PDDL-Based Task Planning of Survey Missions for Autonomous Underwater Vehicles

A generic planning system, taking into account location uncertainty and environmental properties

Lukas Steenstra

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

PDDL-Based Task Planning of Survey Missions for Autonomous Underwater Vehicles

A generic planning system, taking into account location uncertainty and environmental properties

Master of Science Thesis

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

Lukas Steenstra

September 13, 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

PDDL-Based Task Planning of Survey Missions for Autonomous Underwater Vehicles

by

Lukas Steenstra

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: September 13, 2019

Supervisor(s):

dr. ir. J. Sijs

prof. dr. ir. B. De Schutter

Reader(s):

dr. N. Yorke-Smith

dr. A. A. Nunez Vicencio

R. M. Hommes

Abstract

Autonomous Underwater Vehicles (AUVs) are unmanned vehicles that give the opportunity to carry out lengthy and dangerous tasks autonomously. This is particularly useful for survey tasks, where the objective is to search the seafloor for objects. In this thesis work a planning system is developed that can plan a path for survey tasks, while considering environmental challenges such as communication limitations and location uncertainty. To compensate for location uncertainty, the planning system requires a higher level of abstraction compared to conventional path planning algorithms. For that reason, the planning problem is modelled in the Planning Domain Definition Language (PDDL), creating a powerful and flexible planning system which deals with the complex survey problem. Besides that, some additional planners are added to support the PDDL-planner and provide suitable plans for the AUV to carry out.

The resulting plans are evaluated by simulation, showing that the planning system can successfully survey different scenarios. Besides that, the PDDL model is validated by means of the Event-B formal method, in order to obtain mathematical proofs of the validity of the planning model. The results are a step forward in achieving full autonomy of the AUVs. Besides that, a demonstration of the applicability of PDDL in real-world problems is given.

Table of Contents

List of Figures

List of Tables

Acknowledgements

During the writing of this work, I have received a lot of support and assistance. First of all, I would like to thank my supervisor dr. ir. J. Sijs, who played an essential role in the process of this thesis work. Without his advice and encouragement, I would not have been able to make this work such a success. Besides that, he provided me with the opportunity to work at TNO. I would like to thank prof. dr. María D. R-Moreno as well, for her expertise and advice.

Furthermore, I would like to thank my colleagues at TNO for their collaboration. They were always interested in my topic and willing to help me whenever possible. It has been a pleasure working together with them.

Last but not least, I especially want to thank my friends and family who supported and encouraged me while working on this project.

Delft, University of Technology
September 13, 2019

Lukas Steenstra

Chapter 1

Introduction

1-1 Background

1-1-1 Autonomous Underwater Vehicles

An Autonomous Underwater Vehicle (AUV) is an unmanned vehicle which can operate underwater autonomously. These vehicles come in a variety of sizes and capabilities, ranging from more than 10 metres long to small portable vehicles (see Figure 1-1), each being capable of fulfilling different tasks. The common purpose of these vehicles is to achieve persistent autonomy, which is the ability to operate in complex and changing environments without human intervention [1].

Figure 1-1: OceanScan-MST's LAUV [2]

Persistent autonomy is useful for lengthy and costly operations in underwater environments and might even provide opportunities that would otherwise not be possible without the use of AUVs. Typical applications are bathymetry [3, 4], underwater inspections [5], marine geo-science [6] and fish tracking [7].

AUVs give the opportunity to carry out dangerous tasks as well, since these vehicles are unmanned and thus do not involve any risks for human operators. For this reason, the AUV is specifically interesting for Mine Countermeasure (MCM) operations, which are intended to ensure the safety of vessels passing through an area. Such an MCM operation is generally divided into four phases [8]:

i. **Detection** - First, the seafloor is scanned using sonar in order to detect objects, also called contacts. The purpose of this phase is to cover the entire area.

ii. **Classification** - Then, these contacts are classified from the resulting images, determining whether they are mine-like contacts or not.

iii. **Identification** - The mine-like contacts are revisited to identify whether they are actually a mine and what kind of mine it is (also called reacquisition).

iv. **Disposal** - Optionally, the identified mines can be neutralised by bringing a disposal charge next to the mine.

The scope of this thesis work is the *detection* phase, where the objective is to survey a designated area such that all potentially harmful objects in that area will be detected. In this thesis work, the *detection* phase is further referred to as the *survey task*. A task planning system needs to be developed to control the AUV, such that this *survey task* can be performed autonomously in a reliable way.

1-1-2 The Survey Task

Whereas the goal of an MCM operation is to minimize the risk of navigating through a certain area, the sub-goal of the survey task is to cover parts of the area and provide the probability that all possible mines are detected. This probability depends on the environmental properties, the hardware that is used and the quality of the planning system. In Chapter 3-1 the definition of this coverage probability is explained in more detail.

The Light Autonomous Underwater Vehicle (LAUV), which is the vehicle used by TNO, is equipped with a Side-Scan Sonar (SSS) system. This sonar system looks straight in both port and starboard direction (Figure 1-2). While moving forward, these lines of sonar-data can be merged into an image as shown in Figure 1-3. From these images, objects can be detected and classified as either being mine-like or non-mine-like, i.e., the *classification* phase.

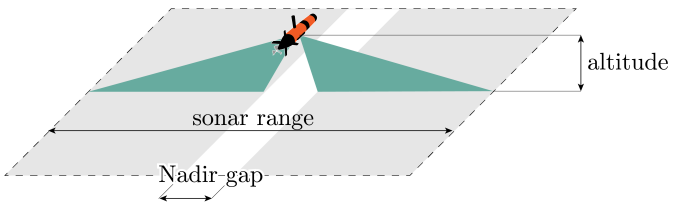


Figure 1-2: This figure shows the geometry of the SSS system. It looks sideways straight in both port and starboard direction. Due to the angle of the sonar transducers, there is a gap underneath the vehicle, which is called the Nadir gap.

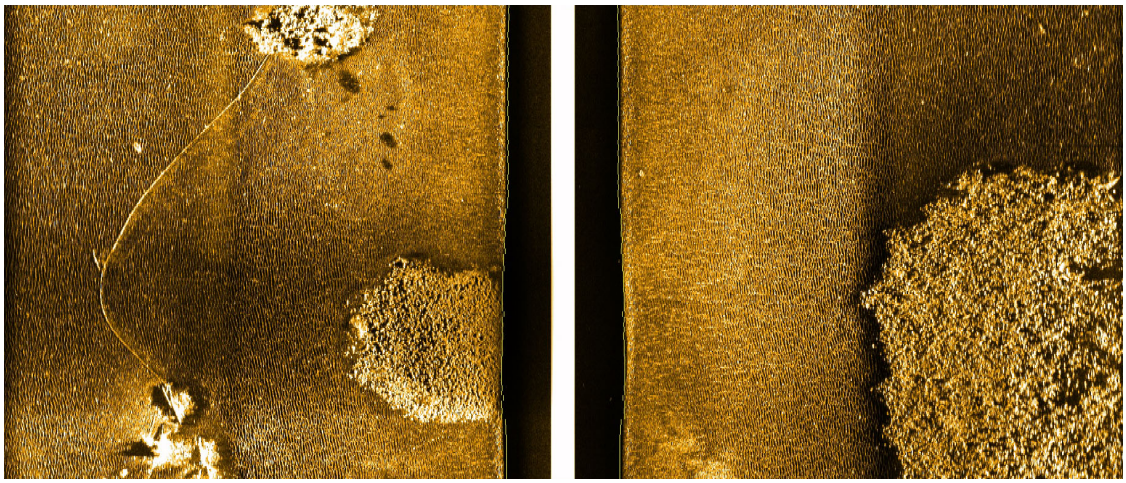


Figure 1-3: An example of an SSS image [9]

To cover a certain area, the AUV generally navigates in a lawnmower pattern over the area (also called *boustrophedon* pattern in literature [10]), such that it covers the entire area with its SSS, as shown in Figure 1-4. The reason to navigate in a lawnmower pattern is that, in order to get qualitative sonar images, the vehicle preferably needs to sail in straight lines. The task-planner is responsible for planning this coverage path.

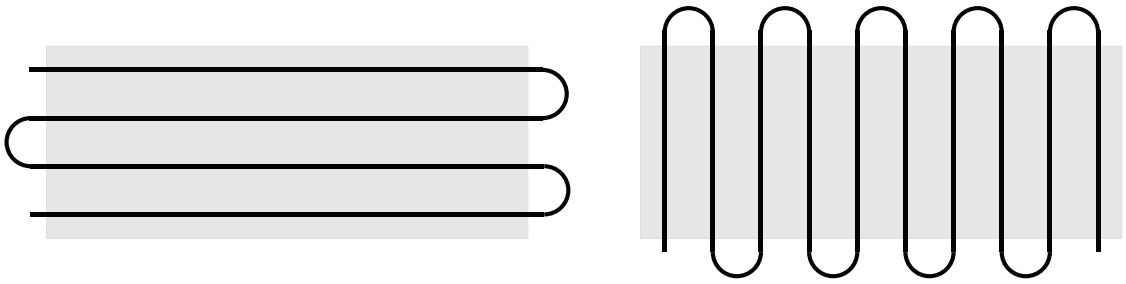
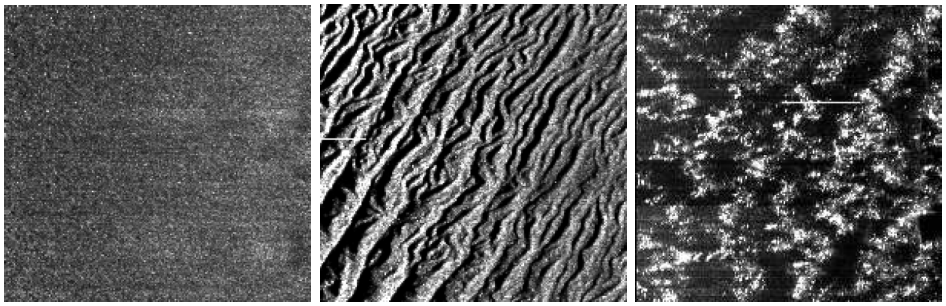


Figure 1-4: To cover a rectangular area, AUVs generally navigate in a lawnmower pattern.

Planning a coverage path is different from conventional path planning, as the focus is not on a destination but rather on the path itself, which is often referred to as *coverage planning*. At first sight, this might seem to be a straightforward planning problem. However, since the AUVs operate in an underwater environment, some major challenges for navigation, perception and communication arise, requiring a higher level of autonomy to successfully perform survey tasks. The most important challenges are as follows:

i. **Underwater currents** - Underwater currents strongly influence the motion of the vehicle. When the current is co-linear with the vehicle, it only influences its velocity. However, when the direction of the currents is perpendicular to the AUV's trajectory, *crabbing* will occur, i.e., the heading of the vehicle is not equal to the direction in which the vehicle is moving, which results in a sideways movement. When this is the case, the effective range of the sonar is decreased.

ii. **Seafloor structure** - The seafloor can have different structures, significantly influencing the sonar performance. Figure 1-5 shows three types of seafloor structures. In the case of ripples, the direction of the AUV influences the amount of *'shadow'* present in the sonar images.

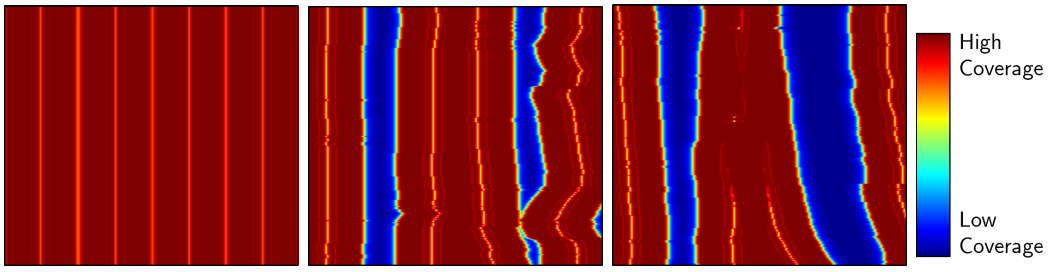


(a) Flat sediment **(b)** Ripples **(c)** Complex

Figure 1-5: Sample sonar images that show three different seafloor types respectively: flat sediment such as sand or mud (a), sand ripples (b) and more complex structures such as vegetation or stone clutter (c) [11].

iii. **Location uncertainty** - Electromagnetic waves are heavily damped underwater, due to the high conductivity of seawater [12]. This means that GPS is not available underwater and hence, the vehicle has no access to its absolute position. For that reason, the AUV navigates using an Inertial Navigation System (INS), which uses an Inertial Measurement Unit (IMU) and gyroscope to calculate its location, orientation and velocity. Its location is estimated by integrating the acceleration measurements of the IMU, which can be noisy and have bias. Consequently, integration drift will occur, which leads to location uncertainty.

Figure 1-6 [13] shows an example where an AUV needs to follow a simple lawnmower pattern. Due to location uncertainty, the AUV might leave unacceptably large gaps uncovered, resulting in low coverage percentages. The increase of location uncertainty can be predicted using uncertainty models. The coverage planner can use this prediction to adjust the distance of the lawnmower legs, accounting for location uncertainty. Figure 1-7 shows a typical uncertainty curve of an uncertainty model for lawnmower patterns.



(a) Desired **(b)** Estimated **(c)** Actual

Figure 1-6: Discrepancy between (a) the desired coverage, (b) the estimated coverage and (c) the actual coverage of an AUV following a lawnmower pattern. This is caused by environmental disturbances and location uncertainty [13].

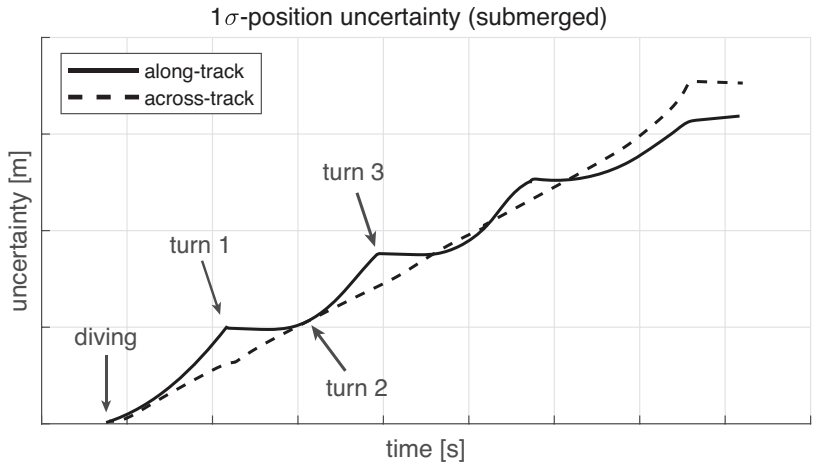


Figure 1-7: This figure shows a typical uncertainty curve of an AUV navigating in a lawnmower pattern. The solid line represents the *along-track* (forward) position uncertainty of the AUV and the dashed line represents the *across-track* (sidewards) position uncertainty. When the AUV makes a turn, the *along-track* position uncertainty is slightly reduced.

iv. **Limited communication** - Due to the high conductivity of seawater, communication between underwater vehicles is difficult. For most applications above water electromagnetic signals are used for communication, which is not an option for AUVs. Acoustic communication is the obvious choice for underwater, but these signals are often noisy, lossy and low-bandwidth in comparison to electromagnetic signals [14]. In addition, the propagation of acoustic waves underwater is dependent on the prevailing conditions, such as temperature, salinity and ambient noise. Besides that, no official communication protocols are developed for AUVs yet, since this is accompanied by a lot of dynamical constraints, making it a complex problem. Therefore, with the deployment of multiple AUVs, communication is a challenge on its own.

The quality of the task planning system can therefore be defined by the number of challenges it can cope with. In Chapter 3 the planning problem is described in more detail. The planning problem is modelled such that the planning system in particular can account for location uncertainty and limited communication.

1-2 Previous Work

For a few decades, this problem has already been researched. Several approaches were made to plan coverage paths for AUVs in survey missions. *Williams et al.* developed a coverage path planner that plans the optimal track spacing of the coverage path, taking into account the properties of the seafloor structure [15]. As function of the range, each seafloor type corresponds to a certain probability of detection, on which the optimal distance between the tracks is based, while maintaining full coverage of the area. Another work of *Williams et al.* takes into account the underwater currents, and adjusts the track direction appropriately to compensate for that [16].

In [17] a genetic algorithm based planner was developed, to perform both the *detection* and *identification* task. The vehicle surfaces between the designated survey areas to compensate for location uncertainty, but does not plan to do that during the survey task itself.

An approach to compensate for location uncertainty while doing coverage planning has been made as well by *Paull et al.*[13]. The probability of detection is used (based on the seafloor properties) and is combined with location uncertainty, to get a probability that the area is covered. In [18] this coverage probability was used to plan optimal locations to navigate to the surface and get a GPS-update, significantly reducing the location uncertainty during the survey task. More recently, in [19] an online dynamic programming-based coverage planning algorithm was proposed, which was able to reduce the operation time even more.

The common factor of all these coverage planners is the usage of domain specific planners to solve the survey problem, i.e., planners that are developed for one specific planning domain. This works well in terms of performance, but does not provide the flexibility that is required for persistent autonomy. To achieve this, the survey problem needs to be solved using a more flexible approach so that it can adapt to different situations and operations in a more convenient way. For this, a more generic definition of the survey problem is required. To the author's knowledge, this has not yet been done for MCM operations.

An interesting research project is PANDORA (which stands for Persistent Autonomy through Learning Adaptation, Observing and Re-planning), where AUVs need to carry out an inspection task, navigate and perform tasks such as turning valves and taking pictures [20]. To achieve this, the generic Planning Domain Definition Language (PDDL) was used to solve these complex tasks. This involved high-level planning, but did not make decisions about more low-level parameters such as location uncertainty.

Although used for a different application, *Muñoz et al.* combined the higher level task planning with the lower level path planning to control mobile robots to perform tasks at waypoints [21]. They proposed a planning system which integrates both an abstract PDDL-planner and a more domain specific path planner. Since the PDDL-planner has a broader view of the tasks that need to be done, this leads to more efficient plans.

1-3 Problem Statement

Research Objectives
The main objective of this thesis is to *develop a PDDL-based task planning system for survey tasks of AUVs, taking location uncertainty and limited communication into account, while minimizing operation time.* To achieve this, the following sub-objectives need to be achieved:

i. Construct a representative simulation environment in order to assess survey plans, and to analyse the dynamical behaviour of AUVs.

ii. Model the survey task in PDDL and select a suited PDDL-solver to solve these problems.

iii. Construct a planning system that integrates the PDDL-based task planner and generates suitable actions for the AUV to carry out.

iv. Evaluate the quality of the plans and validate the PDDL model of the planning system.

Research Questions
Based on the results of the developed planning system, the main research questions are:

i. How can the complex survey problem be modelled into PDDL, and what assumptions need to be made to achieve this?

ii. What are the benefits of using a more generic planning system compared to a domain specific planner?

iii. How can the PDDL model be validated, to ensure that it is correct and brings up valid plans in every possible scenario?

1-4 Thesis outline

This thesis is structured based on the research objectives stated in the previous section. Chapter 2 gives an overview of the simulation environment, discussing the capabilities and models of the simulation environment. Then, the software architecture of the LAUV, as used by TNO, is discussed and an explanation is given how this connects to the simulation environment.

In Chapter 3 a more in-depth description of the planning problem is given. Besides that, the overall structure of the developed planning system is explained as well as its planning procedure. In Chapter 4 it is discussed how the survey problem can be modelled in PDDL, and which PDDL-solver is suited to solve a survey problem. The sub-planners of the planning system are described in detail in Chapter 5, to give a full understanding on how the planning procedure of the planning system works.

Finally, the planning system is evaluated through simulation and the PDDL model is validated in Chapter 6. In Chapter 7, conclusions are made on the results and some recommendations for future research are given.

Chapter 2

Simulation

2-1 Simulation Environment

2-1-1 Advantages of Simulation

To evaluate the task planning system empirically, simulation is a convenient way to validate that the generated plans work. To do so, it was decided to use a realistic physics simulator that would simulate the dynamical behaviour of the AUV and its sensors as realistically as possible. Although the drawback is that it takes more effort to realise such an accurate simulation environment, it has several advantages:

i. A more abstract simulation environment is generally based on a lot of assumptions, restricting the freedom of the simulator to reach states that were not expected. Unexpected realistic behaviour might be overlooked when it is not modelled, i.e., the closer the simulation is to reality, the more confidence one can have in the simulated results.

ii. Having a realistic simulation environment provides dozens of applications. For example, the simulation of each individual sensor (e.g., a sonar system) gives opportunity to generate and analyse useful data before testing in a real world environment. TNO is already using the simulation environment in several projects, and they assisted with the development.

iii. It is worth to mention that a realistic underwater environment has already been developed by *Manhães et al.*, including hydrodynamic models and a model of the AUV dynamics [22]. This simulation environment was used for this thesis project, although it needed some important additions in order to be usable. This is discussed in next subsections.

To make optimal use of the simulation environment, it is necessary to understand the software architecture of the real LAUV platform. If the simulation environment is similar to the real world, it is possible to directly test the already developed software of the LAUV, by replacing the real world with the simulation environment. In Section 2-2, the implementation of the simulation environment with the LAUV software is described.

2-1-2 UUV Simulator

As part of the EU-funded project SWARMs, *Manhães et al.* developed an underwater environment called *UUV Simulator* [22], where UUV stands for Unmanned Underwater Vehicle. It is based on the Robot Operating System (ROS), which nowadays is a widely used framework in the field of robotics. ROS provides a communication framework between different modules, i.e., it connects sensors and actuators of a robotic system. In addition, it is compatible with the open-source simulation platform Gazebo. Gazebo provides multiple physics engines to simulate rigid-body dynamics. The *UUV Simulator* provides the following additional plug-ins to Gazebo:

- **Underwater world** - ROS/Gazebo is generally used for above-ground applications, and hence does not include an underwater environment. The *UUV Simulator* provides an underwater world with different scenarios and seafloors. It includes models of the hydrodynamic and hydrostatic forces on rigid bodies. These computations are based on Fossen's equations of motion [23]. Figure 2-1 shows an image of what this underwater world looks like.

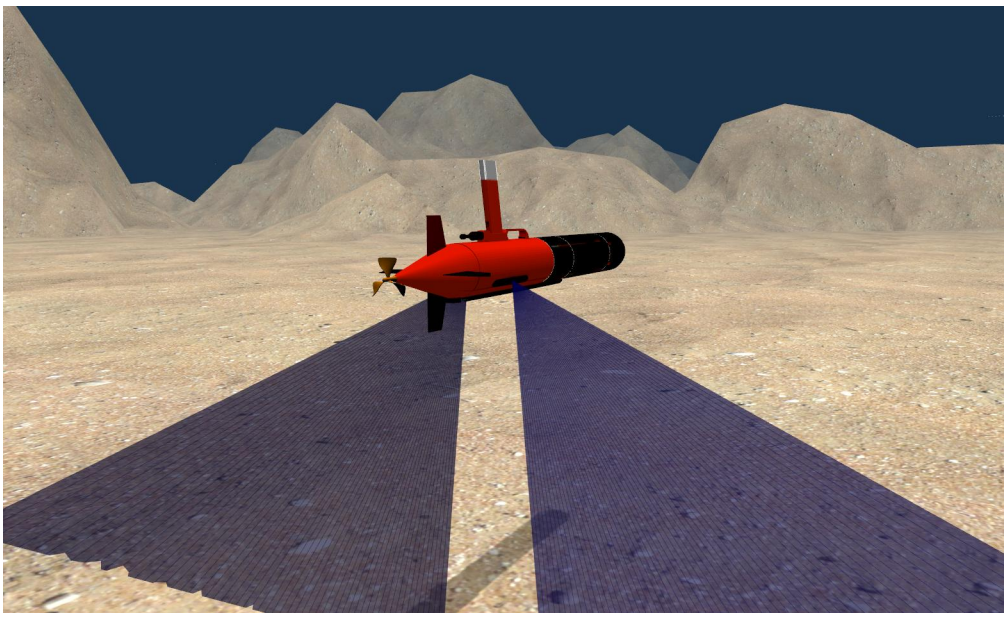


Figure 2-1: This figure shows a screen capture of the Gazebo simulation environment with the underwater world and a model of the LAUV. The blue rays are a visualisation of the sonar signals and display the geometry of the SSS system.

- **Actuators and sensors** - To simulate the interaction between the vehicles and the underwater world, several plug-ins are made for the actuators and sensors. Actuator plug-ins provide thruster dynamics for propulsion and fin dynamics for steering. Several sensors are modelled as well, including an IMU, magnetometer, Doppler Velocity Logger (DVL), pressure sensor, GPS sensor and a camera. A proper sonar plug-in was not available, and uses only a LiDAR as replacement for a sonar system. The LiDAR does not measure the target intensity in contrast to real sonar systems. A more realistic sonar plug-in is currently being developed by TNO.

- **UUV models** - As is visible in Figure 2-1, the *UUV Simulator* includes UUV models as well, with their geometric and inertial properties. There is also a model of the LAUV available, which is especially interesting for this project.

- **Controllers** - Lastly, the simulation environment provides ROS packages that are capable of controlling UUVs between waypoints, or by means of tele-operation. Although these controllers work well for vehicles like the RexROV, it had quite some issues with under-actuated vehicles similar to the LAUV. This certainly needed improvement.

The *UUV Simulator* provides a powerful basis for simulating survey tasks. However, as stated before, it still lacks the necessary functionality. First of all, a suitable controller was needed to be implemented that largely resembles the real behaviour of the LAUV. Besides that, the provided controllers were *'cheating'*, as they used the real position of the vehicle for control, instead of doing pose estimation based on sensor data.

2-1-3 Motion Control

Inertial Navigation System

The real LAUV is equipped with an INS, which uses the measurements of the IMU and other sensors to estimate the position of the vehicle. To simulate this correctly, an INS needs to be implemented that uses the simulated sensor data. This is comparable to the real INS. The following sensors are simulated in the *UUV Simulator*:

- **IMU** - The IMU measures the accelerations in all linear directions as well as the orientation of the vehicle. Note that the accelerations are measured in body frame and not in world frame.

- **Pressure sensor** - The pressure sensor measures the pressure of the water column on top of the vehicle, which can be used to derive its depth.

- **GPS** - The GPS sensor provides an absolute position of the vehicle in geographical coordinates. The simulation environment permitted the GPS signals to be measured underwater as well. This has been adjusted, such that it can only take a measurement when the vehicle is at least between 5 to 10 seconds at the water surface.

- **DVL** - The DVL measures the velocities in all linear directions with respect to the seafloor. These velocities are measured in body frame as well.

All these measurements can be fused into a single pose estimation, using a Kalman filter. How this is done is described in full detail in Appendix A. Figure 2-2 shows a screen capture of the simulation environment where this pose estimation is implemented. The Kalman filter gives a measure of the location uncertainty as well, which is valuable information for control and planning.

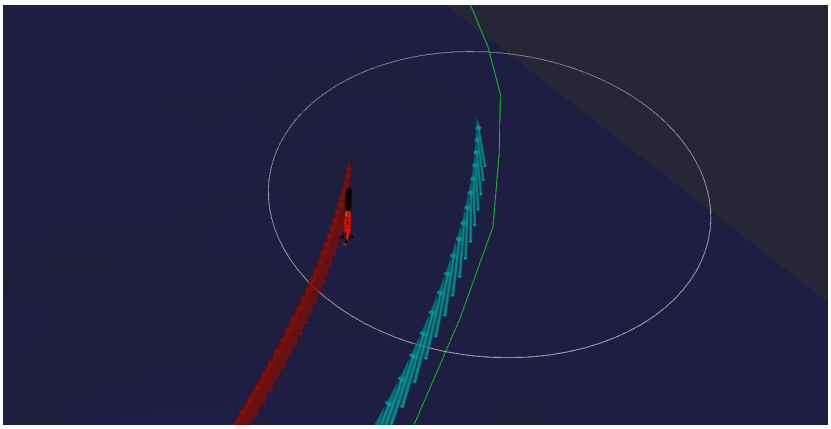


Figure 2-2: This figure shows a screen capture of RViz (a visualisation tool of ROS). The blue arrows represent the estimated poses, whereas the red arrows represent the real poses of the LAUV. The white ring is the 1-σ location uncertainty of its estimated pose. The green line is the trajectory that the vehicle needs to follow. This is now controlled using the estimated pose.

Trajectory Tracking

The *UUV Simulator* provides a trajectory tracking algorithm, which works well for UUVs with multiple actuators. However, for a non-holonomic, under-actuated vehicle like the LAUV, this trajectory tracking algorithm turned out to be insufficient. The algorithm defines a reference marker that moves with a constant velocity over the trajectory. This is easy to implement, but for the LAUV this resulted in undesired behaviour, e.g., when the LAUV passes the reference point, it wants to reverse, resulting in strange manoeuvres which causes the vehicle to lose track.

To cope with that, a simple follow-the-carrot algorithm is implemented, as shown in Figure 2-3. The algorithm first projects the LAUV position on the trajectory. Then, based on a pre-defined look-ahead distance, a reference point on the track is defined. A PID-controller tries to minimize the distance between the LAUV and this reference point, resulting in the vehicle following the trajectory.

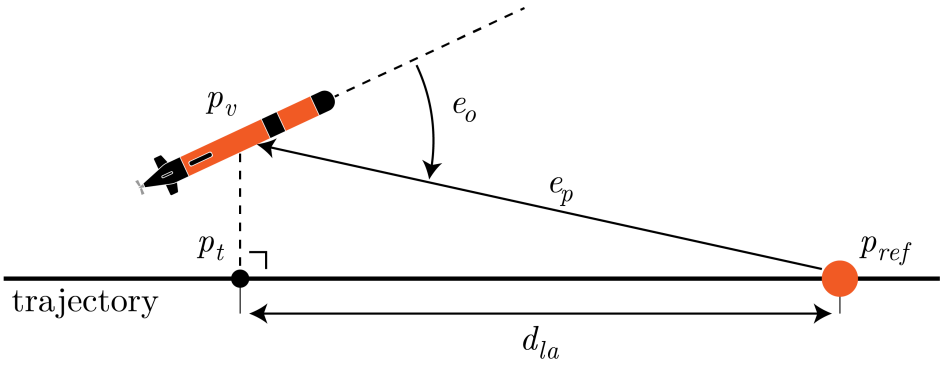


Figure 2-3: This figure shows the follow-the-carrot algorithm, which is a basic trajectory tracking algorithm. The position of the vehicle p_v is projected on the trajectory in the point p_t. Based on the look-ahead-distance d_{la} a reference point p_{ref} on the trajectory is defined. The controller minimizes the position and orientation error (e_p and e_o respectively), as shown in the figure.

Velocity Control

Another issue with the *UUV Simulator* controller is that it was not able to control its velocity. The PID-values were too aggressive resulting in the LAUV to navigate at full speed, regardless of the reference velocity. Reducing the gains was not an option as this made the LAUV to fail following the trajectory properly. Therefore, a cascaded PID solution is implemented, where the inner loop controls the position and orientation of the vehicle and the outer loop controls the velocity of the vehicle.

Controlling the velocity of the vehicle is important, since the real LAUV navigates at different speeds for different tasks. For example, when surveying a certain area, a specific velocity is needed to get proper sonar images. Transiting between waypoints can generally be done at higher velocities. For planning, it is also important to know the time it takes to execute a certain plan. If the velocities do not correspond with the plan, then the execution time of the simulation does not correspond to reality.

2-2 Software Architecture

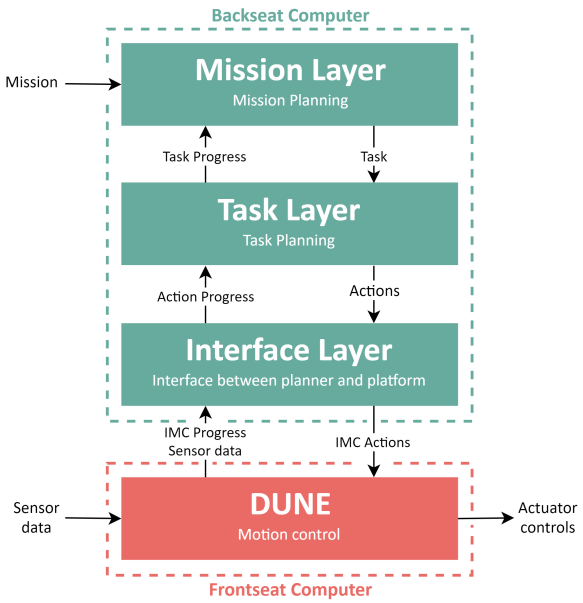
2-2-1 The LAUV Software Architecture

The LAUV is equipped with two on-board computers, a *frontseat* and a *backseat* computer, as shown in Figure 2-4a. The frontseat computer runs the DUNE (DUNE Uniform Navigation Environment) middleware, which is already installed on the LAUV by the manufacturer. This software is responsible for the low-level motion control, obstacle avoidance and controlling the actuators [24]. It uses a general message protocol called Inter-Module Communications (IMC), which therefore is the necessary message protocol to communicate with the frontseat computer. DUNE also reads the sensor data, and sends it to the interface layer using the IMC message protocol.

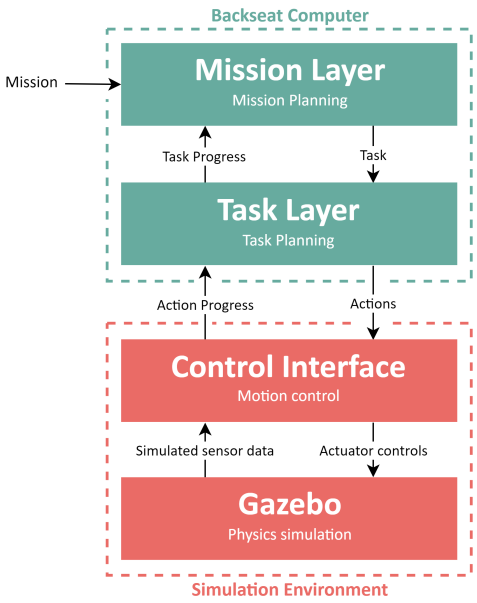
The backseat computer runs the software developed by TNO, and contains a hierarchical structure of planners. The LAUV receives a mission from an operator, which for this use case would be an MCM mission for a specific area. This will be processed by the different layers of the software as follows:

- **Mission layer** - The mission layer generates a mission plan (for one or more vehicles) and assigns tasks to the different vehicles.
- **Task layer** - The task layer receives a task and plans actions in order to execute this task in an effective way. The proposed planning system of this thesis will be located in this layer. The generated actions are low-level actions such as *go to*, *follow path* or *hold position*.
- **Interface layer** - The interface layer translates these actions of the task layer to the required IMC messages, in a way that the frontseat computer understands the actions. The interface layer is also responsible for processing the sensor data it receives.

Each layer returns its progress percentage, such that the layer above can monitor what is going on, and is able to send new tasks or actions at the right moment.



(a) The original software architecture



(b) DUNE and the interface layer are replaced for the simulation environment

Figure 2-4: The diagrams above show the hierarchical software architecture, as it is implemented on the real LAUV (a) and how the simulation environment replaces the real world (b). The mission layer and task layer are responsible for mission planning and task planning respectively. The interface layer translates the actions from the task layer into messages that can be interpreted by the platform software (DUNE), which is responsible for the low-level control. With the simulation environment, both the interface layer and DUNE are replaced for a control interface and the physics simulator Gazebo.

2-2-2 Integration of the Simulation Environment

The backseat computer of the LAUV is based on ROS as well. This makes it evidently easy to connect the simulation environment to the backseat computer. As can be seen in Figure 2-4b, the interface layer and DUNE are replaced by a control interface and the Gazebo simulation environment. The Gazebo simulator replaces the real-world environment and contains the LAUV model with its sensors and actuators. It returns simulated sensor data, which can be used for control. The control interface has the following tasks:

- Estimating the vehicle pose using the INS as described in Section 2-1-3
- Controlling the vehicle with a PID dynamic position controller, using the proposed trajectory tracking method
- Translating the actions of the task layer to actuator controls

Chapters 3, 4 and 5 will focus on the task layer, where the planning system is located. In Chapter 6, the plans of the planning system will be simulated using this simulation environment, in order to give measure to the quality of the plans.

Chapter 3

Planning System

3-1 Problem Definition

As stated in Chapter 1-3, the purpose of this thesis work is to develop a planning system for survey task of AUVs. In Chapter 1-1 the challenges that are present in the underwater environment are described. This section discusses the survey problem in more detail, giving a concise list of requirements for the planning system, in order to take these environmental challenges into account.

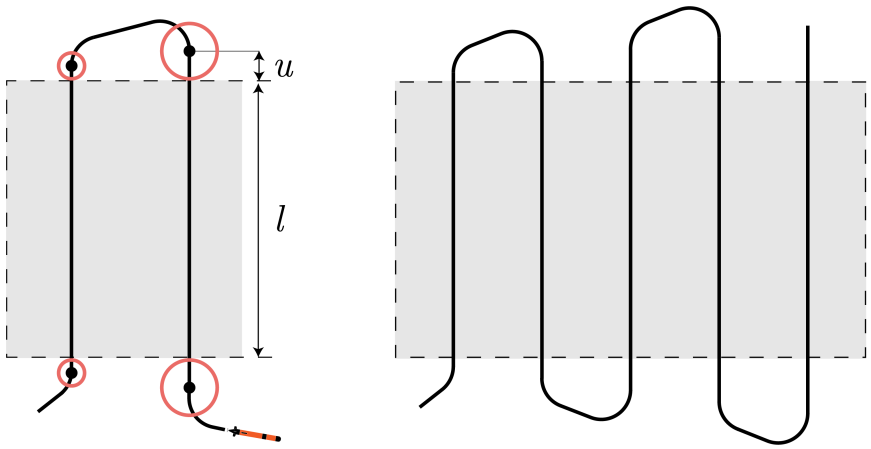
3-1-1 Problem Description

Without considering the challenges described in Chapter 1-1, the survey task is a rather simple problem, which can effortlessly be solved by a path planning algorithm. However, taking into account the growing location uncertainty and the limited communication bandwidth, the planner not only needs to plan a path, but needs to plan higher level actions as well to support its core task of surveying the area.

Location Uncertainty

Due to the integration of the bias and noise on the IMU measurements, location uncertainty grows unbounded. When the AUV is uncertain of its location, then this uncertainty is reflected in its coverage as well (as was visible in Figure 1-6). Two directions of uncertainty can be distinguished:

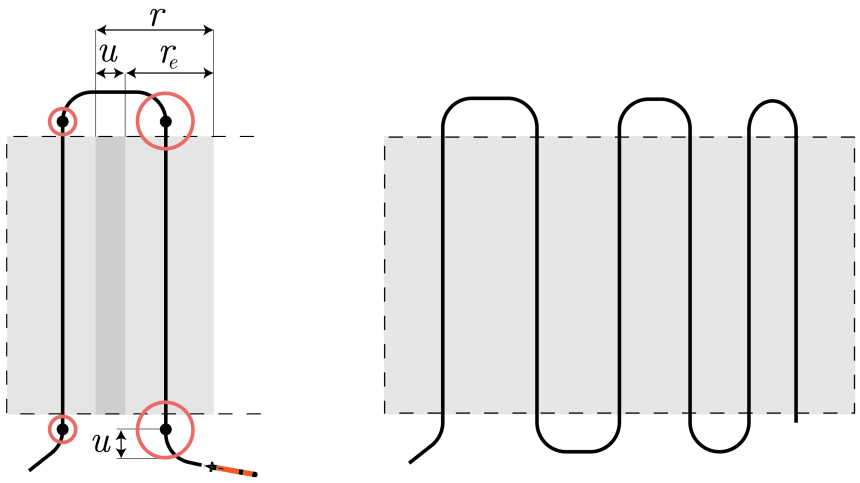
i. **Along-track uncertainty** - Along-track uncertainty is the uncertainty of the vehicle's position collinear with the lawnmower legs. To be sure that the entire track is covered, these legs are elongated. In Figure 3-1 it is illustrated how this along-track uncertainty is compensated.



(a) Uncertainty compensation **(b)** Resulting lawnmower pattern

Figure 3-1: Compensation for along-track uncertainty u by elongating the survey legs. The red rings represent the location uncertainty at that waypoint. The resulting leg length will be $l + 2u$.

ii. **Across-track uncertainty** - Across-track uncertainty is the uncertainty of the vehicle's location perpendicular to the lawnmower legs. This means that, in reality, the track can be located more to the left or more to the right. Laying tracks closer to each other than the sonar range would strictly require for complete coverage, ensures that the entire area is still covered. This might result in a few more lawnmower legs in order to cover the entire area. This is illustrated in Figure 3-2.



(a) Uncertainty compensation **(b)** Resulting lawnmower pattern

Figure 3-2: Compensation for across-track uncertainty u by reducing de distance between the survey legs. The red rings represent the location uncertainty at that waypoint. The effective sonar range r_e is defined as $r_e = r - u$. Laying the tracks next to each other, using the effective sonar range, will compensate for the across-track uncertainty

To which extent these uncertainties are compensated depends on the confidence interval of the location uncertainty. Assuming that the location uncertainty is normally distributed, a 3-σ confidence interval is used for the location uncertainty. This corresponds to a confidence of 99.7% that the AUV is located within the uncertainty ring.

Using these methods to account for uncertainty clearly is at the cost of operation time, especially when additional lawnmower legs are required. Hence, it is valuable to find solutions to reduce this uncertainty during the execution of the survey task. In this thesis, two uncertainty reducing actions are considered:

- **GPS-fix** - Since the AUV has a GPS on board, it has the possibility to navigate to the water surface and receive its absolute position based on the GPS information. This action will further be referred to as a GPS-fix and is equal to resetting the location uncertainty to zero, except for some small uncertainty caused by the GPS sensor itself.
- **Revisit** - Simultaneous Localization and Mapping (SLAM) is the process of mapping features of the environment that it recognizes, simultaneously localizing itself with respect to this map. The seafloor generally is not feature-rich, but contacts can be used as features. When these features are revisited, the vehicle can localize itself with respect to the previous time these features were found. This way, the vehicle can significantly reduce its location uncertainty.

  One important constraint is that these contacts need to be revisited from the same direction the contacts were previously seen. This is due to the fact that objects can look significantly different when viewed from different angles by sonar.

Planning these type of actions require a more high-level task planning algorithm. For that reason, the more generic planning language PDDL is used to describe these actions. The way this is implemented is described in Chapter 4.

Limited Communication

During a survey mission, the AUV will generate data. This data will contain information about its task, its odometry, the bathymetry of the area and information of the detected contacts. When operating with multiple vehicles, it is certainly important to plan actions to communicate with other vehicles or with a surface vehicle.

When communication with other vehicles is required, this needs to be done underwater. Since the bandwidth underwater already is exceptionally low, it is not guaranteed that communication is possible anywhere. There are frequently so-called black zones where no communication is possible at all. Therefore, two specific actions of communicating is considered in this thesis:

- **Communication areas** - Communication areas can be defined where the vehicle is close enough to a surface vehicle (or other vehicles) to be able to communicate. Since these areas are underwater, the transfer speeds are low. Therefore, the vehicle needs to stay in that area as long as necessary to transfer all data. This means that the duration of such an action is dependent on the amount of accumulated data.

- **Surface** - When the vehicle is at the surface, it can just make use of normal wireless communication methods. This way of communicating can be assumed to be instantaneous, as this goes at a much higher transfer rate. Communicating at the surface can therefore nicely be combined with a GPS-fix, making it a useful action to combine communicating with reducing uncertainty. However, this is only possible when communicating with a surface vessel.

A third communication method, is to simply communicate while surveying. In practice, this is done most of the times, although with limited communication bandwidth. Hence, the above two actions are specifically for the case that the distance between the vehicle and the receiver is too big, or the vehicle is in a black zone. Communicating while surveying can possibly be modelled by reducing data accumulation per survey leg. This communication method is therefore not explicitly modelled.

3-1-2 Requirements

Summarizing the background given in Chapter 1 and the problem description of the previous subsection, the planner is required to be able to achieve the following:

- Cover the designated area, by planning survey tracks and transits between the tracks
- Reduce the operation time by planning uncertainty reducing actions such as a GPS-fix or a revisit
- Communicate all information to a surface vehicle (or other vehicles), by planning communication actions

Coverage

To be able to asses whether the above requirements are fulfilled, an exact definition of coverage is required. Coverage is a direct consequence of the sonar performance, which is highly dependent on the environmental properties such as water temperature, salinity and seafloor structure. A Sonar Performance Model (SPM) can determine the Probability of Detection (POD) based on these environmental properties. Figure 3-3 shows a typical POD-curve $p(r)$, decreasing proportional to the distance r from the AUV. Close to the vehicle the POD decreases as well, due to the Nadir gap (which was also illustrated in Figure 1-2).

Coverage can be defined in several ways, either by integrating the POD of the sonar system over the entire area, or by computing the percentage of the survey area that has a POD above a certain threshold. In this thesis work, the second approach is used. A threshold k is introduced, such that the area is covered if:

$$p(r) \geq k \tag{3-1}$$

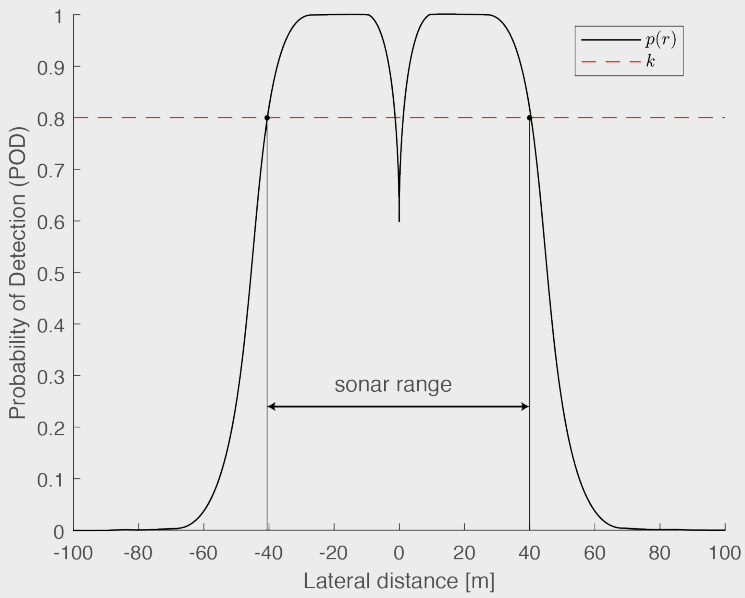


Figure 3-3: Typical POD-curve $p(r)$ as function of the lateral distance r. By applying a threshold k the sonar range can be determined as shown in this figure. Notice that with this simplification the Nadir gap is neglected.

In Figure 3-3 it is illustrated how the sonar range can be extracted from the above inequality. All the area that is within this sonar range is considered to be *fully covered*, and all the area beyond this range is not covered. In Chapter 6 this definition is used to evaluate the coverage of simulated plans.

It is visible that the effect of the Nadir gap is neglected. Moreover, the sonar range is assumed to be constant for the entire area. However, these POD-curves are highly dependent on the environmental properties and can differ significantly during a survey mission. In Chapter 7 a solution is suggested to cope with a variable sonar performance.

Table 3-1 (next page) summarizes what input is given to the planning system and what output is desired from the planning system. Note that the list of inputs is not entirely complete, since some extra parameters can be given to configure the planning system. These parameters will be discussed in Chapter 5.

Table 3-1: Inputs and outputs of the planning system

Input		
Domain	The domain describes all possible actions which the planner can plan for the vehicle, including the preconditions and effects of these actions.	
Problem	*Survey Area*	The survey area is a polygon which needs to be covered by the vehicle
	Coverage	A sonar range is required to determine the coverage of the vehicle. To determine this range, a probability threshold for the POD-curve needs to be provided.
	Location of contacts	Known locations of contacts are useful to do a revisit action for reducing location uncertainty. Several parameters are necessary to successfully plan revisits: (i) the location of the contact, (ii) the uncertainty of its location and (iii) the direction from which the contact was seen.
	Communication areas	Communication areas are polygon-shaped areas where communication can be done. Each area requires a parameter describing the transfer rate that can be achieved at that area.
States	*Vehicle states*	Both the (initial) estimated position and location uncertainty are needed to initialize the planning problem.
	Sonar states	The sonar states describe how well the sonar system can image the seafloor. This, together with the environmental states, is important for the SPM to determine the sonar range.
	Environmental states	The environmental states are important for the planning system, such that it takes into account their consequences on sonar performance. Important states are (i) current speed, (ii) current direction and (iii) seafloor structure.
Output		
Plan	As output, a plan is desired that contains survey actions, uncertainty reducing actions and communication actions that can be translated to low-level actions such as *'go to'*, *'follow path'*, *'hold position'* and *'(de)activate sensor'*	

3-2 Unified Planning System

In order to meet the requirements of Table 3-1, a planning system is needed that encloses the core task planner. This planning system is located in the task layer, as explained in Chapter 2-2. The planning system needs to generate a plan that can be received by the interface layer or controller interface, from a mission given by the mission layer. In this thesis, the task layer is decoupled from the mission layer and custom missions with the input as described in Table 3-1 are given.

For clarity, three definitions are distinguished: *planning system*, *planner* and *solver*. The *planning system* is the overall system which takes care of the entire planning process from beginning to end. The planning system contains several *planners*, each responsible for their own sub-problem. The *solver* is the specific algorithm that is being used by the *planner*, to solve the problem. This section discusses the overall structure of the *planning system* and how it works. The details of the *planners* inside this planning system are given in Chapters 4 and 5.

3-2-1 Structure

Coverage planning, using a generic planning language such as PDDL is a novel approach. Generally PDDL is used for abstract planning problems where details of the problem are omitted. PDDL is a powerful tool to plan abstract actions in an efficient way. However, when omitting details one might decrease the quality of the plan. It would be useful to combine the power of PDDL for making the abstract decisions, while considering the details of a problem.

As an example, *Munoz et al.* developed a planning system called UP2TA (Unified Path-Planning and Task-Planning Architecture), which used the details of a path planner to improve the heuristics of the PDDL task planner, resulting in high quality plans [21]. For decision making in a survey task, location uncertainty plays a prominent role and therefore cannot be omitted. Hence, a planning system is required which allows the task planner to efficiently plan the abstract actions as described in Section 3-1, while taking into account detailed information about location uncertainty and communication constraints.

For this reason, it is decided to split up the planning system into three parts: (i) a *Task Planner* which is the core PDDL-planner of the planning system, (ii) a *Problem Generator* and (iii) a *Path Planner*. The structure of the planning system is shown in Figure 3-4. Splitting up the planning system provides the following benefits:

- PDDL is limited regarding numerical expressions. Therefore, complex equations are avoided in PDDL. The Problem Generator can provide the Task Planner with numerical values, such that these computations are excluded from the planning domain (as defined in Table 3-1).
- On the other hand, a PDDL-based planner is powerful for planning abstract actions, such as GPS-fixes or communication actions. However, planning the entire problem only in PDDL would give an explosion of the search space, as this is a complex combinatorial problem. Even a simple transit problem between waypoints would cause an unreasonable computation time [21]. Consequently, planning the entire problem with a

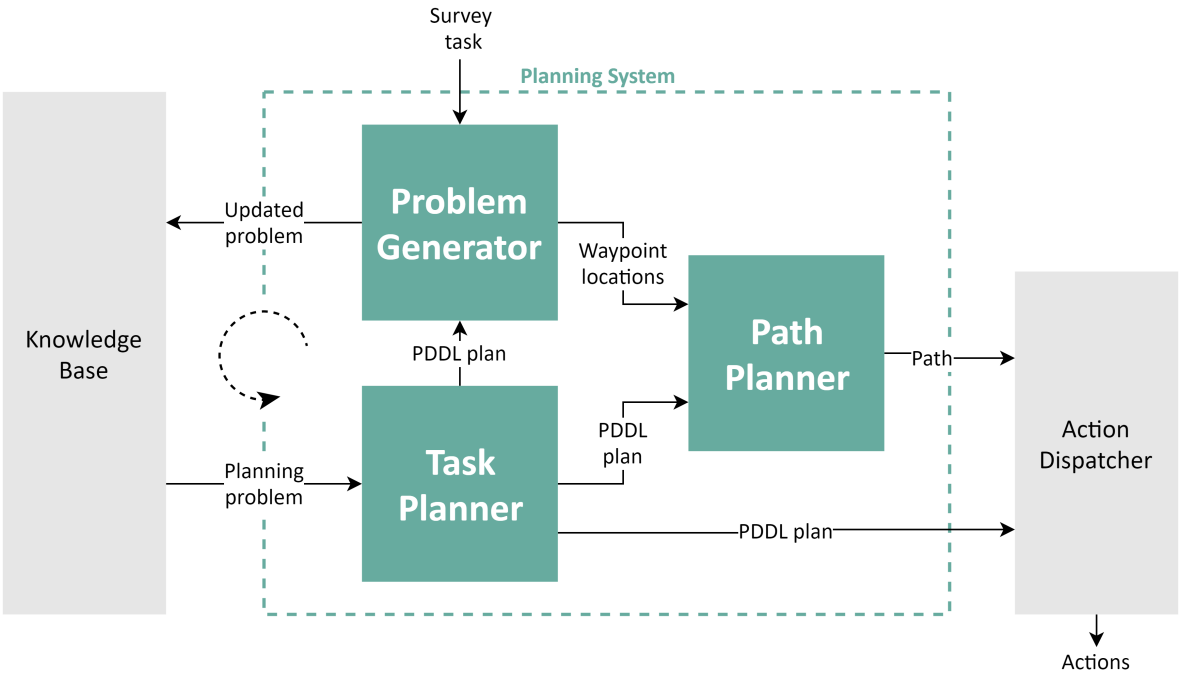


Figure 3-4: This figure shows the structure of the unified planning system. It consists of three parts: a Task Planner, a Problem Generator and a Path Planner. The core planner is the Task Planner, which solves the main task planning problem using a PDDL-solver. It retrieves its information from a Knowledge Base, which stores all the information of the planning problem and is updated by the Problem Generator. The circular arrow indicates that there is some form of iteration between the Knowledge Base, Problem Generator and Task Planner. The action dispatcher translates the final PDDL-plan and path, as planned by the Path Planner, to low-level actions that can be interpreted by the interface layer.

PDDL-based planner only, would be highly inefficient. Splitting up the planning problem such that different planners solve a part of the problem where it is specifically good at, would not only improve efficiency but the quality of the plans as well.

- Another benefit of splitting up the planning system is that it creates a modular system that can be adjusted or improved in a convenient way. This makes it possible to change planners or models without effect the way the planning system behaves, e.g., the location uncertainty model or SPM can effortlessly be replaced.

Thus it is sensible to split up the planning system, such that each module solves a part of the survey problem where it is specifically good at. The three modules, that are shown in Figure 3-4, are able to share information with the other modules, making it a unified planning system. These modules are responsible for the following tasks:

- **Task Planner** - The Task Planner is the core planner of the planning system, containing the PDDL-solver. How the problem is modelled using PDDL is explained in detail in Chapter 4. The entire planning system is based on ROS, so that it connects well with the rest of the software architecture. Therefore, the planning system integrates a ROS-based PDDL planning framework, called ROSPlan. ROSPlan provides all necessary

interaction between a PDDL-solver and ROS [25]. The Knowledge Base, which is part of ROSPlan, stores all planning information which can be used by the PDDL-solver. Inside the Task Planner, the PDDL-solver will be called using different ROSPlan interfaces. The resulting plan is then translated to ROS-messages so that it can be shared with the other modules of the planning system.

- **Problem Generator** - The Problem Generator translates the incoming survey task to a PDDL-problem. It mainly generates a set of waypoints (endpoints of survey legs) and computes the distances, durations and uncertainty growths between these waypoints. It also provides extra planning information, e.g., where and when certain actions are allowed or not. This is explained in detail in Chapter 5-1. The information is stored in the Knowledge Base, such that it can be interpreted by the Task Planner.
- **Path Planner** - The Path Planner is the final step in the planning procedure. It combines the PDDL plan from the Task Planner (describing the order of the waypoints to visit) with the waypoint locations from the Problem Generator to plan a path between those waypoints. The Path Planner is discussed in more detail in Chapter 5-2.

The final plan of the Task Planner and corresponding path of the Path Planner is given to an Action Dispatcher, which basically is the interface between the planning system and the control layer below. The abstract actions of the Task Planner are translated to actions such as *'go to'* and *'follow path'*. These low-level actions are understood by the real vehicle as well as the simulation environment.

3-2-2 Planning Procedure

The way the planners cooperate is shown in Algorithm 3.1 and Figure 3-5. The Problem Generator starts with generating an initial problem, based on the survey task. This problem includes the location of the waypoints, revisit areas and communication areas, as well as the durations and uncertainty growths for travelling between the waypoints.

Algorithm 3.1 Planning procedure of the Planning System (PS), consisting of a Problem Generator (PG), Task Planner (TP) and Path Planner (PP)

```
1: PG: Generate initial problem
2: TP: Find initial plan, and store it as best_plan
3: while best_plan has new GPS-fix or revisit do
4:     PG: Update waypoint locations
5:     PG: Generate sub-problem from GPS-fix or revisit
6:     TP: Plan sub-problem
7:     PS: Optimize the plan and store it as best_plan
8: PG: Update waypoint locations
9: PP: Plan the path according to best_plan
10: return path and best_plan
```

This problem is then planned by the Task Planner, finding a PDDL-plan containing a sequence of actions solving the problem. These actions describe the order of the tracks to be covered,

as well as when to perform actions such as a GPS-fix. When the Task Planner plans a GPS-fix at a certain point, the location uncertainty is reduced. Having less uncertainty, the distance between the survey legs can be increased (as was shown in Figure 3-2). Consequently, the waypoint locations after the GPS-fix are no longer accurate. The same will happen when a revisit is planned, or when the order of the legs is changed. In order to properly connect the Problem Generator with the Task Planner, some form of iteration is needed to update the waypoint locations, as soon as an uncertainty reducing action has been planned.

For that reason, the planning system will start its iteration loop and checks whether the plan contains an uncertainty reducing action, as soon as a plan is found by the Task Planner. When this is the case, all waypoint locations are updated and a new sub-problem will be generated by the Problem Generator, starting from the point that the first uncertainty reducing action has taken place. The Task Planner solves this sub-problem and merges it with the initial plan, which then is the new plan of the planning system.

Important to notice is that, due the numerical limitations of PDDL, the Task Planner needs to use a linearised model of the uncertainty growth. This results in a mismatch between the Problem Generator and Task Planner regarding waypoint locations and distances, making the plan less accurate. For that reason, the planning system will optimize the plan by trying to change the location of a GPS-fix slightly and compare the results. In Chapter 5-3 it is discussed how this is done.

The optimized plan is checked again whether it contains any new uncertainty reducing actions. This process is repeated until no new uncertainty reducing action is planned. As soon as this is the case, all waypoint locations are updated once again, such that the Path Planner can plan a path through these waypoints. The path, together with the last plan is sent to the Action Dispatcher, which ensures that it will be executed by the AUV.

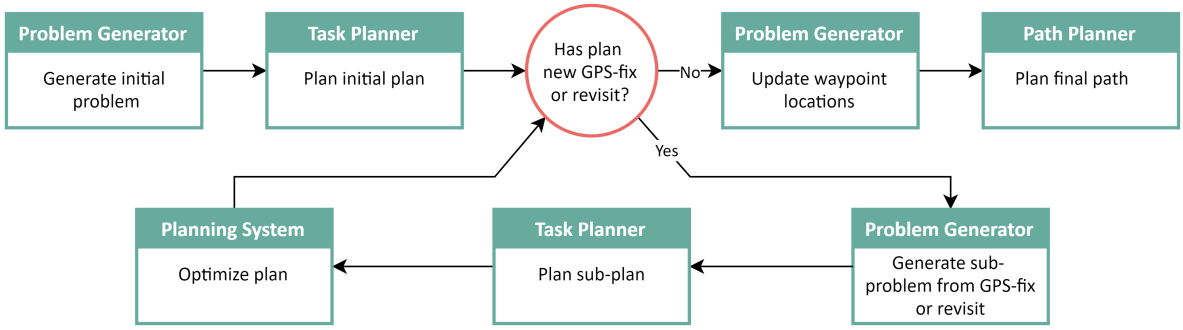


Figure 3-5: Flowchart of the planning procedure

Chapter 4 will describe in detail how the survey problem is modelled and how the Task Planner finds a plan. Then, in Chapter 5 a description of both the Problem Generator and the Path Planner is given.

Chapter 4

Task Planning

In Chapter 3-2 it was stated that the Task Planner is the core of the planning system, which is explained in more detail in this chapter. The reason for splitting up the planning system into multiple planners will become more evident. Section 4-1 will give a brief explanation of the Planning Domain Definition Language (PDDL). Section 4-2 discusses how the survey task is modelled in PDDL. Then, in Section 4-3 it is explained how PDDL-solvers apply heuristic search to find a valid plan. Finally, a selection of solvers is discussed and compared.

4-1 Problem Definition Domain Language

In the previous decades many planning algorithms have been developed, where most of them are domain specific planners. The intention of PDDL is to be a *neutral* specification of planning problems, which means that it does not favour any particular planning approach [26]. For that reason, the slogan of *Drew McDermott*, who is the originator of PDDL, is: *'physics, not advice'*. This indicates that the language should focus on expressing the physical properties of the world rather than advising the planner on how to search for solutions. Modelling the survey task in this language enables the use of multiple PDDL planners for solving the same problem.

A PDDL description always consists of two parts: a problem and a domain. The domain describes the *'physics'* of the planning domain, by defining the planning concepts (*types* in PDDL) and the possible actions that can be planned. Then a problem is defined, which the planner needs to solve based on the available actions in the domain. Listing 4.1 shows an example of a simple PDDL domain file, describing how an AUV can transit between predefined waypoints.

The domain file starts with defining the domain name and some requirements for the PDDL-solver, which for the survey problem is the functionality of `strips`, `typing` and `fluents`. The requirement `strips` comprises the basic PDDL functionality, `typing` allows the planner to define *types* and `fluents` allows the use of numeric values for planning.

Listing 4.1: PDDL example domain file - AUV transit

```
(define (domain auv_transit)
  (:requirements :strips :typing :fluents)
  (:types waypoint)

  (:predicates
    (auv_at ?wp - waypoint)              ; The AUV is at that waypoint
    (is_connected ?from ?to - waypoint) ; Two waypoints are connected
  )

  (:functions
    (total_time)               ; The total time consumed for transiting
    (time ?from ?to - waypoint) ; The duration of one specific transit
  )

  ; Transit from waypoint ?from to waypoint ?to
  (:action transit
    :parameters (?from ?to - waypoint)
    :precondition (and
        (auv_at ?from)    ; The AUV should be at the first waypoint
        (is_connected ?from ?to)  ; A transit needs to be possible
      )
    :effect (and
      ; Change AUV position to the new waypoint
      (not (auv_at ?from) )
      (auv_at ?to)
      ; Increase the total time by the transit duration
      (increase (total_time) (time ?from ?to) )
    )
  )
)
```

The planning states are represented by predicates and functions, which are boolean and numeric expressions respectively. The predicate `auv_at` is true when the AUV is at the waypoint `?wp`. A question mark indicates a parameter of the predicate. The function `total_time` does not need a parameter, as it is just a global variable, representing the total time consumed while transiting.

The remainder of the planning domain file describes the possible actions. This domain only contains one action called `transit` and requires two waypoints as parameter. Then, the preconditions and effects are defined of the action. When the preconditions are true, this action can be planned, and the defined effects will take place.

The problem file describes the actual problem that needs to be solved by the PDDL-solver. An example problem is shown in Listing 4.2. The problem refers to the PDDL `domain` of Listing 4.1, and defines some `objects`. Then all predicates and functions are initialized, being the initial state of the problem. The goal is to reach `wp3` while minimizing the total transit time, represented by the goal and metric respectively.

Listing 4.2: PDDL example problem file - AUV transit

```
(define (problem auv_transit_p0)
  (:domain auv_transit)
  (:objects wp0 wp1 wp2 wp3 - waypoint)

  (:init
      ; Initially the AUV is at waypoint 0
      (auv_at wp0)
      ; Initialize total transit time
      (= (total_time) 0)
      ; Define (one-way) connections between waypoints
      (is_connected wp0 wp1) (is_connected wp0 wp2)
      (is_connected wp1 wp3) (is_connected wp2 wp3)
      ; Initialize the durations for transiting between waypoints
      (= (time wp0 wp1) 2) (= (time wp0 wp2) 1) (= (time wp1 wp3) 2)
      (= (time wp2 wp3) 1)
  )

  (:goal (and
      (auv_at wp3) ; Finish at waypoint 3
    )
  )

  (:metric minimize (total_time)) ; Minimize the transit time
)
```

The PDDL-solver tries to find a valid plan based on the provided problem and domain file. The quality of this plan, or whether a plan is found at all, is highly dependent on the PDDL-solver that is being used. OPTIC, a solver that is used for this thesis, was able to find the optimal solution to this trivial planning problem:

Listing 4.3: Planner output

```
;;;; Solution Found
; States evaluated: 6
; Cost: 2.000
; Time 0.00
0.000: (transit wp0 wp2) [0.001]
0.001: (transit wp2 wp3) [0.001]
```

4-2 Modelling the Survey Domain

This section describes how the domain of the survey task is modelled in PDDL. Initially it was attempted to model the entire planning problem in PDDL, fulfilling all requirements stated in Chapter 3-1. To do so, an enormous amount of numerical expressions, using fluents, are needed to only model the locations of the waypoints, which is preferably avoided in PDDL. Besides that, the sheer amount of possible solutions results in unreasonable computation time. Hence, the planning system is split up, and the Task Planner is only responsible for planning the higher level actions.

4-2-1 Planning Variables

Before modelling the planning actions, the types, predicates and functions need to be defined. The entire planning domain only uses two types: `waypoint` and `area`[1]. Both types are locations, where the waypoint represents a specific point in the environment and the area represents a location where contacts can be revisited or where communication can be done. These locations are conceptual types, i.e., the actual longitude and latitude of these locations are not stored in PDDL. The domain contains four important functions, which are the planning variables and can be considered as the states of the planning problem. The functions are defined as shown in Listing 4.4.

Listing 4.4: Planning variables

```
(:functions
    ; Operation time (optimization metric) [s]
    (total_time)
    ; Accumulated location uncertainty of the vehicle [m]
    (total_uncertainty)
    ; Accumulated data that needs to be communicated [byte]
    (total_data)
    ; The total width that the vehicle currently has covered [m]
    (total_width)
)
```

The function `(total_time)` is the operation time of the AUV, similar to the example of Section 4-1. The location uncertainty is stored in the function `(total_uncertainty)` and `(total_data)` represents the accumulated survey data. The function `(total_width)` is the width of the area that is covered by the survey legs, as illustrated in Figure 4-1.

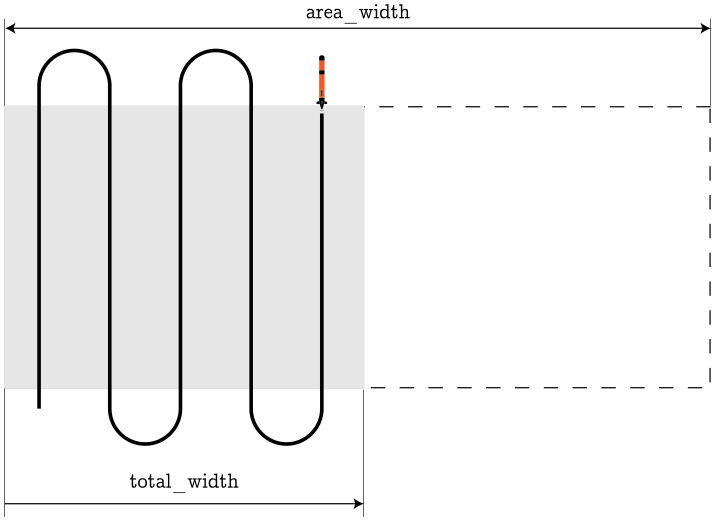


Figure 4-1: The function `(total_width)` represents the covered width by the AUV. The goal of the planning problem is to ensure that `(>= (total_width) (area_width))`.

[1]The reader is referred to Appendix B where the full PDDL domain file is presented. Note that the code snippets in this chapter only show the important aspects of the PDDL domain, as it would otherwise distract the reader and consume too much space.

In the problem file, the goal that needs to be achieved is defined, as shown in Listing 4.5. First of all, the entire area needs to be covered, i.e., the `(total_width)` needs to be larger or equal to the function `(area_width)`, which represents the width of the entire area. It is also desired that at the end of the survey task, all data is communicated. An optimization metric is defined as well, which tries to minimize the operation time of the AUV.

Listing 4.5: Planning goal and metric

```
(:goal (and
    (>= (total_width) (area_width))
    (<= (total_data) 0)
  )
)

(:metric
   minimize (total_time))
)
```

4-2-2 Location Uncertainty

To accomplish this goal, it is important to model the coverage width properly. When the planner comes with a plan that, according to the model, covers the entire area, one must be sure that in reality this coverage is achieved as well. For this, several assumptions are made:

- The position controller of the vehicle is able to follow the path accurately.
- The range of the sonar is constant, neglecting irregularities of the seafloor, as well as fluctuations of the underwater currents.
- The Nadir gap (which was shown in Figure 1-2 and Figure 3-3) is neglected.

As a result a constant `(sonar_range)` can be defined, which is the total width that the SSS can cover. An action `survey` is defined, where the AUV moves between two waypoints in a straight line over the area. During this action, `(total_width)` is increased, which is the only way to reach the goal of the problem. As already discussed in Chapter 3-1 and shown in Figure 3-2, the across-track uncertainty can be compensated by adding the effective width of the sonar to the total width w_{total}:

$$w_{total} = w_{total} + (r - u) \tag{4-1}$$

Where r is the sonar range and u is the location uncertainty at the first waypoint of the survey leg. This automatically implies that the planner will need to plan extra survey legs when the uncertainty grows too large, in order to reach the goal of covering the entire area. Previously, an attempt was made to model each leg as an object, where the goal was to cover each leg-object. The advantage is that less numerical values are required, improving the efficiency of the PDDL-solver. However, this way the across-track uncertainty could not be properly modelled, without introducing a significant amount of numerical expressions in the PDDL domain.

The along-track uncertainty is modelled by adding extra duration to the survey action. This extra time represents the elongation of the track due to the uncertainty (as was illustrated in Figure 3-1). The total duration t_{total} will therefore increase according to the following definition:

$$t_{total} = t_{total} + \frac{l + 2u}{v_s} \tag{4-2}$$

Here, l is the length of a survey leg, and u is the along-track uncertainty at the first waypoint of the leg. Dividing it by a predefined survey speed v_s gives the duration of the survey action. Thus, the survey action cost will increase proportional to the uncertainty. To get an optimal solution, the planner is forced to plan actions that reduce this uncertainty. These actions can be planned between the survey actions, during a transit. How the survey action is implemented in PDDL is shown in Listing 4.6.

Listing 4.6: The complete survey action in PDDL

```
(:action survey
  :parameters (?from ?to - waypoint)
  :precondition (and
      (auv_at ?from)
      (can_survey)
      (is_survey ?from ?to)
      ; The uncertainty should not exceed the sonar range
      (> (sonar_range) (total_uncertainty) )
      ; The data threshold should not be exeeded
      (< (+ (total_data) (survey_data)) (data_threshold) )
    )
  :effect (and
      ; Change AUV position to the new waypoint
      (not (auv_at ?from) )
      (auv_at ?to)
      ; Prevent doing a survey task twice
      (not (is_survey ?from ?to) )
      (not (is_survey ?to ?from) )
      ; After the survey a transit needs to be done
      (can_transit)
      (not (can_survey))
      ; Increase the total time by adding the time for surveying between
         the waypoints, taking into account the along-track uncertainty:
         total_time += (l+2u)/v_s
      (increase (total_time) (/ (+ (leg_length) (* 2 (total_uncertainty))
         ) (survey_speed)) )
      ; Increase the total uncertainty
      (increase (total_uncertainty) (uncertainty_survey) )
      ; Increase the accumulated data
      (increase (total_data) (survey_data) )
      ; Increase the covered width, taking into account the across-track
         uncertainty: total_width += r - u
      (increase (total_width) (- (sonar_range) (total_uncertainty)) )
    )
)
```

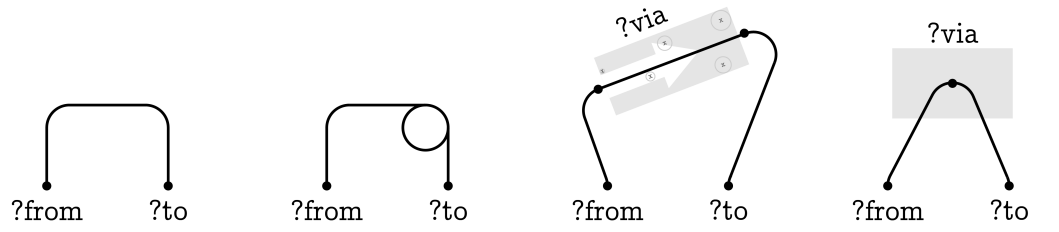
4-2-3 Transit Actions

The survey legs are not directly connected to each other. The AUV needs to transit from one leg to another, which in the most simple case is just a turn. In the case that the survey tracks are not nicely aligned with each other, or that the planner plans the tracks in an arbitrary order, the transit action will be a Dubin's path between these waypoints (as described in Chapter 5-2). In Listing 4.7 some effects of the transit action are shown. This is similar to the transit action of the example shown in Listing 4.1, except for the fact that the uncertainty is increased as well. This is important, as this discourages making long transits.

Listing 4.7: Some effects of the transit action

```
:effect (and
    ; Change AUV position to the new waypoint
    (not (auv_at ?from) )
    (auv_at ?to)
    ; Increase the total time by the transit duration
    (increase (total_time) (time_transit ?from ?to) )
    ; Increase the total uncertainty
    (increase (total_uncertainty) (uncertainty_transit ?from ?to) )
  )
```

During this period of transiting between the survey legs, the AUV is allowed to perform three different actions as well: a GPS-fix, a revisit or a communication action (see Figure 4-2). In the PDDL domain these actions are called `transit_gps`, `transit_revisit` and `transit_comm` respectively.



(a) `transit` **(b)** `transit_gps` **(c)** `transit_revisit` **(d)** `transit_comm`

Figure 4-2: The four different transit types: (a) the default `transit` between two waypoints, (b) a transit while going to the water surface to do a GPS-fix (e.g., by travelling in a helix shape to the surface), (c) a transit while revisiting an area with landmarks to reduce location uncertainty and (d) a transit while navigating through a communication area to communicate with other vehicles. Chapter 5 gives a detailed explanation of how these actions are generated.

Some of the effects of the `transit_gps` action are shown in Listing 4.8, which has some important differences with respect to the normal `transit` action. First of all, the uncertainty is not *increased* by a value, but an uncertainty value is *assigned* to `(total_uncertainty)`, indicating a reset instead of an increase. However, the value for `(uncertainty_gps)` is not equal to zero, as the GPS signal has a small amount of uncertainty, and descending back from the surface will introduce some new uncertainties as well. Consequently, `(uncertainty_gps)` is dependent on the depth of the survey area.

Besides that, the time is increased with a different value than the normal `transit` action. Since the transit is between the same waypoints, a new value is needed (provided by the Problem Generator) to store the time it will cost to perform a transit together with a GPS-fix. Finally, the data is reset to zero as well, since the data transfer at the surface is assumed to be instantaneous (as mentioned in Chapter 3-1).

Listing 4.8: Some effects of the transit_gps action

```
:effect (and
    ; Increase the total time by adding the time for travelling between
       the waypoints and performing a GPS-fix
    (increase (total_time) (time_gps ?from ?to) )
    ; Reset the total uncertainty
    (assign (total_uncertainty) (uncertainty_gps ?from ?to) )
    ; Reset the accumulated data variable
    (assign (total_data) 0)
  )
```

The `transit_revisit` action resets the total uncertainty as well, dependent on the area which is revisited. Therefore, an extra parameter `?via - area` is defined. The total time is increased in the same fashion. These effects are shown in Listing 4.9.

Listing 4.9: Some effects of the transit_revisit action

```
:effect (and
    ; Increase the total time by adding the time for travelling between
       the waypoints and revisiting an area
    (increase (total_time) (time_revisit ?from ?to ?via) )
    ; Reset the total uncertainty
    (assign (total_uncertainty) (uncertainty_revisit ?from ?to ?via) )
  )
```

Finally, the `transit_comm` action obviously sets the amount of data to zero and increases the uncertainty as well. The increase in time requires some extra computation to account for the time it costs to transfer the data, as defined in (4-3), since this can take quite a while.

$$t_{total} = t_{total} + t_{comm} + \frac{d_{total}}{r_b} \tag{4-3}$$

Here, t_{comm} is the time to transit from and to the communication area, d_{total} is the total amount of cumulated data and r_b is the bit-rate. This bit-rate is dependent on the area where communication is done. Listing 4.10 shows how this is modelled in the PDDL domain.

Also note that, in Listing 4.6, it is visible that a constant `survey_data` is used, which is the only source of data. This value is a predicted amount of data that each survey leg will produce. The survey action has a precondition which ensures that the AUV does not accumulate too much data during the survey mission. When this function `(data_threshold)` has a low value, the AUV is forced to communicate more often.

Listing 4.10: Some effects of the transit_comm action

```
:effect (and
    ; Increase the total time by adding the time for travelling between
       the waypoints and communcating at a communication area:
    ; t += t_comm + d/b_r
    (increase (total_time) (+ (time_comm ?from ?to ?via) (/ (total_data)
       (bit_rate ?via))) )
    ; Increase the total uncertainty
    (increase (total_uncertainty) (uncertainty_comm ?from ?to ?via) )
    ; Reset the accumulated data variable
    (assign (total_data) 0)
  )
```

One final important choice in modelling the survey task was to enforce the planner to alternate between transit actions and survey actions. The predicate `(can_survey)` is used, such that a survey action can only be planned when this predicate is true, and a transit action can only be planned when this predicate is false. This way, there cannot be two transit actions directly after each other. Several preconditions were added to increase the efficiency even more (see Appendix B). As a result, the planner has less options to search, making the planning procedure much more efficient.

4-3 Solvers

Having a model of the survey problem in PDDL, enables the usage of a wide variety of solvers. This section will explain how most PDDL-solvers will find a solution using heuristic search. Then, a selection of solvers is discussed and an explanation is given why OPTIC works best as PDDL-solver for this planning domain.

4-3-1 Heuristics

With PDDL planning, especially without metrics, actions do not have explicit costs. For planning this means that there is no obvious action to choose first. Performing an exhaustive search to find a valid set of actions is highly inefficient for combinatorial problems, since even for easy problems the search space might be exceptionally large. Therefore, more advanced methods are required to find a valid plan.

Two well known methods are GraphPlan [27], which is a graph-based approach for finding a valid set of actions, and SatPlan [28] which translates planning to propositional satisfiability. A third approach is planning using heuristic search, which was a novel method used in the Fast Forward (FF) solver. It outperformed all other planners during the International Planning Competition (IPC) in 2000. Even in the most recent competition (IPC 2018) the heuristics of the FF planning method were used [29]. Hence, a basic understanding of how these heuristics work is relevant.

FF-plan uses a search technique called Enforced Hill Climbing (EHC), which is a combination of Hill Climbing (local search) and systematic search using heuristics, as well as a powerful pruning technique (which cuts out search branches that are not valid). These heuristics

are based on the earlier mentioned GraphPlan method. GraphPlan constructs a so-called planning graph which is composed of two alternating types of layers: *proposition layers* and *action layers*, as shown in Figure 4-3. The *proposition layer* contains the propositions that represent the states at that time instance. The first layer of the graph is a *proposition layer* containing the initial states of the problem. Then follows the *action layer*, which contains the possible actions (described in the planning domain) of which the preconditions are met. The edges represent the precondition and effect relations between actions and propositions. There are three types of edges: (a) positive *(add)* relations, (b) negative *(delete)* relations and (c) propositions that are not changed by any action *(no-op)*. This way, a graph can be built, until a proposition layer contains all goal propositions, implying that the goal is reached.

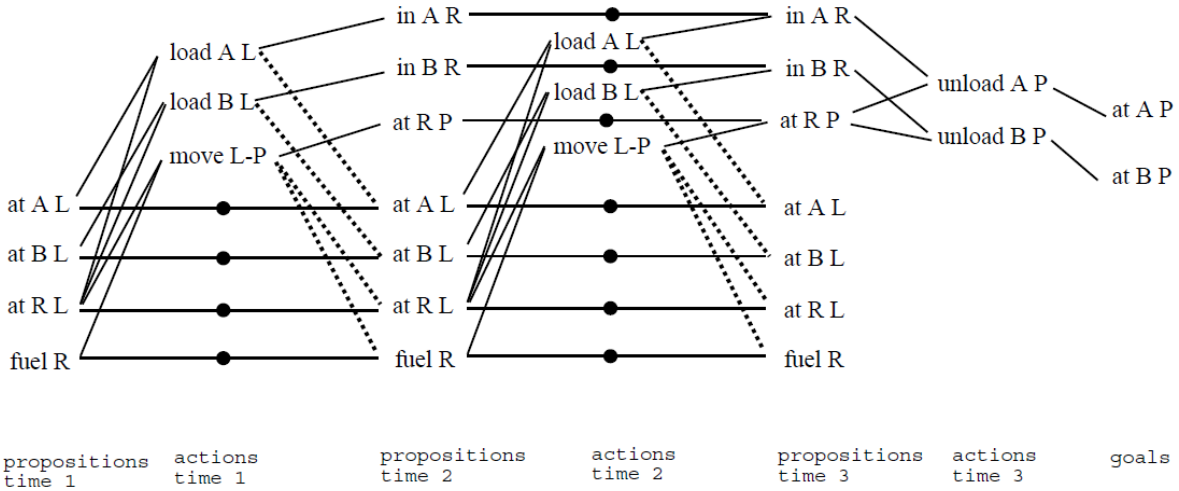


Figure 4-3: Example planning graph for a rocket problem with one rocket R, two pieces of cargo A and B, a start location L and one destination P. The solid lines represent *'add'* relations, the dashed lines represent *'delete'* relations and the lines with a dot are *no-ops*. The graph is not complete, but an indication of what it looks like. [27]

This method is a clever way to find a valid plan using a minimum amount of actions. FF uses a relaxed version of GraphPlan, omitting the *delete* relations, to estimate the number of actions that are probably needed to reach the goal. For each possible action, the states after this action are used as initial propositions of the planning graph. The resulting number of actions is used as heuristic for choosing this next action. This means that the EHC algorithm chooses the action that will probably result in the shortest plan. This is a fast and efficient way to solve a complex combinatorial problem, assuming that it is optimal to use the least amount of actions.

This means that all additional actions that might improve the quality of the plan are not even considered. Initially, the planning domain was defined as a `survey` and `transit` action, with separate actions for GPS-fixes, revisits and communicating at communication areas. This meant that extra actions were needed to find better plans, which is not even considered using the EHC algorithm. For that reason, the special actions were merged with the transit action, so that the best plan is also the plan which uses the least amount of actions.

This way, the least amount of actions points to a plan that needs the least amount of survey legs, but does not directly lead to better solutions as there are many ways to find such a plan.

Therefore, it is necessary to formulate a heuristic function that also takes the metric values into account. Hence, the basic FF algorithm suffices not for the survey problem.

During IPC 2002, PDDL 2.1 was developed, introducing metrics [30]. To comply with this PDDL version, an extension to FF was made, called Metric-FF [31]. It uses the same method as FF, taking into account the fluents in preconditions and effects of actions as well. Negative effects are neglected making the metric values monotonically increasing, which relaxes the planning graph. This works efficiently with numeric planning domains, although through this relaxation optimality might be affected. Also note that still the least amount of actions is taken, although capable of dealing with numerical effects and preconditions. However, the plan costs are still not taken into account. Newer PDDL-solvers, such as OPTIC, optimize the solution often resulting in more qualitative results [32], i.e., plans where the operation time of the AUV is lowest.

Two years later, a different planning method was developed, called Fast Downward (FD) [33]. Instead of using GraphPlan as basis for the heuristics, FD translates the PDDL problem to so-called causal graphs to compute its heuristic function. Due to the structure of these graphs, negative interactions of operators are not needed to be ignored, resulting in more accurate heuristics. Based on FD, a PDDL-solver called LAMA was developed, finding high quality plans [34]. For that reason LAMA is often used to compare different PDDL-solvers based on quality. The drawback of this method however, is that it only accepts one function `(total-cost)` and therefore essential functions such as `(total_uncertainty)` cannot be modelled.

4-3-2 Choosing a Solver

Choosing an appropriate solver for this planning domain is a challenge due to the large amount of planners being developed since the first IPC. The idea of defining a PDDL description of the planning problem is that multiple planners should be able to solve the same problem. However, each solver has its own approach in solving the problem, where the quality of the plans can differ significantly. Moreover, the planner needs to meet the following requirements:

- First of all, the PDDL-solver obviously needs to be able to meet the requirements defined in the PDDL domain file. This means that it needs to be able to handle the basic PDDL STRIPS functionality (which by definition each PDDL-solver should be able to). It needs to allow typing, and the use of functions (fluents). The use of functions and metric values already excludes a significant amount of PDDL-solvers, e.g., the FD planning method can only handle one specific function. However, in order to model location uncertainty properly, this is of main importance, making FD unusable.
- Besides having a valid plan that satisfies all preconditions and goals, it is important that the AUV completes its survey task as efficient as possible. Therefore, the planner should be able to provide qualitative plans. As the problem needs to be solved offline, at the start of its survey task, the computation time is less important.
- A final practical requirement is that the solver needs to be compatible with the ROSPlan framework. It is possible to expand ROSPlan to support more PDDL-solvers, but that was not considered as an option for this thesis as this would consume an unnecessary amount of time.

Multiple PDDL-solvers were tested, but only four of them appeared to be able to consistently solve the problems. These solvers are discussed briefly and a comparison is done between these planners. In Table 4-1 a comparison is shown between computation time and plan quality for the different planners.

Plan quality is defined by the resulting optimization metric: operation time. The lower the operation time of the AUV is, when executing the generated plan, the higher the quality of this plan is. The quality shown in Table 4-1 is therefore the value of `(total_time)` after the last action of the plan provided by the PDDL-solver.

Metric-FF

Metric-FF was the first planner of choice, as it is an efficient planner. However, it soon turned out that it was not finding the optimal plan, i.e., it does not consider the optimization metric properly. For even the simplest problem it failed to plan a simple lawnmower pattern, but instead started zigzagging. Switching the order of the waypoints in the problem file fixed this problem, indicating that it does not translate the action costs into its heuristic function.

Moreover, when modelling the along-track uncertainty in the survey action, the Metric-FF planner fails to find plans, as it cannot deal with non-constant effects on the optimization metric. Therefore, the values shown in Table 4-1 correspond to a planning domain where the along-track uncertainty is not modelled. Problems including communication constraints were unsolvable for the same reason. It can therefore be concluded that Metric-FF is not suited for the survey planning domain.

POPF

POPF was the default PDDL-solver of the ROSPlan framework and a successful planner during the third IPC [35]. Similar to Metric-FF it uses forward search using relaxed planning graphs as heuristics. In contrast to Metric-FF, the solver is able to solve all problems consistently. However, POPF fails to find high quality plans, often not even considering the use of uncertainty reducing actions.

LPG

LPG is a PDDL-solver that uses local search and planning graphs, called *numerical planning graphs* [36]. These graphs are similar to the planning graphs discussed before, however with a slightly different implementation of numerical relations. LPG was also a successful planner at the third IPC.

Using LPG for the PDDL domain of Section 4-2 resulted in higher quality plans, taking slightly more computational time. However, when the communication constraints are added to the problem domain, its computation time is significantly longer, being one of the slowest planners of this selection. This is probably due to the fact that these problems includes more than one goal.

OPTIC

In 2012 an improved PDDL-solver, called Optimizing Preferences and TIme-dependent Costs (OPTIC), was developed [32]. It uses a modified version of the POPF heuristic, and is able to comply with the newest PDDL versions. Instead of minimizing the number of actions, this planner focusses on the optimization metric. After an initial plan is found it repeats the planning procedure in order to improve on this optimization metric.

Table 4-1: In this table a comparison is made between PDDL-solvers for different survey problems. The value n is the number of predicates in the problem file. The number of predicates is proportional to the amount of available actions for the planner, thus giving an indication of the problem size. The best plan is highlighted in bold, both for the computation time as the plan cost. The plan cost is the value of `(total_time)` after the last action, giving an indication of the quality of the plan. All units in this table are in seconds.

		Metric-FF	POPF	LPG	OPTIC	OPTIC*
Problem 1	cost	2113	2342	**1882**	**1882**	2342
$n = 78$	time	**0.00**	0.02	0.76	1.42	0.02
Problem 2	cost	1781	2342	1672	**1504**	2342
$n = 188$	time	**0.00**	0.02	0.76	10.50	0.02
Problem 3	cost	1772	2342	1585	**1446**	2342
$n = 243$	time	**0.00**	0.04	2.76	51.58	0.04
Problem 4	cost	-	2446	**2063**	**2063**	2446
$n = 188$	time	-	**0.10**	312.27	27.44	0.12
Problem 5	cost	1772	2342	1503	**1437**	2342
$n = 352$	time	**0.00**	0.06	2.01	114.68	0.08
Problem 6	cost	-	1676	1658	**1613**	1676
$n = 298$	time	-	**0.22**	326.53	48.46	**0.22**
Problem 7	cost	-	1672	1639	**1596**	1672
$n = 462$	time	-	0.44	62.08	856.42	**0.42**

* Without optimization

In Table 4-1 it is visible that OPTIC has consistently the highest quality, i.e., plans with the lowest cost. However, it is also clear that OPTIC takes significantly more time to solve the problems. Since the planning system is intended to plan offline, the computation time is not that important. Hence, the chosen planner for this planning system is OPTIC.

Each solver has several options to change its behaviour. OPTIC can be used without the optimization loop as well, decreasing the computation time. From this it is directly visible that OPTIC uses the same algorithms as POPF, as the quality of the plans is exactly the same for each problem.

It might be valuable to explore more PDDL-solvers to further improve the quality of the planning system. For this thesis, OPTIC is sufficient to show the capabilities of the planning system and show the benefits of using PDDL as modelling language. The next chapter will discuss how the planning system will provide a relevant problem file for the Task Planner, such that these plans can generated.

Chapter 5

Planning Procedure

5-1 Problem Generation

The Task Planner requires both a domain file and a problem file. The domain file, as modelled in Chapter 4, is static for each planning situation. The problem file however, needs to be generated by the planning system such that the Task Planner can solve it. Recalling the planning system described in Chapter 3-2, this problem file is generated by the Problem Generator. The survey planning domain is kept as simple as possible, avoiding complicated numerical expressions. These computations are passed on to the Problem Generator, which will be discussed in this section.

Figure 5-1 schematically shows the overall structure of the Problem Generator. The Problem Generator consists of five sub-planners, each responsible for its corresponding action in the PDDL domain file:

- **Survey Planner** - The survey planner is the main planner of the Problem Generator, generating the numeric information of the `survey` action. This sub-planner decides where to place the survey legs and its corresponding waypoints. The locations of these waypoints are relevant for all other sub-planners as well as the Path Planner.
- **Transit Planner** - This sub-planner plans when a transit between waypoints is allowed.
- **GPS Planner** - Additionally, the GPS Planner plans when and in what way a GPS-fix can be performed.
- **Revisit Planner** - Based on a list of contacts, revisit areas are defined in this sub-planner.
- **Communication Planner** - Lastly, the transits between waypoints and the different communication areas are planned.

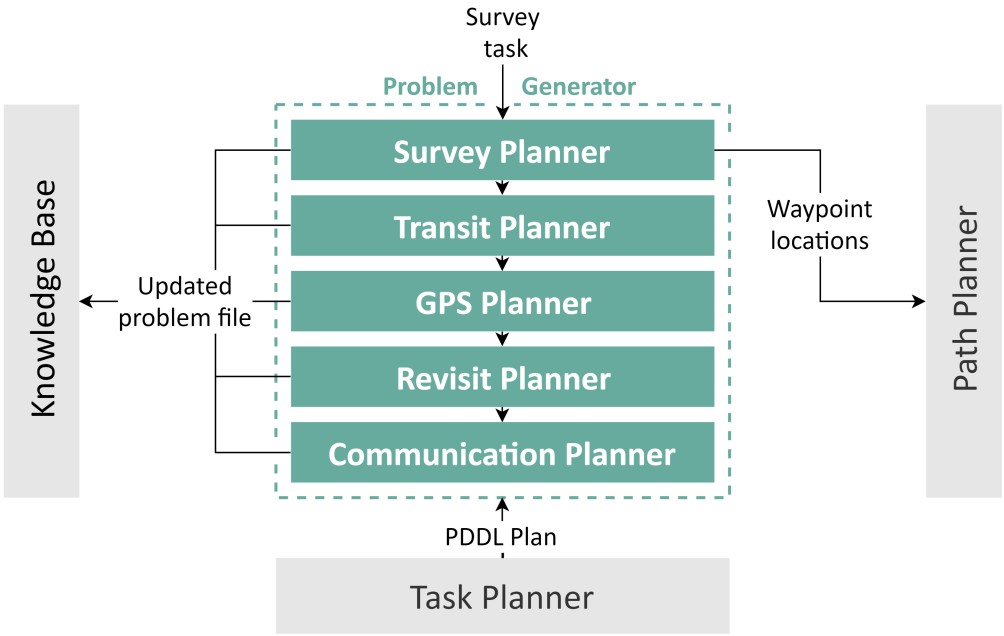


Figure 5-1: The Problem Generator contains several sub-planners. Each sub-planner is responsible for its corresponding action type, and generates the problem. The survey planner is the main sub-planner on which the other sub-planners depend.

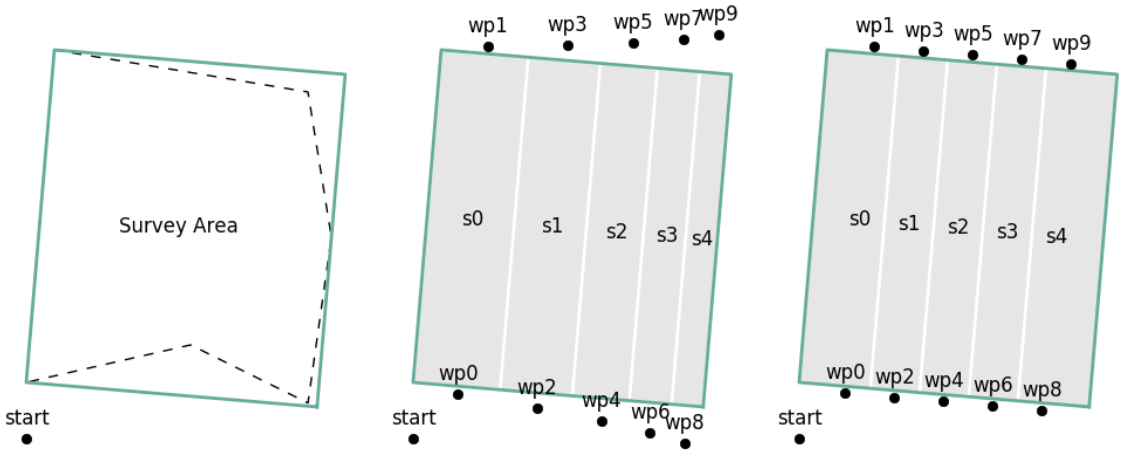
This results in information about when the different actions are allowed, the cost (which is the duration) of the actions and the uncertainty growth resulting from the actions. Especially these numerical values are important, as these values are the parameters of the heuristic function inside the PDDL-solver. All this information is stored in the Knowledge Base, such that the Task Planner can generate a problem file and start planning.

5-1-1 Survey Planner

The whole planning procedure starts with a survey task, which is received by the Problem Generator. This survey task includes a survey area and an initial position of the AUV, being used by the survey planner. The survey planner will split up this area in order to define waypoints and survey legs. In this subsection this procedure is discussed and shown how this is translated to PDDL in a problem file.

It is assumed that the survey area has a rectangular shape, as vessels normally travel in straight tracks. If the survey area appears to have a polygonal shape which is not rectangular, this is first transformed to a rectangular shape by finding its minimum bounding box, as shown in Figure 5-2a. Although algorithms exist that can plan a coverage path over arbitrary polygonal areas [37], this is not included in the planning system. Yet it can be extended by it, when suitable.

Starting from the side of the area closest to the starting point, survey tracks are laid in order, assuming that no uncertainty reducing actions are performed. By default, the survey planner will lay the tracks along the longest edge of the area, but this can be changed to the shortest edge by changing a parameter of the survey planner.



(a) Initial area (dashed line) with bounding box (solid line)

(b) Survey plan

(c) Survey plan for Task Planner

Figure 5-2: The survey planner (a) creates a rectangular area from an arbitrary polygonal survey area, then (b) divides the area into survey legs assuming that no uncertainty reducing action is performed and finally (c) makes the legs equidistant and of equal length for the Task Planner. Waypoints are generated at the endpoints of the line segments, which will be used by all other planners.

This results in tracks that decrease in width and increase in length (see Figure 5-2b), as was discussed in Chapter 3-1. However, this will result in false heuristics, as this already implies the order of the tracks which should be chosen by the Task Planner instead of the Problem Generator. Therefore, the along-track uncertainty is not used such that all track have equal length. For the same reason, the planned legs are distributed equally over the area, such that transiting between the survey legs has equal cost (as shown in Figure 5-2c). This way, enough survey tracks are planned without influencing the heuristics of the Task Planner. The resulting waypoints can then be stored in the problem file as `waypoint` objects. An example is shown in Listing 5.1.[1]

Listing 5.1: Declaration of waypoint objects in the PDDL problem file

```
(:objects
  start wp0 wp1 wp2 wp3 wp4 wp5 wp6 wp7 wp8 wp9 - waypoint
)
```

Besides that, the PDDL problem is initialized by connecting the waypoints using the predicate `(is_survey)`, as is visible in Listing 5.2. Each connection needs to be defined in both ways, such that the planner is able to plan a survey action in both directions. The length of a survey leg, for computing the duration of a survey action, and the uncertainty growth during a survey action is initialized as well. These are the same for each survey leg, neglecting the additional leg length due to the along-track uncertainty. Finally, the initial position is defined as well as the total width of the survey area. The predicate `(can_transit)` indicates that the planner should start with a transit action.

[1] For an example of a complete problem file, the reader is referred to Appendix B

Listing 5.2: Initialization of states by the survey planner

```
(:init
  ; Initial position of the AUV
  (auv_at start)
  ; The AUV needs to start with a transit
  (can_transit)
  ; Connect all waypoint as survey leg
  (is_survey wp0 wp1) (is_survey wp1 wp0) (is_survey wp2 wp3)
  (is_survey wp3 wp2) (is_survey wp4 wp5) (is_survey wp5 wp4)
  (is_survey wp6 wp7) (is_survey wp7 wp6) (is_survey wp8 wp9)
  (is_survey wp9 wp8)
  ; Length of one survey leg [m]
  (= (leg_length) 356.27)
  ; The uncertainty growth during a survey action [m]
  (= (uncertainty_survey) 6.31)
  ; The width of the area that needs to be covered [m]
  (= (area_width) 310.79)
)
```

5-1-2 Transit Planners

The information of the survey action is now properly generated into the PDDL problem file. Having a set of waypoints and their locations, makes it possible to define the transits that can be planned between these waypoints, as well as the durations and uncertainty growth. This subsection describes how this is done in the four different transit planners that were shown in Figure 5-1.

Transit Planner

The transit planner takes the waypoints of the survey planner as input, and determines the connections between waypoints that the AUV is allowed to transit between. Creating these connections has a major influence on the computation time of the PDDL-solver. The solution space grows exponentially with the amount of transit connections. Hence, these connection needs to be chosen wisely, without reducing the freedom of the Task Planner. The current implementation makes for each waypoint a connection with a specified number of closest adjacent waypoints and checks whether the distance between the waypoint is below a certain threshold. The number of connections as well as the threshold are parameters that need to be given to the planning system.

These connections can be stored in the PDDL problem file as shown in Listing 5.3. The time to transit between these waypoints is determined by planning a Dubin's path between the two points (which is explained in Section 5-2). The resulting duration and uncertainty growth is stored in the PDDL problem file as well.

Listing 5.3: Initialization of states by the transit planner - only a part of the connections is shown

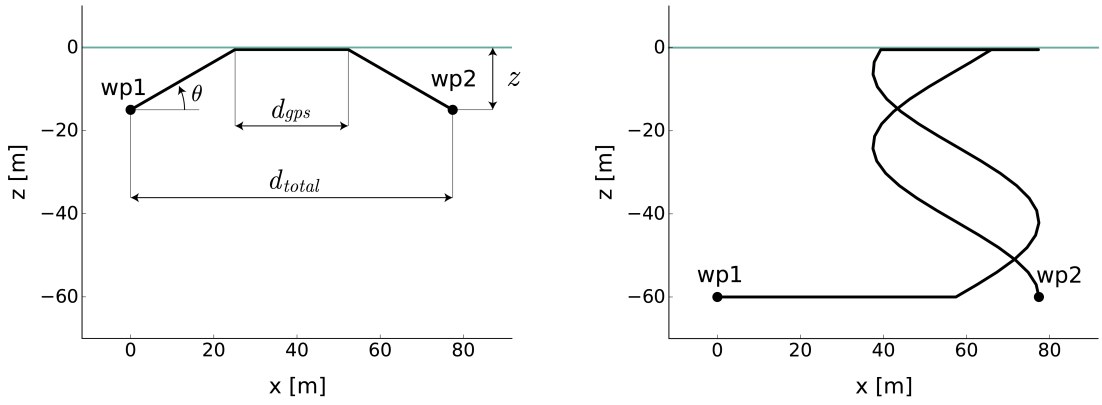
```
(:init
  ; Define all transit connections between the waypoints
  (is_transit start wp0) (is_transit wp0 start) (is_transit wp0 wp2)
  (is_transit wp2 wp0) ;... etcetera
```

```
  ; The time to transit between the waypoints [s]
  (= (time_transit start wp0) 47.22) (= (time_transit wp0 start) 47.22)
  (= (time_transit wp0 wp2) 37.34) (= (time_transit wp2 wp0) 37.34)
  ; The uncertainty growth between the waypoints [m]
  (= (uncertainty_transit start wp0) 1.53)
  (= (uncertainty_transit wp0 start) 1.53)
  (= (uncertainty_transit wp0 wp2) 1.21)
  (= (uncertainty_transit wp2 wp0) 1.21)
)
```

GPS Planner

The GPS planner creates connections between waypoints the same way as the transit planner does. However, to compute the transit time and uncertainty, some extra computations are needed. A GPS-fix can be done in two ways (also shown in Figure 5-3):

- **Simple** - The AUV can navigate the original transit path between two waypoint, while ascending to the surface and descending back to the second waypoint.
- **Helix** - If the distance between the two waypoints is too short to be able to do this, the AUV needs to ascend and descend in a helix shape.



(a) Simple **(b)** Helix

Figure 5-3: There can be planned two types of GPS-fixes: (a) a simple GPS-fix and a (b) helix-shaped GPS-fix. Which type of path is used depends on the depth z of the waypoints, the maximum pitch θ of the AUV and the distance d_gps required to get a reliable GPS signal.

The required distance to be able to do a *simple* GPS-fix, is determined by the depth z of the waypoints, the maximum pitch angle θ of the AUV and the distance d_{gps} which is at least needed to stay at the surface for receiving a reliable GPS-signal. If the required distance is larger than te total distance d_{total} a helix-shaped path is needed to do a GPS-fix, i.e., when the inequality of (5-1) is true, a helix is required.

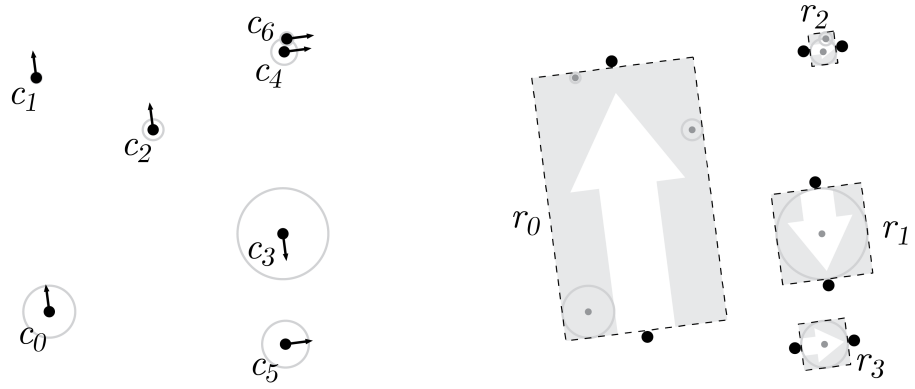
$$d_{total} < \frac{2z}{\tan\theta} + d_{gps} \tag{5-1}$$

When the GPS-path is determined, the duration of the GPS transit can be computed, as well as the uncertainty. Recall that the uncertainty is reset, instead of increased. The uncertainty is the sum of the variance of the GPS-signal and the uncertainty growth while descending back to the second waypoint. These values are stored in the PDDL problem file as well, in a similar fashion as Listing 5.3.

Revisit Planner

As stated in Chapter 3-1, the survey problem might include already discovered contacts that can be revisited by the AUV to reduce its location uncertainty by means of SLAM. These contacts have a location uncertainty themselves as well, and have a preferential direction from which to approach them. Since the contacts are discovered using sonar, they can look much different when detecting it from another viewpoint. Therefore, the revisit planner ensures that the AUV revisits the contacts from the same direction.

To do so, clusters are formed of contacts based on their direction and position, as shown in Figure 5-4. A cluster is formed when (i) the contacts lay close to each other and (ii) when the direction of the contacts does not vary too much. The maximum width of the clusters is defined by the expected sonar performance. The maximum length of the cluster and maximum angle variation between the contacts needs to be provided as a parameter. The resulting clusters form the revisit areas that can be used for planning. The revisit areas can be stored in the PDDL problem file, as shown in Listing 5.4.



(a) Contacts with different orientations **(b)** Clustering of contacts

Figure 5-4: This figure shows how revisit areas r_i are formed, by clustering the contacts c_i. In (a) an arbitrary set of contacts is shown with their direction and location uncertainty ring. The Revisit Planner finds revisit areas as shown in (b), with a given direction (depicted by a white arrow). At both ends of the revisit area, waypoints are defined that can be used for planning a path.

Listing 5.4: Decleration of area objects in the PDDL problem file

```
(:objects
  r0 r1 r2 r3 - area
)
```

At each end of the revisit area a waypoint can be created and a path can be planned between the two original transit waypoints and the waypoints of the revisit area. Having this path, the duration and uncertainty growth can be determined. The location uncertainty is chosen to be the average location uncertainty of all contacts in the area. Taking the average uncertainty obviously is not a good model for SLAM, but acts as a place-holder to provide the planning system some representative planning variables.

Communication Planner

The communication planner is a relatively simple planner, since the communication areas are defined in the survey task. This planner will find the centre of the polygonal communication areas and computes the travel distance between the waypoints when navigating via a communication area. Based on this distance, the duration and uncertainty growth can be computed and stored in the PDDL problem file (see Listing 5.5). An expected bit-rate per area needs to be provided in de PDDL problem file as well, such that the duration of the `transit_comm` can be computed.

Listing 5.5: Problem generation by the communication planner

```
(:objects
  c1 c2 - area ; Define communication areas
)

(:init
  ; Connections between waypoints and communication areas
  (is_comm wp5 wp2 c2) (is_comm wp1 wp2 c2) (is_comm wp3 wp4 c1)
  ; Store the bit rate per area [bit/s]
  (= (bit_rate c1) 10) (= (bit_rate c2) 8)
  ; Transit time between the waypoints via the communication area [s]
  (= (time_comm wp5 wp2 c2) 100.39) (= (time_comm wp1 wp2 c2) 98.15)
  (= (time_comm wp3 wp4 c1) 76.99)
  ; Uncertainty growth for transiting via the communication area [m]
  (= (uncertainty_comm wp5 wp2 c2) 10.7) (= (uncertainty_comm wp1 wp2 c2)
   9.36) (= (uncertainty_comm wp3 wp4 c1) 4.2)
)
```

5-2 Path Planning

At the end of the planning process, the final plan and waypoint locations are given to the Path Planner. the Path Planner takes the order of the waypoints from the Task Planner, and combines them with the waypoint locations of the Problem Generator. Based on the locations and the type of action (transit or survey) it will plan a path through the waypoints. It will take into account the dynamical constraints of the vehicle, such that it is able to smoothly follow the path.

The path planning algorithm is based on Dubin's curves. A curved path is required because the system is non-holonomic, i.e., it cannot move sideways. Assuming constant velocity and maximum steering angle, the minimum radius of the curves can be determined. With this radius, an optimal path can be found between the two waypoints, as shown in Figure 5-5.

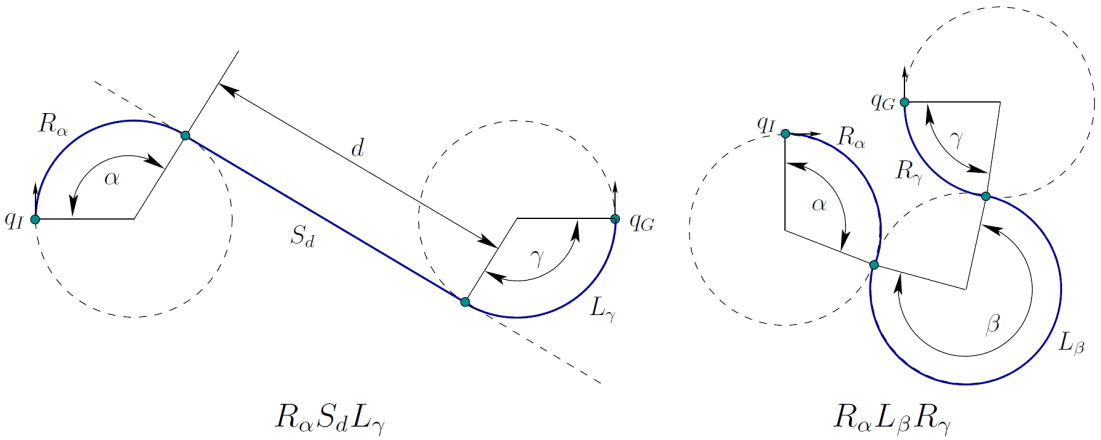


Figure 5-5: The Path Planner uses Dubin's curves to plan a path between two waypoints with given heading. In this figure two examples of Dubin's curves are shown. [38]

A Dubin's curve consists of curves and straight segments, denoted by R for a curve to the right, L for a curve to the left and S for a straight path segment. Each path consists of a sequence of three segments, which together is called a Dubin's word. Dubin's showed that only these six words are possibly optimal [38]:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\} \tag{5-2}$$

These Dubin's curves assume that the entire plane is the configuration space of the vehicle, thus not considering obstacles. However, when planning with multiple vehicles, it might be necessary to introduce no-go areas. These can be modelled as obstacles for the Path Planner. Some attempts have been made to plan a path for non-holonomic robots in an environment filled with obstacles [39, 40], which might be useful to implement.

When a plan is given by the Task Planner, the list of actions can be translated to a sequence of waypoints with a heading. Both waypoints of a survey track need to have the same heading such that the survey track will be a straight line. The same is done for revisit transits. This is important, because straight paths will result in better sonar images. The transit paths are planned such that the AUV will arrive with the right heading at the start of a survey leg or revisit area.

The Path Planner will plan the paths for GPS-fixes as well. The GPS Planner only plans GPS-fixes to predict the duration of a GPS-fix, while the Path Planner needs to define a path in 3D that can be followed by the AUV. To do so, a Dubin's path is planned between the two waypoints (like a normal transit) and then adjusted such that it reaches the surface. When a helix is required, the last turn of the Dubin's path will be expanded to a helix.

As a result, the Path Planner is able to plan a path for each action of the Task Planner. An example of a path is shown in Figure 5-6c. The path is given to the Action Dispatcher, which can send the path to the LAUV by means of a *follow path* message. This is the only type of message needed, since all actions can be translated to a path that needs to be followed.

5-3 Iteration Process

To summarize, the planning system will start by generating a problem file in the Problem Generator, based on the survey task that is given to the planning system. The Problem Generator will generate all necessary numerical values for the Task Planner, to be able to find a plan. Based on this initial problem, the Task Planner finds an initial plan.

However, in Chapter 3-2 it was explained that the planning system needs some form of iteration due to the numerical inaccuracy of PDDL. When an uncertainty reducing action is planned, the waypoint locations will change, making the plan after this uncertainty reducing action invalid. Therefore, the plan after this uncertainty reducing action is re-planned in the following way:

i. The planned actions after the first new uncertainty reducing action are discarded. Figure 5-6a shows an example plan which contains one GPS-fix between `wp2` and `wp4`. The red line is the part of the plan that is kept, and the black line represents the actions that are discarded.

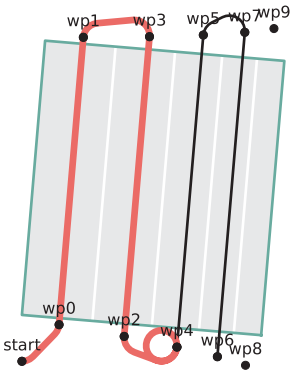
ii. Starting from the waypoint where the GPS-fix ends (`wp4` in this example), a new sub-plan is found that covers the area that is left (the teal rectangle). To compensate for the numerical imprecision of the PDDL model, two alternative plans are produced with a GPS-fix one transit earlier and one transit later (as shown in Figure 5-6b). In this example, it means that the Task Planner will also start planning from `wp3` and `wp7` respectively.

iii. The costs (operation time), of the three plans are compared, and the plan with the lowest cost is chosen. In this example, the second sub-plan turns out to be the best. This results in the final plan as shown in Figure 5-6c.

If the chosen sub-plan appears to have a new uncertainty reducing action, the above three steps are repeated until no new uncertainty reducing action is found. When no new uncertainty reducing actions are planned, all survey legs of the plan are redistributed in such a way that each survey leg has equal overlap. The waypoint locations are updated according to the uncertainty model, to accurately account for location uncertainty. This is visible in the example, where the cost of the final plan is slightly increased in comparison to the second sub-plan due to this update of waypoint locations.

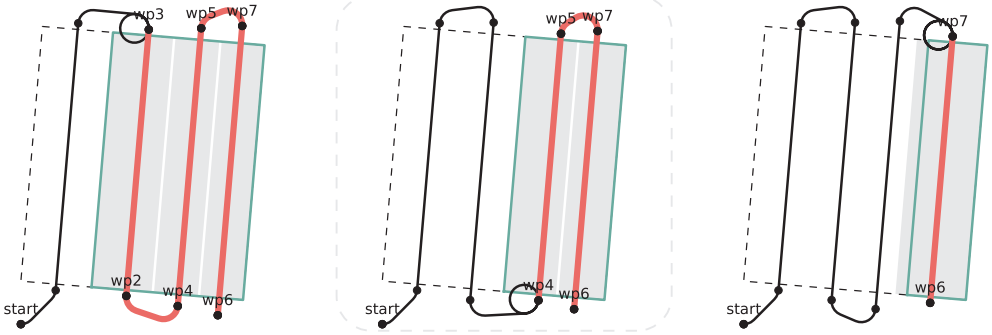
The example showed the iteration process of a plan with a GPS-fix. When a revisit is planned, the waypoint locations will change as well. However, the cost of a revisit action is highly dependent on the waypoint positions. For that reason it will not make sense to consider planning the revisit action one transit earlier or one transit later. Hence, for revisits these two alternative plans are not considered.

Finally, the waypoint locations together with the final plan is sent to the Path Planner. The Path Planner will plan a path, as described in the previous section, and will send it to the vehicle to be executed. In the next chapter, some scenarios are planned and sent to the simulation environment to evaluate the plans.



; Cost: 1501.84
(transit start wp0)
(survey wp0 wp1)
(transit wp1 wp3)
(survey wp3 wp2)
(transit_gps wp2 wp4)

(a) Initial plan, actions after the first GPS-fix are discarded

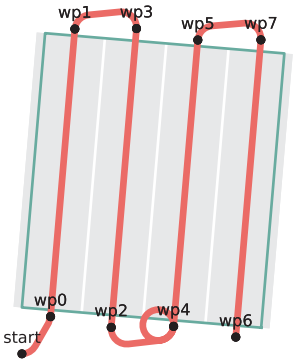


; Cost: 1517.11
(survey wp3 wp2)
(transit wp2 wp4)
(survey wp4 wp5)
(transit wp5 wp7)
(survey wp7 wp6)

; Cost: 1493.90
(survey wp4 wp5)
(transit wp5 wp7)
(survey wp7 wp6)

; Cost: 1515.38
(survey wp7 wp6)

(b) Three sub-plans, starting from the GPS-fix of the initial plan, as well as one transit earlier and one transit later



; Cost: 1496.05
(transit start wp0)
(survey wp0 wp1)
(transit wp1 wp3)
(survey wp3 wp2)
(transit_gps wp2 wp4)
(survey wp4 wp5)
(transit wp5 wp7)
(survey wp7 wp6)

(c) The final plan

Figure 5-6: This figure shows an overview of the planning procedure, showing the plans graphically together with the PDDL-solver output. When an initial plan is found, the actions after the first (new) GPS-fix are discarded (a), where the red line is the planned path and the black line are the discarded actions. Then, three sub-plans are found (b) starting from the waypoint where the GPS-fix was planned, as well as one transit earlier and one transit later. The plan with lowest cost is selected (in this case the second one), which becomes the final plan (c). Notice that the plan cost slightly increased due to the change of waypoint locations.

Chapter 6

Evaluation

There is an important distinction between *evaluation* and *validation*. Evaluation is the assessment of the planning system on how well it behaves in terms of predefined metrics such as operation time and coverage. Validation on the other hand, determines how well the planning domain describes the survey problem, and whether the plans are valid in real world applications. Both topics will be discussed in this chapter, in Section 6-1 and Section 6-2 respectively.

6-1 Simulation Results

To evaluate the quality of the planning system, two main things are examined:

i. Whether the planning system is capable of finding suitable plans for different situations

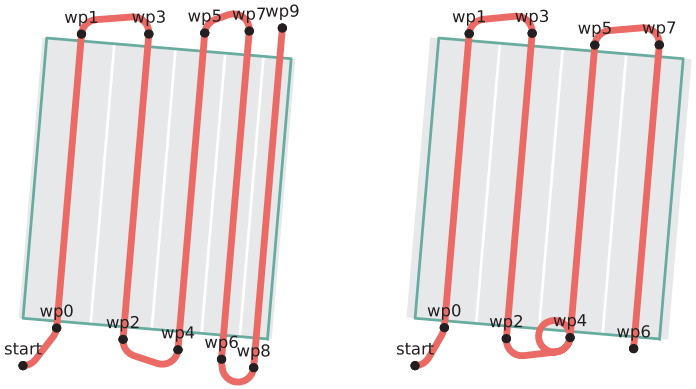
ii. Whether the planning system reliably finds plans that cover the entire area

If this is the case, the requirements stated in Chapter 3-1 are satisfied. To evaluate the coverage of the generated plans by the planning system, the simulation environment described in Chapter 2 is used. A larger area is simulated to clearly show the difference made by the planning system. Since SLAM and acoustic communication is not yet modelled in the simulation environment, the `transit_revisit` and `transit_comm` could not be simulated in the simulation environment. Nonetheless, the plans made by the planning system clearly show how the planning system is able to make decisions in complex situations.

In the following subsection, a few scenarios are tested on the planning system in order to highlight the decisions made by the planning system. Following, a larger scenario is simulated, to show that the planning system is in fact capable of achieving full coverage in a consistent way, in contrast to a basic lawnmower pattern.

6-1-1 Planning Scenarios

To visualize the decisions made by the planning system, the survey area used in Chapter 5, is given to the planning system. This area is approximately 300 by 350 metres, at a depth of 40 metres beneath the surface. When no GPS-fix is allowed, the planning system will only compensate for the location uncertainty (see Figure 6-1a). When following the path, the grey area is what is covered by the AUV, which means that in this scenario the planning system is fully able to cover the entire area without the use of any uncertainty reducing action.



(a) Without GPS-fix **(b)** With GPS-fix

Figure 6-1: Scenario where (a) no uncertainty reducing action is allowed and (b) only GPS-fixes are allowed.

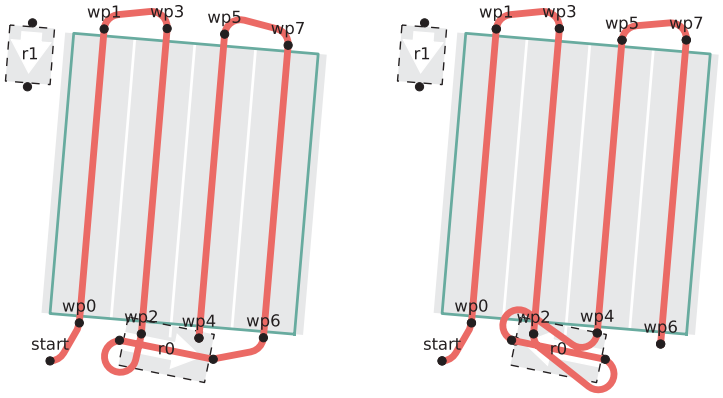
GPS-fixes

When the planning system is allowed to plan a `transit_gps`, it will plan one halfway (see Figure 6-1b). As a result, one survey leg less is required to cover the entire area. This clearly reduces the operation time, although the GPS-fix in this case is a helix shaped path. When the survey area is closer to the surface, more GPS-fixes will be planned, since they will cost less time (and no helix shaped path is required to reach the surface).

Revisits

In the next scenario a revisit area is added at the south of the survey area. In the first case, the contacts need to be revisited from west to east. In Figure 6-2a it is visible that the planning system switches the order of the legs to efficiently revisit the area. This does not happen in the case where the contacts need to be revisited from east to west (see Figure 6-2b). In both cases the number of legs is reduced by one, reducing the operation time compared to the case where no uncertainty reducing actions are used. A small rectangular revisit area was added to the scenario as well, but the planning system decided not to use this area as this would involve long distances, making it unprofitable.

The planning system will change its plan depending on the depth. When the AUV needs to operate at -12.5 m, the planning system will prefer GPS-fixes, while at a depth of -40 m, the planning system will plan revisits (if available) instead. This confirms that the planning system takes operation time into account.

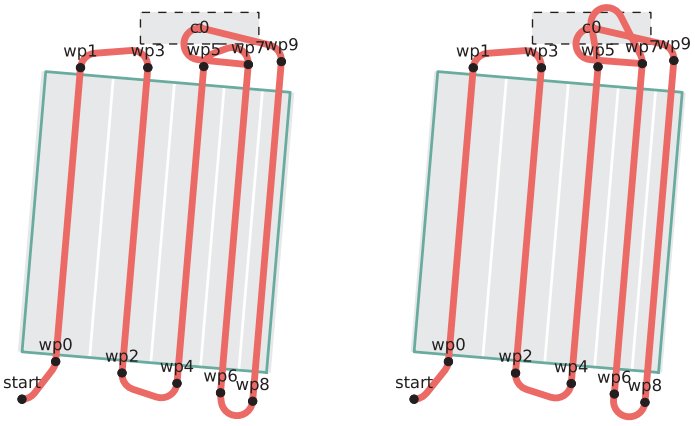


(a) Revisit area in east direction **(b)** Revisit area in west direction

Figure 6-2: Scenario with (a) an area where contacts need to be revisited in east direction and (b) in west direction. The planning system decided not to use the revisit area northwest to the survey area, since operation time will not be reduced by visiting that area.

Communication

Figure 6-3 shows plans for communicating at a communication area. As a consequence of how the goal is formulated, the planner is forced to plan as the last action a `transit_comm` or `transit_gps`. The plan in Figure 6-3a just ends with a `transit_comm`. In Figure 6-3b a data threshold was added, such that the planning system needed to plan a second `transit_comm` so that this threshold is not exceeded.



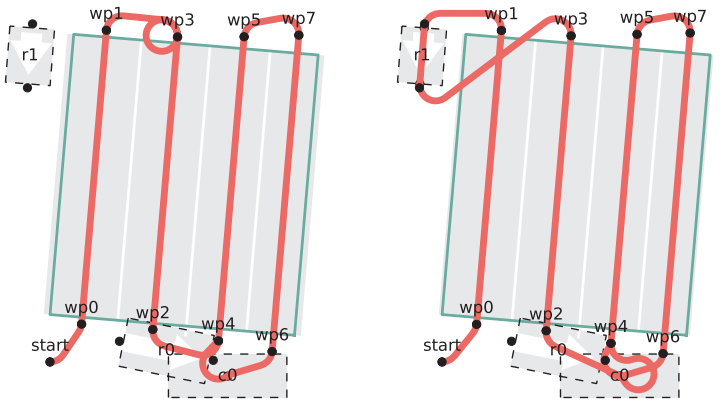
(a) Without threshold **(b)** With threshold

Figure 6-3: Scenario with a communication area, (a) without any data threshold and (b) with data threshold. Communication while doing a GPS-fix was not allowed in this scenario.

From the figures it becomes clear that, if the uncertainty grows too large, the AUV might miss the communication area. The PDDL domain should be improved by adding a precondition which ensures that the uncertainty is not larger than the shortest distance from the centre of the area to one of the area borders.

Combined scenario

The planning system is able to deal with more complex situations as well. Figure 6-4 shows a scenario where the planning system plans a path, while having the freedom to plan all available actions. The planning system can solve this without failing, although it takes significantly more computation time. It also becomes clear that GPS-fixes are the most preferred actions, unless the depth of the survey area is too large.



(a) Communication at the surface is allowed

(b) Communication at the surface is not allowed

Figure 6-4: Scenario with two revisit areas and a communication area. In figure (a) it is allowed to communicate at the surface (i.e., it needs to communicate with a surface vehicle) and in figure (b) this was not allowed.

In Figure 6-4b, the AUV was not allowed to communicate above the surface. Due to the depth of 40 metres, the planning system prefers to revisit the top-left revisit area. This shows a limitation of how the planning domain is formulated. The AUV needs to visit the communication area twice (due to a data threshold) and pass the larger revisit area while sailing to the communication area. It probably would be more efficient to combine a revisit at the bottom area together with a communication action. This however is not possible with the current formulation of the planning domain, since the AUV is forced to alternate between survey legs and single transit actions.

6-1-2 Coverage

The above results show that the planning system is capable of solving a variety of problems, giving plans that are reasonable. However, this does not guarantee that it is capable to cover the entire survey area, which was the first requirement of the planning system (as stated in Chapter 3-1). To evaluate this, a scenario with a larger survey area (approximately 600 by 350 metres) is defined in order to visualize the difference between the plans made by the planning system and a standard lawnmower pattern.

The path generated by the planning system is sent to the control interface, using a *follow path* message. The LAUV is simulated to follow the path in the Gazebo simulation environment. However, the simulation environment lacks three important things: (i) SLAM is not implemented as a localization method, (ii) acoustic communication is not modelled and (iii)

the model of the side-scan sonar was not implemented yet. The latter shortage means that coverage cannot directly be quantified using the sensor data. Hence, for these simulations, the spatial information of the simulation is used to estimate the area that has been covered.

Recall that coverage was defined as the percentage of the area of which the POD is above a certain threshold. Defined this way, having a 100% coverage does not mean that all objects have a POD equal to 1. When this threshold is low, there is an increased chance that objects remain undetected. The planning system should assure that the coverage, as defined, is 100%. The risk for vessels to navigate through the area, is therefore dependent on the POD at which the threshold is set.

Figure 6-5 visualizes how the coverage of the vehicle is derived from the path that the AUV has travelled. It is based on the sonar range that was used for planning, and it is assumed that this sonar range is constant. Only the coverage of the straight survey tracks is considered, since the quality of sonar images is poor when making turns. The resulting coverage area is the intersection of the survey area and the rectangular coverage areas of each leg.

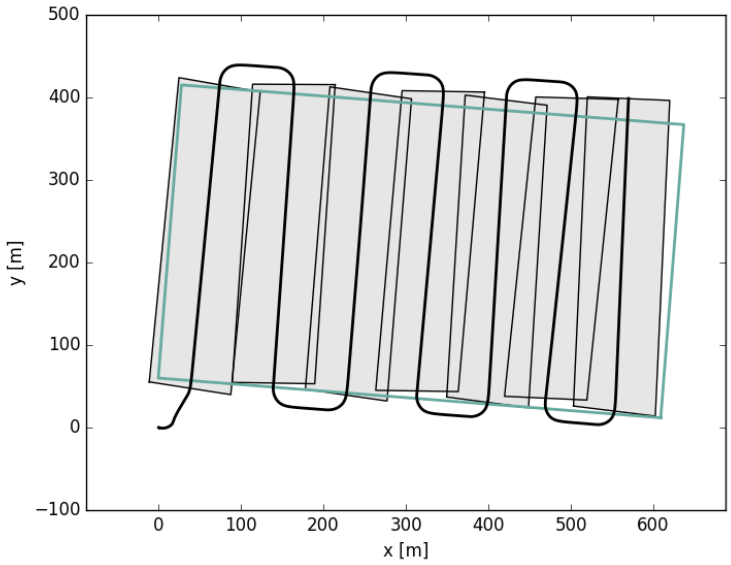
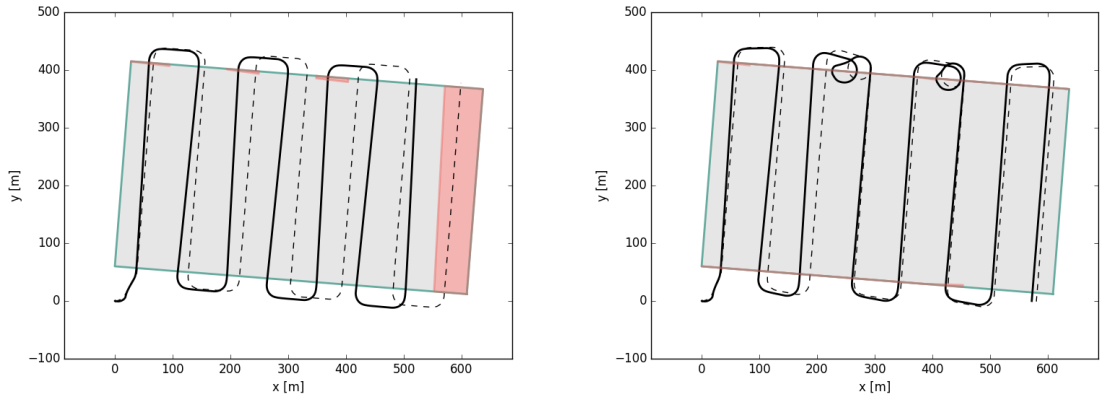


Figure 6-5: The grey rectangular shapes visualize the coverage of each survey leg. The teal rectangle is the survey area and the solid black line is the trajectory followed by the AUV during simulation.

The coverage of a simple lawnmower pattern is compared with a path planned by the planning system. In Figure 6-6, the results of a simulation are shown. The integration drift is clearly visible, resulting in gaps in the coverage. The grey area is the covered area and the red area is the part of the survey area that was not covered. In the figure, only 90% of the area was covered by a basic lawnmower pattern. The planning system planned two GPS-fixes and an additional leg to ensure total coverage, successfully compensating the across-track uncertainty. Still, some small fragments are not covered, which is the result of how the coverage per leg is defined. Therefore, the coverage values only serve as an indication but do not resemble the real coverage. To properly quantify the coverage, the simulation environment should be improved, such that the side-scan sonar model can be utilized.



(a) Basic lawnmower pattern

(b) Path from planning system

Figure 6-6: Simulation of the LAUV, performing a survey task on a rectangular area (600 by 350 metres). The solid black line is the real trajectory of the LAUV, the dashed line is the planned path and the solid green line is the area border. The covered area is marked grey and the area that is not covered is marked red.

Simulating this scenario multiple times, gave varying results due to the randomness in the growth of the location error, i.e., the difference between the real position and the estimated position of the LAUV. In some cases the location error was even less than a metre over the entire survey task, resulting in good coverage for a simple lawnmower pattern. The Gazebo simulation environment uses a pseudorandom number generator to simulate the world, resulting in varying IMU behaviour per simulation. The basic lawnmower pattern might give good result when only small INS noise and bias is simulated. For the same reason the path of the planning system might fail to cover the entire area due to extreme biases on the simulated measurements.

Therefore, the simulation was repeated 100 times for both paths. Since the scenarios can only be simulated real-time, the number of simulations that could be conducted was limited. These simulations took about 200 hours, and give a better representation of how well the planning system behaves, rather than a single simulation. The average results are shown in Table 6-1, clearly showing that the planned path of the planning system achieves better coverage results than a simple lawnmower pattern.

Table 6-1: Comparison of the coverage of the simple lawnmower pattern and the path of the planning system for 100 simulations. It shows the average coverage percentage and average operation time over the simulations. The last column gives the number of simulation at which at least 99.5% coverage is achieved.

	Coverage [%]	Operation time[s]	Full coverage
Lawnmower pattern	96.0	1790.63	12
Planning system	99.6	2458.02	82

The planning system was not able to achieve full coverage for each simulation. This might be caused by a possible discrepancy between the uncertainty growth of the utilized uncertainty model and the real uncertainty growth in the simulations. It turned out that each time the

planning system failed to cover the entire area, this was caused by a large along-track position error which was not sufficiently compensated. As was already discussed, the coverage is not computed entirely correctly, often making the coverage per survey leg a bit too short.

On the other hand, the plans of the planning system never dropped below 96% coverage, while the basic lawnmower pattern failed to achieve 96% coverage 36 times. This means that the planning system is less sensible to unexpected large errors.

Thus, these simulation significantly show that the planning system improves the coverage and is able to find suitable plans for different scenarios. However, the simulation environment might need some improvements to give even better insights on the capabilities of the planning system.

6-2 Domain Validation

The previously shown scenarios are good examples of situations where the planning system behaves as expected. This is however not a guarantee that the PDDL domain description of the survey problem is valid for every possible scenario. Therefore, it is important to validate the PDDL description on whether the plans are valid and whether the domain describes the survey problem properly.

An automatic plan validation tool called VAL, was developed during the third IPC, to test *a posteriori* whether a plan is valid [41]. A plan is valid when, at every time instance, the planning states do not violate any constraint. This is done by simulating the actions and checking them on validity. VAL is also included in the ROSPlan framework, such that one can be sure that the resulting plans are valid within the specified constraints of the planning domain.

Although this is certainly useful, this does not say anything about the planning domain itself. When the planning domain descriptions contains errors, which can easily happen, the plans will be erroneous as well. Hence, an *a priori* validation of the planning domain is needed to determine the validity of the domain itself. This form of validation is not an often treated subject and, to the author's knowledge, has only been attempted by [42], by means of the modelling language Event-B.

6-2-1 Event-B

Event-B is a formal modelling language based on set theory. The purpose of the language is to model software systems in such a way that they are correct by construction [43]. This means that modelling a system in Event-B makes it possible to prove whether a system is correct, i.e., whether the system by definition cannot violate any pre-stated constraints. Since the language uses set theory, these strong properties can be proven mathematically.

Although Event-B is not a planning language, it can be used to check the correctness of the PDDL-model. In [42] a conversion from PDDL to Event-B is proposed, using a custom tool called `PDDL2EventB` . Since this tool was not accessible, the PDDL domain of the survey problem is translated to Event-B manually in the same fashion as was done in [42]. This entire Event-B description, together with a simple problem is shown in Appendix C.

As the name suggests, Event-B is an event-based language. These events can change the states of the system and are similar to PDDL-actions. Thus, the PDDL-actions of the PDDL domain can easily be translated to Event-B events. Figure 6-7 shows how the `survey` action is translated into Event-B.

```
survey: not extended ordinary ›
ANY
◦   from     ›
◦   to   ›
WHERE
◦   grd1:   from ∈ Waypoints not theorem ›
◦   grd2:   to ∈ Waypoints not theorem ›
◦   grd3:   to ≠ from not theorem ›
◦   grd4:   auv_at(from) = TRUE not theorem ›
◦   grd5:   can_survey = TRUE not theorem ›
◦   grd6:   is_survey(from ↦ to) = TRUE  not theorem ›
◦   grd7:   sonar_range > 2*total_uncertainty not theorem ›
◦   grd8:   total_data + survey_data < data_threshold not theorem ›
THEN
◦   act1:   auv_at ≔ auv_at ⩤ {from ↦ FALSE, to ↦ TRUE} ›
◦   act2:   is_survey ≔ is_survey ⩤ {from ↦ to ↦ FALSE, to ↦ from ↦ FALSE} ›
◦   act3:   can_transit ≔ TRUE ›
◦   act4:   can_survey ≔ FALSE ›
◦   act5:   total_time ≔ total_time + (leg_length + 2*total_uncertainty)÷survey_speed ›
◦   act6:   total_uncertainty ≔ total_uncertainty + uncertainty_survey ›
◦   act7:   total_data ≔ total_data + survey_data ›
◦   act8:   total_width ≔ total_width + sonar_range − total_uncertainty ›
END
```

Figure 6-7: This figure shows the survey action, modelled as an Event-B event[1]. The parameters of the event are listed under the `ANY` keyword. The guards listed under the keyword `WHERE` represent the preconditions of the event and the effects are listed as actions under the keyword `THEN`.

First the parameters of the event need to be declared, which are the two waypoints `from` and `to`. Then, guards are defined, which are similar to preconditions, except that some extra guards are required to ensure that (i) the provided waypoints are not equal and (ii) the waypoints are `waypoint` objects. Notice that sets can contain relations, such as `from` $\mapsto$ `to` $\mapsto$ `TRUE`. These relations can be checked and changed (such as in `grd6`), making it possible to model predicates and functions with parameters as well. The effects of a PDDL-action can be modelled as actions in an Event-B event. The actions of the survey event are similar to the effects of the survey action in the PDDL domain, as shown in Figure 6-7.

A powerful way to validate a model in Event-B, is by the use of *invariants*. Invariants are statements that need to be true in the entire reachable space of the system, i.e., invariants should not be violated in any possible state of the system. To proof this, it is first checked whether the initial states satisfy all the invariants. Then, for each event, it is checked whether all invariants stay true after the event, assuming that all invariants were satisfied before the event.

Most invariants come by definition of the types of each variable, e.g., the variable `can_survey` should always have a boolean value. Besides that, using Event-B, one can check additional invariants which are assumed to be true, but not explicitly modelled in the planning domain.

[1]The code snippets of Event-B are screen captures of the Rodin platform, which is free modelling software for Event-B. It is freely available at `www.event-b.org`.

Figure 6-8 shows two invariants that can be checked: `inv1` states that the AUV is always at a waypoint and `inv2` ensures that the data threshold is never exceeded.

```
INVARIANTS
    inv1:   auv_at \ (Waypoints × {FALSE}) ≠ ø not theorem ›
    inv2:   total_data < data_threshold not theorem ›
```

Figure 6-8: Two invariants in Event-B

Event-B comes along with a powerful automatic tool called the ProB model checker, which can mathematically prove whether any invariant is violated. This way, the model can easily be checked on modelling errors. For the survey domain there were 40 proofs needed to prove all invariants (see Appendix C to see a list of all the defined invariants). The only invariant that could not be proven automatically was `total_time` ≥ 0, due to the division by `survey_speed` (see Figure 6-7). This was simply solved by adding the invariant `survey_speed` > 0 such that a division by zero is excluded. It can be stated that the PDDL domain described in Chapter 4 is proven to be correct, assuming that the invariants of Figure 6-8 fully describe correctness. Checking these invariants gives the guarantee that it is not possible for the PDDL-solver to violate them, given that the initial states do not violate any invariant.

In PDDL, the initial states are defined in a problem file. Such a PDDL problem can be defined as a *refinement* of the domain model in Event-B[2]. This refinement is a separate Event-B file that defines all initial predicates and functions, as well as the goal. For each problem it needs to be proven that the initial predicates and functions do not violate the invariants. If that is the case, it is not possible to violate any invariant during planning.

Besides checking the invariants of an Event-B model, the ProB model checker is able to identify whether a model is deadlock-free. A deadlock is a situation where no event can happen, because the guards of all events are false. Therefore, the goal is defined as shown in Figure 6-9, which is an event as well. The guards of this event are the goals that need to be achieved, but it does not have any actions, i.e., it has no effects. This means that this event can be repeated indefinitely as long as the goal conditions hold.

```
goal:   not extended ordinary ›
WHERE
    grd1:   total_width ≥ area_width not theorem ›
    grd2:   total_data = 0 not theorem ›
END
```

Figure 6-9: The goal event in Event-B, where the guards correspond to the goals that are defined in the PDDL domain

Having defined the goal as such, means that reaching the goal state will not result in a deadlock situation. This makes it possible to detect other deadlocks in the domain model. Avoiding deadlocks, i.e., states from which the goal state never can be reached, will improve the efficiency of the PDDL-solver as this reduces its search space. The ProB model checker was able to find a deadlock in the planning domain, as shown in Figure 6-10. When the AUV decides to return to a waypoint where no survey action can be executed, the system is in a deadlock since it is not allowed to do two transits directly after each other.

[2]The reader is referred to [42] and [43] to obtain a better understanding of the Event-B language.

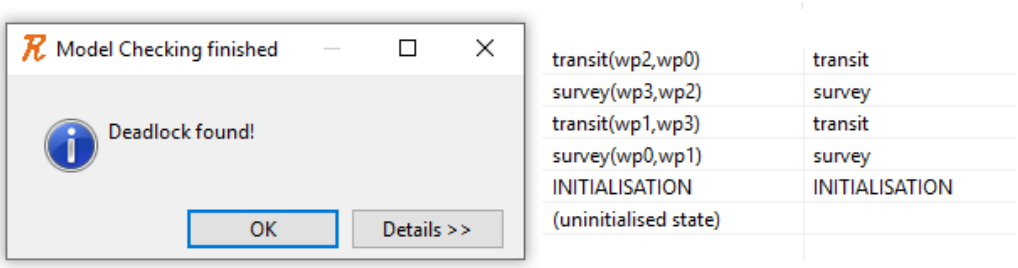


Figure 6-10: A deadlock situation was found by the ProB model checker.

Excluding these deadlock situations beforehand will improve the efficiency of the solver significantly. Table 6-2 shows the decrease in computation time of the OPTIC solver, when excluding the deadlocks from the PDDL domain. To achieve this, a simple predicate `(has_survey ?wp)` was added to specify whether a survey action is possible from that waypoint. This is initially true for each waypoint, and becomes false after a survey action has taken place. Each transit action will check whether its `?to` waypoint has still the possibility to survey from. For most problems, this small addition approximately halved the computation time, which is a significant improvement.

Table 6-2: Comparison of the computation time with and without the deadlock present in the planning domain, for 7 problems (same as Table 4-1). Problem 4 (where only a `transit_comm` was allowed) was not solvable using the deadlock-free PDDL domain. All values are in seconds.

	1	2	3	4	5	6	7
With deadlock	0.80	6.32	29.82	14.82	74.83	29.76	553.21
Without deadlock	0.34	3.34	13.10	-	34.54	17.09	226.20

The way the planning domain is formulated even implies that if the system is deadlock-free, the problem has a valid solution. This gives *a priori* information of the problem being solvable. However, since the translation to Event-B is not automated, this is an elaborate task. Besides that, Event-B is limited with numeric expressions as well, as it only supports integer values.

In summary, the Event-B method is a powerful way to validate the PDDL domain. However, correctness is defined by the invariants that are chosen. For example, in Section 6-1 it was discovered that when the uncertainty grows too large, the AUV might miss a communication area entirely, when performing a `transit_comm` action. To find such a modelling error, an invariant should be added to exclude these situations. Since this was not defined as an invariant in Event-B, this modelling error was not found. Nonetheless, excluding deadlocks from the planning model using the Event-B method did result in a significant reduction of the computation time for solving the problems with a PDDL-solver.

Chapter 7

Conclusion

7-1 Summary

A task planning system was developed for AUVs in survey missions, successfully covering designated areas by planning a path for the AUV to follow. The planning system is based on the generic planning language PDDL, to be able to plan higher level actions more conveniently than domain specific planners. The planning system is able to make decisions regarding location uncertainty and limited communication, while minimizing the operation time. This fulfils the main objective of this thesis work.

To achieve this main objective, first a simulation environment was set up, to smoothly integrate with the software architecture of the LAUV. The *UUV Simulator* has already provided an underwater environment with underwater physics and AUV models. Two important things were added: an INS for pose estimation and a position controller to successfully follow the paths with the vehicle. This facilitated the evaluation of the planning system, in order to examine the behaviour of the LAUV in the underwater environment when executing the plans.

Then, the planning problem was modelled in PDDL. The planning system needs to alternate between two types of actions: a survey action and a transit action. The survey action is the main type of action. This is where the AUV travels in a straight path, covering a part of the survey area. The transit action enable the AUV to travel between the survey legs in four ways: (i) without any special action, (ii) while doing a GPS-fix to get a location update, (iii) while revisiting a specified area where SLAM can be applied and (iv) while travelling through a communication area, transferring data to other vehicles.

Several problems were solved with four different PDDL-solvers, where OPTIC consequently achieved the highest plan quality, i.e., the lowest operation time. Therefore, OPTIC is chosen to be the PDDL-solver of the planning system. As a drawback, it takes the largest amount of computation time, since in order to improve the plan quality, it repeats the planning procedure several times. By translating the planning domain to Event-B, several deadlock situations were removed, which halved the computation time. This shows that the way the planning domain is formulated can significantly affect the computation time of the PDDL-solver.

The planning system consists of three planners: a Task Planner, a Problem Generator and a Path Planner, where the PDDL-solver is part of the Task Planner. PDDL turned out to be limited regarding numerical expressions. Therefore, complex equations needed to be excluded from the planning domain and were solved inside the Problem Generator. The Problem Generator consists of five sub-planners, each providing the required numerical values for a PDDL-action. Through iteration, a final plan is found, such that a path can be planned by the Path Planner. This path, in turn, can be dispatched so that the path can be simulated, or executed by the real vehicle.

For several scenarios the paths generated by the planning system were evaluated. The planning system is capable of planning all different actions in different situations, while taking into account the operation time. Some inefficiencies might occur due to the restrictions in the PDDL domain. More freedom in the PDDL domain might result in better plans, but will make the planning procedure much more computationally expensive.

A plan for a larger survey area was compared with a standard lawnmower pattern by means of simulation, which showed that the planning system is able to consistently achieve better coverage. The coverage is estimated using the spatial information of the simulation environment, giving an approximation of the real coverage. It would certainly be valuable to simulate sonar data to asses coverage in a better way.

7-2 Contribution

Using the results of this thesis, the research questions, as stated in Chapter 1, can be answered. This addresses the new insights that are obtained during this project. The answers to the research questions are listed below:

i. ***How can the complex survey problem be modelled into PDDL, and what assumptions need to be made to achieve this?***

   The results show that it is feasible to model the survey problem in PDDL with sufficient detail, to be able to generate valid problems for different scenarios. The main challenge was to model location uncertainty using the deterministic PDDL language.

   The first attempt was to model the survey problem in full detail, only using PDDL. It soon turned out that this resulted in an explosion of the search space, making it impossible for PDDL-solvers to solve the problem in reasonable time. Therefore, details needed to be extracted from the planning domain, such that PDDL only describes the abstract actions that can be planned, making the solving procedure much more efficient.

   Modelling the survey problem was a constant trade-off between detail and efficiency, because due to the limited numerical capabilities, several simplifications and assumptions needed to be made. The main simplification is the linearisation of the uncertainty growth and the assumption that the uncertainty growth is equal in all directions. Besides that, it was assumed that the sonar performance is constant over the entire area. These assumptions might have consequences on the real coverage when executing the plan.

ii. ***What are the benefits of using a more generic planning system compared to a domain specific planner?***

The main motivation to use a more generic planning system is that it can efficiently handle combinatorial problems and plan abstract actions, which was due to the complexity of the problem a necessity. The usage of PDDL gives the opportunity to use state-of-the-art solvers that are able to find valid plans efficiently through heuristic search.

Besides that, PDDL offers the flexibility to solve different types of problems, using the same planning domain. The planning system is constructed in such a way that it is modular and that it can be used in multiple ways. This thesis shows that a PDDL-based planning system can fluently switch between different scenarios with different requirements. In a broader perspective, the planning system might even be applied for different problems as well, such as the *Identification* task of an MCM operation.

A domain specific planner is more confined for the specific application and does not offer the flexibility a more generic planning system has. On the other hand, a domain specific planner might be able to describe the planning domain more accurately, since when describing problems, PDDL has its limitations. This limitation of PDDL however does force to concisely formulate the planning problem, making it more insightful and well organized.

iii. ***How can the PDDL model be validated, to ensure that it is correct and brings up valid plans in every possible scenario?***

Validation of PDDL domain descriptions is rarely discussed in literature. Mostly, planning systems are evaluated through simulation, showing improvements with respect to previous results. The planning system was evaluated in the same fashion, showing that it indeed improves the coverage and that it is able to find plans in predefined scenarios.

The validation tool called VAL is developed to check whether the generated plans satisfy the constraints of the planning problem at every time instance. However, this does not give an *a priori* validation of the PDDL domain. By translating the PDDL domain to the Event-B modelling language, the planning domain can be validated through mathematical proofs. One can define statements that need to be invariably true for every possible situation and check whether the planning domain suffices. In Event-B, one can also check whether situations might occur where no actions are possible, i.e., whether there are deadlock situations. Preventing these situations improves the efficiency of the planning system, since this confines the search space for the PDDL-solver. Applying this to the survey problem has improved the planning domain.

This thesis work is a step forward in the direction of achieving persistent autonomy for AUVs. It is shown that generic planning methods, such as PDDL, can be used to make decisions about high level actions without omitting important details of the planning problem. In this thesis the focus was on location uncertainty, but environmental properties can also be taken into account, considering the way that the planning system is constructed.

Besides that, this thesis extends the work of [1] and [21] in using PDDL for real world problems, rather than fictional benchmark problems. It is an attempt to bridge the gap between abstract planning and real-world complex problems, making use of abstract and state-of-the-art planning methods for relevant applications. Validation methods, such as the Event-B method, will become more meaningful as well, when it applies to real-world problems.

7-3 Recommendations

The developed planning system is able to show the capabilities of the proposed planning method and is able to plan a path for several interesting scenarios. In order to increase the robustness, accuracy and applicability of the planning system, it is certainly recommended to improve the different planners of the planning system. A few suggestions for improvement are:

- First of all, the Problem Generator could make better use of SPMs. As explained in Chapter 3, the sonar range is assumed to be constant. In environments with complex seafloor structures, the planning system will fail to cover the entire area due to this assumption. When the seafloor structure is known *a priori*, one might split up the area, based on the seafloor type and define a constant sonar range per area. Another approach might be to include this seafloor structure in the survey planner (which is a sub-planner of the Problem Generator), to adjust the leg spacing according to this seafloor type. However, this implies that the PDDL domain needs to be adjusted accordingly.
- The revisit planner currently serves as a place-holder rather than providing realistic numeric values. A proper SLAM model needs to be implemented in order to predict the location uncertainty after transiting via a revisit area.
- The survey planner currently ignores the negative effects of the Nadir gap on coverage. Considering different patterns such as paired-tracks would be a valuable addition.
- More detail, such as battery level, might be worth adding to the planning domain.
- The Path Planner currently assumes that the entire area is free of obstacles. Especially when multiple vehicles are involved, it might be necessary to introduce no-go areas to prevent collisions. In that case, the Path Planner needs to be able to plan paths that avoid traversing these areas.

Besides improving the planning system, the following topics are suggested for future research:

- The planning system still provides offline plans before executing the survey task. It would be interesting to make the planning system adaptive if during the execution of a plan unexpected things happen. For instance, when the sonar range turns out to be lower than expected (due to environmental properties), or when the location error turns out to be larger than was predicted after an uncertainty reducing action.
- Instead of trying to cover the entire area in the least amount of time, it might give insightful results to try to maximize its coverage within a specified time limit.
- The PDDL domain can be extended such that it can be used for the *Identification* phase of an MCM operation as well. This would demonstrate the power and flexibility of PDDL.
- Finally, it might be worth to explore more methods to validate the PDDL domain. An interesting method would be to translate the planning domain into a Mixed Integer Linear Programming (MILP) problem, trying to solve the survey problem analytically.

Appendix A

Pose Estimation

Pose estimation is needed for position control. The vehicle (the LAUV in this case) needs to know its position with respect to the world in order to successfully execute its tasks and move between waypoints at specified locations. Since AUVs have no access to GPS signals underwater, the vehicle has no absolute position measurement.

Hence, several different sensor measurements need to be combined to estimate the position as good as possible. The sensors available on the LAUV are: (a) an Inertial Measurement Unit (IMU) measuring all accelerations and the orientation of the vehicle, (b) a pressure sensor measuring the depth of the vehicle and (c) a Doppler Velocity Logger (DVL) measuring the velocity of the vehicle with respect to the seafloor. The data of these sensors can be fused, using a Kalman filter, which is called sensor fusion in literature. To do so, the following state-space system can be defined:

$$q_{k+1} = Aq_k + Bw_k \tag{A-1a}$$
$$m_k = Cq_k + v_k \tag{A-1b}$$

Here, q_k is the state vector at time-step k and comprises the position, velocity, acceleration, orientation and angular rate of the vehicle respectively:

$$q_k = \begin{bmatrix} x_k & y_k & z_k & \dot{x}_k & \dot{y}_k & \dot{z}_k & \ddot{x}_k & \ddot{y}_k & \ddot{z}_k & r_k & p_k & p_k & \gamma_k & \dot{r}_k & \dot{p}_k & \dot{\gamma}_k \end{bmatrix}^T \tag{A-2}$$

Here x, y and z are the cartesian coordinates in the world frame, and r, p and γ is the roll, pitch and yaw respectively of the vehicle in world frame as well. The vector m_k in (A-1b) is the measurement vector, which reads as:

$$\begin{aligned} m_k = \Big[& \ddot{x}_{IMU,k} \quad \ddot{y}_{IMU,k} \quad \ddot{z}_{IMU,k} \quad r_{IMU,k} \quad p_{IMU,k} \quad \gamma_{IMU,k} \quad z_{press,k} \\ & x_{GPS,k} \quad y_{GPS,k} \quad z_{GPS,k} \quad \dot{x}_{DVL,k} \quad \dot{y}_{DVL,k} \quad \dot{z}_{DVL,k} \Big]^T \\ = \Big[& m_{IMU,k} \quad m_{press,k} \quad m_{GPS,k} \quad m_{DVL,k} \Big]^T \end{aligned} \tag{A-3}$$

Here $m_{IMU,k}$ represents the IMU measurements, $m_{press,k}$ represents the pressure sensor measurements, $m_{GPS,k}$ represents the GPS measurements and $m_{DLV,k}$ represents the DVL measurements. Notice that the GPS measurements are included here. The values of $m_{GPS,k}$ are only available if the vehicle is at the surface at time-step k, otherwise the size of m_k and C will be reduced. Another important thing to mention is, that all measurements are initially with respect to the body frame of the vehicle. Hence, a transformation is needed from body frame $\mathcal{B}$ to world frame $\mathcal{W}$:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^{\mathcal{W}} = J_1 \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{\mathcal{B}} \tag{A-4}$$

$$\begin{bmatrix} r \\ p \\ \gamma \end{bmatrix}^{\mathcal{W}} = J_2 \begin{bmatrix} r \\ p \\ \gamma \end{bmatrix}^{\mathcal{B}} \tag{A-5}$$

These transformation matrices J_1 and J_2 are respectively given by [44]:

$$J_1 = \begin{bmatrix} \cos\gamma\cos p & -\sin\gamma\cos r + \cos\gamma\sin p\sin r & \sin\gamma\sin r + \cos\gamma\sin p\cos r \\ \sin\gamma\cos p & \cos\gamma\cos r + \sin\gamma\sin p\sin r & -\cos\gamma\sin r + \sin\gamma\sin p\cos r \\ -\sin p & \cos p\sin r & \cos p\cos r \end{bmatrix} \tag{A-6}$$

$$J_2 = \begin{bmatrix} 1 & \sin r\tan p & \cos\gamma\tan p \\ 0 & \cos r & -\sin r \\ 0 & \sin r/\cos p & \cos r/\cos p \end{bmatrix} \tag{A-7}$$

Furthermore, in (A-1) two noise vectors w_k and v_k are defined, representing the process noise and measurement noise respectively. These noise signals are assumed to be Gaussian white noise. Due to the fact that the external forces on the body are not modelled in this state space representation, the process noise is the noise on each acceleration:

$$w_k = \begin{bmatrix} w_{\ddot{x},k} & w_{\ddot{y},k} & w_{\ddot{z},k} & w_{\ddot{r},k} & w_{\ddot{p},k} & w_{\ddot{\gamma},k} \end{bmatrix}^T \tag{A-8}$$

The measurement noise is simply the noise on each measurement signal individually:

$$v_k = \begin{bmatrix} v_{IMU,k} & v_{press,k} & v_{GPS,k} & v_{DVL,k} \end{bmatrix}^T \tag{A-9}$$

Finally, only the system matrices need to be defined. The transition matrix A simply integrates over the time-difference Δt to get the states of time-step $k+1$:

$$A = \begin{bmatrix} I_3 & I_3\Delta t & I_3\frac{\Delta t^2}{2} & 0_3 & 0_3 \\ 0_3 & I_3 & I_3\Delta t & 0_3 & 0_3 \\ 0_3 & 0_3 & I_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & I_3 & I_3\Delta t \\ 0_3 & 0_3 & 0_3 & 0_3 & I_3 \end{bmatrix} \tag{A-10}$$

The matrix B transforms the covariances of the accelerations w_k to the states q_k by integration as well:

$$B = \begin{bmatrix} I_3\frac{\Delta t^2}{2} & 0_3 \\ I_3\Delta t & 0_3 \\ I_3 & 0_3 \\ 0_3 & I_3\frac{\Delta t^2}{2} \\ 0_3 & I_3\Delta t \end{bmatrix} \tag{A-11}$$

The measurement matrix C simply maps the state values q_k to the measurement values m_k, and is defined as follows:

$$C = \begin{bmatrix} & 0_3 & & & 0_3 & & & I_3 & & & 0_3 & & & 0_3 & \\ & 0_3 & & & 0_3 & & & 0_3 & & & I_3 & & & 0_3 & \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & I_3 & & & 0_3 & & & 0_3 & & & 0_3 & & & 0_3 & \\ & 0_3 & & & I_3 & & & 0_3 & & & 0_3 & & & 0_3 & \end{bmatrix} \tag{A-12}$$

In order to construct a Kalman filter, two additional matrices are required: Q and R_k. Q_k is the covariance matrix of the process noise and R_k is the covariance matrix of the measurement noise. These matrices are simply the diagonal matrices of the covariances of the signals:

$$Q_k = \text{diag}\begin{pmatrix} \sigma_{\ddot{x},k} & \sigma_{\ddot{y},k} & \sigma_{\ddot{z},k} & \sigma_{\ddot{r},k} & \sigma_{\ddot{p},k} & \sigma_{\ddot{\gamma},k} \end{pmatrix} \tag{A-13}$$

$$R_k = \text{diag}\begin{pmatrix} \sigma_{IMU,k} & \sigma_{press,k} & \sigma_{GPS,k} & \sigma_{DVL,k} \end{pmatrix} \tag{A-14}$$

The covariances of the process noise are chosen to be relatively small numbers, order of magnitude 10^{-2}. Note that it is assumed that the covariance of the noise on the measurement signal equals the covariance of the noise on the transformed signal, which in reality is not true due to the covariances imposed by the orientation. The resulting covariance estimation might therefore be slightly optimistic.

Having the entire model, a conventional Kalman filter can be used to fuse the sensor information into one pose estimation $\hat{q}$. The Kalman filter process consist of a prediction step and an update step. In the prediction step, a first prediction of the state is made using the information of the previous state and the state-space model:

$$\hat{q}_{k|k-1} = A\hat{q}_{k-1|k-1} \tag{A-15}$$

$$P_{k|k-1} = AP_{k-1|k-1}A^T + BQ_kB^T \tag{A-16}$$

This initial prediction can be updated using the Kalman gain K, which can be computed by the following equation:

$$K_k = P_{k|k-1}A^T(R_k + CP_{k|k-1}C^T)^{-1} \tag{A-17}$$

The final pose estimation $\hat{q}_k$ and covariance matrix P_k are given by:

$$\hat{q}_{k|k} = \hat{q}_{k|k-1} + K_k(m_k - Cq_{k|\hat{k}-1}) \tag{A-18}$$

$$P_{k|k} = (I - K_k A)P_{k|k-1} \tag{A-19}$$

The estimated state q_k is used to control the AUV, and the covariance matrix P_k gives the uncertainty of the states and consequently the location uncertainty. These values are of main importance for controlling the AUV, and is essential for online planning. This Kalman filter, as described, was implemented into the simulation environment, and reads in the simulated sensor data. This way, the control of the LAUV is simulated in a more realistic fashion.

Appendix B

Problem and Domain Description

This appendix gives the full PDDL domain description of the survey task. Also an example problem file is given as an indication of what this looks like. This problem file corresponds to the combined problem of Figure 6-4a, although much of it is left out. The entire problem file uses 1115 lines of code and would fill over 20 pages.

B-1 Domain File

Listing B.1: PDDL domain file - AUV Survey

```
(define (domain auv_survey)
  (:requirements :strips :typing :fluents)

  (:types
    waypoint    ; Waypoints to move between
    area        ; Revisit or communication area
  )

  (:predicates
    (auv_at ?wp - waypoint)     ; The location of the AUV

    ; What the AUV is supposed to do at this point
    (can_survey)  ; It should follow a survey track
    (can_transit) ; It should do a transit-type action

    ; Whether it is permitted to survey or transit between waypoints
    (is_survey  ?from ?to - waypoint)
    (is_transit ?from ?to - waypoint)
    (is_gps     ?from ?to - waypoint)
    (is_revisit ?from ?to - waypoint ?via - area)
    (is_comm    ?from ?to - waypoint ?via - area)
  )
```

```
(:functions
  ; Planning variables
  (total_time)         ; Operation time (optimization metric) [s]
  (total_uncertainty) ; Total location uncertainty of the vehicle [m]
  (total_data)         ; Total data that needs to be communicated [byte]
  (total_width)        ; Total width that the vehicle has covered [m]

  ; Survey functions
  (sonar_range)        ; Range of the sonar (both sides) [m]
  (survey_speed)       ; Velocity the AUV is surveying [m/s]
  (area_width)         ; Width of the area that needs to be covered [m]
  (leg_length)         ; Length of one survey leg [m]

  ; Communication functions
  (bit_rate ?a - area)  ; Bit rate at a communication area [byte/s]
  (survey_data)         ; Data obtained after a survey leg [byte]
  (data_threshold)      ; Maximum total data that is permitted [byte]

  ; Time [s] and uncertainty growth [m] for surveying and each transit
     type
  (time_transit ?from ?to - waypoint)
  (time_gps     ?from ?to - waypoint)
  (time_revisit ?from ?to - waypoint ?via - area)
  (time_comm    ?from ?to - waypoint ?via - area)

  (uncertainty_survey)
  (uncertainty_transit  ?from ?to - waypoint)
  (uncertainty_gps      ?from ?to - waypoint)
  (uncertainty_revisit  ?from ?to - waypoint ?via - area)
  (uncertainty_comm     ?from ?to - waypoint ?via - area)
)


; ------------------------------------
;          Survey action
; ------------------------------------

; Travel a survey-leg between two waypoints
(:action survey
  :parameters (?from ?to - waypoint)
  :precondition (and
      (auv_at ?from)
      (can_survey)
      (is_survey ?from ?to)
      ; The uncertainty should not exceed the sonar range
      (> (sonar_range) (total_uncertainty) )
      ; The data threshold should not be exeeded
      (< (+ (total_data) (survey_data)) (data_threshold) )
    )
  :effect (and
      ; Change AUV position to the new waypoint
      (not (auv_at ?from) )
```

```
        (auv_at ?to)
        ; Prevent doing a survey task twice
        (not (is_survey ?from ?to) )
        (not (is_survey ?to ?from) )
        ; After the survey a transit needs to be done
        (can_transit)
        (not (can_survey))
        ; Increase the total time by adding the time for surveying
            between the waypoints, taking into account the along-track
            uncertainty: total_time += (l+2u)/v_s
        (increase (total_time) (/ (+ (leg_length) (* 2 (total_uncertainty
            ))) (survey_speed)) )
        ; Increase the total uncertainty
        (increase (total_uncertainty) (uncertainty_survey) )
        ; Increase the accumulated data
        (increase (total_data) (survey_data) )
        ; Increase the covered width, taking into account the across-
            track uncertainty: total_width += r - u
        (increase (total_width) (- (sonar_range) (total_uncertainty)) )
    )
)


; ---------------------------------
;          Transit actions
; ---------------------------------

; Transit between two waypoints
(:action transit
  :parameters (?from ?to - waypoint)
  :precondition (and
      (auv_at ?from)
      (can_transit)
      (is_transit ?from ?to)
    )
  :effect (and
      ; Change AUV position to the new waypoint
      (not (auv_at ?from) )
      (auv_at ?to)
      ; After the transit a survey needs to be done
      (not (can_transit))
      (can_survey)
      ; Increase the total time by the transit duration
      (increase (total_time) (time_transit ?from ?to) )
      ; Increase the total uncertainty
      (increase (total_uncertainty) (uncertainty_transit ?from ?to) )
    )
)

; Transit between two waypoints and perform a GPS-fix
(:action transit_gps
  :parameters (?from ?to - waypoint)
  :precondition (and
```

```
        (auv_at ?from)
        (can_transit)
        (is_gps ?from ?to)
      )
    :effect (and
        ; Change AUV position to the new waypoint
        (not (auv_at ?from) )
        (auv_at ?to)
        ; After the transit a survey needs to be done
        (not (can_transit))
        (can_survey)
        ; Increase the total time by adding the time for travelling
           between the waypoints and performing a GPS-fix
        (increase (total_time) (time_gps ?from ?to) )
        ; Reset the total uncertainty
        (assign (total_uncertainty) (uncertainty_gps ?from ?to) )
        ; Reset the accumulated data
        (assign (total_data) 0)
      )
  )

  ; Transit between two waypoints and perform a revisit
  (:action transit_revisit
    :parameters (?from ?to - waypoint ?via - area)
    :precondition (and
        (auv_at ?from)
        (can_transit)
        (is_revisit ?from ?to ?via)
      )
    :effect (and
        ; Change AUV position to the new waypoint
        (not (auv_at ?from) )
        (auv_at ?to)
        ; After the transit a survey needs to be done
        (not (can_transit))
        (can_survey)
        ; Increase the total time by adding the time for travelling
           between the waypoints and revisiting an area
        (increase (total_time) (time_revisit ?from ?to ?via) )
        ; Reset the total uncertainty
        (assign (total_uncertainty) (uncertainty_revisit ?from ?to ?via)
           )
      )
  )

  ; Transit between two waypoints and communcate data
  (:action transit_comm
    :parameters (?from ?to - waypoint ?via - area)
    :precondition (and
        (auv_at ?from)
        (can_transit)
        (is_comm ?from ?to ?via)
      )
```

```
    :effect (and
        ; Change AUV position to the new waypoint
        (not (auv_at ?from) )
        (auv_at ?to)
        ; After the transit a survey needs to be done
        (not (can_transit))
        (can_survey)
        ; Increase the total time by adding the time for travelling
           between the waypoints and communcating at a communication area
           :
        ; t += t_comm + d/b_r
        (increase (total_time) (+ (time_comm ?from ?to ?via) (/ (
           total_data) (bit_rate ?via))) )
        ; Increase the total uncertainty
        (increase (total_uncertainty) (uncertainty_comm ?from ?to ?via) )
        ; Reset the accumulated data variable
        (assign (total_data) 0)
      )
  )
)
```

B-2 Problem File Example

Listing B.2: Example PDDL problem file - AUV Survey (partially)

```
(define (problem survey_task)

(:domain auv_survey)

(:objects
  start wp0 wp1 wp2 wp3 wp4 wp5 wp6 wp7 wp8 wp9 - waypoint
  r0 r1 c0 - area
)

(:init
  ; Initialize AUV location
  (auv_at start)
  (can_transit)

  ; Survey and transit connections (partially)
  (is_survey wp0 wp1) (is_survey wp1 wp0) ; et cetera
  (is_transit wp0 wp2) (is_transit wp2 wp0) ; et cetera
  (is_gps wp0 wp2) (is_gps wp2 wp0) (is_gps wp1 wp3) ; et cetera
  (is_revisit wp0 wp1 r0) (is_revisit wp1 wp0 r0) ; et cetera
  (is_comm start wp0 c0) (is_comm wp0 start c0) ; et cetera

  ; Planning variables
  (= (total_time) 0)          ; [s]
  (= (total_uncertainty) 0)   ; [m]
  (= (total_data) 0)          ; [byte]
  (= (total_width) 0)         ; [m]
```

```
  ; Survey functions
  (= (sonar_range) 100)        ; [m]
  (= (survey_speed) 1.2)       ; [m/s]
  (= (area_width) 310.79)      ; [m]
  (= (leg_length) 356.27)      ; [m]

  ; Communication functions
  (= (bit_rate c0) 10)         ; [byte/s]
  (= (survey_data) 31.08)      ; [byte]
  (= (data_threshold) 100)     ; [byte]

  ; Time for transit actions [s]
  (= (time_transit wp7 wp9) 37.34) (= (time_transit start wp1) 289.5)
  (= (time_transit start wp0) 47.22) ; et cetera
  ; Time for GPS actions [s]
  (= (time_gps wp9 wp7) 113.05) (= (time_gps wp0 start) 122.93)
  (= (time_gps start wp1) 303.79) ; et cetera
  ; Time for revisit actions [s]
  (= (time_revisit wp8 start r0) 90.35) (= (time_revisit wp4 wp8 r0)
   79.27) (= (time_revisit wp0 wp1 r0) 197.74) ; et cetera
  ; Time for communication actions [s]
  (= (time_comm wp2 wp0 c0) 217.24) (= (time_comm wp0 wp3 c0) 419.62)
  (= (time_comm wp3 wp0 c0) 419.62) ; et cetera

  ; Uncertainty growth of one survey leg
  (= (uncertainty_survey) 6.31)
  ; Uncertainty growth for transit actions [m]
  (= (uncertainty_transit wp9 wp7) 0.95)
  (= (uncertainty_transit start wp0) 1.2) ; et cetera
  ; Uncertainty growth for GPS actions [m]
  (= (uncertainty_gps wp9 wp7) 0.34)
  (= (uncertainty_gps start wp1) 1.36) ; et cetera
  ; Uncertainty growth for revisit actions [m]
  (= (uncertainty_revisit wp9 start r0) 0.82)
  (= (uncertainty_revisit wp0 wp1 r0) 3.68) ; et cetera
  ; Uncertainty growth for communiation actions [m]
  (= (uncertainty_comm wp1 wp0 c0) 11.01)
  (= (uncertainty_comm wp0 wp2 c0) 5.54) ; et cetera
)

(:goal (and
    ; Cover the entire area
    (>= (total_width) (area_width))
    ; Communicate all data
    (<= (total_data) 0)
  )
)

(:metric
  ; Minimize operation time
  minimize (total_time))
)
```

Appendix C

Event-B Model

This appendix gives the full Event-B model of the PDDL domain, and a simple problem. This entails two *Contexts* and two *Machines*. The first Context simply represent the types in the PDDL domain, and the first Machine describes all predicates, functions and actions of the PDDL domain.

The second Context declares all objects in a problem and the second Machine initializes all predicates and functions and states a goal 'action' as well. To fully understand what this notation means, the reader is referred to [42] and [43].

```
CONTEXT
    C_Survey
SETS
    Waypoints
    Areas
END
```

Figure C-1: Event-B Context of the PDDL domain

```
CONTEXT
    C_Survey1
EXTENDS
    C_Survey
CONSTANTS
    wp0
    wp1
    wp2
    wp3
    wp4
    wp5
AXIOMS
    axm1:   partition(Waypoints, {wp0}, {wp1}, {wp2}, {wp3}, {wp4}, {wp5}) not theorem
END
```

Figure C-2: Event-B Context of a simple PDDL problem

```
MACHINE
    M_Survey
SEES
    C_Survey
VARIABLES
    auv_at
    can_survey
    can_transit
    is_survey
    is_transit
    is_gps
    is_revisit
    is_comm
    total_time
    total_uncertainty
    total_data
    total_width
    sonar_range
    survey_speed
    area_width
    bit_rate
    survey_data
    data_threshold
    leg_length
    uncertainty_survey
    time_transit
    time_gps
    time_revisit
    time_comm
    uncertainty_transit
    uncertainty_gps
    uncertainty_revisit
    uncertainty_comm
INVARIANTS
    inv_auv_at: auv_at ∈ Waypoints ⟶ BOOL not theorem
    inv_can_survey: can_survey ∈ BOOL not theorem
    inv_can_transit:    can_transit ∈ BOOL not theorem
    inv_is_survey:  is_survey ∈ Waypoints × Waypoints ⟶ BOOL not theorem
    inv_is_transit: is_transit ∈ Waypoints × Waypoints ⟶ BOOL not theorem
    inv_is_gps: is_gps ∈ Waypoints × Waypoints ⟶ BOOL not theorem
    inv_is_revisit: is_revisit ∈ Waypoints × Waypoints × Areas ⟶ BOOL not theorem
    inv_is_comm:    is_comm ∈ Waypoints × Waypoints × Areas ⟶ BOOL not theorem
    inv_total_time: total_time ∈ N not theorem
    inv_total_uncertainty:  total_uncertainty ∈ N not theorem
    inv_total_data: total_data ∈ N not theorem
    inv_total_width:    total_width ∈ N not theorem
    inv_sonar_range:    sonar_range ∈ N not theorem
    inv_survey_speed:   survey_speed ∈ N1 not theorem
    inv_area_width: area_width ∈ N1 not theorem
    inv_bit_rate:   bit_rate ∈ Areas ⟶ N not theorem
    inv_survey_data:    survey_data ∈ N not theorem
    inv_data_threshold: data_threshold ∈ N not theorem
    inv_leg_length: leg_length ∈ N1 not theorem
    inv_time_transit:   time_transit ∈ Waypoints × Waypoints ⟶ N not theorem
    inv_time_gps:   time_gps ∈ Waypoints × Waypoints ⟶ N not theorem
    inv_time_revisit:   time_revisit ∈ Waypoints × Waypoints × Areas ⟶ N not theorem
    inv_time_comm:  time_comm ∈ Waypoints × Waypoints × Areas ⟶ N not theorem
    inv_uncertainty_survey: uncertainty_survey ∈ N1 not theorem
    inv_uncertainty_transit:    uncertainty_transit ∈ Waypoints × Waypoints ⟶ N not theorem
    inv_uncertainty_gps:    uncertainty_gps ∈ Waypoints × Waypoints ⟶ N not theorem
    inv_uncertainty_revisit:    uncertainty_revisit ∈ Waypoints × Waypoints × Areas ⟶ N not theorem
    inv_uncertainty_comm:   uncertainty_comm ∈ Waypoints × Waypoints × Areas ⟶ N not theorem
```

Figure C-3: Event-B Machine of the PDDL domain, showing all variables and invariants (part I)

```
EVENTS
  INITIALISATION:    not extended ordinary ›
  END

  survey: not extended ordinary ›
  ANY
    from    ›
    to    ›
  WHERE
    grd1:   from ∈ Waypoints not theorem ›
    grd2:   to ∈ Waypoints not theorem ›
    grd3:   to ≠ from not theorem ›
    grd4:   auv_at(from) = TRUE not theorem ›
    grd5:   can_survey = TRUE not theorem ›
    grd6:   is_survey(from ↦ to) = TRUE  not theorem ›
    grd7:   sonar_range > 2*total_uncertainty not theorem ›
    grd8:   total_data + survey_data < data_threshold not theorem ›
  THEN
    act1:   auv_at ≔ auv_at ⩤ {from ↦ FALSE, to ↦ TRUE} ›
    act2:   is_survey ≔ is_survey ⩤ {from ↦ to ↦ FALSE, to ↦ from ↦ FALSE} ›
    act3:   can_transit ≔ TRUE ›
    act4:   can_survey ≔ FALSE ›
    act5:   total_time ≔ total_time + (leg_length + 2*total_uncertainty)÷survey_speed ›
    act6:   total_uncertainty ≔ total_uncertainty + uncertainty_survey ›
    act7:   total_data ≔ total_data + survey_data ›
    act8:   total_width ≔ total_width + sonar_range − total_uncertainty ›
  END

  transit:    not extended ordinary ›
  ANY
    from    ›
    to    ›
  WHERE
    grd1:   from ∈ Waypoints not theorem ›
    grd2:   to ∈ Waypoints not theorem ›
    grd3:   auv_at(from) = TRUE not theorem ›
    grd4:   to ≠ from not theorem ›
    grd5:   can_transit = TRUE not theorem ›
    grd6:   is_transit(from ↦ to) = TRUE not theorem ›
    grd7:   is_survey ∩ ({to} × Waypoints × {TRUE}) ≠ ∅ not theorem ›
  THEN
    act1:   auv_at ≔ auv_at ⩤ {from ↦ FALSE, to ↦ TRUE} ›
    act2:   can_transit ≔ FALSE ›
    act3:   can_survey ≔ TRUE ›
    act4:   total_time ≔ total_time + time_transit(from ↦ to) ›
    act5:   total_uncertainty ≔ total_uncertainty + uncertainty_transit(from ↦ to) ›
  END

  transit_gps:    not extended ordinary ›
  ANY
    from    ›
    to    ›
  WHERE
    grd1:   from ∈ Waypoints not theorem ›
    grd2:   to ∈ Waypoints not theorem ›
    grd3:   to ≠ from not theorem ›
    grd4:   auv_at(from) = TRUE not theorem ›
    grd5:   can_transit = TRUE not theorem ›
    grd6:   is_gps(from ↦ to) = TRUE not theorem ›
    grd7:   is_survey ∩ ({to} × Waypoints × {TRUE}) ≠ ∅ not theorem ›
  THEN
    act1:   auv_at ≔ auv_at ⩤ {from ↦ FALSE, to ↦ TRUE} ›
    act2:   can_transit ≔ FALSE ›
    act3:   can_survey ≔ TRUE ›
    act4:   total_time ≔ total_time + time_gps(from ↦ to) ›
    act5:   total_uncertainty ≔ uncertainty_gps(from ↦ to) ›
    act6:   total_data ≔ 0 ›
  END
```

Figure C-4: Event-B Machine of the PDDL domain, showing the `survey`, `transit` and `transit_gps` event (part II)

```
transit_revisit:   not extended ordinary ›
ANY
   from     ›
   to   ›
   via   ›
WHERE
   grd1:   from ∈ Waypoints not theorem ›
   grd2:   to ∈ Waypoints not theorem ›
   grd3:   to ≠ from not theorem ›
   grd4:   via ∈ Areas not theorem ›
   grd5:   auv_at(from) = TRUE not theorem ›
   grd6:   can_transit = TRUE not theorem ›
   grd7:   is_revisit(from ↦ to ↦ via) = TRUE not theorem ›
   grd8:   is_survey ∩ ({to} × Waypoints × {TRUE}) ≠ ∅ not theorem ›
THEN
   act1:   auv_at ≔ auv_at ⩤ {from ↦ FALSE, to ↦ TRUE} ›
   act2:   can_transit ≔ FALSE ›
   act3:   can_survey ≔ TRUE ›
   act4:   total_time ≔ total_time + time_revisit(from ↦ to ↦ via) ›
   act5:   total_uncertainty ≔ uncertainty_revisit(from ↦ to ↦ via) ›
END

transit_comm:   not extended ordinary ›
ANY
   from     ›
   to   ›
   via   ›
WHERE
   grd1:   from ∈ Waypoints not theorem ›
   grd2:   to ∈ Waypoints not theorem ›
   grd3:   to ≠ from not theorem ›
   grd4:   via ∈ Areas not theorem ›
   grd5:   auv_at(from) = TRUE not theorem ›
   grd6:   can_transit = TRUE not theorem ›
   grd7:   is_comm(from ↦ to ↦ via) = TRUE not theorem ›
   grd8:   is_survey ∩ ({to} × Waypoints × {TRUE}) ≠ ∅ not theorem ›
   grd9:   bit_rate(via) > 0 not theorem ›
THEN
   act1:   auv_at ≔ auv_at ⩤ {from ↦ FALSE, to ↦ TRUE} ›
   act2:   can_transit ≔ FALSE ›
   act3:   can_survey ≔ TRUE ›
   act4:   total_time ≔ total_time + time_comm(from ↦ to ↦ via) + total_data÷bit_rate(via) ›
   act5:   total_uncertainty ≔ total_uncertainty + uncertainty_comm(from ↦ to ↦ via) ›
   act6:   total_data ≔ 0 ›
END

END
```

Figure C-5: Event-B Machine of the PDDL domain, showing the `transit_revisit` and `transit_comm` event (part III)

```
MACHINE
    M_Survey1
REFINES
    M_Survey
SEES
    C_Survey1
VARIABLES
    auv_at
    can_survey
    can_transit
    is_survey
    is_transit
    is_gps
    is_revisit
    is_comm
    total_time
    total_uncertainty
    total_data
    total_width
    sonar_range
    survey_speed
    area_width
    bit_rate
    survey_data
    data_threshold
    leg_length
    uncertainty_survey
    time_transit
    time_gps
    time_revisit
    time_comm
    uncertainty_transit
    uncertainty_gps
    uncertainty_revisit
    uncertainty_comm
INVARIANTS
    inv1:   auv_at \ (Waypoints × {FALSE}) ≠ ø not theorem
    inv2:   total_data < data_threshold not theorem
```

Figure C-6: Event-B Machine of a simple PDDL problem, showing all variables and invariants (part I)

```
EVENTS
  INITIALISATION:  not extended ordinary
  THEN
    act1:  auv_at ≔ (Waypoints × {FALSE}) ⩤ {wp0 ↦ TRUE}
    act2:  can_survey ≔ TRUE
    act3:  can_transit ≔ FALSE
    act4:  is_survey ≔ (Waypoints × Waypoints × {FALSE}) ⩤ {wp0 ↦ wp1 ↦ TRUE, wp1 ↦ wp0 ↦ TRUE,
wp2 ↦ wp3 ↦ TRUE, wp3 ↦ wp2 ↦ TRUE, wp4 ↦ wp5 ↦ TRUE, wp5 ↦ wp4 ↦ TRUE}
    act5:  is_transit ≔ (Waypoints × Waypoints × {FALSE}) ⩤ {wp1 ↦ wp3 ↦ TRUE, wp1 ↦ wp5 ↦ TRUE,
wp3 ↦ wp5 ↦ TRUE, wp0 ↦ wp2 ↦ TRUE, wp0 ↦ wp4 ↦ TRUE, wp2 ↦ wp4 ↦ TRUE, wp3 ↦ wp1 ↦ TRUE, wp5 ↦ wp1 ↦ TRUE,
wp5 ↦ wp3 ↦ TRUE, wp2 ↦ wp0 ↦ TRUE, wp4 ↦ wp0 ↦ TRUE, wp4 ↦ wp2 ↦ TRUE}
    act6:  is_gps ≔ (Waypoints × Waypoints × {FALSE}) ⩤ {wp1 ↦ wp3 ↦ TRUE, wp1 ↦ wp5 ↦ TRUE, wp3 ↦
wp5 ↦ TRUE, wp0 ↦ wp2 ↦ TRUE, wp0 ↦ wp4 ↦ TRUE, wp2 ↦ wp4 ↦ TRUE, wp3 ↦ wp1 ↦ TRUE, wp5 ↦ wp1 ↦ TRUE, wp5 ↦
wp3 ↦ TRUE, wp2 ↦ wp0 ↦ TRUE, wp4 ↦ wp0 ↦ TRUE, wp4 ↦ wp2 ↦ TRUE}
    act7:  is_revisit ≔ (Waypoints × Waypoints × Areas × {FALSE})
    act8:  is_comm ≔ (Waypoints × Waypoints × Areas × {FALSE})
    act9:  total_time ≔ 0
    act10:  total_uncertainty ≔ 0
    act11:  total_data ≔ 0
    act12:  total_width ≔ 0
    act13:  sonar_range ≔ 6
    act14:  survey_speed ≔ 1
    act15:  area_width ≔ 12
    act16:  bit_rate ≔ (Areas × {0})
    act17:  survey_data ≔ 0
    act18:  data_threshold ≔ 100
    act19:  leg_length ≔ 5
    act20:  time_transit ≔ (Waypoints × Waypoints × {0}) ⩤ {wp1 ↦ wp3 ↦ 1, wp1 ↦ wp5 ↦ 2, wp3 ↦ wp5
↦ 1, wp0 ↦ wp2 ↦ 1, wp0 ↦ wp4 ↦ 2, wp2 ↦ wp4 ↦ 1, wp3 ↦ wp1 ↦ 1, wp5 ↦ wp1 ↦ 2, wp5 ↦ wp3 ↦ 1, wp2 ↦ wp0 ↦
1, wp4 ↦ wp0 ↦ 2, wp4 ↦ wp2 ↦ 1}
    act21:  time_gps ≔ (Waypoints × Waypoints × {0}) ⩤ {wp1 ↦ wp3 ↦ 2, wp1 ↦ wp5 ↦ 3, wp3 ↦ wp5 ↦
2, wp0 ↦ wp2 ↦ 2, wp0 ↦ wp4 ↦ 3, wp2 ↦ wp4 ↦ 2, wp3 ↦ wp1 ↦ 2, wp5 ↦ wp1 ↦ 3, wp5 ↦ wp3 ↦ 2, wp2 ↦ wp0 ↦ 2,
wp4 ↦ wp0 ↦ 3, wp4 ↦ wp2 ↦ 2}
    act22:  time_revisit ≔ (Waypoints × Waypoints × Areas × {0})
    act23:  time_comm ≔ (Waypoints × Waypoints × Areas × {0})
    act24:  uncertainty_survey ≔ 1
    act25:  uncertainty_transit ≔ (Waypoints × Waypoints × {0})
    act26:  uncertainty_gps ≔ (Waypoints × Waypoints × {0})
    act27:  uncertainty_revisit ≔ (Waypoints × Waypoints × Areas × {0})
    act28:  uncertainty_comm ≔ (Waypoints × Waypoints × Areas × {0})
  END

  goal:  not extended ordinary
  WHERE
    grd1:  total_width ≥ area_width not theorem
    grd2:  total_data = 0 not theorem
  END

END
```

Figure C-7: Event-B Machine of a simple PDDL problem, showing the initialization and goal events. The actions of the domain Machine are not shown, but are obviously also present in this machine as refined actions. (part II)

Bibliography

[1] M. Cashmore, M. Fox, D. Long, D. Magazzeni, and B. Ridder, "Artificial intelligence planning for AUV mission control," *IFAC-PapersOnLine*, vol. 28, no. 2, pp. 262–267, 2015.

[2] B. Blondé, "LAUV - The Light Autonomous Underwater Vehicle," 2013. Retrieved 18-10-2018, from `https://business.esa.int/sites/default/files/SW1Clemaco-OceanScan-MST_LAUV_Presentation_v3_0.pdf`.

[3] I. E. Tuphanov and A. P. Scherbatyuk, "Adaptive algorithm of AUV meander pattern trajectory planning for underwater sampling," in *Proceedings of the 10th (2012) ISOPE Pacific/Asia Offshore Mechanics Symposium, PACOMS 2012*, pp. 181–185, 2012.

[4] R. E. Hansen, H. J. Callow, T. O. Sabo, and S. A. V. Synnes, "Challenges in seafloor imaging and mapping with synthetic aperture sonar," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 10 PART 1, pp. 3677–3687, 2011.

[5] 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*, 2016.

[6] R. B. Wynn, V. A. I. Huvenne, T. P. Le Bas, B. J. Murton, D. P. Connelly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons, E. J. Sumner, S. E. Darby, R. M. Dorrell, and J. E. Hunt, "Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience," *Marine Geology*, vol. 352, pp. 451–468, 2014.

[7] G. E. Packard, A. Kukulya, T. Austin, M. Dennett, R. Littlefield, G. Packard, M. Purcell, R. Stokey, and G. Skomal, "Continuous autonomous tracking and imaging of white sharks and basking sharks using a REMUS-100 AUV," in *OCEANS 2013 MTS/IEEE - San Diego: An Ocean in Common*, 2013.

[8] F. Florin, F. Van Zeebroeck, I. Quidu, and N. Le Bouffant, "Classification performances of mine hunting sonar: Theory, practical results and operational applications," *Proceedings of Undersea Defence Technology (UDT) Europe 2003*, 06 2003.

[9] Hydroid, "New generation REMUS 200 for defence applications," 2016. Retrieved 21-02-2019, from `https://www.hydroid.com/new-generation-remus-100-defense-applications`.

[10] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," 1998.

[11] S. Reed, I. T. Ruiz, C. Capus, and Y. Petillot, "The automatic fusion of classified sidescan sonar mosaics using CML-RTS and markov random fields," in *Oceans 2005 - Europe*, vol. 2, pp. 883–888, 2005.

[12] G. Taraldsen, T. A. Reinen, and T. Berg, "The underwater GPS problem," in *OCEANS 2011 IEEE - Spain*, 2011.

[13] L. Paull, M. Seto, and H. Li, "Area coverage planning that accounts for pose uncertainty with an AUV seabed surveying application," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6592–6599, 2014.

[14] Z. A. Saigol, G. Frost, N. Tsiogkas, F. Maurelli, D. M. Lane, A. Bourque, and B. Nguyen, "Facilitating cooperative AUV missions: Experimental results with an acoustic knowledge-sharing framework," in *OCEANS 2013 MTS/IEEE - San Diego: An Ocean in Common*, 2013.

[15] D. Williams, "On optimal AUV track-spacing for underwater mine detection," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4755–4762, 2010.

[16] D. P. Williams, F. Baralli, M. Micheli, and S. Vasoli, "Adaptive underwater sonar surveys in the presence of strong currents," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 2604–2611, 2016.

[17] T. Krogstad and M. Wiig, "Autonomous survey and identification planning for AUV MCM operations," in *Conference: Undersea Defence Technology*, 06 2014.

[18] N. Abreu, N. Cruz, and A. Matos, "Accounting for uncertainty in search operations using AUVs," in *2017 IEEE Underwater Technology (UT)*, pp. 1–8, 2017.

[19] N. Kapetanović, N. Mišković, A. Tahirović, M. Bibuli, and M. Caccia, "A side-scan sonar data-driven coverage planning and tracking framework," *Annual Reviews in Control*, vol. 46, pp. 268–280, 2018.

[20] M. Cashmore, M. Fox, T. Larkworthy, D. Long, and D. Magazzeni, "AUV mission control via temporal planning," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6535–6541, 2014.

[21] P. Muñoz, M. R-Moreno, and D. Barrero, "Unified framework for path-planning and task-planning for autonomous robots," *Robotics and Autonomous Systems*, vol. 82, pp. 1–14, 2016.

[22] 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*, pp. 1–8, 2016.

[23] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, pp. 221–225. John Wiley & Sons Ltd., 2011.

[24] 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.

[25] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtós, and M. Carreras, "ROSPlan: Planning in the robot operating system," in *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, vol. 2015-January, pp. 333–341, 2015.

[26] D. McDermott, "The 1998 AI planning systems competition," *AI Magazine*, vol. 21, no. 2, pp. 35–55, 2000.

[27] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1, pp. 281 – 300, 1997.

[28] H. A. Kautz and B. Selman, "Pushing the envelope: Planning, propositional logic and stochastic search," in *AAAI/IAAI, Vol. 2*, 1996.

[29] D. Gnad, A. Shleyfman, and J. Hoffmann, "DecStar - STAR-topology DECoupled search at its best," in *IPC-9 planne rabstracts*, pp. 40–44, 2018.

[30] M. Fox and D. Long, "PDDL2.1: an extension to PDDL for expressing temporal planning domains," *CoRR*, vol. abs/1106.4561, 2011.

[31] J. Hoffmann, "The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables," *CoRR*, vol. abs/1106.5271, 2011.

[32] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 01 2012.

[33] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research - JAIR*, vol. 26, 09 2011.

[34] S. Richter and M. Westphal, "The LAMA planner: Guiding cost-based anytime planning with landmarks," *CoRR*, vol. abs/1401.3839, 2014.

[35] A. Coles, A. Coles, M. Fox, and D. Long, "Forward-chaining partial-order planning," pp. 42–49, 2010.

[36] A. Gerevini and I. Serina, "LPG: A planner based on local search for planning graphs with action costs.," pp. 13–22, 01 2002.

[37] A. G. Richards, "Flight optimization for an agricultural unmanned air vehicle," *2018 European Control Conference (ECC)*, pp. 2891–2896, 2018.

[38] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[39] J. Backer and D. G. Kirkpatrick, "Finding curvature-constrained paths that avoid polygonal obstacles (extended abstract)," pp. 66–73, 01 2007.

[40] P. R. Giordano and M. Vendittelli, "Shortest paths to obstacles for a polygonal dubins car," *IEEE Transactions on Robotics*, vol. 25, no. 5, pp. 1184–1191, 2009.

[41] R. Howey, D. Long, and M. Fox, "VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL," in *16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 294–301, 2004.

[42] F. Fourati, M. T. Bhiri, and R. Robbana, "Verification and validation of PDDL descriptions using Event-B formal method," *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pp. 770–776, 2016.

[43] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*. New York, NY, USA: Cambridge University Press, 1st ed., 2010.

[44] T. Prestero, "Verification of a six-degree of freedom simulation model for the REMUS autonomous underwater vehicle," Master's thesis, Massachusetts Institute of Technology, 2001.

Glossary

List of Acronyms

AUV	Autonomous Underwater Vehicle
DVL	Doppler Velocity Logger
EHC	Enforced Hill Climbing
FD	Fast Downward
FF	Fast Forward
GPS	Global Positioning System
INS	Inertial Navigation System
IMC	Inter-Module Communications
IMU	Inertial Measurement Unit
IPC	International Planning Competition
LAUV	Light Autonomous Underwater Vehicle
MCM	Mine Countermeasure
MILP	Mixed Integer Linear Programming
OPTIC	Optimizing Preferences and TIme-dependent Costs
PDDL	Planning Domain Definition Language
POD	Probability of Detection
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SPM	Sonar Performance Model

SSS Side-Scan Sonar

TNO the Netherlands Organisation for applied scientific research

UUV Unmanned Underwater Vehicle