Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Fan Gao

Mission Planning and Replanning for ROVs

Master's thesis in Marine Technology
Supervisor: Professor Martin Ludvigsen
July 2020

NTNU
Norwegian University of
Science and Technology

Fan Gao

Mission Planning and Replanning for ROVs

Master's thesis in Marine Technology
Supervisor: Professor Martin Ludvigsen
July 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU

Kunnskap for en bedre verden

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER THESIS IN MARINE CYBERNETICS

SPRING 2020

FOR

STUD. TECHN. Fan Gao

Mission Planning and Replanning for ROVs

Work description (short description)
Remotely Operated Vehicles (ROVs) are commonly used for subsea inspection and intervention tasks as they have high manoeuvrability and payload capacity. By implementing a high level of autonomy, we can not only reduce operator reliance, the cost of operation, but also increases the complexity of missions and give possibilities of breaking through limitations of ROVs. The master thesis focuses on proposing a mission planning and re-planning scheme. In the proposed system, a planner can generate a sequence of actions based on the given operator request and other information input. After completion of all actions, the ROV should reach the goal. Each pre-planned action can be appropriately identified and executed by a lower-level system. Under some predefined circumstances, the re-planning command should be triggered, and the planner will be activated again, after which a new plan is generated and replaces the old one. Finally, to realize online planning, real-time interaction between the planner and actor should be designed.

Scope of work
1. Review the literatures of mission planning and replanning.
2. Propose an algorithm for mission planning.
3. Propose some typical conditions for replanning and propose online planning system.
4. Test the system performance by simulations.
5. Analyse and discuss simulation results.

The report shall be written in English and edited as a research report including literature survey, description of mathematical models, description of algorithms, simulation results, discussion and a conclusion including a proposal for further work. Source code should be provided. It is supposed that Department of Marine Technology, NTNU, can use the results freely in its research work, unless otherwise agreed upon, by referring to the student's work.

The thesis should be submitted within 6 July.

Advisors:

Professor Martin Ludvisgen
Supervisor

Abstract

This thesis proposes an autonomous mission planning and control architecture for a Remotely Operated Vehicle (ROV). The architecture aims to develop a mission control architecture and apply automated planning on the system to guide subsea navigations and operations intelligently. Therefore, the ROV could perform more complex and dangerous tasks autonomously without human operators. This mission planning and control system enables autonomous mission execution for general subsea navigation and operations. The proposed architecture increases safety and autonomy, minimizes the need for human intervention, and enables reduction of time and other costs on exploring and exploiting the ocean.

The architecture is developed as a layered mission planning system for global navigation and local operation. The mission planning and control of local operations is a sub-behavior under global navigation missions. The system focuses on the realization of ROV navigation, tracking a Structure of Interest (SOI), subsea operation, and reactive obstacle avoidance in the underwater environment. For this purpose, the designed system is composed of three parts:

(1) An autonomous framework for plan-based mission control and execution: For a given mission request, the system makes plans automatically and executes the planned actions sequentially to achieve the mission goals. This work draws on [36]'s hybrid control architecture for ROVs and develops a plan-based mission control architecture. The sequential steering of behaviors has been changed to plan-based mission control. The coordination mechanism is introduced in the behavior-based reactive layer of the architecture, such that only one reactive behavior is activated from a reasonable selection of all.

(2) A layered mission planning system guides global navigation tasks and local seabed intervention: Based on the current states and the mission request, the planning system automatically generates desired actions to achieve the goal. These actions are developed using the STRIPS planning language. Best-first search (BFS), heuristic search, and fast-forward (FF) search are implemented and tested. A comparison of these search methods shows that the heuristic search has the most satisfactory performance for a simple planning problem. The mission planning is integrated into the autonomous framework to perform mission planning, control, and execution for ROVs. A sub-planner for a specific operation is proposed as a refinement of the action *Operation* under global navigation tasks.

(3) A re-planning mechanism: Under some circumstances, the action sequence should be adjusted and re-planned to meet new requirements and ensure safety. Three *Lookahead* algorithms are implemented to trigger repeated planning and re-planning when necessary, after each action is completed, and at all times. For ROVs, a Run-Lookahead algorithm is preferred, which is designed to be sensitive to emergency and failure mode and then do re-planning correspondingly.

This method reduces the rate of failure compared to Run-Lazy-Lookahead algorithms and also computational inexpensive compared to the Run-Concurrent-Lookahead algorithm.

The capability and limitations of the architecture are demonstrated through software simulation. Due to the lack of control system on ROV manipulators, the autonomy on operations with the manipulator's arm has not been developed nor physically tested. Control and optimization of autonomous maneuvering should be considered for future research on this subject. The result is a mission planning and control system that automatically generates action sets, which is then executed to steer the ROV to finish missions and perform observation-based operation actions derived from the sub-planner of operation. The mission planner and mission control system can add new actions, remove existing actions, and enable flexibility for more complex missions.

Preface

This report is written based on the research work of my master thesis in the specialization of Marine Cybernetics. The master thesis is a continuation of the project thesis carried out in the winter of 2019 and is the final work in the master's program of the Department of Marine Technology, Faculty of Engineering, NTNU.

The main contribution of the master thesis is the design and development of an autonomous mission planning and control architecture for ROVs to execute general navigation tasks and create a layered framework that enables more refinements on subsea local intervention tasks. The framework developed and simulated in this thesis also integrates the design of a path planner for homing developed by Signe Birch Moltu and interacts with vision-based SLAM obstacle detection and motion estimation using TCP/IP communication.

This master thesis is a result of both individual work and help from fellow students and professors at NTNU. It is assumed that the reader of this report retains a basic knowledge within engineering science and marine control systems.

Trondheim, July 05, 2020

GAO FAN

Acknowledgment

I would first like to thank my supervisor Prof. Martin Ludvigsen of the Department of Marine Technology at NTNU. He helped me a lot with providing me instructive advice and useful suggestions on my thesis. I have no background in automated planning and mission management before the work with this thesis. In the beginning, I really cannot understand how the planner could generate a sequence of symbolic actions. My supervisor provided me some related thesis of previous students, which is inspiring and helpful. When I was developing my mission planning and control system, I met the trouble of integrating the system into the ROV control system. My supervisor introduced Researcher Trygve Olav Fossum, who gave me much instruction in the application of software *Labview*. So I also appreciate Trygve efforts here. Besides, my supervisor also encourages other master students and me to work together. Therefore, the mission planning and control system proposed in this thesis has the opportunity of integrating path planning, vision-based obstacle detection and motion estimation in the system.

I would also like to thank Tore Mo-Bjørkelund for his help with TCP communication. He joined our group meeting twice and provided useful suggestions on router and network settings to achieve TCP/IP connection between two PCs. Without his help, we might not implement joint simulation successfully.

Finally, I must express my profound gratitude to my parents and friends for their continuous support and encouragement.

Fan Gao

Table of Contents

List of Tables

List of Figures

Abbreviations

ROV	=	Remotely Operated Vehicle
UDP	=	User Datagram Protocol
TCP	=	Transmission Control Protocol
cRIO	=	Compact Reconfigurable Inputs and Outputs
HIL	=	Hardware-In-the-Loop
GUI	=	Graphical User Interface
NI	=	National Instruments
OA	=	Obstacle Avoidance
CT	=	Camera Tracking
ST	=	Sonar Tracking
SOI	=	Structure Of Interest
NED	=	North-East-Down
DOF	=	Degree Of Freedom
3D	=	Three-Dimensional
NTNU	=	Norwegian University of Science and Technology
AUR-lab	=	Applied Underwater Robotics Laboratory
DP	=	Dynamic Positioning
IMR	=	Inspection, Maintenance and Repair
HAA	=	Hybrid Agent Architecture
η	=	Position vector in NED frame
ψ	=	Heading/ orientation between NED and BODY frame
BFS	=	Best-first search
DFS	=	Depth-first search
FF	=	Fast-forward search
AI	=	Artificial Intelligence
UUV	=	Unmanned underwater vehicle
DP	=	Dynamic Positioning
DVL	=	Doppler Velocity Log
IMU	=	Inertial Measurement Unit
PDDL	=	Planning Domain Definition Language
STRIPS	=	Stanford Research Institute Problem Solver
HTN	=	Hierarchical Task Network
HSP	=	Heuristic Search Planner

Chapter 1

Introduction

Remotely Operated Vehicles (ROVs) are one kind of underwater mobile vehicles used for subsea tasks such as seabed mapping, inspection and intervention. ROV usually includes three main components: vehicle, umbilical cable and control stations. As the name shows, the operator on the surface vessel or platform can remotely control ROVs to perform some operations. However, remote control means human interaction, which may lead to unsatisfactory or sub-optimal results due to lack of experience and restriction on the execution of complex tasks. Autonomy applied to ROVs could significantly reduce human intervention and enable ROVs to perform tasks accurately and efficiently [14].

According to the classification in [34], four levels of autonomy are defined as *Manual Operation*, *Management by consent*, *Management by exception* and *Fully autonomy*. The level of situational awareness, decision making and control are increased with levels of autonomy. To achieve full autonomy of ROVs, a system of mission planning, control and execution should be developed and implemented, guiding and interacting with the ROV lower-level control system. This thesis aims to develop such a system for general ROV missions. By connecting this mission control system with ROV lower-level control system, ROV is expected to perform fully automatic operations without human intervention. The purpose of developing such a mission management system is to increase the level of autonomy and thus enables automatic task execution. This is carried out by integrating a mission planner and a re-planning algorithm into the mission control architecture. This autonomous mission control system is presented and tested through simulation.

This chapter introduces important information on applying autonomy in ROVs. Section 1.1 presents motivation and background of this thesis. Section 1.2 describes the objective of this thesis. Section 1.3 presents limitations of executions of the work. Section 1.4 describes the contributions of this thesis. Section 1.5 describes the structure of this thesis.

1.1 Background and Motivation

This section introduces the background and motivation of applying autonomy to ROVs. Classification of ROVs and its current applications are presented in Section 1.1.1. Autonomy used on ROVs and its previous approaches are described in Section 1.1.2.

1.1.1 ROVs

Most ROVs are equipped with sensors like IMU, DVL and pressure sensor. Different classes of ROVs are widely used in many regions: commercial and salvage Diving, military, environmental research, oil and energy, shipping, underwater discovery and so forth. ROVs can be classified by size, missions or working depth. Some typical categories based on functions and tasks are listed in [41]:

(1)**Micro- and Mini-class ROVs:** These two types of ROVs are commonly used as an alternative to divers and the main difference between Micro- and Mini-class ROVs is their size and weight.
(2)**General- and Inspection-class ROVs:** The General-class ROVs usually do light survey tasks and have a limitation of operation depth, typically 1000 meters. Inspection-class ROVs can perform observation and data collection tasks, which are typically equipped with sonar and camera.
(3)**The work-class ROVs:** can be subdivided into light work-class and heavy work-class, dependent on the workload. These ROVs are usually equipped with manipulator's arms for operations like installation, repair and sampling.

Reported in [20], NORSOK U-102 classifies ROVs into three major classes: Class 1-Pure observational class, Class 2-Observation with payload options, and Class 3-work class vehicles. Authored by International Marine Contractors Association (IMCA), IMCA R 004 ROV code classifies ROVs into five categories: observation, observation with payload, work-class, towed and bottom crawlers and prototype vehicles. The classification of autonomous ROVs based on the autonomy level is described in [20] where ROVs are classified as semi-autonomous ROVs and autonomous ROVs.

ROVs are featured by an umbilical cable connecting between the ship and ROV, which is used to transmit both electricity and signals [14]. The cable not only enables it to perform high payload work but also transfers high-resolution signals and videos. Therefore, ROVs are applied to some work, such as archaeological operations that AUVs and other underwater vehicles cannot deal with. On the other hand, the unlimited power supply via umbilical cable guarantees a long working time. Also, manipulators are a typical feature for work-class ROVs, with which ROVs can do sampling and intervention. Compared with other underwater vehicles, ROVs also have drawbacks. Due to the existence of umbilical cables, ROVs are limited in spatial range, and the umbilical cable is exposed to environmental loads such as current forces and ocean animals introduced in [1]. ROVs also can get wrapped by umbilical cables from itself or other ROVs due to poor operation and environmental factors. In a worse case, the umbilical cable may be damaged or even cut off by subsea structures. Therefore, when ROVs are under operations, one of the most critical sensors that can trigger emergency alarm is the cable tension.

1.1.2 Autonomy Applied on ROVs

ROVs are marine robots used for a wide range of subsea engineering, archaeological and military tasks, including inspection, survey and intervention. However, the application of autonomy on ROVs is still not fully developed. The agent architecture proposed in [36] can perform well-predefined tasks without human intervention but lack intelligence in dealing with complex and symbolic tasks that require planning. A high-level autonomy is needed to adjust to the increasingly complex missions and strict requirements, especially for situation awareness in emergency and decision making by taking typical situations and necessary factors into consideration. Besides, applying autonomy on ROVs can also reduce working time, energy consumption, and increase safety and reliability [14]. A classification of autonomy levels is proposed by [34]:

(1) Manual Operation [Level 1]: Human operators execute all behaviors, functions and commands.
(2)Management by Concent [Level 2]: The management by consent system can automatically recommend actions that fulfill the required functions and provide specific information and suggestions of decision-making to the operator.
(3)Management by Exception [Level 3]: The management by exception control system can automatically execute missions while human operators' intervention may be required for potential modification.
(4)Fully Autonomous [Level 4]: The fully autonomous control system can execute missions automatically without human intervention.

The level of situational awareness, decision making and control are increased with levels of autonomy. This thesis aims to increase the level of autonomy to level three(Management by exception) for ROV subsea intervention. This thesis proposes a new definition of symbolic actions for ROV mission planning and control (Section 5.2). This enables artificial planning to guide and integrate with control systems for task execution.

1.2 Objectives

The objective of this thesis is to build a novel mission planning and control system, taking fast reaction and conditional re-planning into consideration. Due to a limited amount of time, literature study, integration of a layered mission planning system and mission control architecture is included and developed in this project, while refinements on specific actions and behaviors are under consideration. The 'supervision' system that interleaves planning and acting is also established. The overall objective of the thesis work is:

1. Test and compare different search methods for automated mission planning.
2. Develop a mission control system and integrate mission planning into the system.
3. Develop an algorithm for supervision and repeated planning.

4. Simulate the the mission execution in the Hardware-in-the-loop (HIL) simulation environment.

The thesis only focuses on the mission planning and control part, assuming that situation is well detected. However, the mission planner also requires aid from other systems such as situation awareness, which provides information on obstacles and motion estimation. With the integration of situation awareness and mission planning, the control system can achieve safe and accurate navigation and operation.

1.3 Limitations

There are some limitations to the proposed mission planning and control system. First of all, the thesis focuses on designing a mission planner that works for ROV general missions and integrates it into hybrid mission control and management system. The designed actions are simplified due to limitations of time. For example, the specific execution of manipulation is not included in this thesis. The final hybrid mission planning and control system is described in Chapter 5, including the description of each action.

Second, the mission planner is proposed based on the operating mode of ROVs, which is performing Launch-Descent-Transit to the location of operation, doing the operation, and then moving to the ending point by completing Transit-Descent. Sonar Tracking and Camera Tracking are optional actions used to track and approach the SOI. The developed mission planning and control systems consider limited reactive behaviors and failure modes. More details in failure detection and diagnosis should be counted to achieve autonomy.

Last but not least, the processing of the sonar signal and camera signal is not yet well developed and thus, these signals are virtual and manually set during simulations. Real-time signals for detecting obstacles and seabed structures are simulated during the tests to verify the performance of obstacle avoidance and target tracking behaviors.

Autonomy forms a wide field of research. Full autonomy realization requires the collaboration of several groups working on mission management, situation awareness, ROV guidance and control systems, and so forth. This master thesis tackles one of the key parts—mission planning and control. Integration in these fields is not entirely done and is to be realized in further work.

Despite the above limitations, the proposed system of mission planning and control in this thesis gives successful results through HIL simulations. The performance of the system is satisfactory.

1.4 Contributions

This thesis's contributions are within the design of a hybrid mission planning and control system, consisting of a deliberative layer, a functional reactive layer, and a control execution layer. A combination of the reactive and deliberative layer is called a hybrid architecture, widely used in robotics.

The deliberative layer performs autonomous behaviors based on known situations and events. The mission planner can recognize and categorize the tasks, generating a plan that fulfills the target and deliver a set of actions or commands to relevant mission controllers. One or more sub-planners might be activated as a refinement of an action. The mission planner in this layer enables ROVs to plan tasks automatically and the re-planning structure responds fast when changes and emergencies occur.

The reactive layer responds to contingency by analyzing sensor data, reasoning unexpected or unknown situations, and modifying or interrupting missions. Mission re-planning is laid in both deliberative and reactive layer. The mission should be automatically replanned due to events such as failures in control, environment changes(such as an encounter with unexpected obstacles) and changes of mission targets.

(1) A mission planner for ROV Minerva can generate a proper action sequence for the vehicle to execute. The theory for mission planner is presented in Chapter 2. Simulation of the mission planning algorithms is presented in Section 7.1.

(2) Replanning and supervision algorithms are proposed and applied in the mission control system. This part is simulated within the mission planning and control system. The theory of this part is presented in Chapter 3.

(3) An automatic mission control architecture is developed, including the integration of mission planning and replanning algorithms. The whole system is tested by simulation. Chapter 5 describes how the system works. Chapter 6 introduces how the plan-based mission control system is integrated into the ROV control system and how the ROV performs automatic planning, control and execution of missions. Simulation results are presented and discussed in Chapter 7 and Chapter 8.

To explicitly present the contributions of this thesis, Figure 5.2 shows how the mission planning and control system for ROV Minerva is structured. The contributions of this thesis are mainly within the autonomous mission planning and control architecture.

1.5 Outline of Thesis

Chapter 2 mainly presents the literature study of automated planning, especially classical planning. Typical languages (including *STRIPS*, *Planning Domain Definition Language (PDDL)* and *Hierarchical Task Network (HTN)*) and search methods(including *Heuristic Search* and *Fast-Forward Search*) for classical planning are introduced in this chapter.

Chapter 3 presents three algorithms of replanning and literature study of online planning. Analysis of possible conditions that require replanning is discussed in this chapter.

Chapter 4 presents literature study on mission control architectures. Three different architectures are *Deliberative Architecture* in Section 4.2, *Reactive Architecture* in Section 4.3 and *Hybrid Architecture* in Section 4.4. Section 4.3.1 and Section 4.3.2 introduces typical and commonly used cooperate mechanisms used in *Reactive Architecture* that synthesize an output command.

Chapter 5 proposes a hybrid architecture for automatic mission planning and control of ROVs. Typical actions are defined based on the ROV working property. The architecture is developed based on the literature study from Chapter 2, Chapter 3 and Chapter 4. The proposed mission planner is developed in the Python environment and further encoded into the ROV mission control system in the Labview environment.

Chapter 6 presents how the autonomy framework is integrated with the ROV control system and settings done to achieve a virtual experiment.

Chapter 7 presents simulation results of both individual planning methods and the whole mission planning and control system. The ROV control system is utilized as a platform and basis for the simulation of the mission planning and control system.

Chapter 8 presents a discussion of the simulation results. This chapter analyses the ROV mission planning, control and execution result from the simulation in Labview, to draw a conclusion.

A conclusion is presented in Chapter 9. Moreover, a discussion about further work to achieve ROV full autonomy is presented at the end of this chapter.

Chapter 2

Mission Planning

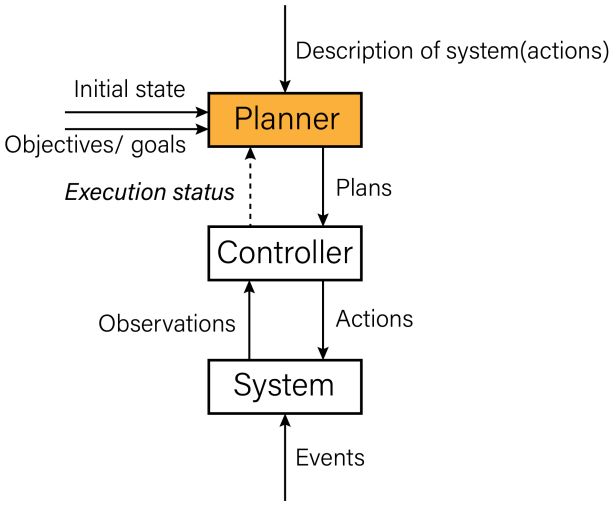

Figure 2.1: Conceptual model of the mission planning and control system (A)

This chapter presents a literature study on automated planning, especially on classical planning problems. Figure 2.1 shows the conceptual mission planning and control model and the theory about the *Planner* is introduced in this chapter. The research focuses on the heuristic search method applied in mission planning problems by [6] and GraphPlan method proposed by [3]. A new fast-forward search method is proposed in [22] that combines heuristic search method with planning graph analysis for the guidance of plan search. Section 2.1 describes AI planning and its branches. Section 2.2 presents some specific planning languages and search methods commonly applied in classical mission planning.

2.1 Background: Artificial Intelligence(AI) Planning

Planning is an Artificial Intelligence technology that seeks to select and organize activities to achieve specific goals, concluded in [18]. Automated planning and scheduling are denoted as AI planning. It usually works accordingly with the realization of action sequences in a control

system. For ROV's classical control problems, intelligent mission execution is not possible. Human operators are required for reasoning, decision making, and monitoring of the system. Fortunately, with the help of AI planning, agents and vehicles are able to make decisions and perform complex actions in a highly automatic way. For known initial states, goal states and a set of predefined actions, automated planning can plan a sequence of actions to approach the goal state with the given world models.

Reasoning and planning are challenging problems in the field of AI [42]. The planning problem is to synthesize a plan which can generate a state containing the desired goals. Initial conditions of the world, desired goals and a description of possible actions should be predefined in the planning problem. Generally, the planning problem is established based on some assumptions of simplification. For example, the most straightforward planning problem is classical planning, with the following restrictive assumptions [18]:

- Finite System: The system has a finite number of states, actions and events.
- Fully observable: The system is fully observable and thus, complete knowledge of the world model for planning is accessible.
- Deterministic: Each action has only one outcome.
- Static: There are no exogenous events. Changes of states only come from actions.
- Attainment goals: A set of goal states is defined.
- Sequential plans: The planning result is a linearly ordered sequence of actions.
- Implicit times: The planner does not consider time duration of actions.
- Off-line planning: The planner does not know the execution status and does not automatically update its initial states.

For a complex planning problem, its difficulty not only depends on the description of actions, initial states and goals but also planning properties in many aspects. For example, whether several actions can be taken concurrently and whether actions are deterministic or nondeterministic will definitely affect the complexity of a planning problem. Some typical planning problems, including classical planning, temporal planning, conditional planning, and probabilistic planning with uncertainties, is introduced in [19]. This thesis mainly focuses on the study of classical planning.

In known environments with known world models, planning can be done prior to execution and is donated as offline planning. In dynamic environments with unknown world models, the planning solution often needs to be revised in time based on newly updated world information. This chapter mainly introduces specific planning languages and search methods for classical planning problems. Introduction on how conditional replanning and online planning work for a dynamic environment is discussed in Chapter 3.

2.2 Classical Planning

The attempt at AI planning has started since the 1970s [16]. A new problem solver was proposed called STRIPS, which attempts to find a solution to transform the initial states to the goals based on certain assumptions. It is reported in [42] that these assumptions dominate research in the field of AI planning for more than ten years and have been the foundation of planning research, namely 'classical planning'.

Typically, states of the current world, possible actions to be taken in the real world and target goals are the three primary inputs to the classical planning problems. The output of a classical planner is a sequence of actions. Notably, the term 'real world' refers to a reasonable representation of the real world since the real world has infinite properties and cannot be fully expressed. For a planner, the output is reasonable and correct only if the representation of the world states, actions and goals correctly reflects the necessary properties of the real world [42].

Three significant achievements in automated planning are STRIPS (1971), Planning Graphs (1997), and Heuristic Search Planner (HSP, in 1998). STRIPS proposed a general framework for researchers to develop more advanced planning languages and algorithms.

2.2.1 Planning Language: Stanford Research Institute Problem Solver (STRIPS)

The Stanford Research Institute Problem Solver is an automated planner developed in [16]. 'STRIPS' was later regarded as the name of the formal language of this planner. It is the cornerstone for most automated planning languages in use today.

As stated in [28], STRIPS made essential contributions to planning research. It finds a way to simplify the frame problem by assuming that the application of action only gives explicitly predefined changes. These changes are expressed in the effects of each action. All other relations and properties of the current situation automatically hold in the successor situation. The STRIPS project introduced a simple syntax in terms of preconditions, add-effects and delete-effects to define an action.

From the STRIPS definition, a state S is defined as a finite set of ground atoms. Besides, all action schemata are assumed to be grounded. Therefore, for STRIPS, there are finitely many possible states, and all possible actions are explicitly defined, as stated in [18]. Actions or operators are defined as:

$$a = (pre(a), add(a), del(a)) \tag{2.1}$$

where $pre(a)$ denotes the preconditions of the action a. The preconditions refer to literals that must be true to apply this action. $add(a)$ denotes the added effects(atoms) of applying the action a and $del(a)$ denotes the deleted effects(atoms) of applying the action a. The combination of

$add(a)$ and $del(a)$ comprises the action's effects, which are literals that the operator will make true. The action is applicable only if all preconditions are satisfied in the current state. Thus, the result of applying an action is defined as:

$$Result(S, \langle a \rangle) = \begin{cases} (S \cup add(a)) \backslash del(a) & pre(a) \subseteq S \\ undefined & otherwise \end{cases} \tag{2.2}$$

Thus, after applying a sequence of n actions, the state S becomes

$$Result(S, \langle a_1, ..., a_n \rangle) = Result(Result(S, \langle a_1, ..., a_{n-1} \rangle), \langle a_n \rangle) \tag{2.3}$$

In a planning task, $P = (A, I, G)$ is a triple that states the set of possible actions (A), initial states (I) and goal states (G). A plan P is a sequence of actions that solves the mission by transforming the initial states to the goal states.

For STRIPS planners, solutions are usually found by applying breadth-first search (BFS), depth-first search(DFS), or the intelligent approach—Heuristic search. Breadth-first-search searches from the initial state and finds all valid solutions that can achieve the goal states. By evaluating all solutions, BFS always finds the most optimal solution to a STRIPS problem, but it could take a long time to get the shortest solution because it is designed to go through all possible solutions. Depth First Search explores as far as possible to find a solution. DFS might not find the most optimal solution to a STRIPS problem, but it is always faster than BFS. A more intelligent way of searching applied in STRIPS is heuristic search, which is also called A* search. A heuristic function is estimated to evaluate the total cost from the initial state to the goal state. The heuristic cost acts to guide the searching process.

2.2.2 Planning Language: PDDL

The Planning Domain Definition Language(PDDL) is one of the standard languages applied in Artificial Intelligence planning. It was proposed based on the STRIPS assumption. By adopting a general formalized description of the planning field, the reuse of relevant research is promoted, and more comparison of systems and methods is possible, supporting faster progress in the planning domain. The Planning domain definition language was first developed and used for the International Planning Competition(IPC) in 1998. The PDDL was designed by [31], evolving into many other editions and extensions. In PDDL, a planning model can be divided into the domain description and the problem description. The domain description is a 'library' that stores predefined actions, while the problem domain states a specific planning problem. The domain, together with problem description, is regarded as input to a planner. The planner output is usually a planned sequence of actions.

In domain definition, predicates and actions are two primary components. Predicates specify predicate names and relevant arguments. There are two kinds of predicates: static and dynamic predicate. the static predicates cannot be changed by actions while the dynamic predicates is changeable. Besides, for each action defined in domain definition, the action name is required and compulsory. Some other components are optional, such as parameters, preconditions, and effects. Parameters are variables that could be instantiated. Preconditions give prerequisites to perform this action. Effects are the consequence of an action.

In problem definition, problem name, domain name, objects, initial state, and goal specification are specified. The domain name must match the term used in the corresponding domain file.

2.2.3 Planning Language: Hierarchical Task Network (HTN)

Stated in 2.2.1 and 2.2.2, STRIPS and PDDL are the most commonly used languages for classical planning. These languages represent planning domains based on state variables. For a planning problem, state variables refer to a set of all possible states of the world. Actions can make restricted changes in the value of state variables when that action is applicable.

Another language for describing planning problems is hierarchical task networks (HTN) in which tasks are categorized as *Goal tasks*, *Primitive tasks* and *Compound tasks*. In this language, state variables are not necessarily involved. For the hierarchical task networks, tasks are given, and each task can be either realized by a primitive action or decomposed into a set of other tasks. HTN planning works through expanding tasks and resolving conflicts iteratively until a conflict-free plan can be found that consists only of primitive tasks proposed by [13].

2.2.4 Planning Graphs

Planning Graphs

The Planning Graph concept is firstly proposed in [3]. Planning Graph Analysis, which constructs and analyses a compact structure to guide its search for a plan, is the main contribution. Two critical features of GraphPlan are: it guarantees the optimal plan solution, and it is insensitive to the order of goals compared with traditional approaches reported in [3]. The Planning Graph Analysis is applicable to STRIPS-like planning domains, which is stated in Section 2.2.1 that preconditions, add-effects and delete-effects are explicitly stated as properties of each action.

A Planning Graph is a leveled graph, with two kinds of nodes and three kinds of edges. Two kinds of nodes are proposition nodes in proposition levels and action nodes in action levels. Three edges are used to represent the relation between propositions and actions: (1) precondition-edges: connect the action nodes in action level i to their preconditions in proposition level i. (2) add-edges: connect the action nodes in action level i to their add-effects in proposition level $i+1$. (3) delete-edges: connect the action nodes in action level i to their delete-effects in proposition level $i+1$. From the initial condition(Level-1), applicable actions in each action level i depend on add- and delete- effects of actions in formal propositions i. Figure 2.2 shows an example of a Planning Graph. The initial states are presented in propositions in time step *1* and the goal states are presented in *Goals*. To construct a Planning graph, all preconditions of an applicable

action in time step *i* must be stated in the propositions in time step *i*. Then, all applicable actions in time step *i* are listed in the Planning Graph and the effects are presented and connected to the propositions in time step *i+1* via add-edges and delete-edges until the effects of applicable actions in time step *m* fulfills all goals.

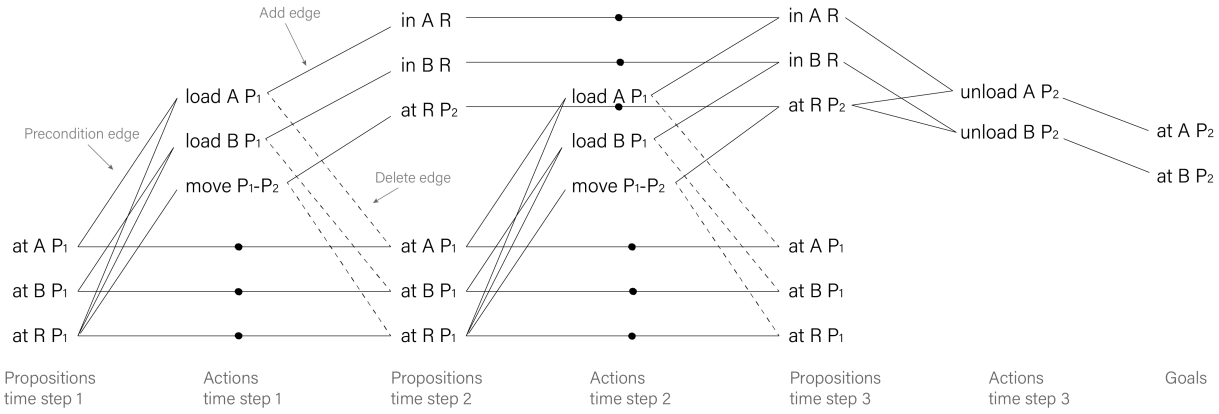

Figure 2.2: An example of the Planning Graph

In the planning graphs, there might be mutual exclusions between two actions and between two literals. If two actions are mutex, no valid plan could contain both of them. If two literals are mutex, no valid plan is available at the same time step. A mutex relation holds between two actions could be inconsistent effects, interference or competing needs. Another kind of mutex relation between two literals might be: one is the negation of the other one or each possible action pair that could achieve the literals is a mutex.

If a good plan exists using t or fewer time steps, then that plan exists as a subgraph of the Planning Graph stated in [3]. The cost of time to create the planning graph structure has a positive correlation with the number of goals, propositions in the initial conditions and properties of each action.

Graph Planning

The basic idea behind graph planning is to use reachability analysis as a heuristic for forward search. There are two stages for Graphplan search: Constructing a data structure to represent information about the executive sequence of actions from the initial state and searching backwards from the goal until it reaches the initial conditions. The first stage is explicitly discussed above. In order to get heuristic values from a planning graph, there are two main steps: Graph expansion and solution extraction. In each loop, graph expansion extends a 'planning graph' forward from the initial state until reaching a necessary condition for plan existence stated in [18]. The solution extraction searches backward from the goal, looking for a proper plan and return the plan and its cost.

There might be many methods for the second stage: searching for a plan based on the constructed planning graph. The one proposed in [3] uses a backward-chaining strategy and recursive search method to search level-by-level until it either succeeds or halt in unsolvable conditions. If the

current goal set of actions turns out to be unsolvable, it will try to find other sets of actions that can reach the current goal. There is a possibility of failing to derive a solvable solution of action sets. Need to mention, goal order does not affect much on the time cost to carry out a solution under this planning method since it follows the recursive search for each goal.

2.2.5 Heuristic Search Planner (HSP)

HSP is developed based on the idea of heuristic search, which performs a forward search with a heuristic function, estimating the cost to the goal. This method is firstly proposed in [4]. Compared to heuristic search applied in path planning, the HSP uses a different heuristic function $h(s)$ by relaxing a problem P into a simpler problem P', which ignores all $del(a)$ of each action. As stated in Section 2.2.1, in a planning task, $P = (A, I, G)$ is a triple that states the set of actions (A), initial states (I) and goal states (G). Therefore, in problem $P' = (A', I, G)$, actions may add new atoms but cannot remove atoms from the existing list.

$$A' = \{(pre(a), add(a), \emptyset) | (pre(a), add(a), del(a)) \in A\} \tag{2.4}$$

By doing so, subgoals become independent of each other. The heuristic function $h'(s)$ can be simply calculated as the number of actions required to reach the goal in the relaxed problem P'. The optimal cost in problem P' is a lower bound on the optimal cost in the original problem P for any combination of initial state and goal state [6]. Thus, the cost function $h'(s)$ in the relaxed problem p' can be regarded as an estimate of the original heuristic function $h(s)$ and becomes an informative and non-overestimating heuristic of problem P.

By applying the heuristic function in the search algorithm, the planning problem could be solved as a heuristic search problem. Furthermore, the heuristic search might proceed too slow and take exponential space. A weighted heuristic function is thus proposed as:

$$f(s) = (1 - w) * g(s) + w * h(s) \tag{2.5}$$

where w is the weighting parameter that ranges from 0.5 to 1.0.

Hill-climbing Search

Hill-climbing search is a simplified heuristic search method that only one of the best children is selected at each step until the goal is reached [5]. The best children's selection is based on the minimization of the heuristic function $h(s)$. In each step, the cost function is taken as the estimated atom costs $g_s(p)$ plus the heuristic $h(s)$. The hill-climbing search applied in HSP cannot promise the optimality of planning results [5]. A new heuristic method for goal estimation and a search strategy combining hill-climbing with the systematic search are proposed in [21]. As he stated: Though it does not guarantee that the solution plans to be optimal, it does find it is close to optimal plans in most cases.

Best-first Search(BFS)
The best-first search could also be used in a heuristic planner. Similar to A^* search, the best-first search keeps an open and a closed list of nodes. Unlike the A^* search, the best-first search weights the heuristic function, and thus the evaluation function becomes:

$$f(s) = g(s) + W * h(s) \tag{2.6}$$

where $g(s)$ is the accumulated cost function and $h(s)$ is the heuristic cost function to the goal. W is always chosen to be larger than 1. With increasing W, the goal can be reached faster while the solution's quality is relatively lower. In general, the value of W ranges from 2 to 10. In [5], the weighted A* searching solution is guaranteed not to exceed the optimal cost by more than a factor of the weighting number W. Therefore, if an optimal solution is expected, setting a low weighing number is a reasonable suggestion.

2.2.6 Fast-Forward(FF) Planning System

Fast Forward planning system is developed based on STRIPS, GraphPlan and heuristic search, firstly proposed in [22]. Figure 2.3 presents the conceptual relations in the fast forward search method. GraphPlan is constructed as a heuristic estimator, which uses a relaxed planning task P' as the estimation of the original planning task P to derive the heuristic cost. An essential difference between FF planning and HSP is the application of GraphPlan that does not rely on independence assumption. Basic HSP assumes facts to be achieved independently and thus ignores potential positive interactions. The estimation of sequential solution length in GraphPlan is usually lower than HSP's estimation. Details about how GraphPlan is constructed are stated in Section 2.2.4. Need to mention, the relaxed GraphPlan will not mark any pair of actions or propositions as mutually exclusive and will never backtrack.

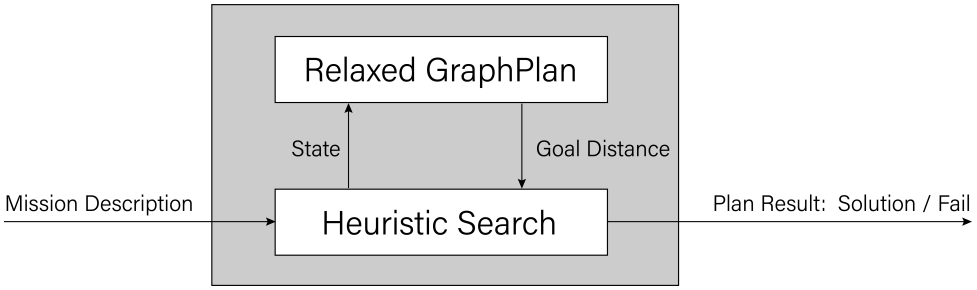

Figure 2.3: FF system architecture

In [22], three methods are introduced that enable the GraphPlan for relaxed planning tasks to return as short solutions as possible. The NOOP-first method is a built-in feature of GraphPlan that ensures a minimality criterion by restricting the repeated selection of actions. The second method—difficulty Heuristic selects appropriate actions with the shortest preconditions when no NOOP is available. Action Linearization may result in shorter plans by executing action a before a' if the action $a \in A_i$ adds a precondition p of another action $a' \in A_i$. The purpose of

applying action linearization is to find a shorter relaxed solution by achieving a smaller number of unsatisfied preconditions.

Relaxed Plan Extraction

Stated in [22], for the relaxed planning graph, we need to mark the number of the layer at which each action or fact firstly appears in the graph, called *layer memberships*. For each action, the layer membership is initialized to infinite, and a counter is initialized to 0. The layer memberships of all facts are initialized to infinite except facts belonging to the fact layer 0, which is given 0 as layer memberships. When a fact f gets its layer membership set, all actions of which f fulfill a precondition get incremented on their counters. For an action a, if the counter reaches the total number of its preconditions, it is put to a list of scheduled actions for the current layer. Based on each established fact layer i, the layer memberships of all actions that scheduled in this step i are set to i, and their add-effects are updated to the scheduled facts in the $i+1$ layer. The newly scheduled facts have their membership set as $i+1$. The process continues until all goals have a finite layer membership updated. The detailed plan extraction procedures is present in the Algorithm 1.

Algorithm 1 Relaxed Plan Extraction

for i:= 1,...,m **do**
  $G_i := \{g \in G | layer - membership(g) = i\}$
end for
for $i := m, ..., 1$ **do**
  for all $g \in G$, g not marked TRUE at time i **do**
    select an action a with $g \in add(a)$ and layer-membership $i-1$, $a's$ difficulty being minimal
    for all $f \in pre(a)$, layer-membership(f) $\neq 0$, f not marked TRUE at time $i-1$ **do**
      $G_{layer-membership(f)}$:= $G_{layer-membership(f)} \cup f$
    end for
    for all $f \in add(a)$ **do**
      mark f as TRUE at times $i-1$ and i
    end for
  end for
end for

In the relaxed plan extraction algorithm, each goal g is assigned in a goal set G_i where i is the number of the first layer that the goal g occurred. Then, there is a backward extraction from the end to the initial layer. At each layer i, an action is selected for each fact in the goal set G_i, based on the minimal difficulty heuristic. Also, the preconditions of these actions are assigned in the corresponding goal set G_{i-1}. Once an action is selected, all of its positive effects are marked true for both layer $i-1$ and i. By doing so, facts that are newly marked as TRUE will not be considered during the searching in this layer i. Besides, the precondition of each selected action is not considered as a new goal, which significantly cut off unnecessary extraction branches.

Enforced Hill-climbing

Similar to hill-climbing search, the enforced hill-climbing also performs a heuristic forward search, evaluating practical actions and choose an action to continue, shown in the Algorithm 2. The enforced hill-climbing search uses a breadth-first search to find the best successor to the state in each step. Under a particular condition that the breadth-first search in one iteration fails to search for a better state with smaller heuristic cost to the goal, the enforced hill-climbing search will break and fail to find a solution, called *Dead End* stated in [22]. If a planning task is *dead-end free*, the problem is solvable, and the enforced hill-climbing search will find a solution. In conclusion, the enforced hill-climbing uses hill-climbing search as a systematic search method and breadth-first search as a local search to find the optimal local solution in each step.

Algorithm 2 Enforced Hill-Climbing

initialize current plan to empty plan $\langle\rangle$
$S := I$ (the initial state)
$h(I) :=$ heuristic distance from initial state to goal
while $h(S) \neq 0$ **do**
    perform breadth first search for a state S' with $h(S') < h(S)$
    if state not found **then**
        output 'Fail', break
    end if
    add the actions on the path to S'
    $S := S'$
end while

Helpful Actions
Helpful actions are selected as a set of promising successors to a specific search state. This method is derived by checking the relaxed GraphPlan, and in any planning state, the most promising actions are those selected in the first step in the relaxed plan. These actions that are filtered out seem to be helpful. By doing so, some searching branches are cut down to increase searching efficiency.

2.3 Hierarchical Mission Planning

For a problem-solving task, centralized planning techniques have the advantages of easy implementation and no inter-communication among different processors. The disadvantage of centralized planning is also evident: the cost of generating a detailed and complete mission plan using only one centralized planner could be very high due to the complexity of the mission and repeated planning for dynamic environment and changing execution state. In [10], the use of a centralized planner is generally incongruous to develop distributed-problem-solving systems where a network of sensors, controllers and executors working together to solve a problem.

On the other hand, a planning task could be decomposed into the planning of every subgoal in the task. Local processors only plan for individual subgoals. The advantage of this planning-task decomposition technique is less complicated and automatic plan decomposition and distribution to different processors. The loss of a subgoal in one processor does not fail the complete plan, but the mechanism for coordinating and generating the allocation of subgoals has to be developed. Communication and interaction among processors should also be considered cautiously and thoroughly. To balance the centralized planning costs and the cost of subgoal coordination, Leslie P. Kaelbling and Tomas Lozano-Perez [10] proposed NOAH in which a hierarchical planning technique is applied as a middle-ground approach.

Algorithm 3 Hierarchical Planning

procedure DO($job, world$)
  if type(job) == PRIMITIVE **then**
    EXECUTE($job, world$)
  else if type(job) == SUBTASK $\wedge\neg$ holds($job.fluent, world$) **then**
    Do($job.refinement(world)$)
  else if type(job) == SEQUENCE **then**
    for $task \in job.tasks$ **do**
      DO($task, world$)
    end for
  else
    type(job)== STATE
    p $\leftarrow$ plan(STATE([],$world$), job)
    if p: **then**
      DO($p, world$)
    else
      Raise failed
    end if
  end if
end procedure

The hierarchical planning system solves the complete planning problem firstly at a high level of abstraction by generating a sequence of abstract actions or sub-tasks that fulfills all the mission goals. Committing to the abstract plan, the first sub-task is planned and solved by sub-planner. The whole hierarchical system [23]: Committing to the sub-plan of the current sub-task, refinement or further planning of generating sub-sub-tasks is carried out on until all sub-tasks are decomposed into primitive actions. Algorithm 3 presents how a mission plan is generated based on the mission requests (goal states). In the Algorithm 3, *.fluent* denotes the desired effect of the current *job*. Primitive action denotes an action that requires no further symbolic planning(task planning) but allows refinement or planning in the geometric domain. The hierarchical task network planner searches for a plan that accomplishes the task network via task decomposition and conflict resolution [12]. In [7], a C-SHOP hierarchical planning

algorithm in nondeterministic domains is proposed that can coop with incomplete and uncertain information.

In some cases, an action is regarded as primitive while in other cases, it is non-primitive. For example, *Go(p1,p2)* could be regarded as a primitive action if only path planning is further required to achieve this action. On the other hand, *Go(p1,p2)* could also be a non-primitive action if, by definition, *Go up steers*, *Go down steers* and *Go within this floor* should be further planned for the Go action.

For a specific planning problem, the hierarchical structure tree is usually not uniform because it may have branches of unequal depth. Thus, the hierarchical planning process is framed in terms of doing jobs, dispatching on the type of jobs to be done in [23]. Jobs could be classified as the following types:

(1) Primitive: As stated above, the primitive action requires no further symbolic planning but is allowed to do further geometric-only planning. This means that a primitive action is defined as an executable action and does not need any refinement from the hierarchical planning system. For example, *Move(A, B)* refers to a primitive action of Moving an object from location A to location B.
(2) State: If the job is a symbolic description of the desired state, it is classified as a *State*. For example, *monkey(on tree)* is a state showing that the monkey is now on the tree.
(3) Sequence: A list of subtasks in sequence. Usually, by doing *plan(STATE([], world), job)*, *p*, the generated job, is a *Sequence*.
(4) Subtask: A subtask usually requires more refinements to determine further its property. A subtask can be refined into a primitive job, a job sequence or a waiting state for planning.

The hierarchical planning system has many advantages, including increasing reliability and flexibility, enhancing real-time response and applicability of complex missions. In real-world problem-solving tasks, solving centralized planning for complete tasks is intractable due to the long horizon and is also expensive in implementation. The main intention of introducing hierarchy is to apply a more efficient and intelligent method for planning a task, although the hierarchical planning result is probably not optimal. By doing so, all planning problems in the hierarchy are short-horizon and thus efficient in searching for a plan. Since each planning problem's cost increases exponentially with the size of state space, a hierarchical planning system is expected to be applied in large state spaces to avoid high planning cost and promise planning efficiency.

Chapter 3

Conditional Mission Replanning

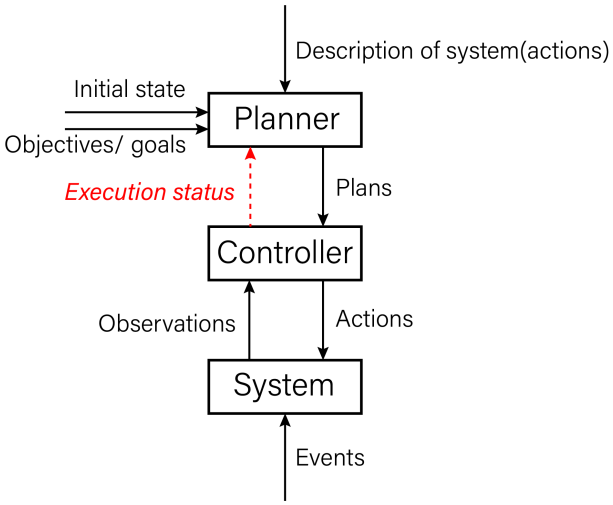

Figure 3.1: Conceptual model of the mission planning and control system (B)

Chapter 2 presents a literature study regarding automated planning and introduces some planning methods with advantages and disadvantages. In classical planning, the planner assumes that the initial world does not change while planning and execution [42]. Thus, in a classical planning problem, the planner will not react to the changing representation of the real world or system errors. This consequent limitation has to be compensated during the design of a plan-based mission control system. Shown in Figure 3.1, a planner can make correct plan only if it keeps tracking of the execution status.

To utilize the classical planner in the mission control system, we need to consider the validity of planning results for the dynamic environment and the changing situation of vehicles. This chapter states the classification of problems and situations that requires replanning of the mission and presents three replanning algorithms for mission control.

3.1 Conditions for Replanning

Planning is a high-level reasoning activity, and predefined plans are commonly used for UUV missions [8]. Nevertheless, the off-line planning result cannot promise completion of the mission when the vehicle is operating in an unknown and dynamic underwater environment. Under some circumstances, the previously applied plan does not work for the current situation. Then, re-planning is required to generate a new plan. The plan might fail to guide navigation and intervention due to several reasons. Some typical reasons and commonly occurred events are introduced by [19] and [27]:

(1) Execution Failure: Suppose the ROV has restarted many times but cannot communicate with the onboard station when navigating on the seabed. Then the failure mode exits the loop, and the ROV should execute return command directly for safety consideration. Another example is that the ROV performs target localization tasks but cannot find the target after a long period. The ROV will experience a warning of time out, terminate the mission, and return to the sea surface;
(2) Unexpected Events: Suppose that the ROV is wrapped by its umbilical tether, where a warning signal will be posted then. It is advised to switch to manual control in order to solve this problem;
(3) Incorrect Information: Suppose that the inaccurate IMU or DVL data results in the wrong measurement of ROV location, the ROV is far away from the desired path and cannot approach the structure of interest. For recovery, a re-calibration of sensors is required, and task replanning should be conducted;
(4) Partial Information: The altitude of ROV cannot be measured before it gets close enough to the seabed. Therefore, in the most descent stage, altitude information is lost. Once the seabed is detected by sonar, a correction of estimated depth and altitude will be carried out.

Apart from the reasons stated above, the budget difference stated in [27] is also a potential reason for replanning, which denotes the difference between the estimated cost and that of real execution. If the difference exceeds the predefined threshold, the plan might be invalid, and thus, replanning for a new mission plan is required. In this thesis, replanning mainly tackles the effect of ignorance and uncertainty, as stated above. Inaccuracy and noise during mission execution are not considered as an essential issue that will not trigger replanning.

3.2 Replanning Algorithm

Repeated planning and replanning are necessary during the mission control and execution to coop with the stated conditions. In [25], two steps for contingency handling are suggested. The first step is diagnosis and the second step is reactive replanning. Regarding the interaction between mission planning and acting, Run-Lookahead, Run-Lazy-Lookahead and Run-Concurrent-Lookahead are three online planning algorithms. Comparisons among these three algorithms are discussed in [19].

(1)Run-Lookahead: Replans when encountering failure modes (Section 3.1) and after finishing each action. Algorithm 4 shows how the Run-Lookahead algorithm do replanning. After finishing

each action, it replans and only performs the first action from the plan. This algorithm works well in the unknown environment.

Algorithm 4 Run-Lookahead

```
procedure RUN-LOOKAHEAD
    s ← abstraction of observed state ξ
    while s ⊭ g do
        π ← Lookahead(Σ, s, g)
        if π = failure then return failure
            a ← pop-first-action(π) ; perform a
        end if
    end while
end procedure
```

(2)Run-Lazy-Lookahead: Replans when necessary. For ROVs, the Run-Lazy-Lookahead algorithm is not recommended since it executes the plan as far as possible and does re-panning only when the failure mode shows that the system fails to execute the plan. This algorithm is applied when re-planning is computationally expensive, or the planned action is robust enough to give desired outcomes. The Run-Lazy-Lookahead algorithm is similar to online planning.

(3)Run-Concurrent-Lookahead: Performs acting and repeated planning at the same time. This procedure is computationally expensive but reduces risks by applying new actions continuously.

3.3 Online Planning

In online planning, the system may need to start before a plan is found [19]. Thus, the planning method designed to run off-line has to be modified to run it online. For a complex task, planning decomposition is necessary for an attempt to limit the length of a plan and exponentially decrease the amount of search and time cost. By implementing planning decomposition, the planning system could coop with real-time changing of a dynamic environment, system warnings and other events. Two typical methods for planning decomposition are planning for subgoals sequentially (Section 3.3.1) and Limited-horizon planning (Section 3.3.2). The advantages and disadvantages are discussed and compared.

3.3.1 Planning for Sequential Subgoals

A possible planning-task-decomposition is achieved by decomposing the mission goal into a set of sequential subgoals [29]. By implementing this method, the subgoal given to the planner at a time does not have to be the goal of a complete task. Once the current subgoal has been achieved, the next subgoal in the goal sequence will be planned and then achieved. Potential advantages of applying this method are [10]:

(1)The requirement for communication with the whole world model is reduced because each subgoal planning might only require a partial world model.

(2) Failure in execution means failure to achieve that subgoal but not a complete mission.

(3) The planning problem is less complex than planning for the whole mission because it only plans for individual subgoals at a time.

(4) This method maximizes parallelism during the planning procedure.

However, the way of formulating these subgoals could be problematic. One possible way is to sort subgoals reasonably to achieve the final goals. Another way for subgoal formulation is by computing an ordered set of landmarks and choose the earliest one as a subgoal stated in [19]. Therefore, a drawback of this planning decomposition is that a distributed mechanism for intelligent subgoal allocation must be developed.

3.3.2 Limited-horizon Planning

In limited-horizon planning, lookahead replanning is called at each time, and the planner starts and searches for a solution based on the current state reported in [19]. The searching process runs until it either finds the complete plan or exceeds the predefined time limit/ cost limit. Then, the planner returns the found complete solution or partially completed solution. The advantage of applying Limited-horizon planning is: Strictly restricts the time cost to find a reasonable plan (partially or completely). The drawbacks are also evident: Optimality cannot be promised [40] and the performance of the planning is related to the settings of time limit.

Chapter 4

Mission Control Architecture

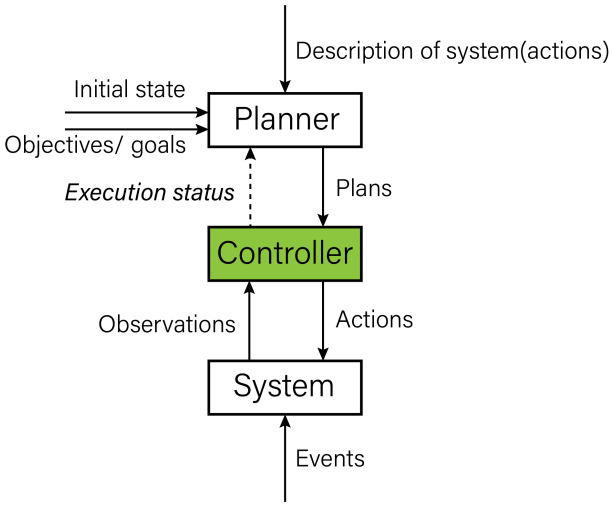

Figure 4.1: Conceptual model of the mission planning and control system (C)

This chapter presents a literature study on the mission control architecture, shown as the *Controller* in Figure 4.1. The study focuses on the work done in a master thesis [36]. The thesis proposes a semi-autonomous hybrid agent architecture for ROV interventions. The hybrid architecture is used as a base for the mission planning and control framework proposed and developed in this thesis. The work is also inspired by the behavior-based coordination mechanism presented in [8]. Section 4.3 presents four commonly used coordination mechanisms used in functional reaction.

4.1 Autonomous Control Architectures

The mission control architecture is a high-level control system including reasoning, planning and guiding, and is one of the most critical components for autonomous robotics intervention. The architecture has the function of making plans, generating mission command under control laws, monitoring the situation for a given mission. A control architecture should also be able

to detect errors and handle emergencies. In [11], the 'agent' is defined as conceptualizing the different aspects of intelligence by exploring biological and cognitive sciences for insights in intelligence. One intelligent mobile robot which integrates the main attributes of intelligence means perception, learning, planning, reasoning and communication. Following this definition, an agent architecture should be self-contained and independent. It has its brain and can interact with the outer world, sensing events and making corresponding changes [32]. Concluded in [35], three kinds of commonly applied agent architectures are deliberative architecture, reactive architecture and hybrid architecture. The design on an agent architecture should be able to perform the following functions: communicating with the other agents and the environment intelligently; performs planning, decision making and learning; the ability of fault detection and reasoning, and reacting to environment changes and other events.

The deliberative architecture is presented in Section 4.2, which is based on planning for the world model. Mission requests are usually in the form of a goal set assigned to the deliberative architecture. Deliberative architecture has a substantial advantage of planning and scheduling for a structured and highly predictable environment [8]. The weakness of this architecture is the inability to react to the dynamic environment and contingencies.

On the contrary, the reactive architecture discussed in Section 4.3 acts to deal with a non-structured and dynamic environment that strongly relies on sensing of the environment and responds quickly to contingencies. The reactive architecture is built with the definition of typical behaviors and coordination mechanism for prioritizing among behaviors in [8], as a consequence of which, the reactive architecture is also called behavior-based architecture.

The third kind of architecture is hybrid agent architecture combing deliberative architecture and reactive architecture together and use a control execution mechanism to make a selection between these two approaches. The hybrid agent architecture is the most commonly used one in AUVs and other autonomous vehicles reported in [8], which takes advantage of both deliberative and reactive architectures present in Section 4.4.

4.2 Deliberative Architecture

Deliberative architectures strongly rely on the world model and planning process reported in [35]. In the deliberative architecture, a mission is specified, planned based on the known environment, and these planned actions are executed sequentially to achieve the mission's goal. Typically, deliberative architecture performs predictable behaviors but is unable to react quickly to errors and emergencies. Reactivity mechanisms in deliberative architectures are so restricted that the control architecture is not able to adapt to unknown or dynamic environments. The deliberative architecture can be classified as *Hierarchical architecture* and *Centralized architecture*.

Hierarchical architecture decomposes a mission into sub-missions such that the plan of a mission is executed as a sequence of primitive actions respective to relevant sub-missions. The control flow goes from upper layers where a plan is generated based on the mission request and world model to lower layers. Each action of that plan is implemented and executed sequentially by

controllers. A deliberative control architecture was proposed and organized in three hierarchical levels [2]: planning level, control level and diagnostic level. The diagnostic level that performs mainly in fault detection and plan recovery.

Centralized architecture is organized as a set of expert modules, each of which has a specialized function and is communicated with others via the central control module. In [38], a task control architecture(TCA) is proposed consisting of a generic central control module and a set of expert modules for specialized function and purpose. Reactivity mechanisms are also considered in the task control architecture by implementing monitors for specific conditions and exceptions' handling.

4.3 Reactive Architecture

In Section 4.2, deliberative architectures are developed based on planning and recognition of the world model. A mission is either predefined as a sequence of behaviors or automatically planned as a sequence of actions by mission planners. The control system of the vehicle then executes the consequent behavior or action result. In contrast, the reactive architecture stated in Section 4.3 performs fast and direct response to deal with non-structured and dynamic environments [8]. Under some circumstances, the reactive behavior of an agent is defined as a predefined function from sensing to the reaction. Fuzzy control systems are commonly used for such a behavior-based agent. In [32], a purely reactive agent architecture is introduced, from which the agent responds directly to their environment without reference in their memory.

There are two main procedures of designing a behavior-based architecture. The first one is deciding how behavior is defined and when it should be activated. A behavior may be defined using rules, procedural representations and potential field variation mechanism proposed by [35]. Different behaviors might be defined in different ways. A simple way to activate a behavior is defining a safe threshold of relevant sensor data or other signals. If the current data exceeds the safe threshold, the corresponding behavior should be activated.

The second procedure is considering which coordination mechanism could be used to choose a global behavior among all activated behaviors. A mechanism is needed to mediate all reactions suggested by behaviors that respond to the current sensory input and to formulate one action to the actuators. An advantage of behavior-based reactive agent architecture that under the design from the developer is [32]: It is possible to prioritize a specific behavior or evaluate the significance of behaviors and rule accordingly. In reactive architecture, the coordination mechanism could be very complicated. Four of the most suitable reactive architectures for AUV controls are [8]: Schema-based approach, Subsumption, Process Description Language, and Action Selection Dynamics. Subsumption and Action Selection Dynamics are competitive methods that only one behavior is selected as the output. The rest two are cooperative methods that a superposition of behavior effects becomes the final output.

4.3.1 Competitive Methods

Subsumption forms a separate layer that decides which behavior should be finally executed. In the Subsumption architecture, behaviors are organized hierarchically. Behaviors with higher hierarchy can suppress outputs of lower behaviors. Individual behaviors run and generate individual goals concurrently. This subsumption architecture constitutes the coordination mechanism. For example, *Cable Tension*, *Obstacle Avoidance* and *Go To* are three behaviors, the first two of which should have a higher priority of the last one. Also, for safety consideration, the situation of *Cable Tension* is expected to overlap that of *Obstacle Avoidance* since the reactive action for cable tension is either station keeping or backtrack to the original waypoints. These possible reactive actions for the contingency of cable tension, to a great extent, release the emergency of collision with obstacles. Fig. 4.2 shows how the subsumption method works for the three behaviors.

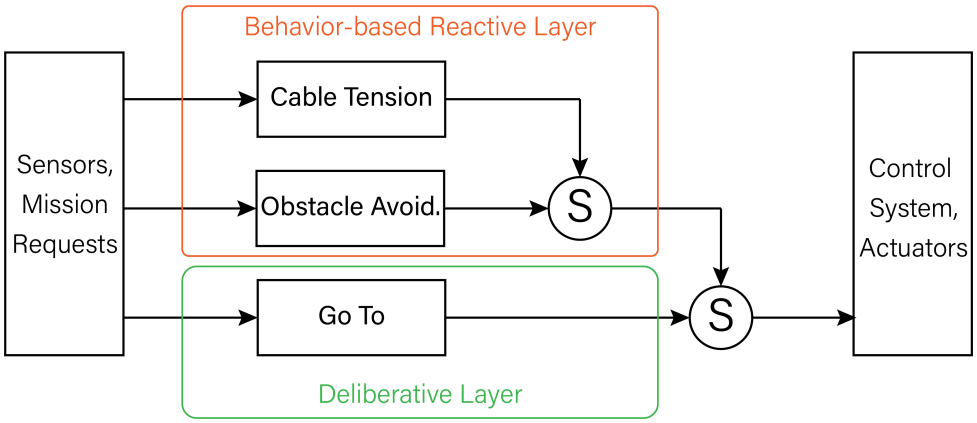

Figure 4.2: Coordination method: Subsumption

Advantage of applying this method as coordination mechanism are robustness and easy tuning of behaviors, while the disadvantages mainly come from the characteristic of competitive coordination method: non-optimal trajectories and difficulty on designing the subsumption laws between different behaviors.

Another competitive method is Action Selection Dynamics. This method is very complex to be implemented, and the concept is similar to STRIPS introduced in Section 2.2.1. The energy concept is additionally applied that the behavior that satisfies the condition list and has the maximum activation energy will be executed in the end.

4.3.2 Cooperative Methods

In Figure 4.3, the schema-based approach formulates the coordination mechanism based on a schema concept and weighs the summation of all schema outputs. Behaviors are defined and expressed with a schema library. Each schema responds proportionally to the sensor data, and a gain value is assigned to each schema that higher gain value denotes more priority and vice verse. The coordination function calculates the summation of weighted schema outputs for the

current state. The coordination result for all state constitutes a potential field. Fuzzy logic is commonly used in this schema-based coordinate mechanism [8].

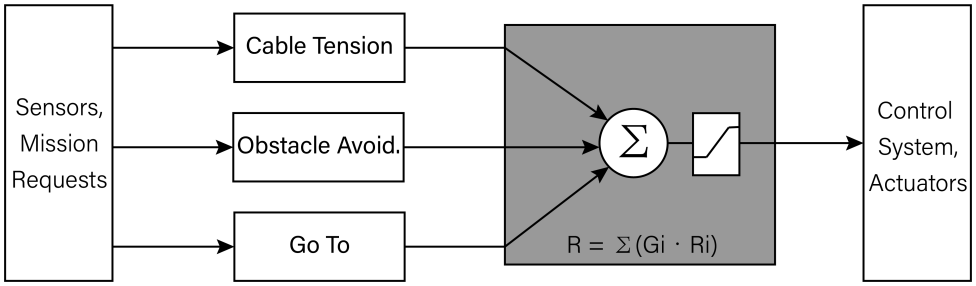

Figure 4.3: Coordination method: Schema-based approach

Fuzzy Logic

Different from the simple on-off switching coordination logic, the fuzzy logic is proposed as a useful mechanism that provides a flexible method to encode behavior arbitration strategies. Fuzzy logic can be regarded as an extension of Boolean logic but replaces $True$ or $False$ in Boolean with truth-value ranging from 0 to 1. This concept is firstly proposed in [43]. A fuzzy logic problem is composed of two subproblems: how to decide the activation situation of behavior at each moment (Behavior arbitration) and how to coordinate results from activated behaviors to carry out a final suitable command to the actuator (Command fusion). The advantages of applying fuzzy logic in behavior arbitration are enabling smooth transitions between behaviors and expression of partial activation of behaviors. The most popular approaches of command fusion are based on vector summation that commands from behaviors are formed as a force vector. By vector summation, the coordinator responds to the force and makes final decisions.

Context-Dependent Blending(CDB) is a general form of behavior combination. The central concept applied in CDB is summed up by [37]:
(1) Based on its function, each behavior is assigned with preferences (similar to weighting a behavior).
(2) A context of activation is assigned to each behavior to determine under which situation it should be activated.
(3) The preferences of all behaviors, weighted by a true value of respective contexts are fused to form a collective preference.
(4) One command is finally selected based on collective preference.

In Process Description Language shown in Figure 4.4, the process is an important concept that represents behaviors receiving sensory information and generates a command if necessary. Each process has a particular influence on some variables, and PDL reacts based on its derivatives. Hence, the control loop should be much faster than the system running speed to avoid instability.

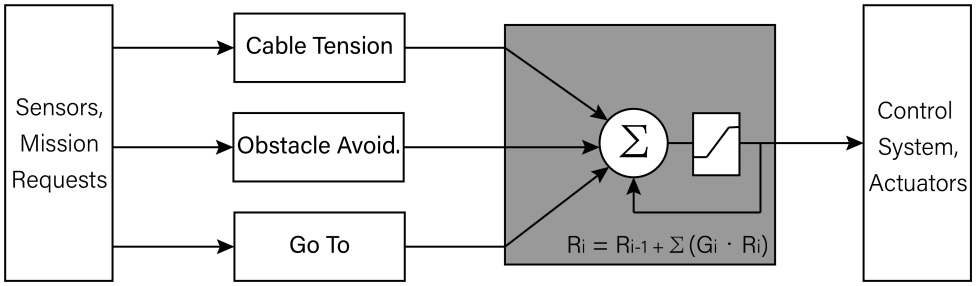

Figure 4.4: Coordination method: Process Description Language

4.4 Hybrid Architecture

As stated in Section 4.2, deliberative architecture performs mainly planned and scheduled tasks but has limited performance on handling unexpected events and emergencies. In contrast, reactive architecture stated in Section 4.3 can respond fast to contingencies as long as the reactive behavior is well designed such that effective action is activated to handle this situation. A hybrid architecture becomes an intermediate approach to overcome the weakness of deliberative and reactive architecture currently used by actual researchers in [35].

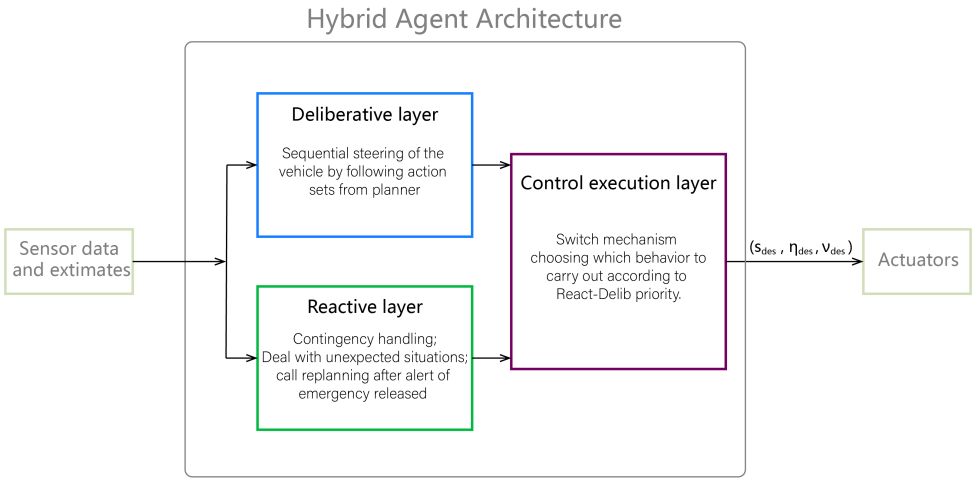

Figure 4.5: Hybrid Agent Architecture

The hybrid architecture consists of three layers: deliberative layer, control execution layer, and functional reactive layer. This architecture is developed by evaluating deliberative and reactive activities and assigning priorities on received behaviors in the control execution layer to give a final output to a low-level control system. The deliberative layer in the hybrid architecture performs predictable functions, and the reactive layer is used for responding to sensed environmental changes and functional errors of the system. Behaviors in both deliberative and reactive architectures act concurrently and independently, and the generated outputs are channeled into a

coordinate mechanism to produce an appropriate response finally. This coordinate mechanism is named as control execution layer. This layer performs management of action output from the deliberative layer and reactive layer, sorting the priority of actions, coordinating, and generating the final action to be executed by the vehicle.

It is concluded in [36] that the hybrid control architecture is a balance between carefully planned and scheduled actions and quickly-responding actions. The deliberative layer under this architecture ensures efficiency and optimality of the ROV performance. The the reactive layer accounts for changing environments and unexpected events. Therefore, the deliberative layer guides the vehicle to accomplish mission requests, and the reactive layer assures safety during mission execution reported in [17].

Chapter 5

Mission Planning and Control system for ROVs

Based on the literature study of the mission planner, replanning algorithms, and mission control architecture presented in Chapter 2, Chapter 3 and Chapter 4, this chapter proposes a hybrid mission planning and control architecture for autonomous ROV interventions. Section 5.1 presents the overview of the hybrid mission planning and control structure. Section 5.2 describes how the layered planning and control system is implemented in deliberative layers to achieve a hierarchy of missions. Section 5.3 presents the development of typical reactive behaviors and how the coordinate mechanism is designed under the reactive layer. In Section 5.4, a simple design of priority assigned to behaviors from deliberative and reactive layers are presented.

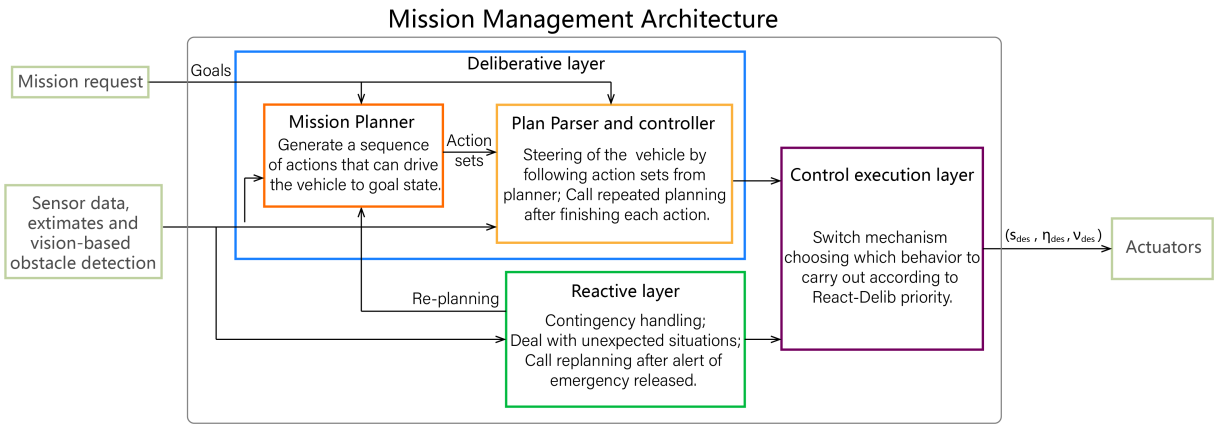

Figure 5.1: Mission Planning and Control Architecture

In the proposed hybrid architecture, the deliberative layer in Section 5.1.1 and the reactive layer in Section 5.1.2 generate desired behaviors concurrently. Two desired behaviors are transmitted to the control execution layer in Section 5.1.3, selecting one with higher priority by following its coordinate mechanism and sending it to the actuator for further execution.

5.1 Structure of Hybrid Mission Control system

The plan-based mission control system controls the ROV to perform subsea operations, guiding the ROV to approach and localize the structure of interest and perform simple observations. Figure 5.1 is a diagram of the hybrid control architecture. The deliberative layer mainly performs plan-based behaviors. The mission planner proposed in Chapter 2 and the supervision algorithm proposed in Chapter 3 lies in the deliberative layer. The reactive layer deals with contingencies, typical reactive behaviors of which are discussed in Section 5.1.2 and Section 5.3. The output from deliberative and reactive layers are actions together with desired positions in the UTM framework and desired velocity in the body framework. The developed coordinate mechanism of prioritizing actions from deliberative and reactive layers is presented in Section 5.1.3 and Section 5.4. The final selected action and corresponding desired position and velocity are transmitted to the low-level actuator for further execution.

Parameters	Description	Reference coordinate framework
$action$	Desired action	-
n	Desired north position	UTM
e	Desired East position	UTM
d	Desired depth	UTM
ψ	Desired heading	UTM
u	Desired surge velocity	Body
v	Desired sway velocity	Body

Table 5.1: Parameters defining the action, desired position and velocity.

The deliberative control provides the achievement of high-level goals, while reactive control ensures adaptation according to the dynamic world. The absence of either control could result in the vehicle being unable to satisfy both the goal and the adaptive mission objectives.

5.1.1 Deliberative Layer

The deliberative layer executes planned and optimal actions under safety assumption, which is running the ROV to the goal state when no emergency occurs, and no system error appears. The main work behind this layer lies in the definitions of each action and corresponding states, the design of planner, and supervision algorithms. A case structure is developed for the activation of corresponding actions by following the instruction of the planner. The deliberative control consists of temporal and spatial logic expressions to determine whether an action has been finished. For example, when *Transit* action is activated, the distance between estimated ROV position and desired position should be less than k such that this action is regarded as 'Finished'. These logic expressions are simple but proven to be practical.

This layer consists of five predefined actions: Launch, Descent, Transit, Operation and Station Keeping. In the deliberative layer in Section 5.2, the planning algorithm searches for actions that can drive the vehicle from initial state to the goal state, generating 'paths' as a set of necessary actions. The mission planner receives the request from operators, searches for a matched mode, and a sequence of actions that can achieve the goal. A 'supervision' algorithm is proposed, which is a parser and controller in the deliberative layer in Fig 5.1. Re-planning might be activated based on the situation awareness and operator command.

5.1.2 Reactive Layer

The reactive layer is mainly designed and used for applying quick response actions to unpredictable situations. Unlike states in the deliberative layer, the reactive layer executes quickly responding actions, assuring ROVs' safety. This layer mainly performs contingency handling and plan modification due to world dynamics, including cable tension, obstacle avoidance and drift of ROVs. Compared to the deliberative layer, the reactive layer has a higher priority, which means that the control execution layer in the mission management system firstly considers the request from the reactive layer. If no request is received from the reactive layer, the execution layer will take requests from the deliberative layer.

5.1.3 Control Execution Layer

Generally, the control execution layer evaluates the two architectures' output behaviors and will transfer inputs from the reactive architecture to the executor once the reactive layer shows alarming. Only one deliberative action is activated at a time in the deliberative layer. Also, a coordinate mechanism is applied in the functional reactive layer to select the behavior with the highest priority among all activated behaviors under this layer. The control execution layer is thus organized as a selection of output between the deliberative layer and the reactive layer. This thesis is responsible for deciding which layer and which action is activated and summing up the necessary parameters required by the control system. The hierarchical classifications of behaviors are applied in this layer, giving the reactive layer a higher priority than the deliberative layer.

5.2 Mission Planning and Control under Deliberative Layer

Figure 5.2 presents the proposed ROV mission planning and control architecture, which is organized as a hierarchical system. The upper layer is planning and control of missions for global navigation. Candidate actions in the global planner are symbolic actions: *Launch, Descent, Transit, Sonar Tracking, Camera Tracking*, and *Operation*. Before a plan is found, the mission control system performs station keeping and waiting for the planner's command. The second layer is developed for the local *Operation* and introduces a sub-planner under this action.

An A* planner for ROV homing and docking is laid in the third layer of operation refinement. This path planner is carried out by Signe B. Moltu and integrated into the mission management

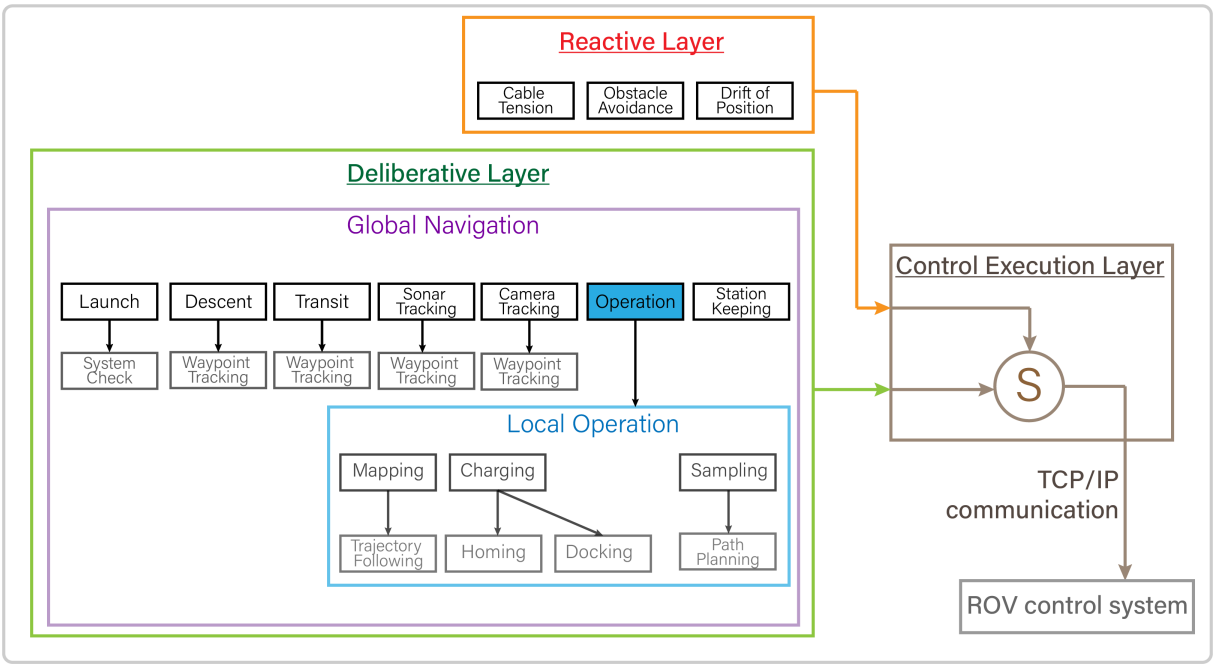

Figure 5.2: Layered structure of the Mission Planning and Control System

system to perform *Charging* and *Sampling*. The path planned for *Charging* consists of two parts: Homing and docking. The path planning is mainly carried out under homing. For the *Mapping* action of local *Operation*, a simple trajectory tracking algorithm is developed.

For a given mission request from a human operator, the global planner firstly search for a reasonable plan and send it to the mission controller for further monitoring and execution. If the specific action is not primitive, more refinements or sub-planning will be carried out to adjust and plan for that specific action until all actions are primitive. The proposed layered plan-based mission control architecture has evident advantages: reducing the planning time by decomposing a complex task in different hierarchies.

What is more, refinement on symbolic actions might be required. As a sub-task of *Charging*, path planning for homing receives the command from mission planning and find an optimal or near-optimal path from the starting point to the goal point. Also, based on the calculated optimal path, motion planning should take acceleration, velocity and path curvature into account and then design a reference model(desired motion) for the vehicle to follow. To properly design a mission structure, a detailed definition of all modes and relevant agencies should be done based on the understanding of possible missions, risks and characteristics of ROVs. Lack of some pre-definition may lead to faults and dangers.

5.2.1 Mission Planning and Control for Global Navigation

The main function of global navigation is to steer the vehicle to a target position, which is also the starting position of *Operation*. During the execution of *Operation*, the global planner halts at the state *Operation*. Sub-planner and sub-mission control for *Operation* (Section 5.2.2) decides which specific action should be carried out. After the vehicle finishes *Operation*, the planner for

global navigation will guide it to the ending location of the mission. *Sonar Tracking* and *Camera Tracking* performs target tracking once the relevant sensor detects the target.

Candidate actions for global navigation are symbolic: *Launch* checks the system connection and sensor status, *Descent* performs vertical motion and *Transit* performs horizontal motion with depth control, *Operation* waits for sub-planning result to execute planned actions.

Launch

The *Launch* is an action to initialize the system. Launch action is necessary for missions. The purpose of this state is to reserve a period to verify that the mission control system is ready to run. After this action's first performance, the system is regarded as 'Started', and Launch action will no longer appear in the action sequence after replanning unless the mission control system is terminated and restarted again. The 'Elapsed time' for this action is selected for 10 seconds. After 10 seconds' self-check, the Launch action will 'Finish' if all sensor works, all communication with onboard station remains connected, and no error occurs during this period. Otherwise, a time reset will be activated, and another 10 seconds' self-check will start. The next action will be activated, referring to the planned action sequence and supervision situation.

Descent

\{}textit{Descent} drives the vehicle from present depth to the desired depth. During the execution of this action, the vehicle moves downwards or upwards to the desired position. To minimize control parameters, the north/east position and the heading direction remain unchanged. The vehicle moves with a constant heave velocity within a thrust limitation. When the vehicle is approaching the desired depth, heave speed will gradually reduce to a safe level to avoid sudden changes and a sharp turn of position and velocity.

Usually, the control system is designed to perform depth control instead of altitude control. Since the Doppler Velocity Log(DVL) sensor cannot perform accurate detection of the seabed before the vehicle reaches an altitude of 30-40 meters, there is no measurement from the DVL for the situation that the vehicle is above an altitude of that threshold. Under some special conditions, the desired depth is not reachable, such as the condition that the altitude sensor shows alarm, but the vehicle has not reached the desired depth. In this case, the vehicle will keep a safe altitude and perform altitude control until finishing this action or altitude alarm is removed.

A drawback is that no sensors are pointing downwards, such that obstacle detection and avoidance becomes a severe problem today. To achieve full autonomy in the future, adding sensors in vertical directions is a promising way to tackle the current drawback. On the other hand, the length of cable is also a key factor to the free range that a vehicle can reach beyond the depth of seabed underneath sea level.

Transit

Transit corresponds to a state with horizontal motion with either a target tracking or a path following mode. The purpose of this state is to automatically steer the vehicle to the desired

position with a constant depth. The heading direction is designed to be along the forward-moving direction. A distance of tolerance is set as one meter around the desired position. When the vehicle moves close enough to the desired position, the waypoint will be updated to the next one. If all waypoints have been visited, the action terminates.

During the *Transit* state, the vehicle may detect the structure of interest by sonar or camera if it moves close enough to the target. Usually, the vehicle first receives a sonar signal of detection before the camera signal. If either sonar or camera signal is detected, the state will switch to *Sonar Tracking* or *Camera Tracking*. Also, an update of the SOI location will be sent to the mission planner, showing the newly detected target position. On the one hand, if sonar and camera tracking work, the vehicle will trace the target's detected location until it gets close enough to the target, determined by the distance tolerance. On the other hand, the vehicle might also reach the desired position without any signal detection. Then, a switch of action will also be performed.

Sonar Tracking

Sonar Tracking is an action where the vehicle uses sonar to detect and get access to a more accurate SOI location. From the description of *Transit*, sonar tracking works when sonar detects the target. However, this thesis does not include the processing of the sonar signal. The real sonar signal must be developed and applied to obstacle detection and target tracking to achieve full autonomy. Sonar is widely implemented in underwater vehicles for scanning of seabed and estimation of distance and relative angle. All objects in the sonar range will be detected. It is possible to detect, recognize, and locate the target based on the vehicle's motion and intensity of the acoustic reflections. In the simulation, the virtual data of the target position detected by sonar is set manually. Figure 5.3 shows how sonar and camera tracking affects the planning and control of missions.

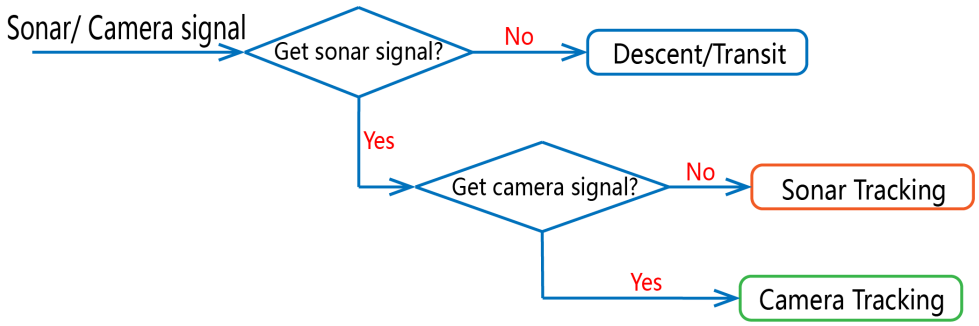

Figure 5.3: Action switch between Descent/Transit, Sonar Tracking and Camera Tracking

Camera Tracking

After sonar tracking is activated, the vehicle will keep tracking and move towards the target. When it gets close enough to the target, camera vision techniques are sufficient to be applied, and camera tracking is activated. During this action, the vehicle is located within a close distance

to the target such that a smooth trajectory and a low velocity should be considered. In the final states, the vehicle should be sufficiently close to the object to perform its main operations.

Operation

Compared to the previous states which possess fixed mode for execution, this state is more flexible and is highly dependent on the type of operations. ROVs could execute a large variety of missions in an underwater environment, such as seabed mapping and sampling. Therefore, this action should be further divided into sub-actions to describe different missions distinctly. For some missions like seabed maintenance, manipulator's arms are needed, while sensor-based navigation is the topic for other missions. Therefore, a refinement of this action should be developed. This thesis proposes some typical observation-based operations (Section 5.2.2).

Station Keeping

Before starting the planner and mission control system, the vehicle performs station keeping. In the action output from the mission planner, there is no station keeping action. Nevertheless, a station-keeping action is necessary for mission control architecture because under some special conditions: performing station keeping is the safest choice. For example, if an error occurred when running the system, the mission should be terminated and do self-check. At this time, ROV cannot be freely released but executes station keeping.

Planning Settings

For a mission planner, typical missions and actions are defined. The search algorithm is then used to find the 'path' from the starting point to the goal condition. For ROVs, six typical states are launch, descent, transit, sonar tracking, camera tracking, and operation. In different missions, the main tasks and available actions are different. For example, during the descent task, there is continuously changed depth and altitude with proper descending speed, while in sonar tracking, the action becomes tracking a target by sonar. However, in all of these missions, collision avoidance is required, which refers to fast reactions under emergency.

In this thesis, a mission planner is developed using STRIPS. Available actions in this planner are Launch, Descent, Transit, and Operation. This planner is designed without including sonar tracking and camera tracking actions due to features of these two actions. The planner cannot decide when and where to perform tracking. It depends on whether the ROV sensor system detects the structure of interest. Therefore, sonar tracking and camera tracking is implemented in mission control architecture instead of the mission planner.

As mentioned in Section 2.2.1, to make a plan for a mission, initial states and goal states are inputs to the planner. In this thesis, the location of both ROV and target is decomposed into vertical and horizontal location. Thus, the initial horizontal location of ROV is $H1$. Moreover, the vertical target position is $V2$. These two definitions are the base for other definitions below. Then, the horizontal target position and expected goal horizontal location of ROV are:

Algorithm 5 Definition: horizontal target location

$hloc^{rov}_{init} \leftarrow H1$
if $dis(hloc^{rov}_{init},\ hloc^{tar}) < 1$ **then**
$hloc^{tar} = hloc^{rov}_{init}$
else
$hloc^{tar} = H2$
end if

where $hloc^{rov}_{init}$ denotes the initial position of the ROV in the horizontal plane. $hloc^{tar}$ is the target position in the horizontal plane. In Algorithm 6, $hloc^{rov}_{goal}$ denotes the goal position of the ROV in the horizontal plane.

Algorithm 6 Definition: horizontal goal position

$hloc^{rov}_{init} \leftarrow H1$
$hloc^{tar} \leftarrow$ horizontal target position
if $dis(hloc^{rov}_{init},\ hloc^{rov}_{goal}) < 1$ **then**
$hloc^{rov}_{goal} = hloc^{rov}_{init}$
else if $dis(hloc^{rov}_{goal},\ hloc^{tar}) < 1$ **then**
$hloc^{rov}_{goal} = hloc^{tar}$
else
$hloc^{rov}_{goal} = H3$
end if

Similarly, vertical locations of ROV and SOI are defined in Algorithm 7 and Algorithm 8. When defining vertical location, $vloc^{tar} = V2$ is set as the reference. Therefore, in the planner, the precondition of action Transit contains $vloc^{rov} = V2$. Initial and goal vertical locations are defined based on the vertical location of SOI.

Algorithm 7 Definition: vertical target position

$vloc^{tar} \leftarrow V2$
if $dis(vloc^{rov}_{init},\ vloc^{tar}) < 1$ **then**
$vloc^{rov}_{init} = vloc^{tar}$
else
$vloc^{rov}_{init} = V1$
end if

Algorithm 8 Definition: vertical goal position

$vloc^{tar} \leftarrow V2$
$vloc^{rov}_{init} \leftarrow$ initial vertical location of ROV
if $dis(vloc^{rov}_{init},\ vloc^{tar}) < 1$ **then**
$vloc^{rov}_{init} = vloc^{tar}$
else if $dis(vloc^{rov}_{goal},\ vloc^{rov}_{init}) < 1$ **then**
$vloc^{rov}_{goal} = vloc^{rov}_{init}$
else
$hloc^{rov}_{goal} = V3$
end if

5.2.2 Mission Planning and Control for Local Operations

Within the mission control system, a minimum of two mission planners is necessary, one of which is a generic control system, and the other of which performs specifically local operation. The generic control system will be hosted by the operation control system under the proposed architecture of the mission control system. Respectively, the generic state is *Operation*, featured by specific sub-states dependent on the type and progress of the operation. The generic mission control system and the operation subsystem are under the deliberative layer of the autonomy framework. At any time, as proposed in Section 5.1.2, the reactive layer has a higher priority of mission control.

The work is to design a planning and control module for actions under local operation. Three possible actions are *Mapping*, *Sampling* and *Charging*. A sub-planner is implemented to guide the vehicle to execute actions based on the current states and operation requests. The planner uses the heuristic search method to search for a near-optimal solution that fulfills the operation request. Since the vehicle is not equipped with manipulators at present, the designed actions are only related to observation behaviors instead of other operations such as installation and repair.

Obstacle avoidance is implemented in both the deliberative and the reactive layer. For actions that require path planning, such as *Sampling* and *Charging*, new detected obstacles are updated to the map. Re-planning is called to generate a new path that avoids both the known and the detected obstacles. For the other actions, a new waypoint is created based on the distance to the obstacle, the size of the obstacle, and safety parameters for avoiding obstacles in Section 5.3.

Mapping

Mapping is designed to perform setpoints following. Under this action, setpoints are set before starting missions and thus are 'fixed'. In this thesis, a simple path of mapping is designed and shown in Figure 5.4.

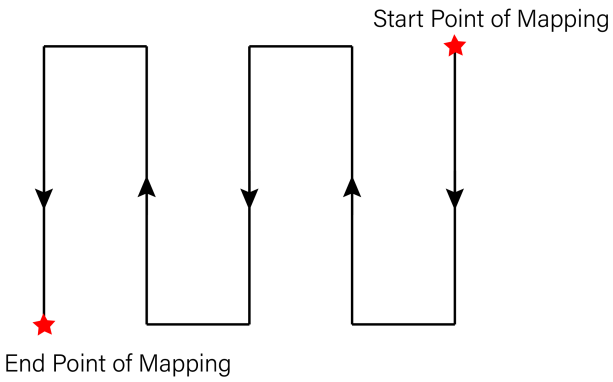

Figure 5.4: An Example of *Mapping* Path

The vehicle may encounter an obstacle during mapping. The proposed solution is to activate reactive obstacle avoidance (in Section 5.3) under the reactive layer of the control architecture, changing its heading and steering to the new waypoint for obstacle avoidance. After reaching the waypoint, the vehicle will continue its mapping task.

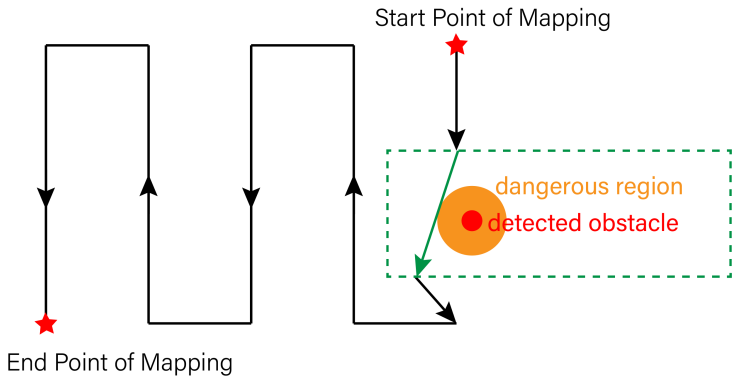

Figure 5.5: An example of *Mapping* with obstacle avoidance

Sampling

Sampling is a proposed action where the vehicle steers to a specific location by following the path planned by a path planner. The path planner is developed by another master student Signe B. Moltu, using A* heuristic search method to generate a reasonable and safe path by considering known obstacles. Most times, obstacle locations are unknown for the underwater environment. Thus, a replanning scheme for path planning is carried out under the action *Sampling*. The old path is replaced by a new one which takes into account known and newly detected obstacles.

Charging

Charging is an action that is implemented in local operations. During the execution process of the local operation, once the power system shows *LOW-BATTERY*, the planner for the local operation will replan and generate a new plan correspondingly. The first action of the new plan is *Charging*. A path planner for homing and docking is implemented under *Charging*, and the

vehicle will follow the generated path to the dock station and perform charging on it. When the power system shows *ENOUGH-BATTERY*, charging is then finished, and the sub-planner for the local operation will replan again to continue its requested task execution. Need to mention, the *Charging* action is not requested by the mission goals. However, if the vehicle lacks power, it has to perform *Charging* before continuing the requested tasks.

This action is designed for future ROVs. There is a trend that ROVs without cables are expected to replace the traditional ROVs equipped with cables. Thus, the new ROV is no longer exposed to large environmental forces and has no possibility of encountering failure mode in cable tension. To solve the energy supplement, subsea dock stations are proposed. Then the ROV can charge on the dock station during the mission execution, which significantly increases the working duration and ROVs capacity of executing high-payload operations.

5.3 Behaviors in Functional Reactive Layer

Three typical reactive behaviors are considered in the reactive layer: *Cable Tension*, *Obstacle Avoidance*, and *Drift of Position*. In Section 4.3, four coordinate mechanisms are introduced. In this thesis, the simplest and direct *Subsumption* method is adopted, which orders the priority as *Cable Tension - Obstacle Avoidance - Drift of Position*. When two or more behaviors are activated at the same time, the behavior with the highest priority among these activated behaviors is carried out by the reactive layer. For example, when the cable tension exceeds the threshold, and an obstacle is newly detected, the vehicle is to perform pre-defined reactive behavior for releasing cable tension instead of avoiding obstacles.

Cable Tension

ROVs are featured by an umbilical cable connecting the ship and ROV, transmitting both electricity and signals. The cable enables the system to perform high payload work and transfer data signals and videos with high resolution. The unlimited power supply via umbilical cable promises a long working time. Due to the existence of umbilical cables, ROVs are limited in spatial range. The umbilical cable is exposed to environmental loads such as current force and ocean animals. ROV also can get wrapped by umbilical cables from itself or other ROVs due to poor operation and environmental factors. What is worse, the umbilical cable may be damaged or even cut off by some structures. Therefore, when ROVs are executing missions, one of the essential sensors that can trigger emergency alarm is the umbilical tension.

The exceeding of cable tension often occurs when the vehicle moves, such as descent, transit, and sonar/ camera tracking. A simple way is to backtrack the vehicle's original trajectory to tackle this emergency. Waypoints for backtracking are recorded every ten seconds. In general, a new path for backtracking consists of ten waypoints from the last 100 seconds. In this thesis, the alarm from cable tension will not trigger immediate backtrack action but wait for five seconds to verify if the alarm is real or fake. In these five seconds, the vehicle executes station keeping command. If the alarm is released within the time duration, no backtrack actions will be applied. Otherwise, the vehicle will move back to its previous position and switch to manual control. Backtrack is

an excellent method to solve the situation when the cable is get wrapped by obstacles or other structures. Nevertheless, this method does not apply to all situations. Therefore, manual control by human operators is necessary to diagnose the events.

Obstacle Avoidance

Obstacle Avoidance is one significant reactive action that must be implemented in the reactive layer. To perform obstacle avoidance on ROVs, the detection of obstacles in an underwater environment using stereo vision or sonar is the prerequisite. Apart from that, an algorithm of deciding waypoints for obstacle avoidance is also required. After receiving information about the obstacle, the reactive layer computes a new heading direction for the vehicle, based on the present heading direction and the location of obstacles that appeared in the camera image.

The obstacle avoidance usually occurs during the state of Transit, sonar/ camera tracking and operations, which mainly occur when the vehicle is moving in the horizontal plane. Therefore, a method of heading change is used to deal with this situation. Figure 5.6 demonstrates how a new waypoint is calculated. When encountering a new obstacle, the direction of heading change is determined by β and ψ. β demonstrates the relative direction of the obstacle to the vehicle, and ψ is the heading angle of the vehicle. If heading angle ψ is smaller than β, the new heading angle is $\psi_{new} = \beta - \alpha$. Otherwise, the new heading angle is $\psi_{new} = \beta + \alpha$. The angle α is calculated based on the diameter of the dangerous region and the distance between the vehicle and the obstacle. D is the diameter of the obstacle, and e is a safety parameter. Thus, R is the radius of the dangerous region. L is the calculated length from the vehicle to the new waypoint. The position of the new waypoint is calculated based on the new heading angle and length L.

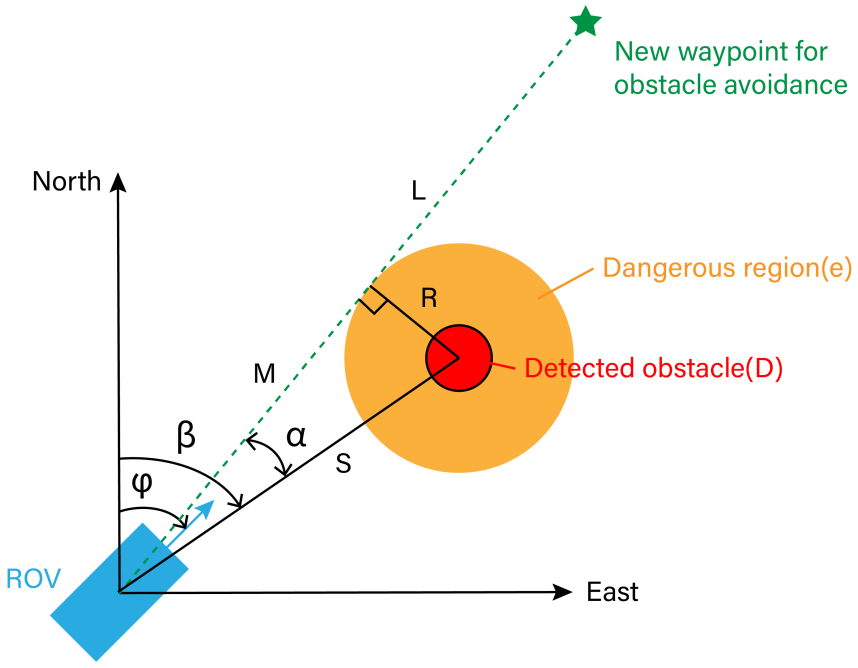

Figure 5.6: Obstacle Avoidance

$$R = \frac{D}{2} + e \tag{5.1}$$

$$S = \sqrt{(x_{rov} - x_{obs})^2 + (y_{rov} - y_{obs})^2 + (z_{rov} - z_{obs})^2} \tag{5.2}$$

$$\sin(\alpha) = \frac{R}{S} \tag{5.3}$$

However, under some particular circumstances, obstacles might also appear in the path of Descent. As stated in Section 5.2.1, no sensors and cameras are installed vertically and thus cannot perform the desired detection of obstacles in the vertical direction, which might be a big problem in further autonomous operation.

Drift of Position

The drift of vehicle position is also one possible reactive action in this layer. On the one hand, the execution of actions is greatly affected by the selection of supervision mode. For example, if the planner runs at all the time, a small drift of position will be compensated directly and immediately. Under this circumstance, a drift check of the vehicle position does not need to be considered. Nevertheless, if a lazy supervision mode is selected, the planner generates a new mission plan only when some command asks replanning. Then, a drift check is vital to promise the achievement and accuracy of mission execution.

On the other hand, the vehicle is exposed to some environmental disturbances, such as current and waves. In some extreme situations, fully actuated thrusters cannot keep the vehicle's position. The vehicle may drift to an undesired position within a short time, and thus the drift check is necessary. If the alarm of the drift check is triggered, a new mission plan will be updated.

The distance between the current position of the vehicle and the desired path of this action is:

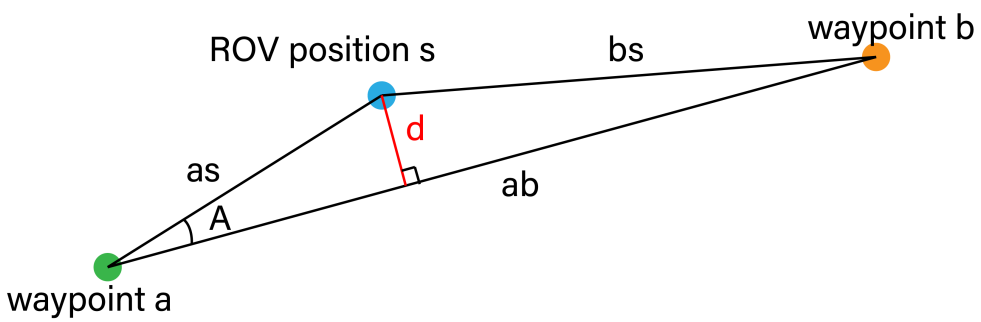

Figure 5.7: Drift of position

$$\cos(A) = \frac{ab^2 + as^2 - bs^2}{2as \cdot ab} \tag{5.4}$$

$$\sin(A) = \sqrt{1 - \cos^2(A)} \tag{5.5}$$

$$d = as \sin(A) \tag{5.6}$$

5.4 Control Execution Layer

Only one action is selected based on the planning result at a time under the deliberative layer. A *Subsumption* method is used to determine which behavior is to be carried out in the reactive layer. Therefore, the control execution layer needs only to choose between reactive and deliberative behaviors. Under this layer, any behavior from the reactive layer always has a higher priority than behaviors from the deliberative layer. Therefore, once a valid command from the reactive layer is sent to the control execution layer, the coordinate mechanism under this layer directly chooses reactive behavior and sends it to the actuators.

Chapter 6

Simulation Setup

This chapter gives an insight into how the ROV control system works in Labview. Section 6.1 presents the general composition of the ROV control system, including sensing system, mode selection, control system and supporting components. The main functions of Njord control system, Frigg graphical user interface and Verdandi simulator are introduced in Section 6.1.1, Section 6.1.2 and Section 6.1.4 respectively. Section 6.2 discusses how the proposed autonomous mission planning control system is incorporated in the ROV control system and how to encode the mission planning system from Python to Labview. Fig. 6.1 presents the main components and their inter-communication of the control system. For simulations, the Verdandi simulator is connected to the Njord control system. For field works, ROV Minerva is connected to the Njord control system.

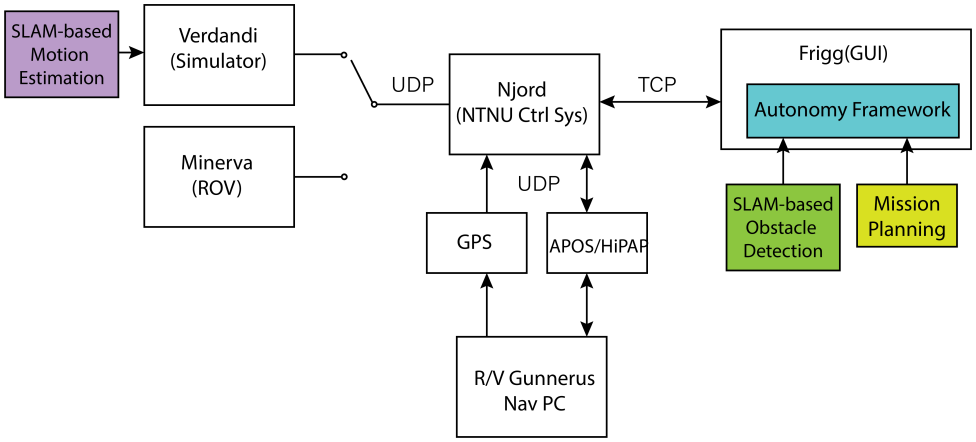

Figure 6.1: Interaction between systems

The work done in this thesis is laid in high-level mission planning and control shown in Figure 6.2. The developed mission planning and control system, referred to as Autonomy Framework in Figure 6.1, is the main contribution of this thesis. Relevant functions and organization of low-level sensing and control system of ROV Minerva are briefly introduced in this chapter.

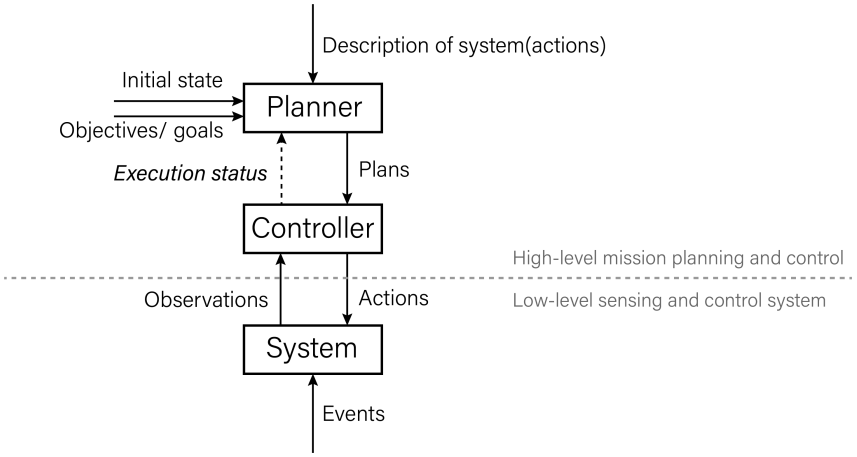

Figure 6.2: Conceptual model of the mission planning and control system (D)

6.1 ROV Control System

The Applied Underwater Robotics Laboratory (AUR-Lab) has been established by NTNU since 2009, enabling multi- and interdisciplinary research and education in marine underwater robotics, marine biology, chemistry [39]. The ROV Minerva is a SUB-fighter 7500 ROV, connecting to the Research Vessel (RV) Gunnerus by a cable. RV Gunnerus is equipped with a High-Precision Acoustic Positioning and underwater navigation (HiPAP) system 500, which is used to measure ROV Minerva location relative to RV Gunnerus.

From 2010 to 2011, a dynamic positioning (DP) and tracking system have been developed by a group of master students, Ph.D. candidates, and NTNU researchers. The established ROV control system in Labview consists of two main parts: Control System(Njord) and Graphical User Interface(Frigg). The Njord control system possesses a navigation system and local thruster controllers reported in [39]. Frigg GUI is a platform for interaction with operators. Generally, the user controls the ROV through a graphical user interface(GUI). It provides four control modes: DP, joystick control, waypoint tracking, and automatic control. The lower-level control system then receives the message via TCP/IP from GUI to execute corresponding commands. Verdandi simulator in Section 6.1.4 creates a virtual environment for simulation and is replaced by the ROV Minerva with a real sensing system for fieldwork. Ida Rist-Christensen[36] implemented a hybrid agent architecture in the Frigg GUI as a base of autonomous intervention. This thesis proposes a novel mission planning, control, and execution framework in Section 6.2, contributing to the automatic module in the Frigg GUI.

6.1.1 Njord Control System

Lower-level control is laid in this module, including a guidance system, controller, observer, thruster allocation, and signal processing system. This thesis does not contain relevant work under this module. Nevertheless, to achieve higher-level autonomous robotics missions, the control system components are necessary and significant. The design of the controller and

thruster allocation directly influences the performance of ROV task execution. Also, the observer is necessary for state estimation, especially under the situation of dead reckoning.

Reported in [39], available outputs to the control system are eight measurements, including depth, altitude, heading, yaw rate, north position, east position, surge velocity, and sway velocity. The output from the Njord control system is the desired rpm for each thruster. Since the ROV Minerva is stable in roll and pitch by itself, the working space of the control system is reduced to four degrees of freedom, including surge, sway, heave, and yaw.

6.1.2 Frigg Graphical User Interface(GUI)

In the graphical user interface, an intuitive presentation of sensing state, system state, data, and map of ROV motion is given. Besides, a selection of four control modes is deployed in this module. Four control modes are DP, manual control, waypoints tracking, and autonomy. Only one control mode could be activated at a time. The work done in this thesis is laid in the Autonomy framework module in the GUI.

6.1.3 Navigation Sensor System

The ROV Minerva is equipped with several sensors, including Doppler Velocity Log(DVL), Inertial Measurement Unit(IMU), and USBL to keep motion estimation and measurement for ROV global and local navigation. The DVL estimates ROV motion by calculating its velocity. IMU performs motion estimation based on the acceleration in pitch and roll and measurement in heading ψ using accelerometers and gyroscopes. The small deviation in acceleration measurements could be accumulated to a large error in motion estimation.

Thus, IMU and DVL cannot be implemented independently as a measurement of the ROV position. GPS is also not accessible to the underwater environment. The HiPAP system delivered by Kongsberg implements the precise ROV position measurement. Both Long BaseLine(LBL) and Ultra-Short BaseLine(USBL) could be applied in the HiPAP system. Due to the ROV working property of spatial independence, the USBL positioning system that transmits acoustic pulse between the operating vessel and ROV is more practical than the LBL positioning system, which relies much on the installation and calibration of baselines on the seabed. Compared to LBL, the disadvantage of USBL is inaccurate measurement as the increasing distance between ROV Minerva and R/V Gunnerus.

6.1.4 Verdandi Simulator

Before the field experiment, simulation of ROV motions should be deployed to test the system's performance and ensure further fieldwork safety. Simulation costs less compared to fieldwork and thus is regarded as a desirable method to check system errors and improve performance. The ROV simulator, Verdandi, has been developed to run together with the control system for simulation purposes.

This HIL simulator is independent of the ROV control system, and thus, any changes made in the control system will not affect the function of the simulator. Once the simulator is developed, no further modification is required for most circumstances to enable convenient troubleshooting and debugging in the control system. The HIL method incorporates hardware components in the simulation that increase credibility compared to pure numerical simulation. Simulation results are presented in Chapter 7.

6.1.5 R/V Gunnerus

R/V Gunnerus is a research vessel owned by NTNU, used for research in many regions, including geology, archaeology, and marine technology. The vessel uses GPS for global positioning and is equipped with a HiPAP system, an underwater acoustic positioning, and navigation system, to measure relative distance and direction between ROV and the vessel. Based on inputs from the HiPAP system, RV Gunnerus GPS, and heave-roll-pitch sensor (MRU), NaviPac is an integrated navigation software that calculates the north and east position of ROV Minerva in UTM coordinates reported in [39].

6.1.6 Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

TCP communication is widely used among the Verdandi simulator, the Njord control system, and the Frigg graphical user interface because of its high reliability. TCP communication guarantees no loss of message during the transmission and is thus the desired protocol for message transmission. The drawback of TCP/IP is its transmission time delay, which is longer than UDP communication. In this thesis, TCP communication is also used in message transmission between SLAM-based motion estimation and obstacle detection, and the control system.

UDP is a simple, quick, but less reliable message transmission protocol than TCP. UDP communication does not guarantee the transmission of a message. When doing HIL simulation, the communication between the Verdandi simulator and the Njord control system is deployed by UDP.

6.2 Autonomy Framework in Labview

This section describes how the Autonomy Framework is integrated into the Graphical User Interface (Section 6.2.1) and how each component in the Autonomy Framework is connected with others (Section 6.2.2 and 6.2.3).

6.2.1 Integrating the Autonomy Framework into the GUI

This thesis's work develops an autonomous mission planning, control, and execution framework that guides the ROV control system to perform some planned tasks by following the generated

actions. The autonomy framework is laid in the autonomy module in the Frigg graphical user interface, as shown in Figure 6.1. TCP is used for inter-communication.

6.2.2 Integrating Planning into the Autonomy Framework

In this thesis, the mission planning is developed in the Python programming environment. A text file is created to integrate mission planning code in Labview, including information of the current state(initial state) and the goal state. This text file is then used by the mission planning code to generate a reasonable plan. A *Python Node* is also added that can call the Python function directly, and thus the planning result is introduced in Labview. The *Python Node* shown in Figure 6.3 supports a large number of data types [33].

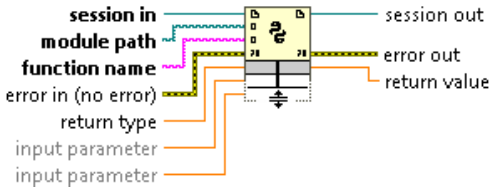

Figure 6.3: Python node in Labview

To configure the Python node, the *session in* that denotes the python version being used, the *module path* that refers to the path of mission planning code and the *function name* from which the developer wants to return value should be specified. Besides, the expected type of return value is specified in *return type*.

6.2.3 SLAM-based Motion Estimation and Obstacle Detection

The computer vision is applied to ROV navigation by two Master students Shuyuan Shen and Erlend Røilid Vollan. To increase the autonomy level of ROV, high-accuracy local navigation, and obstacle detection are required. Visual simultaneous localization and mapping (SLAM) is a computer vision technology that has been widely used in land vehicle navigation. Shuyuan Shen developed a motion estimation system and is used as the measurement of positions for localization. Erlend Røilid Vollan proposed an obstacle detection system in C++. His SLAM-based obstacle detection is applied during a virtual experiment using TCP/IP communication. Shown in Figure 6.1, the SLAM-based obstacle detection is integrated into the autonomy framework proposed in this thesis.

Chapter 7

Simulations

Simulations of the mission planning and control have been performed on the vehicle, testing the mission planning performance and simulating the plan-based mission control and execution framework. The simulator *Verdandi* is used for HIL simulation. As part of inputs, camera and sonar signals are virtually simulated for activating some behaviors such as obstacle avoidance. TCP communication between SLAM-based obstacle detection and mission control architecture has been carried out to prove the ROV's capacity of collision avoidance in Section 7.4.

Section 7.1: Classical Planning; Global navigation and manipulation.

Section 7.2: Reactive control; Collision avoidance and exceedance of cable tension during global navigation.

Section 7.3: Deliberative planning and control; Autonomous mission including sonar and camera tracking, global navigation and local operation.

Section 7.4: Hybrid planning and control; Autonomous mission including both deliberative and reactive control.

7.1 Planning for ROV Missions

In this section, four search methods for mission planning are tested. Section 7.1.1 presents simulation results of global navigation, which has only four candidate actions. Section 7.1.2 compares performances of four search methods applied in the planning of the local operation. An example of local operation using manipulators is used to test the performance of these searching methods. The planning of local operation is more complicated than that of global navigation, which has four possible actions.

7.1.1 Planning: Global Navigation

As discussed in Section 5.2.1, four candidate actions in the mission planning for global navigation are *Launch*, *Descent*, *Transit* and *Operation*. The initial state for planning is the current simplified

state. Human operators set the goal states based on mission requests. For example, the ROV should perform *Launch-Descent-Transit* to the desired position of starting local operation. After finishing operation, the ROV should execute *Transit-Descent* to move to the RV Gunnerus. Therefore, the desired planning result for global navigation is *Launch-Descent-Transit-Operation-Transit-Descent*. The planning results are dependent on the initial states, goal states and possible actions. Therefore, if the ROV starts above the seafloor, it does not need *Descent* to dive down to the seafloor. The planner will give *Launch-Transit-Operation-Transit-Descent* as the planning result.

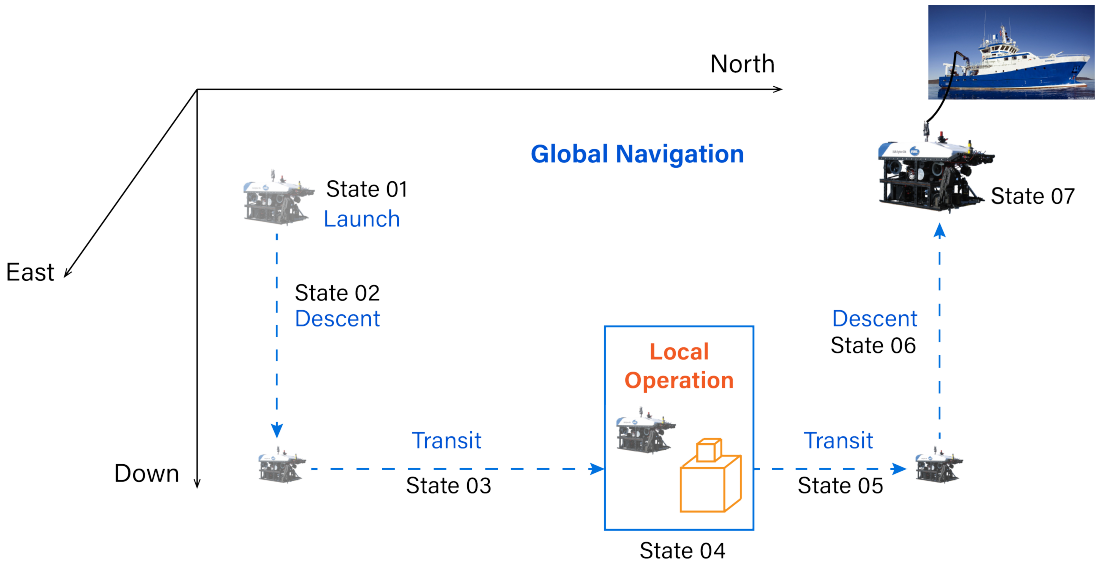

Figure 7.1: Relationship between global navigation and local operation

The simulation is carried out based on the following settings for initial states and goals:

(1) The ROV is located on the sea surface. The position in the horizontal NE plane is not the same as that of the local operation. The ROV is initialized with launch and operation 'not finished'. The initial state corresponds to 'State 01' in Figure 7.1.
(2) The operation place (Target) is located on the seafloor with a known depth different from the initial state and the goal state.
(3) The end position ('State 07' in Figure 7.1) of ROV after finishing the mission is also on the sea surface. It does not equal to the NE position of the local operation.

Therefore, in the text file defining initial state and goal state, the initial states are:

(H1, ROV), (V1, ROV), (H2, Target), (V2, Target), (NotStart, ROV), (NotFinish, Target)

The goal states are:

(H3, ROV), (V3, ROV), (Start, ROV), (Finish, Target)

The simulation result of planning for global navigation is shown in Table 7.1. A comparison of testing results using four searching methods is presented in Table 7.2. All search methods find the optimal plan with length six. Best-first search and heuristic search use the shortest time to

find the solution. The fast-forward searching method significantly shortens the iterations during planning compared to heuristic searching. However, the fast-forward search time cost is much longer than the best-first and heuristic search due to the construction and extraction of Planning Graph. Helpful actions significantly shorten the running time of fast-forward searching.

Number	**Actions**
1	Launch
2	Descent (V1, V2)
3	Transit (H1, H2)
4	Local Operation
5	Transit (H2, H3)
6	Descent (V2, V3)

Table 7.1: Planning results of global navigation

Method	**Run Time(s)**	**Iterations**	**Plan Length**
BFS	0.000997304916381836	18	6
HPS	0.0009975433349609375	12	6
FFS	0.006953716278076172	7	6
HFFS	0.002991914749145508	7	6

Table 7.2: Comparison of four searching methods applied on ROV global navigation planning. BFS: Best-first Search; HPS: Heuristic Planning Search; FFS: Fast-forward Search; HFFS: Fast-forward Search with helpful actions.

Although fast-forward search results in longer running time, it is hard to conclude that heuristic search is better than the fast-forward search for other cases. The fast-forward search significantly reduces iterations used to find the solution. Thus for other complex tasks and missions, it is reasonable to predict that fast-forward searching might be more efficient than heuristic search. In this task of global navigation, the heuristic searching method is implemented in the mission planning and control architecture simulated in Section 7.2, 7.3 and 7.4 because of its satisfactory performance and the shortest time cost.

7.1.2 Planning: Local Operation

A more complex planning task is tested in this section. The task goal is to replace an old device with a new device on the seafloor. In order to change the device, the subsea system has to be terminated for safety consideration. The manipulator can use a tool to stop the system by switching the valve from 'on' to 'off'. It is assumed that the manipulator holds the new device

in the initial state. Finally, the manipulator should hold the old device after installing the new device and switching the valve condition to 'on'. The searching result is shown in Table 7.3.

Possible actions are *Approach*, *Pick up*, *Put Down*, *Install* and *Switch*. By performing *Approach*, the vehicle can move from to a specific place. *Pick up* enables the vehicle to pick up something using its manipulator. Only one item could be picked up at a time, and only one item is able to be held by the manipulator at a time. By doing *Put Down*, the vehicle will put an item down, releasing its manipulator. *Install* is an action that performs the removal of the old device and installation of the new device. *Switch* is required when the operating status of the valve is to be switched from 'on' to 'off', and vice verse. The vehicle has to use a tool to switch the operating state of the valve.

Number	**Actions**
1	Approach(tool)
2	PutDown(new device)
3	Pickup(tool)
4	Approach(valve)
5	Switch(valve)
6	Approach(new device)
7	PutDown(tool)
8	Pickup(new device)
9	Approach(old device)
10	Install(old device, new device)
11	Approach(Tool)
12	PutDown(old device)
13	Pickup(tool)
14	Approach(valve)
15	Switch(valve)
16	Approach(old device)
17	PutDown(tool)
18	Pickup(old device)

Table 7.3: Planning results of local operation

The testing results for local operation planning is shown in Table 7.4. All search methods find the optimal plan with length 18. The heuristic search uses the shortest time to find the solution after 80 iterations. The Fast-forward searching method significantly shortens the iterations to 53 compared to heuristic searching. However, the time cost of fast-forward search is much longer

than the best-first and the heuristic search. Helpful actions significantly shorten the running time for fast-forward searching.

Although fast-forward search results in longer running time in this case, we cannot conclude that the other two searches are better than the fast-forward search. The fast-forward search significantly reduces iterations used to find the solution. Thus, for other complex tasks and missions, it is reasonable that fast-forward searching might be more efficient than heuristic search. In this task of manipulation, FF shows satisfactory performance and the potential extension of task complexity. Therefore, FF search method is implemented in the mission planning and control architecture for local operation simulated in Section 7.3 and 7.4.

Method	Run Time(s)	Iterations	Plan Length
BFS	0.0050318241119384766	81	18
HPS	0.004985809326171875	80	18
FFS	0.1346120834350586	53	18
HFFS	0.07579731941223145	43	18

Table 7.4: Comparison of four searching methods applied on ROV local operation planning. BFS: Best-first Search; HPS: Heuristic Planning Search; FFS: Fast-forward Search; HFFS: Fast-forward Search with helpful actions.

7.2 Reactive Control

On most occasions, the ROV operates in an unknown dynamic environment. Thus, the valid design of collision avoidance is essential to ensure safety. Collision avoidance is simulated in several trials, proving satisfactory results. The testing result is discussed in 7.2.1. Another necessary reactive behavior of ROV underwater operation is to deal with exceeding of cable tension. Under some special conditions, the cable might be wrapped by some underwater structures, especially when the vehicle is operating close to the seabed. The testing results of cable tension is presented in Section 7.2.2. Section 7.3.1 presents simulation results of sonar tracking and camera tracking.

7.2.1 Obstacle Avoidance

In this simulation, four 'unknown' obstacles are manually set to test the obstacle avoidance performance. Table 7.5 shows the locations of four obstacles. An obstacle will be detected only when the distance between the vehicle and the obstacle is less than five meters. The detecting distance is determined by the general capability of vision-based obstacle detection. Consequently, the mission planning and control system will not know where these obstacles are before encountering them. During the simulation, only the first three obstacles are detected. The *Obstacle 04* is more than five meters away from the whole trajectory of the vehicle and thus is not detected.

Obstacle	North Position [m]	East Position [m]	Depth [m]	Diameter [m]
01	7036914	570120	15	2
02	7036912	570114	15	1
03	7036922	570112	15	1
04	7036926	570125	15	2

Table 7.5: Positions of four 'unknown' obstacles in the simulation of reactive control

Figure 7.2 presents position estimates and desired positions of the vehicle in the North-East (NE) plane, giving satisfactory simulation results of reactive obstacle avoidance. This mission simulation's starting position is (7036892, 570128, 10), and the desired position is (7036932, 570112, 0). The vehicle is performing *Transit* action at η_{OA}^0 when the first obstacle (*Obstacle 1*) is detected. Then, a temporary waypoint for OA is generated, and the vehicle moves toward that waypoint. Before reaching the waypoint for avoiding *Obstcale 01*, the vehicle detects a new *Obstcale 02* at η_{OA}^1 and generates a new waypoint to avoid it. The vehicle reaches η_{OA}^2 and finishes obstacle avoidance for *Obstacle 2*. Then, the vehicle moves towards η_{end} and meets the *Obstacle 3* after a few seconds of transit. Finally, the vehicle reaches η_{OA}^3 and ends at η_{end}.

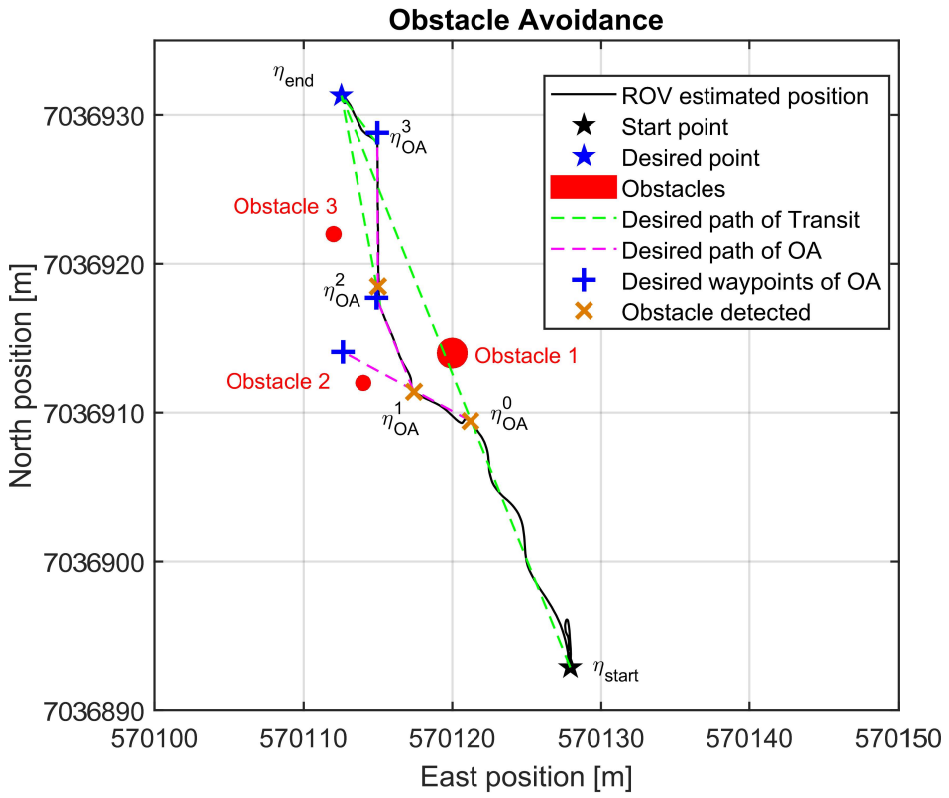

Figure 7.2: ROV position in NE plane of Obstacle Avoidance

The safety radius for OA is set as $R = \frac{d}{2} + e$, where d denotes the diameter of the detected obstacle and $e = 2$ is a safety parameter. Figure 5.6 and Equations 5.1 show how the new waypoint is calculated. The testing result shows that the obstacle avoidance algorithm successfully enables the vehicle to avoid collisions during mission execution. The mission planning and control system has the ability to encounter more than one obstacle at a time and update its waypoints based on the position of the closest obstacle.

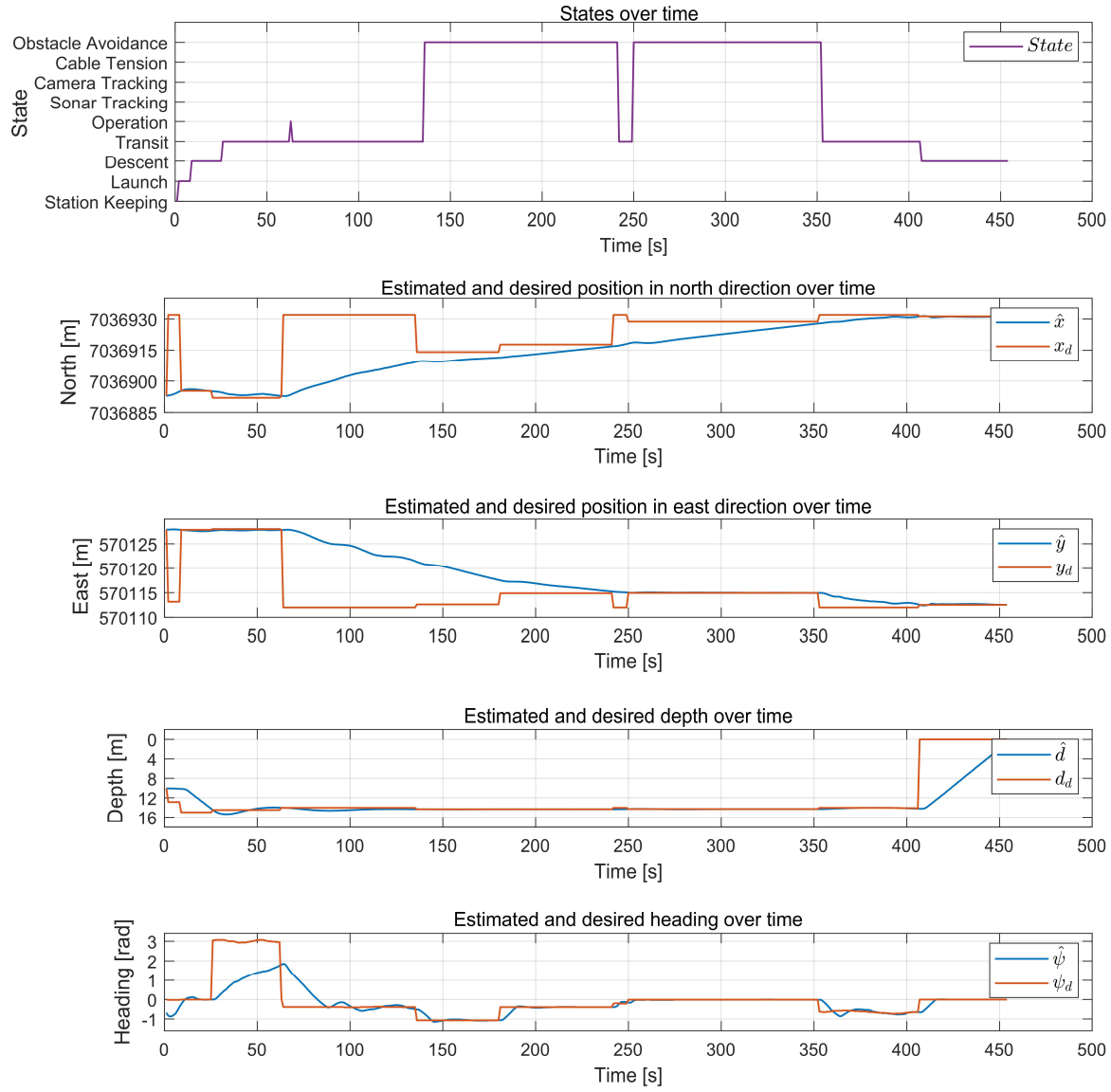

Figure 7.3: ROV execution state, position and heading of Obstacle Avoidance

Estimated and desired north, east positions, depths, and heading angles are shown in Figure 7.3. The execution states are also presented in the figure. The vehicle firstly performs *Launch-Descent* to the desired depth. Then, the vehicle moves to the operating location. Since no operation is requested in this simulation, the mission planning and control system converts the state from 'operation unfinished' to 'operation finished' as soon as it reaches the operating position. This is

shown in Figure 7.3, where there is a sharp corner at around 65s in the first sub-figure. Then, the vehicle does *Transit-Descent*, returning back to the end position. Collision avoidance is tested during *Transit* to the end position. The first two obstacles are detected, and corresponding obstacle avoidance behaviors are performed when the state becomes '8' from 135s to 240s. The third obstacle is detected and avoided when the state becomes '8' again. The desired positions are directly derived from the mission planning and control system, instead of the guidance system or the control system. Thus, sudden changes in desired positions are presented in the figure.

This simulation aims to test the performance of reactive obstacle avoidance for the mission planning and control system. Therefore, *Local Operation* behavior is significantly shortened, and the obstacle avoidance is tested during *Transit* action in this section. The mission planning and control system is also able to tackle obstacle avoidance when preforming other actions. Simulation results are presented in Section 7.4, where both reactive and plan-based obstacle avoidance are simulated during one testing.

7.2.2 Cable Tension

Many situations might induce exceeding of cable tension. The most normal conditions are that the cable gets wrapped by some structures. Manual operation is the optimal solution to deal with this situation, analyzing and reasoning the situation by human operators. Before switching to manual operation, the vehicle firstly performs backtracking its trajectory to release the cable tension.

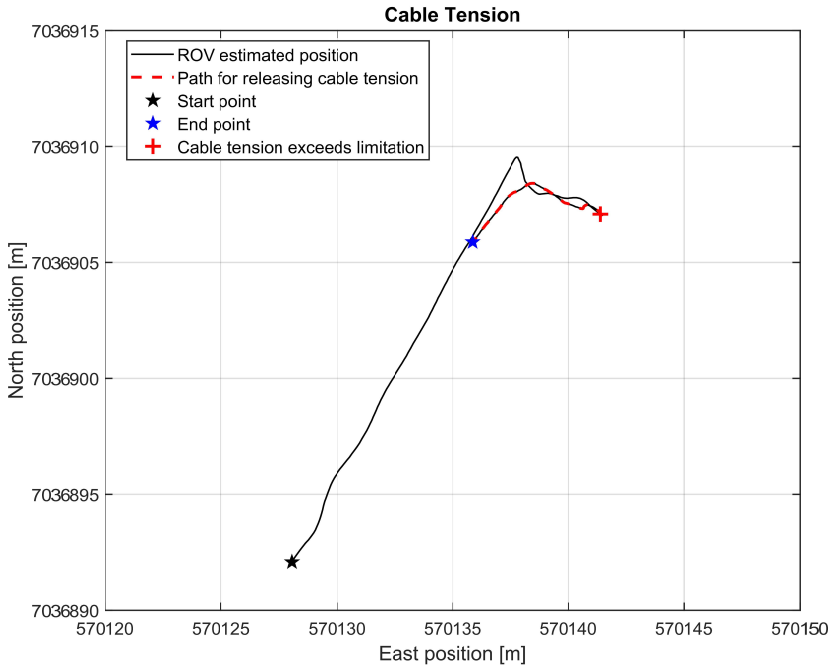

Figure 7.4: ROV position in NE plane of Cable Tension

Figure 7.4 presents the vehicle trajectory of releasing cable tension. The exceedance of cable tension is simulated by manually clicking the 'Cable Tension' button during testing. Then, the

system updates the status of cable tension in the mission control system, and the vehicle starts to trace back its original paths.

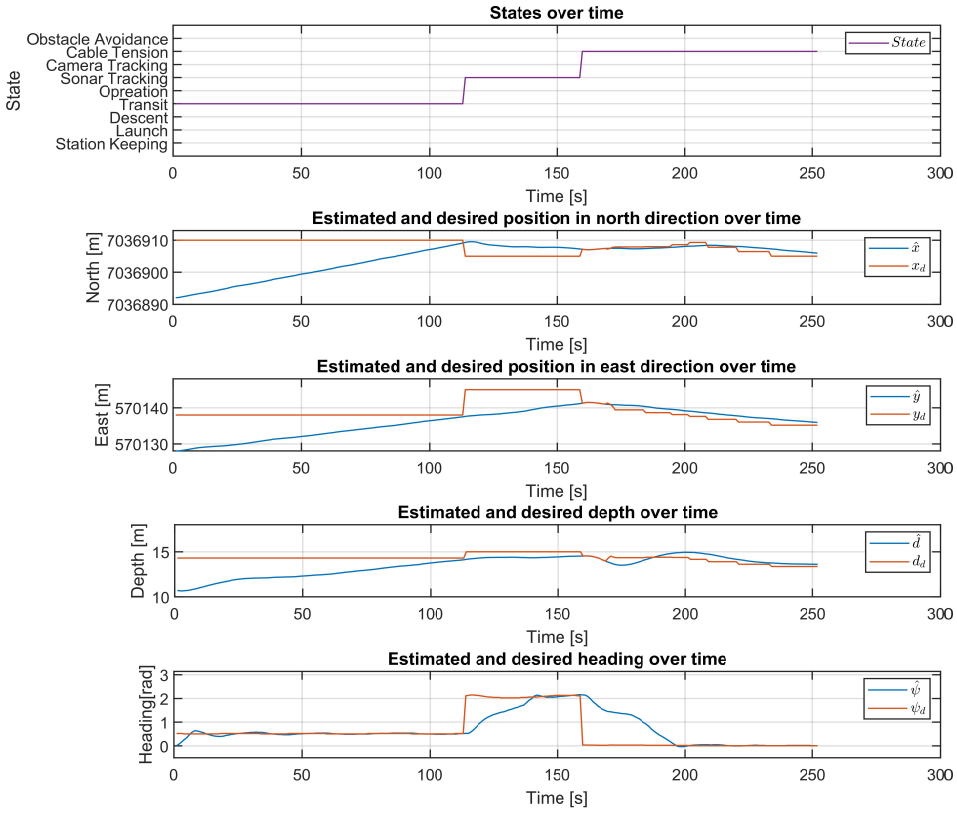

Figure 7.5: ROV execution state, position and heading of Cable Tension

Figure 7.5 presents the currently executed actions, north positions, east positions, depths and heading angles with time. It shows that the exceedance of cable tension occurs at around 160s when the vehicle is executing sonar tracking. The system switches to *Cable Tension* action immediately and goes back to the position that it was at 100s ago. Finally, the autonomous mission execution is replaced by manual control to figure out the reasons for cable tension exceedance.

7.3 Deliberative Planning and Control

This section presents the deliberate behaviors of the proposed mission planning and control system. Simulations are carried out vial HIL simulation. Section 7.3.1 shows the simulation results of sonar and camera tracking and Section 7.3.2 presents the entire autonomous mission of global navigation and local operation without reactive control, from *Launch* to returning back.

7.3.1 Tracking of Targets

In this section, sonar tracking and camera tracking are tested during *Transit*. The sonar signal and camera signal are virtually simulated. Firstly, the vehicle performs *Transit* to an estimated

location of the target. By manually clicking the 'Sonar detection' button, the system knows the sonar-detected target position. The predefined location is updated and sent to the planner. The replanning of the mission enables the vehicle to generate a new plan and track the new target position. It is assumed that the sonar detects a target before the camera finds it because the detecting range of sonar is usually more extensive than that of the camera. When the camera also successfully detects the target (manually set), replanning, and tracking of target location detected by the camera are performed. This means that the position data from camera detection is regarded as the most accurate data when both sonar and camera detect the target. Table 7.6 shows the target positions from estimation, sonar tracking, and camera tracking in the NE plane.

Location	North Position [m]	East Position [m]	Depth [m]
Target location	7036910	570138	15
Sonar detected location	7036905	570145	15
Camera detected location	7036900	570152	15

Table 7.6: Target positions from estimation, sonar tracking and camera tracking

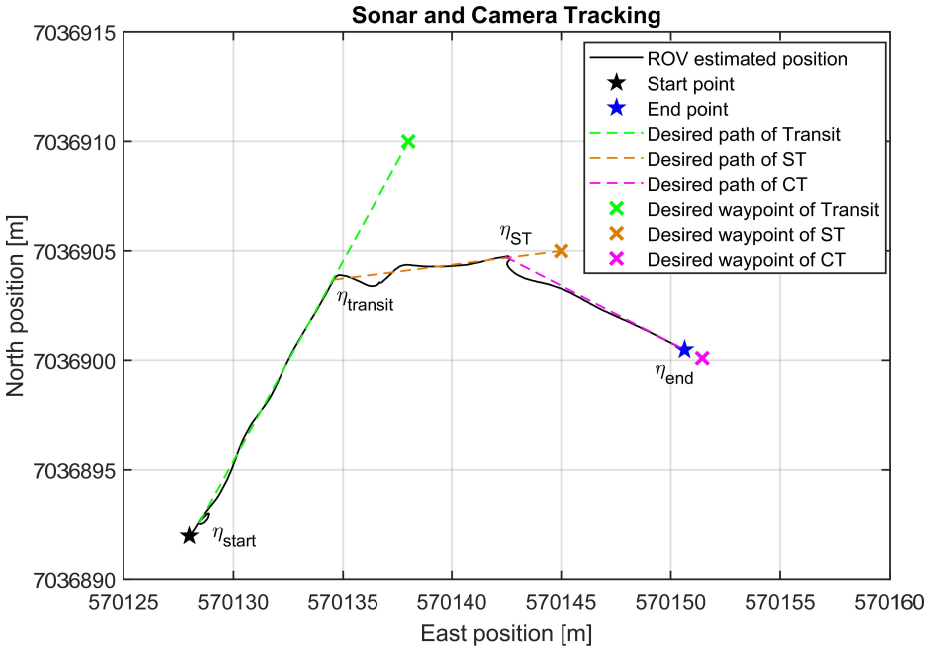

Figure 7.6: ROV position in NE plane of Sonar and Camera Tracking

This simulation presents the partial autonomous mission from launch to camera tracking. Figure 7.6 shows the vehicle trajectory of performing sonar and camera tracking. The mission starts at 10 m depth and descends to the desired depth. Then, the vehicle transits towards the estimated location of the target. At $\eta_{transit}$, the sonar detects the target, and the detected location is updated as desired. The vehicle then moves to the desired waypoint of sonar tracking, and the camera detects the target at η_{ST}. During this simulation, it is assumed that the target position derived

from the camera signal is more reliable than the sonar signal. Thus the vehicle ends at η_{end}, which is close to the target position detected by the camera.

Figure 7.7 presents the currently executed actions, north positions, east positions, depths, and heading angles with time. From the plotting, the estimated ROV position keeps tracking the desired position, which proves the ROV capability of following the command from the mission planning and control system. During *Launch* and *Descent*, the vehicle position in the NE plane is not strictly restricted. Thus, the NE desired position shown in the plot has some oscillation. Another problem shown in the plotting is that the heading angle fluctuates a lot and then reaches the desired number, which is overshoot.

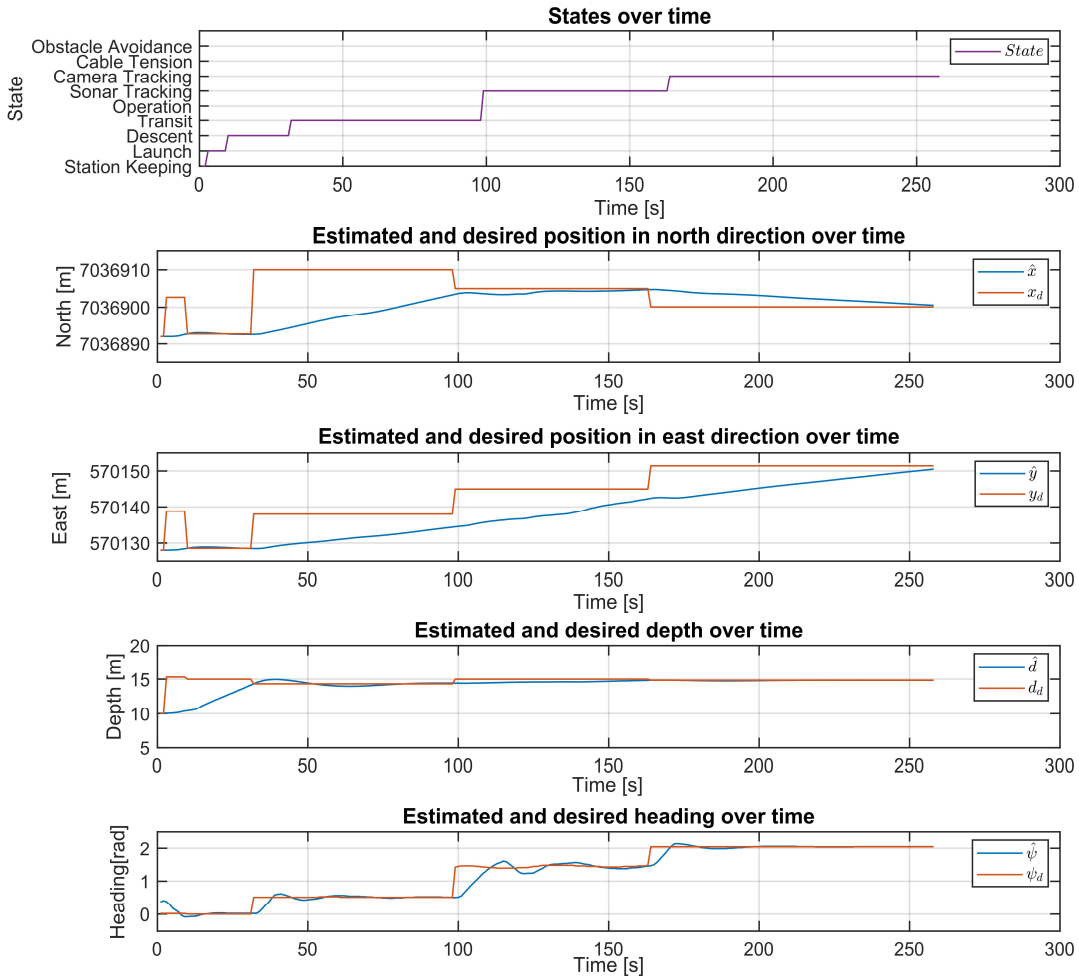

Figure 7.7: ROV execution state, position and heading of Sonar and Camera Tracking

7.3.2 Global Navigation and Local Operation

In this section, deliberative behaviors are tested, including global navigation and local operation. This section describes the mission execution under deliberative control and how local operations are implemented. *Mapping* and *Charging* are designed and tested as actions belonging to local operations. The deliberative behaviors (actions) of the mission planning and control architecture consists of *Launch*, *Descent*, *Transit*, *Operation* (including *Mapping* and *Charging*). The

deliberative actions *Transit* and *Descent* are also tested in the previous trail in Section 7.2. The mission planning and control system has the capability of performing tasks and missions starting at any positions and states. This section presents the entire autonomous mission except reactive behaviors. One action might be activated more than once if the mission planning results require. For example, as shown in Figure 7.1, *Descent* and *Transit* are planned to be executed twice before and after local operation. Table 7.7 shows six waypoints of mapping.

Mapping Waypoints	**North Position[m]**	**East Position[m]**	**Depth[m]**
01	7036912	570143	15
02	7036887	570143	15
03	7036887	570138	15
04	7036912	570138	15
05	7036912	570133	15
06	7036887	570133	15

Table 7.7: Desired waypoints of Mapping

Figure 7.8 and 7.9 shows vehicle trajectories in 2D and 3D. The mission requested is to perform mapping during local operation. *Charging* is tested during mapping under local operation in the deliberative layer. When the battery status is manually set as *LOW BATTERY*, replanning of the local operation is performed immediately, and the vehicle executes *Charging* to charge at the charge station. The testing is carried out with HIL simulation. Figure 7.8 shows that there are many fluctuations when the vehicle changes its heading. This thesis focuses on the development of the mission planning and control system and is not responsible for improving the performance of waypoint tracking. Optimization of the guidance and control system could be further considered as other research.

From Figure 7.8, the vehicle starts at η_{start} and then moves towards waypoints generated from the mission planner sequentially. The vehicle firstly performs *Launch* and *Descent* to the desired depth of operating. Then, the vehicle transits to the starting position of mapping and performs mapping, tracking six waypoints (η_m^0, η_m^1, η_m^2, η_m^3, η_m^4, η_m^5) sequentially. During this process, 'LOW BATTERY' is manually set, indicating that the battery will be out of power soon, and the vehicle has to charge for power. The charging station is set at [7036875, 570135, 27]. Therefore, when 'LOW BATTERY' state is activated, the vehicle has to replan to generate a new plan to tackle this situation. In this simulation, the battery state is manually set as 'low' when the vehicle is at η_c. Under the *Charging* action, a path planner for homing and docking is implemented, which generates a set of waypoints to guide the vehicle reaching the dock (charge) station. The path planner is designed by Signe B. Moltu and integrated into the mission planning and control architecture to generate the necessary path for *Charging*.

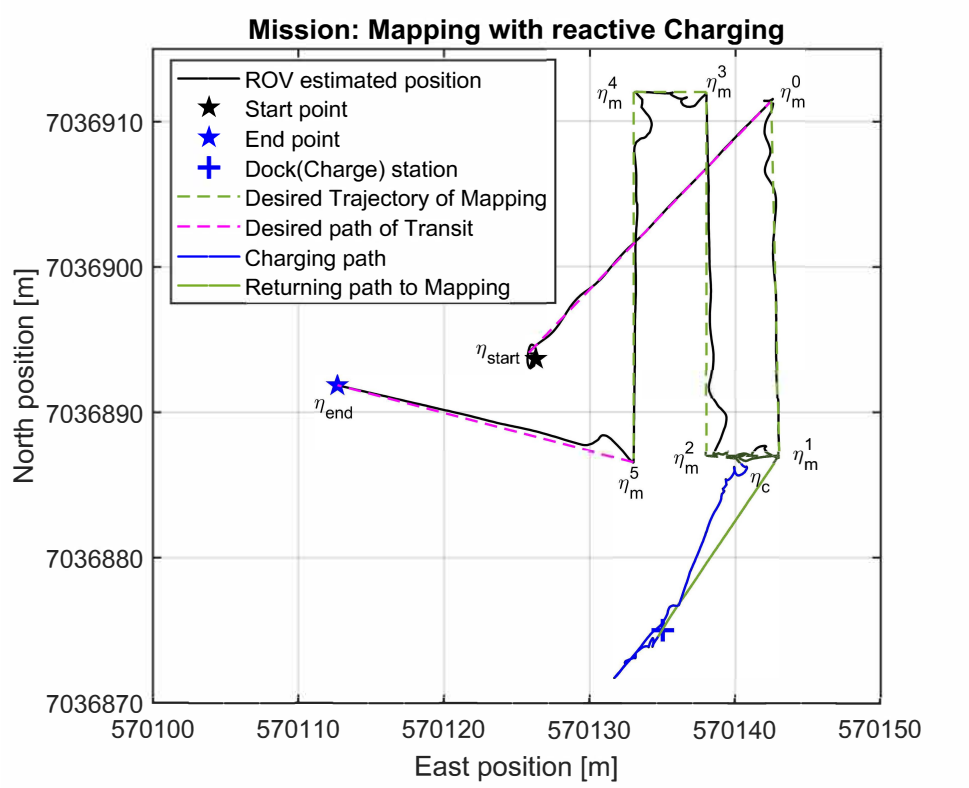

Figure 7.8: ROV position in NE plane of mission: Mapping with Charging

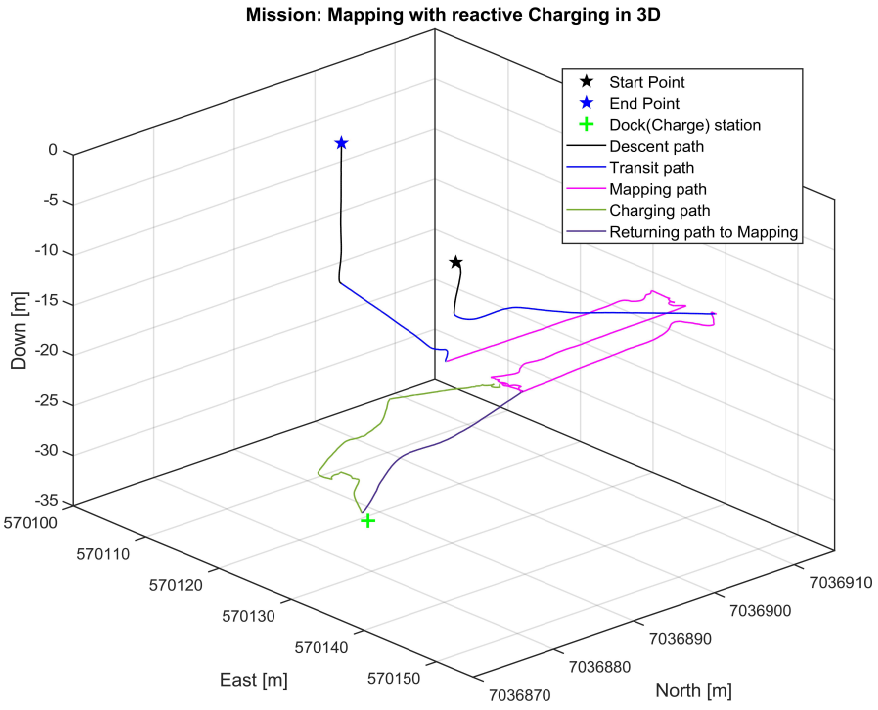

Figure 7.9: ROV position in NED plane of mission: Mapping with Charging

Since Figure 7.8 shows the vehicle's trajectory in North-East (NE) plane, *Descent* trajectory is not plotted here. Figure 7.9 presents trajectories of all actions. The waypoints of mapping are defined before starting the mission. The path planner plans the waypoints of charging for homing and docking once *Charging* action is activated. Obstacles are not simulated here, and all actions/ behaviors are planned under the deliberative layer.

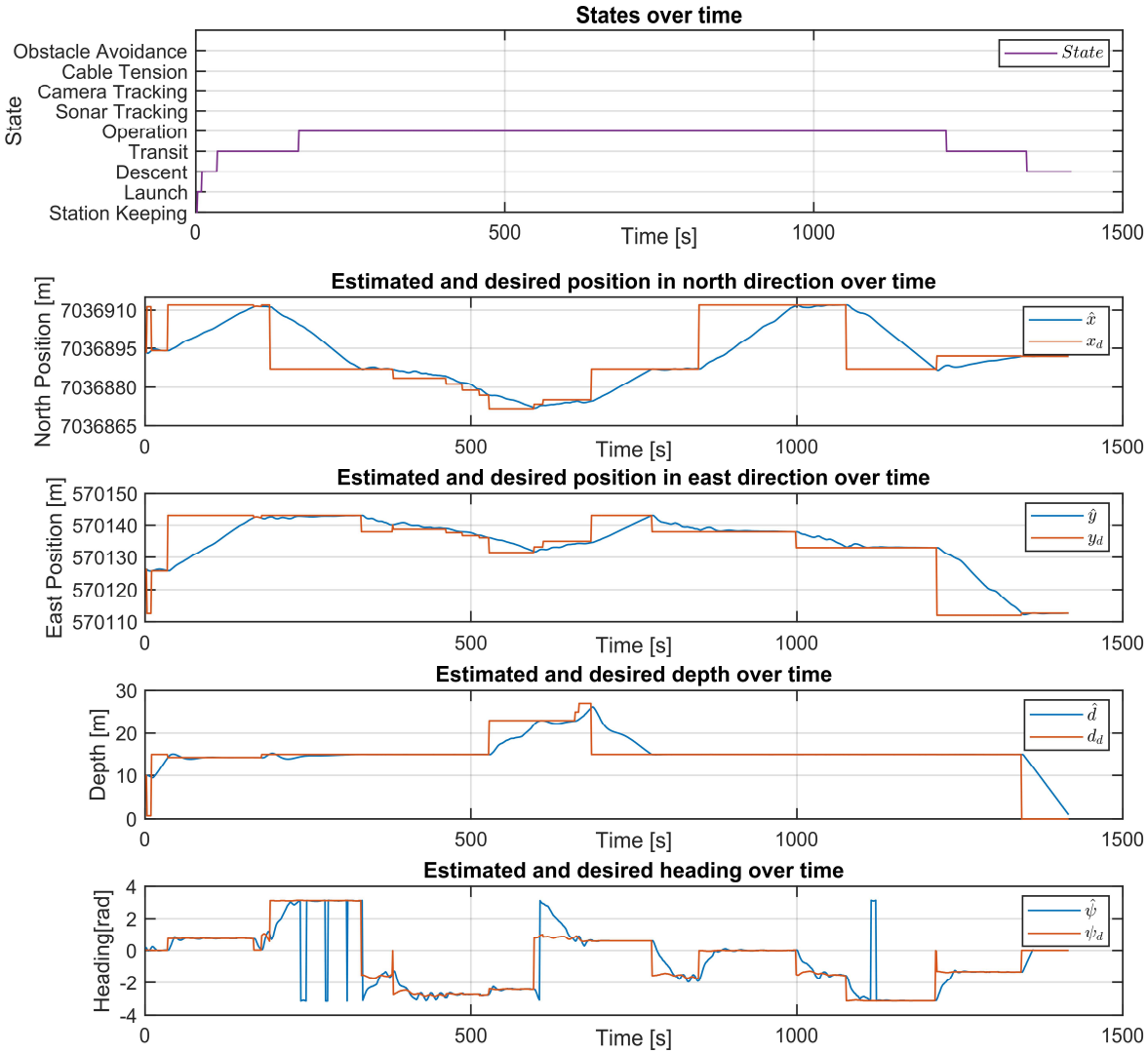

Figure 7.10: ROV execution state, position and heading of mission: Mapping with Charging

Figure 7.10 presents the currently executed actions, north positions, east positions, depths, and heading angles with time. From 200s to 1220s, the vehicle is performing the local operation. From the simulation results, it is evident that the planning algorithm always finds correct plans to guide missions. The vehicle can successfully track the desired waypoints generated by the mission planning and control system. It is observed that many jumps in the heading angle occur during this simulation. These jumps come from the conversion in radians to the interval $[-\pi, \pi]$. Most of these jumps occur when the desired heading angle is π.

Another simulation is implemented to prove the capability and flexibility of the mission planning and control system. Figure 7.11 and 7.12 presents the testing results in 2D and 3D, respectively. The desired trajectory of mapping is the same as the last simulation shown in Figure 7.8 and 7.9. The end position and the moment of starting charging are changed. The starting position of *Charging* is at η_c which is different from the last simulation shown in Figure 7.8.

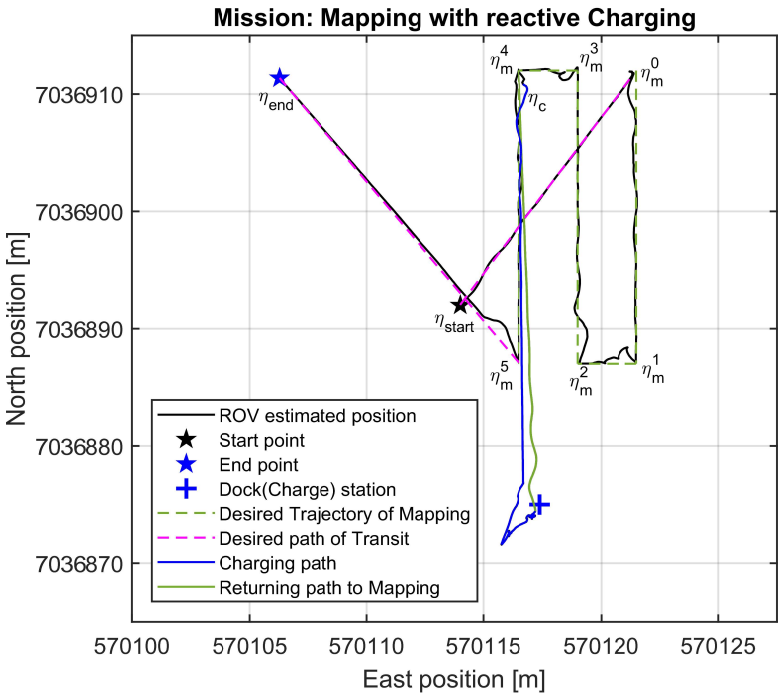

Figure 7.11: ROV position in NE plane of mission: Mapping with Charging (alternative)

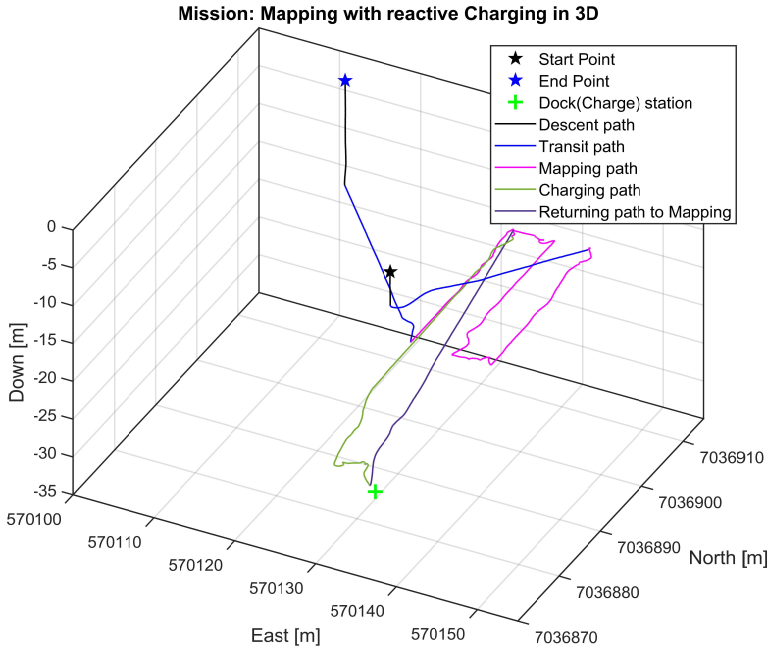

Figure 7.12: ROV position in NED plane of mission: Mapping with Charging (alternative)

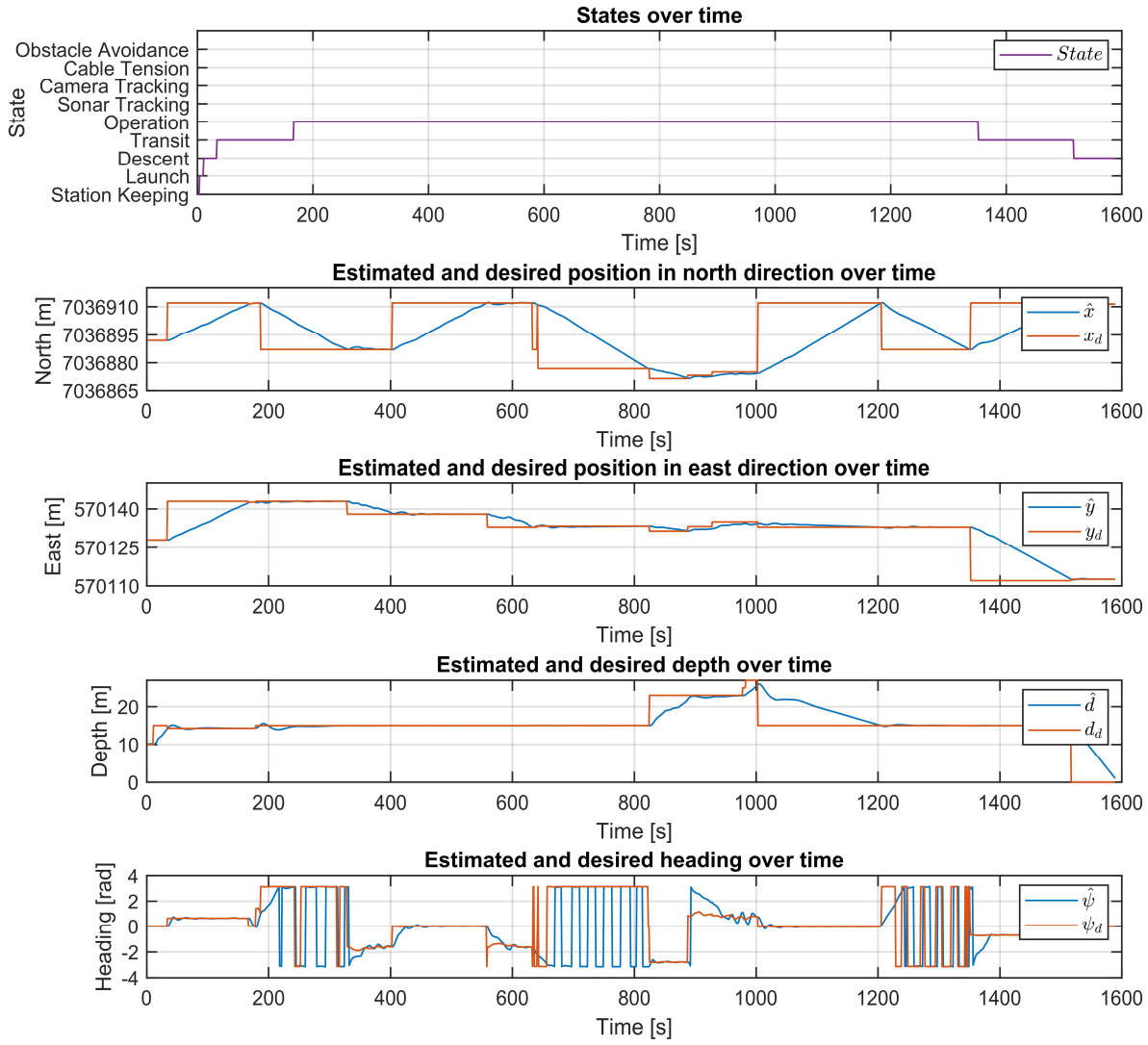

Figure 7.13: ROV execution state, position and heading of mMission: Mapping with Charging (alternative)

7.4 Hybrid Planning and Control

This section presents the final simulation of the ROV mission planning and control system. The simulation implements mapping and sampling with charging and obstacle avoidance. Obstacle avoidance is an important behavior that is tested in different ways. For actions that require path planning, replanning will be activated to generate a new plan when a new obstacle is detected. The new plan will definitely avoid the detected obstacle. For example, obstacle avoidance is tested during *Sampling*, where the path is replanned due to the detection of new obstacles. For other actions without path planning, reactive obstacle avoidance will be performed when encountering obstacles.

A list of actions and their reactive collision avoidance behaviors are presented in Table 7.8. *LO* in the table denotes local operation. From the design of all actions, *Charging* and *Sampling*

are the only two actions that path planning is implemented and thus use replanning to avoid obstacles.

Action	Collision avoidance behavior
Launch	Reactive obstacle avoidance
Descent	Reactive obstacle avoidance
Transit	Reactive obstacle avoidance
LO-Mapping	Reactive obstacle avoidance
LO-Charging	Replanning to generate a new path
LO-Sampling	Replanning to generate a new path

Table 7.8: Collision avoidance behaviors of each action

7.4.1 Mission: Global Navigation and Local Operation with OA

The positions of four obstacles are manually set. A code for obstacle detection is programmed to test the distance between the vehicle's current position and obstacles. When the distance becomes shorter than five meters, the vehicle will perform reactive obstacle avoidance. Under some circumstances, more than one obstacle might be encountered at a time. The system chooses the nearest one as the current obstacle. It executes obstacle avoidance correspondingly, which is tested in this simulation.

Obstacle	North Position[m]	East Position[m]	Depth[m]	Diameter[m]
Obstacle 01	7036900	570144	15	2
Obstacle 02	7036907	570120	15	1
Obstacle 03	7036900	570114	15	1
Obstacle 04	7036893	570105	15	2

Table 7.9: Positions of four 'unknown' obstacles in the simulation of hybrid control

Figure 7.14 and Figure 7.15 shows the vehicle trajectory in the NE plane and NED plane during this simulation. The vehicle is planned through the deliberative layer to approach the desired location, perform the local operation, and return to the desired end position. While doing so, the vehicle detects and avoids obstacles on its path.

For this simulation, the local operation goals are finishing mapping with predefined trajectories and performing sampling at a specific region. The vehicle first executes *Descent* and *Transit* to the starting position of local operation. The first action planned by the sub-planner is *Mapping*. The starting position of mapping is η_m^0, shown in Figure 7.14. The action *Charging* is implemented during this simulation when the energy status is manually set as *LOW-BATTERY* at η_c, shown in

the plotting of ROV trajectory in NE plane. After mapping, the vehicle is planned to perform *Sampling* action by moving to the sampling region. The sampling region is predefined by a sampling center and the radius of the region. The sampling center is defined as [7036920, 570112, 15] in this simulation, and the radius is five meters. After finishing *Sampling*, the vehicle finishes all operations and returns to the end position of the mission by executing *Transit* and *Descent*. Therefore, the vehicle performs *Descent-Transit-Local Operation (Mapping (OA-Charging) -Sampling (OA))-Transit (OA)-Descent* to achieve the mission requests and avoid collisions.

In this simulation, four obstacles are detected during *Mapping*, *Sampling* and *Transit*. The size of obstacles is different, and the safety radius and corresponding desired waypoints for OA are different. The vehicle first encounters the *Obstacle 1* at $\eta_{OA}^{1-start}$ during *Mapping* (from η_m^0 to η_m^1), and moves to η_{OA}^{1-end} temporarily to avoid this obstacle. Then, the vehicle continues its *Mapping* task. The *Obstacle 2* is encountered when the vehicle is executing *Sampling*. As presented in Table 7.8, the system replans and generates a new path to the sampling region by adding the newly detected obstacle to the known obstacle list.

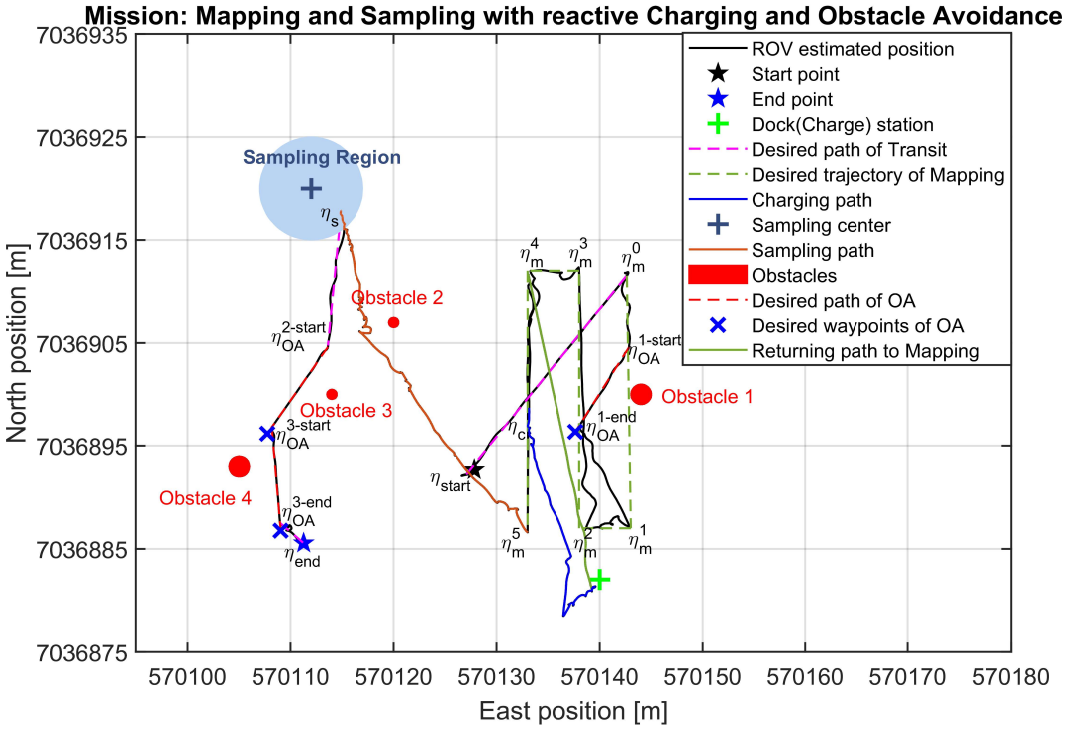

Figure 7.14: ROV position in NE plane of mission: Mapping and Sampling with Charging and OA

Obstacle 3 and *Obstacle 4* are detected during *Transit*. The vehicle detects the *Obstacle 3* at $\eta_{OA}^{2-start}$ and steers to the waypoint for obstacle avoidance. At $\eta_{OA}^{3-start}$, the vehicle has not reached the waypoint of avoiding *Obstacle 3* but encountered a new obstale — *Obstacle 4*. Since the vehicle is closer to *Obstacle 4* compared to *Obstacle 3*, the current OA behavior guides the vehicle to η_{OA}^{3-end} to avoid *Obstacle 4*. After reaching η_{OA}^{3-end}, the system switches to deliberative control, executing *Transit-Descent* to the end position of the mission.

The 3D plotting of the vehicle trajectory in Figure 7.15 shows explicitly all actions performed in this simulation. In the 2D figure, *Descent* action is not shown because the vehicle moves vertically when performing *Descent*. During this simulation, *Charging* and *Sampling* actions use path planning to generate desired waypoints.

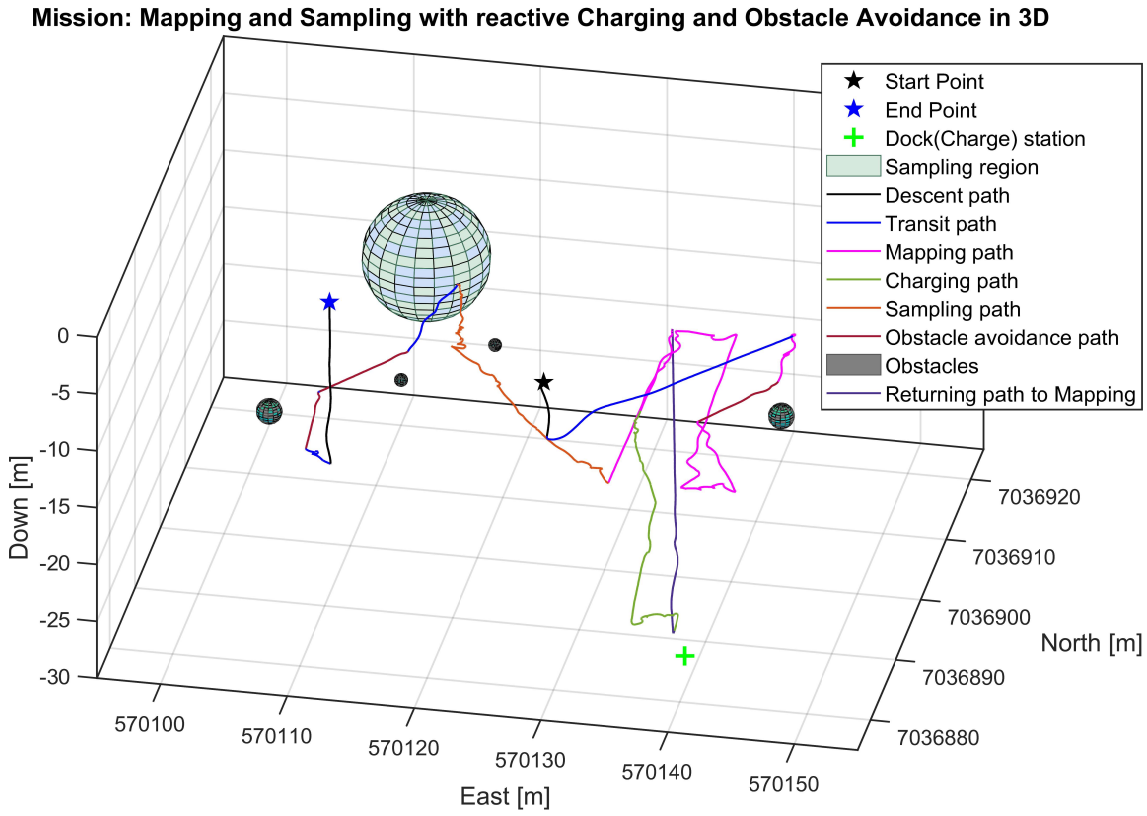

Figure 7.15: ROV position in NED plane of mission: Mapping and Sampling with Charging and OA

Figure 7.16 shows the north position, east position, depth and heading angle of the vehicle with time. It is observed that many jumps in the heading angle occur. As stated in Section 7.3, these jumps come from the conversion in radians to the interval $[-\pi, \pi]$, which occur when the desired heading angle is π.

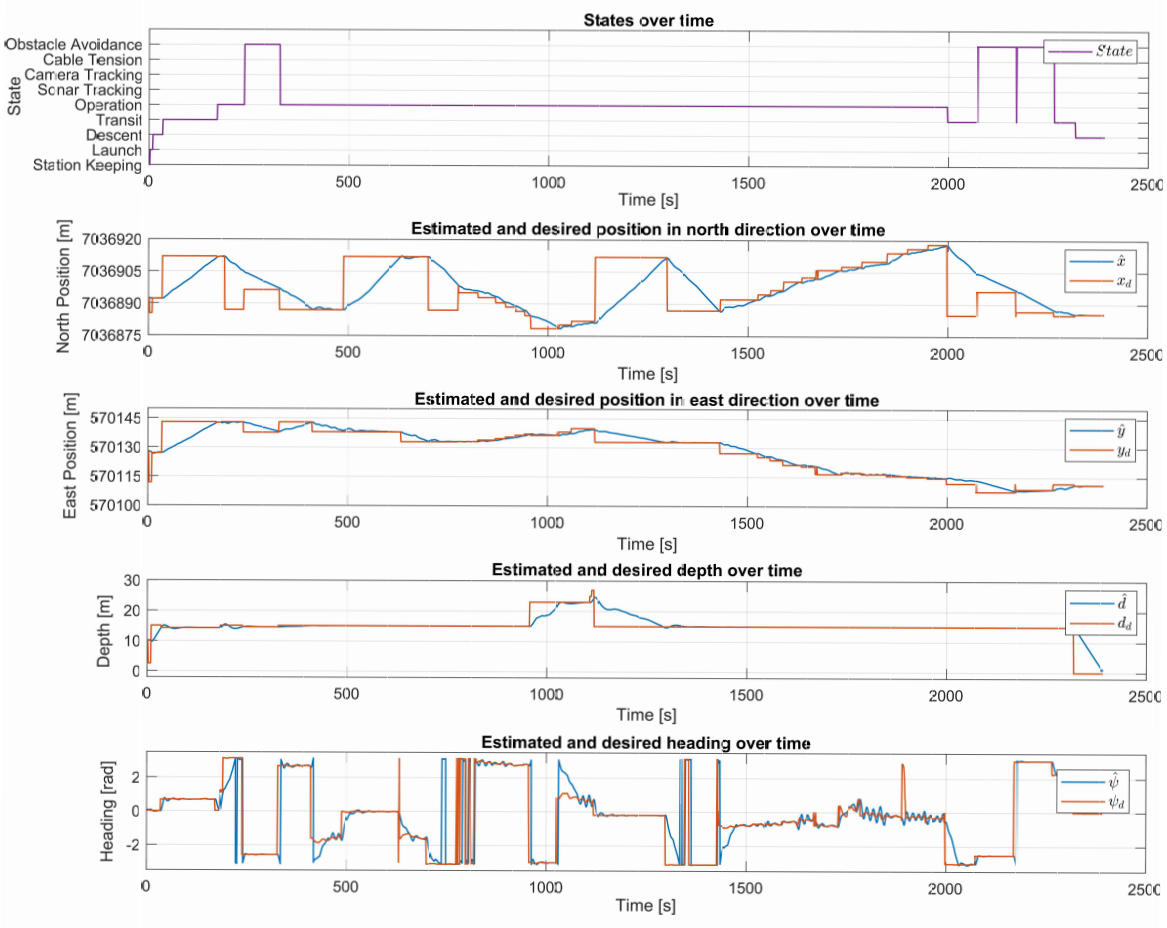

Figure 7.16: ROV execution state, position and heading mission: Mapping and Sampling with Charging and OA

To recap, the HIL simulations have tested all actions of the mission planning and control system. The system's capacity has been proven through this simulation, showing that the vehicle can perform global navigation and local operation under hybrid control. The simulation verifies and enhances the capability of the sub-planner for local operation, showing that the sub-planner can plan and generate desired and correct action sequence to achieve operation goals.

Chapter 8

Discussion

To sum up, the simulation results in Chapter 7 have proved satisfactory behavior of the mission planning and control system. The performance of the mission planner, mission control and execution, and contingency handling are presented. In Section 8.1, the performance of four different planning methods are compared. Section 7.2 shows the performance of contingency handling within the reactive layer. Analysis of the simulation results is presented in Section 8.2. Section 7.3 presents HIL simulation results of the deliberative actions of the mission management system. The results are discussed in Section 8.3. An integration of deliberative and reactive control is tested in Section 7.4, and the result is discussed in Section 8.4.

8.1 Automated Planning

The four automated planning algorithms were tested in Section 7.1. The performance of planning is satisfactory. All of these searching methods can find the optimal plan within a reasonable time.

The testing results show that integrating Graph Plan greatly increased the time cost for simple planning tasks such as global navigation. The plan lengths of four algorithms are all six, which is the shortest and optimal. Therefore, the fast-forward search is not recommended for simple tasks. By comparing the results of the fast-forward search with and without helpful action, it is observed that *Helpful Actions* shorten the running time of fast-forward planning.

For the testing in Section 7.1.2, a relatively complex planning task is performed. The results show that the fast-forward search significantly shortens the iterations used to find a plan, compared to other search methods. However, the time cost of fast-forward search is longer than the best-first and heuristic search due to the construction and extraction of Planning Graph. It is believed that the fast-forward search has great potential for finding a plan for complex tasks.

8.2 Reactive Control

The simulation for reactive actions mainly focuses on two behaviors in Section 7.2: Obstacle avoidance and exceedance of cable tension. The results show that the vehicle can successfully avoid obstacles and perform backtrack to release cable tension as reactive control.

There are two modes for obstacle avoidance. When the vehicle is executing actions that require path planning, the reactive collision avoidance behavior is to replan the path by taking the newly detected obstacle into account. Since the path planning algorithm is laid in deliberative actions, this replanning behavior is designed under the deliberative layer of the mission planning and control system. The path planner is developed by Signe B. Moltu and integrated into the mission planning and control system proposed in this thesis. The path planner's performance is good but is expected to increase the safety margin to ensure safety when encountering an obstacle.

For the other actions, obstacle avoidance is carried out by changing its heading and move to a new waypoint generated based on the law discussed in Section 5.3. This reactive behavior is designed under the reactive layer of the mission planning and control system. The position of the newly generated waypoint for obstacle avoidance depends on the relative position between the vehicle and the obstacle, the heading angle and the size of the obstacle. The obstacle avoidance algorithm successfully steers the vehicle to a safe waypoint temporarily and terminates reactive control after reaching that waypoint.

It is challenging to deal with the situation when the exceedance of cable tension occurred. There might be many reasons including but not limited to being entangled by subsea structures. The proposed solution here is to backtrack its trajectory and switch to control manually until solving the problem. For ROVs without cables, the exceedance of cable tension will not occur. Therefore, when performing hybrid control simulations taking charging into account, cable tension will never exceed the safety threshold because only ROVs without cables need to charge during operation.

8.3 Deliberative Planning and Control

The capability of deliberative planning and control has been proven during simulations in Chapter 7. Results show that under the instruction of mission planner, the mission control system can control the vehicle from a random starting position to a target position where the local operation is to be carried out, and then guide the vehicle to return to the end position of this mission. Under the *Local Operation* action in the deliberative layer, a sub-planner is developed to generate a plan that fulfills the operation goals. As stated before, the deliberative control is organized as a layered or hierarchical system, where *Local Operation* is one of the actions for global navigation and is also an independent subsystem with its sub-planner and sub-control mechanism.

The mission control receives guidance from the mission planner, activates relevant actions, and sends commands to the low-level control system to achieve the mission goals. Consequently, the vehicle can start the mission in any state, operate at a specific position, and end with desired states. Also, the planning and control of operations are laid in the action *Local Operation* of

the global navigation. Modification of local operations is, to a great extent, independent on that of global navigation, assuring flexibility and applicability of the mission planning and control system.

The proposed autonomy framework for mission planning and control has great potential for improvement, especially for local operations. The possible actions for local operations are *Mapping*, *Charging* and *Sampling*. These actions are simply designed as trajectory following, path planning for homing and docking, and general path planning in this thesis. However, more refinements are to be carried out under these actions. For example, manipulation control is necessary to achieve actual sampling. The current work only enables the vehicle to approach the sampling region by path planning. Also, the sub-planner can introduce more actions as local operations, such as observation of a structure. The design of other actions is to be realized in further work.

The *Run-Lookahead* algorithm triggers replanning after finishing each action. A proper plan is generated based on the current states, the goals and possible actions, ensuring the robustness of mission execution. For example, when the vehicle has finished the *Descent* action, the system will replan to generate a new plan, considering the newly updated states. Therefore, the design of the *Run-Lookahead* algorithm for replanning enables the human operator to modify and even delete some goals during the mission execution procedure. This design enables the operator to change the end positions of the mission, the locations and the goals of operations, which makes the mission execution more flexible and operable.

In all simulation results, some oscillations and fluctuations are observed when a new desired waypoint is updated because the desired heading and position change at the same time. All thrusters are working simultaneously to steer the vehicle towards the new waypoint. A sudden change of desired heading direction leads to oscillations before stabilizing at the desired value. To optimize the trajectory following problem, a possible solution could be calling a heading change before moving the vehicle to the desired waypoint. The control law is divided into two parts: Change heading angle if the vehicle is not headed towards the desired waypoint; Move the vehicle to the desired waypoint only if its heading is towards the desired vehicle. Another possible solution to improving the performance of trajectory tracking is to optimize the controller in the heading. These suggestions are not considered in this thesis and are proposed for further work.

Sonar Tracking and *Camera Tracking* are developed under *Descent* and *Transit* actions. When the sonar or camera detects the SOI, the position of SOI detected will be updated to the mission management system. Then the vehicle will move towards the new position of SOI. The testing results show that the mission planning and control system can update the detected position within a short time and guide the vehicle to that position as desired. It is assumed that the camera data of SOI is more reliable than sonar data. Thus the camera data is regarded as the true SOI position if both sonar and camera detect the SOI. However, this assumption proposed in this thesis is not always reasonable nor safe. Sensor fusion could be further considered, developed and optimized for the localization of the SOI.

Besides, although some actions are developed under the deliberative layer of the mission planning and control architecture, these actions are reactive. For example, when the vehicle is performing the local operation and *low-battery* state is manually triggered, the mission planning and control system will then replan to generate a new plan where the first action is *Charging*. Therefore, *Charging* is regarded as a reactive action developed under the deliberative layer. The mission planning and control architecture also has the potential to add other reactive actions/ behaviors further.

Charging is implemented in the deliberative layer of local operations. The location of the charge station is manually set during the simulation. A path planning algorithm for homing and docking developed by Signe B. Moltu is used under the *Charging* action. The simulation result shows that the vehicle can correctly perform path planning and follow the instruction to approach the charge(dock) station. In this thesis, *Charging* is only designed for local operations. For further work, the charging behavior could also be implemented during global navigation.

8.4 Hybrid Planning and Control

Discussions on reactive and deliberative control are presented in Section 8.2 and 8.3. The individual simulation results show satisfactory performance. Simulations on hybrid control are also carried out in Section 7.4 as testing of full missions. The performance is as desired. The mission planning and control architecture can guide the vehicle from the initial position to the location of local operations. Then, the vehicle performs planned operations and returns to the desired end position of the mission after achieving local operations.

Local operations are *Mapping*, *Charging* and *Sampling*. The simulation of *Mapping* is successful, but oscillations are evident when changing the desired waypoint to a new one. As stated in Section 8.3, the oscillation comes from the sudden change of desired heading angle and position. This could affect the accuracy of seabed mapping and is to be optimized in the future. Two suggestions regarding the optimization methods are discussed in Section 8.3. Both *Charging* and *Sampling* use path planning to generate a set of waypoints to approach the charge station and sampling region. The Constant Jerk Guidance is applied in the ROV control system, tracking of a waypoint means one *STOP* and one *GO*. Thus, oscillations are inevitable. The distance between two waypoints could be too short to induce many oscillations in the vehicle's trajectory.

The sub-planner and sub-control for local operations are also open for modifying and adding actions. During the whole simulation process, predefined parameters, waypoints, and other settings could be modified. They would be directly used for replanning and control.

The maximum velocity of the vehicle is $0.3[m/s]$ for actions in the deliberative layer and $0.1[m/s]$ for actions in the reactive layer. These maximum speeds are set according to the capacity of the control system and thrusters equipped in the ROV Minerva. A higher speed is assigned to the deliberative layer to ensure the efficiency of mission execution. On the other hand, the lower speed for the reactive layer ensures safety and avoids collisions and other contingencies.

Chapter 9

Conclusion and Recommendations for Future Work

Based on the simulation results in Chapter 7 and the discussion in Chapter 8, this chapter concludes the literature study, the development and simulation performance of the autonomy framework in Section 9.1. Section 9.2 describes some suggestions and recommendations for future researches.

9.1 Conclusion

The motivation of the research of this thesis is to increase autonomy for ROV operations. This thesis's main contribution is the proposal and development of a hierarchical mission planning and control system for managing ROV complex missions in the subsea environment. The framework is also able to handle contingencies and problems carried by the dynamic environment and system errors.

This thesis proposed a layered mission planning and control system by integrating mission planning and replanning into the system. The concept of hybrid control is adopted: The deliberative layer is responsible for planning and controlling the vehicle to achieve regular missions. The planner uses a best-first search, heuristic search and fast-forward-search algorithm to search for a plan. The planner applied in the deliberative layer generates desired action sequences that guide the ROV to perform operations at desired locations and return. The reactive layer deals with contingencies, i.e., obstacle avoidance and exceedance of cable tension. The control execution layer acts as the coordination mechanism. It determines actions from either the deliberative layer or the reactive layer should be executed.

The results show that the mission planning and control system can maneuver the vehicle to achieve its mission goals without human intervention. Thus, the autonomy level of this mission planning and control system belongs to the third level of autonomy: *Management by exception*. The system can guide the vehicle to perform deliberate actions and accomplish mission requests

without instruction and decision making from human operators. The system also has the ability of contingency handling to some extent, such as obstacle avoidance. However, this system cannot be categorized as *Fully autonomous* because it cannot deal with those unconsidered events, i.e., sensor errors. Also, the design and refinement under each action have not been fully developed. For example, the *Charging* action is implemented as a path following where a path planner generates the path. However, more design on how the vehicle is to land on the dock station should be considered. Many other aspects should also be considered to achieve *Fully autonomous*, including but not limited to risk analysis, sensor fusion and emergency response. Nevertheless, the development of such a system for ROV operations is successful and satisfactory, providing a framework for autonomous mission planning and control. More improvements on the system's property could be added to increase the autonomy of ROV operations.

In conclusion, this mission planning and control system enables the ROV to execute mission requests automatically and perform complex tasks, being less dependent on human intervention.

9.2 Further Work

There are some suggestions regarding the improvement of the mission planning and control system. The mission planning and control system is developed based on depth control. Although a safety check of altitude is carried out during mission execution, a possible direction for future work could be a combination of depth and altitude control. By integrating altitude control, the ROV is able to perform more accurate operations on the seabed. Besides, as discussed in Section 8.3, the simultaneous changes in desired heading and position could induce many oscillations. One suggestion is to perform heading control before position control. At the same time, optimization on the guidance and control system could be carried out to generate a few waypoints to the desired path and avoid sudden change of ROV velocity. One of the most challenging problems of underwater intervention is the identification and localization of SOI. More researches on target detection and localization should be further studied.

For deliberative control, further work is suggested to focus on introducing more actions and how each action could be performed. The mission planning and control system can perform some typical actions, such as *Mapping*. This system also has the potential to introduce more actions in the future. Observation of a SOI is an essential action that requires target detection and motion control for observation. Besides, manipulation is a challenging but essential kind of task. A control system of manipulators could be developed. Then, tasks related to manipulation could be introduced to the mission planning and control system. In [24], a practical method for minimization of manipulator kinetic energy and integral-type criteria for global optimization is proposed, which could be a reference for further work on manipulator control and optimization.

For reactive control, the design of contingency management is to be optimized. For example, fuzzy logic is a possible direction of developing a coordinate mechanism for reactive control. However, the prerequisites of applying fuzzy logic are available sensor data and sensor fusion. Only by receiving sensor signals as much as possible, the system has a high possibility of analyzing the situation accurately and deal with the contingencies correctly. More creative design on

both contingency detection and assessment is expected to be carried out to handle unanticipated changes in the underwater environment. In [25], two steps of dealing with contingencies are introduced, including contingency identification and assessment, and planning/replanning for missions.

Although the performance of obstacle avoidance is satisfactory in the simulation, the system cannot deal with situations that the obstacle is too close to the desired waypoint. When the ROV is less than five meters to an obstacle, the obstacle is detected, and an obstacle avoidance behavior is performed. Thus, the vehicle will never achieve that waypoint and cannot continue its mission execution. This is to be solved in future work. A possible solution is ignoring that waypoint, which is too close to the detected obstacles.

Bibliography

[1] FA Azis, MSM Aras, MZA Rashid, MN Othman, and SS Abdullah. Problem identification for underwater remotely operated vehicle (rov): A case study. *Procedia Engineering*, 41:554–560, 2012.

[2] D Barnett, Stephen McClaran, E Nelson, M McDermott, and G Williams. Architecture of the texas a&m autonomous underwater vehicle controller. In *Proceedings of Symposium on Autonomous Underwater Vehicle Technology*, pages 231–237. IEEE, 1996.

[3] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1995.

[4] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *European Conference on Planning*, pages 360–372. Springer, 1999.

[5] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence. 2001 Jun; 129 (1-2): 5-33.*, 2001.

[6] Blai Bonnet and Héctor Geffner. Hsp: Heuristic search planner. 1998.

[7] Abdelbaki Bouguerra and Lars Karlsson. Hierarchical task planning under uncertainty. In *3rd Italian Workshop on Planning and Scheduling (AI* IA 2004). Perugia, Italy*, 2004.

[8] M Carreras, J Batlle, P Ridao, and GN Roberts. An overview on behaviour-based methods for auv control. In *MCMC2000, 5th IFAC Conference on Manoeuvring and Control of Marine Crafts*. Citeseer, 2000.

[9] G Casalino, E Zereik, E Simetti, S Torelli, A Sperindé, and A Turetta. A task & subsystem priority based control strategy for underwater floating manipulators. *IFAC Proceedings Volumes*, 45(5):170–177, 2012.

[10] Daniel D Corkill. Hierarchical planning in a distributed environment. In *IJCAI*, volume 79, pages 168–175, 1979.

[11] Ioan Dumitrache and Monica Drâgoicea. Agent-based theory applied in mobile robotics. *IFAC Proceedings Volumes*, 41(2):13719–13724, 2008.

[12] Kutluhan Erol. *Hierarchical task network planning: formalization, analysis, and implementation*. PhD thesis, 1996.

[13] Kutluhan Erol, James A Hendler, and Dana S Nau. Umcp: A sound and complete procedure for hierarchical task-network planning. In *Aips*, volume 94, pages 249–254, 1994.

[14] Ocean Exploration and Research. What is an rov? : Ocean exploration facts: Noaa office of ocean exploration and research, 2019.

[15] J. J. Fernandez, M. Prats, P. J. Sanz, J. C. Garcia, R. Marin, M. Robinson, D. Ribas, and P. Ridao. Grasping for the seabed: Developing a new underwater robot arm for shallow-water intervention. *IEEE Robotics Automation Magazine*, 20(4):121–130, 2013.

[16] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189 – 208, 1971.

[17] Trygve O Fossum, Martin Ludvigsen, Stein M Nornes, Ida Rist-Christensen, and Lars Brusletto. Autonomous robotic intervention using rov: An experimental approach. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–6. IEEE, 2016.

[18] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[19] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[20] Jeevith Hegde, Ingrid Bouwer Utne, and Ingrid Schjølberg. Applicability of current remotely operated vehicle standards and guidelines to autonomous subsea imr operations. In *ASME 2015 34th International Conference on Ocean, Offshore and Arctic Engineering*. American Society of Mechanical Engineers Digital Collection, 2015.

[21] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *International Symposium on Methodologies for Intelligent Systems*, pages 216–227. Springer, 2000.

[22] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[23] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[24] Kazem Kazerounian and Zhaoyu Wang. Global versus local optimization in redundancy resolution of robotic manipulators. *The International Journal of Robotics Research*, 7(5):3–12, 1988.

[25] Ross A Knepper, Siddhartha S Srinivasa, and Matthew T Mason. Hierarchical planning architectures for mobile manipulation tasks in indoor environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 1985–1990. IEEE, 2010.

[26] D. M. Lane, J. B. C. Davies, G. Casalino, G. Bartolini, G. Cannata, G. Veruggio, M. Canals, C. Smith, D. J. O'Brien, M. Pickett, G. Robinson, D. Jones, E. Scott, A. Ferrara, D. Angelleti, M. Coccoli, R. Bono, P. Virgili, R. Pallas, and E. Gracia. Amadeus: advanced manipulation for deep underwater sampling. *IEEE Robotics Automation Magazine*, 4(4):34–45, 1997.

[27] David M Lane, Francesco Maurelli, Petar Kormushev, Marc Carreras, Maria Fox, and Konstantinos Kyriakopoulos. Pandora-persistent autonomy through learning, adaptation, observation and replanning. 2012.

[28] Derek Long and Maria Fox. Progress in ai planning research and applications. 04 2020.

[29] Domenico Maisto, Francesco Donnarumma, and Giovanni Pezzulo. Divide et impera: subgoaling reduces the complexity of probabilistic inference and problem solving. *Journal of the Royal Society Interface*, 12(104):20141335, 2015.

[30] Giacomo Marani, Song K Choi, and Junku Yuh. Underwater autonomous manipulation for intervention missions auvs. *Ocean Engineering*, 36(1):15–23, 2009.

[31] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[32] Robin R Murphy. *Introduction to AI robotics*. MIT press, 2019.

[33] NI. Python node, 2018.

[34] Committee Operations, Naval Board, Division Sciences, and National Council. *Autonomous vehicles in support of naval operations*. 09 2005.

[35] Pere Ridao, Joan Batlle, Josep Amat, and GN Roberts. Recent trends in control architectures for autonomous underwater vehicles. *International Journal of Systems Science*, 30(9):1033–1056, 1999.

[36] Ida Rist-Christensen. Autonomous robotic intervention using rov. Master's thesis, NTNU, 2016.

[37] A. Safiotti. Fuzzy logic in autonomous robotics: behavior coordination. In *Proceedings of 6th International Fuzzy Systems Conference*, volume 1, pages 573–578 vol.1, 1997.

[38] Reid G Simmons. Structured control for autonomous robots. *IEEE transactions on robotics and automation*, 10(1):34–43, 1994.

[39] Asgeir Sørensen, Fredrik Dukan, Martin Ludvigsen, Daniel Fernandes, and Mauro Candeloro. *Development of dynamic positioning and tracking system for the ROV Minerva*, pages 113–128. 01 2012.

[40] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic mpc improvement. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 336–343. IEEE, 2017.

[41] Lerus Training. Rov: Classifications - tasks - tools, 2019.

[42] David E Wilkins. *Practical planning: extending the classical AI planning paradigm*. Elsevier, 2014.

[43] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

Appendix

Manuscript for Paper Contribution to Ocean's 20.

Increased Autonomy and Situation Awareness for ROV Operations

Fan Gao
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Signe B. Moltu
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Erlend R. Vollan
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Shuyuan Shen
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Martin Ludvigsen
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway
martin.ludvigsen@ntnu.no

***Abstract*—This paper proposes a semi-autonomous mission planning and management architecture for a Remotely Operated Vehicle (ROV). The work has focused on three main aspects to increase the autonomy of ROV inspections: developing a plan-based mission management architecture for ROV global navigation and local intervention, integrating path planning as a refinement of actions, and real-time communication with visual VSLAM (VSLAM) systems for obstacle detection and motion estimation. The architecture is inspired by the hybrid agent architecture [1] using a deliberative and a reactive layer to perform planned tasks and handle contingencies. Hardware-in-the-loop (HIL) simulations are carried out to test and verify the capability and feasibility of the proposed layered mission management architecture. The results demonstrate that the mission planning and management system can automatically generate the correct plan and guide the ROV to achieve its mission goals.**

***Index Terms*—autonomy, obstacle avoidance (OA), mission planning, visual-SLAM, path planning**

I. Introduction

Enabling autonomy in ROV operations will increase efficiency and save costs [2]. The general purpose of the work in this paper is to develop a mission planning and management system for underwater navigation and operation, and integrate vision-based situation awareness in ROV localization and obstacle detection. Under manual operation, human operators face risks of wrongly analyzing the situation, performing unnecessary or even fatal operations, increasing operation risks and costs. A plan-based mission management system can significantly avoid the above problems and simplify the execution procedure.

By placing a docking station on the seabed, long-term resident ROVs are feasible. With a docking station placed on the seabed, ROVs can charge their batteries and receive and transfer data to the operators onshore. Creating functionality for this is a big step in the direction of fully autonomous long-term missions. Furthermore, in order to increase ROV autonomy, situation awareness and accurate local positioning are required. Existing acoustic positioning systems covers the global position of ROVs well, but tends to lack adequate local positioning precision. Stereo cameras are cheap sensors and can be used for ROV local navigation when paired with computer vision techniques. Visual simultaneous localization and mapping (VSLAM) is the problem of using visual inputs to concurrently construct a map of an unknown environment while estimating its location within it. VSLAM has been extensively researched on land-based vehicles proving good results. Developing subsea-based VSLAM systems could increase the local positioning accurately and simultaneously grant local situational awareness by providing a local map.

An overview of existing ROV standards considering underwater vehicle autonomy is presented in [2]. Four levels of autonomy were suggested in [3], being Manual Operation, Management by consent, Management by exception, and Fully autonomous. The level of situational awareness, decision making and control are increased with increasing levels of autonomy. This paper aims to increase the level of autonomy towards level three (Management by exception) for ROV subsea intervention. A new definition of symbolic actions for ROV mission planning and management is proposed, enabling automated planning to guide and integrate with mission management systems for task execution.

A. Related Work

This paper is an extension of the work done in [4]. The extension is concerned with implementing a governing mission

planner, deciding what the sequence of actions to carry out to reach the goal states. In addition, a path planner is added to ensure safe transit around known obstacles. VSLAM is further included to detect and warn about unknown obstacles.

A similar approach to enhance path planning was tested in [5]. An A* algorithm was proposed for the path planning of an autonomous underwater vehicle (AUV) in a partially-known environment with promising results. VSLAM was suggested for localization, where only a simple VSLAM method was used with a forward-facing sonar.

The benefits of choosing the A* algorithm as the path planning method for underwater vehicles were described in [6]. The algorithm has high maturity, is easy to implement and store, and has a low computational cost. The downsides, however, were that the algorithm has low efficiency and is not suitable for large scale space searches.

Filtering-based approaches and graph-based formulations are two typical kinds of method appiled to solve a SLAM problem. With the improvement of computer computing power, graph-based SLAM has been mainstream in recent decades. A graph is constructed whose vertices represent vehicle positions or landmarks and the edge between two vertices represents a sensor observation that constrains the vertices proposed by Lu and Milios [7]. The sparse linear algebra makes it efficient to solve the optimization of the error minimization problem.

The outline of the paper is as follows: Section II presents the ROV mission management architecture, the mission planning methods, path planning for homing and docking, and vision-based motion estimation and obstacle detection strategy. The simulation results are presented and discussed in Section III. Section IV concludes on the performance of the proposed semi-autonomous mission planner and suggests modifications for further work.

II. Method

A. ROV Control System and HIL Simulator

The ROV control system used for simulations has been developed by the Applied Underwater Robotics Laboratory (AUR Lab) since 2010. The control system was firstly developed for DP and trajectory tracking [8]. As shown in Fig. 1, the control system is built on two basic modules: Frigg Graphical User Interface (GUI) and Njord Control system. The Frigg GUI enables high-level control and mode selection of missions, while the Njord control system performs low-level control, including a guidance system and a Nonlinear PID-controller. The proposed Autonomy Framework is added in Frigg GUI, providing autonomous mission execution command for lower-level control.

The Verdandi Simulator is a HIL simulator which incorporates hardware components into the numerical simulation environment. The HIL simulations are necessary to test the performance of the system and debug the system before fieldwork. The simulator constructs a mathematical model of the ROV and can take disturbance and environmental forces into consideration during virtual experiment. HIL simulations

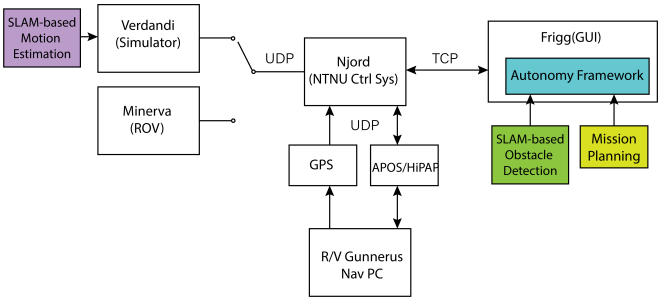

Fig. 1. Structural chart of the ROV control system

are implemented by connecting the Verdandi simulator and the ROV control system using TCP connection.

B. Mission Planning and Management Architecture

A design of hybrid mission planning and management system is proposed, consisting of a deliberative layer performing planned actions to achieve mission goals, a reactive layer responding fast to non-predictable events and a control execution layer acting as a coordinate mechanism that determines if actions from either the deliberative or the reactive layer should be executed by the lower-level controllers. Fig. 2 is a diagram of the hybrid mission planning and management system.

The deliberative layer performs autonomous behaviors based on known situations and events. The mission planner in this layer enables the ROV to plan tasks automatically and performs necessary re-planning. For the given user commands and known environment derived from sensors and a world model, the mission planner can recognize and categorize the tasks, generating a plan that fulfills the mission requests, and deliver a set of actions or commands to the mission controller. The mission will be re-planned automatically due to events such as failures in control, environment changes (such as the encounter with unexpected obstacles), and changes of mission target. In the deliberative layer, a layered design is implemented for global navigation and local operations as sub-missions. With actions of *Station Keeping*, *Launch*, *Descent*, *Transit* and *Operation*, global navigation enables the vehicle to approach a target location where local operation is to be performed. After the performed operation, the ROV is asked to go back to the predefined ending location. The sub-planner acts as the refinement of *Local Operation*. Possible actions for *Local Operation* are *Mapping*, *Sampling* and *Charging*. A fast-forward search [9] method for mission planning is implemented to generate a near-optimal plan that fulfills the goals. An A* path planning algorithm for homing and general path planning is implemented in operation as refining of sampling and charging. A docking option is also implemented to simulate full homing and docking behavior.

The reactive layer responds to contingencies by analyzing sensor data, reasoning unexpected or unknown situations, modifying and interrupting missions. Exceedance of cable tension and obstacle avoidance are the two reactive behaviors implemented in the reactive layer. For actions that require

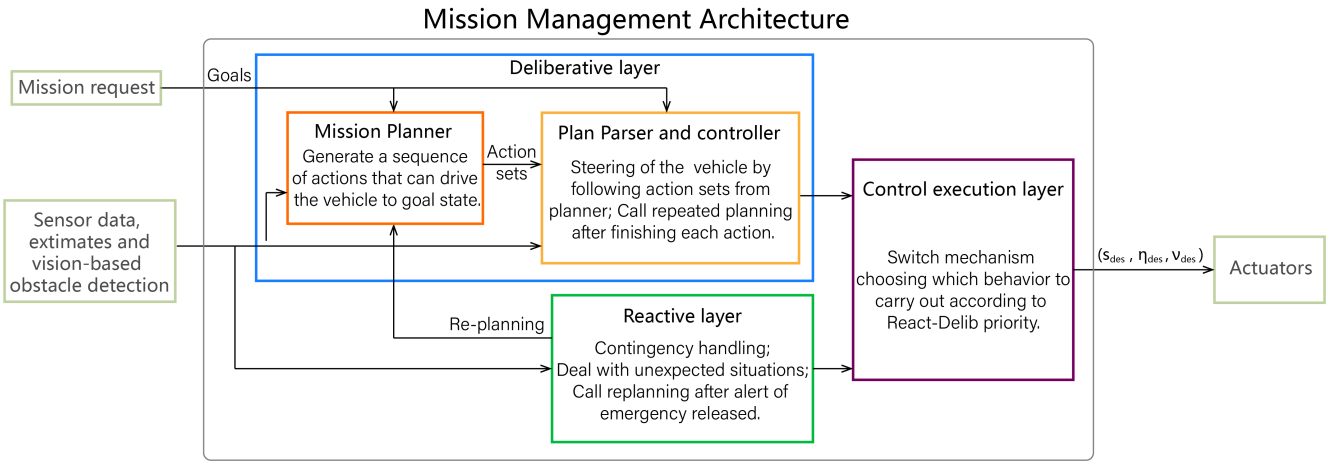

Fig. 2. Mission Planning and Management Architecture

path planning, such as sampling and charging, new detected obstacles are updated to the map, and re-planning is called to generate a new path that avoids both the known and the detected obstacles. For the other actions, a new waypoint is created based on the distance to the obstacle, the size of the obstacle and safety parameters for avoiding obstacles, as seen in Fig. 3.

A method of heading change is used to deal with this situation. When encountering a new obstacle, the direction of heading change is determined by β and ψ. β demonstrates the relative direction of the obstacle to the vehicle, and ψ is the heading angle of the vehicle. If the heading angle ψ is smaller than β, the new heading angle is $\psi_{new} = \beta - \alpha$. Otherwise, the new heading angle is $\psi_{new} = \beta + \alpha$. The angle α is calculated as $sin(\alpha) = \frac{R}{S}$, based on the diameter of the dangerous region and the distance between the vehicle and the obstacle. d is the diameter of the obstacle, and e is a safety parameter. Thus, $R = \frac{D}{2} + e$ is the radius of the dangerous region. L is the calculated length from the vehicle to the new waypoint. The position of the new waypoint is calculated based on the new heading angle and length L.

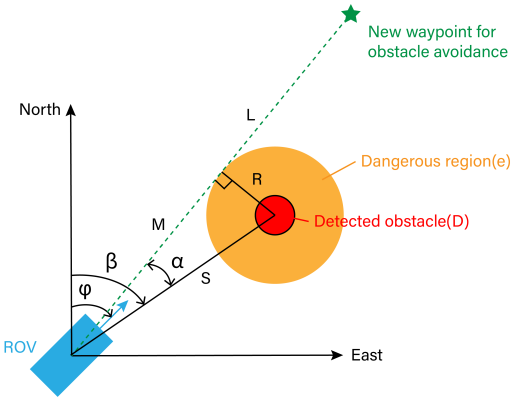

Fig. 3. Obstacle avoidance behavior

The global coordination function evaluates the output behaviors from the two architectures and will transfer inputs from the reactive architecture to the executor once it is activated. Since only one deliberative action is activated at a time in the deliberative layer and a coordinate mechanism is applied in the functional reactive layer to select the behavior with the highest priority among all activated behaviors under this layer. The control execution layer is thus organized as a selection of outputs between the deliberative layer and the reactive layer. It is responsible for deciding which layer and which action is activated and summing up the necessary parameters required by the control system.

C. Integration of Path Planning

The A* algorithm, created by [10], was selected for path generation during missions that require path planning because it is easily adaptable, simplistic and performs well in known environments with low obstacle density [11]. Euclidean distance, presented in (1), was used for the heuristic function since it allows for movement in all eight neighboring tiles in a 2D grid space.

$$\sqrt{(x_{current} - x_{goal})^2 + (y_{current} - y_{goal})^2} \qquad (1)$$

As the entire mission uses depth-control, a 2D path is justified. However, as there will be fluctuations and uncertainties in depth, known obstacles above and below two meters of the desired depth will be seen as relevant obstacles by the algorithm. This is also reasonable considering the specifications of the simulated ROV.

The path created by the A* algorithm can often include unnecessary turns for an underwater vehicle (UUV), which can move in any direction, not just in a straight path from the center of one tile to the center of the next. To make the path more smooth and avoid unnecessary turns, the smoothing method *Moving Average Filter* was used, implemented with the equation shown in (2).

$$ys(i) = (y(i-2)+y(i-1)+y(i)+y(i+1)+y(i+2))/5 \quad (2)$$

Because of the Constant Jerk Guidance [12] used in the ROV control system, a high number of waypoints would mean a high number of stop's and go's. For this reason, collinear waypoints were firstly removed altogether from the generated path from the A* algorithm. However, as suggested in [13], the distance between waypoints should be seen in relation to the requirement of position computation. With longer path segments, the possibility of drift is higher. For this reason, a max waypoint distance is set. A comparison between the original path and the modified path can be seen in Fig. 4.

1) Docking Behavior: As the intended docking station for the mission is a platform docking station, a simple docking behavior is created. A pre-defined path, which can be seen in Fig. 5, is calculated from the size and orientation of the docking station. The path is intended to guide the UUV from the endpoint of the homing path to a waypoint with a fixed distance from the entrance of the dock, then to the next waypoint above the intended dock position, before descending down to land on it. The exact docking mechanism has not been addressed in this paper.

D. Integration of VSLAM-based motion estimation

The flow diagram of the VSLAM system is presented in Fig. 6. The VSLAM system for motion estimations is programmed with C++ using open-source libraries OpenCV and g2o. Images from underwater environments often have different contrast in different image parts due to poor lighting conditions. It is hard to detect features in the low contrast part, so image enhancement is necessary. Every time a new frame is sent to the system, contrast limited adaptive histogram equalization (CLAHE) [14] is implemented to enhance the contrast of stereo images. It works by mapping the histogram of the image to another histogram with a wider distribution of intensity values, so the intensity values are spread over the whole range.

ORB algorithm [15] is implemented to detect and describe features. Then feature matching is performed by using

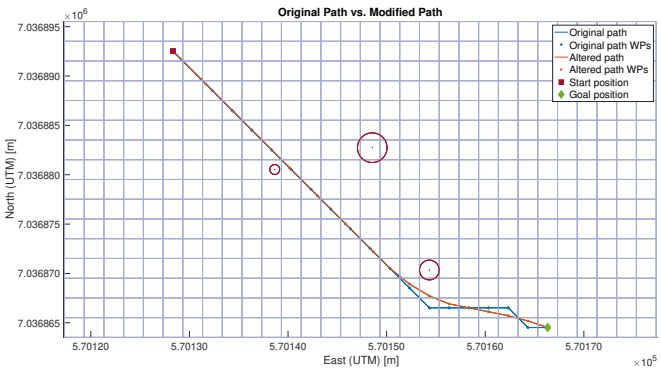

Fig. 4. Comparison between original path and improved path

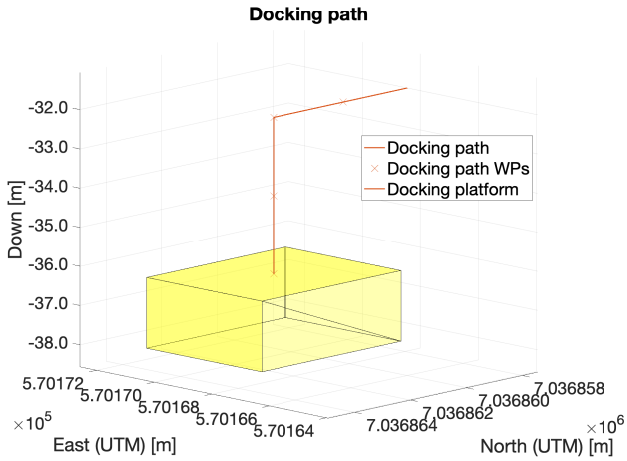

Fig. 5. Docking path

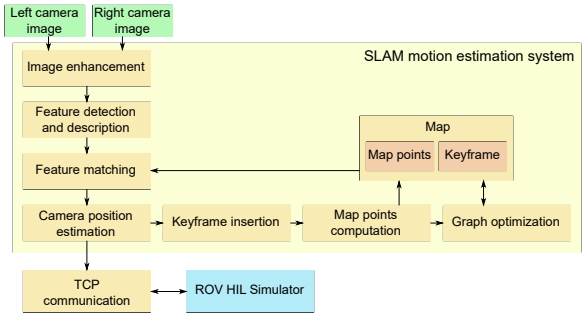

Fig. 6. Flow chart for VSLAM-based motion estimation

Hamming distance for ORB descriptors to find projection relationships between 2D pixel features and 3D map points from the VSLAM map. Generally, the matching results are not good enough to directly estimate the camera position because mismatching is hard to avoid. Distance filter and the RANSAC algorithm [16] are used together to remove mismatches. Next, the camera position is estimated by using the EPnP algorithm [17] based on the matched 3D-2D points in feature matching.

The map of the VSLAM system stores all map points and keyframes. It is initialized when the first stereo image frame is processed. Then the map is updated when new keyframes are added. A keyframe is chosen if the number of matched features in the current frame is small, and new map points are required. Features on the left and the right camera images are matched to compute new map points. Graph optimization [18] is implemented to minimize the projection error between observed and predicted image pixel locations to find the best map point positions and keyframe camera positions.

The local positions of the camera are converted to the local positions of the ROV based on the camera coordinates in the ROV body frame. The ROV positions are transferred to the ROV mission planner using TCP communication. The

system obtains the initial position of the ROV from the mission planner when it starts running so that the local position can be converted to a global position.

E. Integration of VSLAM-based Obstacle Detection

The real-time VSLAM-based Obstacle Detection revolves around utilizing the outputs of the renowned Visual VSLAM method ORB-SLAM2 [19]. The outputs of ORB-SLAM2 are the estimations of the ROV pose and point clouds of the surrounding environment of the ROV. From these two outputs, the closest detected obstacle is inferred. The obstacle detection system is implemented in C++ using the framework Robot Operating System (ROS). It is comprised of: a camera driver for running the stereo camera rig of the ROV Minerva, an image processing part using the open-source library OpenCV, an existing ROS implementation of ORB-SLAM2[1], a point cloud processing part using the open-source library PCL and a communication part communicating with the ROV Autonomy Framework providing the closest detected obstacle. The communication is conducted using TCP connection. The system architecture is displayed in Fig. 7.

The stereo camera rig of Minerva consists of two Allied Vision Prosilica GC1380C mounted horizontally displaced. By considering the overlapping field of view and expected disparity values, the horizontal displacement, or baseline, is set to 0.2 meters. The cameras are configured in binned mode, reducing the resolution, but increasing the light sensitivity and signal to noise ratio. The image processing undistorts and rectifies the stereo image pairs based upon obtained underwater calibration parameters, and additionally, contrast enhances the images using CLAHE. The point cloud processing first removes point cloud points associated with the seabed by fitting a plane using RANSAC; the remaining points are clustered using the Euclidean based clustering method of [20]. The obstacle sizes $\mathbf{o}_d$ and position $\mathbf{o}_p$ are inferred using

$$\begin{aligned} \mathbf{o}_d &= \mathbf{c}_{max} - \mathbf{c}_{min} \\ \mathbf{o}_p &= \frac{\mathbf{c}_{max} + \mathbf{c}_{min}}{2} \end{aligned} \tag{3}$$

where $\mathbf{c}_{max}$ and $\mathbf{c}_{min}$ are the maximum and minimum point position of the cluster. The closest detected obstacle is determined by finding the obstacle with the shortest Euclidean distance to the current estimated ROV pose. The LabView communication handles the TCP connection with the Autonomy Framework and generates messages containing information about the closest detected obstacle on the format in (4) if the distance threshold of five meters is violated.

$$\left[\; x_o^g, \quad y_o^g, \quad z_o^g, \quad d_o \; \right] \tag{4}$$

The message is an array of doubles containing the obstacle position in the NED frame x_o^g, y_o^g and z_o^g, and the obstacle spherical diameter d_o. The spherical diameter is determined by selecting the largest estimated dimension of $\mathbf{o}_d$.

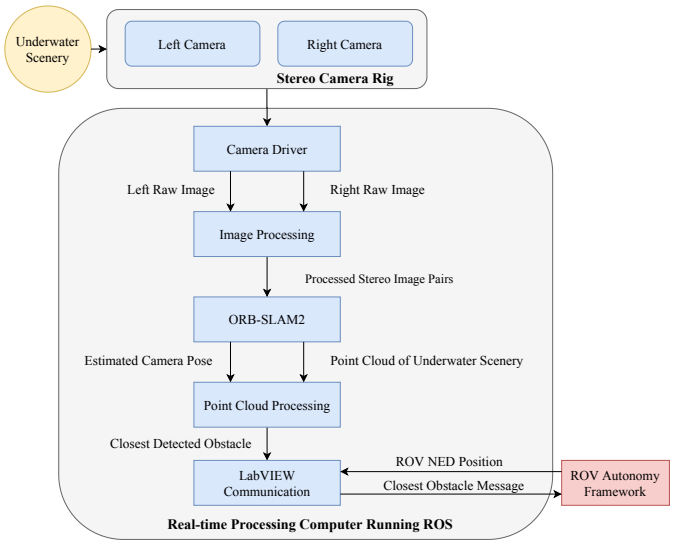

Fig. 7. System architecture of the vision based obstacle detection system

[1]http://wiki.ros.org/orb slam2 ros

III. Simulation Results

A. VSLAM-based motion estimation

The VSLAM system performance is tested on a seabed image set from a mission at Stokkbergneset in February 2017 by NTNU ROV SF-30k. For the simulation, 60 paired left and right camera images with a time interval of 2 seconds are used. The result presented in Fig. 8 is simulated on this image set and the corresponding navigation data from the ROV control system.

The result shows that the VSLAM motion estimation successfully estimates the position of ROV in this simulation. The estimated orientation of ROV movement is slightly different from the true orientation, which causes errors in both north and east positions.

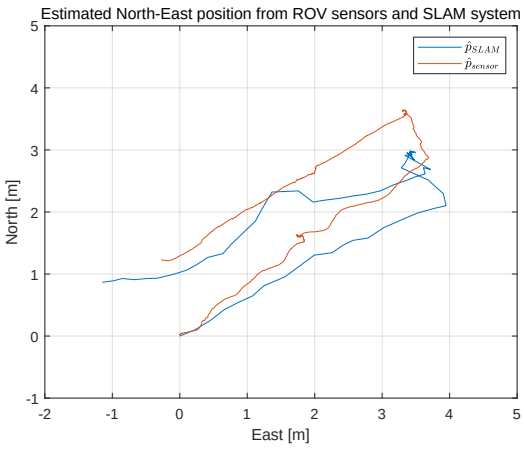

Fig. 8. Simulation of VSLAM-based motion estimation

B. VSLAM-based obstacle avoidance

The real-time VSLAM-based obstacle detection system is tested at the Marine Cybernetics Lab (MC-Lab) at Marin teknisk senter, NTNU. The stereo camera rig is mounted to a rod and moved through a measured underwater obstacle course installed in the basin of MC-Lab. Subsea light conditions are simulated by having no ambient light and an artificial light source installed on the rod. The results are presented in Fig. 9 where the measured obstacle course is plotted together with the estimated trajectory (OS2), estimated point cloud and the currently estimated closest obstacle outputted if the distance to the previously detected closest obstacle is larger than 0.15 meters. The obstacle course consisted of two boxes, a stepladder and rod.

Fig. 10 presents position estimates and desired positions of the vehicle in the North-East (NE) plane, giving satisfactory simulation results of reactive obstacle avoidance. A VSLAM-based obstacle detection code is programmed to test the distance between the vehicle's current position and obstacles. When the distance becomes shorter than five meters, the vehicle will perform reactive obstacle avoidance. Table III-B shows the location for three detected obstacles. Under some circumstances, more than one obstacle might be encountered at a time. The system chooses the nearest one as the current obstacle and executes the obstacle avoidance correspondingly.

The vehicle is performing the *Transit* action when the first obstacle is detected. The vehicle firstly moves to the desired position and then detects the *Obstacle 01*, performing obstacle avoidance correspondingly. Before reaching the waypoint for avoiding *Obstacle 01*, the vehicle detects a new *Obstacle 02* and generates a new waypoint to avoid it. The third obstacle is detected after reaching the waypoint for avoiding *Obstacle 02*. The test result shows that the obstacle avoidance algorithm successfully guides the vehicle to avoid collisions during mission execution. The mission planning and management system can encounter more than one obstacle at a time and update its waypoints based on the position of the closest obstacle. The system is also able to tackle obstacle avoidance when performing other actions. Reactive obstacle avoidance is also tested in Section III-C.

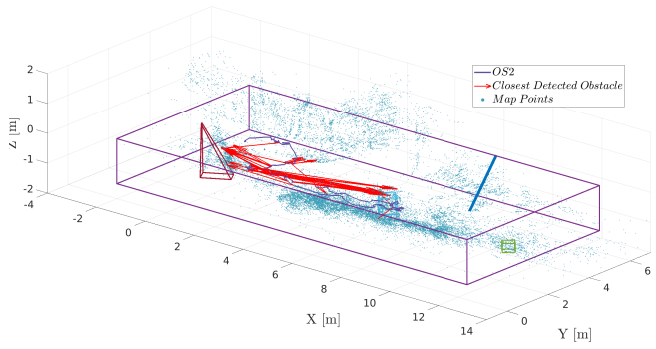

Fig. 9. Laboratory experiment of VSLAM-based obstacle detection.

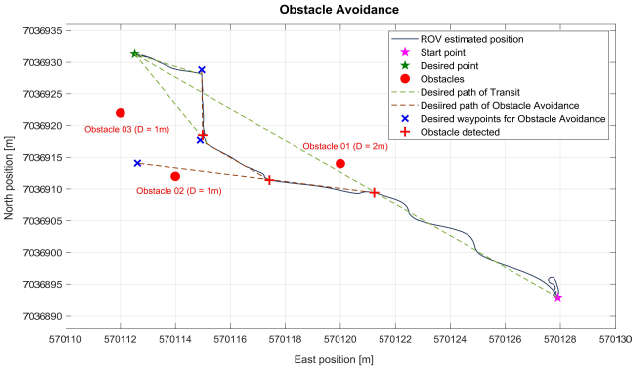

Fig. 10. Simulation of obstacle avoidance

TABLE I
OBSTACLES IN SIMULATION OF OBSTACLE AVOIDANCE

Obstacle	North	East	Depth	Diameter	[-]
01	7036914	570120	15	2	m
02	7036912	570114	15	1	m
03	7036922	570112	15	1	m

C. Autonomous mission planning and management

1) Mapping, charging and OA: Obstacle Avoidance and *Charging* are tested during *Mapping* as reactive actions. The positions of the two manually set obstacles are listed in Table III-C1. The testing is carried out with hardware-in-the-loop (HIL) simulation. Fig. 11 shows that there are a lot of fluctuations when the vehicle changes its heading.

The performance of waypoint tracking is mainly determined by the low-level control system.

In the mission, the vehicle starts at (7036892, 570128, 10) in UTM coordinates and then moves towards waypoints generated from the mission planner sequentially. The vehicle firstly performs *Launch* and *Descent* to the desired depth of operation. Then, the vehicle transits to the starting position of mapping and performs mapping, tracking six waypoints sequentially, found in Table III-C1. The desired trajectory of mapping is defined before starting the mission, while the path planner plans the waypoints of charging for homing once the *Charge* action is activated. During mapping, 'low battery' is manually set, indicating that the vehicle has to abort mission and move to the docking station to charge. The charging station is set at (7036882, 570140, 27).

TABLE II
OBSTACLES IN SIMULATION OF MAPPING, CHARGING AND OA

Obstacle	North	East	Depth	Diameter	[-]
01	7036900	570144	15	2	m
02	7036886	570125	15	2	m

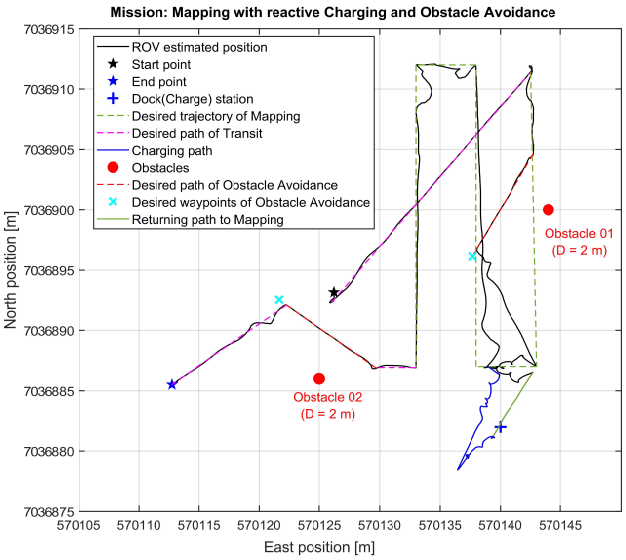

Fig. 11. Simulation of mapping, charging and OA

TABLE III
WAYPOINTS OF MAPPING

Waypoint	North Position	East Position	Depth	[-]
01	7036912	570143	15	m
02	7036887	570143	15	m
03	7036887	570138	15	m
04	7036912	570138	15	m
05	7036912	570133	15	m
06	7036887	570133	15	m

2) Full mission with OA: A similar mission as above, now with all four global states and three local states executed, was simulated. The resulting plots can be seen in 2D in Fig. 12 and in 3D in Fig. 13. Four "unknown" obstacles were detected by the VSLAM obstacle avoidance. The positions of theses obstacles are presented in Table III-C2. The action sequence of the full mission is *Launch-Descent-Transit-Mapping (OA and Charging)-Sampling (OA)-Transit (OA)-Descent*.

This simulation includes testing of obstacle avoidance for actions *Transit*, *Mapping* and *Sampling*. For *Transit* and

TABLE IV
OBSTACLES IN FULL MISSION

Obstacle	North Position [m]	East Position [m]	Depth [m]	Diameter [m]
01	7036900	570144	15	2
02	7036907	570122	16	1
03	7036900	570112	15	1
04	7036893	570105	15	2

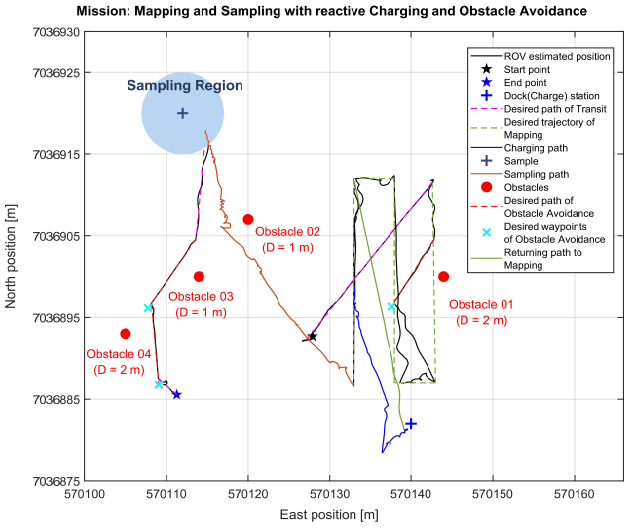

Fig. 12. 2D simulation of mission employing all states

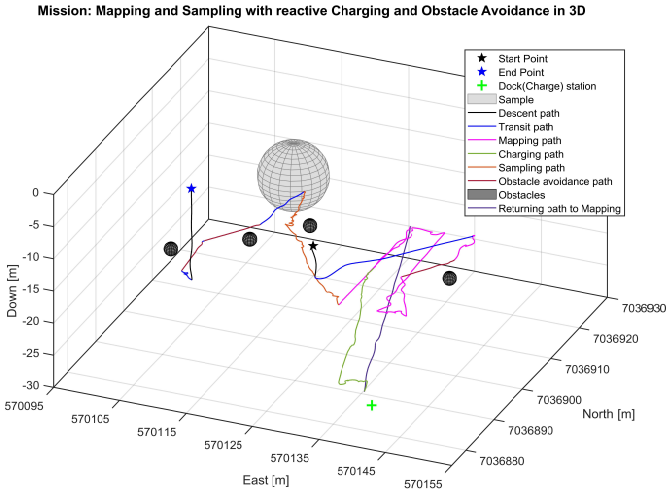

Fig. 13. 3D simulation of mission employing all states

Mapping, the reactive behavior is turning its heading and move to a temporary waypoint for obstacle avoidance. During *Sampling*, obstacle avoidance is implemented by adding the newly detected obstacle in the obstacle list and replan to generate a new path with the A* algorithm which avoids the new obstacle.

IV. CONCLUSION

We have successfully implemented the autonomous mission planning and management system in the ROV control system, integrating vision-based situation awareness for motion estimation and obstacle detection. The system uses mission planning, guiding the vehicle to achieve the mission requests, and path planning is applied for local operations.

The use of the A* algorithm in an autonomous mission planning and management proved to be a good choice, as

it was convenient to implement it in the already existing framework. Adapting it to react to newly detected obstacles was also done in a successful manner, creating a robust path planner for the intended short range missions.

The VSLAM-based obstacle detection system performs well being capable of providing the currently closest obstacle in its locally estimated surroundings. However, as the laboratory experiments results imply, the removal of every seabed associated map point is not successful as some of the closest detected obstacles are located at the basin floor, and the positional consistency of the newly and previously mapped obstacles degrades over the running time due to accumulated drift.

The VSLAM-based motion estimation presents good simulation results, using images derived from previous sea-trials. The testing result is valid in short periods. The deviation increases with the testing process. However, real-time VSLAM-based motion estimation has not be tested yet.

Since conducting the joint missions for this paper, both the docking behavior and A* path planning behaviors have been altered. A safety distance around obstacles have been added in the A* algorithm, and an ultra-short-baseline transducer has been simulated to situate at the dock to ensure more reliable position measurements when docking.

A. Further work

To verify the capability and performance of the autonomous mission planning and management system, sea-trails should be conducted. During simulations, sensor noise and environmental forces (such as current and waves) are not simulated. Also, the simulations use depth control during the execution of the mission. For further improvement, a combination of depth and altitude control should be designed such that the ROV could execute actions more accurately also relying on measurements from a doppler velocity log.

Up to now, the mission planning and management system is only capable of performing observations. Sampling and docking behaviors are simplified as trajectory tracking generated by path planning. More refinement should be designed for the actual realization of these actions.

The path planning algorithm is developed in a 2D plane while the ROV moves in a 3D underwater environment. The path planning in 3D can be further optimised including a full 3D model to the path generation procedure.

The VSLAM system for motion estimation is tested in a short time simulation. Loop closure techniques are expected to be added in the VSLAM system to reduce estimation error for long time position estimation.

REFERENCES

[1] I. Rist-Christensen, "Autonomous robotic intervention using rov," Master's thesis, NTNU, 2016.

[2] J. Hegde, I. B. Utne, and I. Schjølberg, "Applicability of current remotely operated vehicle standards and guidelines to autonomous subsea imr operations," in *ASME 2015 34th International Conference on Ocean, Offshore and Arctic Engineering*. American Society of Mechanical Engineers Digital Collection, 2015.

[3] C. Operations, N. Board, D. Sciences, and N. Council, *Autonomous vehicles in support of naval operations*, 09 2005.

[4] T. O. Fossum, M. Ludvigsen, S. M. Nornes, I. Rist-Christensen, and L. Brusletto, "Autonomous robotic intervention using rov: An experimental approach," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, 2016, pp. 1–6.

[5] J.-H. Li, M.-J. Lee, S.-H. Park, and J.-G. Kim, "Real time path planning for a class of torpedo-type AUVs in unknown environment," in *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, Sep. 2012, pp. 1–6, iSSN: 2377-6536.

[6] D. Li, P. Wang, and L. Du, "Path Planning Technologies for Autonomous Underwater Vehicles-A Review," *IEEE Access*, vol. 7, pp. 9745–9768, 2019, conference Name: IEEE Access.

[7] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[8] A. Sørensen, F. Dukan, M. Ludvigsen, D. Fernandes, and M. Candeloro, *Development of dynamic positioning and tracking system for the ROV Minerva*, 01 2012, pp. 113–128.

[9] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[11] A. Koubâa, H. Bennaceur, I. Chaari, S. Trigui, A. Ammar, M.-F. Sriti, M. Alajlan, O. Cheikhrouhou, and Y. Javed, *Robot Path Planning and Cooperation*. Springer, 2018, vol. 772.

[12] F. Dukan, "ROV motion control systems," Ph.D. dissertation, Norwegian University of Science and Technology, Faculty of Engineering Science and Technology, Department of Marine Technology, Trondheim, 2014.

[13] J. Yuh, T. Ura, and G. A. Bekey, *Underwater Robots*. Springer Science & Business Media, Dec. 2012, google-Books-ID: YwfaBwAAQBAJ.

[14] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer vision, graphics, and image processing*, vol. 39, no. 3, pp. 355–368, 1987.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[16] K. G. Derpanis, "Overview of the ransac algorithm," *Image Rochester NY*, vol. 4, no. 1, pp. 2–3, 2010.

[17] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[19] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras," vol. 33, no. 5, pp. 1255–1262.

[20] R. B. Rusu, *Clustering and Segmentation*. Springer Berlin Heidelberg, vol. 85, pp. 75–85, series Title: Springer Tracts in Advanced Robotics.

NTNU

Norwegian University of
Science and Technology