory and existing ontologies that are relevant for the use case to learn how an ontology can be developed and implemented.
3. Develop a semantic knowledge model of the use case in the form of an ontology.
4. Implement the ontology in such a way that it can be used to assess the capabilities of an AUV that performs NMCM search operations and to quantify these capabilities in the form of measures of performance (MOPs)

Consequently, the goal that was set for the development and implementation of the ontology was to be able to automatically infer whether a vehicle with a given set of hardware and software components can perform a given task and, if so, how well this task can be executed. Realising this goal would be helpful for planning search operations, since the corresponding planning system could then be made "aware" of the capabilities of the available vehicles, allowing the system to assign and plan tasks more optimally. Additionally, the goal was set to keep the ontology general enough to be applicable for other (autonomous mobile robot) use cases than the one considered here.

1-3 Thesis outline

The first phase of this final thesis project consisted of performing a literature survey to obtain the necessary theoretical background for this research. This background was required for the first two sub-problems defined in the previous section. Therefore, an overview of this background concerning the use case and the topic of ontologies is given in Chapter 2. After the literature survey, the process of developing an ontology for the use case was started. This development process and the resulting developed ontology are described in Chapter 3, focussing on inferring the capabilities of a vehicle. The implementation of the ontology for inferring a vehicle's capabilities is also discussed in Chapter 3. The second part of sub-problem 4, the quantification of a vehicle's capabilities in the form of MOPs, is covered in Chapter 4. This chapter describes how the ontology was extended to enable the use of MOPs. In addition, Chapter 4 illustrates how these MOPs can be used by showing how data obtained from simulations was processed to obtain values for the MOPs and how these values can be incorporated in the developed ontology. The last chapter, Chapter 5, is then used to present the conclusions of this research and to give recommendations for future work.

Chapter 2

Background

In Chapter 1, the two main topics of this thesis were introduced, namely the use case and knowledge representation through ontologies. Before discussing the ontology that was developed in this final thesis project, some background on both these topics is required. This chapter provides this background for the use case in Section 2-1 and for knowledge representation through ontologies in Section 2-2.

2-1 Use case

The relevant background for the use case concerns Autonomous Underwater Vehicles (AUVs), Naval Mine CounterMeasures (NMCM) operations and the use of AUVs in NMCM search operations. Since this research was carried out at the Netherlands Organisation for applied scientific research (TNO), the AUV that is used at TNO, together with the operating procedures that were developed for it, was chosen as the vehicle to focus on for the use case. The goal of studying the use case was to get an idea of the concepts that are relevant for it and hence should be included in the ontology for this final thesis project.

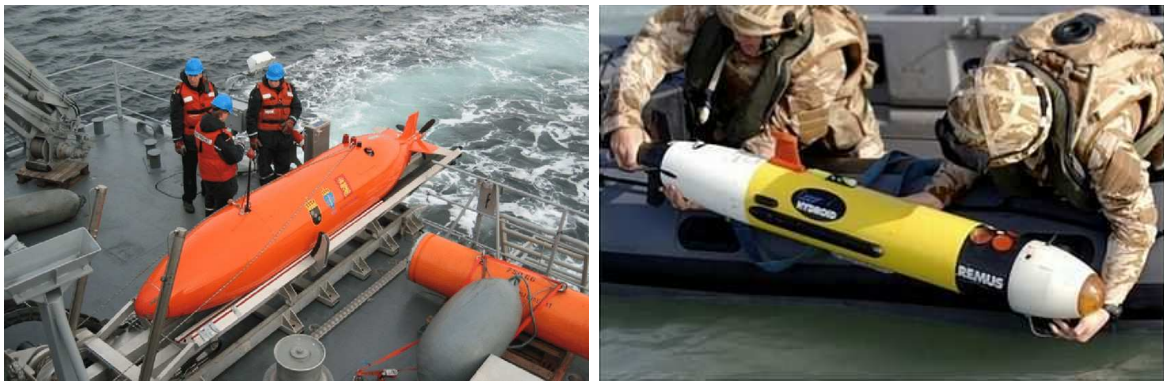
2-1-1 Autonomous Underwater Vehicles and the LAUV

AUVs are autonomous mobile robots that can operate underwater and swim freely. They are part of the family of Unmanned Underwater Vehicles (UUVs), which also includes Remotely Operated Vehicles (ROVs). An important difference between ROVs and AUVs is that ROVs are controlled by human operators and are therefore connected to a host ship through an umbilical cable, which also provides power to the ROV, whereas AUVs operate independently of continuous human control and carry their own power source. In general, ROVs and AUVs are used for different applications. ROVs are suited for tasks that require some sort of manipulation, because they are controlled by a human operator and can be supplied with sufficient power by the host ship. Examples of ROV applications are construction, maintenance and inspection of underwater structures or retrieval of objects on the seabed. AUVs, however,

are mostly used for underwater survey tasks because they can autonomously cover large areas of the seabed, also at very large depths, and can be equipped with advanced imaging sensors such as side looking sonars. Applications of AUVs include oceanographic research, hydrographic surveying and searching for downed aircraft. [1, 16, 17]

The underwater environment poses some major challenges that need to be taken into account in the design and the deployment of AUVs in order to enable them to function autonomously. The most prominent challenge is the poor propagation of electromagnetic waves underwater. This makes the methods that are commonly used above water for communication (e.g. VHF radio), navigation (e.g. GPS) and perception (e.g. video camera) unsuitable for the underwater domain. Acoustic signals, however, propagate better in water than in air, which is why sonar systems are often used for sensing and navigation and why communication is also often done acoustically. Then again, the underwater environment is not ideal for acoustic propagation either, due to for instance the low propagation speed compared to the speed of light above water and the distorting effects of phenomena such as refraction, reflection and scattering. In addition, the propagation of acoustic signals is highly dependent on several underwater conditions (such as temperature, pressure and salinity), which are subject to change. These changing underwater conditions can also affect an AUV's navigation performance, for instance due to changing underwater currents. Another challenge is that only a very small part of the seabed has been mapped and the seabed is also subject to change, so it is often not possible to use underwater maps for navigation. A third major challenge is that AUVs have to carry their own power source, which limits their endurance. These challenges indicate that the tasks that an AUV can execute and the performance it can achieve depend on the equipment it has and on its level of autonomy, which determines the extent to which the AUV can adapt to unknown and changing circumstances. [1, 18, 19]

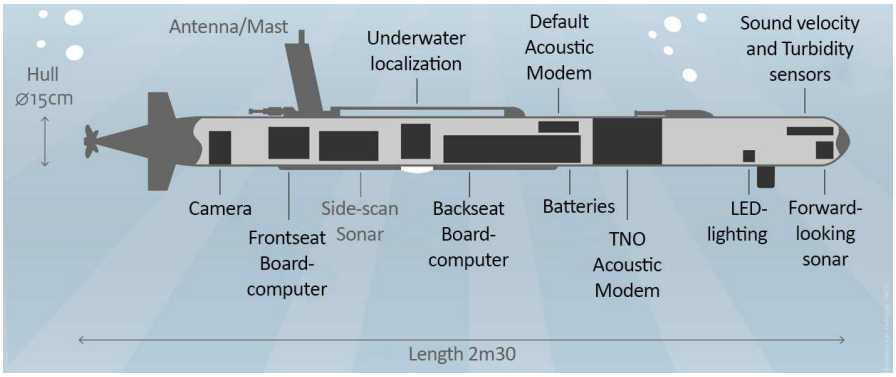
AUVs are often torpedo shaped and propelled using a propeller combined with actuated fins. They come in many different sizes, with weights ranging from a few kilograms to a few tonnes. Examples of AUVs that are used for NMCM operations are the HUGIN 1000 AUV, which is a large AUV used by the Norwegian navy, and the REMUS 100 AUV, which is a man-portable AUV used by the Dutch navy amongst others. These two AUVs are shown in Figure 2-1a and 2-1b respectively.



(a) HUGIN 1000 AUV [20] **(b)** REMUS 100 AUV [21]

Figure 2-1: Two examples of AUVs that are used for NMCM operations

(a) Exterior of the LAUV [23]

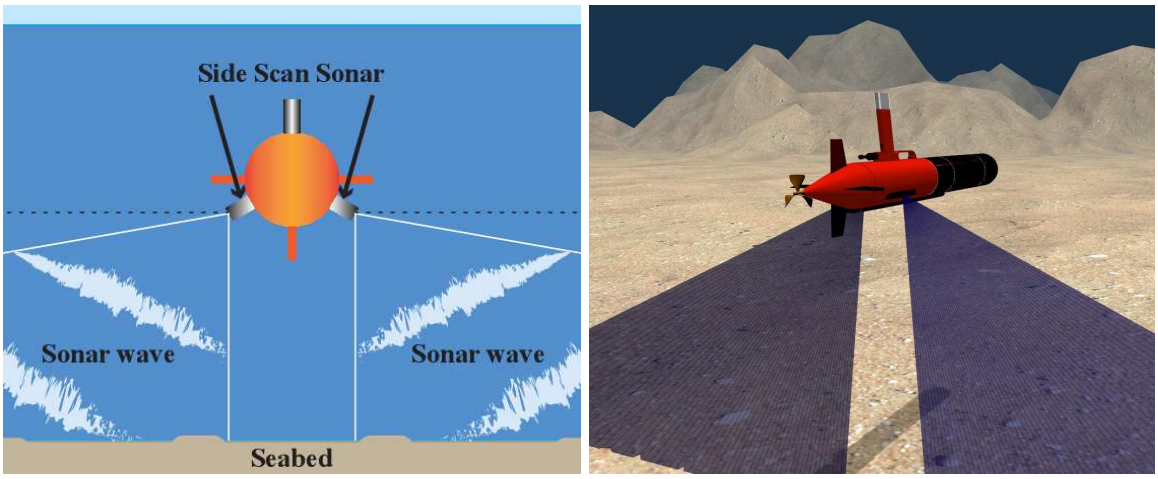


(b) Cross section of the LAUV [24]

Figure 2-2: Overview of the LAUV

The AUV used by TNO is the Light Autonomous Underwater Vehicle (LAUV), which was developed by OceanScan - Marine Systems & Technology. The LAUV was designed to be a lightweight, easily launchable vehicle with a modular design and with hardware and software architectures that are open for the user to implement own sensors and algorithms [22]. This open architecture makes it possible to use the LAUV as a test vehicle for experimenting with different methods and algorithms for instance for planning, navigation and data processing. In Figure 2-2, a picture of a LAUV's exterior and a schematic cross section of the LAUV of TNO are shown.

The LAUV of TNO is equipped with several sensor systems that are required for performing a search operation, as is illustrated in Figure 2-2b. It uses GPS for navigating above water and is equipped with an Inertial Measurement Unit (IMU), combined with a Doppler Velocity Log (DVL) and magnetometers, for underwater navigation. The IMU measures the accelerations and angular rates of the vehicle, which can be integrated to obtain the vehicle's position and orientation (together referred to as the vehicle's pose). Measurements of the vehicle's speed relative to the seabed, provided by the DVL, and of the vehicle's magnetic course, provided by the magnetometers, are used to improve the estimates of the vehicle's pose. A Forward Looking Sonar (FLS) is used for detecting obstacles underwater. In addition, the LAUV has three imaging sensors that can be used for searching the seabed, namely a Side-scan Sonar (SSS), suited for long range imaging, and a Multi-Beam Echo Sounder (MBES) and camera that can be used for high resolution (shorter range) imaging. [25]



(a) Schematic rear view of an AUV with SSS [26] **(b)** LAUV with SSS in a simulation environment

Figure 2-3: Side-scan Sonar principle

In Figure 2-3a, a schematic rear view of an AUV with an SSS is shown. An SSS system on an AUV emits sound waves (sonar waves in Figure 2-3a) to the sides of the vehicle and then receives the echoes of these waves resulting from reflection on the seabed or objects on the seabed. These echoes can then be used to create an image of the seabed as the AUV moves over it, which is why the SSS is suitable for searching an area of the seabed. Figure 2-3b shows a visualisation of the range of the sound waves transmitted by an SSS system mounted on a LAUV in a simulation environment.
An MBES makes use of the same working principle of transmitting and receiving sound pulses, but the beam form of the emitted sound pulses, and the range and resolution of the system are different. On an AUV, the MBES is generally located on the bottom of the vehicle, so the emitted sound pulses are directed towards the seabed instead of to the sides of the vehicle, as is the case for the SSS system. In addition, the sound pulses are emitted in a narrower beam and the range for detecting objects is smaller, which makes the MBES more suitable for taking close-up images than for scanning a large area of the seabed. [27]

2-1-2 Minehunting operations and the use of AUVs

In general, the goal of a NMCM operation is to clear a certain area of the sea of (naval) mines. In NMCM three methods exist for clearing mines: minesweeping, minehunting and Explosive Ordnance Disposal (EOD). In minehunting a large part of the operation consists of searching the seabed and water volume for mines using sensor systems, which is a task that can be performed by AUVs such as the LAUV. [4]
In a traditional minehunting operation, performed with a minehunting vessel, four phases can be distinguished. In the detection phase the vessel is navigating a pattern while using a detection sonar to detect objects. Once an object is detected a classification sonar is used to determine whether the detected object is likely to be a mine (classification phase). If so, another method is used to determine whether the object actually is a mine (identification phase). Identified mines are then detonated or first displaced and then detonated (neutralisation phase). Identification and neutralisation can be done by divers or using ROVs deployed

from the minehunting vessel. When AUVs are used in a minehunting operation, the approach of the operation is generally different. First a full search of the area is performed by one or more AUVs and subsequently the detected objects that are likely to be mines are revisited and identified, using divers, ROVs or AUVs. When using divers or some types of ROVs, identified mines can also immediately be neutralised. When AUVs are used, however, the identified mines have to reacquired another time for neutralisation. [28, 14]
The LAUV of TNO is equipped with sensors that are suitable for long range imaging, and thus for searching objects, and with sensors for high resolution imaging, which can be used for identification. Therefore, in this thesis the assumption is made that the LAUV performs both the search task and the identification task, so the minehunting operation is divided into three phases [28]:

- **Search, classification and mapping** - In this phase, the LAUV navigates a pattern over the area that is to be searched, in most cases a lawnmower pattern (parallel tracks), while imaging the seabed with its SSS. The images are processed on board to detect and classify objects on the seabed and the estimated locations of these objects are saved.
- **Reacquisition and identification** - In this phase, the contacts that resulted from the first phase are revisited by the LAUV and imaged with its camera or MBES. The resulting images are processed to determine whether the reacquired contacts are mines.
- **Reacquisition and neutralisation** - In this phase the mines that were identified in the images obtained in the previous phase are revisited and neutralised. This phase will not be considered in this thesis since it cannot be performed by the LAUV.

Control and planning in the LAUV for minehunting operations

At TNO, a software architecture was developed for the LAUV, which enables it to plan and execute the search and identification tasks described above. This software architecture is described in [25] In this architecture, the TNO developers have differentiated between three levels of abstraction, namely the action level, task level and mission level (from low to high abstraction level). As shown in Figure 2-2b, the LAUV is equipped with two on-board computers: the frontseat and backseat board computers. The frontseat board computer runs software developed by the manufacturer of the LAUV and is used for control the actions of the vehicle. Actions are lowest level activities that the LAUV can perform as defined by the manufacturer. An example of an action is navigating to a waypoint or switching a sensor on or off. The backseat board computer runs the software that was developed by TNO and is used for mission and task planning and control. In general, the backseat computer performs mission and task planning by dividing a mission into a schedule of tasks and a task into a schedule of actions. Using these schedules, the backseat computer then requests the frontseat computer to perform the planned actions and monitors this execution using the information that is sent back by the frontseat computer. The backseat computer can also coordinate a mission for multiple vehicles. One of the vehicles is then assigned as the master and manages the mission by assigning parts of the mission or separate tasks to the different vehicles. Communication between the vehicles can be done acoustically when under water and via WiFi when above water. The different missions and tasks that the LAUV can perform are defined in [25]. The two main tasks that are defined for the LAUV correspond to the search task (search, classification and mapping) and the identification task (reacquisition and identification) that were defined above. [25]

Factors influencing AUV minehunting performance

As mentioned in Chapter 1, the duration and the effectiveness of an NMCM operation are important measures that describe the achieved performance. The overall effectiveness of an NMCM operation can be characterised by the percentage clearance, which is the percentage of mines in the operation area that have been successfully neutralised [6]. In a minehunting mission the achieved performance in the different phases that were described above contributes to the overall performance.

Search, classification and mapping phase For the search, classification and mapping phase, the performance measures that are important for the overall performance are the duration and the probability of correct detection and classification. This probability represents the chance that a mine like object in the operation area is detected and correctly qualified as a contact that should or should not be revisited in the identification phase. The two factors that mainly determine this probability are the percentage of the search area that is actually scanned by the AUV (coverage percentage) and the probability that an object in the scanned areas is correctly detected and classified. This coverage percentage indicates how much of the search area has been within the range of the SSS in the search phase and depends on several factors, such as the type of path that was planned to cover the search area, the navigation accuracy and the range of the SSS system.
Other relevant performance measures for this phase are the false alarm rate, which gives an indication of the number of contacts that will be revisited in the identification phase but are likely to be identified as being non-mine, and the accuracy of the location estimate of the classified contacts.

Reacquisition and identification phase The last two measures mentioned in the previous paragraph influence the total duration of the reacquisition and identification phase, since a high false alarm rate in the search phase means that more contacts than necessary need to be revisited for identification and a low accuracy of the location estimate of a contact means that it will take longer to reacquire this contact. The resulting duration of the reacquisition and identification phase again influences the overall duration of the minehunting operation.
A performance measure indicating the effectiveness of the reacquisition and identification phase is the probability of correct identification, which depends on the probability that a classified contact is reacquired and imaged and the probability that it can be identified from these images. For this phase also, the false alarm rate (indicating the number of classified objects incorrectly identified as mines) and the accuracy of the location estimate of the identified objects are relevant performance measures that influence the performance in the last phase (reacquisition and neutralisation). [6, 29]

Measuring or estimating the performance measures mentioned above is quite complicated because it involves unknown parameters, such as the actual number of mines in a real operation area, and because there are a lot of interdependencies between these measures and other influencing factors. These influencing factors can be divided into two categories, namely external factors, which are related to the environment in which the operation takes place, and internal factors, which are related to the vehicle itself. The external factors include factors

related to the underwater conditions, seabed properties and target properties. The internal factors include factors related to sensing, processing, actuation and planning. Examples of these factors are shown in Tables 2-1 and 2-2, which are not intended to give a complete overview of all factors influencing performance. These tables were drawn up using [6, 29, 30].

Table 2-1: Examples of external factors

External factors	
Underwater conditions	Temperature Salinity Clarity Currents
Seabed properties	Sediment type Clutter Ripples Vegetation
Target properties	Mine type Mine size Burial

Table 2-2: Examples of internal factors

Internal factors	
Sensing aspects	Sensor configuration Sensor quality Sensor settings
Processing aspects	Sensor fusion Data processing algorithm quality
Actuation aspects	Operation speed and height Navigation smoothness
Planning aspects	Coverage planning Mission and task planning

Since one of the goals that was set for the ontology was to be able to infer how well a vehicle can perform a certain task, it is important to include the factors that can influence this task performance in the ontology. Knowledge about these factors and the way in which they can influence the performance of an AUVs on a task can be very useful for a planning system that should plan this task or the corresponding mission. Chapter 4 will discuss how these performance aspects were included in the developed ontology and will illustrate how some of the factors shown in Tables 2-1 and 2-2 can influence the performance of the LAUV.

2-2 Ontologies

Ontologies are used for representing semantic knowledge, which can be useful for autonomous vehicles. In this section, some theory on ontologies is given and some examples of ontologies in autonomous robotics are shown. Furthermore, some methods for developing ontologies and for implementing them are touched upon.

2-2-1 Ontology theory

A useful definition for ontologies is given by Guarino et al. in [31]: “Computational ontologies are a means to formally model the structure of a system, i.e., the relevant entities and relations that emerge from its observation, and which are useful to our purposes." The word ‘computational’ is used here to distinguish it from Ontology in the philosophical sense, which can be explained as the study of ‘being’ and ‘reality’ [31].

The main components of an ontology are concepts, relationships and instances, of which the concepts are organised in a hierarchy or taxonomy. A concept is a generalization of a set

of things in a domain and concepts can be connected through relationships. The concepts and relationships define the structure of the ontology and indicate what kind of things can exist in a domain. Instances are the individuals that are represented by a concept and thus can be seen as specific realisations of this general concept. The instances form the population of the ontology and show what things and relationships actually exist in the domain. An ontology can be represented in the form of a graph in which the nodes represent concepts or instances and the edges represent relationships. Other ontology components that are often mentioned are axioms and attributes. Axioms are statements that define and constrain concepts and relationships. Attributes are used to describe properties of concepts and relationships. [8, 12, 31, 32, 33]

The structure of the ontology (concepts and relationships) can be compared with the schema of a database in the sense that they both prescribe what kind of instances or data can exist. However, the reason of existence of an ontology structure and of a database schema is different. A database schema is meant for structuring data in order to store and query it efficiently, whereas an ontology structure focuses on meaning and exists to create understanding of a domain [34]. This thesis is concerned with ontologies rather than databases and to make a clear distinction between the structure and the population of an ontology, the term 'schema' will be used to refer to the concepts and relationships of the ontology and the term 'data' will be used to refer to the instances, even though this terminology is theoretically inaccurate, as was explained above.

2-2-2 Ontologies in autonomous robotics

As mentioned in Chapter 1 ontologies are widely used in the robotics community. The best known example probably is the KnowRob knowledge processing infrastructure for autonomous service robots by Tenorth and Beetz [35], which includes ontologies for semantically describing robots and their environment [36, 37]. In the field of AUVs ontologies have also already been used, for instance in the PANDORA project, where ontologies were used to describe the external world of the AUV and to describe the vehicle's capabilities [11]. Another example is the work of Cashmore et al. in which an ontology is used to organise knowledge about the external world in order to support the vehicle's planning process [38]. A third example is the SWARMs project, where a set of ontologies was developed to facilitate cooperation between different kinds of maritime robotic vehicles [15]. An interesting aspect of the Smart and Networking Underwater Robots in Cooperation Meshes (SWARMs) ontology is that it can incorporate uncertainty by using constructs of the PR-OWL ontology, proposed by Carvalho et al., which provides means for representing probabilities in ontologies [15, 39, 40].

The work done by Migueláñez et al. [8] and by Papadimitriou et al. [14, 41] is especially interesting because they use ontologies for an NMCM use case and also consider the planning aspect. Migueláñez et al. developed a set of ontologies to improve the Situation Awareness (SA) of AUVs. They came up with an overarching core ontology containing concepts such as 'platform', 'payload' and 'capability' and two application ontologies specifically for status monitoring and for mission planning [8]. Papadimitriou et al. focus on adaptive mission planning and make use of the KnowRob system for the implementation of their ontologies. They developed three ontology sets related to the external world, the vehicle itself and the planning aspect respectively [14, 41].

An argument for using ontologies that is often mentioned is that they enable the exchange of knowledge between systems and humans and that they are reusable. In addition, some definitions for ontologies mention that ontologies should represent a shared conceptualisation of a domain [8, 12, 32]. In view of this, the efforts of an IEEE working group to develop a common ontology for the field of robotics and automation are very relevant. The goal of this Ontologies for Robotics and Automation Working Group (ORA WG) is to develop a standard to provide an overall ontology and associated methodology for knowledge representation and reasoning in robotics and automation [42]. The first resulting standard was released in 2015 and consists of the Core Ontologies for Robotics and Automation (CORA) [42].

The upper ontology called Suggested Upper Merged Ontology (SUMO) was used as a basis for the CORA. In Figure 2-4 an overview of the main concepts of the CORA and their relations to SUMO concepts (in gray) is shown. The arrows in this figure represent subclass relationships. In the standard that contains the CORA also some ontologies that extend the CORA are presented, namely CORAX, which focusses on robot design, interaction and environment, RPARTS, which focusses on robot parts, and POS, focussing on robot position, orientation and pose concepts. [10, 33, 43]

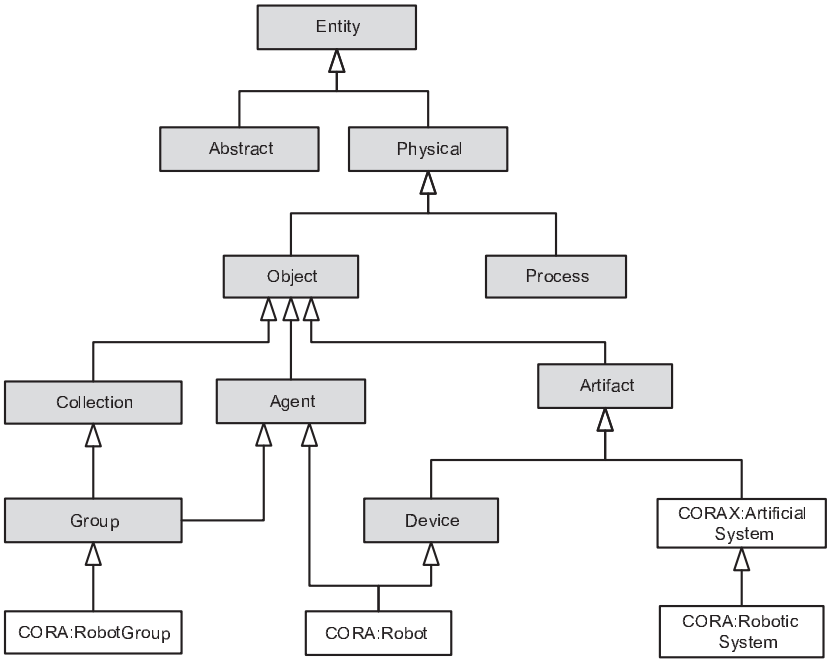


Figure 2-4: Overview of the main concepts in CORA: robot, robot group, robotic system [43]

As Figure 2-4 shows, the concepts in the CORA are still quite general, which is why two subgroups of the ORA WG are working on extending the CORA [42]. These two subgroups are the Robot Task Representation Subgroup and the Autonomous Robot Subgroup. The first subgroup is developing a Task Ontology, which is very relevant for this thesis because the concept of a task is important for planning an operation and thus is expected to play an important role in an ontology focussing on planning. The main concepts and relationships of the Task Ontology under development are shown in Figure 2-5 [44].

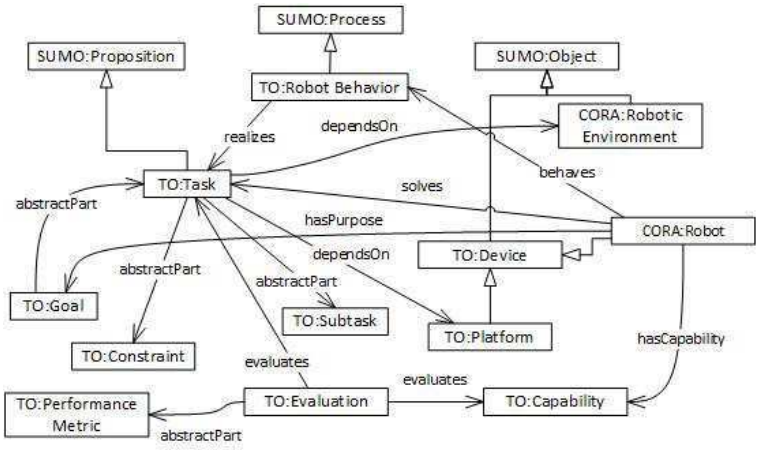


Figure 2-5: Initial structure for the Task Ontology [44]

Since the CORA and related ontologies are meant to serve as a standard for the robotics and automation field, the goal was set to use these ontologies as a starting point for the development of an ontology for the use case considered here. In addition, this standard illustrates the usefulness of broadly applicable ontologies, so keeping the ontology as general as possible was set as another goal for the ontology development process. In light of this, it is worthwhile to look into some ontology development methods, which will be done in the next subsection.

2-2-3 Ontology development methods

The ontology development method that was used for developing the CORA is called METHONTOLOGY and is described in [10]. The ontology development process of METHONTOLOGY includes three types of activities, namely project management activities (e.g. planning), development-oriented activities and support activities (e.g. knowledge acquisition). The development-oriented activities are:

- Specification of the purpose, scope and intended usage of the ontology.
- Conceptualization of an informally perceived view of a domain to obtain a conceptual model of the ontology.
- Formalization of the conceptual model.
- Implementation of the formalized ontology in an ontology representation language.
- Maintenance of the developed ontology.

Figure 2-6 gives an overview of the ontology development process of METHONTOLOGY including all activities of the three types mentioned above. [33, 10]

Another interesting method is Robot Standard Ontological Development Life Cycle (RoSaDev), which a few of the authors of the CORA standard helped develop. A visualisation of this life cycle is shown in Figure 2-7. Compared to METHONTOLOGY, RoSaDev includes similar activities such as conceptualization and formalization, but RoSaDev also explicitly focusses on validation of the developed ontology and on integration into the standard ontology set. The method also proposes approaches for performing the four different steps of RoSaDev.

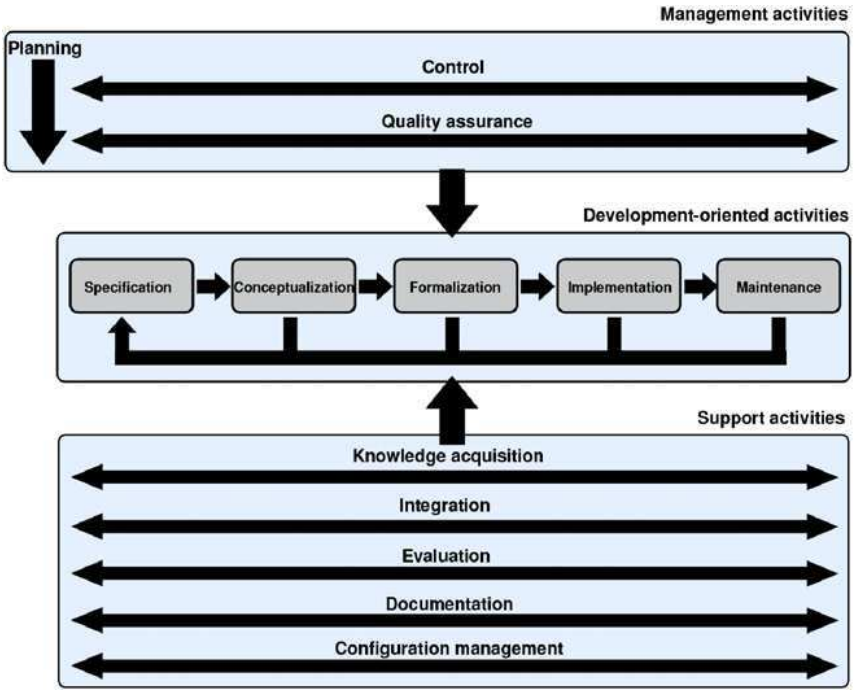


Figure 2-6: Ontology development life cycle of METHONTOLOGY [10]

For the first two steps, which consist of identifying, developing and formalizing concepts, a collaborative approach is proposed since the goal is to develop a shared conceptualization of a certain domain. A middle-out approach is advised for developing use cases and validating the developed conceptualization. This validation step then should lead to integration into the standard in an incremental way. Finally, the four steps of the RoSaDev development process should be performed iteratively to obtain a comprehensive ontology that is integrated into the standard. [45]

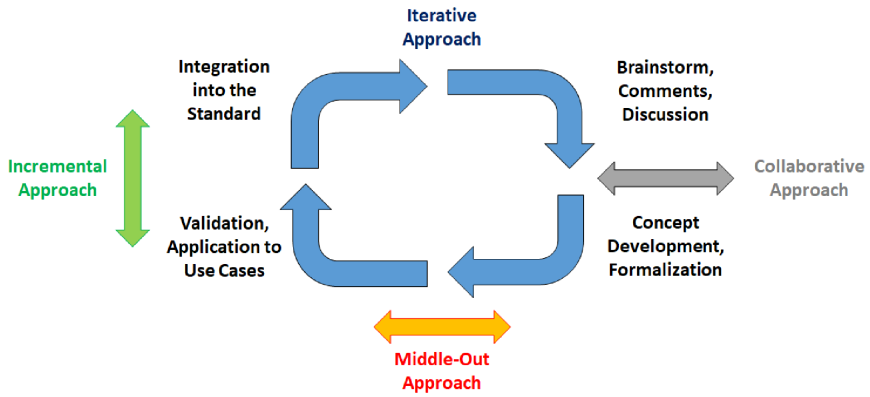


Figure 2-7: Robot Standard Ontological Development Life Cycle [45]

2-2-4 Ontology implementation

To be able to use ontologies in an actual autonomous system a method for implementing them in the system is required. There are several ontology languages and tools as well as some database tools that can be used to implement ontologies, of which a few relevant ones will be discussed here.

According to Corcho et al. [46], ontology languages can be divided into two groups, namely classical ontology languages that evolved from existing knowledge representations languages, based on logics, and ontology markup languages (also called web-based ontology languages) that are based on existing markup languages such as HTML and XML. This first group of languages is hardly used anymore, but some of the second group are and the most commonly used one is the Web Ontology Language (OWL), which is based on XML.
OWL is developed for the Semantic Web and is designed for representing knowledge about things and the relations between them. The Semantic Web is an extension to the world wide web with the goal to create a web of data that is machine readable and in which the meaning of data can be processed by machines as well [47]. OWL thus is machine-readable and can be used to enable computers to reason with the knowledge in an ontology. An important reason for the development of OWL was to facilitate ontology development and sharing via the web. Several tools exist for developing of and reasoning with ontologies expressed in OWL, such as the editors Protégé and SWOOP and the reasoners Fact++ and Hermit. [48, 49]

Apart from these tools and languages that are intended for implementing ontologies, some database tools can be used for this purpose as well [50]. Especially tools for graph databases can be suitable because graphs are a means to represent ontologies. Examples of graph database management systems are Neo4j, OrientDB and GRAKN.AI [51]. Graph database management systems can have different underlying data models, of which property graphs, hypergraphs and triples are the most common ones [52]. Property graphs consist of nodes and directed relations between nodes, which can both have properties. Hypergraphs also consist of nodes and relations, but these relations can connect more than two nodes, which is impossible in a property graph. Triples are used to store subject-predicate-object relations, which together form a graph of nodes and relations. Triples actually form the basis of the Web Ontology Language. [52, 53, 49]

In the literature study that was done for this thesis, the different available languages and tools for implementing ontologies were reviewed and compared and the choice was made to use GRAKN.AI, hereafter referred to as Grakn, as the implementation tool for this thesis. Grakn is an open-source graph database management system developed specifically for Artificial Intelligence (AI) applications and first released in 2016 [51]. The underlying data model of Grakn is a labelled hypergraph [54]. Grakn makes use of schemas to structure the data in a graph database, or knowledge graph as it is called in Grakn, which enables the implementation of an ontology as a graph database in Grakn. The use of a schema makes it possible to perform automated reasoning over the knowledge in a knowledge graph. Grakn has its own querying language, called Graql, and Graql queries can be used to retrieve or modify information stored in a knowledge graph. In addition, Grakn provides analytics capabilities, allowing users to perform statistics queries of numeric resources (e.g. mean or standard deviation) and graph queries (e.g. shortest path between nodes). [55]
The decisive reasons for choosing Grakn were that it enables the implementation of an ontology in a flexible and extendible way, it facilitates reasoning with and querying of the data

in an ontology and it is easy to learn and work with. In addition, Grakn allows for working with large data sets and is compatible with the software that is used at TNO. The only major disadvantage of Grakn is that it does not conform to any semantic standards, making it harder to share knowledge with systems that do not use Grakn and making it impossible to directly use existing ontologies implemented in OWL.

Chapter 3

Developed ontology

A large part of the work done in this final thesis project consisted of the development of an ontology. This chapter describes the process of developing this ontology, in Section 3-1, and the resulting ontology itself, in Section 3-2, except for the ontology elements related to performance, which will be described in Chapter 4. In Section 3-3, the implementation of the developed ontology in Grakn for inferring a vehicle's capabilities will be discussed, focussing on how querying and reasoning is used to achieve the goal that was set for the ontology.

3-1 Ontology development process

The process of developing an ontology turned out to be very iterative and non-linear, which corresponds to how the development process is presented in the methodologies that were described in Chapter 2 (Figures 2-6 and 2-7). The combination of taking a conceptual point of view and an application or implementation oriented point of view proved to be invaluable for making progress in this process. Additionally, feedback from experts with different backgrounds helped to make the ontology more comprehensive and intuitive. In this section, the development process is described, using the development methodologies described in Chapter 2 as a reference.

3-1-1 Specification of the purpose and scope for the ontology

Initially, the domain for the ontology that was to be developed was quite broad, namely planning for an Autonomous Underwater Vehicle (AUV) in Naval Mine CounterMeasures (NMCM) operations. The first step in the development process therefore was to find a more specific domain for developing an ontology and implementing knowledge. This step can be regarded as part of the Specification phase in the METHONTOLOGY method. The goal of this phase was to specify the domain for which the ontology would be developed by finding a set of main keywords describing this domain. In addition, this phase was meant for determining the intended usage of the ontology.

To obtain an overview of the broad domain, a set of sources selected for their relevancy for the use case was studied and used to draw up a list of relevant concepts. This set consisted of the following sources:

- A document containing a concept message set for autonomous systems operating MCM missions [56].
- A document describing the software architecture designed and used at TNO for the LAUV software [25].
- The IEEE Standard Ontologies for Robotics and Automation [33].
- An article by Balakirsky et al. describing the initial efforts in developing a robot task ontology as an extension to the CORA [44].
- An article by Migueláñez et al. about using an ontology framework for improving the situational awareness of AUVs, applied in an MCM use case [8].
- The ontologies that were used in the PANDORA project, with the aim to achieve persistent autonomy in AUVs, as described in [57] and [11], which are based on the KnowRob structure, described in [35].
- An article by Li et al. describing the SWARMs ontology, developed to support cooperation of underwater robots [15].
- Two articles by Papadimitriou et al. about using ontologies for adaptive mission planning and fault recovery for AUVs in MCM missions [14, 41].
- An article by Hongfei et al. about using ontologies for situational awareness in Unmanned Underwater Vehicles [58].

Based on these sources a list of concepts that are relevant for autonomous mobile robots in general and for the domain of search missions with AUVs in particular was drawn up. This list was unstructured and contained concepts for different levels of abstraction, such as "waypoint", "pressure sensor", "mission plan", "robot part" and "water salinity".

Next, the most relevant concepts from this list were selected following a bottom-up approach; the sources that relate to the domain on the lowest conceptual level, in this case sources that concern the actual application of AUVs in NMCM missions and the components, software and algorithms used for this, were used to decide which concepts to select. This approach was chosen because looking at the actual application gives a clear indication of what concepts are actually necessary for an ontology that should be useful for this application. Therefore, the selection of concepts was based on the first two sources from the list above, which concern the structure and the deployment of the AUV used at TNO. In this process, very general concepts such as "robot group", "event" or "asset" were removed from the list and concepts specific for the use case were selected, as well as concepts that generalise a set of those specific concepts. For example, the components of the LAUV of TNO, such as "pressure sensor", "forward looking sonar" or "MBES" were selected as specific concepts, as well as the corresponding generalising concept, which was "sensor component" for the examples given here.

The resulting list of concepts still was quite unstructured, so the decision was made to categorise the concepts. Five categories were identified, namely space/time concepts, planning/mission/tasks concepts, internal world/AUV status concepts, external world concepts and other concepts. This categorisation was inspired by the way in which ontologies are divided in several articles that were found during the literature study for this thesis, such as in [14, 41, 8, 15]. In these articles, often the following three areas for knowledge/understanding are identified as important for an autonomous vehicle: the environment/external world,

the mission/goal/planning aspects and the system itself (especially its capabilities). In an exploratory study by TNO about autonomous systems these three areas were also identified as important for realising understanding in an autonomous vehicle [59].
As mentioned before, the second one of these three knowledge areas, the planning aspect, was chosen to focus on, because planning is an important aspect for underwater survey missions with an AUV, especially when multiple AUVs are used. Examples of concepts that were placed in this category are the tasks of the LAUV of TNO ("search task", "identification task"), but also more general concepts such as "mission plan", "goal" or "task performance".

Next, the list of categorised concepts was searched for the concepts that were most relevant for planning, while also looking for possible connections between these concepts, in order to find a set of main concepts that could be used to start building the ontology. The result of this search that was considered most promising for further research was the connection between AUV *components*, *capabilities* and *tasks*, which actually also concerns the last of the three knowledge areas (the system itself). A similar connection was found to be useful for developing ontologies for representing knowledge in an AUV in the article by Papadimitriou, Saigol and Lane. The connection between *components* and *capabilities* was also used by Migueláñez et al. in an ontology to aid mission planning [8]. Additionally, the concept *task* is receiving a lot of attention in the development of extensions to the CORA, as can be concluded from the article by Balakirsky et al. about the developing a robot task ontology [44]. This connection between *components*, *capabilities* and *tasks* is relevant for planning because planning a mission requires drawing up a schedule of tasks, such that executing the resulting combination of tasks leads to achieving the mission goal. Furthermore, the available vehicle(s) should be able to perform these tasks, which requires them to have some specific capabilities, and what capabilities a vehicle can have depends on the components it is equipped with. Considering this, the idea for developing the ontology matured and the goal was set to be able to use the ontology to automatically infer whether an AUV could perform a given task or mission based on the components it is equipped with. Additionally, it was considered useful to include performance aspects in the ontology, because often a certain level of performance is desired for a mission or a task, which should be taken into account when planning this mission or task.

The first of the three knowledge areas (environment/external world) was not considered for the initial development of an ontology because the expectation was that for this area proper identification and definition of concepts is quite complex. This expectation stemmed from the fact that there are many methods for modelling the environment of an AUV and that this environment is also uncertain and subject to change. It is, however, undeniable that environmental concepts are relevant for planning, for instance considering that the time a mission takes and the performance that can be achieved are highly dependent on the size and layout of the area of operation and the prevailing underwater conditions.

In summary, this initial specification phase resulted in a focus on the connection between AUV *components*, *capabilities* and *tasks*, and in a list of concepts relevant for planning. In addition, the goal was set for the ontology to be able to use it to automatically infer a vehicle's capabilities based on the components it is equipped with. The LAUV was selected as the example vehicle for the ontology.
A common way to indicate the intended purpose and scope of an ontology is to define a set of competency questions (CQs), which are questions in natural language that the ontology should be able to answer [60, 61]. Therefore, a (non-exhaustive) list of competency questions has been defined for the ontology, which is shown below.

Competency Questions:

Q1. What kind of components can the LAUV have?

Q2. Which tasks are necessary for an NMCM search operation?

Q3. What capabilities can an AUV that performs NMCM search operations have?

Q4. Which capabilities are necessary for executing the tasks in an NMCM search operation?

Q5. With which components should the LAUV be equipped in order to have a certain capability?

Q6. How can the performance of an AUV on a certain task be described?

Q7. Which factors influence task performance?

The last two of these competency questions will not be addressed in this chapter, but are covered in Chapter 4. The next subsection will discuss how the results of the specification phase were used to develop a conceptual model of the domain, resulting in the schema of the ontology, which is described in Section 3-2.

3-1-2 Conceptualisation of the domain and validation

Developing a conceptual model of the domain and formalising this model to create an ontology constitute the second and third set of development-oriented activities discerned in the METHONTOLOGY method [10]. The combination of these activities is also mentioned in the RoSaDev (see Figure 2-7) [45].
To develop a conceptual model from the results obtained in the initial specification phase, some effort was put in finding the dependencies between components, capabilities and tasks within the list of relevant concepts that were found before. The document describing the software architecture for the LAUV of TNO [25], which was explained briefly in Section 2-1, served as inspiration for identifying the components and tasks concepts, since this document describes all software components and lists all tasks that are defined for the LAUV. Based on these tasks, a list of capabilities a LAUV must have to perform these tasks was drawn up. In addition, the LAUV components (as found in [25], [22], [62] and [23]) served as inspiration for determining which capabilities the vehicle can possess and which components or set of components can provide these capabilities. This process resulted in a list of capabilities and components for AUVs in NMCM missions. The resulting list was then used to set up a schema for an ontology, also using the article by Balakirsky et al. [44] and incorporating advice by TNO experts.
At this point, it useful to again mention the distinction between the ontology schema and data, which was explained in Section 2-2. In the process of modelling the domain and extracting the concepts and relationships for the ontology schema, the level of detail turned out to be decisive. The goal for the ontology schema was to keep it quite general, such that it is also applicable to other (autonomous mobile robot) use cases than the one considered here. On the other hand, the ontology needs to be detailed enough to be valuable for this specific use case, so it needs to be suitable for answering the CQs defined above. The approach that was used to achieve a balanced level of detail, was to extract general concepts

and relationships for the ontology schema from the use case of this final thesis project and then verify whether these are general enough by applying them to other use cases. Examples of such general concepts are `component` or `task`. These concepts are generalisations of concepts such as "Side-scan Sonar" or "identification task", which are specific for the use case, but they can also be used for another use case, such as an autonomous car that has to do a transportation task (`task`) using a camera (`component`) for imaging its surroundings. Using these general schema concepts, ontology data was then developed for the use case, to test whether the ontology is detailed enough to be valuable for the use case. In the example given here, the "SSS" and the "identification task" thus are instances of `component` and `task` concepts respectively. Developing ontology data for validating the schema once again was an iterative process and some important considerations of this process will be discussed in the remainder of this chapter. This process of validating the developed ontology based on use cases is also mentioned in RoSaDev as a phase that follows the phase of concept development and formalisation [45].

The initial idea for the ontology schema, based on the components, capabilities and tasks that were found, is represented in Figure 3-1. The red rounded rectangles and green arrows in this and all other figures showing schemas in this thesis represent concepts and relationships respectively. The figure shows the dependency of `tasks` on `capabilities` and of `capabilities` on `components` and on other `capabilities`. In addition, the concept `mission` is included, which consists of tasks and thus is connected to the `task` concept in Figure 3-1 via a *requires* relationship.

Figure 3-1: Initial basic schema

Using this basic schema, the list of capabilities that were found for the use case were organised into chains of capabilities that depend on each other, where the final capability of a chain is one that is required for performing a task. For instance, the `task` "Survey area" requires the `capability` "transiting", which itself requires other `capabilities` such as "path planning" and "localisation". These `capabilities` were made dependent of `components` (both hardware and software), such as a path planner for the `capability` "path planning" and an IMU for "localisation". In this way, a capability that is required for a task can be translated to a list of components that is required for that task, using the chain of capabilities preceding this capability and the components they require. In Figure 3-2, a visualisation of the example described above is shown.

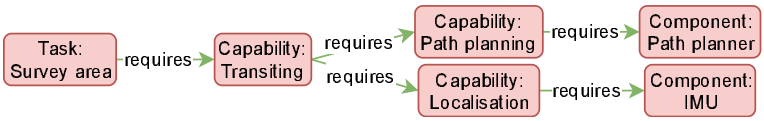


Figure 3-2: Example of instances for the initial basic schema

This basic schema of Figure 3-1, however, has two major shortcomings, which became apparent when trying to apply the schema to the use case. The first shortcoming is that some

capabilities can be realised through different (sets of) components, so modelling the dependency of `capabilities` on `components` as a direct requirement relationship in the schema is undesirable. For instance, the `capability` "propulsion" can be realised using a propeller or using a thruster, which are components with different principles of operation. The second shortcoming is that the requirement relationship between `capability` and `component` is based on the assumption that the presence of a set of components on a vehicle implies that these components can function together to achieve a certain capability. For example, the propulsion capability requires both a propeller and an electric motor (to drive this propeller), but these two components should also be mechanically connected to be able to function together. This requirement for a proper connection is not incorporated in the requirement relationships of the propulsion capability with the two individual components. Similarly, the fact that both a sensor and a sensor data processing component are present in a vehicle, does not mean that the processing component can actually receive the sensor data from the sensor, for instance because the sensor is not connected to a power source. Therefore, the assumption that just the presence of a set of components on a vehicle suffices for realising a capability entails an unacceptable simplification.
To overcome both of these shortcomings, the concept `configuration` was added to the schema in combination with input/output relationships between components. A configuration is defined as a set of connected components on a vehicle that together realise a capability. For one capability several configurations can exist, which makes it possible to model that different sets of components can realise a certain capability. The next section will further discuss the concepts and relationships that were introduced in this section.

3-2 Developed ontology

The ontology that resulted from the process described above is discussed in this section. The schema and corresponding data are explained separately in the two subsequent subsections.

3-2-1 Developed ontology schema

An overview of the developed schema is given in Figure 3-3. This figure shows the main concepts of the schema and the relationships that can exist between them. Definitions for these concepts and relationships are given in Appendix A. The overall structure of the schema will be explained below. In this explanation, some sections of Figure 3-3 will be shown in separate figures.

The concepts of the schema include the ones mentioned in the previous section: `component`, `configuration`, `capability`, `task` and `mission`. The previously described connections between these concepts are also represented in the schema: a mission consists of tasks (indicated by the *is part of* relationship in Figure 3-3), a task requires a capability (*requires capability* relationship), a capability is realised by a configuration (*is realised by* relationship) and a configuration consists of connected components (*is part of* and *IO relation* relationships). A capability was defined here as the skill or ability to perform specific behaviour supporting the ability to execute a task or mission. A component was defined as a physical or virtual element that can be part of a vehicle and fulfils a certain function that is relevant for one or more configurations.

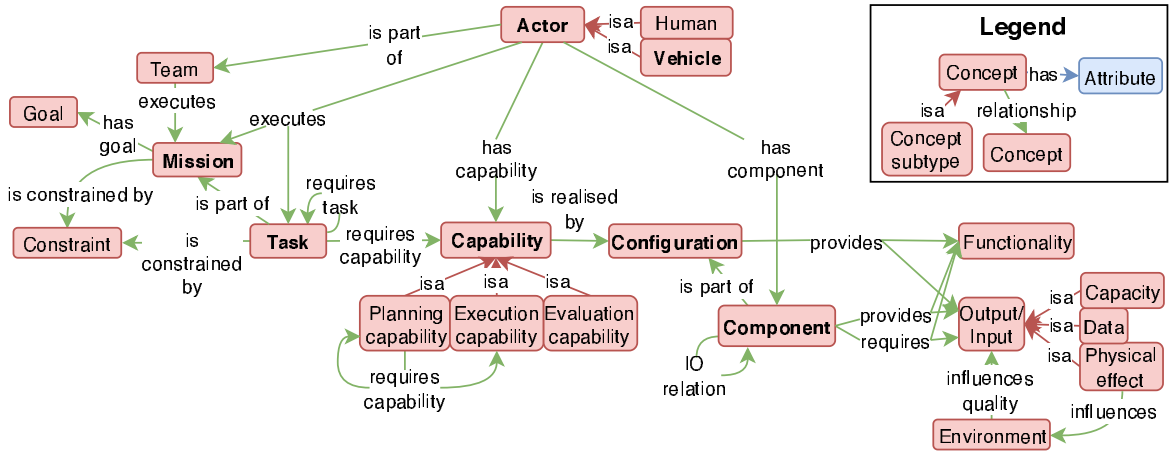


Figure 3-3: Schema overview

A concept that was mentioned in the previous section, but was not included in the initial basic schema of Figure 3-1 is `vehicle`, which is an AUV in the use case considered here. In the schema, this concept is implemented as a subtype of the concept `actor`, together with the concept `human`, as is indicated by the red arrows marked with "isa" in Figure 3-3. This implementation was chosen to be able to capture that some tasks can be executed by autonomous mobiles robots or by humans, thus by an actor. This might be less relevant for the use case considered here, but can be necessary in other autonomous vehicle applications.

`mission`, `task` and related concepts

The section of the schema in Figure 3-4 shows that an actor can have capabilities (*has capability* relationship) and can perform tasks, and thus also missions, given that it has the required capabilities (*executes* relationship). In addition, an actor can be equipped with components (*has component* relationship) and can be part of a `team` (*is part of* relationship). In this schema, a `task` is seen as an activity or set of activities that is executed by one actor (that has the required capabilities) and that is always part of a `mission`, which consists of one or more tasks. The different tasks of a mission might be executed by different actors, which are then considered to be part of team executing that mission. A mission generally has a goal that is to be achieved by executing the mission (`goal` concept and *has goal* relationship). Furthermore, missions and tasks can have constraints that limit the way in which they can be executed (`constraint` concept and *is constrained by* relationships), for instance no-go areas in a transit task. The *requires task* relationship indicates that a task can also require another task to be performed first, for instance detection of targets before identification can be performed.

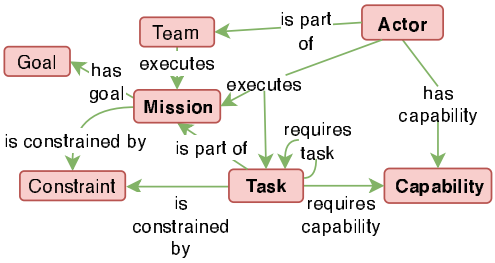


Figure 3-4: Schema section for the `task` and `mission` concepts

Subtypes of the `capability` concept

The schema section of Figure 3-5 shows that three subtypes have been defined for the `capability` concept, namely `planning capability`, `execution capability` and `evaluation capability`. An execution capability indicates the ability to perform an action that can directly be executed by a configuration of components and that does not require extensive planning, for instance the capabilities "localisation" and "contact detection and classification". A planning capability indicates the ability to plan and execute a more extensive action or set of actions. Therefore, a planning capability can also require an execution capability corresponding to one of these actions or require another planning capability for planning such an action. Examples of planning capabilities are "area survey planning" and "contact reacquisition planning". An evaluation capability indicates the ability to perform an evaluation of actions that have been or are being executed. Evaluation capabilities were added to complete the cycle of planning, execution and evaluation, but will not be further considered in this thesis.

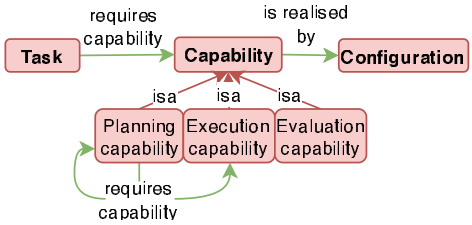


Figure 3-5: Schema section for the `capability` concept

`component`, `configuration` and related concepts

The concepts and relationships shown in the section of Figure 3-6 are essential for describing the usefulness of the `configuration` and `component` concepts. A configuration is a set of connected components that together realise a capability and components are elements that can be part of a vehicle and fulfil a certain function that is relevant for one or more configurations. Both for a configuration and for a component, its function is defined in the ontology by the functionalities and outputs it provides (*provides* relationship). The concept `functionality` refers to the purpose that a component can serve when used. Functionalities can be regarded as component level capabilities. Examples of functionalities are "electrical power supply" and "high resolution underwater imaging". The `output/input` concept is used to model that what is exchanged via IO relations between components. Examples of `output/input` instances are "sonar images" and "electrical power". Apart from providing functionalities and outputs, components can also require functionalities and inputs to be able to function (*requires* relationship). A sensor component, for example, generally requires electrical power to be able to output sensor data. Components that are part of a configuration should have all their requirements fulfilled via IO relations with other components on the vehicle.

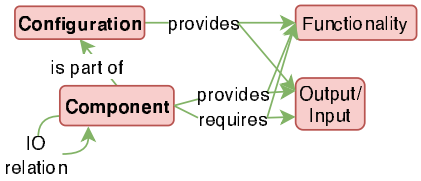


Figure 3-6: Schema section for the `configuration` and `component` concepts

Although both the `configuration` concept and the `component` concept are connected to the `functionality` concept and the `output/input` concept via a *provides* relationship, the implication of this relationship is different; components can directly provide functionalities

and outputs, whereas configurations can provide functionalities and outputs only through the components that are part of the configuration. The reason for adding the *provides* relations of the `configuration` concept to the schema was that these were used to characterise a configuration in the ontology. A more obvious choice for characterising a configuration would be to list the components that are part of it, but this approach was not used here because the purpose of a configuration is to realise a capability. Therefore, the actual composition of the configuration is not important; it only matters what kind of utility the combined components provide and this can be characterised by the set of functionalities and outputs provided by the components in the configuration. On the other hand, it is still useful to know which components are part of a configuration, which is why the *is part of* relationship between `component` and `configuration` was included in the schema.

The choice was made to use both outputs and functionalities to represent what a component or a configuration provides because using only one of both was deemed insufficient. Using only functionalities would require ignoring the fact that components might need specific data for providing their functionality, which can not be expressed as a requirement for a functionality. This is illustrated in Example 3.1.

Example 3.1. *A localisation algorithm might require its input data to be defined with respect to some specific reference frame to be able to produce a correct position estimate, so this specific data should be included as an input requirement for the algorithm.*

On the other hand, the meaning and relevance of data is not always obvious from its format and data of different types sometimes can be used for the same purpose, which makes using only data for showing what a component or configuration provides insufficient. Examples 3.2 and 3.3 illustrate this.

Example 3.2. *A list of waypoints can represent a transit path, but also a search pattern and the distinction is not clear from the data format itself. This can be solved by adding the meaning of a list of waypoints that a component produces in the form of a functionality that the component provides.*

Example 3.3. *Both an MBES and a camera can be used for high resolution imaging, but they produce data of very different formats, so it is useful to specify that both of these components provide the functionality "high resolution imaging", which is required for the capability "identification".*

Finally, a component can require a certain functionality to be able to provide another functionality or an output, whereas this component does not require an input from the component that provides the required functionality. Therefore, this requirement for a functionality can not be represented as a requirement for an input. This is illustrated in Example 3.4.

Example 3.4. *A pose control algorithm can only provide the functionality "pose control" if the functionalities "propulsion" and "steering" (actuation) are available, whereas the algorithm does not require input from the components that can provide these functionalities (for instance the propeller and the actuated fins).*

These examples also illustrate that evaluating schema design decisions using a use case can yield valuable insights, based on which the ontology schema can be improved.

Subtypes of the `output/input` concept

In the schema section in Figure 3-7, the `output/input` concept and its subtypes are shown. As mentioned before, the `output/input` concept represents that what is exchanged via IO relations and what thus is output for the transmitting component and simultaneously is input for the receiving component. An obvious type of output/input is data, such as sonar images or vehicle positions, which is thus modelled as the subtype `data` of the `output/input` concept, but resources such as mechanical and electrical power are also modelled as a subtype of the `output/input` concept named `capacity`. The last subtype of `output/input` is called `physical effect` and is added to be able to model that a component can have an effect on the environment or can obtain input from the environment. For example, the output of a component such as a propeller or fin can be modelled as a physical effect and the input of a sensor component that measures an environmental property, such as the underwater temperature, is modelled as a physical effect as well.

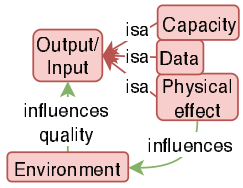


Figure 3-7: Schema section for the `output/input` concept

`environment` concept

The last concept of the schema in Figure 3-3 that was not described yet is `environment`, also shown in Figure 3-7. This concept is used to model the surroundings in which an actor executes tasks. The `environment` concept is included in the schema because some components of an actor can influence the environment or can be influenced by it. A physical effect caused by a component influences the environment (*influences* relationships) and the environment can influence sensor data, as for instance sonar data is influenced by the sound velocity underwater (*influences quality* relationship). As explained before, the focus of this ontology is not on the environment or external world, so the `environment` concept will not be expanded in the schema.

3-2-2 Identified data instances

The schema that was described in the previous subsection defines what kind of concepts and relationships can exist in the developed ontology. Instances of those schema elements were defined to obtain an ontology that is detailed enough for the use case considered here. Since the LAUV of TNO served as the example vehicle for the use case, the developed ontology data is based on this specific vehicle and the software architecture that was designed for it. As mentioned before, the process of developing the ontology was not linear and development of the schema and the data went hand in hand. The initial list of concepts that was mentioned in Subsection 3-1-1, which was mainly based on [25] and [56], was used as the basis for both the ontology schema and the data.

During the process of identifying data for the use case, it was found to be important to have a clear understanding of the idea behind developing an ontology for this particular use case. As was mentioned before, the goal here was to be able to automatically infer whether a vehicle

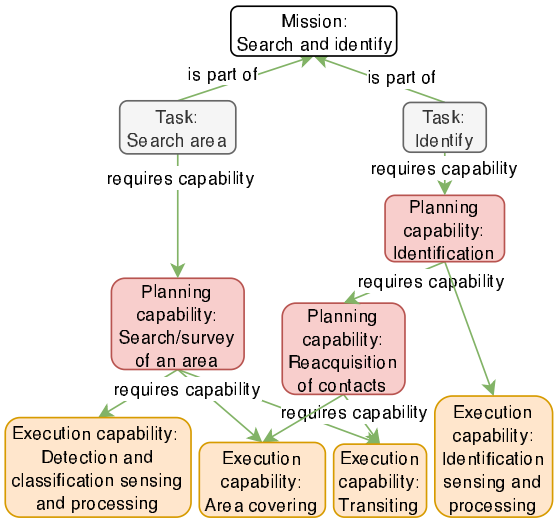


Figure 3-8: Data for the `mission`, `task` and `capability` concepts

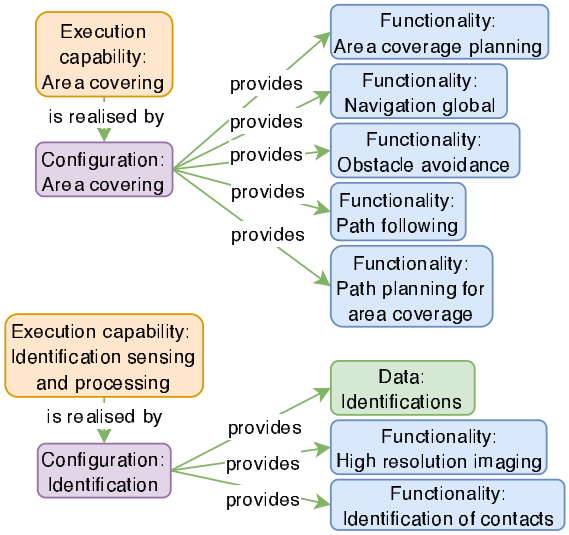


Figure 3-9: Data for the `capability` and `configuration` concepts

with a given set of components can perform a given task or mission. Being able to do this is valuable for planning tasks and missions because when a vehicle can make use of the ontology, this vehicle can be "aware" of its capabilities and thus be aware of the tasks and missions it is able to do. As a result, the vehicle's capabilities can be taken into account when planning, such that tasks can be planned more optimally.

In this subsection, a selection of the identified ontology data will be presented to give an overview of how the use case is modelled in the ontology, showing also the coherence between the ontology schema and data. The data will be presented in the form of a few cross sections of the ontology on different levels of detail. In the figures that will be used for showing these cross sections often some relationships and concepts will be left out in order to keep the figures comprehensible.

`mission`, `task`, `capability` and `configuration` instances

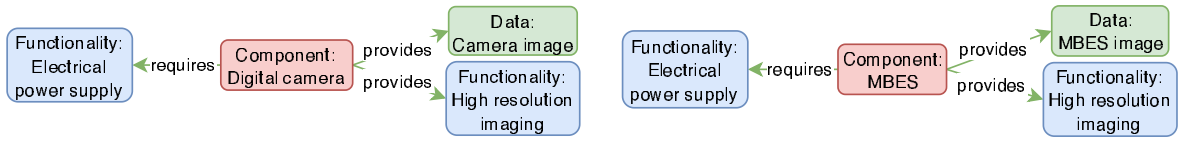
A `mission` is the highest level operation that a vehicle can execute. A typical mission for the use case considered here is to search a certain area of the seabed for mine-like objects and to identify these objects through high-resolution imaging. This mission is named "search and identify" and is shown as an instance of the concept `mission` in Figure 3-8. The figure shows that this mission consists of the tasks "search area", which corresponds to what is done in the search, classification and mapping phase described in Subsection 2-1-2, and "identify", which corresponds to what is done in the reacquisition and identification phase of a mine-hunting operation. For executing these tasks some planning capabilities are required, which themselves can require planning and execution capabilities, shown in Figure 3-8 in red and orange respectively.

Two of the execution capabilities shown in Figure 3-8 are also shown in Figure 3-9. The capability "area covering" represents the ability to plan and navigate a search pattern (e.g. a

lawnmower pattern) over a given target area. This capability is required both for searching an area for a search task and for reacquiring contacts in a identification task, as is shown in Figure 3-8. The capability "identification sensing and processing" represents the ability to obtain data through high resolution imaging and to process this data for identifying the objects that have been imaged. Figure 3-9 also shows the configurations (in purple) that can realise these two execution capabilities. Configurations are characterised in the ontology data by the set of functionalities and outputs that the components in the realising configuration should provide. To express this characterisation in the data, the configurations are defined to provide these same functionalities and outputs, as is illustrated in Figure 3-9 (with `output/input` instances shown in green and functionalities shown in blue). For example, the configuration "identification" should consist of components that provide the output "identifications" and the functionalities "high resolution imaging" and "identification of contacts".

Configurations and components

A configuration on a vehicle consists of a set of components that are connected through input/output relations. These IO relations between components indicate that something is transmitted between the connected components. For each component in the data, the outputs it generates and the inputs it should receive to be able to function are laid down using the *provides* and *requires* relationships. In addition, the functionalities the component provides and requires are defined in the data in the same way. The advantage of using these two relationship types for components is that it allows for automatic inference of the IO relations that can exist between components and of the possible compositions of configurations, as will be explained in the Section 3-3. In Figure 3-10, these *requires* and *provides* relationships with instances of the `functionality` and `data` concepts are shown for two components in the data, namely "digital camera" and "MBES".

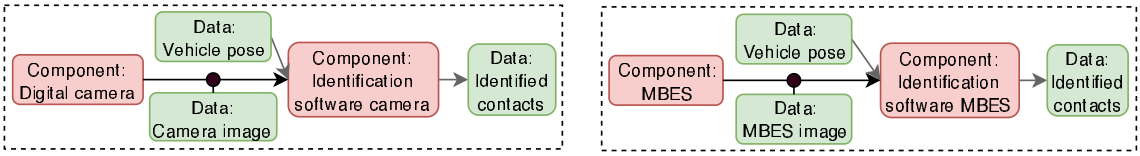


(a) Camera component **(b)** MBES component

Figure 3-10: *provides* and *requires* relationships for two components

The figure shows that both components require the functionality "electrical power supply" and provide the functionality "high resolution imaging", which is needed for the "identification" configuration (as was shown in Figure 3-9). The data output of both components, however, is different. This illustrates that two components can provide the same functionality while providing different data outputs, as was suggested in Example 3.3.

In Figure 3-11, two versions of the "identification" configuration for the LAUV are shown: Figure 3-11a shows the version of the configuration that makes use of the digital camera on board and Figure 3-11b shows the version that uses the MBES.



(a) Camera-based identification configuration **(b)** MBES-based identification configuration

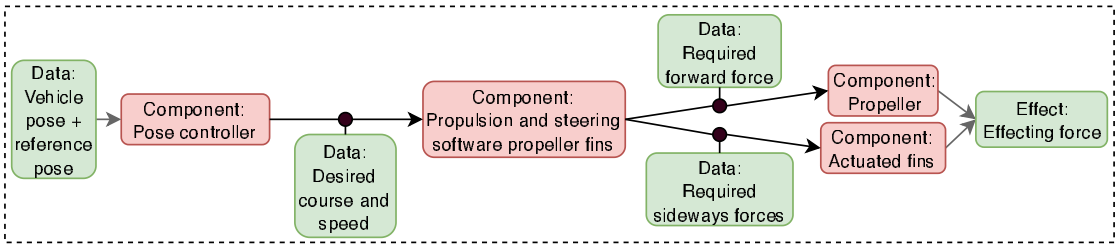
Figure 3-11: Two different sets of components for the identification configuration

The black arrows in Figure 3-11 represent IO relations and are connected to the `data` instance that is transmitted via this relation. The gray arrows represent input or output of data to or from a component. Functionalities are not shown in this figure, but both the "identification software for camera" component and the "identification software for MBES" component provide the functionality "identification of contacts". Therefore, for both versions of the "identification" configuration, the combined components provide the data and functionalities that the configuration should provide, as was shown in Figure 3-9. Figure 3-11 thus illustrates that a configuration can be composed of two different sets of components.

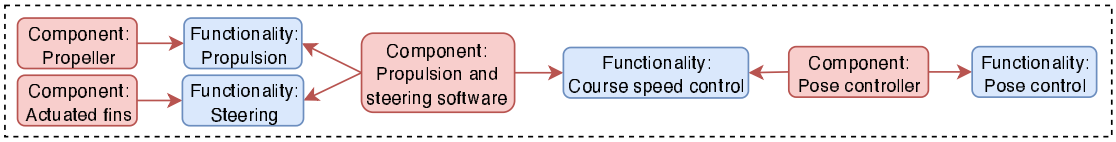
The other configuration that was mentioned in Figure 3-9, the "area covering" configuration, is more complicated than the "identification" configuration in the sense that it consists of more components. The complete "area covering" configuration as it is on the LAUV is shown in Figure B-5 in Appendix B. Some subsets of the components in this configuration, however, were also considered to represent distinct capabilities, such as "localisation", "obstacle avoidance", "path and trajectory planning" and "propulsion and steering control". Therefore, configurations were defined for these capabilities, using the functionalities and outputs that are provided by the components in the corresponding subsets of the "area covering" configuration.
The components and IO relations that make up the configurations for the "obstacle avoidance" capability and the "path and trajectory planning" capability on the LAUV are shown in Appendix B. For the "localisation" capability and the "propulsion and steering control" capability, the realising configurations are shown and explained below. These two configurations will be used as test cases for implementing performance assessment in Chapter 4, which is why these configurations will be described in more detail in this section. The configurations for the two execution capabilities of Figure 3-9 that have not yet been explained ("detection and classification sensing and processing" and "transiting") are discussed in Appendix B as well.

Propulsion and steering control configuration The "propulsion and steering control configuration" gives a vehicle the capability to control its pose (position and orientation) through propulsion and steering such that it can correct an error with respect to a reference pose. Figure 3-12 gives an overview of the components that are part of this configuration in the LAUV of TNO.

Figure 3-12a shows the IO relations for the configuration. In this figure, the software components of the configuration are shown together with the data they receive and produce. These software components control the actuating components on the right side of Figure 3-12a, which output something that is modelled as a `physical effect`. This "effecting force" represents the effect that the actuation of the propeller and fins have on the vehicle's pose in



(a) Propulsion and steering control configuration with IO relations



(b) Components of the propulsion and steering control configuration with functionality instances

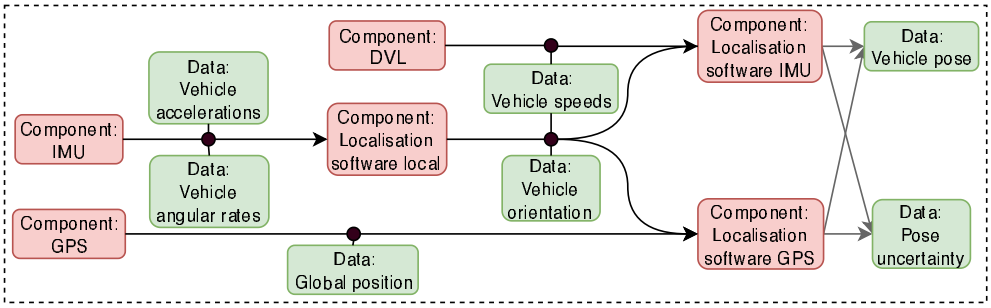
Figure 3-12: Propulsion and steering control configuration

the environment and thus is significantly different from the "required forward force" and "required sideways forces" in the figure, which represent setpoints or reference values and are thus modelled as instances of the `data` concept.

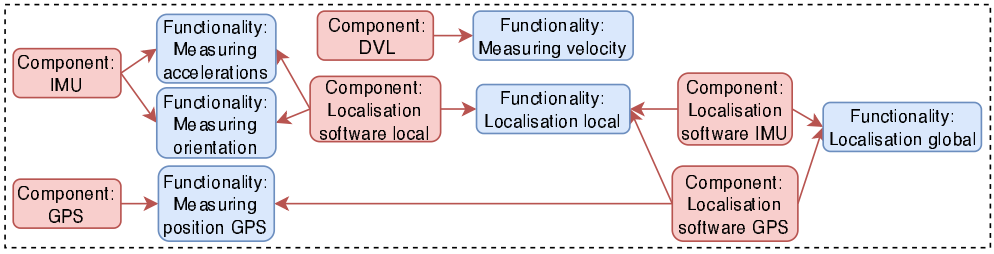
Figure 3-12b also shows the components of the "propulsion and steering control configuration" in the LAUV, but here the relationships of the components with `functionality` instances are shown. The red arrows pointing to the right towards a functionality indicate that the component at the starting side of the arrow provides this functionality and arrows pointing to the left towards a functionality indicate that the corresponding component requires this functionality (support functionalities such as "electrical power supply" are not shown in Figure 3-12b). This figure is shown here to illustrate that the *provides* and *requires* relationships between components and functionalities can contain information that is complementary to the information contained in IO relations. A notable difference between Figures 3-12a and 3-12b is that the order in which the components appear is reversed. The information contained in both of these figures is relevant for the composition of the configuration.

Localisation configuration The "localisation configuration" gives a vehicle the capability to determine its position and orientation in the world, both under water and at the surface ("localisation capability"). In Figure 3-13, the components that are part of this configuration in the LAUV are shown in the same way as was done for the "propulsion and steering control configuration" in Figure 3-12.

In Figure 3-13a, the IO relations for the configuration are shown. The figure shows the sensor systems that can be used for localisation in the LAUV, namely IMU, GPS and DVL, and the data they output. The software components shown in the figure then process this data to obtain an estimation of the vehicle pose. Figure 3-13b shows the relationships of the components of this "localisation configuration" with `functionality` instances. For this configuration, the difference between Figures 3-13a and 3-13b is less pronounced than was the case for the "propulsion and steering control configuration".



(a) Localisation configuration with IO relations



(b) Components of the localisation configuration with functionality instances

Figure 3-13: Localisation configuration

3-3 Implementation

To be able to make use of the developed ontology, it was implemented using a software tool called Grakn, which was introduced in Chapter 2. The goal of implementing the ontology was to enable automatic inference of a vehicle's capabilities, based on the components the vehicle is equipped with. Since the higher-level control of the LAUV of TNO is realised through a software architecture that makes use of the Robot Operating System (ROS) via the ROS Python client, the decision was made to use the Python client for Grakn as well.
During the development of the ontology, implementing what was developed proved to be a useful way to validate the ontology and this led to several adjustments. So implementation actually was part of the development process instead of just the final stage of this process, which is also what is suggested in Figure 2-6. This figure shows the METHONTOLOGY method and in this method, implementation is regarded as one of the phases in an iterative ontology development process [10].
What was also noticed during the development of the ontology is that the choice of the implementation tool can have a key influence on how the ontology can be used in an actual system. Especially the possibilities for reasoning and inference using the ontology are dependent on the tool that is chosen. In Subsection 3-3-2, the reasoning and inference that was implemented for this ontology and the way in which this was implemented in Grakn are explained. But first the implementation of the ontology itself and the way in which it can be queried in Grakn are described in Subsection 3-3-1. This section ends with a subsection on the work that still needs to be done to realise the implementation of the ontology in an actual vehicle.

3-3-1 Implementation and querying of the ontology

Implementing an ontology schema in Grakn is done by defining the schema in Grakn's query language, called Graql. The complete Graql code that was written to implement the developed ontology schema in Grakn is shown in Appendix C, but some significant elements and aspects of this code will be highlighted here.

In Graql, schema concepts are called entities and all entities are organised in a hierarchy. The element at the top of this hierarchy is denoted as `entity` and all other concepts have an inheritance relationship either with `entity` or with a concept that is below `entity` in the hierarchy. In Figure 3-3, inheritance relationships are denoted with the word "isa", whereas in Graql they are denoted with the word `sub`. So in Grakn, the code shown in Listing 3.1 is used to define that the concept `vehicle` is a subconcept of `actor`, as was shown in Figure 3-3, and that `actor` is a subconcept of `entity`.

Listing 3.1: Graql code for defining schema concepts

```
1 define
2   actor sub entity;
3   vehicle sub actor;
```

The use of inheritance relationships in the ontology schema was affected by the choice of Grakn as the implementation tool, since in Grakn multiple inheritance is not supported [55]. So in Grakn, a concept can be a subconcept of only one higher level concept (a concept can have only one *isa* relationship with another concept), although a concept can have multiple subconcepts. In the CORA, however, several concepts inherit from multiple other concepts, as is illustrated in Figure 2-4 where for instance a `robot` is a subconcept of both `agent` and `device`. As a result, implementing the CORA directly in Grakn is impossible, which made it harder to implement the developed ontology as an extension to the CORA. Therefore, the decision was made to avoid the use of multiple inheritance and to use the CORA as an inspiration rather than as the basis for the ontology.

Another consequence of using Grakn was that the relationships that were defined in the ontology schema had to be implemented in a somewhat different way than it would be done in other ontology software tools. In Grakn, relationships are undirected and can exist between two or more things [55], whereas in OWL, for instance, relationships are directed and always connect two elements [55, 54]. All things that are connected through a relationship in Grakn play a certain role in that relationship, so these roles were used to implement the directionality that was used in the relationships in the ontology schema. For instance, for the *has capability* relationship in Figure 3-3 the `actor` concept was defined to have the role of subject and the `capability` concept was given the role of object in this relationship. The corresponding Graql code for defining these concepts, their roles and the *has capability* relationship is shown in Listing 3.2.

Listing 3.2: Graql code for defining the *has capability* relationship

```
1 define
2   actor sub entity,
3     plays actor_subject;
4
5   capability sub entity,
```

```
6       plays capability_object;
7
8     have_relation sub relation,
9       relates subject,
10      relates object;
11
12      has_capability sub have_relation,
13        relates capability_object as object,
14        relates actor_subject as subject;
```

The Graql keyword `plays` is used to indicate which roles a schema element can play, so in this case for instance the `capability` concept can play the role `capability_object`. The keyword `relates` is used to define which roles can be played in the corresponding relationship (or `relation`, as it is called in Graql). Listing 3.2 also shows another Grakn feature, namely the use of subtypes of relationships. A general relationship called `have_relation` with the roles `subject` and `object` was defined in the schema, and the *has capability* relationship was modelled as a subtype, called `has_capability`, of this general relationship. The two roles of this relationships were also modelled as subtypes of the roles of the general relationship, which is indicated by the word `as` in the Graql code. Using such general relationships was found to be helpful for querying large data sets. For the same reason, several subtypes of some concepts (for instance the `component` concept) were defined in the implementation of the schema. These subtypes or inheritance relationships are not necessarily required for the ontology, but can be practical for querying a data set for a specific set of relationships or concept instances in Grakn. In addition, some attributes were added in the Grakn implementation to make retrieval of ontology data easier. In Grakn, attributes need to be defined in combination with a datatype, for instance `string` or `boolean`. In Listing 3.3, the Graql code for defining an attribute for the concept `actor` is shown. This attribute is called `name` and has the datatype `string`. The keyword `has` in Graql indicates that a schema element (in this case `actor`) can have the corresponding attribute. In Grakn, both entities and relations and attributes can have attributes [55].

Listing 3.3: Graql code for defining an attribute

```
1 define
2   actor sub entity,
3     has name;
4
5   name sub attribute,
6     datatype string;
```

After defining the schema in Graql and loading it into Grakn, ontology data can be added by performing `insert` queries in which the entity, attribute or relation that is to be inserted is defined [55]. In Listing 3.4, two examples of Graql `insert` queries are shown: the first one for inserting an instance of the `actor` concept and the second one for inserting an instance of the `capability` concept, both with the `name` attribute. In Graql, the symbol $ is used to represent variables.

Listing 3.4: Graql code for inserting an entity

```
1 insert $x isa actor, has name "LAUV-1";
2 insert $y isa capability, has name "localisation";
```

Listing 3.5 shows an example of a query for inserting a `has_capability` relationship. In this case, the `insert` query is preceded by a `match` clause, which is used to retrieve data instances that follow the particular pattern that is defined in the `match` clause [55]. This `match` clause is necessary because in Grakn the roles in a relationship that is to be inserted need to be played by existing data instances and these instances thus need to be found in the data [55]. In this example query, the `match` clause would be used to retrieve the instances that were inserted using the queries in Listing 3.4 and then a `has_capability` relation between these instances would be inserted in the ontology. In Graql, round brackets () are used to group the elements that are connected in a relationship and in Listing 3.5, the roles that these elements play in the relationship that is inserted are specified as well (`actor_subject` and `capability_object`).

Listing 3.5: Graql code for inserting a relationship

```
1 match
2   $x isa actor, has name "LAUV-1";
3   $y isa capability, has name "localisation";
4 insert (actor_subject: $x, capability_object: $y) isa has_capability;
```

The `insert` queries that were described above can be performed in several ways. In this final thesis project, the ontology data that was defined for the use case was saved in CSV files and loaded into Grakn using the Python client for performing the `insert` queries. In the CSV files, all instances for the schema concepts and attributes were defined and for all relations defined in the schema, the instances for which these relationships had to be inserted were listed.

After loading the ontology schema and data, the ontology can be queried in order to retrieve specific data instances or schema types or to perform some calculations with specific data sets. In Grakn, the `match get` query can be used to retrieve data instances or schema elements. In this type of query, the `match` clause again is used to specify the pattern that is sought for. Using the `get` command, the ontology elements that match this pattern then can be retrieved, for instance through the Python client of Grakn. In addition, some calculations can be done with the results of a `match get` query by adding an aggregation command such as `count`, `mean` or `sum`. These three commands can be used to count the number of results or to find the mean or sum of a set of attribute values respectively. [55]
Another option for querying the ontology data is to use the `compute` command. This command can be used to calculate statistical values, such as means or standard deviations, over large sets of data. Furthermore, metadata of the ontology data can be obtained with this command; for example, the shortest path (of instances and relations) between two data instances can be computed, as well as the number of other instances an instance is connected to. [55]

3-3-2 Reasoning with the ontology

One of the reasons for selecting Grakn as the implementation tool was that it offers useful methods for querying and for reasoning with the ontology. The goal for the implementation of reasoning with the ontology was to be able to use the ontology for automatic inference of a vehicle's capabilities based on the components it is equipped with. Before discussing how this goal was reached, first the way in which reasoning can be performed in Grakn is explained.

Reasoning over ontology data in Grakn can be done using pre-defined rules, which can be included in the Graql definition of the ontology schema. These rules are composed of a set of conditions (the "when" part of the rule) and a conclusion (the "then" part of the rule). The `when` part of a rule has the same function as the `match` clause that was described before, and is also structured in the same way. The `then` part of a rule, on the other hand, has the same structure as the `insert` command in a `match insert` query, but its function is different. In a `match insert` query, the instances that are found for the pattern in the `match` clause are used to insert new data, for instance a new relationship between these instances. In the `then` part of a rule, however, no new data is inserted, but the ontology data that corresponds to the pattern in the `when` part is used to create queryable relationships. These "inferred" relationships thus are not included in the ontology data, but they can be queried, which then initiates an automated reasoning process in which the ontology data is searched for the pattern defined in the `when` part of the corresponding rule.
The advantage of using rules is that they can be included in the schema in Grakn and will thus hold for any data that is inserted in the ontology; also, the inferred conclusions do not have to be deleted separately when the corresponding data (that fits the pattern of the `when` part) is deleted. A `match insert` query, however, inserts data at the time the query is performed, thus only taking into account the ontology data at that specific moment, and the inserted relationships have to be deleted separately if the corresponding data is deleted.
In Listing 3.6, the Graql code for an example rule is shown. This example rule or query could be used for the developed ontology and would have the following meaning for the use case: if a vehicle $v executes a task $t and that task $t is part of the mission $m, then the vehicle $v also executes the mission $m. If the example rule would be used, the `executes_mission` relationship would not be inserted in the data, but it would be possible to query for it at any moment.

Listing 3.6: Graql code for an example rule

```
when{
  $v isa vehicle; $t isa task; $m isa mission;
  (executing_actor: $v, executed_task: $t) isa executes_task;
  (containing_mission: $m, contained_task: $t) isa is_part_of_mission;
},
then{
  (executing_actor: $v, executed_mission: $m) isa executes_mission;
};
```

In Subsection 3-2-1, the developed ontology schema was described and for some of the described relationships, conditions were mentioned, such as that an actor can only execute a task if it has the required capabilities. Setting these conditions in the schema is a good example of what the reasoning rules can be used for. Additionally, some form of reasoning is required for achieving the goal of enabling automatic inference of a vehicle's capabilities. A part of the reasoning that was implemented for realising this goal consisted of automatic inference of the IO relations that can exist between components and inference of the possible compositions of configurations, as was mentioned in the previous section. These inferred configuration compositions were then used to infer which capabilities a vehicle could have.
Initially, rules were used for implementing reasoning because they could be included in the schema and would thus hold for every moment in time, which is especially useful when the ontology data changes over time. However, this approach was abandoned later, switching to

using `match insert` queries for reasoning, for two practical reasons: in the first place, some restrictions on the patterns that can be used in the `when` part of a rule, especially concerning negation patterns, turned out to be problematic for achieving the implementation goal, and these restrictions do not hold for the `match` pattern in a query. Secondly, querying data for inferred relationships (defined through rules) proved to be very slow for the developed ontology, making the use of actual relationships (defined through `match insert` queries) preferable. The implemented queries for inferring which IO relations can exist in a vehicle and what configurations and capabilities it can have are explained in the following two paragraphs.

Inference of IO relations For inferring the IO relations that can exist in a vehicle, the *provides* and *requires* relationships of the vehicle's components with functionalities and output/input instances were used. The idea behind the implemented reasoning process is that components that provide a certain output should be connected with components that require that output as their input, provided that for both components the requirements for functionalities can all be fulfilled. In addition, the IO relations for capacity instances (such as "electrical power") will be inserted first (before the IO relations for data instances), such that first all the components' requirements for capacities are fulfilled before creating the relations that represent the data connections between the components. The first step in the reasoning process therefore is to query the ontology for all the components that the vehicle has and all the functionalities they provide and require. The resulting data is then used to iteratively check for all components whether the functionalities they require are provided by other components in the vehicle that have their functionality requirements fulfilled. This process results in a list of functionalities that are available since they are provided by components that have all their functionality requirements fulfilled. The next step is to make connections between the components that provide capacities and the components that require these capacities through IO relations, taking only those components into account that have all their functionality requirements fulfilled. Subsequently, the same is done for components providing and requiring data, imposing the additional condition that for these components the requirements for capacities must be fulfilled via the IO relations created in the previous step. For the developed ontology in which a vehicle and its components with all their requirements are inserted as ontology data, the result of performing this reasoning process is that IO relations are created, both for capacities and data, between the components that can have all their functionality requirements fulfilled.

Inference of configurations and capabilities The IO relations that are created through the reasoning process described in the previous paragraph form the basis for inferring which configurations and capabilities the corresponding vehicle can have. As mentioned before, configurations are characterised in the ontology data by the set of functionalities and outputs that should be provided by the connected components that constitute the configuration. Therefore, the first step for inferring which configurations a vehicle can have is to query the ontology for all components connected through IO relations and for the outputs and functionalities that these components provide. The result of this query can then be compared with the combination of outputs and functionalities that is needed for a configuration in order to identify the set of connected components in the vehicle that together can provide this combination and thus can form the corresponding configuration. Performing this reasoning process for all configurations then results in a list of configurations that the vehicle can have.

This list can then be used to infer the capabilities a vehicle can have, since capabilities are directly coupled to configurations through a *is realised by* relationship. The result of this reasoning process combined with the process described in the previous paragraph is that the capabilities of the vehicle have been inferred using the components it is equipped with, which was the goal that was defined for the implementation of the ontology. Using these capabilities, the tasks a vehicle can perform then can be found by querying the ontology for all the tasks that only require capabilities that the vehicle can have, or, in other words, for all tasks that do not require a capability that the vehicle can not have. This last query is one of the queries that requires a double negation pattern which is not allowed in Graql rules, but is allowed in Graql queries [55].

3-3-3 Remaining work for implementation in a vehicle

The ultimate goal of implementing the ontology is to use it in an actual vehicle, more specifically the LAUV of TNO, which was also why the Python client was chosen to interact with Grakn. Due to time limitations this goal was not yet achieved, so this subsection is used to give a general idea of what is still to be done.
In short, the most important step that needs to be taken is to connect the ontology that is implemented in Grakn to the vehicle's software, which is implemented in ROS. Coupling Grakn and ROS has been done before at TNO, so this should not be a problem, but the expectation is that some form of interfacing needs to be developed to realise a well-functioning connection. An aspect of this that might be challenging is to find a robust method for updating rapidly changing data in Grakn, especially if the ontology will be used not only for planning prior to executing a task but also for real-time applications such as online monitoring of the vehicle's capabilities. In addition, some effort will probably have to be put into developing a method for interfacing the ontology with the planning system that is being developed for the LAUV. Other implementation issues concerning performance prediction or monitoring will be addressed in the next chapter.

Chapter 4

Performance assessment

The ontology that was described in Chapter 3 makes it possible to infer the capabilities of an AUV, given the components it is equipped with. A planning system then can use this knowledge to plan the tasks of a mission such that tasks are only assigned to vehicles that are actually capable of performing these tasks. When multiple vehicles with similar capabilities are available, however, it would be very useful to have an idea of how well each vehicle can perform a task. This way, a planning system can assign the tasks of a mission to the available vehicles that are expected to perform best on these tasks, which enables the planning system to plan a mission in an optimal way. To be able to assess how well a vehicle can perform a task, a measure (or set of measures) that characterises the performance on a task needs to be found. Additionally, a method needs to be developed for inferring how well a vehicle can perform a task given the components it is equipped with. Finally, the developed ontology needs to be extended, such that it can be used for this performance assessment process.

In the first section of this chapter, the extensions that were made to the developed ontology in order to incorporate a performance assessment methodology into it will be described, as well as what this methodology entails. The two configurations that were described in the previous chapter ("propulsion and steering control configuration" and "localisation configuration") were used as test cases for both the ontology and the developed performance assessment methodology. This methodology is data-driven, so simulations were performed to obtain the required data and to test whether the method works for both configurations. The simulations that were done and the results that were obtained will be described in Section 4-2. The last section of this chapter then describes how the results of the simulations can be used in combination with the extended ontology and what more is still needed to be able to infer what tasks a vehicle can perform and how well it can do so.

4-1 Extensions to the ontology for performance assessment

As was done in Chapter 3, the extensions to the ontology will be explained by first describing the schema elements that were added, in Subsection 4-1-1, and then discussing the corresponding ontology data. The "propulsion and steering control configuration" and the "localisation configuration", which were discussed in Section 3-2, will be used as test cases, so the data instances corresponding to the added schema elements were identified for these two configurations, as will be discussed in Subsections 4-1-2 and 4-1-3.

4-1-1 Performance related schema elements

The starting point for extending the ontology was the developed schema as it was shown in Figure 3-3. The goal of adding performance aspects was to be able to predict, and possibly also monitor, the level of performance that a vehicle can achieve for a certain task based on the components it is equipped with. The initial idea for realising this was to define performance measures on component level, configuration level and task level and then find methods for calculating or inferring task performance by combining the relevant configuration performance measures, which themselves would be calculated or inferred using the relevant component performance measures. So, to get an idea of useful measures of performance, the LAUV software architecture document [25] was studied again and used, together with a TNO document describing measures of effectiveness and measures of performance for the NMCM use case [30], to draw up a list of performance measures corresponding to some of the previously identified tasks and capabilities. However, this initial idea for calculating or inferring task performance using configuration and component performance turned out to be very hard to implement for the use case considered here because of the many interdependencies between performance measures on different levels and the many external factors influencing performance. Furthermore, finding meaningful performance measures for all identified components proved to be quite challenging. Therefore, the decision was made to use a data-driven approach instead of a theoretical one and to take configuration performance as a starting point for predicting or assessing task performance.

The resulting method that was devised for assessing performance can be construed as a kind of look-up table that outputs the values for a set of performance measures for a specific configuration, task or mission given the prevailing conditions that are relevant for that configuration, task or mission. In Figure 4-1, a schematic illustration of this method is shown for the "identification configuration" that was shown in Figure 3-11a.

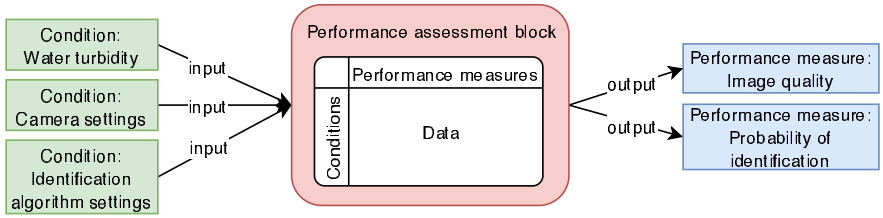


Figure 4-1: Schematic representation of the performance assessment method for the identification configuration

The "performance assessment block" in the middle of this picture contains a look-up table that is filled with data values for the performance measures of this configuration under different conditions. This look-up table is then used to output values for these performance measures for the conditions that are given as input to the performance assessment block. These conditions include both external factors (e.g. water turbidity) and internal factors (e.g. camera settings).

The schema elements that were added to be able to include the performance assessment approach that was described above in the ontology are shown in Figure 4-2, together with the main concepts and relationships of Figure 3-3, which were discussed in Chapter 3. The blue rounded rectangles in Figure 4-2 represent attributes, which are used to model properties of schema elements. The blue arrows denoted with the word "has" show which elements the attributes belong to.

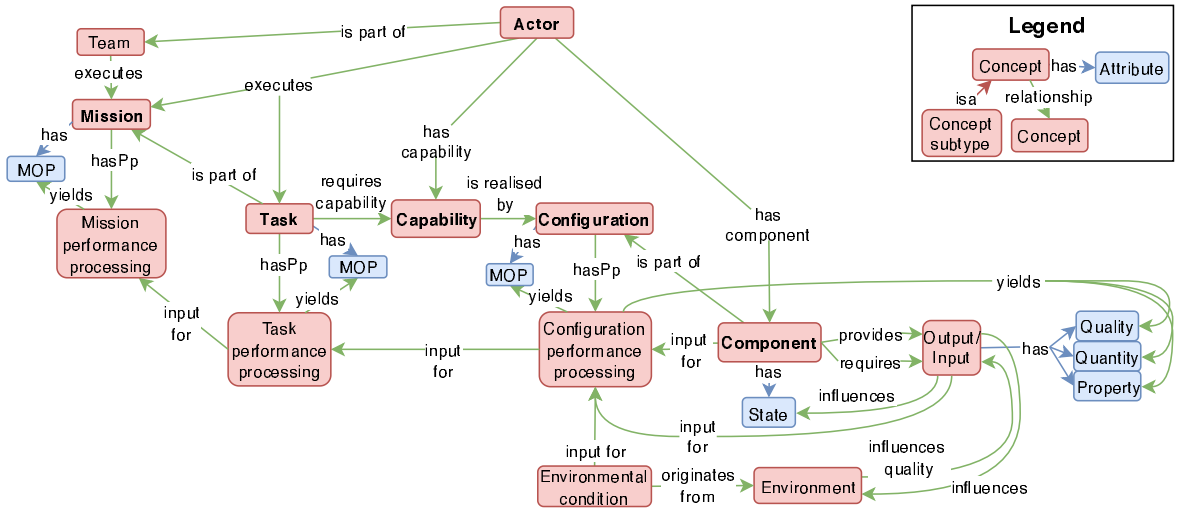


Figure 4-2: Schema with the elements relevant for performance

The added schema elements that correspond to the "performance assessment block" of Figure 4-1 are the concepts `configuration performance processing`, `task performance processing` and `mission performance processing`. These concepts are called "performance processing" because they do not only contain a look-up table but also comprise the process or algorithm that, when given the prevailing conditions, uses the data in this table to output predictions for the relevant performance measures.

The performance measures are modelled as attributes of the corresponding configuration, task or mission and are shown in Figure 4-2 denoted with the letters MOP (short for Measure of Performance). The output relation between the performance processing concepts and the MOP attributes is modelled, and shown in the figure, as a relationship called *yields*. For the `configuration performance processing`, the performance measures that are outputted by it can also be modelled as attributes of the `output/input` concept, as will be explained below. Additionally, the figure shows that each `performance processing` concept receives some sort of input (indicated with the *input for* relationship), where this input represents the conditions that were mentioned before. For the `configuration performance processing`, the input and output relationships have been further developed because configuration performance was

taken as the starting point for developing a performance assessment method and because the ontology data for the `configuration` and `component` concepts was most extensive at this point in the development process.

Schema elements relevant for configuration performance

In Figure 4-3, the schema elements of Figure 4-2 that are relevant for configuration performance are shown again. As indicated by the *input for* relationships in this figure, three different kinds of inputs (i.e. conditions) were modelled for the `configuration performance processing` concept, namely `environmental condition`, `component` and `output/input`. These three inputs will be explained below:

- The `environmental condition` concept represents a state or property of the environment (as indicated by the *originates from* relationship with `environment`) that can affect the performance of a configuration, as for instance the underwater currents affect how well a vehicle can navigate underwater. This concept thus is used to include external factors as conditions.
- The components that are part of a configuration are given as an input to the `configuration performance processing` concept for two reasons: firstly, the combination of components in a configuration can influence its performance, as for instance localisation underwater happens more accurately when using a DVL than without one, and secondly, the *state* of a specific component (modelled as an attribute of `component`) can affect configuration performance, as for instance the settings of a detection algorithm can affect whether a contact is detected or not.
- Finally, outputs/inputs that are transmitted via IO relations between components can also serve as an input for the `configuration performance processing`, because an output/input itself or its properties can affect configuration performance, as for instance the quality of sonar data can affect whether or not an object can be detected.

For the other two performance processing concepts (for tasks and for missions), the preceding performance processing concept is modelled as an input because, generally, configuration performance affects task performance for a task that requires a capability that is realised by this configuration, and this task performance affects the overall mission performance for the mission that the task is part of.

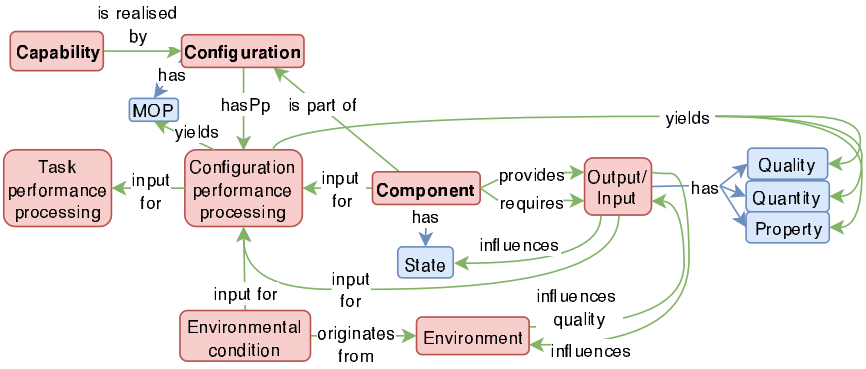


Figure 4-3: Schema elements that are relevant for configuration performance

For the `configuration performance processing`, two different kinds of performance outputs were modelled, as is indicated with the *yields* relationships in Figure 4-3. The first of these outputs is the *MOP* that was mentioned before. The second performance output consists of the three attributes of the `output/input` concept (namely *quality*, *quantity* or *property*). These attributes represent characteristics of outputs/inputs and can sometimes be an indication of configuration performance. For instance, the expected mean localisation error can be seen as an indication of the performance of the "localisation configuration" and can be modelled as the *quality* of the output/input "estimated vehicle pose". This expected error can be known from previous experiments or simulations and thus can be an output of the configuration performance processing for the "localisation configuration". Similarly, the expected type of trend in the increase of this localisation error (e.g. linear increase) can be seen as a *property* of the "estimated vehicle pose" and also gives an indication of the performance of the "localisation configuration".

For the "propulsion and steering control configuration" and the "localisation configuration", the relevant conditions and appropriate performance measures were identified and added to the ontology data for the use case. The following two subsections will discuss these conditions and performance measures.

4-1-2 Performance related ontology data for the propulsion and steering control configuration

The "propulsion and steering control configuration" should enable the vehicle to control its pose such that it can reach a given reference pose. Therefore, the performance measures for this configuration should indicate how well the vehicle's pose is controlled. For reference, the "propulsion and steering control configuration" of the LAUV is shown again in Figure 4-4. In the LAUV, the pose is controlled with a PID controller (shown as "pose controller" in Figure 4-4), which actuates the vehicle's propeller and fins. In a real scenario, the vehicle's actual position and orientation (ground truth) are not known and the vehicle has to use its estimated pose for control. For assessing the performance of the "propulsion and steering control configuration", however, the actual pose of the vehicle needs to be known to rule out possible errors in the pose produced by the "localisation configuration". Therefore, the required data for performance assessment was obtained by performing simulations in a realistic virtual environment, since in simulations the actual pose is known.

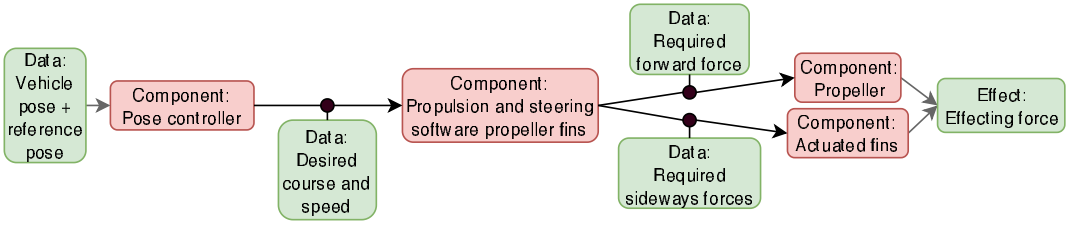


Figure 4-4: Propulsion and steering control configuration with IO relations

Conditions (input for the configuration performance processing)

For this configuration, it is useful to know its robustness against environmental disturbances in the form of currents, since these are common in the areas in which the search operations of the use case take place. Therefore, the direction and speed of the current in the operation area were identified as conditions for this configuration's performance processing and were both modelled in the ontology as `environmental condition` instances. In this respect, the navigation depth is also important, since the depth at which the vehicle sails can influence the way in which currents affect the vehicle's pose. The navigation depth therefore was included as a condition and was modelled as a *property* of the planned path. Another influencing factor is how the pose controller is configured. The tuning of the PID gains of this controller on the LAUV is assumed to be fixed, but other controller settings such as the reference speed and the way in which the reference pose is determined are variable. For the controller used in the LAUV, the reference pose is determined by projecting the vehicle pose on the trajectory that the vehicle should follow and then shifting this projected pose ahead on the trajectory over a given distance. This distance is called the "look-ahead distance" and can be varied. A schematic representation of the "look-ahead distance" is shown in Figure 4-5, where the orange shape represents the LAUV. These controller settings were included as conditions for the configuration performance and were modelled in the ontology as part of the *state* attribute of the controller component.

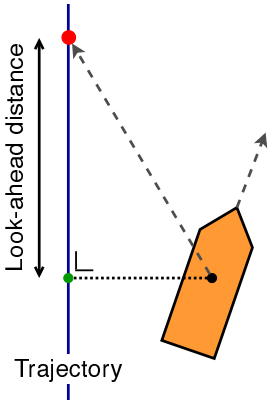


Figure 4-5: Look-ahead distance

The performance for this configuration was tested by letting the vehicle follow a trajectory consisting of a 90 degree turn and a straight track, while monitoring its pose. The radius of the initial turn was expected to influence the configuration performance on the straight section of the trajectory, so this initial turn radius was taken into account as a condition, modelling it as a *property* of the planned path. In Table 4-1, an overview of the conditions that were described above and the way in which they were modelled in the ontology is shown.

Table 4-1: Conditions for the propulsion and steering control configuration

Condition:	**Modelled in ontology as:**
Current direction	instance of `environmental condition`
Current speed	instance of `environmental condition`
Navigation depth	*property* attribute of `data` instance "planned path"
Controller settings	*state* attribute of `component` instance "pose controller"
Initial turn radius	*property* attribute of `data` instance "planned path"

Performance measures (output of the configuration performance processing)

While analysing possible performance measures for this configuration, it was observed that the task for which a configuration is used can influence what kind of performance measures are relevant. The same holds for the conditions that are relevant, as will be explained for the

"localisation configuration". Therefore, a specific task or situation was taken as a reference for determining the relevant performance measures and conditions. In this case, the situation in which an AUV is surveying an area of the seabed in a lawnmower pattern (parallel tracks) was used as a reference. In this situation, the straight sections of the trajectory are the tracks during which the area is imaged and the turns in the trajectory form the transitions between two tracks. Especially on those straight tracks, the pose control performance is important since it can have an effect on how well the area is surveyed. The performance measures that were defined for this configuration therefore all concern the pose control performance on the straight track after the initial turn.

The first defined performance measure is the "settling distance", which is the distance it takes the vehicle after the initial turn to reach a stable position with respect to the reference track. This measure gives an idea of how long it takes after a turn for a vehicle to be able to properly execute a survey track. The other defined performance measures all concern the performance after the settling distance, in the "steady state" situation. These measures include the mean values of the steady state position error, orientation error and speed error with respect to their reference values and the standard deviations of the steady state orientation error and speed error. The means of the steady state position, orientation and speed errors were included as performance measures because they give an indication of how well the pose and speed of the vehicle are controlled in the prevailing conditions. The standard deviations of the steady state orientation error and speed error were taken into account because variations in orientation and speed can be detrimental to the imaging quality when surveying.

The settling distance was modelled in the ontology as an *MOP* attribute of the configuration, since it is specific to the configuration and cannot be modelled as an attribute of one of the `output/input` instances that are relevant for this configuration. The other performance measures were modelled as *quality* attributes of `physical effect` instances, since the vehicle's actual position, orientation and speed represent a state of the vehicle in the environment. Table 4-2 summarises the performance measures that were described above and the way in which they were modelled in the ontology.

Table 4-2: Performance measures for the propulsion and steering control configuration. st. dev. is short for standard deviation.

Performance measure:	**Modelled in ontology as:**
Settling distance	*MOP* attribute of the configuration
Position error (mean)	*quality* attribute of `physical effect` instance "position"
Orientation error (mean)	*quality* attribute of `physical effect` instance "orientation"
Speed error (mean)	*quality* attribute of `physical effect` instance "speed"
Orientation error (st. dev.)	*quality* attribute of `physical effect` instance "orientation"
Speed error (st. dev.)	*quality* attribute of `physical effect` instance "speed"

4-1-3 Performance related ontology data for the localisation configuration

The purpose of the "localisation configuration", shown again for the LAUV in Figure 4-6, is to determine the vehicle's position and orientation in a global coordinate frame. The performance measures for this configuration therefore should give an idea of how well the position and orientation are estimated. Ideally, the performance measures should indicate

what the difference is between the estimated position and orientation and the real position and orientation. To assess this difference based on data, the actual pose of the vehicle must be known. Therefore, again simulations were used to obtain data for assessing the configuration performance. The resulting data can be used to estimate the position and orientation errors for real situations.

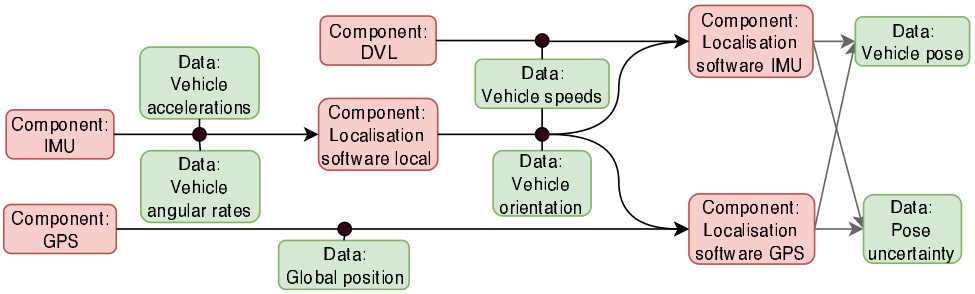


Figure 4-6: Localisation configuration with IO relations

Conditions (input for the configuration performance processing)

In AUVs, often multiple sensor systems are used for underwater localisation and the resulting sensor data is then fused by means of localisation software. In the LAUV, the main element of this software is a Kalman filter. This filter produces an estimate of the vehicle's pose, based on the sensor data it receives, and also outputs covariance values for this pose, which depend on the quality of the sensor data that the filter receives and on the combination of sensors that is used. These two aspects, therefore, were defined to be conditions for this configuration's performance processing. The combination of sensors that is used can be inferred by the configuration performance processing instance because all components that are part of the configuration are connected to the performance processing via an *input for* relationships. The quality of the sensor data is represented by the covariance values of the data, and is included in the ontology as *quality* attributes of `data` instances. This data, with attributes, is input for the performance processing instance.

The type of path that the AUV is travelling can also influence the performance of the "localisation configuration", for instance because different sensor systems are used when travelling under water than when travelling at the water surface. This path type depends on the task in which the configuration is used. Therefore, again the situation in which an AUV is surveying an area of the seabed was used as a reference, since in this situation localisation has a critical influence on the overall performance. The search pattern that is used for surveying can affect the localisation performance, as for instance the consecutive time underwater and the number of turns in the path can influence the quality of the pose estimation. Therefore, the search pattern that is used was defined to be a condition for the performance processing instance, as well as some characteristics of the search pattern. The most commonly used pattern is a lawnmower pattern and its relevant characteristics are the length of the straight parallel tracks (also called legs), the depth at which the pattern is navigated and whether or not GPS fixes are performed and if so how often. A GPS fix is a manoeuvre of the AUV in which it goes to the water surface and activates its GPS to determine its position. By doing this, the position error of the localisation configuration is practically reset to 0, since localisation with

GPS is very accurate. This is useful because underwater often Inertial Navigation Systems are used, which rely on accelerometers and gyroscopes and thus result in position errors that grow boundlessly with time due to integration drift.

The search pattern type and its characteristics are modelled in the ontology as *property* attributes of the `data` instance "planned coverage path". The conditions that were described above are listed in Table 4-3.

Table 4-3: Conditions for the localisation configuration

Condition:	**Modelled in ontology as:**
Sensor data covariances	*quality* attribute of `data` instances
Combination of sensors used	`component` instances
Search pattern type	*property* attribute of `data` instance "planned coverage path"
Leg length	*property* attribute of `data` instance "planned coverage path"
Navigation depth	*property* attribute of `data` instance "planned coverage path"
GPS fix frequency	*property* attribute of `data` instance "planned coverage path"

Performance measures (output of the configuration performance processing)

The square root of the covariance values that are outputted by the Kalman filter for each element of the pose (position and orientation in three directions), which thus represent standard deviations, can be regarded as an approximation of the uncertainty of the estimated pose. In a real scenario, this uncertainty is the only available indication of the accuracy of the estimated pose, since the actual pose is not known under water. In simulations, however, this uncertainty approximation, which will be denoted as pose uncertainty, can be compared with the error between the estimated pose and the actual pose, denoted as pose error. The resulting data then can be used to get an indication of how well the pose uncertainty approximates the actual pose error. The performance measures that were identified for this configuration therefore concern both the pose uncertainty and the pose error and the coherence between these two.

In data obtained from initial simulations for the "localisation configuration", it was observed that the pose uncertainty and especially the pose error can vary quite a lot, but do follow a certain trend (for instance a linear increase). Therefore, the decision was made to fit trend lines through the data and use the means of these fitted lines for assessing performance. For the pose uncertainty and the pose error, three different performance measures were defined: the type of line that can be fitted through the increase of the uncertainty or error, a parameter that characterises this fit and a value that indicates the goodness of fit. These measures were modelled in the ontology as *property* and *quality* attributes of the `data` instances "pose uncertainty" and "estimated pose". Finally, a measure that compares the fit through the uncertainty and through the error was defined and modelled as an *MOP* attribute of the configuration, since it is specific to the configuration and cannot be modelled as an attribute of one of the `output/input` instances that are relevant for this configuration.

In Subsection 4-2-2, the choice for these performance measures based on trend lines fitted through the data will be further explained. Table 4-4 again lists the performance measures that were mentioned above.

Table 4-4: Performance measures for the localisation configuration

Performance measure:	**Modelled in ontology as:**
Pose uncertainty, trend line type	*property* attribute of `data` instance "pose uncertainty"
Pose uncertainty, fit parameter	*quality* attribute of `data` instance "pose uncertainty"
Pose uncertainty, goodness of fit	*quality* attribute of `data` instance "pose uncertainty"
Pose error, trend line type	*property* attribute of `data` instance "vehicle pose"
Pose error, fit parameter	*quality* attribute of `data` instance "vehicle pose"
Pose error, goodness of fit	*quality* attribute of `data` instance "vehicle pose"
Coherence of uncertainty and error	*MOP* attribute of the configuration

4-2 Simulations for collecting performance data

The simulations that were done to obtain values for the described performance measures under several conditions were performed using a ROS and Gazebo based UUV Simulator, which is described in [63] and was developed for the SWARMs project that was mentioned in Chapter 2. Gazebo is an open-source robotics simulator, which was extended by the authors of [63] to enable simulating in underwater scenarios. Several simulated vehicles, including the LAUV, have been integrated in the UUV Simulator. At TNO, the virtual model of the LAUV has been extended and adapted in such a way that it forms a good virtual representation of the actual LAUV and its software architecture. In Figure 4-7, a screenshot of the visualisation of the UUV Simulator, with the virtual LAUV in it, is shown.

Figure 4-7: Screenshot of the UUV Simulator visualisation with the virtual LAUV

The simulation environment is set up in such a way that simulations can be done in which the vehicle navigates a given path under predefined environmental conditions, using given vehicle settings, while all desired variable values are recorded in a file. The resulting file can then be used to analyse the performance of a configuration in the vehicle under the conditions that were set for the corresponding simulation.

The following two subsections will discuss what paths, vehicle settings and environmental conditions were used in the simulations for the two configurations considered here and will present the results that were obtained by analysing the recorded simulation data.

4-2-1 Simulations for the propulsion and steering control configuration

Simulation scenarios

In the simulations that were done for the "propulsion and steering control configuration", the LAUV was given a path consisting of a 90 degree turn followed by a straight track of 300 metres at a constant depth of 18 metres below the water surface. The variables that were recorded during the simulations are the vehicle's actual position, orientation and speed. Three of the conditions that were mentioned in the previous section were varied for the different simulations, using the parameter values shown in Table 4-5. Both the current direction and the current speed were varied in order to test the robustness of the configuration against currents under water.

Table 4-5: Conditions used for simulations for the propulsion and steering control configuration

Condition	Values
Current direction	0°
	45°
	90°
	135°
	180°
	225°
	270°
	315°
Current speed	0 m/s
	0.2 m/s
	0.4 m/s
Initial turn radius	10 m
	20 m

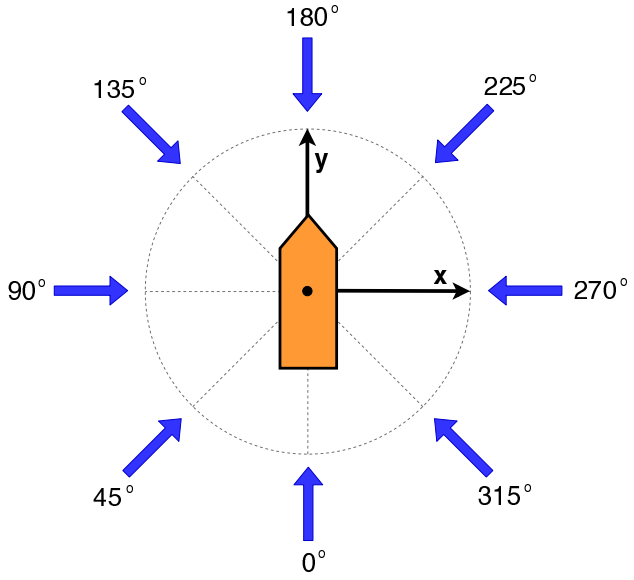


Figure 4-8: Schematic top view of the different current directions (blue arrows), shown with the vehicle (in orange) for reference

The current direction is defined relative to the direction of the straight track of the path (i.e. the direction of the y axis of the global frame, as is shown in Figure 4-8). A current direction with an angle of zero degrees thus implies that the vehicle (shown in orange in Figure 4-8) moves with the current on the straight track and an angle of 90 degrees means that the current comes in from the left on the straight track, as is illustrated in Figure 4-8. The orange shape in this figure represents a top view of the vehicle pointing in the direction of the straight track and the blue arrows represent the different current directions that were used in the simulations.
For the current speed, three different values were used, one of which is 0 m/s (the no current scenario). Apart from the current direction and speed, the radius of the 90 degree turn before the straight track was varied too, using two different values. This initial turn radius was varied to see the effect on how long it takes the vehicle to achieve a "steady state" condition when travelling the straight section of the path after this 90 degree turn.

Simulations were performed for all different combinations of values for the three conditions shown in Table 4-5. The total number of different simulation scenarios thus was 34, as the current direction did not need to be varied for a current speed of 0 m/s, and for all of these scenarios, ten different simulations were performed. In these simulations, the vehicle's pose controller was given the actual vehicle pose to use for control, instead of the pose estimated by the "localisation configuration", to eliminate the influence of the "localisation configuration" on the actual pose and obtain performance data that can be used to assess the performance of the "propulsion and steering control configuration" only. The controller settings were kept constant for all simulations since it was still under development at the time of simulating. The reference speed for the controller was set to 2 metres per second and the look-ahead distance was set to 12 metres.

Results for two single simulations

After performing the simulations for all different scenarios, the recorded data was analysed and processed to obtain values for the defined performance measures. Before presenting these results, first some of the data that was recorded in two individual simulations will be shown in order to give an idea of what these simulations entailed and what difference a change in the conditions can make. The data that will be presented here is from simulations for two of the 34 scenarios that were described above. In the first scenario, there were no currents and the initial turn radius was 20 metres, which the configuration should be able to deal with. For the second scenario, however, the current speed was set to 0.4 metres per second with a direction of 90 degrees and the initial turn radius was set to 10 metres.

In all simulations for this configuration, the starting point and path of the vehicle were chosen such that it would first perform a straight track of 30 metres, then perform a 90 degrees turn with the given radius and then travel a straight track of 300 metres that is aligned with the y axis (as defined in the simulation environment, illustrated in Figure 4-8). In all simulations, a data recording frequency of 10 Hz was used.

Results for the travelled path In Figures 4-9 and 4-10, the actual path that the vehicle travelled is shown for the two selected simulations (in x and y direction). The black dots shown on the path correspond to the positions at which the controller started or finished controlling the vehicle's pose along the straight part of the track. Both figures show a small anomaly in the path, namely the horizontal section going from (0,0) to (-50,0) in Figure 4-9 or to (-40,0) in Figure 4-10. This section of the path was not travelled by the vehicle, but appears in the figures because the virtual vehicle was loaded into the simulation environment with a short delay and therefore the very first data points for the position were given the value (0,0) instead of the real starting point of the vehicle.
Figure 4-9 shows that the controller was indeed able to keep the vehicle on the planned track in the no current scenario. In the other simulation, however, the vehicle deviated from the planned path after starting the 90 degrees turn and did not fully reach the straight section of the path due to the currents coming in from the left side with respect to the direction of the straight track.

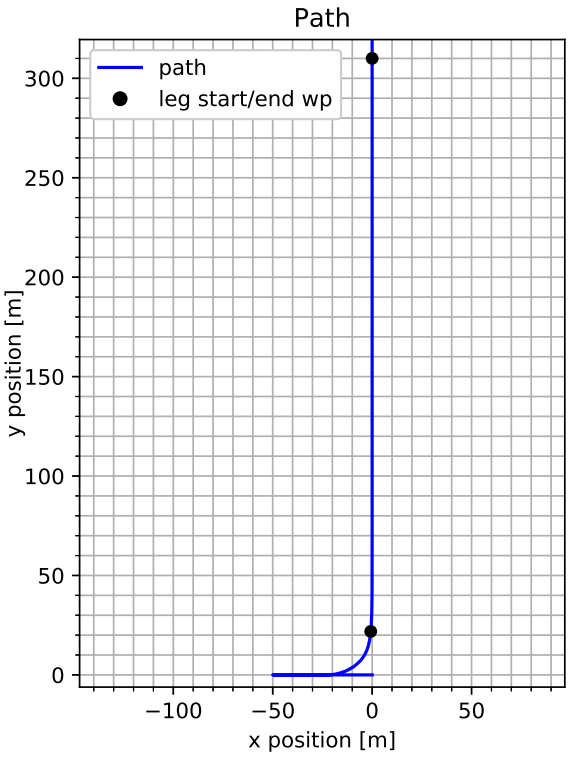


Figure 4-9: Path navigated in simulation without currents and with large initial turn radius

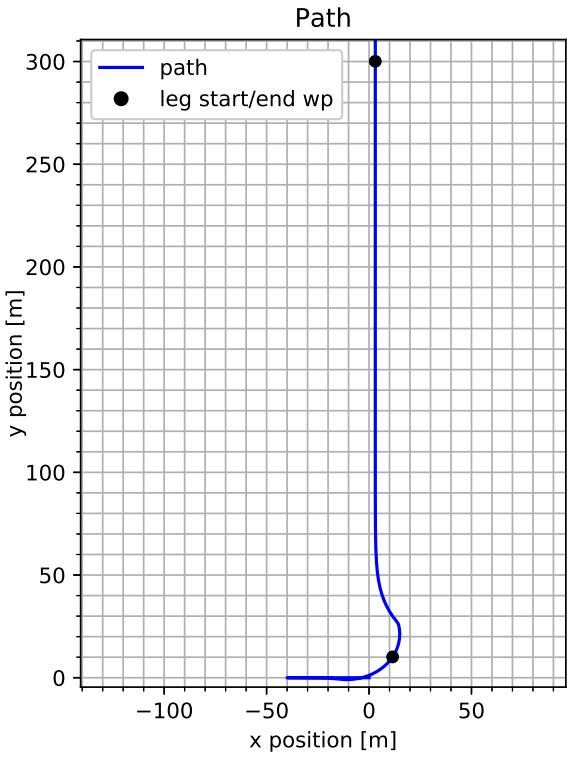


Figure 4-10: Path navigated in simulation with currents and with small initial turn radius

Results for the position error Figures 4-11 and 4-12 show the position error of the vehicle over time at the straight section of the path, so at the path sections in between the black dots in Figures 4-9 and 4-10. For this configuration, only the position error in x direction is considered, since this says most about whether the vehicle is on track or not, which is crucial in a survey task. The error was defined as the reference value (x=0) minus the actual position value.

Figure 4-11 shows that in the no current scenario, the vehicle actually reaches the correct track and the initial error is quite small. In the other scenario, however, the vehicle is still 3 metres to the right of the track in its steady state situation. In addition, the error in the initial part of the straight track rises to 15 metres and it takes the vehicle much longer to reach its steady state situation. The red dot in both figures denotes the moment at which the vehicle is considered to have reached its steady state situation. This steady state situation was defined as the situation in which the vehicle stays within a fixed distance in x direction around the steady state x value, which is the value that the position in x direction converges to (as can be seen in Figures 4-9 and 4-10). The value of this fixed distance in x direction within which the vehicle has to stay was chosen to be five percent of 0.7 metres, which was the position error at the start of the straight track in the reference scenario with no currents and a large initial turn radius (which was the scenario that was used for the results shown in Figure 4-11). So the red dots in Figures 4-11 and 4-12 are placed at the points after which the position error stays within 0.035 metres of the steady state error value, which is 0.0 metres in Figure 4-11 and -3.1 metres in Figure 4-12. The red dots in these figures also mark the

moments at which the settling distance has been travelled, because the settling distance was defined as the distance in y direction from the start of the straight track to the point at which the vehicle reaches its steady state situation.

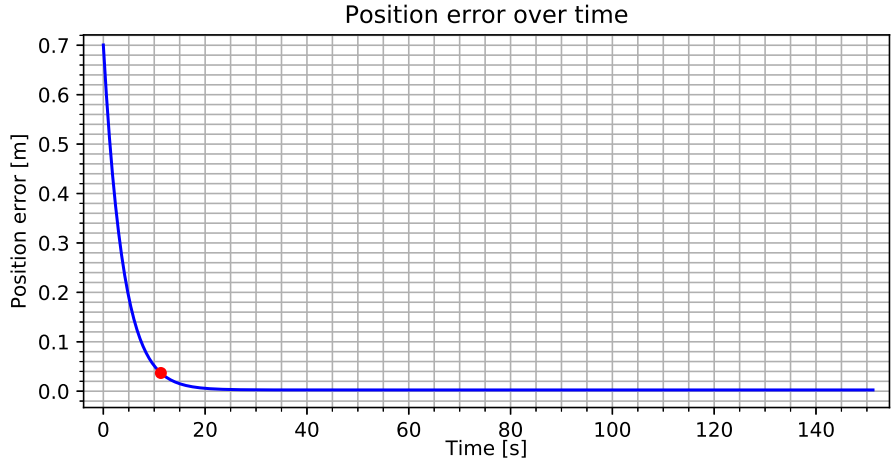


Figure 4-11: Position error in simulation without currents and with large initial turn radius

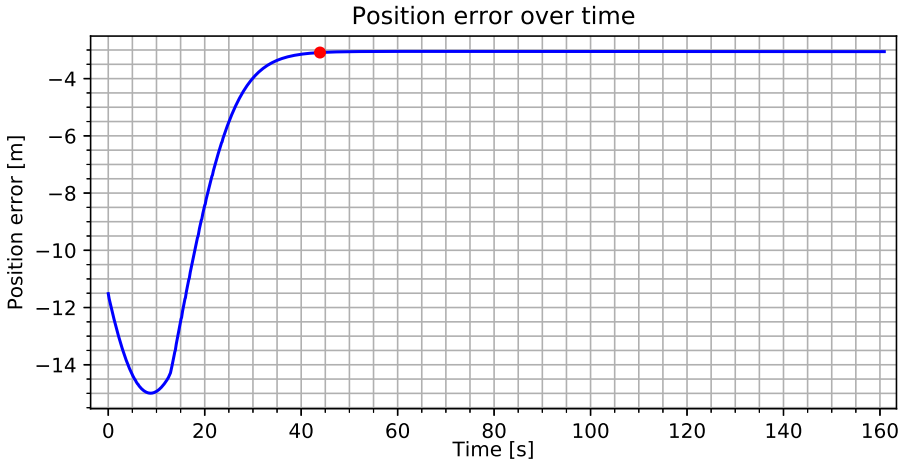


Figure 4-12: Position error in simulation with currents and with small initial turn radius

Results for the orientation error In Figures 4-13 and 4-14, the orientation error over time along the complete path is shown. The orientation error is defined as the reference orientation, which is the orientation that the vehicle should have on the straight track, minus the actual orientation in the global frame (with the x and y axes as shown in Figures 4-9 and 4-10 and the z axis pointing upwards). For the angles around the x and y axes, the reference value is 0, such that the vehicle is horizontal in the water with its antenna pointing upward. The reference for the angle around the z axis is set to 90 degrees, since the vehicle should travel along the positive y axis on the straight part of the path. The two vertical black lines in Figures 4-13 and 4-14 indicate the moments at which vehicle starts and stops travelling along the straight track of the planned path. In both simulations, the vehicle starts with its front

pointing in positive x direction, which corresponds to an actual angle around the z axis of 0 degrees. The orientation error for the z axis therefore is 90 degrees at the start of both simulations. In both cases, too, there are no large variations in the orientation for the x and y axes. There is, however, a major difference between both simulations when looking at the orientation error for the z axis. In Figure 4-13, the orientation error for the z axis reaches 0 degrees, which means that vehicle actually pointed in positive y direction when it was travelling the straight track. In Figure 4-14, on the other hand, the orientation error for the z axis does not converge to 0 and shows a large overshoot after the 90 degrees turn, which corresponds to what the travelled path in Figure 4-10 looks like.

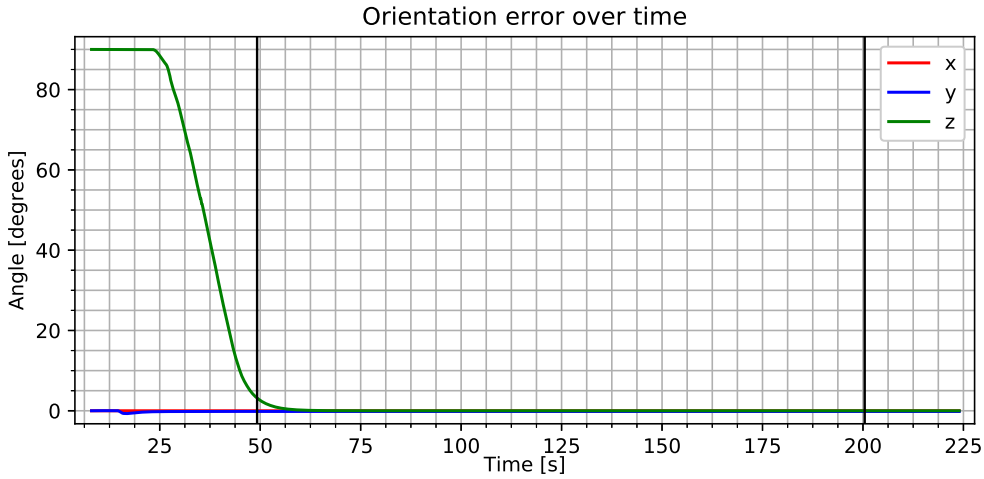


Figure 4-13: Orientation error in simulation without currents and with large initial turn radius

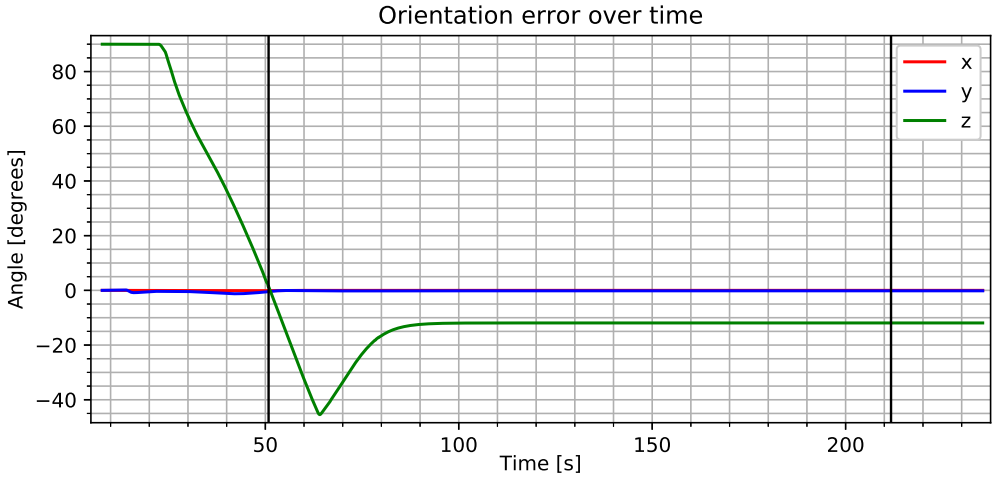


Figure 4-14: Orientation error in simulation with currents and with small initial turn radius

The six images that were described above indicate that in the first simulation the "propulsion and steering control configuration" worked well and could send the vehicle along the planned path, whereas in the second simulation the performance of the configuration was worse due to the changed conditions. Below, this observation will be explored by looking at the values that were obtained for the defined performance measures over all performed simulations.

Overall simulation results (measures of performance)

Results for the settling distance The first performance measure that was defined for this configuration is the settling distance, which was explained in the discussion of Figures 4-11 and 4-12. The values that were found for the settling distance in all simulation scenarios are shown in Figure 4-15.

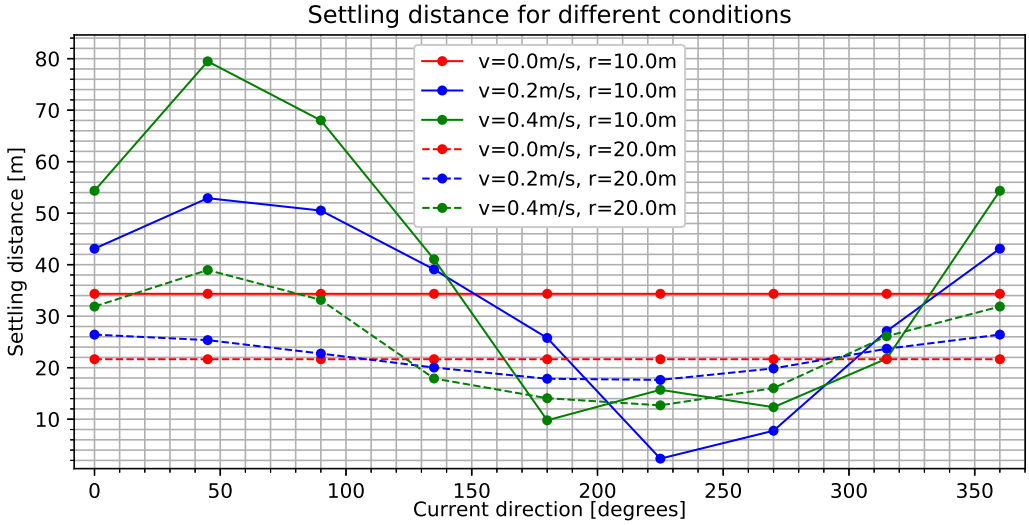


Figure 4-15: Settling distances. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

The figure shows that when the initial turn radius was small (indicated in the legend with "r=10.0m"), the effect of a change in the current direction on the length of the settling distance was more pronounced than it was for a larger initial turn radius (indicated in the legend with "r=20.0m"). In addition, a larger current speed ("v=0.4m/s") led to larger differences in the settling distance with a changing current direction than was the case for a current speed of 0.2 metres per second. Of the two no current conditions, the condition with the larger initial turn radius led to the smallest settling distance. Another observation that can be made from Figure 4-15 is the presence of currents can also have a positive effect on the settling distance, which is most obvious for the scenario with a current speed of 0.2 metres per second, a current direction of 225 degrees and an initial turn radius 10 metres (the minimum of the blue solid line in the figure). In the simulations for this scenario, the settling distance was about 20 metres smaller than in the no current situation with a large initial turn radius.

Results for the steady state position error In Figure 4-16, the values that were found for the second performance measure - the mean steady state position error in horizontal direction (x direction) with respect to the straight track - are shown for all scenarios. The values shown in this figure were calculated for the steady state situation, which means for the part of the straight track in which the effects caused by the initial 90 degrees turn have died out. This can be observed in the figure from the fact that the lines for the scenarios in which the current speed is the same but the initial turn radius differs overlap. In addition, the figure shows that the vehicle was right on track in the no current situations and in the cases where the current

flowed in the direction of the straight track or in opposite direction. On the other hand, when the current came in from the left side of the track, the vehicle had a position error to the right of the track and vice versa, which corresponds to what was expected. The largest mean position error (about 3 metres) was reached in the scenarios in which the current came in perpendicular to the track and the current speed was largest, which is also as expected.

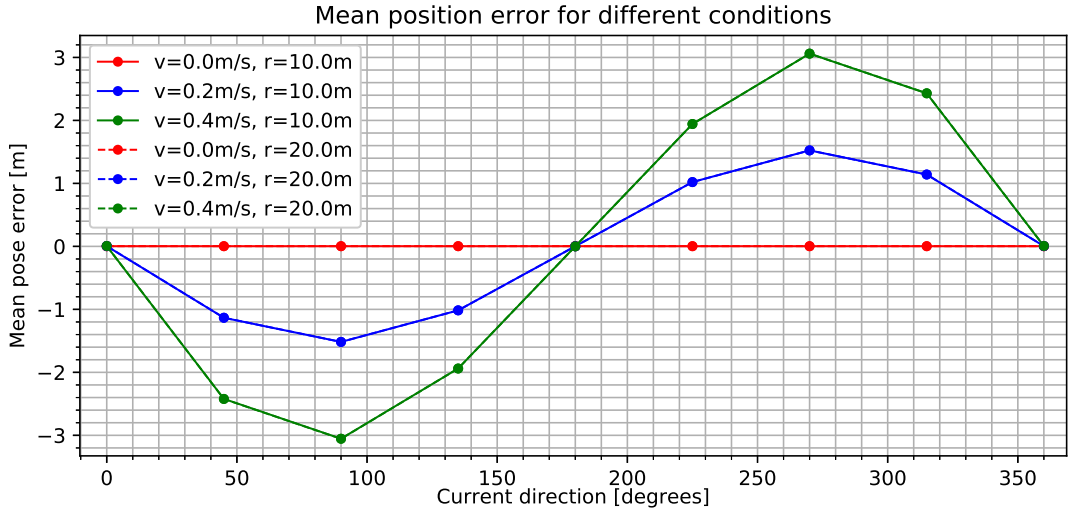


Figure 4-16: Mean steady state position error. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

Results for the steady state orientation error (in z, x and y direction) The third performance measure was the mean steady state orientation error of the vehicle. The error values that were found for the orientation in z, x and y direction are shown for all simulation scenarios in Figures 4-17, 4-18 and 4-19 respectively.

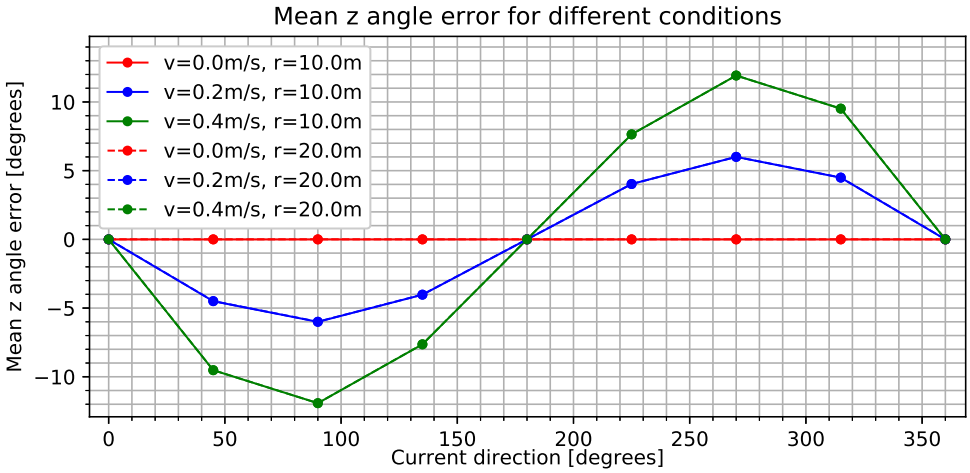


Figure 4-17: Mean steady state orientation error in z direction. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

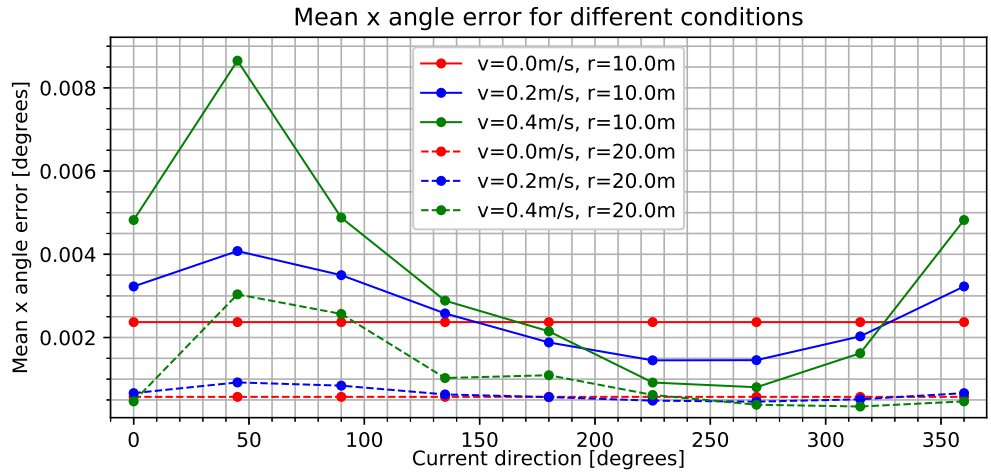


Figure 4-18: Mean steady state orientation error in x direction. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

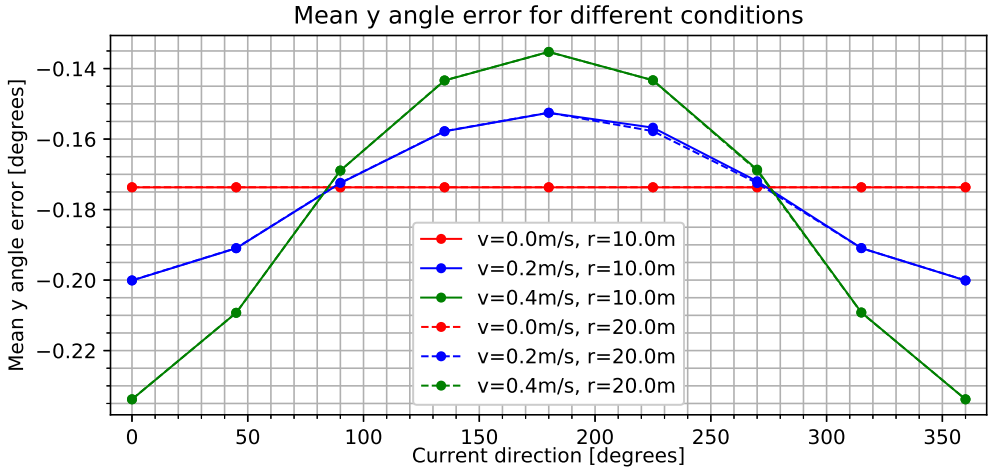


Figure 4-19: Mean steady state orientation error in y direction. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

The orientation error in z direction is shown first because it is somewhat easier to imagine and understand than it is for the other two directions, since the error in z direction corresponds to the heading of the vehicle with respect to the direction of the straight track. In Figure 4-17, the lines for the scenarios with the same current speed but different initial turn radius again overlap. The mean values for the orientation error in z direction show much more variation than for the x and y directions. This is as expected, since the LAUV uses a propeller and actuated fins to control its position and thus needs to vary its heading for changing its position. In the no current scenarios and the scenarios where the current flows in the direction of the straight track or in the opposite direction, the mean error is 0, which means that the vehicle's heading is equal to the direction of the straight track. Since for these scenarios the mean position error was also 0, the vehicle thus perfectly follows the straight track in these cases. The lines of Figure 4-17 actually very much resemble those of the mean position error, in Figure 4-16. This is intuitive, because when the position error is positive, the vehicle is on

the left side of the track and the vehicle's pose controller should thus make the vehicle head towards the track, which corresponds to a positive error for the orientation in z direction. Therefore, in Figure 4-17, too, the minima and maxima are reached for the scenarios where the currents flows perpendicular to the straight track.

For the orientation error in x direction, the mean values (shown in Figure 4-18) are very small, which is as expected. However, the fact that the lines for the scenarios with the same current speed but different initial turn radius do not overlap is still conspicuous. For both the z and y directions (Figures 4-17 and 4-19), these lines do overlap. The initial turn radius, therefore, does seem to have an effect on the mean orientation error in x direction in the steady state situation, although this effect is very small and may be negligible in actual operations.
In the blue and green solid lines in Figure 4-18, which represent the scenarios with currents and a small initial turn radius, also a trend with changing current directions can be recognised. The explanation for these variations in the mean value of the orientation error in x direction was found by looking at the data that was recorded in individual simulations. The recorded values for the orientation error in x direction over time showed that this error generally had not yet reached a steady state value at the moment that was defined to be the start of the steady state situation for the vehicle (which was based on the position error in x direction). Therefore, the mean values shown in Figure 4-18 do not represent the steady state orientation error in x direction, but they reflect the effects of the control efforts of the vehicle to get through the initial turn.

For the orientation error in y direction, shown in Figure 4-19, a clear trend with changing current directions is visible. Just as for the x direction, the values for the mean orientation error in y direction are quite small, but in this case it is clear that the absolute value of the error is maximal when the current flows in the direction of the straight track and minimal when it flows in opposite direction. The data that was recorded for the orientation error in y direction in individual simulations showed that the mean values in Figure 4-19 do represent the steady state values of this error. The values in Figure 4-19 thus correspond to the steady state error for the orientation in y direction. The fact that this steady state error varies with the current direction and speed suggests that this error is the result of hydrodynamic effects of the current on the vehicle.

Results for the steady state speed error The fourth performance measure for this configuration is the mean steady state speed error. The values that were found for this error for all simulation scenarios are shown in Figure 4-20. In this figure, the lines for the scenarios with the same current speed but different initial turn radius again overlap. It is useful to mention here that the PID controller that was implemented in the virtual vehicle for control of propulsion and steering was still under development at the time of simulating and especially the control of the vehicle's speed was not yet as desired. The fact that the mean speed error was about 0.1 metres per second in the no current scenarios illustrates this. However, the values that were found and especially the trends in those values can be used to say something about the configuration performance. The figure shows, for instance, that the maxima for the speed error, and thus the minima for the actual speed, were reached when the current flowed in the direction opposite to that of the straight track, which is as expected. In addition, the values of the speed error for scenarios where the current flowed perpendicular to the straight track were very similar to those of the no current scenarios and the minimum speed error

values (and thus maximum speeds) were reached when the current flowed in the direction of the straight track.

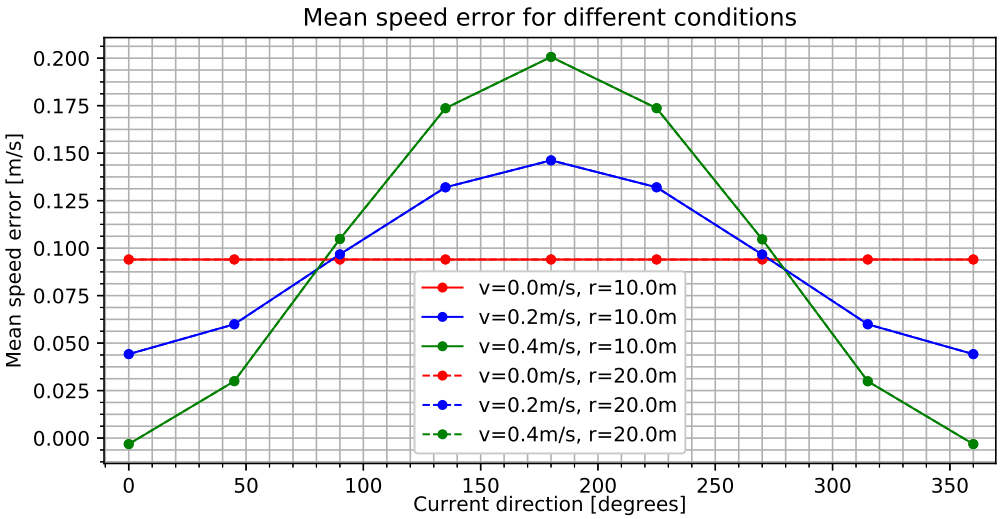


Figure 4-20: Mean steady state speed error. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

Results for the standard deviations of the steady state orientation and speed errors Figures 4-21 and 4-22 show the values that were found for the last two performance measures: the standard deviations of the orientation error in z direction and of the speed error, both for the steady state situation. The mean values for these two errors were shown in Figures 4-17 and 4-20.

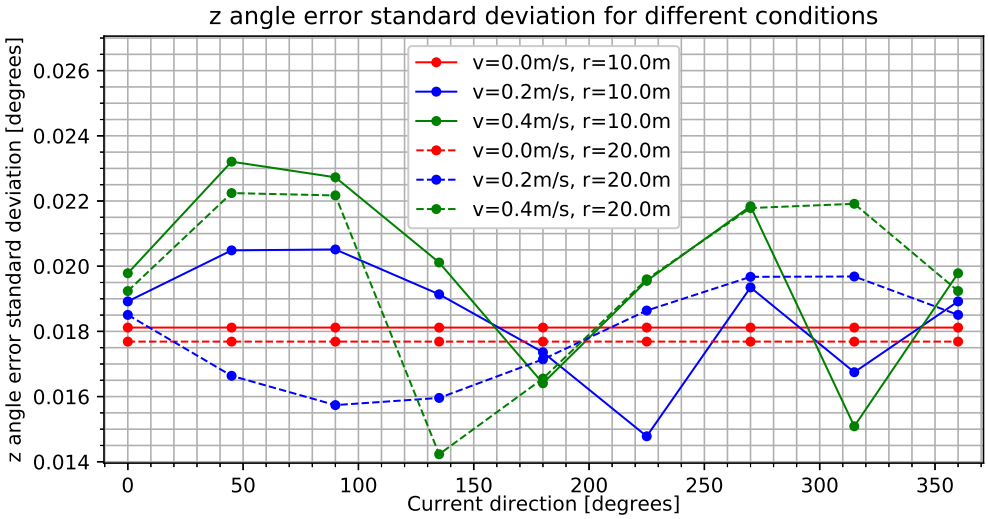


Figure 4-21: Standard deviation of the orientation error in z direction. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

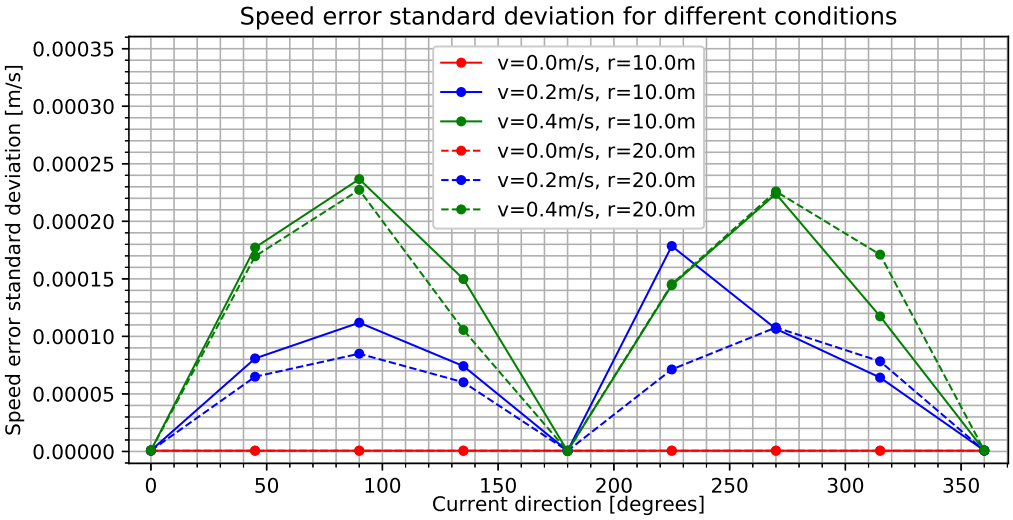


Figure 4-22: Standard deviation of the speed error. In the legend, "v" is short for current speed and "r" is short for initial turn radius.

The standard deviations for these two errors give an indication of how large the variations in the vehicle's heading and speed are and thus how steady the vehicle is travelling along the straight track of the path. The magnitude of these variations is important for a survey task because large variations in the heading and speed can be detrimental to the quality of the sonar images that are collected during surveying. For both errors, the values for the standard deviation are quite small compared to the values that were found for the mean error, so both the orientation in z direction and the speed had very constant values in the steady state situations. This indicates that, in the scenarios that were tested, the vehicle had a very stable heading and speed, which would have been favourable for the imaging quality if the vehicle had been performing an actual survey task.

Application of the simulation results

To illustrate what the described simulation results can be used for, a small performance assessment use case for the "propulsion and steering control configuration" will be described here. The search task of the LAUV will be used as a reference.
The situation of the use case is as follows: a LAUV with the "propulsion and steering control configuration" of Figure 4-4 is tasked with navigating a lawnmower pattern over an area of the seabed in order to search this area for objects. To survey the area properly, the LAUV should travel along a planned lawnmower path consisting of straight tracks and turns. In the survey area, there is a current in a direction that is perpendicular to the direction of the straight tracks. This situation is illustrated in Figure 4-23, where the black dotted line represents the planned path and the blue arrow represents the current. The question now is how well the LAUV is expected to follow the straight tracks of the planned path, given the current speed and the radius of the turns in the planned path. To answer this question, the obtained results for the settling distance and for the mean steady state position error can be used. Suppose that the current speed is 0.4 metres per second and that the turn radius is 20 metres, the expected values for the settling distance and for the steady state position error

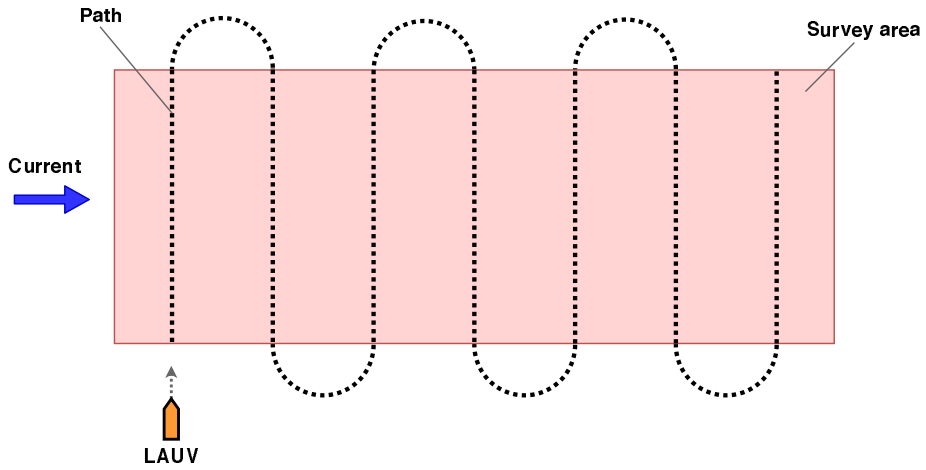


Figure 4-23: Illustration of the example use case

in horizontal direction then are 33 metres and 3.0 metres respectively (as read from Figure 4-15 and 4-16). In the situation as it is illustrated in the figure, the LAUV would thus travel 3 metres to the right of the straight tracks after using 33 metres in vertical direction on each track to achieve a steady state. This information can be used as a prediction of performance, but could also be exploited to improve performance; the straight tracks of the planned path could for instance be extended by 33 metres and shifted 3 metres to the left to compensate for both the expected settling distance and the expected position error. Other performance measures, such as the expected orientation error and the expected standard deviation of the speed error, could be used to make an estimate of the expected imaging quality.

4-2-2 Simulations for the localisation configuration

Simulation scenarios

In the simulations for the "localisation configuration", the LAUV was given a path in the form of a lawnmower pattern and during the simulations the actual position of the vehicle and its estimated position, with covariances, were recorded. The conditions that were varied for the different simulations are the combination of sensors that was used, the length of the legs in the lawnmower pattern and the number of legs in between GPS fixes. For the turns in between the legs in the lawnmower pattern, a fixed radius of 20 metres was used. The different conditions that were used are shown in Table 4-6. Simulations were done for all different combinations of values for the three conditions in this table, so, in total, there were 18 different simulation scenarios. The lengths of the legs were chosen to be similar to the leg lengths that are used at TNO for trials with the LAUV. The values for the number of legs in between GPS fixes were selected based on the position error growth that was observed in test simulations. For each of the 18 different simulation scenarios, one or more simulations were performed such that usable data was obtained for at least ten sets of consecutively performed legs in between GPS fixes. The simulations always started with the LAUV at the water surface to enable it to perform a GPS fix before starting with the planned lawnmower path.

Table 4-6: Conditions used for simulations for the localisation configuration

Condition	Values
Combination of sensors systems	IMU, GPS
	IMU, GPS, DVL
Leg length	300 m
	550 m
	800 m
Number of legs between GPS fixes	3 tracks
	4 tracks
	6 tracks

Results for two single simulations

As was done for the "propulsion and steering control configuration" in the previous subsection, first the data that was recorded in two single simulations will be presented to show what happened in these simulations for two of the 18 different scenarios. In these two scenarios, the same path was used (a lawnmower pattern with a leg length of 550 metres, three legs in between GPS fixes and a total number of 30 legs), but in one scenario only the IMU and the GPS were used for localisation, while in the other scenario the DVL had been added to the configuration. In all cases, the data again was recorded with a frequency of 10 Hz.

Results for the travelled path In Figures 4-24 and 4-25, the path that the vehicle has travelled is shown for the two selected simulations. Figure 4-24 shows the travelled path for the simulation in which only IMU and GPS were used in the configuration and Figure 4-24 shows it for the simulation in which DVL was used additionally. In both cases, the vehicle started at position (0,0), which is in the upper left corner of both figures. The blue line in both figures represents the vehicle's estimated position over time and the red line represents the path that the vehicle actually travelled. On these lines, the start and end points of the legs are shown with dots in the respective colours. The blue lines very accurately follow the planned path (not shown in the figures), which indicates that the vehicle would have been able to accurately follow this path if it would have had access to its real position. Due to errors in the estimated position, however, the path that the vehicle actually travelled sometimes diverges from the planned path, especially in Figure 4-24.

The depth of the vehicle is not shown here, but after every three legs the vehicle went to the water surface to take a GPS fix and then went back to navigation depth (6 metres below the water surface) to continue the lawnmower path. The vehicle was programmed to go to the surface and back in a turn (helix), which explains the circular path sections after every three legs in both figures. In Figure 4-24, the points at which the GPS fixes are actually performed can be observed in the estimated path (with some effort). At these points, the estimated position suddenly shifts, and afterwards, the estimated path is aligned with the actual path for a while. The reason for this shift in the red line is that the error in the estimated position, which has grown after travelling a few legs under water due to integration drift, is reset to 0 by taking an accurate position measurement via the GPS. In Figure 4-25, these shifts in the estimated position can not be observed because the position error just before a GPS fix is much smaller than it was in the simulation for Figure 4-24. Along the whole path,

actually, the error in the estimated position is much smaller in Figure 4-25 than in Figure 4-24, as can be seen from the fact the red and blue line are much more aligned in Figure 4-25 than in Figure 4-24. This result is as expected since in the simulation of Figure 4-25 the localisation configuration was extended with a DVL. This DVL provided measurements of the vehicle's velocity and by combining these with the measurements outputted by the IMU in a sensor fusion algorithm (in this case a Kalman Filter), more accurate position estimates were obtained under water than was the case when using only an IMU.

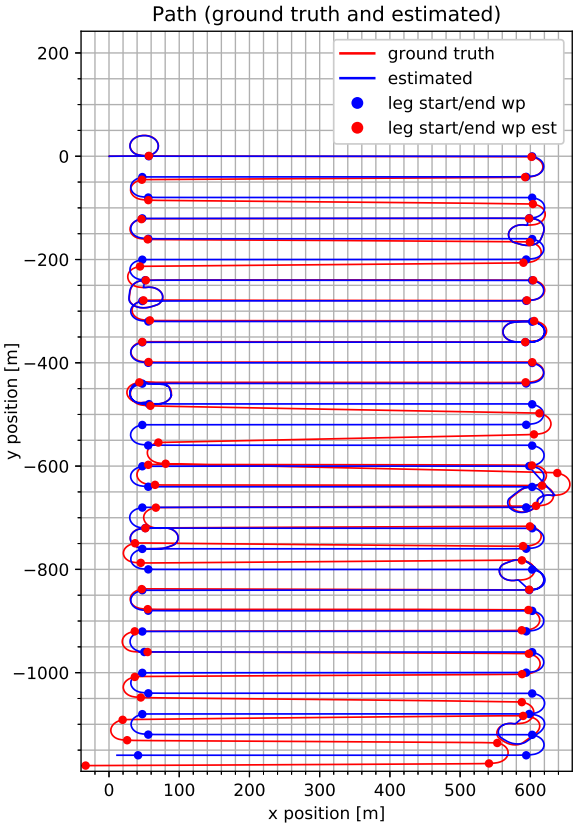


Figure 4-24: Path travelled in simulation without DVL

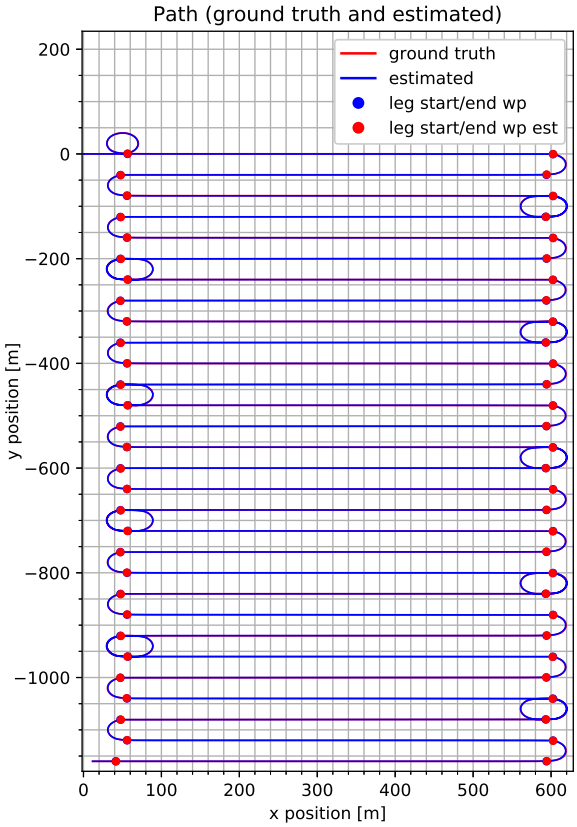


Figure 4-25: Path travelled in simulation with DVL

Results for the position error and the position uncertainty Figures 4-26 and 4-27 show the position error and the position uncertainty over time for both simulations. The position error is calculated as the difference between the estimated position and the actual position and the position uncertainty is calculated by taking the square root of the individual covariance values for the position in x and y direction as outputted by the Kalman filter.

Both the error and the uncertainty are shown separately for the x direction and the y direction. As can be seen in Figure 4-24 and 4-25, the x direction corresponds to the direction that the legs are oriented in and the y direction is perpendicular to the legs. For this configuration, only the position error and uncertainty in x and y direction were analysed, since the error

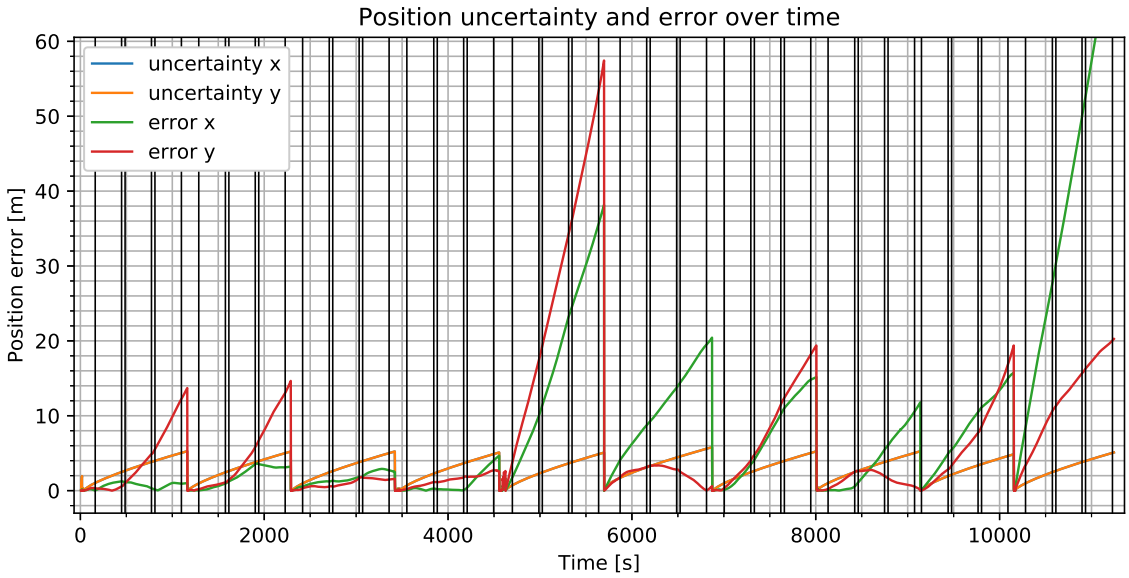


Figure 4-26: Position uncertainty and error over time in simulation without DVL

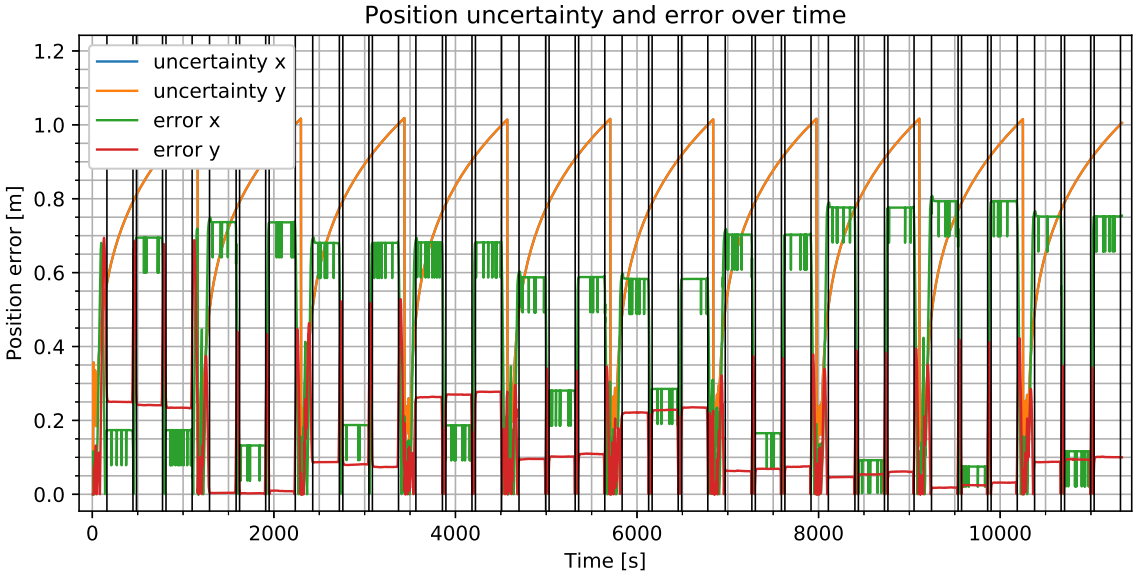


Figure 4-27: Position uncertainty and error over time in simulation with DVL

and uncertainty in these directions are most important for the coverage performance of the vehicle. The vertical black lines in Figures 4-26 and 4-27 indicate the moments at which the vehicle was at the start or end point of a leg.
In both simulations, the covariances outputted by the Kalman filter were the same for the x and y directions, so in Figures 4-26 and 4-27, the lines for the uncertainty in x and y direction (in blue and orange respectively) overlap. Another similarity between both figures is that the moments at which a GPS fix is performed can be clearly identified by the steep drops in the uncertainty values. In many other aspects however, the figures are very different.

The first major difference is that the uncertainty and especially the error reach much higher values in the simulation without DVL than in the one with DVL. For the position error, this corresponds to what was seen in Figures 4-24 and 4-25 for the travelled path in both cases. In addition, the progression of both the uncertainty and the error is very different for the two figures. In Figure 4-26, the growth of the uncertainty in between GPS fixes seems to be linear, as is the case for the error, although less clearly so. In Figure 4-27, on the other hand, the growth of the uncertainty seems to follow the line of a square root function. The position error in this case seems to be constant over a leg, where the error in y direction has similar values for all three legs in between GPS fixes and the error in x direction seems to switch between a high and a low value. The small peaks in the green line in Figure 4-27 are thought to be caused by small differences in timing between the updates of the actual position and the estimated position during the simulation (the position error is calculated as the difference between the actual and estimated positions).

Trend lines through the uncertainty data In Figures 4-28 and 4-29, the uncertainty data that was shown in Figures 4-26 and 4-27 is shown again, but then cropped for each set of three legs in between GPS fixes and normalised for time. In Figure 4-28, the data is also normalised for the first uncertainty value in the first leg, such that all lines start at the point (0,0). These figures thus show the growth of the uncertainty over three consecutive legs for ten different sets of legs in between GPS fixes, as indicated by the ten coloured lines in both figures.

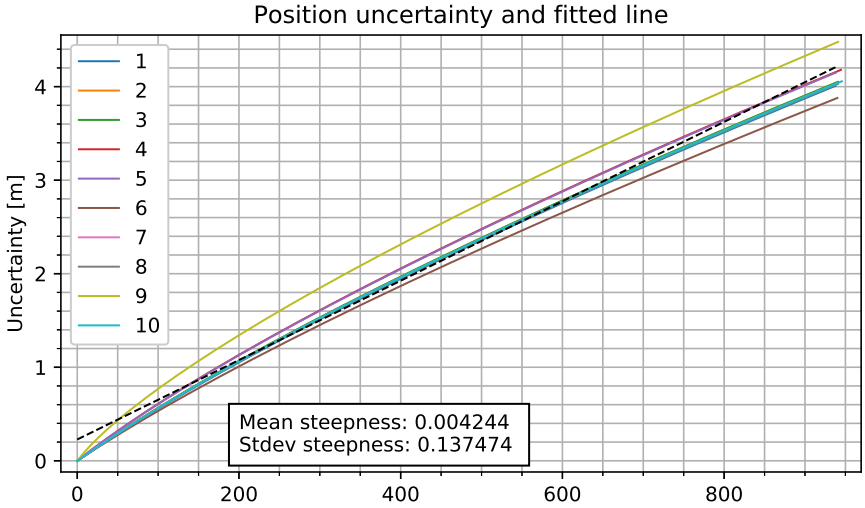


Figure 4-28: Normalised uncertainty data of simulation without DVL and the mean trend line fitted through this data

In Subsection 4-1-3, it was mentioned that two of the performance measures that were defined for this configuration are the type of line that can be fitted through the pose uncertainty and a parameter that characterises this fit. In the simulations that were performed, only the uncertainty for the position in x and y direction was taken into account for analysing performance and the data as it is shown in Figures 4-28 and 4-29 was used for this analysis. To illustrate this, both figures show a black dashed line that represents the mean trend line fitted through all ten lines in the figures. For Figure 4-28, a linear line was fitted through

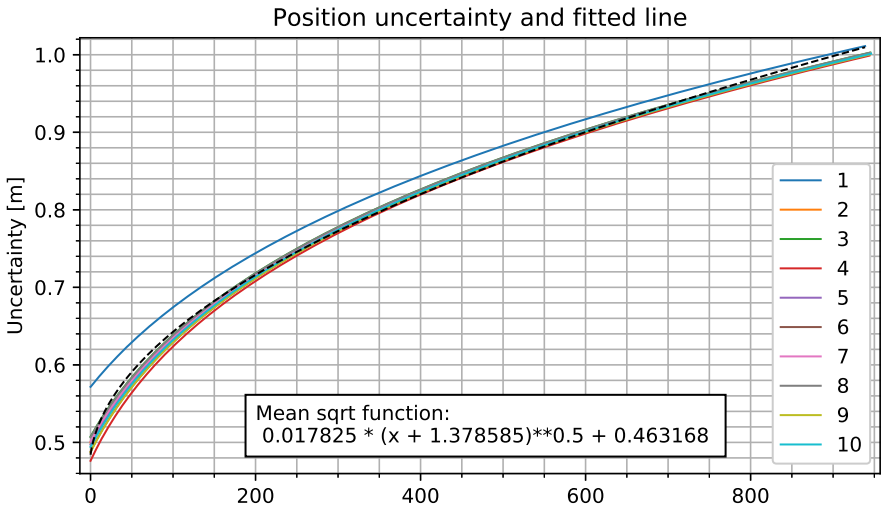


Figure 4-29: Normalised uncertainty data of simulation with DVL and the mean trend line fitted through this data

the normalized uncertainty data and the corresponding steepness of the line is shown in the white box in the figure, together with the standard deviation of the fit (which will be explained below). This steepness value can be used as a parameter for characterising the fitted line. For Figure 4-29, a square root function was fitted through the ten lines in the figure, and the parameters that were found for this function are shown in the white box in the figure. Below, the methods used for fitting these lines and the values that were obtained for the performance measures will be discussed.

Overall simulation results (measures of performance)

As was shown in Figures 4-26 and 4-27, the progression of the position uncertainty and the position error is very different when a DVL is or is not used in the "localisation configuration". This also affects the way in which the values for the performance measures for the configuration should be determined. Therefore, for most of the performance measures defined in Subsection 4-1-3, the values that were found will be shown here in separate figures for the cases without and with DVL.

The first three performance measures that will be discussed here are the type of line that can be fitted through the position uncertainty data, a parameter that characterises this fit and a value indicating the goodness of fit. Only the position uncertainty in x and y directions will be taken into account here and in all simulations the uncertainty outputted by the Kalman filter was the same for the x and y directions.

Type of trend line and parameter characterising the fit for the position uncertainty data
For the scenarios without DVL, the uncertainty seemed to grow in a linear manner in between GPS fixes, so the first performance measure was given the value "linear" for these cases. For the scenarios with DVL, however, a square root function was used to fit lines through the uncertainty data, so the first performance measure was given the value "square root".

The definition of the basic square root function that was used is given in Equation 4-1 and was already shown in Figure 4-29. The parameters c_1, c_2 and c_3 in this function represent constants.

$$f_{fit}(x) = c_1 \cdot \sqrt{(x + c_2)} + c_3 \tag{4-1}$$

Since the type of line that was used for fitting trend lines through the position uncertainty data was different for the cases without and with DVL, the parameter that was used as a characterisation of the fit was also different for these cases. For the cases without DVL and thus with linear trend lines, the steepness of these fitted lines was considered to be a suitable parameter. Therefore, for the simulation scenarios without DVL, the data that was recorded for all sets of legs in between GPS fixes was combined for each particular scenario, and linear lines were fitted through the combined uncertainty data. This fitting process was performed using Python, and more specifically the command `polyfit` of the numpy package [64]. This command can be used to obtain a least squares first degree polynomial fit through a dataset. The resulting values for the steepness of the lines fitted through the uncertainty data for the 9 different simulation scenarios without DVL are shown in Figure 4-30. The figure shows that the value of the steepness of the fitted line decreases with an increase of the leg length (indicated in the legend) and with an increase of the number of legs between GPS fixes (indicated along the x axis). The conclusion that can be drawn from this is that the steepness of the line fitted through the uncertainty data decreases as the distance that the vehicle consecutively travels under water increases.

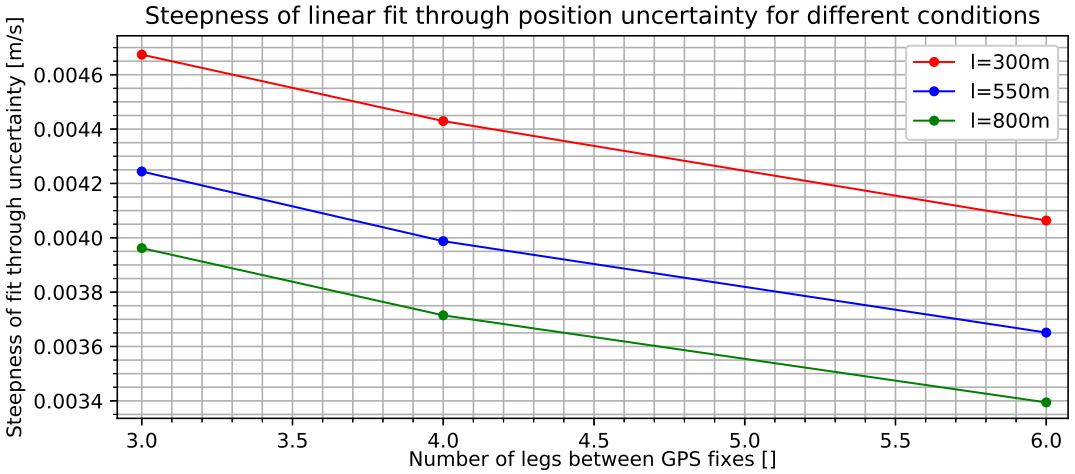


Figure 4-30: Steepness of fit through uncertainty data (without DVL). In the legend, "l" is short for leg length.

For the cases with DVL, the function of Equation 4-1 was used to fit trend lines through the uncertainty data. The first parameter (c_1) in this equation was considered to be the most significant characteristic of this function, so this parameter was used as the second performance measure (the parameter that characterises the fit) for the uncertainty data. The values that were found for this parameter are shown in Figure 4-31 for the 9 different simulation scenarios with DVL. These values were obtained by fitting a line through the uncertainty data using the command `curve_fit` of the scipy.optimize package in Python [65]. With this command, a function can be fitted through data using a non-linear least

squares algorithm. In this case, the function of Equation 4-1 was given as the model function for the fitting algorithm.
Figure 4-31 shows that the value of factor c_1 of the fitted function decreases with an increase of the number of legs between GPS fixes and with an increase of the leg length. This indicates that the value of this factor decreases when the distance that the vehicle consecutively travels underwater increases.

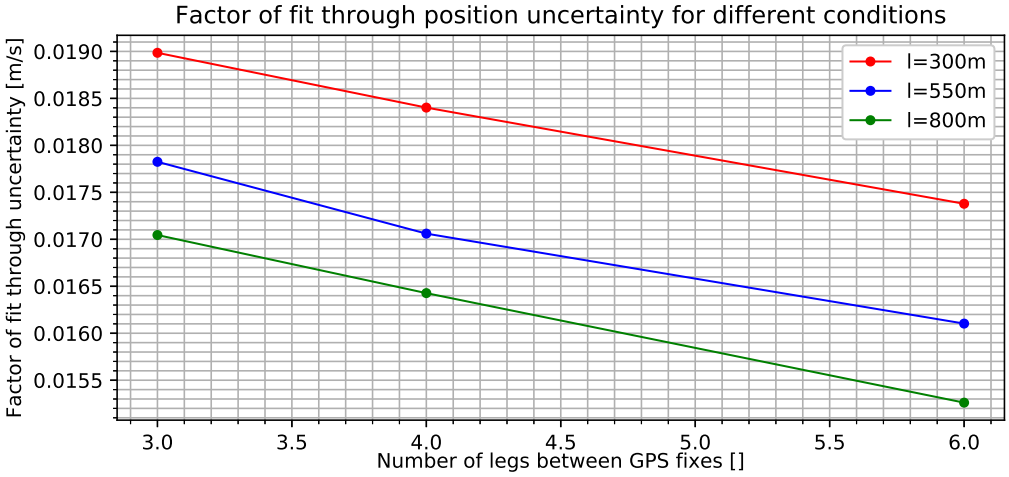


Figure 4-31: Factors for fits through uncertainty data (with DVL). In the legend, "l" is short for leg length.

Goodness of fit for the position uncertainty data The third measure of performance for the "localisation configuration" is a parameter that should indicate the goodness of fit for the fitted trend lines. The parameter that was chosen for this is the standard deviation of the difference between the data points and the fitted line. The value for this standard deviation (σ_{fit}) was calculated by taking the square root of the mean of the square of the difference between data points and the corresponding value on the fitted line, as shown in Equation 4-2.

$$\sigma_{fit} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f_{fit}(x_i))^2} \tag{4-2}$$

In this equation, N is the total number of data points, (x_i, y_i) is a specific data point and f_{fit} is the function that represents the fitted line.
The values that were found for σ_{fit} for the position uncertainty in the scenarios without DVL are shown in Figure 4-32. For these scenarios, the function representing the fitted line can be written as $f_{fit}(x) = c_1 \cdot x + c_2$, where the parameter c_1 represents the steepness of the fitted line. The figure shows that the standard deviation for the fit through the uncertainty data is quite small in all cases, which indicates that the fitted linear lines form a good representation of the data sets.

In Figure 4-33, the values that were found for σ_{fit} for the position uncertainty in the scenarios with DVL are shown. For these cases, the function f_{fit} was equal to the function of Equation 4-1. The figure shows that the obtained values are very small for all scenarios, so the fitted lines form a good representation of the uncertainty data.

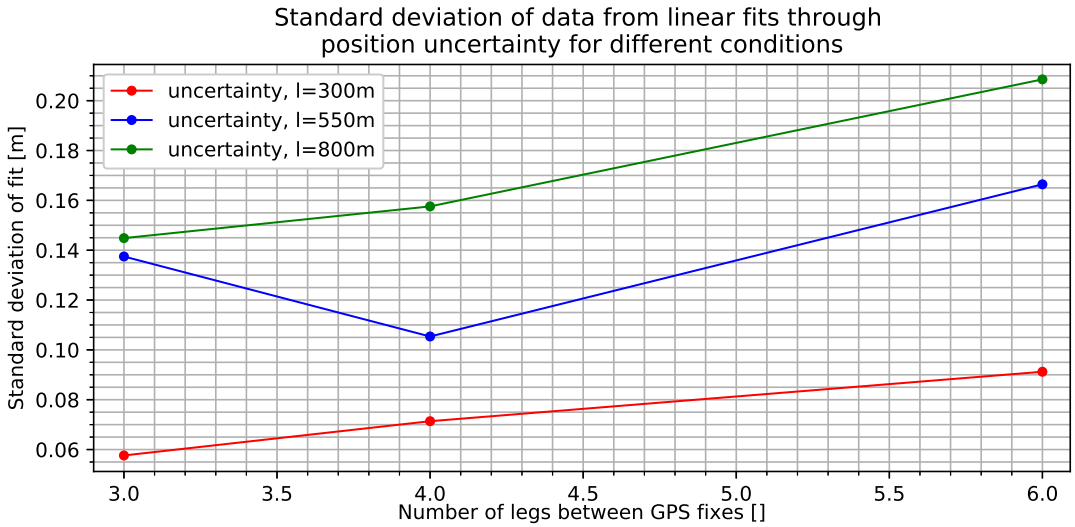


Figure 4-32: Standard deviations for fits through uncertainty data (without DVL). In the legend, "l" is short for leg length.

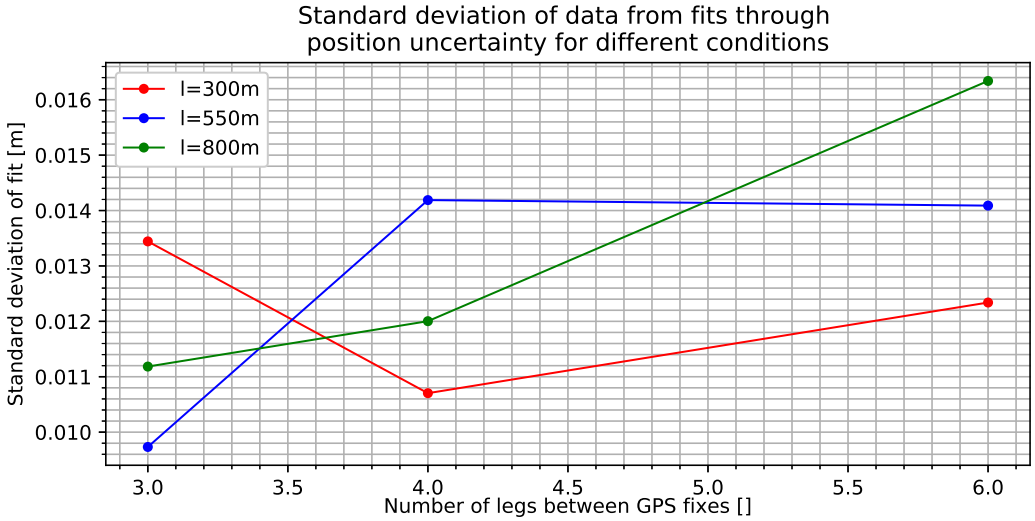


Figure 4-33: Standard deviation for fits through uncertainty data (with DVL). In the legend, "l" is short for leg length.

The three performance measures that were described above for the position uncertainty data were also used for the position error data that was recorded in all simulations for the "localisation configuration", as will be explained below. Again, only the position error in x and y direction was taken into account.

Type of trend line and parameter characterising the fit for the position error data In the scenarios without DVL, the position error showed a linear increase over time, so the type of trend line that can be fitted through the data is "linear". In addition, the steepness of these linear trend lines can be used to characterise the fit, in the same way as was done for the position uncertainty data for the cases without DVL. The resulting steepness values

that were found for the position error in x and y direction in all 9 scenarios without DVL are shown in Figure 4-34. The steepness values for the position error data show much more variation than those found for the uncertainty data, which were shown in Figure 4-30. This is as expected, because the growth of the position error over time showed much more variation over different leg sets than the growth of the position uncertainty, as was illustrated in Figure 4-26. In addition, the trend that was seen in the steepness values for the position uncertainty in Figure 4-30 is not obvious in the steepness values for the position error in Figure 4-34.

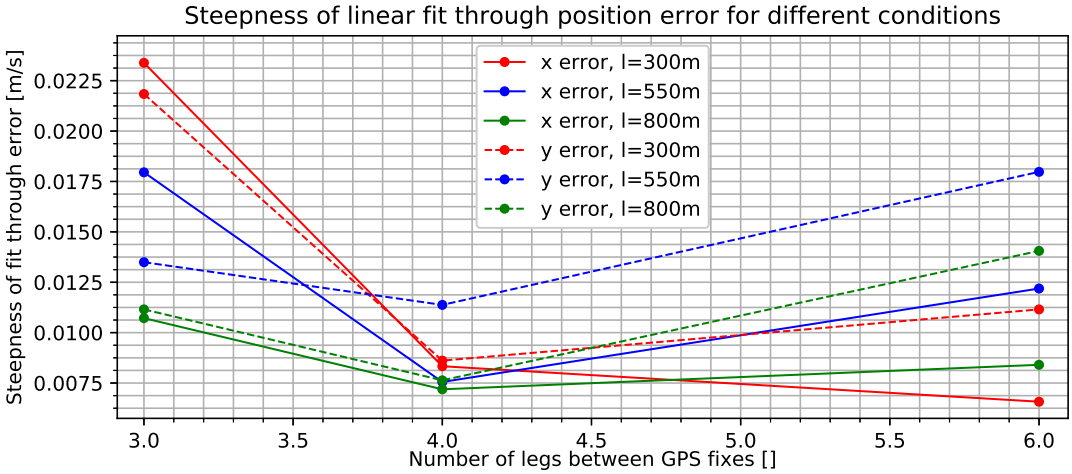


Figure 4-34: Steepness of fit through error data (without DVL). In the legend, "l" is short for leg length.

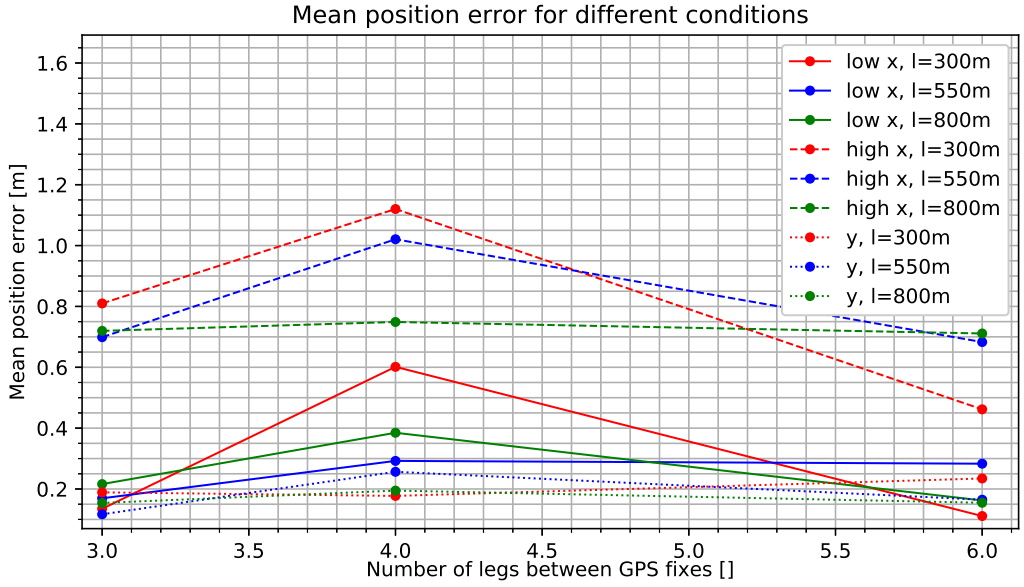


Figure 4-35: Mean position error values (with DVL). In the legend, "l" is short for leg length.

For the scenarios with DVL, the position error data did not show a continuous trend over consecutive legs in between GPS fixes, as was shown in Figure 4-27, so it is not desirable to fit a continuous function through this data. However, the position error values do seem to

be constant over a leg and, in general, the error in y direction has similar values for all legs in a set of consecutive legs and the error in x direction seems to switch between a high and a low value. Therefore, the decision was made to use a horizontal trend line to approximate the position error in y direction in between GPS fixes and a discontinuous horizontal trend line that switches between two values for the error in x direction. The constant error values corresponding to these horizontal trend lines were chosen to be equal to the mean values of the position errors over all leg sets that were recorded for a scenario, using one mean value for the y direction and two mean values ("low" and "high") for the x direction. These mean error values therefore also are suitable parameters for characterising the fit of the trend lines through the position error data for the scenarios with DVL. The three mean error values were thus calculated for every scenario with DVL and are shown in Figure 4-35. The figure shows that there is some variation in the values over the different scenarios. Based on these results, however, no conclusion can be drawn about trends in the mean error values over a changing number of legs or over a changing leg length.

Goodness of fit for the position error data For the third performance measure for the position error data (the value indicating the goodness of fit), again the function of Equation 4-2 was used. The function of the fitted line (f_{fit}) corresponded to a linear line in the cases without DVL, with the steepness of the line equal to the values shown in Figure 4-34. For the cases with DVL, the constant values of Figure 4-35 were used as f_{fit}. The resulting values that were found for σ_{fit} are shown in Figure 4-36 for the cases without DVL and in Figure 4-37 for the cases with DVL.

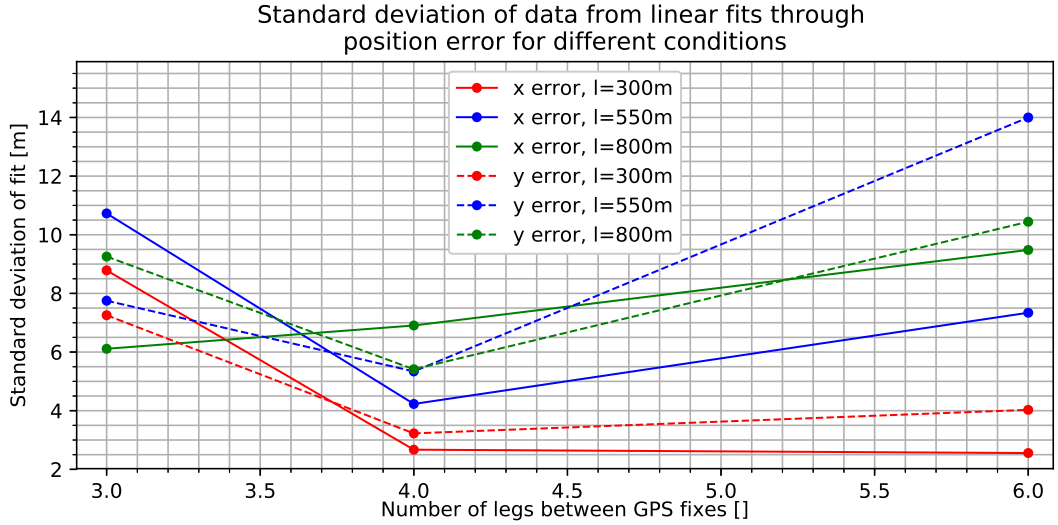


Figure 4-36: Standard deviations for fits through error data (without DVL). In the legend, "l" is short for leg length.

Figure 4-36 shows that the standard deviations for the lines fitted through the position error data for the cases without DVL are very high, especially when compared to the standard deviations that were found for the lines fitted through the uncertainty data (shown in Figure 4-32). These high values for the standard deviations could also explain why it is hard to find a trend in the corresponding steepness values of Figure 4-34. In part, these high standard

deviation values can be explained by the relatively low number of simulations that were performed for each scenario. As was illustrated in Figure 4-26, the growth of the position error differs much more for the different leg sets than it does for the position uncertainty, so a line fitted through the position error data will have a worse value for the goodness of fit than a line fitted through the position uncertainty data for the cases without DVL.

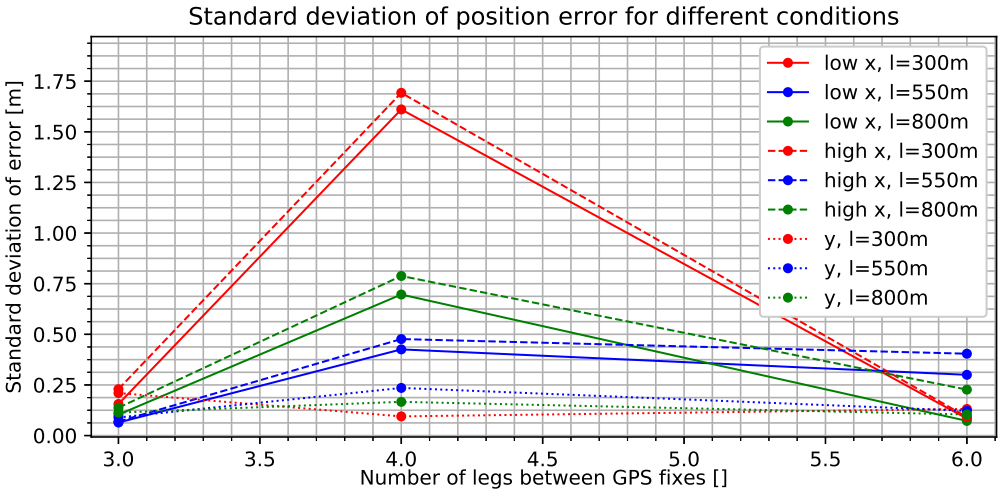


Figure 4-37: Position error standard deviation (with DVL). In the legend, "l" is short for leg length.

For the scenarios with DVL, too, the standard deviations for the lines fitted through the position error data are quite high, although generally much lower than those found for the scenarios without DVL. By comparing the mean error values of Figure 4-35 with the standard deviations of Figure 4-37, the conclusion can be drawn that, although the position error values are constant over a leg, the variation of this error value over all legs for a simulation scenario is quite large. For some scenarios, the value of the standard deviation is even larger than the value of the corresponding mean. Most of the times, this was caused by outliers in the position error data, which are thought to be caused by erroneous processes in the virtual Kalman filter when a GPS fix is performed. Performing more simulations for each scenario can help to attenuate the effect of these outliers in the data on the obtained values for the mean and standard deviation of the position error.

Coherence of fits through the position error and uncertainty data Apart from the three performance measures that were described above for both the position uncertainty and the position error, a fourth performance measure was defined with the aim of comparing the uncertainty data with the error data to get an indication of how well the position uncertainty values represent the actual position error. Since trend lines were used to characterise the position uncertainty and error for each scenario, these fitted lines will be used for comparing the uncertainty data with the error data.
For the scenarios without DVL, a linear trend line was used for both the uncertainty data and the error data. Therefore, the ratio between the steepness values of the trend lines for the position error and for the position uncertainty was chosen as the measure to represent the coherence between the fitted lines. The values that were found for this ratio are shown

in Figure 4-38 for all 9 scenarios without DVL. Not surprisingly, the values obtained for this performance measure show a similar pattern as the steepness values for the trend lines through the position error data, shown in Figure 4-34, since the steepness values for the position uncertainty are relatively small and are similar for the different conditions. Another observation that can be made from Figure 4-38 is that the obtained values are quite high in most cases, which means that the steepness values for the position error are much higher than those of the position uncertainty. This indicates that the position uncertainty values outputted by the Kalman filter, as it was implemented in the virtual vehicle at the time of performing the simulations for this configuration, do not provide an accurate estimate of the actual position error.

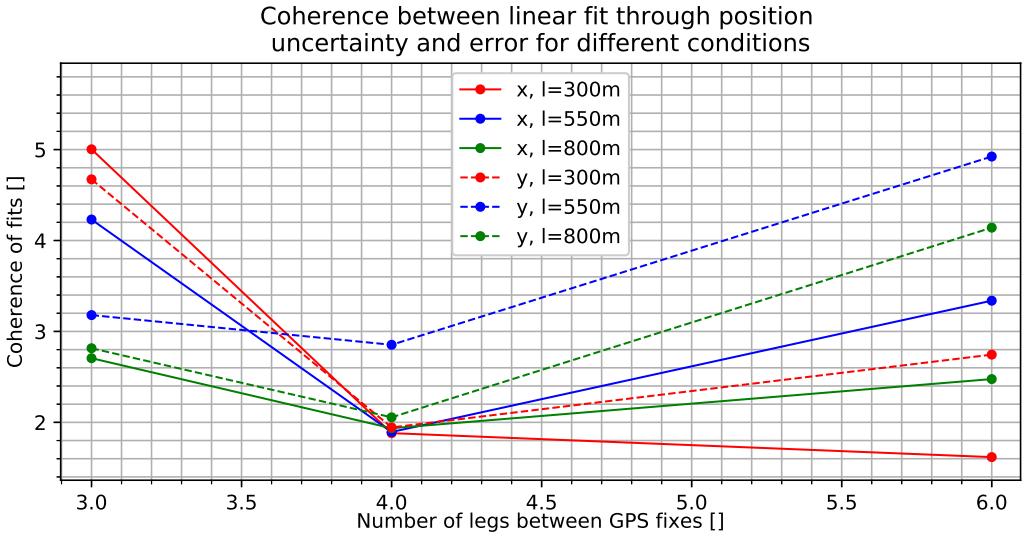


Figure 4-38: Coherence of fits through error and uncertainty data (without DVL). In the legend, "l" is short for leg length.

For the scenarios with DVL, the type of trend line that was used for the uncertainty data is different from the type used for the error data and the parameters that were used for characterising both fits thus are different as well. Therefore, the lines fitted through the position uncertainty data and through the position error data could not be compared in the same way as was done for the cases without DVL and thus the conclusion was drawn that there is no coherence between the fitted lines for the cases with DVL. The values for this last performance measure thus were set to "none" for the scenarios with DVL.

Application of the simulation results

For this configuration too, the situation in which the LAUV has to navigate a lawnmower pattern over an area of the seabed can be used to illustrate how the described simulation results can be applied. The values that were obtained for the performance measures under different conditions can be used to predict how the "localisation configuration" of the LAUV will perform. For instance, if the LAUV does not have a DVL and has to travel a lawnmower path with legs of 550 metres and with a GPS fix after every 6 legs, the results described above can be used to make predictions of the position uncertainty and position error of

the vehicle along the path. For this example, the uncertainty and error are expected to grow linearly, where the uncertainty grows with an expected steepness of 0.00365 metres per second and the error grows with an expected steepness of 0.0125 metres per second for the x direction and 0.0180 metres per second for the y direction (as read from Figures 4-30 and 4-34). The expected position error after travelling one leg under water at a speed of 2 metres per second thus is 3.4 metres (absolute value) in the direction of the leg (x direction) and 5.0 metres (absolute value) in the direction perpendicular to the leg (y direction). The expected value for the position uncertainty after one leg, however, is only 1.0 metres. Based on these expected error values, an estimation can be made of how well the vehicle will cover the area of the seabed. Additionally, this information can also be used to predict the accuracy of the location estimates of objects that are found during the survey of the area, which is relevant for reacquisition of these objects. Again, these performance predictions can also be used to adapt the planned path in order to improve the expected performance.

4-3 Performance processing in the ontology

In the previous section, the values that were obtained for the performance measures of two configurations under varying conditions were described. These values can be included in the developed ontology using the concepts, relationships and attributes that were described in Section 4-1. The question now is how the resulting ontology can be used to infer what tasks a vehicle can perform and how well it can do so, which was the overall goal that was set for the development and implementation of the ontology.

With the data that has been added to the ontology so far, the tasks that a vehicle can perform can be inferred through the reasoning process that was described in Chapter 3, but it is not yet possible to infer how well a vehicle can perform those tasks. The reason for this is that no performance measures for tasks have yet been added to the ontology and the corresponding performance processing instances are still missing too. In addition, first the performance data for more configurations than just the two discussed above should be added to the ontology to be able to assess task performance for all relevant tasks. For the “search task”, for example, also performance data for the configurations that realise capabilities such as “detection and classification sensing and processing” or “path and trajectory planning” is required for assessing task performance. However, the ontology data that has been added so far can be used to explain how performance assessment for tasks could be realised in the ontology, as will be discussed below.

The idea is that, when given a task, the relevant (environmental) conditions and the available vehicles with the components that they are equipped with, the ontology can be used to obtain the set of values of the performance measures for that specific task for each available vehicle. These sets of values should give an indication of which vehicle is best suited for the task under the given conditions. The inference of the values for the task performance measures should happen through automatic reasoning within the ontology. In this reasoning process, the information that is given about the conditions and the available vehicles should be used to obtain the performance measure values for the configurations of each vehicle via the corresponding configuration performance processing instances and these values should then be used to obtain the task performance measure values via the performance processing instance of the task in question. This reasoning process is illustrated in Figure 4-39.

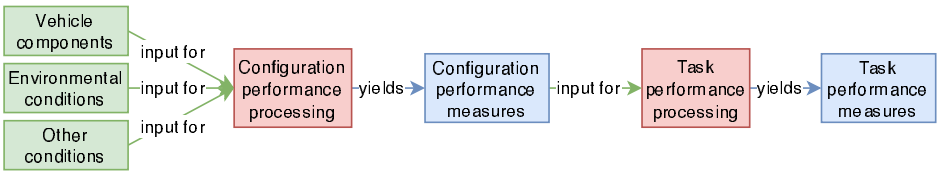


Figure 4-39: Illustration of the reasoning process for inferring task performance

The green blocks in this figure represent the conditions that are given as input to the configuration performance processing instances. These performance processing instances then output the configuration performance measures (shown in blue) for the corresponding configurations, which then serve as input for the task performance processing instance of the task in question. This task performance processing instance uses the performance measures of all configurations that are needed to execute the task for inferring the values for the task performance measures.

What is expected to be most challenging in realising this idea for performance assessment is to choose the performance measures and conditions in a such a way that the overall ontology remains comprehensible and the amount of data that is required for inferring performance measure values remains manageable. Another challenging aspect is that, ideally, it should also be possible to collect and supplement the performance data for the performance processing instances through several methods, for instance simulations, real experiments, model calculations or processing of existing data.

To give an idea of how configuration performance measures can be used to assess task performance, a small example will be given here for a task that requires the capabilities corresponding to the two configurations that were described before.
In this example, the task of the LAUV is to navigate a lawnmower pattern under water with a current flowing perpendicular to the direction of the legs, as is illustrated in Figure 4-40.

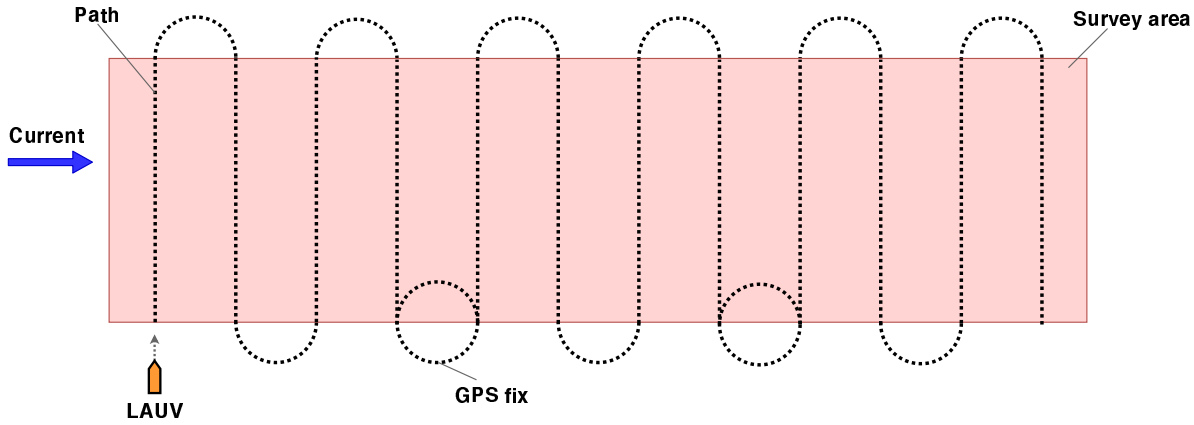


Figure 4-40: Illustration of the example use case

The lawnmower pattern consists of 12 legs of 300 metres, with turns with a radius of 20 metres in between the legs and with a GPS fix after every four legs. The speed of the current is 0.2 metres per second and the "localisation configuration" and the "propulsion and steering control configuration" are the same as they were presented previously (including DVL). The

connection between both configurations is that the estimated pose that is outputted by the "localisation configuration" will be used by the "propulsion and steering control configuration" for performing pose control. The performance measures of these two configurations can be read from the results that were presented in Section 4-2, but the question here is how the vehicle as a whole performs on the described task. A way to express the expected performance on this path following task is to predict for which parts of the legs of the planned path the vehicle will be travelling steadily and within a certain range of the path. To determine this, the values that were found for the settling distance and the mean steady state position error (performance measures of the "propulsion and steering control configuration") and the mean values found for the position error in x and y direction (performance measures of the "localisation configuration") can be used. The values for these performance measures for the situation as it was described above are shown in Table 4-7, where the x direction represents the direction that the legs are in and the y direction represents the direction that is perpendicular to the legs. The mean steady state position error (the second performance measure in the table) is also in the direction perpendicular to the legs.

Table 4-7: Configuration performance measures relevant for the path following task

Performance measure:	**Value:**
Settling distance	20 metres
Mean steady state position error	1.5 metres
Mean position error in x direction	0.6 (low) or 1.1 (high) metres
Mean position error in y direction	0.2 metres

Based on these configuration performance measure values, the conclusion can be drawn that the LAUV will need about 20 metres on every leg to reach a steady state situation and in this steady state situation, the position error is expected to be 1.1 metres in the direction of the leg and 1.7 metres perpendicular to the leg (using the highest estimates). These results give an indication of how well the vehicle is expected to perform on the path following task. If the type of performance data that was used here would have been available for multiple vehicles, it would have been possible to compare these task performance results for all vehicles in order to determine which vehicle would be best suited for the task.

Chapter 5

Conclusions

5-1 Summary

The objective that was set for this final thesis project was to develop an ontology for assessing the capabilities of AUVs that perform NMCM search operations, in order to support the planning of these operations, and to implement this ontology to show its relevance for the use case. To initiate the development of the ontology, first the use case was analysed to obtain an overview of the domain that was to be modelled. In addition, existing ontologies that are relevant for the use case were studied to learn which approaches were used by other researchers to model similar knowledge domains. In this survey of existing ontologies, the work done by an IEEE working group to develop a common ontology for the field of robotics and automation was found to be very relevant. The set of ontologies that resulted from this work, the Core Ontologies for Robotics and Automation (CORA) [33], and the Task Ontology [44] that was developed as an extension to the CORA were used as a reference for the development of an ontology in this final thesis project.

The initial study of the use case and of existing ontologies resulted in a set of main concepts that would form the basis of the ontology for the use case, namely *actor*, *task*, *capability*, *configuration* and *component*. Moreover, the goal was set for the development and implementation of the ontology to be able to automatically infer whether a vehicle with a given set of components can perform a given task and, if so, how well it can execute this task.
In short, the connections between the five main concepts that were modelled in the ontology are the following: an *actor* (i.e. vehicle for the use case considered here) is equipped with *components* and has to execute a *task*; to be able to execute this *task*, the *actor* needs a set of *capabilities*, which it can possess if it has a set of *configurations*, consisting of connected *components*, that realise the required *capabilities*.
To be able to use these general concepts and the connections between them for inferring the capabilities of an AUV that performs NMCM search operations, instances of these concepts that are specific to the use case were identified. The general concepts, the instances that were identified for them and the relationships between them together form the ontology that was developed in this final thesis project and this ontology was implemented using a tool called

Grakn. The implementation of the ontology allowed for inference of the capabilities of an AUV based on the components it is equipped with, using the reasoning methods of the Grakn software. The idea behind the corresponding reasoning process is that the components of a vehicle can be used to infer the configurations that are available on this vehicle, and these configurations are realisations of capabilities. The inferred configurations thus can be used to infer what capabilities a vehicle can have and thus what tasks a vehicle can perform, since tasks require a set of capabilities.

To realise the second part of the goal that was set for the implementation of the ontology, namely inferring how well a vehicle can execute a task, a methodology for assessing performance had to be developed and included in the ontology. After attempting a model-based approach for performance assessment, the decision was made to use a data-driven method, taking configuration performance as a starting point for assessing task performance. The underlying idea of the developed method is that the performance of a configuration or on a task can be characterised using measures of performance, and that the values of these measures vary with the conditions under which a configuration functions or a task is performed. These performance measure values for different conditions can be obtained from data, for instance from simulations or experiments, and can then be saved in a kind of look-up table together with the conditions that they were determined for. The resulting data sets can be used to predict the performance measure values under given conditions for a configuration or a task.

To include this performance assessment method in the ontology, three main conceptual elements were added, namely *condition*, *measure of performance* and *performance processing*. The *performance processing* concept represents the look-up table that was mentioned before, as well as the reasoning process that uses this table to output the performance measure values when given the prevailing conditions. For two of the *configuration* instances that were identified for the use case, the instances for the *condition*, *measure of performance* and *performance processing* concepts were defined and included in the ontology. In addition, performance data was obtained for these configurations by performing simulations in a virtual underwater environment. From the data that was recorded during these simulations, performance measure values could be found for the two configurations under different conditions. Based on these results, the conclusion was drawn that the developed performance assessment method and the corresponding ontology implementation are successful for *configuration* instances and show potential for task performance assessment as well.

5-2 Conclusions

The main conclusion of this final thesis project is that it is possible to develop and implement an ontology for inferring the capabilities of an AUV that performs NMCM search operations. Inferring a vehicle's capabilities is useful for planning these operations, since a planning system can plan tasks more optimally if it is "aware" of the capabilities of the available vehicles. Furthermore, the performance assessment methodology that was developed and incorporated in the ontology is expected to be suitable for predicting how well a vehicle can execute a task, since this method was shown to be successful for predicting configuration performance.

Another goal that was set for the development of the ontology was to keep it general enough to be applicable for other autonomous mobile robot use cases, while also making it specific

enough for the use case considered here to be able to achieve the overall goal that was set for the ontology. This desired balance between generality and specificity was accomplished by using instances of concepts to achieve the required level of detail for the use case considered here, while keeping the concepts themselves general enough to be suitable for other autonomous mobile robot use cases.

The development of the ontology was an iterative process in which it was found to be important to have a clear understanding of the objective that was to be achieved by developing and implementing the ontology. In the development process, implementation of the ontology proved to be an important aspect, on the one hand because implementing the developed ontology was found to be a valuable way to validate it and, on the other hand, because it turned out that the choice of the implementation tool has an influence on how the ontology can be used. In this final thesis project, Grakn was chosen as the implementation tool because of the possibilities it offered for querying of and reasoning with the ontology, which was required for achieving the goal that was set for the ontology.

The results that were obtained from simulations for the performance measures of two configurations showed that knowledge about configuration performance under different conditions is useful not only for predicting performance, but also for improving the expected performance on a task by adapting the planning of that task. For example, the expected performance on a path following task in an area with currents can be improved by shifting the planned path to compensate for the expected position error of the vehicle, which can be known from the performance measures of the configuration that handles pose control.

Another finding of the development of a method for performance assessment was that there are many factors that influence performance for the use case considered here and that there are many interdependencies between these factors. Also, performance can be defined and quantified in many different ways and on different levels of complexity. Therefore, the decision was made to take configuration performance as a starting point and to use a data-driven method for performance assessment instead of a theoretical one. Still, the developed method requires the definition of performance measures for characterising performance for a configuration or task, which was found to be challenging.

5-3 Future work

In this final thesis project, an ontology was developed that can be used to infer whether an AUV with a given set of components can perform a given task for an NMCM search operation. However, some extensions to this ontology are required to also be able to infer how well this vehicle can execute the given task. Most of the work that needs to be done for realising this goal consists of defining conditions and performance measures for all configurations and tasks of the use case and obtaining the corresponding performance data to find values for these performance measures under different conditions.

All data that has been added to the ontology so far is for one specific vehicle (the LAUV of TNO), so another suggestion for future work is to add concept instances and performance data for other vehicles to the ontology. This way, comparing the configuration or task performance measure values of multiple vehicles under the same conditions will be possible.

The developed ontology was implemented using the Grakn software, but it was not yet used in an actual vehicle or by an actual planning system. The ultimate goal of implementing the ontology is to make the knowledge that it contains available to a planning system, such that this system can plan the tasks of an NMCM operation more optimally. To achieve this goal, the ontology and corresponding reasoning processes need to be incorporated in the software architecture of a planning system, which will probably require the development of some form of interfacing. In addition, the planning system needs to be adapted in a such a way that the knowledge that is provided by the ontology can be used to improve the planning process.

A suggestion for improvement of the developed performance assessment methodology is to use data processing methods to gain more information from the performance data that is recorded in simulations or experiments. For instance, by finding trends in the values of performance measures over changing conditions, interpolation and extrapolation of the performance data could be used to obtain performance measure values for more different conditions than were tested in simulations or experiments. However, to be able to find trends for the performance measure values that were found so far, more simulations are needed in order to reduce the effects of outliers in the recorded data.

Taking a broader view, a promising direction for further research is to use the developed ontology not only for determining a vehicle's capabilities before an operation or for predicting performance, but also for online monitoring of a vehicle's capabilities and performance. In addition, another potential application for the ontology is online performance management. If the performance of a vehicle can be monitored online, the knowledge in the ontology about the expected performance under different conditions could be used, for instance, to adapt component or configuration settings to improve performance.

Finally, it is expected to be advantageous to integrate the developed ontology with other ontologies, such as the CORA and the Task Ontology that were mentioned before. Integration with other ontologies can lead to improvements in the developed ontology and may result in a broader applicability for other use cases. After all, the ability to assess the capabilities and performance of an autonomous vehicle can be a useful asset for improving vehicle autonomy.

Appendix A

Definitions of schema elements

In the three sections of this appendix, definitions are given for the concepts, relationships and attributes that are used in the developed ontology schema, as described in Chapters 3 and 4. These definitions are inspired by the following sources: [44, 25, 66, 67]

A-1 Concepts

This section gives definitions for the concepts used in the ontology schema. The definitions are given in alphabetical order of the concepts, where subtypes of concepts, recognisable by an indentation, are defined directly below their supertype concepts.

Actor An autonomous entity, i.e. either a person or an autonomous mobile system, that can have capabilities, can execute a mission and can execute tasks.

- **Human** A person that can have components and capabilities and can act as an autonomous entity performing tasks.
- **Vehicle** An autonomous mobile system that is equipped with components that are part of the vehicle, can have capabilities and can execute tasks.

Capability Skill or ability to perform specific actions supporting the ability to execute a task or mission. An actor can have a capability and a task can require an actor to have certain capabilities to be able to execute that task.

- **Planning capability** The ability to plan and execute a more extensive action or set of actions. A planning capability can require one or more execution capabilities.
- **Execution capability** The ability to perform an action that can directly be executed by a configuration of components and that does not require extensive planning
- **Evaluation capability** The ability to perform an evaluation of actions that have been or are being executed.

Component A physical or virtual element that can be part of a vehicle and that fulfils a certain function that is relevant for one or more configurations.

Configuration Set of components connected through input/output relations that together provide a capability. All input and functionality requirements of all components in the configuration must be fulfilled.

Constraint Limitation to how a task or mission can be executed by an actor.

Environment The surroundings in which an actor executes tasks and which affects the task and the way in which it can be executed.

Environmental condition A state of the environment that can affect the way in which a task can be executed.

Functionality The purpose a component can serve when used.

Goal The desired result or end state for a mission.

Mission Highest level operation to achieve an overall goal, which can be informational or physical or both. A mission can be executed by one or more actors and can consist of one or more tasks that have to be performed in order to fulfil the mission.

Output/Input Concept that is used to model what is exchanged via IO relations. Something that is transmitted from or to a component.

- **Capacity** A resource that a component can require to function and that can be provided by another component.
- **Data** Information produced by sensing components or software components that can be transmitted between components
- **Physical effect** A phenomenon that is part of the environment and that is caused by a (actuating) component or affects a (sensing) component such that it results in internal activity.

Performance processing Implementation of a method for assessing performance, which, when given a set of prevailing conditions, outputs the corresponding values for a set of performance measures for a configuration, task or mission.

- **Configuration performance processing** Performance processing for configurations.
- **Task performance processing** Performance processing for tasks.
- **Mission performance processing** Performance processing for missions.

Task A lower level operation that can be executed by one actor and can be part of a mission. A task can consist of one or more actions that have be executed by the actor to finish the task. Executing a task requires the executing actor to have one or more specific capabilities.

Team A set of actors that together perform a mission by individually executing tasks in a coordinated way.

A-2 Relationships

This section gives definitions for the relationships used in the ontology schema, alphabetically ordered by the relationship names. For every relationship, the schema elements that can be related through the relationship are shown in brackets, indicating the direction of the relationship with an arrow.

executes (actor;team → task; mission) Relationship that specifies that an actor (or team of actors) operates in order to execute a certain task or mission. An actor can execute only one task at a time. An actor can execute only one mission at a time, but one mission can be executed by multiple actors in a team.

has (concept → attribute) Relationship that specifies the attributes (i.e. properties) a concept has.

hasCapability (actor → capability) Relationship that specifies that an actor possesses the skill or ability to perform specific actions corresponding to a capability.

hasComponent (actor → component) Relationship that specifies that an actor (human or vehicle) possesses or is equipped with a component.

hasGoal (mission → goal) Relationship that specifies that a mission is coupled to a goal that is to be achieved by executing that mission.

hasPp (configuration;task;mission → performance processing) Relationship that specifies that a configuration, task or mission has a corresponding performance processing instance.

influences (physical effect; output/input → environment; state) Relationship that specifies that a physical effect or an output/input has a direct or indirect effect on the environment or on a component state respectively.

influencesQuality (environment → output/input) Relationship that specifies that the environment can influence some properties of an output/input.

inputFor (performance processing; component; output/input; environmental condition → performance processing) Relationship that specifies that the element at the starting end of this relationship can influence the performance for a configuration, task or mission and thus is input for the corresponding performance processing concept.

IOrelation (component → component) Relationship that specifies that some form of output/input can be transmitted between the components connected through this relationship.

isa (concept → concept) Relationship that specifies that a concept is a subtype of another concept.

isConstrainedBy (mission; task → constraint) Relationship that specifies that the way in which a mission or task can be executed is limited by a constraint.

isPartOf (actor → team, task → mission, component → configuration) Relationship that specifies the elements (actors/tasks/components) that are part of another type of element (team/mission/configuration).

isRealisedBy (capability → configuration) Relationship that specifies that a certain configuration can be used to realise a capability.

originatesFrom (environmental condition → environment) Relationship that specifies that an environmental condition represents a state or property of the environment.

provides (component; configuration → functionality; output) Relationship that specifies that a component or the combined components in a configuration can yield a functionality or output when in operation.

requires (component → functionality; input) Relationship that specifies that the presence of a certain functionality or input is needed for a component to be able to function.

requiresCapability (task; capability → capability) Relationship that specifies that a task or capability requires a capability. An actor must have this capability to be able to execute the task in question or to possess the capability in question.

requiresTask (task → task) Relationship that specifies that for a task to be performed, another task needs to be executed first.

yields (performance processing → measure of performance; quality; quantity; property) Relationship that specifies that the process represented by a performance processing concept results in an output that consists of performance measures.

A-3 Attributes

This section gives definitions for the attributes used in the ontology schema, alphabetically ordered by the attribute names. For every attribute, the schema elements that can have this attribute are shown in brackets.

Measure of performance (MOP) (configuration; task; mission) Attribute that represents a quantity that gives an indication of performance for a configuration, task or mission.

Property (output/input) Attribute of output/input that represents a characteristic of the output/input that is not necessarily in the form of a numeric value.

Quality (output/input) Attribute of output/input that gives an indication of the reliability or usefulness of the output/input in the form of a numeric value.

Quantity (output/input) Attribute of output/input that represents a characteristic of the output/input in the form of a numeric value.

State (component) Attribute of component that gives an indication of the condition the component is in.

Appendix B

Ontology data: configurations

Figures B-1, B-2, B-3, B-4 and B-5 show, for five configurations defined in the ontology data, which components can together form these configurations in the LAUV of TNO. For each configuration, the IO relations that can exist between the corresponding components are depicted together with the data that can be transmitted via these connections. The black arrows in the figures represent the IO relations and are connected to the `data` instances (shown in green) that are transmitted via these relations. The gray arrows represent input or output of data to or from a component (shown in red).

The following five configurations are shown:

- Detection and classification configuration (Figure B-1): this configuration gives a vehicle the capability to obtain data through long range imaging and to process this data for detection and classification of (mine-like) objects.
- Obstacle avoidance configuration (Figure B-2): this configuration gives a vehicle the capability to detect obstacles in front of the vehicle and to adjust the path of the vehicle in order to evade these obstacles.
- Path and trajectory planning configuration (Figure B-3): this configuration gives a vehicle the capability to plan a path of poses for transiting or for surveying an area, taking possible constraints, such as no-go areas and vehicle manoeuvring limitations, into account.
- Transiting configuration (Figure B-4): this configuration gives a vehicle the capability to plan and navigate a path to a given target location. Figure B-4 shows a simplified representation of this configuration: two subsets of connected components are shown as configuration blocks (in purple). The components and relations that constitute these blocks are shown in Figures 3-12a and 3-13a.
- Area covering configuration (Figure B-5): this configuration gives a vehicle the capability to plan and navigate a search pattern (e.g. a lawnmower pattern) over a given target area. Some subsets of the components in this configuration also constitute distinct configurations.

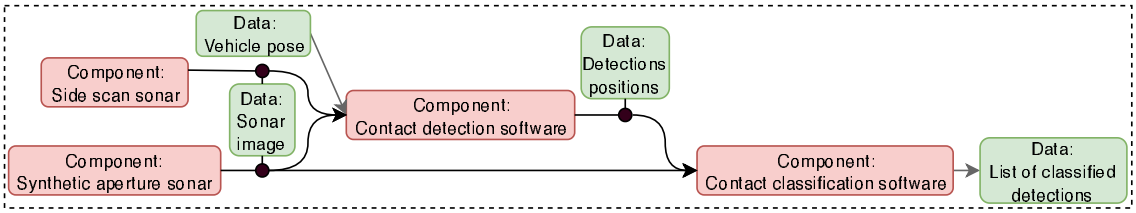


Figure B-1: Detection and classification configuration with IO relations

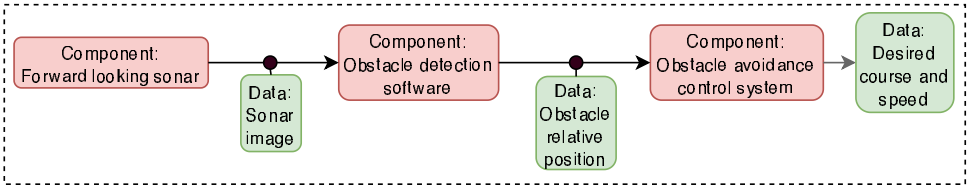


Figure B-2: Obstacle avoidance configuration with IO relations

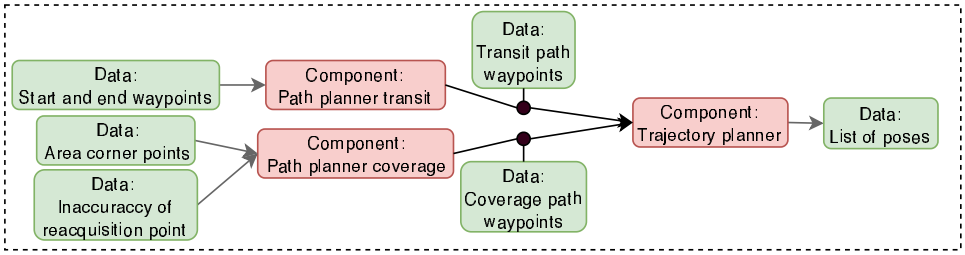


Figure B-3: Path and trajectory planning configuration with IO relations

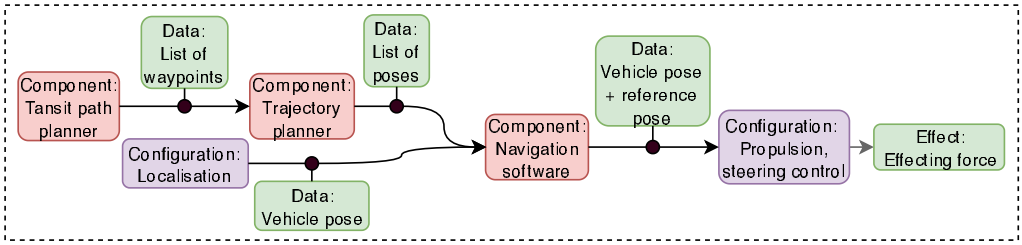


Figure B-4: Transiting configuration with IO relations

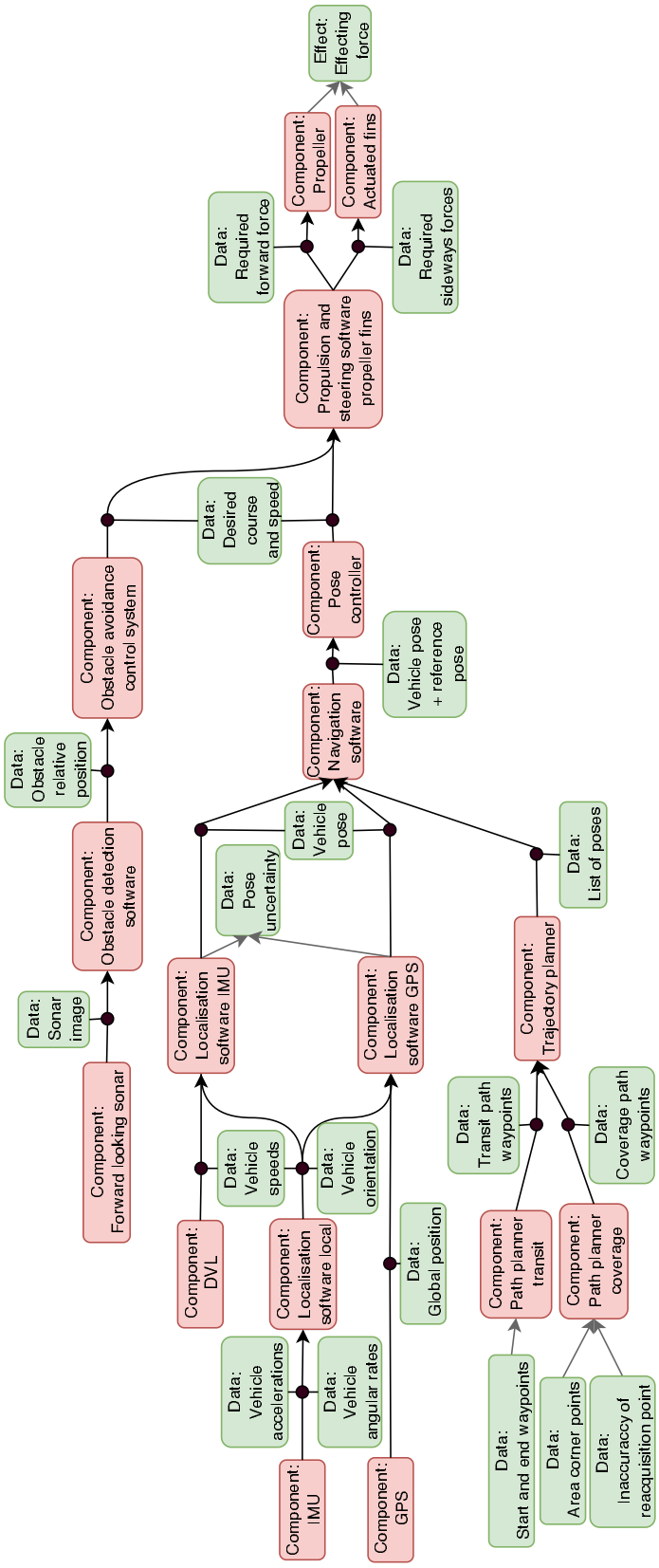


Figure B-5: Area covering configuration with IO relations

Appendix C

Grakn implementation of the ontology schema

The listing below (Listing C.1) shows the Graql code that was written to implement the developed ontology schema in Grakn. In the code, first the entities (which is what concepts are called in Grakn) are defined, followed by the attributes and the relationships (or relations, as they are called in Grakn). Comments in the code are denoted with a number sign (#). The main signal words in this code (apart from `entity`, `attribute` and `relation`) are `sub`, `has`, `plays`, `datatype`, `relates` and `as`. The meaning of these words in the code will be explained below [55]:

- The word `sub` indicates that the corresponding element is a subtype of the element following this word (for instance, `vehicle sub actor` means that `vehicle` is a subtype of `actor`).
- The word `has` indicates that the corresponding element has the attribute following this word. In Grakn, both entities and relationships and attributes can have attributes.
- The word `plays` indicates that the corresponding element can have the role shown after this word in a relationship. In Grakn, both entities and attributes and relationships can play roles in a relationship.
- The word `datatype` indicates that the values of the corresponding attribute should have the datatype shown after this word. If an attribute has the datatype `string`, then regular expressions (indicated by the word `regex`) can be used to limit the possible values of this attribute.
- The word `relates` indicates that one of the roles that can be played in the corresponding relationship is the role that follows this word. When this role is followed by the word `as` combined with another role, this means that the corresponding relationship is a subtype of another relationship and the role before the word `as` is a subtype of the role following this word.

Listing C.1: Graql code for the implementation of the developed ontology schema

```
# Ontology schema suitable for Grakn version 1.5.7
define

## ENTITIES
# capability
capability sub entity,
  has name,
  has code,
  has description,
  plays realised_capability,
  plays capability_object,
  plays requiring_capability,
  plays required_capability;

  planning_capability sub capability;
  execution_capability sub capability;
  evaluation_capability sub capability;

# configuration
configuration sub entity,
  has name,
  has code,
  has description,
  has performance_measure,
  plays realising_configuration,
  plays containing_configuration,
  plays requiring_configuration,
  plays providing_configuration,
  plays configuration_subject;

  planning_configuration sub configuration;
  execution_configuration sub configuration;
  evaluation_configuration sub configuration;

# component
component sub entity,
  has name,
  has code,
  has description,
  has component_state,
  plays requiring_component,
  plays providing_component,
  plays component_object,
  plays contained_component,
  plays in_component,
  plays component_out,
  plays table_condition;

  hardware_component sub component;
    support_component sub hardware_component,
      has support_component_type;
```

```
52      sensor_component sub hardware_component;
53        payload_sensor sub sensor_component,
54          has payload_sensor_type;
55        navigation_sensor sub sensor_component,
56          has navigation_sensor_type;
57      effector_component sub hardware_component,
58        has effector_type;
59
60    software_component sub component;
61      sensing_processing_function sub software_component,
62        has sensing_processing_function_type;
63      navigation_function sub software_component,
64        has navigation_function_type;
65      planning_control_monitoring_function sub software_component,
66        has planning_control_monitoring_function_type;
67
68  # functionality
69  functionality sub entity,
70    has name,
71    has code,
72    has description,
73    has functionality_type,
74    plays required_functionality,
75    plays provided_functionality,
76    plays functionality_object;
77
78  # input/output
79  input_output sub entity,
80    has name,
81    has code,
82    has unit,
83    has quality,
84    has quantity,
85    has property,
86    plays required_input,
87    plays provided_output,
88    plays influenced_part;
89
90    data sub input_output,
91      has data_format,
92      plays required_data,
93      plays provided_data,
94      plays transported_data;
95    capacity sub input_output,
96      has capacity_type,
97      plays required_capacity,
98      plays provided_capacity,
99      plays transported_capacity;
100   physical_effect sub input_output,
101     has data_format,
102     plays required_physical_effect,
103     plays provided_physical_effect;
104
```

```
105 # actor
106 actor sub entity,
107   has name,
108   plays actor_subject,
109   plays executing_actor,
110   plays contained_actor;
111
112   human sub actor;
113
114   vehicle sub actor,
115     has vehicle_type;
116
117 # team
118 team sub entity,
119   plays executing_team,
120   plays containing_team;
121
122 # mission
123 mission sub entity,
124   has name,
125   has code,
126   has description,
127   has performance_measure,
128   plays containing_mission,
129   plays mission_subject,
130   plays executed_mission;
131
132 # task
133 task sub entity,
134   has name,
135   has code,
136   has description,
137   has performance_measure,
138   plays requiring_task,
139   plays required_task,
140   plays executed_task,
141   plays contained_task,
142   plays task_subject;
143
144 # goal
145 goal sub entity,
146   plays goal_object;
147
148 # constraint
149 constraint sub entity,
150   plays constraint_object;
151
152 # performance processing (includes table)
153 performance_processing sub entity,
154   has name,
155   has code,
156   plays performance_processing_object,
157   plays outputting_processing_object,
```

```
158   plays table;
159
160   configuration_performance_processing sub performance_processing;
161
162 # target object
163 target_object sub entity,
164   has size,
165   has shape,
166   has material,
167   has position,
168   has target_label,
169   plays influencing_factor;
170
171 # environmental condition
172 condition sub entity,
173   has name,
174   has code,
175   has unit,
176   has value,
177   plays table_condition;
178
179   environmental_condition sub condition,
180     plays influencing_factor;
181
182     underwater_condition sub environmental_condition; # eg water_depth,
          currents, sea_state, bathymetry, sound_speed_profile,
          water_clarity
183
184     seabed_condition sub environmental_condition; # eg sediment_type,
          ripples, vegetation, has clutter, burial
185
186 ###################################################################
187 ## ATTRIBUTES
188 # capability, method, component, mission, task
189 name sub attribute, datatype string; # also for actor
190 code sub attribute, datatype string;
191 description sub attribute, datatype string;
192
193 # component
194 component_state sub attribute, datatype string,
195   has unit,
196   has value;
197
198 # vehicle
199 vehicle_type sub attribute, datatype string, regex "(LAUV|REMUS|HUGIN)";
200
201 # component (subtypes)
202 support_component_type sub attribute, datatype string, regex "(
      on_board_computer|power_supply_system|internal_communication_system|
      electric_motor)";
203 navigation_sensor_type sub attribute, datatype string, regex "(IMU|GPS|
      compass|DVL)";
```

```
204 payload_sensor_type sub attribute, datatype string, regex "(
        forward_looking_sonar|side_scan_sonar|synthetic_aperture_sonar|
        digital_camera|MBES|current_sensor|
        conductivity_temperature_pressure_sensor|turbidity_sensor|
        sound_speed_sensor|salinity_sensor)";
205 effector_type sub attribute, datatype string, regex "(propeller|
        thrusters_forward|thrusters_sideways|actuated_fins)";
206 sensing_processing_function_type sub attribute, datatype string, regex "(
        obstacle_detection_software|detection_software|classification_software
        |identification_software_camera|identification_software_MBES)";
207 navigation_function_type sub attribute, datatype string, regex "(
        localisation_software_local|localisation_software_IMU|
        localisation_software_GPS|
        propulsion_and_steering_software_propeller_fins|
        propulsion_and_steering_software_thrusters|pose_controller|
        obstacle_avoidance_control_system|trajectory_planner|
        path_planner_transit|path_planner_area_coverage|navigation_software)";
208 planning_control_monitoring_function_type sub attribute, datatype string,
         regex "(status_monitor_system|area_covering_software|
        survey_planning_system|reacquisition_planning_system|
        identification_planning_system)";
209
210 # input/output, performance
211 qqp sub attribute, datatype string,
212   has name,
213   has code,
214   has unit,
215   has data_format,
216   has value,
217   plays table_condition,
218   plays performance_output;
219
220   quality sub qqp;
221   quantity sub qqp;
222   property sub qqp;
223
224 unit sub attribute, datatype string;
225 data_format sub attribute, datatype string, regex "(scalar|vector|list)";
226 capacity_type sub attribute, datatype string, regex "(electrical_power|
        processing|memory|internal_communication|mechanical_power)";
227 value sub attribute, datatype double;
228
229 # functionality
230 functionality_type sub attribute, datatype string, regex "(
        support_functionality|sensing_functionality|effecting_functionality|
        processing_functionality|pcm_functionality)";
231
232 # requirement relation
233 essential sub attribute, datatype boolean;
234
235 # target_object
236 size sub attribute, datatype string;
237 shape sub attribute, datatype string, regex "(cilinder|cone)";
```

```
material sub attribute, datatype string;
position sub attribute, datatype string;
target_label sub attribute, datatype string, regex "(MILEC|NONMILEC|MILCO
    |NONMILCO|MINE|NOMBO)";

# performance measure
performance_measure sub attribute, datatype string,
  has name,
  has code,
  has unit,
  has data_format,
  has value,
  plays performance_output;

  configuration_performance_measure sub performance_measure;

# conditions
condition_type sub attribute, datatype string, regex "(presence|settings)
    ";

###########################################################################
## RELATIONS
# requirement_relation
requirement_relation sub relation,
  has essential,
  relates required_part,
  relates requiring_part;

  requires_capability sub requirement_relation,
      relates required_capability as required_part,
    relates requiring_task as requiring_part,
    relates requiring_capability as requiring_part;

  requires_task sub requirement_relation,
    relates required_task as required_part,
    relates requiring_task as requiring_part;

  requires_functionality sub requirement_relation,
    relates required_functionality as required_part,
    relates requiring_component as requiring_part,
    relates requiring_capability;

  requires_input sub requirement_relation,
    relates required_input as required_part,
    relates required_capacity as required_part,
    relates required_data as required_part,
    relates required_physical_effect as required_part,
    relates requiring_component as requiring_part,
    relates requiring_configuration as requiring_part;

# provide_relation
provide_relation sub relation,
  relates provided_part,
```

```
289   relates providing_part;
290
291   provides_functionality sub provide_relation,
292     relates provided_functionality as provided_part,
293     relates providing_component as providing_part,
294     relates providing_configuration as providing_part;
295
296   provides_output sub provide_relation,
297     relates provided_output as provided_part,
298     relates provided_capacity as provided_part,
299     relates provided_data as provided_part,
300     relates provided_physical_effect as provided_part,
301     relates providing_component as providing_part,
302     relates providing_configuration as providing_part;
303
304 # have relation
305 have_relation sub relation,
306   relates subject,
307   relates object;
308
309   has_capability sub have_relation,
310     relates capability_object as object,
311     relates actor_subject as subject;
312
313   has_component sub have_relation,
314     relates component_object as object,
315     relates actor_subject as subject;
316
317   has_goal sub have_relation,
318     relates mission_subject as subject,
319     relates goal_object as object;
320
321 # is realised by
322 is_realised_by sub relation,
323   relates realised_capability,
324   relates realising_configuration;
325
326 # is constrained by
327 is_constrained_by sub relation,
328   relates task_subject,
329   relates mission_subject,
330   relates constraint_object;
331
332 # executes
333 execution_relation sub relation,
334   relates executed_part,
335   relates executing_part;
336
337 executes_task sub execution_relation,
338   relates executed_task as executed_part,
339   relates executing_actor as executing_part;
340
341 executes_mission sub relation,
```

```
342   relates executed_mission as executed_part,
343   relates executing_actor as executing_part,
344   relates executing_team as executing_part;
345
346 # is part of relation
347 is_part_of sub relation,
348   relates contained_part,
349   relates containing_part;
350
351   is_part_of_mission sub is_part_of,
352     relates contained_task as contained_part,
353     relates containing_mission as containing_part;
354   is_part_of_team sub is_part_of,
355     relates contained_actor as contained_part,
356     relates containing_team as containing_part;
357   is_part_of_configuration sub is_part_of,
358     relates contained_component as contained_part,
359     relates containing_configuration as containing_part;
360
361 # influences quality
362 influences_quality sub relation,
363   relates influencing_factor,
364   relates influenced_part;
365
366 # IO relation
367 IOrelation sub relation,
368   relates component_out,
369   relates in_component,
370   relates transported_capacity,
371   relates transported_data;
372
373 # can_have_functionality
374 can_have_functionality sub relation,
375   relates actor_subject,
376   relates functionality_object,
377   relates providing_component;
378
379 # performance_relationships
380 has_performance_processing sub have_relation,
381   relates configuration_subject as subject,
382   relates task_subject as subject,
383   relates mission_subject as subject,
384   relates performance_processing_object as object;
385
386 yields_performance sub relation,
387   relates performance_output,
388   relates outputting_processing_object;
389
390 input_for_processing sub relation,
391   relates table_condition,
392   relates table,
393   has condition_type;
```

Bibliography

[1] M. L. Seto, ed., *Marine Robot Autonomy*. Springer, 2013.

[2] B. K. Sahu and B. Subudhi, "The state of art of autonomous underwater vehicles in current and future decades," in *2014 First International Conference on Automation, Control, Energy and Systems (ACES)*, pp. 1–6, Feb 2014.

[3] "Naval mine warfare vision." https://www.eguermin.org/Wordpress/wp-content/uploads/2014/08/NMW-VISION.pdf, last accessed on 28/10/18.

[4] "Naval mine counter measures." https://www.eguermin.org/welcome/naval-mine-warfare/naval-mine-counter-measures/, last accessed on 10/10/18.

[5] M. Nakjem, R. Fardal1, and B. Haugsvær, "Future norwegian unmanned maritime mine countermeasures (MMCM) concept," in *Proceedings of Undersea Defence Technology (UDT) 2017*, 05 2017.

[6] Ø. Midtgaard, I. Alm, T. Olsmo Sæbø, M. Geilhufe, and R. Hansen, "Performance assessment tool for AUV based mine hunting," in *Proceedings of the International Conference on Synthetic Aperture Sonar and Synthetic Aperture Radar 2014*, vol. 36, pp. 120–129, 09 2014.

[7] M. Cramer, J. Beach, T. Mazzuchi, and S. Sarkani, "Understanding information uncertainty within the context of a net-centric data model: A mine warfare example," *The International C2 Journal*, vol. 3, no. 1, 2009.

[8] E. Migueláñez, P. Patrón, K. Brown, Y. Petillot, and D. Lane, "Semantic knowledge-based framework to improve the situation awareness of autonomous underwater vehicles," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 5, pp. 759–773, 2011.

[9] A. Atyabi, S. MahmoudZadeh, and S. Nefti-Meziani, "Current advancements on autonomous mission planning and management systems: An AUV and UAV perspective," *Annual Reviews in Control*, 2018.

[10] E. Prestes, J. Carbonera, S. Fiorini, V. Jorge, M. Abel, R. Madhavan, A. Locoro, P. Gonçalves, M. Barreto, M. Habib, A. Chibani, S. Gérard, Y. Amirat, and C. Schlenoff, "Towards a core ontology for robotics and automation," *Robotics and Autonomous Systems*, vol. 61, pp. 1193–1204, 11 2013.

[11] F. Maurelli, M. Carreras, J. Salvi, D. Lane, K. Kyriakopoulos, G. Karras, M. Fox, D. Long, P. Kormushev, and D. Caldwell, "The PANDORA project: A success story in AUV autonomy," in *OCEANS 2016 - Shanghai*, pp. 1–8, April 2016.

[12] F. Maurelli, Z. Saigol, and D. Lane, "Cognitive knowledge representation under uncertainty for autonomous underwater vehicles," in *Proceedings of IEEE ICRA'14 IEEE Hong Kong, Workshop on Persistent Autonomy for Underwater Robotics*, 2014.

[13] C. Schlenoff, S. Foufou, and S. Balakirsky, "Performance evaluation of robotic knowledge representation (PERK)," in *PerMIS*, 2012.

[14] G. Papadimitriou and D. Lane, "Semantic based knowledge representation and adaptive mission planning for MCM missions using AUVs," in *OCEANS 2014 - TAIPEI*, pp. 1–8, April 2014.

[15] X. Li, S. Bilbao, T. Martín-Wanton, J. Bastos, and J. Rodriguez, "SWARMs ontology: A common information model for the cooperation of underwater robots," in *Sensors*, 2017.

[16] B. Siciliano and O. Khatib, eds., *Springer Handbook of Robotics*, ch. 43 Underwater Robotics. Springer-Verlag, 2007.

[17] R. Bogue, "Underwater robots: a review of technologies and applications," *Industrial Robot: the international journal of robotics research and application*, vol. 42, no. 3, pp. 186–191, 2015.

[18] P. Patrón and Y. R. Petillot, "The underwater environment: A challenge for planning," in *In Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group*, 2008.

[19] J. Bellingham, B. Kirkwood, and K. Rajan, "Tutorial on issues in underwater robotic applications," in *ICAPS 2006*, 2006.

[20] P. E. Hagen, R. Hansen, and B. Langli, "Synthetic aperture sonar for the HUGIN 1000-MR AUV," in *Proceedings of Undersea Defence Technology (UDT) 2006*, 06 2006.

[21] J. Keller, "Navy undersea warfare researchers to purchase additional REMUS 100 UUV from Hydroid Inc.." https://www.militaryaerospace.com/articles/2013/04/NUWC-REMUS-100.html, last accessed on 2/5/19.

[22] A. Sousa, L. Madureira, J. Coelho, J. Pinto, J. Pereira, J. B. Sousa, and P. Dias, "LAUV: The man-portable autonomous underwater vehicle," *IFAC Proceedings Volumes*, vol. 45, no. 5, pp. 268 – 274, 2012. 3rd IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles.

[23] "Light autonomous underwater vehicle." http://www.oceanscan-mst.com/, last accessed on 6/8/19.

[24] H. de Boer, "Experimenteren onder water." https://magazines.defensie.nl/materieelgezien/2017/01/mg201701onderwaterdrones, last accessed on 7/8/19.

[25] M. Van Riet, S. Giodini, V. Newsum, J. Van De Sande, and B. Quesson, "LAUV software design," tech. rep., TNO, Sept 2018.

[26] D.-H. Gwon, J. Kim, M. H. Kim, H. G. Park, T. Y. Kim, and A. Kim, "Development of a side scan sonar module for the underwater simulator," *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 662–665, 2017.

[27] M. B. Brissette and J. E. H. Clarke, "Side scan versus multibeam echosounder object detection: A comparative analysis," in *Proceedings of HYDRO'99 Conference*, 1999.

[28] Ø. Midtgaard and M. Nakjem, "Unmanned systems for stand-off underwater minehunting," in *Proceedings of Undersea Defence Technology (UDT) 2016*, 06 2016.

[29] L. Paull, S. Saeedi, M. Seto, and H. Li, "Sensor-driven online coverage planning for autonomous underwater vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 18, pp. 1827–1838, Dec 2013.

[30] I. Mulders and A. Van Velsen, "V1614 WP400 MOE MOP Overview v4," tech. rep., TNO, 2018.

[31] N. Guarino, D. Oberle, and S. Staab, "What is an ontology?," in *Handbook on Ontologies* (S. Staab and R. Studer, eds.), Springer, second ed., 2009.

[32] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs.," in *SEMANTiCS (Posters, Demos, SuCCESS)*, vol. 1695 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016.

[33] Ontologies for Robotics and Automation (ORA) Working Group, "IEEE standard ontologies for robotics and automation," *IEEE Std 1872-2015*, pp. 1–60, April 2015.

[34] M. Uschold, "Ontologies and database schema: What's the difference?." https://pdfs.semanticscholar.org/b44f/a4592b69183c1965d0075dea1a3bc58dfbfe.pdf, last accessed on 6/5/19.

[35] M. Tenorth and M. Beetz, "KnowRob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013.

[36] M. Tenorth and M. Beetz, "Representations for robot knowledge in the KnowRob framework," *Artificial Intelligence*, vol. 247, pp. 151 – 169, 2017. Special Issue on AI and Robotics.

[37] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," *2011 IEEE International Conference on Robotics and Automation*, pp. 5589–5595, 2011.

[38] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, and F. Maurelli, "Dynamically extending planning models using an ontology," in *Proceedings of the 2nd ICAPS Workshop on Planning and Robotics (PlanRob-15)*, pp. 79–85, 2015.

[39] R. N. Carvalho, *Probabilistic Ontology: Representation and Modeling Methodology*. PhD thesis, George Mason University, Fairfax, VA, USA, 2011. AAI3471053.

[40] R. N. Carvalho, K. B. Laskey, and P. C. Costa, "PR-OWL - a language for defining probabilistic ontologies," *International Journal of Approximate Reasoning*, vol. 91, pp. 56 – 79, 2017.

[41] G. Papadimitriou, Z. Saigol, and D. M. Lane, "Enabling fault recovery and adaptation in mine-countermeasures missions using ontologies," in *OCEANS 2015 - Genova*, pp. 1–7, May 2015.

[42] S. R. Fiorini, J. Bermejo-Alonso, P. Gonçalves, E. P. de Freitas, A. O. Alarcos, J. I. Olszewska, E. Prestes, C. Schlenoff, S. V. Ragavan, S. Redfield, B. Spencer, and H. Li, "A suite of ontologies for robotics and automation [industrial activities]," *IEEE Robotics Automation Magazine*, vol. 24, pp. 8–11, March 2017.

[43] S. R. Fiorini, J. L. Carbonera, P. Gonçalves, V. A. Jorge, V. F. Rey, T. Haidegger, M. Abel, S. A. Redfield, S. Balakirsky, V. Ragavan, H. Li, C. Schlenoff, and E. Prestes, "Extensions to the core ontology for robotics and automation," *Robot. Comput.-Integr. Manuf.*, vol. 33, pp. 3–11, June 2015.

[44] S. Balakirsky, C. Schlenoff, S. R. Fiorini, S. Redfield, M. Barreto, H. Nakawala, J. L. Carbonera, L. Soldatova, J. Bermejo-Alonso, P. J. G. Fatima Maikore, E. D. Momi, V. R. S. Kumar, and T. Haidegger, "Towards a robot task ontology standard," in *Proceedings of the ASME 2017 International Manufacturing Science and Eng. Conference*, pp. V003T04A049–V003T04A049, American Society of Mechanical Engineers, 2017.

[45] J. I. Olszewska, M. Houghtaling, P. Gonçalves, T. Haidegger, N. Fabiano, J. L. Carbonera, S. R. Fiorini, and E. Prestes, "Robotic ontological standard development life cycle," in *IEEE ICRA 2018 WELCARO workshop*, 2018.

[46] Ó. Corcho, M. Fernández-López, and A. Gómez-Pérez, "Ontological engineering: Principles, methods, tools and languages," in *Ontologies for Software Engineering and Software Technology*, 2006.

[47] W3C, "Semantic web." https://www.w3.org/standards/semanticweb/, last accessed on 29/10/18.

[48] D. L. McGuinness and F. van Harmelen, "OWL web ontology language overview," 2004.

[49] P. Hitzler, M. Krötzsch, B. Parsia, P. F Patel-Schneider, and S. Rudolph, "OWL 2 web ontology language primer (second edition)," December 2012.

[50] S. Abburu and S. B. Golla, "Ontology storage models and tools: An authentic survey," *Journal of intelligent systems*, vol. 25, no. 4, pp. 539–553, 2016.

[51] Solid IT, "Knowledge base of relational and NoSQL database management systems." https://db-engines.com/en/, last accessed on 07/11/18.

[52] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Inc., 2013.

[53] A. B. M. Moniruzzaman and S. A. Hossain, "NoSQL database: New era of databases for big data analytics - classification, characteristics and comparison," *International Journal of Database Theory and Application*, vol. 6, no. 4, 2013.

[54] S. Klarman, "Knowledge graph representation: GRAKN.AI or OWL?." https://blog.grakn.ai/knowledge-graph-representation-grakn-ai-or-owl-506065bd3f24, last accessed on 07/11/18.

[55] GRAKN.AI, "Grakn documentation portal." https://dev.grakn.ai/docs/, last accessed on 16/07/19.

[56] A. Bouchard, J. Sijs, P. Patrón, G. Davies, and T. Furfaro, "Common message schema," tech. rep., NATO SCI-288 Research Task Group (RTG), June 2018.

[57] D. Lane and Y. Petillot, "Bayesian updating to ontology - technical approach," work-package deliverable, PANDORA project, July 2015.

[58] Y. Hongfei, W. Hongjian, L. Hongli, and W. Ying, "Research on situation awareness based on ontology for UUV," *2016 IEEE International Conference on Mechatronics and Automation*, pp. 2500–2506, 2016.

[59] J. Barnhoorn, D. Bekers, H. Bouma, J. Van Diggelen, J. De Gier, P. Kerbusch, L. Kester, M. Kruithof, M. Lomme, I. Mulders, M. Peeters, K. Schutte, J. Sijs, A. Van Velsen, J. Voogd, R. Van Vossen, J. Van Der Waa, and P. Wessels, "RVO Autonomy: Landscaping study for understanding, reasoning, learning, optimization, and human-machine teaming," tech. rep., TNO, Dec 2017.

[60] M. Gruninger and M. Fox, "Methodology for the design and evaluation of ontologies," in *Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal*, 1995.

[61] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. van Deemter, and R. Stevens, "Towards competency question-driven ontology authoring," in *The Semantic Web: Trends and Challenges*, pp. 752–767, Springer International Publishing, 2014.

[62] L. Madureira, A. Sousa, J. Braga, P. Calado, P. Dias, R. Martins, J. Pinto, and J. Sousa, "The light autonomous underwater vehicle: Evolutions and networking," in *2013 MTS/IEEE OCEANS - Bergen*, pp. 1–6, June 2013.

[63] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A Gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*, IEEE, sep 2016.

[64] SciPy.org, "numpy.polyfit." https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.polyfit.html, last accessed on 30/07/19.

[65] SciPy.org, "scipy.optimize.curve_fit." https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html, last accessed on 31/07/19.

[66] W. Van Vught, "Modelling the predictability of task execution in a Human-Agent-Robot Team," Master's thesis, Vrije Universiteit Amsterdam, 2017.

[67] Oxford University Press, "Oxford English Dictionary." https://www.lexico.com/en, last accessed on 05/07/19.

Glossary

List of Acronyms

AI	Artificial Intelligence
AUV	Autonomous Underwater Vehicle
CORA	Core Ontologies for Robotics and Automation
CQ	Competency Question
CSV	Comma-Separated Values
DVL	Doppler Velocity Log
EOD	Explosive Ordnance Disposal
FLS	Forward Looking Sonar
GPS	Global Positioning System
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IO	Input/Output
LAUV	Light Autonomous Underwater Vehicle
MBES	Multi-Beam Echo Sounder
MCM	Mine CounterMeasures

MOP Measure of Performance

NMCM Naval Mine CounterMeasures

ORA WG Ontologies for Robotics and Automation Working Group

OWL Web Ontology Language

PANDORA Persistent Autonomy Through Learning Adaptation Observation and Re-planning

PID Proportional Integral Derivative

ROS Robot Operating System

RoSaDev Robot Standard Ontological Development Life Cycle

ROV Remotely Operated Vehicle

SA Situation Awareness

SSS Side-scan Sonar

SUMO Suggested Upper Merged Ontology

SWARMs Smart and Networking Underwater Robots in Cooperation Meshes

TNO the Netherlands Organisation for applied scientific research

UUV Unmanned Underwater Vehicle

VHF Very High Frequency

W3C World Wide Web Consortium

XML eXtensible Markup Language