FACULDADE DE CIÊNCIAS DA UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

LLM-Based Mission Planning and Execution for USV Inspection through Reinforcement Learning Skill Composition

Bárbara Lisboa Santos

Mestrado em Inteligência Artificial

Supervisor: Prof. Andry Maykol Pinto

July 28, 2026

LLM-Based Mission Planning and Execution for USV Inspection through Reinforcement Learning Skill Composition

Bárbara Lisboa Santos

Mestrado em Inteligência Artificial

Approved in oral examination by the committee:

President: Prof. Miriam Santos
Examiner: Prof. João Marques

July 28, 2026

Resumo

A inspeção e manutenção periódica de infraestruturas marítimas com Veículos de Superfície Não Tripulados (VSNTs) dependem da interação eficaz entre o operador humano e o sistema autónomo. Programar cada missão como uma sequência explícita de comandos de baixo nível impõe uma elevada carga cognitiva ao operador e apresenta escalabilidade baixa à medida que as missões aumentam em duração e complexidade; em vez disso, o operador deve ser capaz de definir o objetivo concreto da inspecção e ditar a sua decomposição e sequenciação ao veículo, que organiza as capacidades disponíveis a bordo num plano executável. Para que esta forma de interação seja concretizada, o plano resultante necessita de ser executado de forma fiável sob as perturbações características dos ambientes marinhos não estruturados. No entanto, a literatura existente demonstra que os paradigmas atuais abordam a especificação de alto nível e a execução robusta de forma isolada, e que nenhuma abordagem oferece uma especificação de objetivos flexível juntamente com uma execução que se mantém robusta a longo prazo, sem replaneamento contínuo ou intervenção humana.

Esta dissertação propõe uma estrutura que acopla o planeamento de missões baseado em Grandes Modelos de Liguagem (GML) com a execução de habilidades baseadas em Aprendizagem por Reforço (AR), organizada em torno da separação do raciocínio semântico e da atuação física. A inovação reside na aplicação desta combinação à inspeção marítima, onde um objetivo de inspeção é decomposto e sequenciado em habilidades aprendidas que o executam sem que o operador especifique a sequência de comandos subjacente. As suas contribuições conceptuais são uma hierarquia abstrata de quatro camadas que traduz a intenção do operador em comandos passo-a-passo do veículo, e uma ontologia de missão que especifica o que um comando bem formulado deve conter. Estas contribuições são implementadas em duas componentes complementares. A primeira é um Planeador de Missões Agêntico, uma arquitetura hierárquica de dois agentes na qual um Agente de Missão *fine-tuned* decompõe um comando em linguagem natural num conjunto ordenado de tarefas. Um Agente de Tarefas *fine-tuned* expande cada tarefa em habilidades de navegação, perceção e gestão de dados simultaneamente executáveis extraídas de uma biblioteca predefinida, sujeitas à aprovação explícita do operador antes de qualquer execução. A segunda é uma habilidade de sondagem por Aprendizagem por Reforço Hierárquico (APH), formulada pelo *options framework*, na qual uma política de alto nível atua sobre uma observação centrada em LiDAR e direciona uma política de navegação de baixo nível pré-treinada para sondar um objeto *a priori* desconhecido a uma distância de referência escolhida.

A habilidade de sondagem completa órbitas completas de estruturas não vistas com uma taxa de sucesso de até 92% e é transferida dos simuladores VMAS para Gazebo sem retreinar. Para o planeador de missões, o *LoRA fine-tuning* demonstra ser necessário para decomposição estrutural fiável, com os agentes selecionados a bordo a atingirem pontuações F1 de 0,89 e 0,97 a nível das missões e tarefas, respetivamente, juntamente com alucinação próxima de zero e validade completa do esquema. A *pipeline* inteira é validado de ponta a ponta em simulação e o Planeador de Missões Agêntico é lançado no VSNT Nautilus no Centro de Testes ATLANTIS em Viana do

Castelo, Portugal, onde executou uma missão de inspeção com quatro tarefas sob condições e perturbações reais do mar. Os resultados apoiam a proposta de desacoplamento entre o planeamento semântico e a execução aprendida e robusta como uma rota prática para a autonomia do VSNT em missões.

Palavras-chave: Aprendizagem por Reforço Hierárquica • Modelos de Linguagem de Grande Dimensão • Arquitetura Agêntica • Operações Marítimas • Veículos Autónomos de Superfície

Abstract

The periodic inspection and maintenance of maritime infrastructure with Unmanned Surface Vehicles (USVs) depends on effective interaction between the human operator and the autonomous system. Programming each mission as an explicit sequence of low-level commands places a high cognitive load on the operator and scales poorly as missions grow in length and complexity; instead, the operator should be able to state the concrete inspection objective and dictate its decomposition and sequencing to the vehicle, which arranges the available onboard capabilities into an executable plan. Realising this form of interaction requires the resulting plan to be executed reliably under the disturbances characteristic of unstructured marine environments. However, the existing literature shows that current paradigms address high-level specification and robust execution in isolation, and that no single approach provides flexible objective specification together with execution that remains robust over long horizons without continual replanning or human intervention.

This dissertation proposes a framework that couples Large Language Model (LLM)-based mission planning with Reinforcement Learning (RL)-based skill execution, organised around the separation of semantic reasoning from physical actuation. The novelty lies in bringing this coupling to maritime inspection, where a stated inspection objective is decomposed and sequenced into learned skills that execute it without the operator specifying the underlying command sequence. Its conceptual contributions are a four-layer abstraction hierarchy that translates operator intent into vehicle commands step by step, and a mission ontology that specifies what a well-formed command must contain. These are realised in two complementary components. The first is an Agentic Mission Planner, a hierarchical two-agent architecture in which a fine-tuned Mission Agent decomposes a free-form natural language command into an ordered set of tasks. A fine-tuned Task Agent then expands each task into concurrently executed navigation, perception, and data-management skills drawn from a predefined library, subject to explicit operator approval before any actuation. The second is a hierarchical RL survey skill, formulated within the options framework, in which a high-level policy acts over a Light Detection and Ranging (LiDAR)-centric observation and leads a pre-trained low-level navigation policy to survey an *a priori* unknown object at a chosen standoff distance.

The survey skill completes full orbits of unseen structures with a success rate of up to 92% and transfers from the Vectorised Multi-Agent Simulator (VMAS) to the Gazebo simulators without retraining. For the mission planner, Low-Rank Adaptation (LoRA) fine-tuning is shown to be a necessary condition for reliable structured decomposition, with the selected onboard agents attaining F1 scores of 0.89 and 0.97 at the mission and task levels respectively, alongside near-zero hallucination and full schema validity. The complete pipeline is validated end-to-end in simulation and the Agentic Mission Planner is deployed on the Nautilus USV at the ATLANTIS Test Centre in Viana do Castelo, Portugal, where it executed a four-task inspection mission under real sea-state disturbances. The results support the proposed decoupling of flexible semantic planning from robust learned execution as a practical route to mission-level USV autonomy.

Keywords: Hierarchical Reinforcement Learning • Large Language Models • Agentic Architecture • Maritime Operations • Autonomous Surface Vehicles

The United Nations Sustainable Development Goals

The United Nations Sustainable Development Goalss (SDGs) provide a global framework for achieving a better and more sustainable future for all. It includes 17 goals [1] to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

The specific Sustainable Development Goals mentioned have the following names:

SDG 7 Ensure access to affordable, reliable, sustainable and modern energy for all

SDG 13 Take urgent action to combat climate change and its impacts

SDG	Target	Contribution	Performance Indicators and Metrics
7	7.a	Enhancing the reliability of renewable energy infrastructure by developing an agentic mission planner that orchestrates specialized Reinforcement Learning (RL) skills for autonomous structural inspections.	Mission completion rate of the agentic planner; success rate and accuracy of the specific RL skill developed during simulated inspection tasks.
13	13.b	Strengthening the technical readiness of autonomous systems to enable the transition to low-carbon maritime maintenance fleets.	Reduction in required human supervision; robustness of behaviours in real sea states.

[1] https://sdgs.un.org/goals

Acknowledgements

É, B, acho que chegaste ao fim de mais uma etapa. E como em todas as outras, não chegaste aqui sozinha.

Gostaria de começar por agradecer ao meu orientador, Prof. Andry, por todo o feedback e incentivo que foram fundamentais para estruturar e desenvolver este trabalho, e por apoiar a minha vontade de realisar algo que eu gosto imenso de fazer. Obrigada ainda por me dar a oportunidade de me juntar a uma equipa mais do que 5 estrelas.

À Maria, agradeço especialmente por todas as discussões, por todos os momentos que tirei para tirar dúvidas que eu sei que acabam por obstruir o teu trabalho mais do que aquilo que parece, por todas as idas a Viana, e principalmente por acreditares tanto em mim. Deste-me um apoio incomparável, até aos mais minimos dos detalhes, e um motto que vou levar para a vida.

À restante equipa que me acompanhou quase diariamente durante os últimos meses — Francisco, Leite, Alex, Xico, Tiago, Dionísio, Celso, MecaPedro, Lourenço, Diogo, Luís, Daniel e Rafa — um enorme obrigada, do fundo do meu coração, por todos as conversas nos almoços e idas ao café que eu não tomei.

À Sara, obrigada por todas as perguntas de como estava a correr a escrita, e pequenos almoços de mútuo apoio que fomos tendo. Aos amigos — Inês, Lano e Tiago, entre outros — que estiveram presentes nas pequenas coisas, nas mensagens, nos desabafos e nas conversas de todos os dias, obrigada por tornarem esta fase mais leve.

Ao Toni, obrigada por me aturares diariamente praticamente desde o início da faculdade, e por me deixares tão frequentemente abusar da tua paciência. Obrigada por te deixares ser um rubber duck e me ouvires reclamar de problemas que não percebias, e por me ajudares a acalmar a pipoca quando estava demasiado perto de fritar. Beep beep.

Ao meu Caramelo, e à minha Estrelinha, que já me acompanham desde a primária e são a melhor companhia felpuda pela qual eu poderia pedir, prometo continuar a dar-vos muitos miminhos.

Por fim, mas longe de ser menos importante, quero agradecer aos meus pais, pelo apoio incondicional que me deram, não só agora mas desde sempre. Obrigada por criarem todas as oportunidades possíveis para eu poder estar hoje onde estou e por me ensinarem que para fazer acontecer basta esforçar-me para isso. Obrigada por acreditarem fielmente que eu ia "dar gente".

Bárbara Lisboa Santos

Official Acknowledgements

This work is funded by the European Union's Horizon Europe research and innovation programme, under the Grant Agreement No. 101189723 (AEROSUB).

Bárbara Lisboa Santos

Declaração de Honra

Declaro que a presente dissertação é de minha autoria e não foi previamente apresentada e avaliada nesta ou em outra instituição. As referências a outros autores (afirmações, ideias, pensamentos), assim como a utilização de trabalhos publicados respeitam escrupulosamente as regras da atribuição, e encontram-se devidamente indicadas no texto e nas referências bibliográficas, de acordo corn as normas de referenciação. Tenho consciência de que a prática de plágio ou de autoplágio constituem ilícitos académicos, conforme estabelecido no Código de Ética da U.Porto.

Declaro, ainda, que utilizei ferramentas de Inteligência Artificial para a realização de parte(s) da presente dissertação, e essa utilização e os seus resultados estão devidamente assinalados e foram objeto de cuidada verificação para deteção de erros, imprecisões, atribuições infundadas ou outras formas de enviesamento de conteúdos e argumentos, bem como de determinação de autoria.

Bárbara Lisboa Santos

"Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"

Lewis Carroll

Contents

List of Figures

List of Tables

List of Acronyms

AC Actor-Critic
AI Artificial Intelligence
ASV Autonomous Surface Vehicle
DQN Deep Q-Network
GAE Generalised Advantage Estimate
HRL Hierarchical Reinforcement Learning
LiDAR Light Detection and Ranging
LLM Large Language Model
LoRA Low-Rank Adaptation
MDP Markov Decision Process
MLP Multilayer Perceptron
PPO Proximal Policy Optimisation
RL Reinforcement Learning
ROS Robot Operating System
SAC Soft Actor-Critic
SDG Sustainable Development Goals
TD3 Twin Delayed Deep Deterministic Policy Gradient
USV Unmanned Surface Vehicle
VMAS Vectorised Multi-Agent Simulator

Chapter 1

Introduction

1.1 Context and Motivation

The global transition towards renewable energy has propelled a sustained expansion of offshore generation, with installations placed progressively farther from shore and spread across an increasingly diverse set of technologies, from fixed and floating wind to solar arrays deployed directly on water. Cumulative offshore wind capacity alone has now surpassed ninety gigawatts worldwide and continues to rise year on year[1]. This growth steadily enlarges the population of assets that must be kept in service throughout their operational lifetime, and with it the volume of inspection and maintenance activity required to sustain safe and efficient generation. As these fleets scale and move into more remote and exposed waters, the recurring task of monitoring their condition becomes a defining operational concern rather than an occasional one.

Maritime infrastructure, including offshore energy platforms, subsea assets, port structures, and aquaculture installations, requires regular inspection to detect structural degradation and damage before it compromises safety or operation [8]. These inspections have traditionally relied on crewed vessels and human divers, an approach that is costly, slow, and exposes personnel to hazardous conditions [8]. Unmanned Surface Vehicles (USVs) offer an attractive alternative, enabling repeated, autonomous data acquisition around structures of interest without placing operators at risk and at a fraction of the cost [43].

Realising this potential, however, requires the vehicle to operate autonomously over extended time horizons in unstructured marine environments characterised by uncertainty, partial observability, and persistent wind, wave, and current disturbances. An inspection deployment is rarely a single manoeuvre: it unfolds as a sequence of tasks, each a self-contained assignment that advances the operator's broader objective, and each accomplished through skills, the reusable navigational and perceptual behaviours that can be parametrised and invoked as conditions require. The problem addressed in this dissertation is centered on how to carry out such an extended sequence reliably while remaining responsive both to disturbances and to the operator's intent, a

[1] Global Wind Energy Council. *Global Offshore Wind Report 2026*. https://www.gwec.net/reports/globaloffshorewindreport. Accessed: 2026-06-27. 2026.

problem that gives rise to two challenges.

The first challenge concerns dependable operation at the lowest level, where the vehicle must keep its behaviour steady in close proximity to the structure under inspection. Marine environments are highly stochastic and subject to unmodelled disturbances, and the short ranges characteristic of inspection leave little margin for error against currents, waves, and structures whose geometry is not known in advance [43]. Manoeuvring of this kind still depends to a large extent on human pilots or on hand-engineered behaviours that are labour intensive to tune and are brittle once conditions depart from those anticipated [77]. The former keeps personnel directly involved in the close-quarters, hazardous work that autonomy is intended to remove, while the latter generalises poorly across the range of disturbances met in the field.

The second challenge concerns mission specification and the assurance of safe execution, the latter increasingly governed by regulation such as the EU AI Act[2], which require that high-risk autonomous systems remain under effective human oversight. The maritime sector is moving steadily towards autonomous operation, motivated both by the desire to withdraw crews from dangerous settings and by the longer-term gains in efficiency and sustainability that autonomy promises [51]. This shift does not remove the human from the picture, it repositions them: operators are increasingly expected to act not as continuous pilots but as supervisors who set objectives, retain oversight, and carry responsibility for the vehicle's conduct [3]. A difficulty stands in the way of this arrangement. Operators are experts in inspection rather than in robotics and reason in terms of what they wish to have surveyed and why, whereas conventional systems accept direction only as low-level geometric commands such as fixed waypoints or predefined paths [74]. Spanning that distance by hand, in conditions that call for continual adaptation, places a heavy cognitive burden on the operator and scales poorly as missions and fleets grow. What is needed instead is a way for operators to convey intent in their own high-level terms and have the vehicle interpret it, advancing a form of human-robot interaction in which supervisor and vehicle hold a common picture of the mission's goal rather than communicating solely through low-level geometry.

Meeting these two challenges calls for capabilities that no single approach supplies on its own. A method that excels at translating free-form intent into well-formed objectives gives little assurance that the behaviour it requests is feasible for the vehicle or safe to carry out, whereas a method that produces robust low-level behaviour offers no inherent means of deciding which behaviour to perform, and when. This dissertation is motivated by the observation that these strengths are complementary: Large Language Models (LLMs) can supply flexible, high-level mission specification, while Reinforcement Learning (RL)-trained skills can supply robust low-level execution. The LLM sits at the upper level, where it interprets the operator's intent: it takes a goal expressed in the high-level language of inspection and turns it into a machine-ready mission specification, broken into a sequence of staged tasks that match what the vehicle is actually able to do. Each task then falls to the lower level, where the skill must hold its behaviour steady against wind, wave, and current while serving the many goals that inspection demands rather than a single

[2] Artificial Intelligence Act, available at https://artificialintelligenceact.eu (accessed 30 June 2026).

fixed one. It is this twofold demand, robustness to disturbance together with the pursuit of many goals, that motivates treating the RL problem hierarchically.

1.2 Objectives

The objective of this dissertation is to design and validate a framework that translates free-form natural language inspection commands into robust physical execution on a USV, by coupling LLM-based mission planning with RL-based skill execution. The framework is required to interpret operator intent expressed in natural language, decompose it into a structured and auditable plan over a fixed library of reusable skills, and execute that plan reliably in unstructured marine environments, while running entirely within the onboard compute constraints of the vehicle.

To achieve this aim, the dissertation pursues the following specific objectives:

- To design a hierarchical mission execution architecture that separates semantic reasoning from physical execution across four abstraction layers, namely Mission, Task, Skill, and Action, structured by a mission ontology that constrains the content of a well-formed operator command and preserves an operator review gate over the generated plan before any actuation begins.
- To develop a hierarchical RL skill that composes a low-level navigation primitive into an object-relative boundary survey behaviour, inferring the geometry of an a priori unknown target entirely from onboard LiDAR observations and sustaining a stable orbit at a commanded standoff under environmental disturbance.
- To realise the upper layers of the architecture through two fine-tuned language models, deployed entirely onboard, that decompose a free-form natural language command into an ordered sequence of parametrised skills drawn from a fixed library, and to benchmark candidate models in their base and fine-tuned configurations so as to select, for each layer, the configuration offering the best balance of decomposition accuracy, schema reliability, and inference latency within the onboard compute budget.
- To validate the integrated framework end-to-end, first in physics-based simulation and subsequently through field deployment on the Nautilus USV in an unstructured marine environment, assessing the fidelity of the semantic-to-physical translation and the robustness of execution under real wind, wave, and current disturbances.

Through these objectives, the dissertation seeks to demonstrate that the combination of LLM-based mission specification and RL-based skill execution constitutes a viable approach to adaptive yet trustworthy mission planning and execution for USV inspection, particularly in settings where natural-language specification, modular skill execution, robustness to disturbance, and onboard autonomy are essential design requirements.

1.3 Contributions

A substantial portion of the work presented in this dissertation, in particular the hierarchical agentic planning architecture and its experimental validation in both simulation and field trials, has been consolidated into a journal manuscript prepared for peer review. At the time of writing, the manuscript has been submitted to *Engineering Applications of Artificial Intelligence* (EAAI)[3], an Elsevier journal whose scope on applied artificial intelligence for engineering systems aligns closely with the contributions described above. The dissertation and the manuscript share their core technical content, while the former provides the fuller treatment of methodology, derivations, and supporting results that the article format necessarily condenses. The resulting article is identified below:

- B. Santos, M. I. Pereira, D. F. Campos, P. N. Leite and A. M. Pinto, "Agentic Mission Planning in Unstructured Environments: A Learning-based Hierarchical Approach for Autonomous Surface Vehicles", *Engineering Applications of Artificial Intelligence* (EAAI).

1.4 Dissertation Structure

The remainder of this dissertation is organised as follows.

Chapter 2 reviews the state of the art and its limitations, covering RL and its hierarchical and skill-based extensions, classical and adaptive task and mission planning, and the use of language models for task specification. The chapter closes with a critical review that identifies the gap addressed by this work.

Chapter 3 presents the proposed methodology in two parts. The first develops the hierarchical RL skill for target survey, including its Semi-Markov Decision Process (MDP) formulation, the reward design, the high- and low-level policies, the simulation environment, and the transfer strategy. The second describes the Agentic Mission Planner, detailing its layered architecture, the Mission and Task Agents and their fine-tuning, and the skill and action layers responsible for execution.

Chapter 4 reports the experimental evaluation. It first characterises the hierarchical skill in simulation, then benchmarks the candidate language models, analyses the robustness of the Mission Agent, and finally validates the complete pipeline in end-to-end simulation and in field trials on the Nautilus USV.

Chapter 5 summarises the principal findings, discusses the limitations of the proposed framework, and outlines directions for future work.

[3] https://www.sciencedirect.com/journal/engineering-applications-of-artificial-intelligence (accessed 30 June 2026).

Chapter 2

Literature Review

2.1 Reinforcement Learning

RL offers a formulation for sequential decision-making in uncertain environments, where an agent learns a policy by interacting with the environment to maximise the cumulative reward. In robotics, RL has been applied to a broad set of problems, such as control, navigation, and task execution, where the dynamics of the system could be complex, partially known, or difficult to model explicitly. Beyond low-level control, RL can also be viewed as a candidate framework for high-level decision-making, where policies reason over long-horizon outcomes, abstract actions, and delayed rewards associated with broader strategic objectives.

This section discusses the basic formulation of RL and the methodological issues that are most relevant to long-horizon decision-making and policy learning for physical systems. The discussion in this section establishes the foundations required for Hierarchical Reinforcement Learning (HRL) and mission execution, where temporal abstraction and skill-based decision making are introduced to address these challenges.

2.1.1 Markov Decision Processes and Long-Horizon Decision-Making

RL problems are typically modeled as a MDP. These are characterised by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition dynamics, and $\gamma \in [0, 1)$ the discount factor [56, 66]. The MDP formulation also assumes the Markov property, according to which the future behaviour of the system depends only on the current state and action. The objective of an RL agent is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximises the expected discounted cumulative reward, also referred to as the return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.1}$$

where r_{t+k+1} denotes the reward received k steps after time t. The expected return under policy π from state s is captured by the state-value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t \mid s_t = s\right], \tag{2.2}$$

and the action-value function, which conditions additionally on the action taken:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t \mid s_t = s, a_t = a\right]. \tag{2.3}$$

These functions satisfy the Bellman equations, which express a recursive relationship between the value of a state and the values of its successors [56, 66]:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} \mathcal{T}(s' \mid s,a)\left[\mathcal{R}(s,a) + \gamma V^{\pi}(s')\right]. \tag{2.4}$$

This modelling abstraction is the foundation of most of the theoretical work in RL and provides a common platform for algorithm design and analysis, as formalised in the existing literature on stochastic decision processes and RL [56, 66]. However, this formulation implicitly assumes that decisions are made at a fixed temporal resolution and that actions are abstracted as being completed within a single decision step, which limits its expressiveness for mission-level decision-making.

In robotic systems, MDPs are used to model the interaction between a robotic agent and its environment over time, where states are typically represented as encodings of information about them, while actions are represented as control inputs or higher-level commands [66]. While this abstraction is suitable for short-horizon or episodic control problems, it becomes less appropriate for long-horizon mission execution, where decisions span extended time scales and involve sequencing multiple behaviours. Although the MDP model provides an expressive and widely applicable formulation, its application to robotics often requires approximations due to partial observability, sensor noise, and unmodeled dynamics [31].

In long-horizon decision-making, reward signals can be sparse or delayed, making credit assignment difficult and increasing the variance of learning updates [66]. These delayed rewards are closely tied to decisions about skill sequencing, switching, and termination, where the outcome of a high-level choice may only be observable after the execution of an entire behaviour [2]. The choice of discount factor, which determines how future rewards are weighted relative to immediate rewards, directly affects the planning horizon, but large discount factors can also contribute to instability and high variance in learning algorithms, particularly in conjunction with function approximation [66]. Standard MDPs are not well suited to capture such temporally extended actions, motivating extensions such as Semi-MDPs, in which actions may have variable durations and delayed effects [66]. These issues also point to a fundamental trade-off in RL for robotics: while the MDP model provides a clean mathematical formulation, learning effective policies for long-horizon tasks remains challenging in practice.

2.1.2 Model-Based and Model-Free Reinforcement Learning

Model-based RL involves techniques that assume or learn a model of the system dynamics and reward function that can be used for planning, prediction, or policy optimisation, drawing on concepts from optimal control and stochastic planning [50, 66]. This explicit modelling capability enables anticipating long-term consequences of action sequences, which can be advantageous in structured environments where accurate models are available [50]. However, model-based approaches are sensitive to model inaccuracies, and errors in the learned or assumed dynamics can compound during planning, particularly over extended time horizons [34].

Model-free RL, on the other hand, involves techniques that learn policies or value functions directly from experience data, without building an explicit model of the environment [66]. Examples of model-free RL include Q-learning [76], policy gradient algorithms [67], and actor-critic methods [36]. Model-free RL has been successful in high-dimensional and continuous control tasks, especially when combined with deep function approximation [41, 49], but these methods are often associated with high sample complexity and reward sensitivity [22].

In field robotics in particular, unmodeled dynamics, environmental disturbances, and partial observability can cause planning errors to compound over long-duration missions, limiting the reliability of purely model-based approaches [34]. Model-free RL methods avoid explicit modelling errors but are often associated with high data requirements, which can be expensive and dangerous to obtain in the context of physical systems [34]. This contrast highlights a fundamental trade-off between data efficiency and robustness [50]. Hybrid approaches that combine elements of both model-based and model-free RL have also been explored, for example, by using learned models for short-horizon rollouts and model-free updates for robustness [29]. Regardless, the model-based and model-free distinction provides a useful framework for understanding the trade-offs between different RL approaches in robotics decision-making problems [66].

2.1.3 Value-Based and Policy-Based Methods

RL algorithms can be broadly categorised according to what is learned during training: value-based methods learn an estimate of the expected cumulative reward associated with states or state-action pairs, while policy-based methods directly learn a mapping from states to actions [66].

Value-based methods rely on the estimation of a value function, typically the action-value function $Q^\pi(s,a)$, which represents the expected return of taking action a in state s and following a given policy thereafter [66]. The policy is then derived implicitly by selecting actions that maximise this estimate, $\pi(s) = \arg\max_a Q^\pi(s,a)$. The Q-learning update rule provides a model-free method for estimating Q^π from experience:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t) \right], \tag{2.5}$$

where α is the learning rate [66]. Deep Q-Network (DQN) [49] is a prominent example of a value-based method, combining Q-learning with deep neural function approximation to achieve

strong performance in high-dimensional discrete action spaces [49]. However, value-based methods are generally not well-suited to continuous action spaces, as maximising over actions becomes intractable when the action space is continuous.

Policy-based methods, on the other hand, directly parameterise and optimise the policy $\pi_\theta(a \mid s)$, typically by computing gradients of an objective function with respect to the policy parameters θ. The policy gradient theorem [66] establishes that the gradient of the expected return $J(\theta) = \mathbb{E}_{\pi_\theta}[G_t]$ with respect to θ can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t\right], \tag{2.6}$$

which enables gradient ascent on the expected return without requiring a model of the environment. This makes policy-based methods naturally applicable to continuous action spaces. Actor-critic methods combine both paradigms, using a learned value function (the critic) to reduce the variance of policy gradient estimates while retaining the flexibility of direct policy optimisation [66]. In practice, the return G_t in Equation 2.6 is replaced by an advantage estimate $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, which reduces variance by subtracting a state-dependent baseline. Proximal Policy Optimisation (PPO) is a widely used on-policy actor-critic algorithm that stabilises training by constraining the magnitude of policy updates via a clipped surrogate objective [59]:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{2.7}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the updated and old policies, $\hat{A}_t$ is the estimated advantage, and ϵ is a clipping hyperparameter that bounds the policy update.

2.1.4 On-Policy and Off-Policy Learning

A further distinction between RL algorithms concerns whether learning is performed on-policy or off-policy [66]. This distinction has practical implications for sample efficiency, stability, and applicability in robotic systems.

On-policy methods update the policy using data collected under the current policy, meaning that experience cannot be reused across updates [66]. This generally leads to higher sample complexity, as data must be continuously collected during training. However, on-policy methods tend to be more stable and easier to tune, as the learning target remains consistent with the data distribution [60]. PPO [59] is a representative on-policy algorithm, and its stability properties have contributed to its widespread adoption in robotic RL [57].

Off-policy methods, on the other hand, can learn from data collected under a different policy, typically by storing past experience in a replay buffer and reusing it across multiple updates [49, 66]. This improves sample efficiency significantly, which is particularly relevant in robotics where data collection is costly. However, off-policy learning can introduce instability, as the learned

value function may be updated using data from a distribution that differs from the current policy [22, 41]. Soft Actor-Critic (SAC) [23] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [18] are prominent off-policy algorithms that incorporate stabilisation mechanisms, such as target networks and clipped double Q-learning, to mitigate these issues [18, 23].

The choice between on-policy and off-policy learning involves a trade-off between stability and sample efficiency that is especially relevant in the context of robotic systems, where data collection is constrained and training is often conducted in simulation prior to deployment [34].

2.1.5 Sim2Real Transfer

Because of the limitations imposed by real-world experimentation, RL in robotics is often carried out in simulated environments [34]. These allow for the collection of data on a large scale, systematic evaluation, and exploration of behaviours that are difficult or dangerous to test on real-world robots. Moreover, improvements in physics engines and simulation software have further improved the realism and effectiveness of simulation-based learning for robotic RL [44, 73], making simulation a practical and necessary tool for evaluating long-duration Autonomous Surface Vehicle (ASV) missions where real-world trials are costly, time-consuming, and operationally constrained [45].

However, policies trained in simulation have been found to have lower performance when evaluated on real-world robotic systems, a problem that has been generally described as the sim-to-real gap [39], primarily due to differences in simulated and real-world dynamics, sensor models, actuation delays, and environmental conditions. These small differences in modelling can compound over time, and repeated execution errors may accumulate across mission stages and ultimately lead to mission-level failure [40]. Several approaches have been developed to address the sim-to-real transfer problem, often in conjunction with execution monitoring and recovery mechanisms that must operate under model inaccuracies themselves, further compounding sim-to-real effects at the mission level. Domain randomisation adds stochasticity to simulation parameters to promote robustness [72], while system identification aims to make simulation models more accurate in modelling real-world behaviour [69]. Additionally, fine-tuning policies with small amounts of real-world data is also a popular approach, although it also introduces some of the costs of real-world experimentation [58].

At the mission level, RL policies that reason over abstract states and skill outcomes are generally less sensitive to low-level simulation inaccuracies than policies trained directly on continuous control, as they rely on aggregated execution outcomes rather than precise dynamic fidelity [68]. Nevertheless, the methodological considerations discussed in this section, spanning problem formalisation, algorithm selection, and on- and off-policy training, reflect the broader challenges of applying flat RL to long-horizon tasks, where decisions must be made across multiple time scales and behaviours must be sequenced adaptively. These limitations motivate hierarchical and skill-based extensions of RL, which are discussed in the following section.

2.2 Hierarchical and Skill-Based Reinforcement Learning

HRL is an extension of RL that incorporates temporal and structural abstractions to facilitate scalable decision-making in complex, long-horizon tasks. Through the use of multiple levels of control in decomposing behaviour, HRL is able to tackle some of the most important challenges that arise in flat RL, including the use of sparse rewards, inefficient exploration, and lack of transferability, which are especially common in robotic tasks.

This section will discuss the main differences of HRL compared to flat RL and how these are related to temporal abstraction.

2.2.1 HRL and Temporal Abstraction

The key concept that drives HRL is the application of temporal abstraction, where decisions are made on multiple time scales. Rather than choosing primitive commands at each time step, an agent can have temporally extended behaviours that represent sequences of basic actions, which shortens the planning horizon in higher-level decision-making and helps to overcome reward assignment problems in long-horizon tasks that unfold over mission-level time scales.

One of the most popular formalisations of temporal abstraction is the options framework [68], where higher-level actions are represented as temporally extended behaviours. Formally, an option $\omega \in \Omega$ is defined as a triple $\omega = (\mathcal{I}_\omega, \pi_\omega, \beta_\omega)$, where $\mathcal{I}_\omega \subseteq \mathcal{S}$ is the initiation set of states from which the option can be invoked, $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0,1]$ is the intra-option policy that governs execution, and $\beta_\omega : \mathcal{S} \to [0,1]$ is the termination condition that determines the probability of the option terminating in each state [68]. Using this framework, decision-making problems can be formulated as Semi-MDPs, which allow hierarchical policies to be learned using variants of standard RL algorithms while naturally supporting execution monitoring and decision-making at skill boundaries during mission execution. The high-level policy $\pi_\Omega(s)$ selects over the set of available options Ω rather than over primitive actions, operating at a coarser temporal resolution and receiving rewards accumulated over the duration of each option. Many subsequent HRL methods can be viewed as instantiations or extensions of this framework, such as value function decomposition algorithms [11] and manager-worker architectures [75].

More recent efforts have concentrated on combining hierarchical representations with deep RLs, allowing function approximation to be applied to both high-level and low-level policies [52]. These methods have shown improved scalability and performance in problems with sparse rewards or long delays, however, HRL adds new complexity to the design process, such as choosing useful abstractions, handling interactions between levels, and stabilising the learning process.

2.2.2 Skill Representation and Acquisition

One of the most important considerations in HRL is skill representation and acquisition, which are the basic units of higher-level decision-making. Skills are generally formalised as reusable

policies that solve intermediate tasks or display stable behavioural patterns [68]. There are two broad approaches to acquiring skills: skill libraries and autonomous skill discovery.

Predefined skills are often used in robotics, where low-level controllers for tasks such as navigation, obstacle avoidance, or manipulation can be designed using domain knowledge and traditional control techniques [27]. These skills contrast with learned skills in that their structure, operating conditions, and failure modes are explicitly specified rather than inferred from data. This is particularly useful for skills that require interpretability and safety, which are important in real-world applications. In safety-critical robotic systems, predefined skill libraries enable offline validation, bounded execution behaviour, and controlled interaction with the environment, making them more suitable for deployment under strict operational constraints [19]. Their explicit structure also facilitates inspection, verification, and reasoning about mission execution at the skill level. While predefined skills can provide these benefits, they may not generalise well outside their original context or be easily adapted to new situations.

Unlike predefined skills, which encode explicit task semantics and are validated prior to deployment, autonomously discovered skills are derived entirely from interaction data, without access to task-level annotations or behavioural specifications. These methods aim to uncover latent behavioural structure rather than encode task semantics explicitly. Some approaches to skill discovery use information-theoretic learning, diversity maximisation, or latent variable models to promote a set of behaviours that cover the state space [12, 21, 62]. However, such generic objectives can lead to skills that are difficult to interpret or align with mission-level requirements. While skill discovery can create improved flexibility and avoid the need for manual skill design, it is not clear whether discovered skills are relevant to the overall task.

In many HRL systems, a hybrid approach is taken, where engineered low-level controllers are combined with learned high-level policies or skill discovery is guided by inductive biases from the application domain [15].

2.2.3 Skill Transfer and Generalisation

In HRL, the decision-making process is usually divided into at least two levels: a high-level policy that decides on skills or sub-tasks, and low-level policies that control the environment based on the decisions made by the high-level policy. The high-level policy makes decisions at a lower resolution in time and typically uses abstract representations of state, while the low-level policies are involved in continuous control [37].

Modularity and the ability to reuse skills are facilitated by this division of labor, as skills can be used across different mission specifications without having to be relearned [1]. In this context, skill reuse is primarily relevant as a mechanism for mission-level generalisation, allowing a fixed set of skills to support variations in task objectives, ordering, or constraints. Reuse of skills has been demonstrated to improve the efficiency of learning, speed up training, and improve generalisation, especially when multiple tasks are learned simultaneously [11, 17]. This reuse directly contributes to improved sample efficiency at the mission level, since learning is concentrated on sequencing and selection rather than on execution. This transfer can be achieved by reusing learned

controllers, sharing embeddings of skills, or adapting high-level policies to new task descriptions while keeping the skill set constant [24].

However, the robustness of the learned skills, as well as the relationship between tasks, are critical to the success of skill reuse. In particular, reuse is limited when skills are strongly context-sensitive, as variations in environment dynamics, operating conditions, or mission assumptions may violate the conditions under which a skill was learned or validated. In HRL, there are also often coordination issues that need to be addressed, such as ensuring that the decisions made by the high-level policy are consistent with the low-level constraints and that learning signals are correctly propagated between levels [52].

2.2.4 HRL in Mobile and Field Robotics

HRL has been studied in mobile and field robotics, where the presence of long-term autonomy, partial observability, and environmental uncertainties makes hierarchical control more attractive than flat control techniques [2, 54]. In practical scenarios, these issues are exemplified by field-specific constraints such as sensor noise, actuation delays, communication dropouts, and hardware failures, which can compound over the course of long missions. In mobile and field robotics, HRL has been used in navigation and exploration tasks, where high-level decision-making involves the choice of waypoints, regions, or behaviours, and low-level control involves the implementation of motion and obstacle avoidance behaviours [13, 38]. In mobile robotics, HRL has also been used to enable robots to dynamically switch between sensing, motion, and interaction modes within a mission. These hierarchical control strategies have enabled robots to perform long-term missions and adjust their behaviour according to changing environments [26].

Applications of field robotics, including aerial, ground, and underwater robots, further illustrate the advantages of HRL. In these applications, robots must operate in environments where communication is limited, energy is a concern, and environmental conditions are dynamic, as is common in offshore, underwater, and infrastructure inspection missions. HRL has been used in structure mission execution in these applications, where robots are able to plan behaviour sequences such as transit, inspection, and data acquisition while ensuring robustness at the control level [79]. However, there are often discrepancies between simulation-based analysis and real-world deployment, especially regarding failure handling, timing uncertainty, and performance degradation over the course of long missions.

2.3 Task and Mission Planning in Robotics

Planning of tasks and missions in robotics deals with the generation, organization, and execution of long-term action sequences that enable robots to accomplish high-level tasks. In mobile robotics, the mission planning endeavour involves connecting symbolic task representations with continuous control, in environments with uncertainty, partial observability, and changing conditions.

This section provides an overview of classical, hybrid and adaptive planning methods and their shortcomings.

2.3.1 Classical Task and Mission Planning

Traditional task and mission planning methods come from the field of symbolic Artificial Intelligence (AI), where planning is modeled as a search problem in discrete state and action spaces. Actions are usually represented by logical preconditions and effects, and planning is done offline by searching the action space using domain models expressed in languages such as STRIPS [14] or PDDL [16]. These methods offer strong guarantees regarding correctness and support the explicit representation of domain knowledge.

The major problem with traditional planners is that they are based on several assumptions that are hard to meet in real-world robotic missions such as deterministic action effects, complete knowledge of the environment, and constant task requirements [53]. However, in outdoor or maritime environments with high levels of uncertainty and dynamics, these assumptions are commonly violated due to noise, disturbances, and unexpected events. This means that offline plans can become stale during execution, which requires expensive replanning or human intervention.

This has limited the use of purely symbolic planning methods for long-duration autonomous robotic missions and led to the need for integrating planning with feedback and adaptation at the execution level [28].

2.3.2 Hybrid and Adaptive Planning

To bridge the planning and control gap, hybrid planning architectures have been increasingly used in robotics. Hybrid planning architectures integrate symbolic or task-level planners with frameworks that deal with continuous control, perception, and reaction [32]. Execution-level models like finite state machines and behaviour trees are typically used to represent mission logic, deal with contingencies, and provide responsiveness during execution [46]. Behaviour trees, in particular, have gained increasing popularity because of their modularity, hierarchical representation, and ability to provide fallback plans. These enable the composition of complex missions from reusable task modules while retaining interpretability and debugability [10]. However, hybrid planning architectures tend to rely on manually designed task decompositions and transition graphs. As the complexity of the mission grows, these become increasingly hard to scale and adapt, particularly when the outcomes of tasks are stochastic or context-dependent. This has sparked interest in the use of learning-based components to generalize over scenarios and alleviate the need for manual design.

Adaptive mission planning is a response to the need for robots to dynamically change task sequences based on changes in the environment or task outcome [33]. Instead of planning and executing tasks in a fixed sequence, adaptive methods combine planning and execution, monitoring

system state and replanning or switching tasks as needed. Task decomposition is a key component of this process, as decomposing abstract missions into more concrete tasks allows dynamic reordering.

2.4 Language Models for Task Specification

Planning for tasks and missions in robotics relates to the challenge of expressing high-level goals as action sequences that can be carried out by autonomous robots despite physical, temporal, and safety constraints. The classical solution to this problem uses symbolic planning paradigms, in which tasks are formally expressed in programming languages and solved by search or logic-based reasoning. Although these paradigms provide interpretability and correctness proofs, they demand accurate domain knowledge and assume that tasks are formally expressed in machine-readable formats. However, with the growing need for robots to function in unstructured environments and to interact with lay users, these assumptions have become too limiting, and there is a need to resort to natural language and machine learning approaches for task specification and planning.

2.4.1 Natural Language Task Specification

Specification of natural language tasks seeks to empower users to express their intentions and constraints to robots through free-form linguistic descriptions instead of planning languages. Initial research in this area was concerned with following constrained instructions, where natural language ones were translated to symbolic forms using semantic parsing and grammars [47, 71]. These studies showed that language could be a practical way to interact with robotic planners, but they were constrained by their limited vocabularies, rigid parsing pipelines, and strong assumptions about the structure of language and the world.

More recent work has focused on grounding language in perceptual states, object affordances, and action portrayal, highlighting a representational mismatch between abstract linguistic descriptions and the continuous, partially observable robot state. In these studies, language is viewed as an abstract depiction of intention that needs to be linked to the robot's internal understanding of the world. This grounding problem often involves translating language into symbolic goals, predicates, or task graphs that can be reasoned with by classical planners. Alternatively, language can be translated directly into parameterised skills or behaviours that can be executed hierarchically without symbolic planning [30, 48]. These two views of language translation (language-to-symbolic and language-to-skill translation) represent different trade-offs between expressiveness, interpretability, and execution robustness, but they are often handled together in a single instruction understanding pipeline [70].

2.4.2 LLMs for High-Level Task Planning

The development of LLMs has significantly broadened the applicability of language-based task planning in robotics. Because of their ability to perform reasoning on long contexts and their

capability to produce structured outputs, LLMs have been used as high-level planning components that have the ability to break down tasks into subgoals, produce action sequences or intermediate representations such as pseudo-code or symbolic plans [25, 42, 64].

One of the typical architectural frameworks involves the use of LLMs as front-end interfaces for existing planning or control systems. In such frameworks, the language model interprets the user's intent and provides goals, constraints, or task representations that are then processed by symbolic planners, behaviour trees, or skill-based controllers [61]. In such a manner, LLMs can improve the flexibility and usability of the human-robot interface while maintaining the reliability and structure of the downstream planning components. However, because LLMs do not have explicit models of robot dynamics and environment constraints, their outputs are often required to be validated, refined, or corrected before execution [5]. This has motivated hierarchical architectures in which the language model is decoupled from low-level control, with execution delegated to a separate, trusted subsystem. A representative instantiation of this pattern in the maritime domain is the AI Captain system proposed by Christensen *et al.* [9], which combines a high-level LLM planner with a classical guidance, navigation, and control module for low-level vehicle execution on an autonomous surface vehicle. The LLM is responsible for interpreting operator intent and generating or revising mission plans through a conversational interface, while the module handles continuous control independently of the language model. Human approval is retained as an execution safeguard, positioning the operator in a supervisory role throughout the mission. Although the system was validated on a real ASV platform, the demonstrated missions remain relatively short in horizon, and operator intent is supplied through prompts that enumerate the required tasks explicitly and in execution order, so cases in which intent is more abstract or unordered, which LLMs can in principle decompose into ordered, actionable steps [25], are left unexercised.

2.4.3 Integration of LLMs with RL for Task Execution

A paradigm that is emerging involves the integration of LLMs with learning-based control, such as RL, in a way that assigns them different roles in hierarchical or modular architectures. In this paradigm, LLMs are used at the task or mission level for specification, decomposition, and sequencing, while learned policies are tasked with low-level execution and adaptation [64]. This approach combines the abstraction and reasoning strengths of language models with the robustness of interaction-trained policies in dynamic settings [5].

As mentioned before, RL has been extensively used in robotics, particularly in navigation and manipulation tasks, where policies need to deal with disturbances, partial observability, and model errors [34, 63]. When integrated with LLM-based task specification, formal task or mission descriptions can inform the choice and sequencing of pre-defined skills, which can be supported by learned value functions or RL-trained policies. This allows flexible task execution without requiring end-to-end language-to-action learning or replacing existing low-level controllers [5].

More recent research uses language outputs as high-level goals, skill selectors, or context inputs to learned control policies, such as RL [65]. Other methods utilise RL to ground and refine language plans through trial-and-error interaction, enabling language-based planners to be

corrected or refined through experience [7]. Although these methods demonstrate promise for long-horizon autonomy, there are still challenges in aligning representations between language models and control policies, achieving sample efficiency, and maintaining safety during learning and execution [6, 78]. Consequently, the integration of LLMs and RL is an active and potentially viable area of research for task and mission planning, especially when language is utilised to define high-level structure and execution is left to trusted controllers or skill libraries. These factors inform architectures, in which decision-making occurs at the mission level given explicit mission specifications and execution is based on a pre-defined set of reusable skills.

2.5 Critical Review

The preceding sections have examined the principal bodies of work relevant to mission-level autonomy for ASV inspection, spanning RL, hierarchical and skill-based control, classical and adaptive mission planning, and language-based task specification. Considered together, these literatures expose a recurring gap: no single paradigm simultaneously provides flexible high-level mission specification and execution that remains robust over long horizons without recourse to continual replanning or human intervention. This section summarises the main limitations identified across the reviewed approaches.

Flat RL provides a principled formulation for sequential decision-making, but its direct application to long-horizon missions is limited. The standard MDP abstraction assumes that decisions are taken at a fixed temporal resolution and that actions are completed instantly [66], which is poorly matched to robotics applications, where behaviours span extended and variable durations, and whose effect is frequently only noticeable several steps afterwards. Long-horizon settings further introduce sparse and delayed rewards, complicating credit assignment and increasing the variance of learning updates [66]. In physical systems, these difficulties are compounded by the sim-to-real gap, where small modelling discrepancies accumulate across mission stages and can ultimately lead to mission-level failure [19, 39]. Flat RL therefore struggles to support decision-making that must reason over abstract actions and delayed outcomes, which motivates the use of temporal abstraction.

HRL addresses several of these issues by decomposing behaviour across multiple time scales, shortening the effective planning horizon and improving exploration and transfer in sparse-reward tasks [54, 68]. However, hierarchical control introduces difficulties of its own. Predefined skill libraries offer interpretability, offline validation, and bounded execution behaviour, which are desirable in safety-critical maritime operations, yet skills constructed through traditional, hand-engineered methods tend to generalise poorly outside of their original context. Autonomously learned skills increase flexibility and robustness to disturbances through interaction, but their internal behaviour is harder to interpret and may not align cleanly with mission-level requirements [12]. Across both, reuse is constrained when skills are strongly context-sensitive, and coordination between levels remains a persistent design challenge [37]. A further open question concerns how the high-level objective is specified: HRL provides mechanisms for composing skills but needs the

mission goal to be given, leaving the translation from operator intent to a usable mission specification unresolved.

Classical task and mission planning offers correctness guarantees and the explicit representation of domain knowledge, but rests on assumptions of deterministic action effects, complete environmental knowledge, and stable task requirements [53]. These assumptions are frequently violated in outdoor and maritime settings, where noise, disturbances, and unexpected events can gradually erode the validity of offline plans as execution conditions diverge from those assumed at planning time [28]. Hybrid and adaptive architectures improve responsiveness by merging symbolic planners with execution-level representations such as behaviour trees [10, 46], yet they generally depend on manually designed task decompositions and transition structures that become difficult to scale and adapt as missions grow in complexity or as task outcomes become stochastic. These limitations indicate the need for a more flexible mechanism for specifying missions, and for an execution layer robust enough that an offline plan need not be continually revised.

LLMs offer a promising route to such flexibility, interpreting operator intent expressed in natural language and producing structured subgoals, action sequences, or intermediate plan representations [25, 64]. However, because LLMs lack explicit models of robot dynamics and environmental constraints, their outputs require validation, refinement, or correction before execution [25]. The AI Captain system [9] illustrates the prevailing response to this limitation, pairing a high-level LLM planner with a classical guidance, navigation, and control module and retaining human approval as an execution safeguard. While effective, the reliance on a classical control stack restricts adaptability to conditions that fall outside its operational assumptions, leaving the system dependent on operator revision rather than on an execution layer capable of absorbing disturbances on its own. The demonstrated missions also remain relatively short in horizon, with operator intent supplied through prompts that enumerate the required tasks explicitly and in execution order, leaving more abstract or unordered specifications largely unexamined. Recent work integrates LLMs with learned control, assigning specification and sequencing to the language model and low-level execution to RL-trained policies [5, 64, 65]. This division of responsibilities is well motivated, but open challenges remain in aligning representations between language models and control policies, achieving sample efficiency, maintaining safety during learning and execution, and sustaining performance over long mission horizons [78].

Chapter 3

LLM-Based Mission Planning with RL Skill Composition

This dissertation addresses the challenge of achieving mission-level autonomy for ASVs conducting inspections of maritime infrastructure. In operational inspection scenarios, such as the periodic survey of offshore platforms or harbour infrastructure, a human operator must be able to express what the vehicle is to accomplish without manually composing low-level sequences. At the same time, the vehicle must execute the resulting plan robustly across extended time horizons and under the disturbances characteristic of unstructured marine environments. These two demands, flexible high-level specification on the one hand and robust long-horizon execution on the other, are difficult to tackle together, and the survey of existing work showed that no single paradigm satisfies both simultaneously.

The framework proposed in this dissertation responds to this gap by coupling LLM-based mission planning with RL-based skill execution: LLMs provide a flexible, high-level interface for specifying missions, while RL provides execution that remains robust under disturbance without continual replanning or human intervention. Semantic reasoning is performed at planning time, where a free-form operator command is progressively refined into a structured, parametrised plan that is presented for explicit operator approval before any actuation begins; physical execution is then delegated to a fixed library of validated, RL-trained skills operating in a continuous, high-frequency closed loop.

The approaches surveyed in Chapter 2 reveal limitations in existing autonomous mission execution systems, motivating the two contributions developed in this dissertation. The first is the Agentic Mission Planner: a hierarchical, two-agent planner that accepts free-form natural language commands and decomposes them, through a Mission Agent and a subsequent Task Agent, into ordered sequences of parametrised skills drawn from a predefined library. This component targets the specification side of the problem, providing a flexible operator interface together with an operator review gate that preserves a safety guarantee over the generated plans. The second contribution is a hierarchical skill for target survey, in which a high-level policy reasons over a Light Detection and Ranging (LiDAR)-centric observation and composes a low-level navigation

primitive to orbit an *a priori* unknown object at a commanded standoff distance. This component focuses on behaviour execution, demonstrating that a complex, object-relative inspection behaviour can be acquired through interaction and sustained robustly under environmental disturbances.

The survey skill is built around the same parametrised interface and termination logic as the rest of the skill library; the Task Agent therefore selects it exactly as it would any other skill, and the resulting standalone capability can substitute for any skill that fulfils the same operational role. The remainder of this chapter describes the proposed concept in turn, beginning with the overall architecture, proceeding through the Mission and Task Agents, and concluding with the formulation of the skill library itself.

3.1 Hierarchical Abstraction Layering

The Agentic Mission Planner is structured as a hierarchical, multi-stage translation engine that progressively refines a free-form natural language command issued by a human operator into actuation commands for the vehicle. This progressive refinement is partitioned across four specialised abstraction layers: Mission, Task, Skill, and Action. Rather than treating mission execution as a single end-to-end mapping from language to control, the architecture decomposes this translation into a sequence of well-defined sub-problems, each handled by a model suited to its abstraction level and operational requirements. This architecture is illustrated in Figure 3.1.

Within this hierarchy, the four layers are defined as follows. A *Mission* represents the high-level semantic objective provided by the operator. A *Task* is a discrete, logically structured assignment that constitutes one component of the mission objective. A *Skill* is a parametrised, modular autonomous capability drawn from a predefined library, encompassing both physical navigation behaviours and perceptual operations. An *Action* represents the most granular tier of the hierarchy: for navigation skills, actions consist of continuous, high-frequency hardware signals such as linear and angular velocity commands, whereas for perceptual skills, they involve discrete sensor triggers or data processing steps.

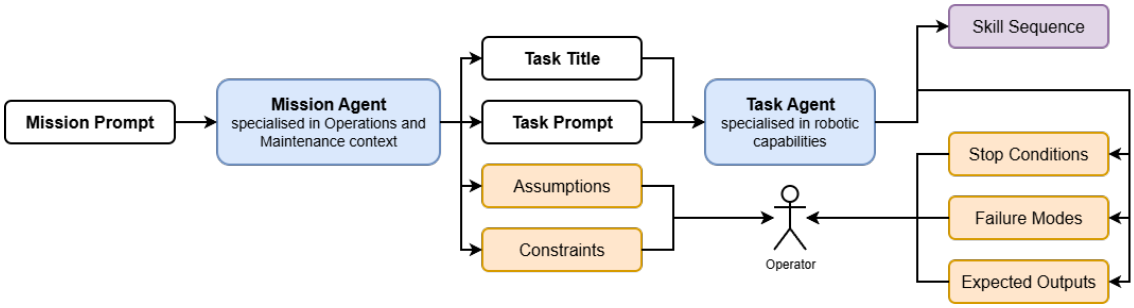


Figure 3.1: The proposed architecture. The operator's natural language prompt is decomposed by the Mission Agent into structured tasks, which the Task Agent then expands into the final skill sequences, subject to operator review.

The translation across these layers proceeds top-down. At the upper levels, two specialised language models perform semantic decomposition at distinct stages of abstraction. The first, referred to as the Mission Agent, receives the operator's natural language command as its sole input and produces a structured, ordered set of high-level tasks that collectively fulfil the stated objective, formalised as:

$$\mathcal{M} \longrightarrow \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}. \tag{3.1}$$

The second model, referred to as the Task Agent, receives each individual task produced by the Mission Agent and translates it into a concrete sequence of sets of parametrised skills drawn from the library:

$$\mathcal{T}_i \longrightarrow \{S_1, S_2, \ldots, S_k\}. \tag{3.2}$$

At the lower levels, the Skill layer triggers the appropriate execution modules, with navigation skills governed by RL policies trained offline in physics-based simulation, and the Action layer passing the resulting continuous control signals to the vehicle's hardware interface.

A key design principle of the architecture is the separation of semantic reasoning from physical execution. The LLM-based upper layers operate at planning time, establishing the complete skill sequence before any actuation begins, after which the lower layers operate in a continuous, high-frequency closed loop. Because this plan is fixed and operator-approved before execution, an erroneous decomposition cannot reach the actuators, and every executed manoeuvre remains traceable to a specific task in the approved plan. The separation further gives each layer a well-defined interface, so that a language model, perception module, or navigation policy can be replaced without modifying the others. These same properties also align the architecture with the governance expectations emerging for safety-critical AI systems: the mandatory approval gate instantiates the meaningful human oversight required of high-risk systems under the EU AI Act[1] (Art. 14), while the task-to-manoeuvre traceability supports the record-keeping and auditability that the same regime demands (Art. 12), positioning the architecture favourably with respect to forthcoming regulatory requirements.

The Mission and Task Agents are implemented as specialised open-source language models deployed entirely onboard the vehicle, ensuring network independence and eliminating the connectivity dependencies identified in prior autonomous maritime systems [9]. Once the agents have processed the operator command, the resulting plan, comprising the full task and skill decomposition with all associated parameters, is presented to the operator via a custom mission control interface for review and explicit approval before any execution is initiated. During execution, the interface provides continuous situational awareness by highlighting the currently active skill, and exposes an abort control that halts the mission at any point. The following sections detail each layer of the architecture in turn.

[1] Artificial Intelligence Act, available at https://artificialintelligenceact.eu (accessed 30 June 2026).

3.2 Mission Agent and Ontology

To guide this process performed by the Mission Agent, a mission ontology is proposed, defining the core semantic elements required to transform a natural language mission description into executable vehicle behaviour. This ontology constrains the information that must be present in any operator command and provides the structural representation on which the upper layers of the architecture operate. Specifically, a well-formed mission command must contain four key elements, as exemplified in Figure 3.2: (i) an *objective*, specifying what the vehicle must accomplish, for example, inspecting a designated structure; (ii) a *target or location*, expressed either as explicit coordinates or as a descriptive reference to a structure or feature whose position must be resolved at runtime; (iii) *sensor and data requirements*, identifying the measurement modalities that must be captured, such as video, LiDAR, thermal imagery, or bathymetry, so that the appropriate perception tasks can be selected; and (iv) *operational constraints or contextual information*, such as area boundaries, known hazards, or return-to-base instructions.

The Mission Agent constitutes the topmost layer of the architecture and serves as the primary interface between the human operator and the autonomous planning pipeline. Its sole input is the free-form natural language command provided by the operator, structured according to the four-element ontology, and its output is a structured, ordered set of high-level tasks that collectively fulfil the stated objective.

To instantiate and test the proposed ontology for maritime operations, a domain-specific task vocabulary should be constructed, defining the complete set of high-level tasks available for selection by the Mission Agent. This vocabulary organises the available tasks into four functional categories: System tasks, which perform pre-mission health checks across the vehicle's sensor suite; Positioning and Manoeuvre tasks, which govern the vehicle's spatial behaviour; Survey and Inspection tasks, which coordinate the acquisition of mission-relevant data by combining navigation primitives with active perception; and Data Management tasks, which handle post-collection

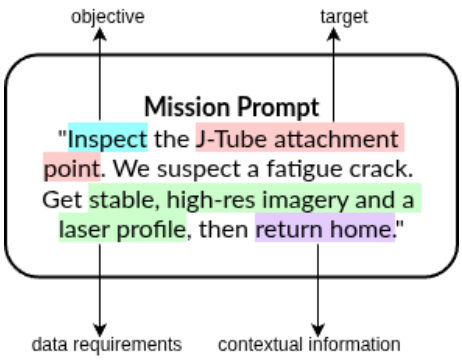


Figure 3.2: The proposed mission ontology, illustrated through an annotated example prompt. A well-formed command must express four key elements, here colour-coded within the natural language description: the objective (”Inspect”, red), the target or location (”J-Tube attachment point”, purple), the sensor and data requirements (”stable, high-res imagery and a laser profile”, green), and the operational constraints (”return home”, blue).

information processing. This categorisation supports interpretability of the Mission Agent's outputs during operator review, as the functional role of each generated task is immediately apparent from its category.

At execution time, the vocabulary provides the operator with a structured reference against which the generated plan can be reviewed, as each task title in the Mission Agent output corresponds to a named entry in the vocabulary with a well-defined functional role. The operator is not required to reference this vocabulary directly when formulating a command; the Mission Agent is responsible for inferring the appropriate tasks from the natural language prompt through the proposed ontology.

For each task it identifies, the Mission Agent's output specifies a task title drawn from the domain vocabulary, a set of task-level parameters such as target coordinates or target description strings, and a concise natural language task prompt that encapsulates the intent of that task and serves as the direct input to the Task Agent in the subsequent layer. Alongside the task sequence, the Mission Agent also produces a set of assumptions, enumerating the conditions implicitly assumed in order to generate the plan, and a set of constraints, identifying operational conditions that, if violated, could lead to incomplete or faulty mission execution. Together, these elements provide the operator with sufficient context to evaluate the soundness of the generated plan before approving execution.

3.3 Task Agent

The Task Agent is the second layer of the Agentic Mission Planner and serves as an intermediate parametrisation stage between the mission representation produced by the Mission Agent and the skill execution layer. It receives the ordered sequence of tasks generated by the Mission Agent and translates each individual task into a concrete, parametrised set of skills drawn from a predefined library, as formalised in Equation 3.2.

This translation is non-trivial: a single mission element may influence multiple skills, while a single skill may require parameters derived from different parts of the original task prompt. Implementing this mapping through hardcoded rules would require substantial manual engineering and would be difficult to maintain or generalise across different mission types. The use of an LLM-based Task Agent therefore provides a flexible mechanism for translating structured mission intent into executable skill configurations while minimising the amount of explicit task design required from the operator.

The Task Agent operates over a predefined skill library that defines the complete set of autonomous capabilities available for execution on the vehicle. The library contains 37 distinct skills, organised into three functional domains: Navigation, Perception, and Mission Management. Navigation skills compute the hydrodynamic movement of the vehicle, producing continuous actuation commands. Perception skills manage sensor data processing and environmental awareness, producing discrete outputs such as object detections or depth measurements. Mission Management

skills handle data logging and state management operations that support mission execution without directly commanding the vehicle.

With a library of this size, each task requires the Task Agent to select one of several navigation skills, choose any compatible subset of filter and perception skills to run alongside it, and assign parameters to each, rather than choosing from a short fixed list of commands. The size and structure of the skill library therefore determine the expressiveness of the planning pipeline and the range of missions the architecture can support.

A key design decision in the Task Agent is the compositional structure of its output, which reflects the concurrent nature of real maritime task execution. Rather than producing a flat sequence of skills to be executed strictly in series, each step in the Task Agent output is organised around three simultaneously active components.

The first is a *navigation skill*, which constitutes the primary autonomous behaviour required to fulfil the task objective. This skill governs the physical movement or stationary behaviour of the vehicle during that step and defines the primary termination condition for the step. Examples include *GoToXY* for waypoint navigation or *StationKeep* for position holding. The second component is a *filter skill list*, containing skills that run in parallel with the navigation skill and condition its execution by imposing safety constraints or operational boundaries without interrupting the primary manoeuvre. Filter skills such as *ObstacleAvoidance* or *BoundaryEnforcement* continuously monitor the environment and modify or constrain the navigation skill's outputs to maintain safe operation throughout execution. The third component is a *perception and data skill list*, containing skills that operate concurrently with the navigation skill to augment it with environmental awareness and data acquisition. These skills, such as *GetSeabedDepth* or *SetStreamRecording*, execute alongside the primary navigation behaviour and collect the sensor data or state information required to fulfil the task's data requirements as defined in the mission ontology.

This three-part structure mirrors the concurrency of maritime task execution, in which the vehicle navigates, enforces safety boundaries, and acquires data simultaneously rather than in sequence. The Task Agent must therefore not only select and sequence the navigation skill for each step, but also determine which filter and perception behaviours accompany it, and parametrise all three components consistently with the task objective and the parameters propagated from the Mission Agent.

In addition to the three skill components, each step in the Task Agent output also specifies stop conditions, failure modes, and expected outputs. Stop conditions define the mathematical or logical criteria that are expected to be satisfied for the step to be considered complete and execution to advance to the subsequent step. Failure modes enumerate the conditions under which the step should be considered to have failed, enabling the operator to anticipate and plan for contingencies during review. Expected outputs describe the data or state changes that the step is expected to produce upon successful completion.

3.4 Skill Formulation

Skills are realised by execution modules of three distinct kinds, distinguished by how their control or processing logic is obtained: deterministic skills, whose behaviour is given by a fixed, hand-specified function of sensor or state data; and learned skills. The latter include RL-based skills, whose behaviour is learned by a single policy trained offline and frozen at deployment; and HRL-based skills, in which a high-level policy composes a learned low-level primitive to realise a temporally extended behaviour. Across all kinds, a navigation skill's execution loop is exited once its termination condition, evaluated against the current vehicle state, is satisfied, at which point any concurrently active filter and perception skills are deactivated and the Skill layer advances to the next configuration in the approved plan.

Deterministic Skills

A deterministic skill realises its behaviour through a fixed, hand-specified function of sensor or system state rather than through a learned policy. Its mapping from input to output is defined explicitly in advance by the designer, rather than inferred from data. This stands in contrast to a learned skill, whose input/output mapping is acquired through training and is not specified by hand. The distinction concerns only how a skill's behaviour is obtained, and is independent of what the skill does: a skill may be hand-specified regardless of whether it senses the environment, maintains mission state, or commands the vehicle.

RL-Based Skills

A primitive navigation skill is realised by a dedicated RL policy, trained offline in physics-based simulation to produce the continuous behaviour the skill encodes. Since unstructured marine environments are highly stochastic and subject to unmodelled disturbances, handcrafted controllers are difficult to design with sufficient robustness for reliable field deployment; RL policies instead acquire manoeuvring behaviour through environment interaction, developing implicit representations of the vehicle dynamics that generalise across the disturbance conditions encountered during training. At deployment, these policies are loaded with frozen neural network weights, so that the state-to-action mapping is fixed at runtime and the policy does not continue to learn during execution, ensuring predictable behaviour and eliminating the risk of policy drift or unintended adaptation to novel conditions encountered in the field.

Termination of an RL-based skill is specified per behavioural archetype and reflects the logical completion of the manoeuvre it encodes, evaluated as a fixed criterion against the current vehicle state rather than learned. A position-reaching skill terminates once the vehicle's distance to a goal falls below a prescribed tolerance; a station-keeping skill terminates once a specified duration has elapsed while the vehicle remains within an acceptable position radius; a heading-alignment skill terminates once the angular error to a target heading crosses a defined threshold; and a skill conditioned on a physical or environmental criterion terminates once that criterion, rather than a purely geometric or temporal one, is satisfied for a minimum duration.

The continuous outputs of these policies are, before reaching the vehicle's hardware interface, strictly clipped to the vehicle's maximum actuation limits, ensuring that no command exceeding the physical capabilities of the vehicle is ever issued regardless of the policy output.

3.4.1 HRL-Based Skill: Target Survey

The most complex skill in the library, and the one developed as part of this dissertation, is a HRL-based skill for boundary survey. The boundary survey task addressed in this work requires a ASV to trace the perimeter of an *a priori* unknown object using only the returns from an onboard LiDAR sensor. The agent receives no information about the object's geometry at policy execution time; the structure of the environment is inferred entirely from sensor observations acquired during operation. The desired behaviour is a steady, counter-clockwise orbit of the object at a fixed standoff distance, keeping the object continuously within the sensor's field of view and advancing monotonically around its perimeter, with the object maintained on the vehicle's port side throughout.

Semi-MDP and Options Framework Formulation

As discussed in Section 2.2, the standard MDP formulation is poorly suited to tasks in which decisions are taken at different temporal resolutions and behaviours unfold over variable durations. The boundary survey task exhibits precisely this structure: a high-level planning decision, namely the selection of a waypoint along the object's perimeter, initiates a temporally extended execution phase during which a low-level controller drives the vehicle toward that waypoint over multiple simulation steps. This two-level temporal structure motivates a formulation in terms of the options framework [68], which extends the standard MDP to a Semi-MDP by allowing actions to persist for variable durations and by associating rewards with the outcomes of entire behavioural episodes rather than individual timesteps.

Formally, the *SurveyTarget* scenario is defined as a Semi-MDP $\langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where $\mathcal{S}$ is the high-level observation space, $\mathcal{A}$ is the primitive action space of the vehicle, Ω is the set of options available to the high-level policy, $\mathcal{R}$ is the semi-Markov reward function evaluated at option termination, $\mathcal{T}$ is the environment transition dynamics, and $\gamma \in [0, 1]$ is the discount factor. Each option $\omega \in \Omega$ is a triple $\omega = (I_\omega, \pi_\omega, \beta_\omega)$, where $I_\omega \subseteq \mathcal{S}$ is the initiation set, π_ω is the intra-option policy governing execution, and β_ω is the termination condition, following the formalism introduced in Section 2.2.

In the present instantiation, the high-level policy selects a body-frame waypoint at each decision step. This waypoint is then handed to a low-level, RL-based *GoToXY* skill, which constitutes the intra-option policy π_ω and drives the vehicle toward the selected waypoint for a maximum option horizon of H simulation steps. The option terminates upon any of the following conditions

being satisfied:

$$\text{done}_{\text{opt}} = \underbrace{\left(\|\mathbf{p} - \mathbf{w}^{\mathcal{W}}\| < r_{\text{wp}}\right)}_{\text{waypoint reached}} \vee \underbrace{(c_{\text{timer}} \leq 0)}_{\text{timeout}} \vee \underbrace{\text{terminal event}}_{\text{see below}}, \tag{3.3}$$

where $\mathbf{p}$ is the vehicle's world-frame position, $\mathbf{w}^{\mathcal{W}}$ is the active world-frame waypoint, r_{wp} is the waypoint capture radius, and c_{timer} is the remaining option timer. The high-level policy operates at the temporal scale of options, receiving observations and issuing new waypoints only at option boundaries, while the low-level policy operates at every simulation timestep Δt within the active option, as illustrated in Figure 3.3. Rewards are accumulated over the duration of each option and returned to the high-level policy only at option termination, making the reward function semi-Markov by construction.

State Space

The high-level observation space $\mathcal{S}$ is LiDAR-centric and contains no privileged information about the object's geometry. The complete observation presented to the policy at each option boundary is the concatenation:

$$o = \left[\, \tilde{\mathbf{d}},\ \tau,\ \hat{\mathbf{v}},\ \tilde{d^{\star}} \,\right] \in \mathbb{R}^{K+4}, \tag{3.4}$$

formed from the normalised LiDAR ranges ($\tilde{\mathbf{d}}$), option timer (τ), and commanded standoff distance ($\tilde{d^{\star}}$), and the body-frame velocity ($\hat{\mathbf{v}}$).

The vehicle carries a single LiDAR sensor whose point clour is processed to yield K rays distributed uniformly over a full 360° aperture with maximum range L, and the measured ranges

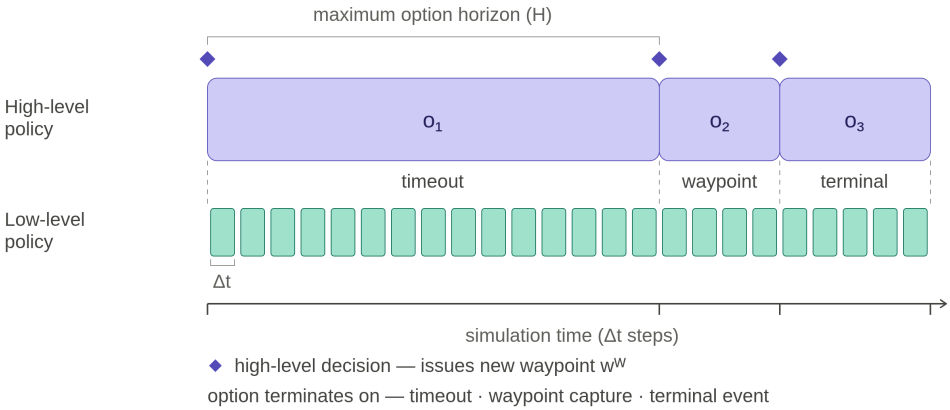


Figure 3.3: Temporal abstraction in the hierarchical controller. The high-level policy operates at the granularity of options (top), issuing a new body-frame waypoint $\mathbf{w}^{\mathcal{W}}$ only at option boundaries, while the frozen low-level *GoToXY* policy executes at every simulation timestep Δt (bottom). Each option runs for at most H steps and terminates as soon as any condition of done_{opt} is met: option o_1 exhausts the full horizon and terminates on timeout, whereas o_2 and o_3 end earlier upon waypoint capture and a terminal event, respectively. (AI-generated illustrative diagram)

enter the observation after normalisation and clipping:

$$\tilde{\mathbf{d}} = \mathrm{clip}(\mathbf{d}/L, 0, 1)\,, \tag{3.5}$$

The world-frame velocity $\mathbf{v}$ is rotated into the vehicle body frame and normalised by the maximum linear speed,

$$\hat{\mathbf{v}} = \frac{R(\theta)^\top \mathbf{v}}{v_{\max}^{\mathrm{lin}}}, \tag{3.6}$$

where $R(\theta)$ is the planar rotation matrix associated with the vehicle heading θ. The remaining option duration, τ is normalised against the option horizon H and clipped between 0 and 1, informing the policy of the time remaining before the current option expires. The commanded standoff distance is normalised by the sensor range, $\tilde{d}^\star = d^\star/L$. This value is included in the observation in order to make the policy aware of the radius at which it is required to orbit on the current episode, allowing a single network to realise the survey behaviour across a range of standoff distances.

Action Space and Waypoint Mapping

The high-level policy outputs a continuous action $a = (a_x, a_y)$ with $a_i \in [-1, 1]$ for $i \in \{x, y\}$, which is decoded into a body-frame waypoint. The mapping is designed so that the proposed waypoint always lies ahead of the body-frame, with a forward offset drawn from a bounded range and a lateral offset spanning both sides of the vehicle's centreline:

$$w_x = w_x^{\min} + \tfrac{1}{2}(a_x + 1)\,(w_x^{\max} - w_x^{\min}), \qquad w_y = a_y\, w_y^{\max}. \tag{3.7}$$

The body-frame waypoint $\mathbf{w}^{\mathcal{B}} = (w_x, w_y)$ is then transformed into the world frame and attached to the goal landmark:

$$\mathbf{w}^{\mathcal{W}} = \mathbf{p} + R(\theta)\,\mathbf{w}^{\mathcal{B}}. \tag{3.8}$$

This parametrisation ensures that the action space encodes relative navigation intent rather than absolute coordinates, making the policy invariant to the vehicle's global pose and naturally supporting the object-relative orbiting behaviour required by the task. Figure 3.4 illustrates this parametrisation.

Termination Conditions

An episode terminates when any of three conditions is met: the agent collides, loses sight of the target, or completes a lap around it. For the first, the vehicle footprint is approximated by a fixed convex polygon $\mathcal{H}$, expressed in the body frame and scaled to the vehicle length ℓ and width w, with a flat stern edge and a tapered bow approximating the vessel outline. A collision is registered when any boundary landmark penetrates to within a small clearance of this footprint:

$$\mathrm{done}_{\mathrm{col}} = \left(\min_i \mathrm{dist}\left(\mathbf{b}^{(i)}, \mathcal{H}\right) < r_{\mathrm{pt}} + \epsilon\right), \tag{3.9}$$

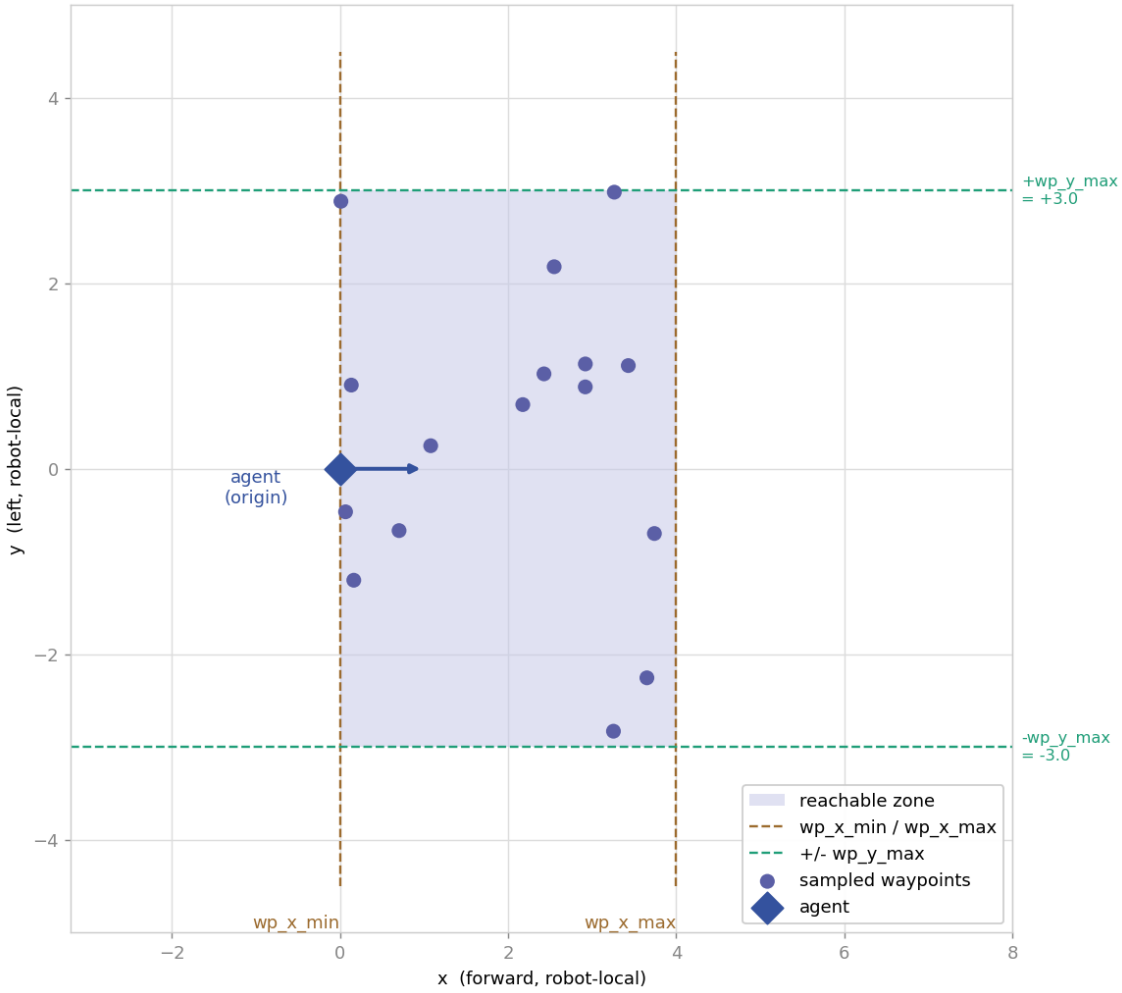


Figure 3.4: Body-frame parametrisation of the high-level action space. The continuous action $\mathbf{a} = (a_x, a_y)$ for $i \in \{x, y\}$ is decoded into a waypoint within the shaded reachable zone ahead of the vehicle, with forward offset w_x and lateral offset w_y expressed in the body frame relative to the vehicle heading (forward = $+x$). The blue markers show waypoints sampled across the action space; the agent is shown at the origin.

where $\mathrm{dist}(\mathbf{b}^{(i)}, \mathcal{H})$ is the Euclidean distance from boundary landmark $\mathbf{b}^{(i)}$ to the footprint polygon, taken as zero when the landmark lies inside it, r_{pt} is the boundary landmark radius, and ϵ is a small collision margin.

The second condition, loss of the object, uses a binary visibility flag ν derived from the LiDAR. The object is declared lost when an option concludes, either by reaching its waypoint or by exhausting its timer, and at that moment there is no LiDAR information:

$$\begin{aligned} h_k &= \mathbb{1}[d_k < L], \\ \nu &= \max_k h_k \in \{0, 1\}, \\ \mathrm{done}_{\mathrm{lost}} &= \Big[\Big(\|\mathbf{p} - \mathbf{w}^{\mathcal{W}}\| < r_{\mathrm{wp}}\Big) \vee (c_{\mathrm{timer}} \leq 0)\Big] \wedge (\nu = 0)\,. \end{aligned} \tag{3.10}$$

This couples loss of the object directly to the sensor rather than to an explicit range cap, terminating the episode once the vehicle can no longer perceive the structure it is surveying.

Finally, the third condition, lap completion, couples episode termination to successful task execution: an episode that ends via done_{lap} corresponds to a complete, uninterrupted orbit of the object, which constitutes the primary success criterion for the boundary survey task. Lap completion is detected by accumulating the vehicle's signed progress around the object's perimeter throughout the episode; once this cumulative signed advancement reaches a full revolution of the perimeter, in either direction, the condition is satisfied. The symmetry of this accumulator with respect to orbit direction means that a backtracking trajectory accumulating an equivalent amount of total perimeter traversal also triggers termination.

The union of all three conditions defines the episode termination signal:

$$\text{done} = \underbrace{\text{done}_{\text{col}}}_{\text{collision}} \vee \underbrace{\text{done}_{\text{lost}}}_{\text{object lost}} \vee \underbrace{\text{done}_{\text{lap}}}_{\text{lap completed}} . \tag{3.11}$$

These conditions act as the terminal event referenced in Equation 3.3, allowing an active option to be interrupted before its horizon expires whenever a safety-critical or task-completion state is reached.

Reward Design

The reward function is defined at the granularity of options and is evaluated only at option termination. No reward is emitted at intermediate simulation timesteps; instead, statistics are accumulated over the duration of the active option and combined into a single scalar that is returned to the high-level policy when the option terminates. Accordingly, the reward is designed to score the quality of the high-level decision, namely the placement of each waypoint.

The total option reward is the sum of four contributions: a waypoint standoff term, a perimeter-progress term, a visibility term, and a set of terminal penalties,

$$R_\omega = r_{\text{wp}} + r_{\text{prog}} + r_{\text{vis}} + r_{\text{col}} + r_{\text{lost}} + r_{\text{in}}, \tag{3.12}$$

each of which is detailed below.

The waypoint standoff term rewards the high-level policy for proposing waypoints that lie at the desired distance from the object boundary. Let $d^\star$ denote the desired standoff distance and δ the standoff tolerance. At the start of each option, the nearest-boundary distance of the proposed world-frame waypoint $\mathbf{w}^{\mathcal{W}}$ is computed as $d_{\text{wp}} = \min_i \|\mathbf{w}^W - \mathbf{b}^{(i)}\|$, and the standoff error $e_{\text{wp}} = |d_{\text{wp}} - d^\star|$ is recorded. The corresponding reward is the indicator

$$r_{\text{wp}} = w_{\text{wp}} \mathbb{1}\left[e_{\text{wp}} \leq \delta \right], \tag{3.13}$$

awarding the full weight when the proposed waypoint lies within one tolerance band of the desired standoff. The term therefore scores the placement decision as a pass-or-fail standoff test, and the

policy is not penalised for any failure of the low-level controller to reach the waypoint, except insofar as such a failure results in a collision or in losing the target.

The progress term encourages the vehicle to advance monotonically around the object perimeter in the desired survey direction, keeping the object on the vehicle's left side. Progress is measured as the signed displacement, around the object perimeter, of the vehicle's nearest point of contact with the boundary between the start and the end of the option, reflecting the shortest path of advancement rather than the absolute distance travelled. The progress reward is proportional to a clipped measure of this signed displacement, so that advancement in the desired direction yields a positive reward, backtracking yields a negative reward, and the contribution saturates once the option advances the equivalent of a few boundary points in either direction, bounding the per-option influence of the term.

The visibility term rewards the policy for keeping the object within the LiDAR field of view throughout the option, which is a prerequisite for the LiDAR-centric observation to remain informative. Reusing the binary visibility flag $v_t \in \{0, 1\}$, the visible fraction over an option of T steps is $f_{\text{vis}} = \frac{1}{T}\sum_{t=1}^{T} v_t$, and the visibility reward is

$$r_{\text{vis}} = w_{\text{vis}}\,(2f_{\text{vis}} - 1)\,, \tag{3.14}$$

which is symmetric about zero, rewarding options during which the object remained continuously visible and penalising those during which it left the sensor aperture, while the small weight ensures that visibility acts as a shaping signal rather than dominating the standoff and progress objectives.

Finally, three terminal penalties discourage unsafe or task-violating behaviour and are applied when the corresponding condition is met during the option. A collision penalty κ_{col} is incurred if the vehicle approaches the boundary closer than the collision threshold defined in Equation 3.9 at any step of the option; an inside-target penalty κ_{in} is incurred if the proposed waypoint lies inside the object or the vehicle enters the object interior at any step, discouraging the policy from directing the vehicle across the structure rather than around it; and a lost-target penalty κ_{lost} is incurred when the option terminates with the object outside the LiDAR field of view, coinciding with the lost termination condition of Equation 3.10. The aggregate terminal penalty is

$$r_{\text{col}} + r_{\text{in}} + r_{\text{lost}} = \kappa_{\text{col}}\,\mathbb{1}\,[\text{collision}] + \kappa_{\text{in}}\,\mathbb{1}\,[\text{inside}] + \kappa_{\text{lost}}\,\mathbb{1}\,[\text{lost}]. \tag{3.15}$$

These conditions coincide with the terminal events that interrupt an active option, so a penalised option is also the final option of its episode, coupling the per-option reward to the episode-level safety and task-completion criteria. The complete set of reward weights and thresholds is summarised in Table 4.4, in Section 4.1.2.

3.5 Chapter Conclusion

This chapter has presented the proposed framework for mission-level ASV autonomy, which couples LLM-based mission planning with RL-based skill execution around a single organising prin-

ciple: the separation of semantic reasoning, performed at planning time, from physical actuation, delegated to a fixed library of validated skills. This principle is realised through a hierarchical, four-layer translation engine that progressively refines a free-form operator command across the Mission, Task, Skill, and Action layers. A *Mission* captures the high-level semantic objective; a *Task* is a discrete assignment that advances one component of that objective; a *Skill* is a parametrised, modular capability drawn from a predefined library; and an *Action* is the lowest tier, comprising the continuous actuator commands or discrete sensor triggers issued during execution. Since the layers are connected by well-defined interfaces, and since the generated plan is fixed and subject to explicit operator approval before any actuation begins, an erroneous decomposition cannot directly affect the vessel, each executed manoeuvre remains traceable to an approved task, and the language, perception, and execution components can be developed and replaced independently.

Two complementary contributions instantiate this architecture. The first, the Agentic Mission Planner, addresses the specification side of the problem through two fine-tuned language models deployed entirely onboard. A Mission Agent, guided by a four-element mission ontology, decomposes the operator command into an ordered set of high-level tasks drawn from a domain-specific vocabulary, while a subsequent Task Agent expands each task into a concurrently executed configuration of skills selected from a given library of distinct capabilities and parametrised consistently with the task objective. The compositional structure of the Task Agent output, organised around a primary navigation skill running alongside parallel filter and perception skills, mirrors the concurrency of maritime task execution, in which the vehicle navigates, enforces safety boundaries, and acquires data simultaneously rather than in sequence.

The second contribution, the hierarchical survey skill, addresses the execution side. Formulated as a Semi-MDP within the options framework, it equips a high-level policy that reasons over a LiDAR-centric observation, free of any privileged knowledge of the object's geometry, and composes the low-level *GoToXY* navigation primitive to orbit an *a priori* unknown structure at a commanded standoff distance. Its observation, body-frame action mapping, termination conditions, and option-level reward were each formulated to make the high-level decision, namely the placement of each waypoint, the unit of learning. This skill exposes the same parametrised interface and termination logic as the remainder of the library, so that the Task Agent selects it exactly as it would any other skill; the flexible semantic planner and the robust learned behaviour thereby combine into a single, coherent execution pipeline.

Chapter 4

Methodology and Results

4.1 Methodology

This section reports the implementation details that realise the concepts formulated in Chapter 3: the models, hardware, and training procedures behind the Agentic Mission Planner (Section 4.1.1), the reward values, simulation framework, and training details behind the hierarchical survey skill (Section 4.1.2), and the real-world test site used for field validation (Section 4.1.3). The results obtained with this methodology are reported in Sections 4.2–4.5.

4.1.1 Agentic Mission Planner

To identify the most suitable language model for each abstraction layer and to quantify the contribution of supervised fine-tuning, a comparative benchmarking study was conducted, evaluating candidate models in both their base and Low-Rank Adaptation (LoRA) fine-tuned configurations across the Mission and Task levels independently.

Five metrics are reported across both benchmark levels. *F1* is computed over the set of predicted versus expected task or skill names, capturing partial credit for decompositions that recover most but not all of the reference elements. *Hallucination rate* measures the proportion of outputs containing names absent from the reference vocabulary, providing a direct indicator of the degree to which a model generates out-of-domain content. *Schema Validity* reports the proportion of outputs that parse as well-formed JSON objects conforming to the expected output schema, reflecting the model's capacity to produce structured, machine-readable output. *Latency* reports the mean inference time per query, measured on the deployment hardware, as this directly determines the planning time experienced by the operator before execution is initiated. *Tokens* reports the mean total number of tokens generated per query.

The candidate model set was constrained by two practical requirements imposed by the deployment hardware: models must be open-source to permit onboard execution without network dependencies, and their parameter count must remain within the memory budget of the onboard GPU. These constraints led to the selection of models up to 3 billion parameters from openly available model families, specifically the Llama and Qwen2.5 families, which represent the most

mature and widely adopted open-source instruction-tuned model lines at this scale. An exception was made for a 4-bit quantised variant of Llama-3.1-8B, which, despite exceeding the nominal parameter limit, fitted within the VRAM budget after quantisation and was therefore included as an upper-bound reference for decomposition quality. For each model family, both base and instruction-tuned variants were evaluated where available, as instruction tuning is known to improve structured output compliance in zero-shot settings. The complete candidate set evaluated at both levels is as follows: Llama-3.2-1B, Llama-3.2-1B-Instruct, Llama-3.2-3B, Llama-3.2-3B-Instruct, Qwen2.5-3B, Qwen2.5-3B-Instruct, and Llama-3.1-8B (4-bit quantised). For base models, the vocabulary list was given as part of the prompt during inference. The models ultimately selected from this set, and the procedure used to adapt and deploy them, are described next; the comparative results that motivated their selection are reported in Section 4.2.1.

Semantic decomposition at the Mission layer is performed by a fine-tuned model, adapted to the structured decomposition task using a custom LoRA adapter trained on a dataset of {*mission prompt*, *expected task sequence*} pairs under a supervised fine-tuning objective. This training dataset was constructed manually, and with assistance from a separate LLM to increase coverage of mission phrasings, resulting in 125 training samples, 25 validation samples and 25 test samples designed to cover the diversity of high-level maritime mission templates expressible through the proposed ontology, spanning different combinations of objectives, target types, sensor modalities, and operational constraints. The dataset was constructed exclusively from tasks belonging to the domain vocabulary, presented in Table 4.1, so that the fine-tuning process implicitly constrains the Mission Agent's output distribution to that terminology without requiring the vocabulary to be explicitly supplied to the model at inference time.

The base model is quantised to 4-bit precision using the Unsloth library to satisfy the onboard memory budget of the deployment hardware. The LoRA adapter was trained with a rank of 16, selected through experimentation as the configuration offering the best balance between representational capacity and computational overhead. Training was conducted over 5 epochs with a learning rate of 2×10^{-4} and a batch size of 1; these hyperparameters were likewise determined through experimentation, with higher learning rates found to produce unstable training and larger batch sizes exceeding the available GPU memory during fine-tuning. Fine-tuning was performed offline on a machine equipped with an Intel i5 CPU, an NVIDIA RTX 4070 GPU, and 16 GB of RAM.

Its output is a structured JSON object whose schema is designed to provide both the information required for downstream execution and the contextual transparency needed for operator review. The output contains three top-level fields: a *tasks* array, an *assumptions* list, and a *constraints* list. Each entry in the *tasks* array is composed of three fields: a task title drawn from the domain vocabulary; a parameter dictionary carrying any task-level arguments, such as target coordinates or target description strings; and a task prompt, which is a concise natural language instruction that encapsulates the intent of that task and serves as the direct input to the Task Agent in the subsequent layer. The *assumptions* list enumerates the conditions the architecture has implicitly assumed in order to generate the plan, while the *constraints* list identifies operational

Table 4.1: Complete list of available tasks organised by functional category.

Cat.	Task	Description	Parameters
System	Diagnostics	Sensor health check (GPS, cameras, horizon, lens)	—
Positioning & Manoeuvre	Transit	Navigate to a known coordinate	target_position $[x, y]$
	FollowTarget	Track and follow a moving target	target_id / target_position
	PerformOrbit	Circular or arc orbit around a structure	center, radius, speed, arc
	ParallelStructurePass	Straight pass parallel to a structure face	side_id, length, dist., speed
	DockingApproach	GPS-based approach to a dock/berth	goal_position $[x, y]$
	VisualDockingApproach	Camera-guided final approach	target_feature, distance
	DriftAudit	Cut propulsion and log passive drift	duration
	KeepStation	Hold position at current location	duration
Survey & Inspection	HorizontalPanorama	Panoramic imagery across sector headings	sectors, overlap, wait
	ThermalHotspotLocalizer	Sweep to localise thermal hotspots	scan_width_degrees
	FeatureStare	Lock onto a feature and record steady video	feature_type, duration
	GlareFreeApproach	Reposition for glare-free imagery	target_feature
	ThermalTimeLapse	Periodic thermal measurements at station	duration, interval
	BoatLandingClearanceSweep	LiDAR scan of boat landing area	—
	BathymetryGridSurvey	Lawnmower seabed depth survey	center, width, length, spacing
	SubseaCableTrack	Follow a subsea cable via sonar	start, heading, max_dist
	SurveyTarget	Inspection square around a target position	target_position $[x, y]$, radius
	SearchAndDetect	Search area to find and confirm a specific target	target_description
Data	SmartDataHarvest	Prioritise data upload over limited bandwidth	target_list
	AutoGenerateReportSummary	Compile observations into a mission report	—

conditions that, if violated, could lead to incomplete or faulty mission execution. Together, these fields provide the operator with sufficient context to evaluate the soundness of the generated plan before approving execution.

Semantic decomposition at the Task layer is performed in much the same way, by a fine-tuned model equipped with a custom LoRA adapter trained on {task prompt, expected skill set} pairs under a supervised fine-tuning objective. As with the Mission Agent, the base model is quantised to 4-bit precision using the Unsloth library, and its training dataset was generated programmatically using a dedicated script that systematically sampled task configurations from the full skill vocabulary: for each sample, a task type was drawn from the available task set, and a valid skill sequence was constructed by sampling a primary navigation skill consistent with the task type, a subset of compatible filter skills, and a subset of compatible perception skills, with all parameters populated from the parameter spaces defined in the skill library, presented in Table 4.2, and a corresponding task prompt generated in natural language. This programmatic procedure allowed the dataset to be constructed at a scale substantially larger than the mission-level one, yielding 10,000 training samples, 500 validation samples and 500 test samples, with train, validation and test sets constructed separately to contain distinct samples, ensuring systematic coverage of the skill vocabulary and parameter space.

Table 4.2: Comprehensive list of robot skills on Navigation, Perception, and Mission Management.

Cat.	#	Skill	Description	Parameters
Navigation	1	GoToXY	Navigates to given coords	x, y, ψ
	2	StationKeep	Maintains position within specified radius	Radius, Time
	3	RotateToHeading	Pivots in place to a compass bearing	Target ψ
	4	LockHeading	Maintains nose direction regardless of drift	None
	5	NudgeYaw	Performs a small, precise rotation	$\Delta\psi$
	6	ArcTurn	Executes a sweeping turn	Radius, $\Delta\psi$
	7	DriftWithCurrent	Cuts propulsion to float naturally	Time
	8	VisualStationKeep	Holds visual feature centered in frame	Target ID
	9	ObstacleAvoidance	Alters path to maintain safety margin	Min. Dist.
	10	BoundaryEnforcement	Constrains motion within virtual boundary	Poly-coords
	11	WaitForStability	Waits until pitch/roll drops below threshold	Threshold, Time
Perception	12	DetectObjects	Runs object detector for specific classes	Class List
	13	TrackObjects	Maintains persistent identities over time	Object ID
	14	OCRScanning	Reads text in a specific frame region	Region
	15	GetThermalStats	Returns statistical temp data for region	ROI
	16	LocateHotspot	Returns coordinates of hottest point	Threshold
	17	CheckLensClearness	Detects if droplets obscure the lens	Camera ID
	18	GetImageQualityMetrics	Analyzes current video frame usability	Camera ID
	19	CalculateVisualOffset	Returns pixel error from center to target	Target ID
	20	GetSectorDistance	Returns min LiDAR distance in a slice	Bearing
	21	DistanceToObstacle	Returns LiDAR distance in fixed direction	FOV
	22	FitCylinderModel	Filters point cloud to find large curve	Cloud data
	23	CheckPlanarity	Checks if surface is flat or curved	ROI
	24	DetectSonarFeature	Simple blob detection on sonar display	Type
	25	GetSeabedDepth	Returns current depth below ASV	Sensor ID
	26	GetHorizonAngle	Determines how tilted the horizon is	Sensor ID
	27	GetGPSLocation	Returns absolute geographic position	Lat/Lon
Management	28	SelectActiveStream	Switches focus between fixed cameras	Stream ID
	29	SetStreamRecording	Starts or stops writing data to disk	Sensor ID, File Tag
	30	BufferLoopCapture	Saves recent video from buffer	Buffer Size
	31	CaptureGeoSnapshot	Saves frames with metadata sidecar	Sensor ID, note
	32	LogLidarSlice	Saves current LiDAR sweep as 2D profile	Scan Angle
	33	LogEnvironmentState	Saves snapshot of environmental sensors	None
	34	GetRelativePose	Returns pose relative to local frame	Target
	35	VerifyGeotagQuality	Checks horizontal accuracy of the GPS	Threshold
	36	CreateMapMarker	Drops pin on internal map at location	Label
	37	RegisterObservation	Updates global state of data logger	Observation

The LoRA adapter for the Task Agent was trained with a rank of 8, lower than the rank of 16 used for the Mission Agent, reflecting the more constrained output distribution of the task-level decomposition problem; higher ranks were found to offer no measurable improvement in decomposition accuracy while increasing training time and memory requirements. Training was conducted over 2 epochs with a learning rate of 2×10^{-4} and a batch size of 1, on the same offline hardware as the Mission Agent.

Its output is likewise a structured JSON object, containing a top-level *skill_sequence* list in which each entry corresponds to one step in the execution plan and is structured around the navigation-skill, filter-skill, and perception-skill components described in Section 3.3. Each step therefore contains a *navigation_skill* field with the skill name and its parametrised arguments, a *filter_skills* list with zero or more concurrently active safety skills and their parameters, and a *perception_skills* list with zero or more concurrently active perception or data management skills and

their parameters, alongside *stop_conditions*, *failure_modes*, and *expected_outputs* fields. A representative example is provided in Listing 4.1, corresponding to a single waypoint step from a target inspection task, where the vehicle navigates to a waypoint adjacent to the target while enforcing a boundary constraint and recording georeferenced imagery.

Listing 4.1: Representative Task Agent output for a single waypoint step within a target inspection task.

```
{
  "skill_sequence": [
    {
      "navigation_skill": {
        "name": "GoToXY",
        "parameters": {
          "x": 7.1,
          "y": 7.1,
          "yaw": 0.0
        }
      },
      "filter_skills": [
        {
          "name": "BoundaryEnforcement",
          "parameters": {
            "poly_coords": [[-20,-20],[20,-20],
                            [20,20],[-20,20]]
          }
        }
      ],
      "perception_skills": [
        {
          "name": "GetGPSLocation",
          "parameters": {}
        },
        {
          "name": "SetStreamRecording",
          "parameters": {
            "sensor_id": "camera_front",
            "file_tag": "inspection_wp1"
          }
        }
      ],
      "stop_conditions": [
        "Distance to goal position below 1.0 m"
      ],
      "failure_modes": [
        "Goal position unreachable within timeout",
```

```
        "Boundary violation detected"
      ],
      "expected_outputs": [
        "Vehicle at waypoint within tolerance",
        "Georeferenced imagery captured"
      ]
    }
  ]
}
```

Both agents are deployed together onboard the vehicle rather than on separate hardware. To guarantee memory-efficient onboard execution, the Mission Agent adapter is released from GPU memory before the Task Agent adapter is loaded, so that only one adapter occupies VRAM at any time. The complete architecture executes locally on the vehicle's onboard compute hardware, consisting of an Intel i7 CPU, an NVIDIA GeForce RTX 3060 GPU (6 GB VRAM), and 16 GB of RAM.

To validate this pipeline end-to-end before field deployment, two representative missions were designed to evaluate the agentic architecture in simulation, spanning distinct operational objectives and complementary skill compositions. Both missions were executed in the Gazebo simulator using the Nautilus ASV model, within a simulated marine environment, with continuous vehicle manoeuvring governed by the dedicated RL navigation policies Their setup is described below; the corresponding results are reported in Section 4.2.

Mission 1 was initiated with the operator prompt: *"Locate the underwater debris field and map the surrounding seabed area with a 20x20 bathymetry grid with 5 m spacing."* This free-form natural language instruction was passed to the Mission Agent, which decomposed the objective into two sequential tasks. The first, a *SearchAndDetect* task, is responsible for localising the debris field prior to any survey activity, reflecting the architecture's inference of an implicit localisation prerequisite from the descriptive target reference. The second, a *BathymetryGridSurvey* task, translates the operator's spatial specification directly into a structured execution plan. The Task Agent subsequently expanded the *BathymetryGridSurvey* task into a sequence of ten sequential *GoToXY* skills, with concurrent *GetSeabedDepth* perception calls accompanying each waypoint to capture bathymetric measurements throughout the survey. This decomposition is also a representative instance of the Mission Agent output schema described above: it is given in full in Listing 4.2, and its corresponding mission control interface is depicted in Figure 4.1.

Listing 4.2: Representative Mission Agent output for a bathymetric survey command, corresponding to the Mission 1 setup described above.

```
{
  "tasks": [
    {
      "title": "SearchAndDetect",
      "parameters": {
        "target_description": "underwater debris field"
```

```
      },
      "task_prompt": "Search the area and locate the underwater
                      debris field."
    },
    {
      "title": "BathymetryGridSurvey",
      "parameters": {
        "center": [-99, -99],
        "width": 20,
        "length": 20,
        "spacing": 5
      },
      "task_prompt": "Map the seabed around the located debris
                      field with a 20x20 grid at 5m spacing."
    }
  ],
  "assumptions": [
    "The debris field is within the operational area.",
    "Sonar and depth sensors are operational."
  ],
  "constraints": [
    "Survey must not exceed designated boundary.",
    "Vehicle must return to base upon completion."
  ]
}
```


For the purposes of evaluating the physical execution of this plan, the required perception skills were treated as simulated operations that directly output the necessary environmental coordinates, as autonomous seabed mapping falls outside the scope of the present work.

Mission 2 was initiated with the operator prompt: *"Find Durius and collect data near the J-tubes. Summarize it, then come back."* This unstructured command, which makes no reference to explicit coordinates and employs domain-specific terminology not present in generic language corpora, was passed to the Mission Agent. From this input, the Mission Agent decomposed the objective into four distinct sequential tasks: a *SearchAndDetect* task to acquire and localise the target structure, a *SurveyTarget* task to organise the data collection inspection, an *AutoGenerateReportSummary* task for data consolidation and logging, and a final *Transit* task to return the vehicle to its origin. This decomposition demonstrates the Mission Agent's capacity to interpret domain-specific terminology, as neither "Durius" nor "J-tubes" are generic descriptors, yet the model correctly inferred the operational intent and mapped it to an appropriate and complete task sequence. The Task Agent then expanded each task into its corresponding skill configuration. The *SurveyTarget* task was expanded into eight sequential *GoToXY* skills, arranged in proximity to the target structure, with *SetStreamRecording* and *GetGPSLocation* perception calls assigned at each waypoint to capture georeferenced inspection data. The *AutoGenerateReportSummary* task

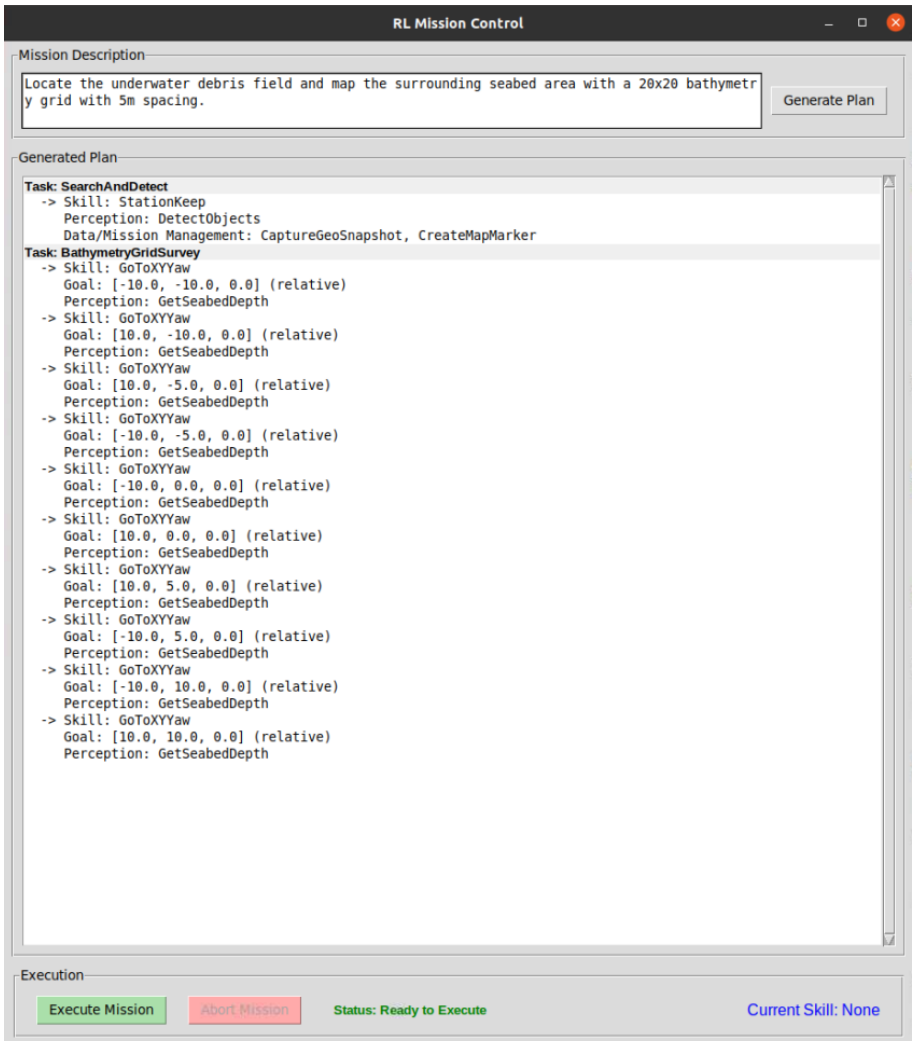


Figure 4.1: Mission User interface showing the generated plan for a bathymetric survey, decomposed into a *SearchAndDetect* task followed by a *BathymetryGridSurvey* task with ten sequential *GoToXY* skills and concurrent *GetSeabedDepth* perception calls.

triggered a *StationKeep* skill alongside concurrent *RegisterObservation* and *CreateMapMarker* mission management skills, consolidating the collected data before the vehicle returned to its origin via a final *GoToXY* call. The complete generated plan, as displayed in the mission control interface, is shown in Figure 4.2.

As in Mission 1, target localisation was provided externally, as autonomous structure detection falls outside the scope of the present work. Continuous vehicle manoeuvring in both missions is governed by the same RL navigation policies.

Beyond these two missions and the comparative benchmarking study, which evaluates decomposition quality in aggregate over a held-out test set, a qualitative robustness analysis was additionally conducted on the selected Mission Agent. Aggregate metrics offer a useful measure of overall decomposition quality but limited insight into the specific behaviours that contribute to successful or unsuccessful outcomes, so this analysis instead examines selected examples from the

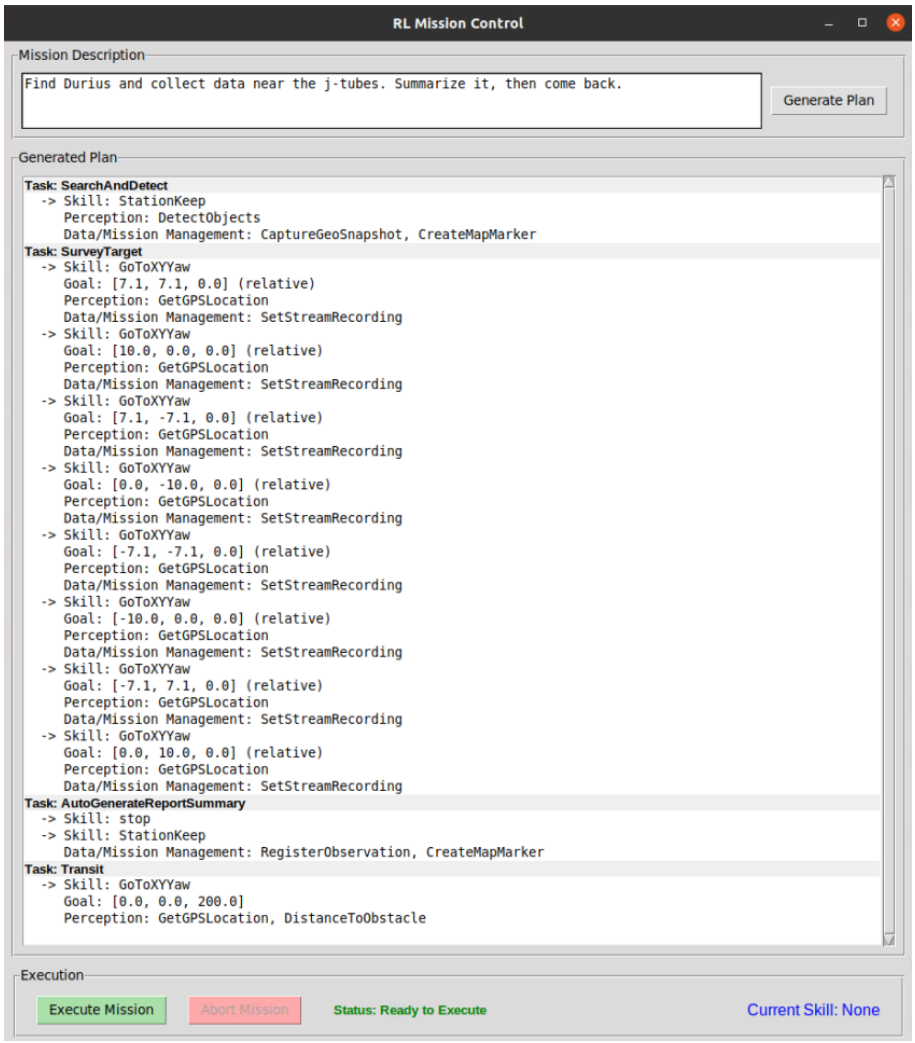


Figure 4.2: Mission User interface showing the generated plan for a target inspection mission, decomposed into four sequential tasks: *SearchAndDetect*, *SurveyTarget*, *AutoGenerateReportSummary*, and a return *Transit*, each expanded into the corresponding skill sequences by the Task Agent.

test set, together with their corresponding Mission Agent outputs. By analysing individual predictions, it is possible to identify recurring patterns in the model's reasoning, assess how effectively mission intents are decomposed into structured tasks, and characterise common failure modes that are not readily apparent from aggregate metrics alone, contextualising the benchmark results with a deeper view of the Mission Agent's strengths and limitations on the evaluation distribution.

To characterise this behaviour, the 9 incorrect predictions produced by the fine-tuned Llama-3.2-3B-Instruct Mission Agent over the 25-sample test set were manually reviewed. Each incorrect output was examined against its expected task sequence and annotated with a failure type according to the nature of the discrepancy. The resulting failure taxonomy and the findings of this review are reported in Section 4.2.2.

4.1.2 Hierarchical Survey Skill

Evaluation Protocol

The high-level policy is trained with the PPO configuration and environment parameters described below. This section sets out the protocol under which a trained policy is evaluated, together with the two controlled experiments used to characterise its behaviour: a study of generalisation to the concave L-shape class, and an ablation of the reward terms; results for all three are reported in Section 4.3.

Each trained policy is evaluated over 100 independent episodes. At every reset a target shape is drawn from the previously described classes and a new object instance is generated, so that reported performance reflects generalisation over unseen object instances rather than memorisation of specific geometries. An episode is counted as successful when it terminates through lap completion, corresponding to a complete orbit of the object. The following metrics are recorded:

- Success rate: the fraction of episodes terminating through lap completion.
- Collision rate: the fraction of episodes terminating through collision with the object boundary.
- Loss-of-target rate: the fraction of episodes terminating through the lost-target condition, in which an option concludes with no LiDAR ray returning a value.
- Mean standoff error: the time-averaged absolute deviation of the vehicle's distance to the nearest boundary point from the desired standoff distance $d^\star$, in metres.
- Episode length: the number of simulation steps to termination.

All quantitative evaluation is performed in the Vectorised Multi-Agent Simulator (VMAS) environment. The trained survey policy is also deployed in the Gazebo simulator, against the four fixed structures described in Section 4.1.3, to assess transfer beyond the VMAS environment in which it was trained. This Gazebo evaluation applies the same metrics but reports them alongside representative trajectories over 40 episodes distributed across the 4 targets, rather than over the randomised 100-episode protocol used in VMAS; and the Gazebo results are interpreted as qualitative evidence of transfer rather than as a precise performance estimate. It is also important to note that for deployment in this simulator a lower maximum velocity was used, due to the simulator's realist modelling of hull inertia and hydrodynamic drag, which lead the agent to overshoot when arriving at the waypoints; the reduced velocity therefore results in longer, but more controlled, trajectories.

High-Level Policy

The high-level policy is responsible for proposing body-frame waypoints at each option boundary and is parameterised as a stochastic Actor-Critic (AC) network. Both the actor and the critic are

implemented as fully connected Multilayer Perceptrons (MLPs) with the same depth and width and Tanh activation functions, the precise dimensions of which are listed in Table 4.3. The actor takes the observation vector o as input and outputs the mean $\mu \in \mathbb{R}^2$ and standard deviation $\sigma \in \mathbb{R}^2$ of a diagonal Gaussian distribution over the action space, from which the waypoint action is sampled via a Tanh-squashed Normal distribution:

$$a \sim \text{TanhNormal}(\mu_\theta(o), \sigma_\theta(o)), \qquad a \in [-1,1]^2, \tag{4.1}$$

where θ denotes the actor parameters. The Tanh squashing ensures that sampled actions remain within the bounded action space without hard clipping, while preserving a well-defined log-probability for use in the policy gradient update.

The mean and standard deviation are extracted from the final linear layer of the actor by a deterministic parameter mapping. Denoting the $2d_a$-dimensional output of that layer by $\mathbf{y}$, where $d_a = 2$ is the action dimension, the vector is partitioned into two equal halves $\mathbf{y} = [\mathbf{y}_\mu, \mathbf{y}_\sigma]$, and the distribution parameters are obtained as

$$\mu = \mathbf{y}_\mu, \qquad \sigma = \text{softplus}(\mathbf{y}_\sigma + b) + \sigma_{\min}, \qquad \text{softplus}(x) = \log(1 + e^x), \tag{4.2}$$

where the bias b is chosen so that the standard deviation initialises near unity, and $\sigma_{\min}$ is a small positive floor guaranteeing a strictly positive, numerically valid scale. The mean is therefore read off directly, while the standard deviation is passed through a softplus to enforce positivity.

The critic shares the same MLP architecture but produces a scalar state-value estimate $V_\phi(o) \in \mathbb{R}$ from the same observation vector. The two networks share no trunk and are optimised jointly.

The high-level policy is trained using PPO, and the training is conducted end-to-end with the low-level *GoToXY* policy: the high-level actor proposes waypoints, the low-level policy executes them over the option horizon, and the resulting semi-Markov rewards are returned to the high-level actor only at option termination. The low-level policy weights are not updated at any point during high-level training.

The clipped surrogate objective is:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\Big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\,\hat{A}_t\Big)\right], \tag{4.3}$$

where $r_t(\theta) = \pi_\theta(a_t \mid o_t)/\pi_{\theta_{\text{old}}}(a_t \mid o_t)$ is the probability ratio between the updated and old policies, $\hat{A}_t$ is the Generalised Advantage Estimate (GAE), and ε is the clipping hyperparameter. The full training objective combines the policy loss with a critic regression loss and an entropy bonus:

$$\mathcal{L}(\theta, \phi) = \mathcal{L}^{\text{CLIP}}(\theta) - c_1\mathcal{L}^{\text{critic}}(\phi) + c_2\mathcal{H}[\pi_\theta(\cdot \mid o_t)], \tag{4.4}$$

where c_2 is the entropy coefficient that encourages exploration throughout training, and advantage estimates are computed using GAE with discount factor γ and trace-decay parameter λ.

Training is carried out in the VMAS simulation environment described below, parallelised

across N_{env} independent environment instances. At each iteration a fixed number of environment transitions is collected synchronously across all instances, constituting one batch of on-policy experience. The batch is then subdivided into minibatches and processed for a fixed number of gradient update epochs per iteration, following the standard PPO training loop, with parameters updated by the Adam optimiser at a fixed learning rate and gradient norms clipped to a maximum value. Training proceeds for a maximum number of iterations, with early stopping applied on the mean episode reward: if the mean episode reward does not improve over a fixed patience window, training is halted and the checkpoint achieving the highest mean episode reward is retained. The complete set of training hyperparameters is summarised in Table 4.3, and the training distribution covers all five target shapes described below simultaneously, with shapes drawn uniformly at each episode reset.

Low-Level Policy

The intra-option policy π_ω is a pre-trained *GoToXY* RL policy that drives the vehicle from its current position toward a world-frame target pose $(\mathbf{w}^{\mathcal{W}}, \psi)$, issuing linear and angular velocity commands at every simulation timestep within an option. It was trained independently of the present work in a separate VMAS pose-tracking environment, and posteriorly finetuned on a Gazebo environment, and remains frozen throughout all high-level training and evaluation reported in this thesis. The policy is a stochastic AC network operating on a six-dimensional observation built solely from the vehicle state relative to the target pose, namely the goal-frame relative position and velocity, the wrapped heading error, and the angular velocity; it therefore carries no information about the surveyed object and is agnostic to the boundary survey task. Its two-dimensional output is scaled by $[v^{\text{lin}}_{\max}, v^{\text{ang}}_{\max}]$ and applied to the differential-drive model.

This strict separation between the two levels, the high-level policy operating at option boundaries and the low-level policy operating at every timestep, ensures that the high-level policy only needs to learn where to direct the vehicle, while the low-level policy is responsible for how to reach each proposed waypoint under the vehicle's differential-drive dynamics and the environmental disturbance.

Sim2Sim Transfer

The boundary survey policy is trained entirely in simulation and is intended for deployment on the physical Nautilus ASV. As discussed in Section 2.1.5, transfer of simulation-trained policies is hindered by discrepancies between simulated and real dynamics, sensing, and actuation. Two complementary mechanisms narrow this gap, applied at the two levels of the hierarchy.

At the low level, the reality gap is addressed during training of the *GoToXY* policy. The simulated vehicle model is parameterised after the physical Nautilus: its mass, linear drag, and actuator gains and delays are set to match the real platform, so that the dynamics the low-level

policy learns to control approximate those encountered at deployment. As noted above, this low-level policy, together with its disturbance randomisation and Gazebo fine-tuning, was developed independently of the present work and is treated here as a fixed, pre-validated component.

At the high level, no additional transfer mechanism is applied: the boundary survey policy is trained in VMAS and deployed without further adaptation. This is justified by the abstraction at which it operates. The high-level policy reasons over a LiDAR-centric observation and selects waypoints at option boundaries, receiving rewards aggregated over entire options rather than precise per-timestep dynamics. Its decisions therefore depend on execution-level outcomes, namely whether a waypoint was reached, how far the vehicle advanced around the perimeter, and whether the object remained visible, which are substantially less sensitive to the low-level dynamics mismatch that drives the reality gap.

Training Environment

At each episode reset, a target object is randomly generated and placed in the environment. The object centre is positioned at the configured spawn origin, perturbed by a bounded uniform jitter in each axis. A base radius r is drawn uniformly from a configured range, so that object scale varies across resets, and the object boundary is represented by N ordered points $\{\mathbf{b}^{(i)}\}_{i=0}^{N-1}$, traversed counter-clockwise, each instantiated as a landmark of radius r_{pt}. The boundary points are obtained by uniform arc-length resampling along the generated shape. The object is additionally rotated by an angle drawn uniformly from $(-\pi, \pi)$ before placement, so that no repeated orientation is presented to the policy during training.

A shape is drawn uniformly at each reset from a set of five random generators, illustrated in Figure 4.3:

- **Circle.** An exact circle of radius r.
- **Rectangle.** A rectangle with a randomly drawn aspect ratio, inscribed within the base radius.
- **L-shape.** A rectangle with a rectangular cutout of randomly drawn depth, producing an asymmetric concave boundary.
- **Polygon.** A convex-ish polygon with a randomly drawn number of vertices, placed at jittered angular positions and with randomly scaled radii.
- **Abstract.** A closed curve defined by a radial profile with superimposed sinusoidal perturbations at harmonics $k \in \{2, 3, 5\}$:

$$\rho(\phi) = r \cdot \text{clip}\Big(1 + \sum_{k \in \{2,3,5\}} a_k \sin(k\phi + \psi_k),\ c_{\text{lo}},\ c_{\text{hi}}\Big), \tag{4.5}$$

  where the amplitudes a_k and phases ψ_k are sampled uniformly with the clipping limits $c_{\text{lo}}, c_{\text{hi}}$ bound the radial profile, producing smoothly varying closed curves.

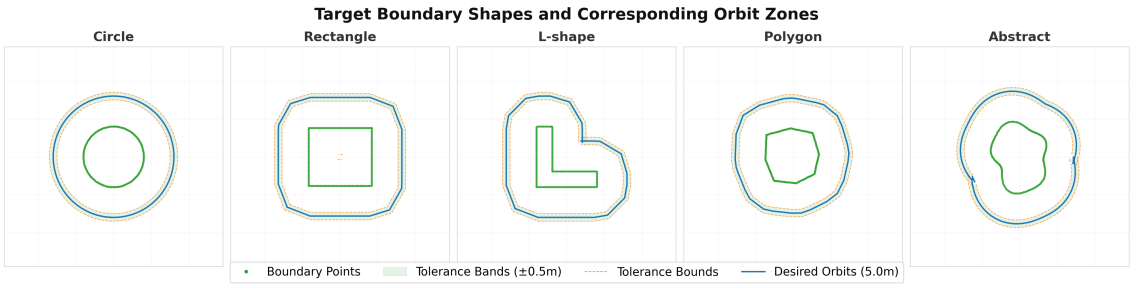


Figure 4.3: The five randomly generated target shapes used during training, and their corresponding desired orbits: circle, rectangle, L-shape, polygon, and abstract. All shapes are resampled to N boundary points by uniform arc-length sampling and rotated by a random angle at each reset.

The diversity of shape classes attempts to expose the policy to a broad range of local boundary curvatures and perimeter lengths, reducing the risk of overfitting to any single object geometry. At evaluation time, the same distribution is preserved, so that reported performance reflects generalisation over unseen object instances rather than memorisation of specific shapes.

At each reset, the vehicle is spawned relative to a randomly selected boundary point, displaced along that point's outward normal by a spawn distance equal to the desired standoff with the addition of a uniform jitter, clamped to a safe interval that prevents spawning in collision or beyond the effective LiDAR range:

$$\mathbf{p}_0 = \mathbf{b}^{(j)} + \rho_{\text{spawn}}\,\mathbf{n}^{(j)}, \qquad \rho_{\text{spawn}} = \text{clip}\big(d^\star + \xi,\ \rho_{\min},\ \rho_{\max}\big), \tag{4.6}$$

where $\mathbf{b}^{(j)}$ and $\mathbf{n}^{(j)}$ are the position and outward normal of the selected boundary point and ξ is the uniform spawn-distance jitter. The vehicle's initial heading is set tangentially, as the bearing of the outward normal offset by $\frac{\pi}{2}$ in the sense consistent with the object-on-left convention, and is then perturbed by a bounded heading jitter. This initialisation places the vehicle at a small offset from the correct operating distance from the object with a heading broadly consistent with the desired survey direction, reducing the proportion of each episode spent on initial approach behaviour and concentrating training on the boundary-following task itself.

Progress and Lap-Completion Computation

The lap-completion condition and the progress-reward term are computed in this implementation from the discretised boundary representation described above: the N ordered boundary points $\{\mathbf{b}^{(i)}\}_{i=0}^{N-1}$ provide a nearest-point index against which both signed perimeter displacement and full-revolution lap completion are evaluated.

A scalar accumulator $\Delta_{\text{cum}} \in \mathbb{R}$ tracks the signed sum of boundary displacements across consecutive steps throughout the episode. At each simulation step, the signed index displacement between the previously nearest boundary point and the currently nearest boundary point is com-

puted and added to this accumulator:

$$\Delta_{\text{cum}} \leftarrow \Delta_{\text{cum}} + \Delta i_t, \tag{4.7}$$

where Δi_t is the signed boundary-index displacement between the previously nearest and the currently nearest boundary point, computed with wrapping over the N boundary points. The lap termination condition is then

$$\text{done}_{\text{lap}} = |\Delta_{\text{cum}}| \geq N, \tag{4.8}$$

satisfied once the vehicle has traversed a cumulative displacement equal to the full perimeter of the target, in either direction.

The same indexing scheme underlies the progress reward. Progress is measured through the signed displacement of the nearest-boundary index between the start and the end of the option, where $i^{\star}_{\text{start}}$ and $i^{\star}_{\text{end}}$ denote the nearest boundary indices at option start and termination. The signed index displacement is computed with wrapping over the N boundary points,

$$\Delta_{\omega} = \left(\left(i^{\star}_{\text{end}} - i^{\star}_{\text{start}}\right) + \tfrac{N}{2}\right) \bmod N - \tfrac{N}{2}, \tag{4.9}$$

following the same wrapping convention as the lap accumulator of Equation 4.7, so that Δ_{ω} reflects the shortest-path direction around the perimeter rather than the absolute index difference. The progress reward is then

$$r_{\text{prog}} = w_{\text{prog}} \,\text{clip}\left(\frac{\Delta_{\omega}}{4}, -1, 1\right), \tag{4.10}$$

so that advancement in the desired direction yields a positive reward, backtracking yields a negative reward, and the contribution saturates once the option advances the equivalent of four boundary points in either direction, bounding the per-option influence of the term.

Implementation Details

The numerical values of all hyperparameters and environment constants referenced symbolically in the preceding sections are aggregated in the following tables: the high-level policy network and training hyperparameters are listed in Table 4.3, the reward weights and thresholds in Table 4.4, and the simulation and target-generation parameters in Table 4.5. Training spans N_{env} parallel environment instances, and the product of the frames per batch and maximum number of iterations gives a total of 2.4×10^6 environment transitions over a full training run.

Controlled Experiments: L-Shape Oversampling

The L-shape is the only class in the training distribution whose boundary is strictly concave, presenting a re-entrant corner that the orbiting behaviour must survey without cutting across the interior. The policy trained on the five classes sampled with equal probability was found to attain markedly lower success on the L-shape than on the convex classes, motivating a controlled

Table 4.3: High-level policy network and training hyperparameters.

Group	Parameter	Value
Network	Hidden layers (depth)	3
	Units per layer	128
	Activation	Tanh
PPO	Clip ε	0.2
	Entropy coefficient c_2	5×10^{-4}
	GAE γ	0.99
	GAE λ	0.9
Training	Parallel environments N_{env}	16
	Frames per batch	8,000
	Minibatch size	512
	Epochs per iteration	20
	Learning rate	3×10^{-4}
	Max gradient norm	1.0
	Max iterations	300
	Early stopping patience	20
Option	Option horizon H	15 steps
	Waypoint capture radius r_{wp}	0.25 m

Table 4.4: High-level policy reward weights and thresholds.

Group	Parameter	Value
Standoff	Desired standoff range $[d^{\star}_{\text{min}}, d^{\star}_{\text{max}}]$	[4.0, 7.0] m
	Standoff tolerance δ	0.5 m
	Waypoint weight w_{wp}	3.0
Progress	Progress weight w_{prog}	1.0
Visibility	Visibility weight w_{vis}	0.2
Penalty	Collision penalty κ_{col}	−10.0
	Inside-target penalty κ_{in}	−5.0
	Lost-target penalty κ_{lost}	−3.0

study of whether the deficit can be removed by adjusting the training distribution alone, leaving the network, reward, and optimisation unchanged.

A new policy, *oversampled*, is compared with the previous one. The *oversampled* policy augments the generator pool with two additional L-shape generators, yielding an effective pool of seven generators of which three are L-shapes, so that the L-shape is presented at each reset with probability $3/7$ rather than $1/5$. The two configurations are otherwise identical, including the network architecture, reward weights, optimisation hyperparameters, and total training budget. Per-class results for this new policy are reported in Section 4.3.

Table 4.5: Simulation environment and target-generation parameters.

Group	Parameter	Value
World	Integration timestep Δt	$0.2, \mathrm{s}$
	Substeps per step	1
	Current speed range	$[0.02, , 0.08]$
Vehicle	Length ℓ	$3.0, \mathrm{m}$
	Width w	$1.4, \mathrm{m}$
	Mass	130
	Linear drag	0.01
	Collision margin ϵ	$0.02, \mathrm{m}$
	Max linear velocity $v_{\max}^{\mathrm{lin}}$	$2.3, \mathrm{m/s}$
	Max angular velocity $v_{\max}^{\mathrm{ang}}$	$0.43, \mathrm{rad/s}$
LiDAR	Number of rays K	64
	Maximum range L	$10.0, \mathrm{m}$
Target	Boundary points N	150
	Boundary point radius r_{pt}	$0.25, \mathrm{m}$
	Base radius range	$[3.0, , 10.0], \mathrm{m}$
	Spawn-centre jitter	$\pm 1.0, \mathrm{m}$
	Rotation range	$(-\pi, , \pi)$
Spawn	Spawn-distance jitter ξ	$[-3.0, , 3.0], \mathrm{m}$
	Spawn-distance clamp $[\rho_{\min}, \rho_{\max}]$	$[3.0, , 7.0], \mathrm{m}$
	Heading jitter	$\pm\pi, \mathrm{rad}$
Rectangle	Aspect ratio	$\mathcal{U}(0.65, , 1.45)$
L-shape	Cut fraction	$\mathcal{U}(0.25, , 0.55)$
Polygon	Vertices	$\mathcal{U}5, \ldots, 9$
	Radial scale	$\mathcal{U}(0.70, , 1.20)$
Abstract	Harmonics k	$2, 3, 5$
	Amplitude a_k	$\mathcal{U}(-0.3, , 0.3)$
	Phase ψ_k	$\mathcal{U}(0, , 2\pi)$
	Clip limits $[c_{\mathrm{lo}}, c_{\mathrm{hi}}]$	$[0.55, , 1.45]$

Controlled Experiments: Reward Ablation

The option reward of Equation 3.12 combines three shaping terms, namely the waypoint standoff term r_{wp}, the perimeter-progress term r_{prog}, and the visibility term r_{vis}, alongside the terminal penalties. To quantify the contribution of each shaping term, three ablated variants are trained, each removing a single term by setting its weight to zero while retaining the terminal penalties, which encode the safety and task-termination semantics of the environment and are therefore held fixed:

- No standoff ($w_{\mathrm{wp}} = 0$): the policy receives no signal rewarding waypoints placed at the commanded distance from the boundary.

- No progress ($w_{\text{prog}} = 0$): the policy receives no signal rewarding monotonic advancement around the perimeter.
- No visibility ($w_{\text{vis}} = 0$): the policy receives no signal rewarding continued observation of the object within the LiDAR aperture.

Each variant is trained from scratch and evaluated against the full-reward policy, with the full-reward policy serving as the reference. Aggregate results for the different configurations are reported in Section 4.3.

4.1.3 Validation Scenarios

VMAS

The *SurveyTarget* scenario is implemented within the VMAS [4], a vectorized, GPU-accelerated, and PyTorch-based simulation framework designed for the development and evaluation of RL policies. VMAS supports the parallelisation of large numbers of independent environment instances, enabling efficient policy training through the simultaneous collection of experience across multiple rollouts. The scenario is configured as a single-agent environment, with one ASV operating per environment instance.

The simulated vehicle is a hydrodynamic box of length ℓ and width w, driven by a differential-drive dynamics model, and is approximated for collision detection by the fixed footprint polygon $\mathcal{H}$ introduced in Section 3.4.1. The high-level action is decoded into a body-frame waypoint, which is passed to the frozen low-level *GoToXY* policy. A spatially uniform current disturbance with randomised speed and direction is applied at each reset to emulate environmental forcing on the vehicle.

Gazebo

The survey skill is additionally evaluated in Gazebo [35], an open-source 3D robotics simulator that couples a rigid-body physics engine with sensor models and integrates natively with Robot Operating System (ROS). Relative to VMAS, Gazebo trades the large-scale environment parallelism used for policy training for higher-fidelity, single-instance simulation, making it well suited to validating a trained policy under more realistic dynamics before field deployment.

For the Gazebo evaluation of the survey skill alone, four fixed structures present at the test site are modelled: Durius, the floating inspection buoy described below, and three differently shaped docks, namely a rectangular one (16 m × 4 m), an L-shaped one (8 m × 8 m), and a T-shaped one (10 m × 8 m).

Field Deployment Site

The real-world field trials reported later in this chapter were conducted at the ATLANTIS Coastal Test Centre [55], in Viana do Castelo, Portugal, using the Nautilus ASV. The physical vehicle,

depicted in Figure 4.4, is a monohull differential surface vessel measuring 3.0 × 1.4 × 1.5 m (length × width × height) with a maximum velocity of 3.0 $\mathrm{m\,s^{-1}}$. It possesses a robust sensor suite for localization and perception, including a 3 Hz RTK-GPS module (0.01 m accuracy), a 20 Hz 9-DOF IMU for heading estimation (1.0° RMSE), and a 128-channel 3D LiDAR (10 Hz) with a 2.5 cm mean range error.

Onboard the Nautilus, the Action layer passes the clipped continuous control outputs of the RL navigation policies to the vehicle's hardware interface via ROS, which provides the communication infrastructure between the high-level autonomy stack and the low-level motor controllers.

The inspection target itself, Durius, is a near-circular repurposed oil & gas floating buoy (16 m diameter). The real-world field trials reported in Section 4.5 were carried out under mild sea-state conditions, with wind speeds ranging from approximately 1 $\mathrm{m\,s^{-1}}$ to 2 $\mathrm{m\,s^{-1}}$ and gusts reaching 4 $\mathrm{m\,s^{-1}}$ to 6 $\mathrm{m\,s^{-1}}$, and a significant wave height of 1.2 m.

4.2 Agentic Mission Planner - Simulation Results

This section reports the results obtained with the models, training procedure, and evaluation setup described in Section 4.1.1: the comparative model benchmarks (Section 4.2.1), a qualitative robustness analysis of the selected Mission Agent (Section 4.2.2), and the two end-to-end missions executed in Gazebo simulation (Section 4.2.3).

Figure 4.4: The Nautilus ASV navigating in the ATLANTIS Test Centre, in Viana do Castelo, Portugal.

4.2.1 Model Benchmarks

Mission-to-Tasks Benchmark

The Mission-to-Tasks benchmark results are presented in Table 4.6. Without fine-tuning, all candidate models exhibit poor performance across all metrics. Non-instruction base models, specifically Llama-3.2-1B, Llama-3.2-3B, and Llama-3.1-8B, generate outputs that are almost entirely unparseable, yielding 0% schema validity and hallucination rates of 100%, confirming that these models do not follow structured output instructions without task-specific adaptation. Instruction-tuned variants produce marginally better results in zero-shot settings: Llama-3.2-3B-Instruct reaches an F1 of 0.61 at baseline, and Qwen2.5-3B-Instruct achieves 0.54, yet neither base configuration achieves reliable schema-valid output production or consistent vocabulary grounding.

LoRA fine-tuning produces substantial and consistent improvements across all models and all metrics. Schema validity reaches 98–100% across all fine-tuned configurations, and hallucination rates fall to near zero, confirming that the adapters successfully constrain the output distribution to the reference task vocabulary. This pattern holds uniformly across model scales and families, demonstrating that the benefit of fine-tuning is not contingent on model capacity but rather on task-specific adaptation.

Among fine-tuned models, the F1 scores range from 0.68 for Llama-3.2-1B to 0.89 for Llama-3.2-3B-Instruct, with the 4-bit quantised Llama-3.1-8B reaching a comparable 0.88, as illustrated in Figure 4.5. The instruction-tuned Qwen2.5-3B-Instruct is a notable exception to the general trend: its fine-tuned F1 of 0.70 represents a degradation relative to Qwen2.5-3B (0.84), and its hallucination rate rises to 32% after fine-tuning, suggesting that the instruction-tuned Qwen2.5 variant is less amenable to the supervised fine-tuning regime applied here, possibly due to conflicts between its instruction-following alignment and the constrained output distribution imposed by the LoRA adapter. This result underscores that instruction tuning and supervised fine-tuning are not always complementary, and that the interaction between the two must be verified empirically for each model family.

Latency is a less critical consideration at the Mission level, as the Mission Agent is invoked

Table 4.6: Mission-to-Tasks benchmark results (base prompt engineering vs. LoRA fine-tuned). $n = 25$ samples.

Model	Task F1		Hallucination (%)		Schema Valid (%)		Latency (s)		Tokens	
	Base	LoRA	Base	LoRA	Base	LoRA	Base	LoRA	Base	LoRA
Llama 3.2 1B	0.00	0.68	100.0	4.0	0.0	98.0	0.1	11.2	1.0	486.4
Llama 3.2 1B Inst.	0.52	0.74	0.0	0.0	100.0	100.0	3.4	12.5	163.0	531.4
Llama 3.2 3B	0.00	0.80	100.0	8.0	0.0	100.0	0.2	17.6	1.0	436.8
Llama 3.2 3B Inst.	0.61	0.89	0.0	4.0	100.0	100.0	6.6	18.7	198.4	472.0
Qwen2.5 3B	0.52	0.84	0.0	4.0	100.0	100.0	7.1	25.5	163.0	501.1
Qwen2.5 3B Inst.	0.54	0.70	0.0	32.0	100.0	100.0	7.5	26.4	174.6	504.6
Llama 3.1 8B (4-bit)	0.00	0.88	100.0	0.0	0.0	100.0	0.3	23.1	1.0	496.1

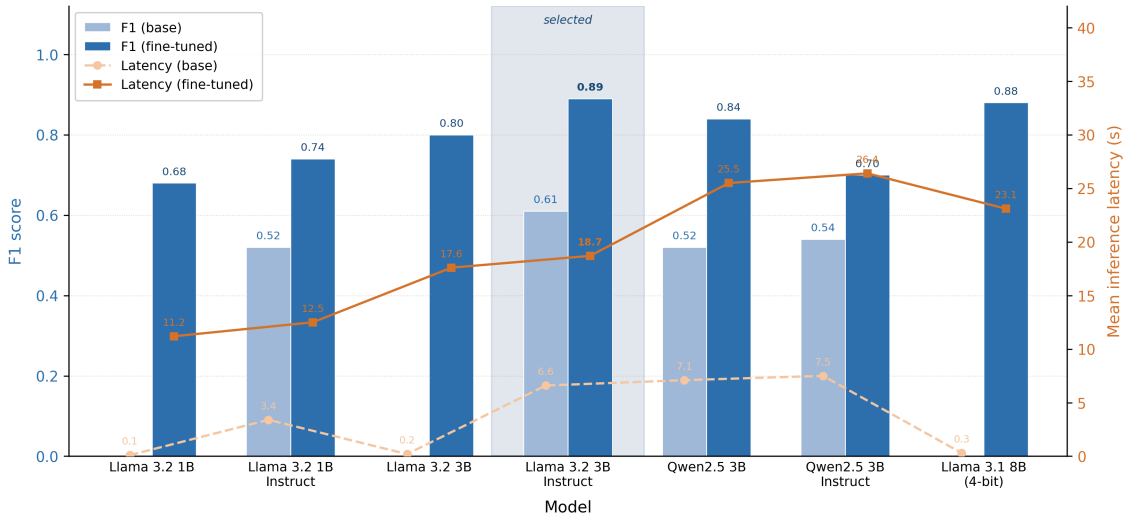


Figure 4.5: Comparison of fine-tuned F1 scores (bars, left axis) and mean inference latency (lines, right axis) across all Mission Agent candidate models evaluated in base and LoRA fine-tuned configurations. The selected model, Llama-3.2-3B-Instruct, is highlighted.

only once per mission command, but its inference time constitutes the first component of the total planning latency experienced by the operator. As shown in Figure 4.5, the latency gap between fine-tuned models is substantial: Llama-3.2-1B achieves the lowest latency at 11.2 s but also the lowest F1 (0.68), while the 4-bit Llama-3.1-8B incurs a mean latency of 23.1 s, making it less suitable for the onboard compute budget despite its competitive decomposition performance. Llama-3.2-3B-Instruct offers the best balance of decomposition quality (F1 = 0.89), schema reliability (100% validity), and a better inference latency of 18.7 s, being accordingly selected as the Mission Agent.

Task-to-Skills Benchmark

The Task-to-Skills benchmark results are presented in Table 4.7. The pattern of baseline performance is broadly consistent with the Mission-to-Tasks results, with non-instruction base models producing empty or malformed outputs and yielding zero F1 and zero schema validity across all configurations. Instruction-tuned base models show a more marked partial capability at this level relative to the Mission benchmark: Llama-3.2-3B-Instruct achieves a base F1 of 0.67, and Qwen2.5-3B-Instruct reaches 0.50, suggesting that the task-to-skills mapping has a more structured and constrained nature that partially aligns with the priors of instruction-tuned models even without fine-tuning. Nevertheless, base performance remains insufficient for reliable deployment, with schema validity and hallucination rates remaining inconsistent.

LoRA fine-tuning again produces a decisive and uniform improvement. All fine-tuned models converge to high F1 scores in the range 0.95–0.99, alongside zero hallucination and near-universal schema validity. This convergence across model scales indicates that the task-to-skills mapping is learnable across a wide range of model capacities once sufficient supervised signal is provided,

Table 4.7: Task-to-Skills benchmark results (base prompt engineering vs. LoRA fine-tuned). n = 500 samples.

Model	Skill F1		Hallucination (%)		Schema Valid (%)		Latency (s)		Tokens	
	Base	LoRA	Base	LoRA	Base	LoRA	Base	LoRA	Base	LoRA
Llama 3.2 1B	0.00	0.97	0.0	0.0	0.0	98.0	0.1	5.4	1.0	239.3
Llama 3.2 1B Inst.	0.25	0.97	0.6	0.0	43.6	98.0	10.3	5.5	619.5	239.3
Llama 3.2 3B	0.00	0.99	0.0	0.0	0.0	100.0	0.2	9.2	1.0	232.0
Llama 3.2 3B Inst.	0.67	0.99	11.6	0.0	100.0	100.0	5.7	9.2	195.1	237.2
Qwen2.5 3B	0.47	0.99	0.0	0.0	99.8	100.0	10.4	11.9	285.1	234.8
Qwen2.5 3B Inst.	0.50	0.95	0.0	0.0	100.0	95.2	5.5	13.0	145.3	251.7
Llama 3.1 8B (4-bit)	0.00	0.99	0.0	0.0	0.0	100.0	0.4	10.3	1.0	232.0

and that model capacity is not the primary bottleneck at this abstraction level. The constrained vocabulary of named skills with well-defined semantics reduces the linguistic complexity of the decomposition problem relative to free-form mission parsing, making the task-level mapping more tractable for smaller models.

The Qwen2.5-3B-Instruct model is again an outlier: despite fine-tuning, it attains the lowest fine-tuned F1 of 0.95 among all candidates, and its schema validity drops to 95.2%, in contrast to the 100% validity achieved by the Llama-3.2-3B and Llama-3.2-3B-Instruct models. This suggests a residual tendency to deviate from the prescribed output format despite fine-tuning, which would require additional validation logic at the skill execution boundary and is undesirable in a safety-critical deployment context.

Given the convergence in decomposition quality across fine-tuned models, the critical differentiating factor at the Task level is inference latency. The Task Agent is invoked once per task in the decomposed mission sequence, meaning that its latency compounds across multi-task missions and directly determines the total planning time presented to the operator before any execution is initiated. As shown in Figure 4.6, this compounding effect grows linearly with mission length and produces increasingly large planning time differences between candidates as m increases. Llama-3.2-3B and Llama-3.2-3B-Instruct achieve the highest fine-tuned F1 among Llama variants (0.99) but requires a mean inference time of 9.2 s per task. Both Llama-3.2-1B and Llama-3.2-1B-Instruct achieve equivalent F1 scores of 0.97 at mean latencies of 5.4 s and 5.5 s respectively, representing approximately half the inference time of the 3B models. Between the two 1B candidates, Llama-3.2-1B-Instruct is preferred: the marginal latency difference of 0.1 s per task is negligible in practice, as visible in Figure 4.6, while the instruction-tuned variant provides greater consistency in structured output compliance across prompting variations, aligning with the same model family used at the Mission level and simplifying deployment. This model is, therefore, selected as the Task Agent.

Summary

Taken together, these results establish two principal conclusions. First, supervised fine-tuning via LoRA is a necessary condition for reliable structured decomposition at both abstraction lev-

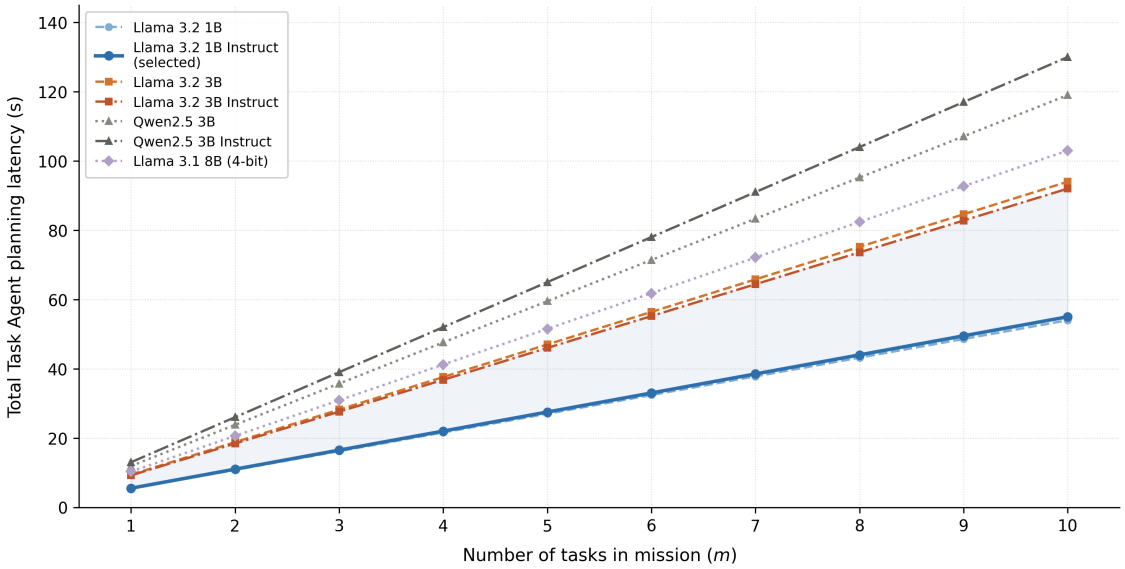


Figure 4.6: Total Task Agent planning latency as a function of the number of tasks in a decomposed mission, for all fine-tuned candidate models. The shaded region highlights the compounding latency saving of the selected Llama-3.2-1B-Instruct model relative to the Llama-3.2-3B-Instruct candidate, amounting to a saving of 37 s at $m = 10$ tasks. The latency value used for each model is the average latency over all test samples.

els: base models, including instruction-tuned variants, cannot consistently produce schema-valid, vocabulary-grounded outputs without task-specific adaptation, irrespective of model scale. Second, once fine-tuned, compact models are capable of matching the decomposition performance of larger counterparts on structured tasks, enabling the deployment of a capable two-agent planning hierarchy entirely within the onboard hardware budget of the vehicle. The selected agents, Llama-3.2-3B-Instruct at the Mission level and Llama-3.2-1B-Instruct at the Task level, achieve F1 scores of 0.89 and 0.97 respectively, with near-zero hallucination rates and schema validity of 100% and 98%, while remaining within the inference latency constraints imposed by the onboard compute hardware.

4.2.2 Robustness Analysis

This analysis reveals that the Mission Agent's generalisation limitations are confined and easy to detect: all 9 incorrect outputs produced schema-valid JSON containing exclusively in-vocabulary task names, with the single exception discussed below. This finding confirms that fine-tuning successfully constrains the output distribution to the reference task vocabulary, such that the model degrades within the schema rather than outside it. As a consequence, all failures remain detectable and operator-reviewable through the mission control interface before any execution is initiated, preserving the safety guarantee provided by the operator review gate.

The four failure types identified, along with their frequency and representative prompts, are summarised in Table 4.8. The following subsections analyse each failure mode in detail.

Table 4.8: Failure Types and Representative Prompts.

Failure Type	Count	Schema Valid	Representative Prompt
Prerequisite omission	4	✓	"I need a full 360-degree inspection of the offshore wind turbine at a 15-meter stand-off. Make sure to run a system diagnostic before you start."
Parameter propagation	2	✓	"Head out to [45.0, 10.0] to check out the crashed helicopter. Give it a full systematic survey, then come home."
Task substitution	2	✓	"Find the rogue shipping container drifting nearby, capture thermal footage for 4 minutes, and summarize what you see."
Hallucination	1	✓	"Sweep the boat landing area for any issues, then dock at the north edge of the landing."

Prerequisite Omission

Prerequisite omission is the most frequent failure mode, accounting for 4 of the 9 incorrect predictions. This failure occurs when the Mission Agent correctly identifies the primary task implied by an operator command but omits an implicit upstream task that must precede it for the plan to be executable. In the context of the current task vocabulary, the most common manifestation of this failure involves the *SearchAndDetect* task: when a mission references a target using informal or descriptive language rather than explicit coordinates, a *SearchAndDetect* step is required to localise the target at runtime before any navigation task can be meaningfully parametrised. When this prerequisite is omitted, the immediately following *Transit* or positioning task receives a placeholder coordinate of $[-99, -99]$, which is the sentinel value used by the architecture to indicate an unresolved location. This placeholder is immediately apparent to an operator during plan review, as it signals that a required localisation step is absent.

A representative instance of this failure is provided by the prompt *"I need a full 360-degree inspection of the offshore wind turbine at a 15-meter stand-off. Make sure to run a system diagnostic before you start."* The expected decomposition for this command is *Diagnostics* → *SearchAndDetect* → *Transit* → *PerformOrbit*, since the wind turbine is referenced descriptively rather than by coordinate and must therefore be located before the vehicle can be directed towards it. The Mission Agent produces instead *Diagnostics* → *Transit* → *PerformOrbit*, bypassing the *SearchAndDetect* task and directly issuing a *Transit* with the unresolved $[-99, -99]$ coordinate.

Analysis of the cases in which this failure occurs relative to those in which it does not reveals that the model's behaviour is sensitive to surface-level phrasing of the target description rather than reflecting a systematic inability to reason about localisation prerequisites. The Mission Agent correctly includes *SearchAndDetect* in the majority of analogous cases encountered in the test set, suggesting that the failure is triggered by specific lexical or syntactic cues in the target description, such as generic category terms (e.g., "offshore wind turbine") as opposed to contextual references that more strongly imply the need for prior detection.

Parameter Propagation Errors

Parameter propagation errors account for 2 of the 9 incorrect predictions and represent a qualitatively distinct failure from prerequisite omission. In these cases, the task sequence itself is correctly identified, but contextual information provided in the operator prompt, most commonly coordinate data or numerical parameters, is not carried forward consistently across all tasks that require it.

In one instance, the operator provides explicit coordinates $[45.0, 10.0]$ for a known target and requests a systematic survey followed by a return to base. The Mission Agent correctly identifies the task sequence *Transit* $\rightarrow$ *SurveyTarget* $\rightarrow$ *Transit*, but inserts a spurious intermediate *Transit* step parametrised with $[-99, -99]$ and fails to propagate the known coordinates $[45.0, 10.0]$ to the *SurveyTarget* task. The structural decomposition is therefore sound, and the task names are all correct, yet the parameter bindings across task boundaries are inconsistent.

In the second instance, a partial arc orbit specified as West-to-South in the operator prompt is correctly mapped to a *PerformOrbit* task, but the sector parameters defining the arc extent are not preserved in the output, defaulting instead to a full 360° orbit. As with the previous case, task identity resolution succeeds while parameter propagation fails.

This failure mode indicates that the Mission Agent's difficulty lies not in understanding the semantic intent of the command but in maintaining contextual bindings across multiple output fields.

Task Substitution

Task substitution occurs in 2 of the 9 incorrect predictions and represents cases where the Mission Agent correctly interprets the operational intent of the command but maps a sub-component of it to a semantically related yet incorrect task from the vocabulary.

In the representative case identified, the sub-prompt *"capture thermal footage for 4 minutes"* is mapped to *KeepStation* rather than *ThermalTimeLapse*. The model correctly resolves the temporal and positional behaviour implied by the instruction, generating a 4-minute duration parameter, but does not connect the thermal modality to its dedicated task. This suggests that the model's representation of the command conflates the stationary behaviour required for thermal data acquisition with the station-keeping task itself, failing to propagate the sensor modality information to the task identity selection. The duration parameter is correctly extracted in both observed cases of this failure type, reinforcing the interpretation that the failure is specific to task identity resolution under sensor modality descriptions rather than a general failure of semantic understanding.

From a vocabulary design perspective, this failure highlights a representational proximity between tasks that share positional behaviour but differ in their sensor requirements. Tasks such as *KeepStation*, *ThermalTimeLapse*, and *FeatureStare* all involve a degree of positional stationarity, and their disambiguation requires the model to attend to sensor-level details in the command that are secondary to the spatial structure of the task. This behaviour may improve with a larger and

more diverse training dataset that provides more examples of sensor-specific task disambiguation, or through the addition of task descriptions to the prompt context at inference time.

At the execution level, task substitution is the most operationally transparent of the failure modes identified. No downstream parameter failures are associated with this failure type, as the incorrect task is otherwise well-formed and fully parametrised.

Hallucination

Hallucination is the rarest failure mode, appearing in a single case across the 25-sample test set. In this instance, the model processes a prompt requesting a docking manoeuvre and produces a *Berthing* task name, which is absent from the reference vocabulary, alongside an otherwise correct *BoatLandingClearanceSweep* → *Transit* sequence. The output remains schema-valid, and the invented name is immediately detectable by a vocabulary check at the task validation boundary, requiring no semantic interpretation to identify. This is the only output across all 9 incorrect predictions that contains an out-of-vocabulary term.

The hallucinated instance, *Berthing*, is semantically proximate to *DockingApproach*, the in-vocabulary task that represents a GPS-based approach to a dock or berth. This proximity suggests that the hallucination arises from a failure to retrieve the exact term rather than a complete hallucination from the task domain.

The scarcity of hallucination errors, a single instance across 25 out-of-distribution prompts, constitutes strong evidence that LoRA fine-tuning effectively binds the Mission Agent to the predefined task vocabulary even when processing novel or unconventional instructions.

4.2.3 End-to-End Simulation

The setup of the two representative missions evaluated below, including the operator prompts and the Mission and Task Agent outputs they elicited, is described in Section 4.1.1.

Mission 1: Bathymetric Survey

The resulting trajectory for Mission 1 is shown in Figure 4.7. The trajectory shows an initial *StationKeep* for search and detection of the area, followed by several *GoToXYs* creating a lawnmower pattern.

Across twenty independent simulation runs, the vehicle successfully followed the planned lawnmower pattern across the survey area, with all ten waypoint termination conditions met within the prescribed one-metre tolerance and no skill-level failures recorded in any run. Quantitatively, the mission completed, on average, in 2919 steps (std = 5), corresponding to 583.7 s of simulated execution time (std = 1.1 s), with the RL navigation policy achieving a mean waypoint tracking error of 0.971 m (std = 0.03 m) across all ten *GoToXY* waypoints and a mean station keeping error of 0.117 m (std = 0.003 m) during the localisation phase. The RL navigation policy produced smooth transitions between successive waypoints throughout the survey.

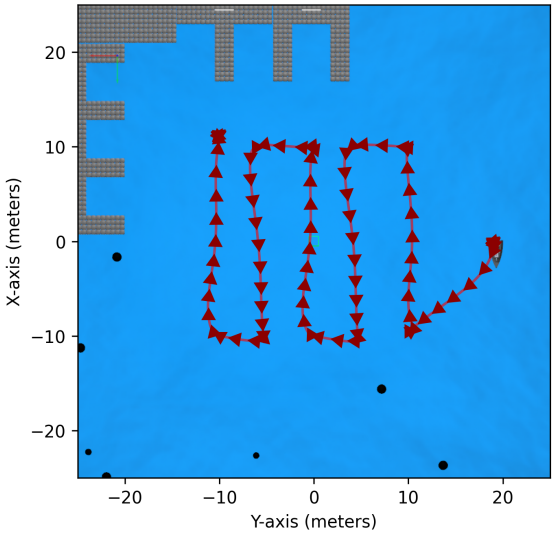


Figure 4.7: Simulated trajectory for Mission 1: Area survey, demonstrating the ASV successfully executing a continuous grid search pattern.

These results validate the full top-down translation for a long-horizon, area-coverage mission: from a single free-form operator command, the Mission Agent correctly inferred the need for a prior localisation step, the Task Agent expanded the survey task into a geometrically consistent waypoint sequence with appropriate concurrent perception, and the RL policy executed the resulting plan.

Mission 2: Target Inspection

The resulting trajectory for Mission 2 is shown in Figure 4.8. Like in the previous mission, this trajectory shows an initial *StationKeep* for search and detection of the target, followed by *GoToXYs* for surveying it. Afterwards, the vehicle keeps station again for information summarization, and finally goes back to the dock.

Across twenty independent simulation runs, the vehicle successfully completed the full four-task inspection sequence in every trial, with all waypoint termination conditions met within the prescribed one-metre tolerance and no skill-level failures recorded in any run. Quantitatively, the full pipeline completed, on average, in 838 steps (std = 42), corresponding to 167.6 s of simulated execution time (std = 8.4 s), with the RL navigation policy achieving a mean waypoint tracking error of 0.932 m (std = 0.05 m) across all nine *GoToXY* waypoints and a mean station keeping error of 0.425 m (std = 0.027 m) during the search and detect and report generation phases.

These results validate the framework across a more compositionally demanding scenario than Mission 1. The Mission Agent successfully resolved implicit localisation requirements, interpreted domain-specific terminology, and produced a four-task decomposition covering navigation, perception, data management, and return behaviours from a single short operator command. The Task Agent correctly selected and parametrised concurrent perception and management skills

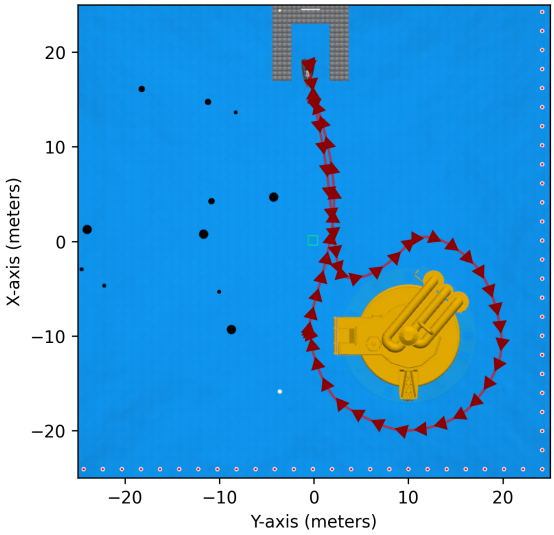


Figure 4.8: Simulated trajectory for Mission 2: Target-centric inspection, demonstrating the ASV establishing an orbit around the target.

alongside the primary navigation skill at each step, populating a plan of sufficient specificity for direct execution. The RL navigation policy then executed the resulting waypoint sequence, confirming that the semantic-to-physical translation pipeline operates correctly across the full abstraction hierarchy in simulation.

Summary

Across both simulation missions, the end-to-end pipeline consistently translated free-form natural language operator commands into structured, executable plans and subsequently executed those plans across twenty independent runs per mission. The Mission Agent correctly decomposed both objectives, including the inference of implicit prerequisite tasks and the interpretation of domain-specific references, while the Task Agent produced compositionally correct skill sequences with appropriate concurrent perception and management behaviours at each step. The RL navigation policies provided reliable execution of the generated waypoint sequences, with mean tracking errors of 0.971 m and 0.932 m for the survey and inspection missions respectively, confirming the viability of the proposed architecture as a complete semantic-to-physical pipeline prior to field deployment.

4.3 Hierarchical Survey Skill - Simulation Results

This section reports the results obtained with the policy, reward, and simulation environment described in Section 4.1.2.

4.3.1 VMAS Simulation Results

Overall Performance

Aggregated over the 100 evaluation episodes, the policy trained on the five shape classes under uniform sampling completes a full orbit in 87.0% of episodes, with the remaining 13.0% terminating in collision and no episode terminating through loss of target. The absence of loss-of-target terminations indicates that the visibility shaping is sufficient to keep the object within the LiDAR aperture throughout the survey. The time-averaged standoff error over successful episodes is 0.86 m, which, relative to the commanded standoff range of 4.0 m to 7.0 m, corresponds to tracking the orbit radius to within roughly a fifth of the commanded distance.

The per-class breakdown in Table 4.9 shows that this aggregate figure conceals a strongly bimodal outcome across the shape distribution. Four of the five classes, namely the circle, rectangle, polygon, and abstract shapes, are surveyed with a 100% success rate and no collisions, with standoff errors below one metre in every case. The L-shape is the sole exception: its success rate falls to 35.0% and its collision rate rises to 65.0%, accounting for the entirety of the aggregate collision count. The reduced mean episode length on the L-shape relative to the convex classes is consistent with collisions terminating those episodes early, before a full orbit is completed.

The deficit is attributable to the concave geometry of the L-shape. Its re-entrant corner requires the policy to follow a boundary that folds inward, a configuration absent from the four convex classes, and the policy trained under uniform sampling encounters it in only one episode in five.

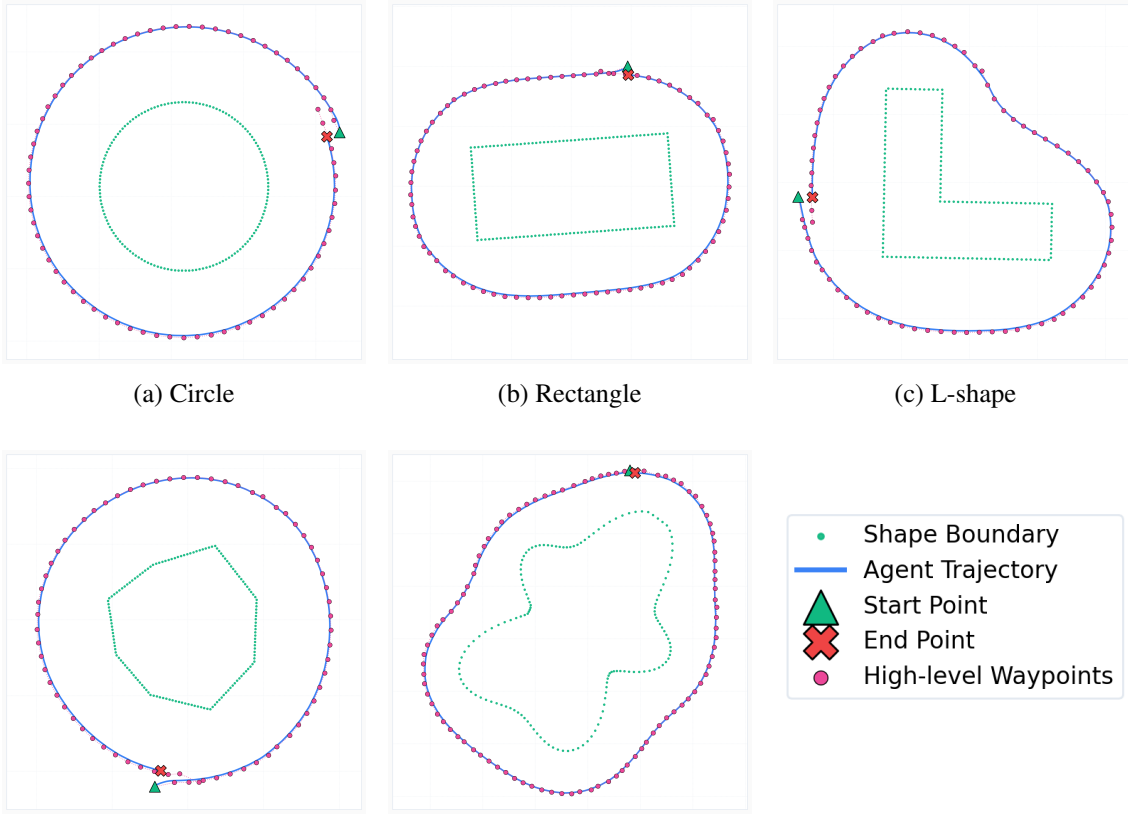
Trajectory Analysis

To characterise the qualitative behaviour underlying the aggregate metrics, Figure 4.9 shows one representative successful episode for each of the five shape classes, with their complete lap trajectory.

For the convex classes the realised orbit closely mirrors the object geometry. On the circle the trajectory is a concentric loop of approximately constant radius, with the high-level waypoints distributed evenly around the perimeter. The executed path tracks the commanded waypoint sequence almost exactly, indicating that the frozen low-level controller follows each waypoint with

Table 4.9: Per-class and overall performance of the uniform-sampling policy over 100 evaluation episodes (20 per shape class). Standoff error and episode length are reported as mean (standard deviation); rates are percentages of episodes.

Shape	Success (%)	Collision (%)	Loss of Target (%)	Standoff Error (m)	Ep. length (steps)
Circle	100.0	0.0	0.0	0.80 (0.37)	644.6 (100.7)
Rectangle	100.0	0.0	0.0	0.83 (0.27)	746.3 (135.7)
L-shape	35.0	65.0	0.0	1.10 (0.38)	357.4 (265.6)
Polygon	100.0	0.0	0.0	0.75 (0.30)	614.7 (102.5)
Abstract	100.0	0.0	0.0	0.97 (0.48)	667.6 (110.3)
Overall	**87.0**	**13.0**	**0.0**	**0.86 (0.37)**	**606.1 (202.3)**



(a) Circle (b) Rectangle (c) L-shape

(d) Polygon (e) Abstract (f) Legend

Figure 4.9: Representative successful survey trajectories for the five shape classes. The dotted curve is the object boundary, the solid line the executed vehicle trajectory, the circular markers the high-level waypoints issued at option boundaries, and the triangle and cross the start and end of the episode. The coincidence of start and end in every panel confirms a completed orbit.

little residual error. On the rectangle and the polygon the orbit rounds the convex corners into smooth arcs rather than reproducing them as sharp angles, a direct consequence of maintaining a fixed standoff around a convex vertex.

The abstract and L-shape classes expose the behaviour that distinguishes boundary survey on non-convex geometry. On the abstract shape the executed trajectory is markedly smoother than the boundary itself: rather than following the inward lobes of the radial profile, the policy orbits the outer envelope of the object, so that the distance to the nearest boundary point grows as the vehicle passes each concavity. The L-shape shows the same effect at its sharpest. The re-entrant corner of the boundary is not entered; instead the trajectory traces a convex loop that bulges around the concavity, with the vehicle surveying the notch from its outer extent rather than driving into it. This following behaviour keeps the vehicle clear of the boundary on concave geometry but increases the standoff to the nearest boundary point. On the L-shape, when the agent abandons this more conservative behaviour to try to match the desired standoff ditance, it ends up colliding.

L-Shape Generalisation

On the concave class, whose results are detailed in Table 4.10 the success rate rises from 35.0% under uniform sampling to 80.0% under oversampling, while the collision rate falls from 65.0% to 10.0%. The improvement carries through to the aggregate: overall success increases from 87.0% to 92.0% and the overall collision rate falls from 13.0% to 2.0%, since collisions under the uniform policy were concentrated almost entirely on the L-shape. Presenting the concave geometry more frequently during training therefore allows a single policy, with the network, reward, and optimisation held fixed, to survey the re-entrant corner that the uniform policy could not.

The improvement is not without cost, and two regressions are visible across the remaining classes. First, the time-averaged standoff error roughly doubles under oversampling, rising from 0.86 m to 1.59 m in aggregate and increasing on every class, including the convex ones on which the uniform policy was already perfect. Second, a loss-of-target failure mode appears that was absent under the uniform policy: the overall loss-of-target rate rises from 0.0% to 6.0%, with each convex class incurring a 5.0% rate and the L-shape a 10.0% rate. The four convex classes consequently slip from a 100% success rate to 95%, the lost five per cent being redistributed into the new loss-of-target terminations rather than into collisions.

These trade-offs are consistent with the change in surveying behaviour induced by the concave class. Negotiating the re-entrant corner without collision favours a more conservative orbit that stands further off from the boundary, and this larger-standoff behaviour generalises to all shapes rather than being confined to the L-shape, raising the standoff error uniformly. The same conservatism occasionally pushes the vehicle toward the edge of the effective LiDAR range at a concavity, where the object can momentarily leave the sensor aperture and trigger the loss-of-target condition, accounting for the new failure mode. For the boundary survey task, in which a collision-free, completed orbit is the primary success criterion, the exchange of a 65% collision rate on the concave class for a modest reduction in standoff precision and a small loss-of-target rate is favourable. The degraded standoff accuracy nonetheless remains a limitation of the oversampling approach: because the more conservative orbit is learned globally rather than applied selectively to concave geometry, the policy sacrifices standoff precision on the convex classes where it was not required. This suggests that conditioning the surveying margin on the local

Table 4.10: Per-class and overall performance of the oversampled policy over 100 evaluation episodes (20 per shape class). Success, collision, and loss-of-target are percentages of episodes; standoff error is the mean absolute deviation from the commanded standoff, in metres.

Shape	Success (%)	Collision (%)	Loss of Target (%)	Standoff Error (m)
Circle	95.0	0.0	5.0	1.53
Rectangle	95.0	0.0	5.0	1.38
L-shape	80.0	10.0	10.0	1.60
Polygon	95.0	0.0	5.0	1.82
Abstract	95.0	0.0	5.0	1.61
Overall	**92.0**	**2.0**	**6.0**	**1.59**

boundary curvature, rather than raising the standoff uniformly, could recover the lost accuracy while retaining the improved handling of the re-entrant corner.

Reward Ablation

Aggregate results for the different ablation configurations are reported in Table 4.11.

Removing the progress term has a considerable effect on the success rate. The no-progress policy completes no orbit on any shape, its success rate falling to 0.0%, and 96.0% of episodes terminate through loss of target after a mean of only 77.8 steps, an order of magnitude shorter than the full-reward policy. The progress term is the only component of the reward that compensates advancement around the perimeter; the standoff and visibility terms reward, respectively, distance from the boundary and continued observation, neither of which requires the vehicle to circumnavigate the object. With the progress incentive removed, the policy has no reason to advance and collapses to a degenerate behaviour that fails to orbit, losing the object within the first few options. The progress term is therefore the load-bearing component of the reward, necessary for any successful survey to emerge.

Eliminating the standoff term induces a heavily different behaviour. The no-standoff policy actually completes more orbits than the full-reward policy, raising the success rate to 98.0% and lowering the collision rate to 2.0%, but its time-averaged standoff error rises to 2.24 m, nearly three times that of the full-reward policy. Deprived of any incentive to hold the commanded distance, the policy defaults to a wide, conservative orbit that completes the lap reliably and safely while disregarding the standoff specification entirely. Thus, the standoff term does not drive task completion, which the policy achieves more readily without it, but is essential to surveying at the commanded distance, the distinguishing requirement of the task. The shorter mean episode length of the no-standoff policy is consistent with this less constrained orbit, unconcerned with delicate and shape accurate manouvers.

Dropping the visibility term reintroduces the loss-of-target failure mode while leaving the standoff broadly intact. The no-visibility policy maintains a standoff error of 1.09 m, comparable to the full-reward policy, and orbits at a similar episode length, but its loss-of-target rate rises from 0.0% to 20.0%, and this lost fraction depresses the success rate from 87.0% to 77.0%. The visibility term contributes little to either circumnavigation or standoff accuracy; its specific function

Table 4.11: Aggregate performance of the full reward and the three reward ablations over 100 evaluation episodes. Success, collision, and loss-of-target are percentages of episodes; standoff error and episode length are reported as mean (standard deviation). The no-progress policy completes no successful episode, so its standoff error is undefined.

Configuration	Success (%)	Collision (%)	Loss of Target (%)	Standoff Error (m)	Ep. length (steps)
Full reward	87.0	13.0	0.0	0.86 (0.37)	606.1 (202.3)
No standoff	98.0	2.0	0.0	2.24 (0.23)	310.3 (50.8)
No progress	0.0	4.0	96.0	–	77.8 (22.4)
No visibility	77.0	1.0	20.0	1.09 (0.46)	648.7 (209.3)

is to keep the object within the LiDAR aperture, and its removal allows the object to leave the field of view on roughly one episode in six, exactly the failure mode it was designed to suppress.

Taken together, the ablation shows that the three shaping terms are complementary and largely non-overlapping in function: the progress term produces the orbit, the standoff term sets its radius, and the visibility term protects sensor coverage. Only the progress term is strictly necessary for a successful survey, while the standoff and visibility terms refine the quality and safety of the orbit rather than enabling it. The full reward is the only configuration that satisfies both survey-specific requirements at once: the no-standoff policy completes more orbits but abandons the commanded distance, and the no-visibility policy holds the distance but loses the object on one episode in six, whereas the full reward attains the lowest standoff error of any orbit-completing configuration while keeping the loss-of-target rate at zero. It therefore does not maximise the raw success rate, trading a margin of completion for the accurate standoff and maintained visibility that together define a useful survey.

4.3.2 Gazebo Simulation Results

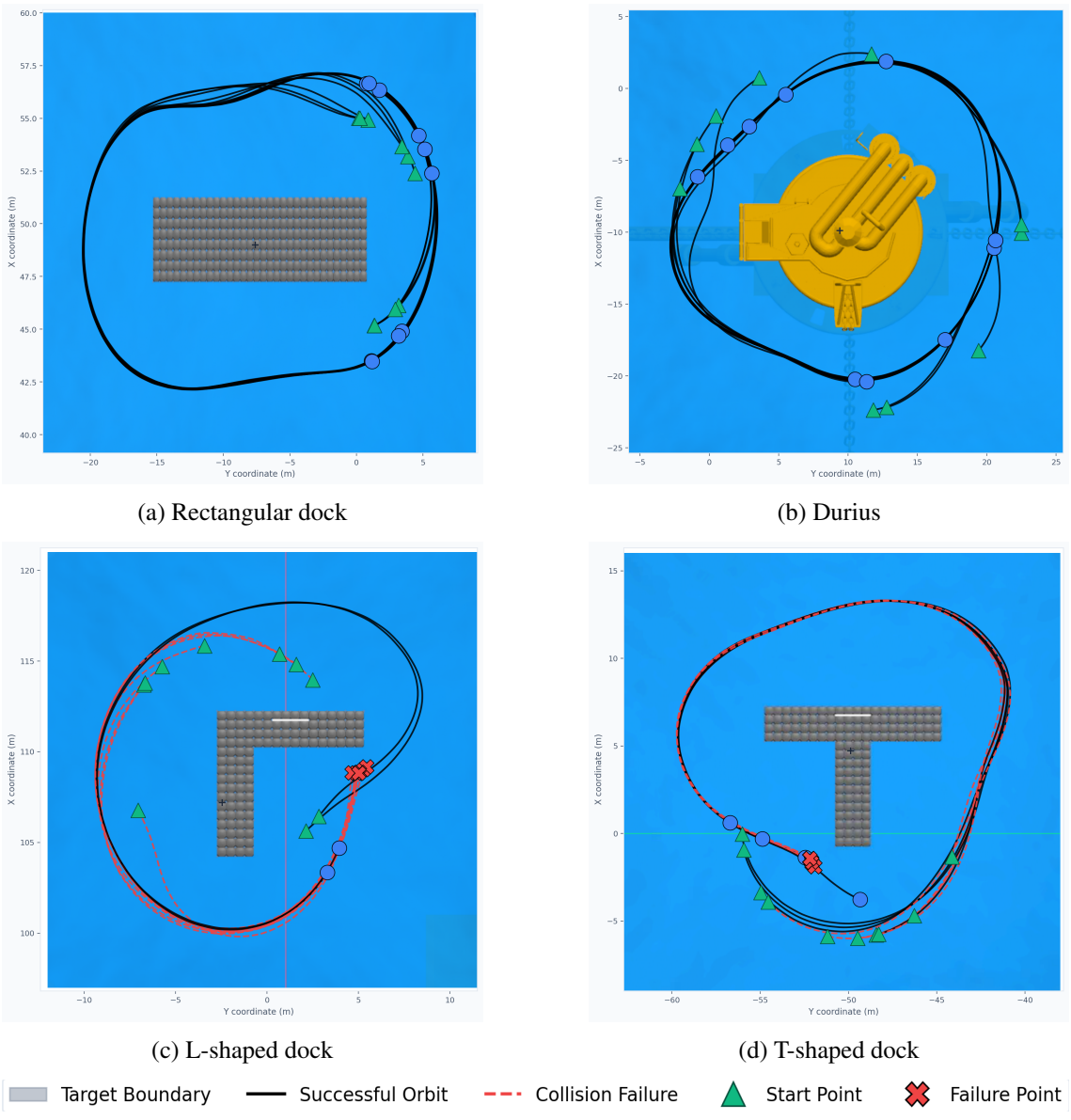
The trained policy is evaluated in Gazebo over forty episodes, ten on each of four fixed structures spanning convex and concave geometries, with the aggregate and per-target metrics reported in Table 4.12.

On the two convex structures the policy reproduces its VMAS behaviour faithfully. Both the rectangular dock and Durius are surveyed with a 100% success rate and no collisions, and the executed orbit tracks the commanded 5 m standoff closely, with mean standoff errors of 0.51 m and 0.54 m respectively. The corresponding panels of Figure 4.10 show the vehicle holding a steady offset from the boundary throughout the orbit, the executed path holding close to the commanded standoff, which indicates that the high-level waypoint policy transfers to the Gazebo dynamics without adaptation.

The two concave structures are representative of the most complex scenario. On the L-shaped dock the success rate falls to 20% with a collision rate of 80%, and on the T-shaped dock to 40% with a collision rate of 60%. A collision is registered when the vehicle closes to within 0.5 m of the target boundary, and the corresponding trajectory panels show these terminations occurring at

Table 4.12: Gazebo evaluation over forty episodes across four fixed structures, ten episodes per structure. Success, collision, and loss-of-target are percentages of episodes; standoff error and episode length are mean (standard deviation) over successful episodes. The commanded standoff is 5 m for all trials, and a collision is registered when the vehicle closes to within 0.5 m of the target boundary.

Target	Episodes	Success (%)	Collision (%)	Loss of Target (%)	Standoff Error (m)	Ep. length (steps)
Rectangular Dock	10	100.0	0.0	0.0	0.51 (0.05)	743.9 (9.8)
Durius	10	100.0	0.0	0.0	0.54 (0.12)	757.4 (12.0)
L-shaped Dock	10	20.0	80.0	0.0	0.92 (0.05)	424.5 (105.2)
T-shaped Dock	10	40.0	60.0	0.0	0.75 (0.14)	606.7 (42.0)
Overall	**40**	**65.0**	**35.0**	**0.0**	**0.59 (0.16)**	**633.1 (146.5)**



(a) Rectangular dock

(b) Durius

(c) L-shaped dock

(d) T-shaped dock

(e) Legend

Figure 4.10: Representative survey trajectories in Gazebo for the four evaluation structures: the rectangular dock and Durius (convex), and the L-shaped and T-shaped docks (concave). Solid lines are successful orbits, dashed lines are collision failures terminating at the marked failure point, triangles mark episode start points, and the shaded region is the target footprint. Since a collision is registered at a 0.5 m margin from the boundary, the failure points are marked at this distance from the footprint rather than in contact with it. The collisions on the concave docks occur at the re-entrant corners of the boundary.

the re-entrant corners of the boundary. The successful concave orbits are completed at the larger standoff errors of 0.92 m and 0.75 m respectively, consistent with the wider clearance required to cross the concave region.

Aggregated over the forty episodes, the policy attains a 65% success rate with no loss-of-target terminations, the failures confined to the concave structures. The transfer of the convex-target (the most commonly found in the field) behaviour into Gazebo without further training supports that a high-level policy operating on LiDAR-centric observations and option-level decisions is largely insensitive to the low-level dynamics mismatch that drives the reality gap. The residual difficulty on concave geometry identifies the local boundary curvature, rather than the simulator transfer, as the principal limitation of the current policy.

4.4 Hierarchical Skill Integration

The two contributions developed in this dissertation have so far been evaluated in isolation: the hierarchical survey skill in Section 4.3, and the Agentic Mission Planner in Section 4.2.3, the latter executing its inspection through primitive navigation skills. This section reports their integration within a single pipeline and, in doing so, exercises the exchangeable-block property of the proposed architecture. The layered design of Section 3.1 assigns each element a well-defined role, so that the block fulfilling a given job may be replaced by any alternative that follows the same inputs and outputs without disturbing the surrounding layers. The mission reported here exploits this property at the task interface: the inspection stage of the mission, ordinarily expanded by the Task Agent into a sequence of primitive navigation skills, is instead fulfilled directly by the hierarchical *SurveyTarget* skill of Section 3.4.1, which is treated here as a single high-level block occupying the task slot. The mission is executed in the Gazebo simulator on the Nautilus ASV model.

The *SurveyTarget* skill is admitted at this level of abstraction because it is not a primitive behaviour but a complete hierarchical controller in its own right. As formalised in Section 3.4.1, the skill is a Semi-MDP in which a high-level policy reasons over LiDAR observations and selects body-frame waypoints at option boundaries, while a frozen low-level *GoToXY* primitive executes each waypoint over the option horizon. This internal two-level structure mirrors, in learned form, the very decomposition that the Task Agent performs symbolically for other tasks: the high-level policy decides where to direct the vehicle, exactly as a sequence of waypoints generated by the Task Agent would, and the low-level primitive determines how to reach each one. The skill therefore operates at task granularity rather than skill granularity, and is in this sense equivalent to a task.

To enable a direct comparison with the waypoint-based inspection of Mission 2, the mission is initiated with the same operator prompt: *"Find Durius and collect data near the j-tubes. Summarize it, then come back."* From this command the Mission Agent produces the same four-task decomposition reported for Mission 2, namely a *SearchAndDetect* task to localise the target structure, a *SurveyTarget* task to perform the inspection, an *AutoGenerateReportSummary* task to consolidate

the collected data, and a final *Transit* task to return the vehicle to its origin. The *SearchAndDetect*, *AutoGenerateReportSummary*, and *Transit* tasks are expanded by the Task Agent in the usual manner: the *SearchAndDetect* task is realised through a *StationKeep* skill that holds the vehicle in place while the structure is detected; the *AutoGenerateReportSummary* task triggers a further *StationKeep* skill alongside the data-consolidation management skills; and the closing *Transit* task is executed by a *GoToXY* skill returning the vehicle to its origin.

The *SurveyTarget* task is handled differently. In Mission 2 the Task Agent expanded this task into eight sequential *GoToXY* skills tracing a fixed inspection pattern around the target. In the present mission the Task Agent call is substituted at this task by the *SurveyTarget* skill block: the located target position is routed into the skill, the vehicle is brought onto the target by a transit, and the surveying inspection is then carried out end-to-end by the hierarchical policy from LiDAR observations alone, without an externally supplied waypoint sequence. The remainder of the pipeline is unaffected by this substitution, since the skill block consumes the same task parameters and returns the same termination signal as the Task Agent expansion it replaces. As in the preceding simulation missions, target localisation is provided externally, and continuous manoeuvring is governed throughout by the frozen RL navigation policies.

The executed trajectory for a representative run is shown in Figure 4.11. The vehicle departs from the harbour structure, transits to the Durius target, completes a full orbit at the commanded standoff of 5 m under the *SurveyTarget* skill, and returns to its origin along a path that nearly retraces the outbound transit. The single continuous survey, in contrast to the segmented waypoint pattern produced in Mission 2, reflects the inspection being executed as one temporally extended block rather than as a discretised sequence of primitive skills.

Quantitative execution metrics for the mission, averaged over 20 independent executions, are reported in Table 4.13. The mission completes in a mean of 1410.05 simulation steps, corresponding to 282.01 s of simulated execution time, with a standard deviation of 1.79 s across repetitions. The two *GoToXY* transits terminate with mean completion errors of 0.97 m and 0.94 m, within the prescribed one-metre tolerance and consistent with the 0.971 m and 0.932 m tracking errors observed for the navigation policy in Missions 1 and 2. The station-keeping phases hold position to a mean error of 0.29 m, comparable to the 0.425 m reported for Mission 2. Most significantly, the *SurveyTarget* skill tracks the commanded standoff to a mean absolute error of 0.53 m over the survey. This figure is effectively identical to the 0.54 m obtained for the same structure under the standalone Gazebo evaluation of Table 4.12, indicating that the survey skill retains its standoff precision when embedded as a task-level block in the full pipeline rather than executed in isolation. The low standard deviations reported in Table 4.13 further indicate that this performance is consistent across repetitions rather than the product of a single favourable run.

These results close the loop between the two contributions of the dissertation and substantiate the exchangeable-block property of the architecture. A single free-form operator command is decomposed by the Mission Agent into a structured plan, and the inspection stage of that plan is fulfilled not by a symbolic Task Agent decomposition but by a learned hierarchical skill occupying the same task slot, with no modification to the surrounding layers and no loss of standoff accuracy

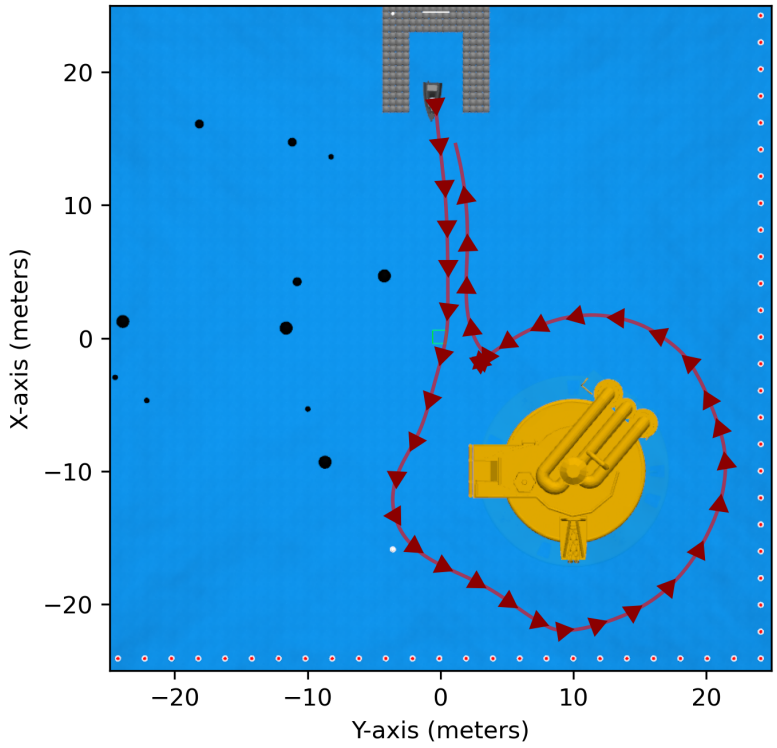


Figure 4.11: End-to-end trajectory for the integrated inspection mission in Gazebo, shown for a representative run from the set of repetitions. The vehicle launches from the harbour structure (top), transits to the Durius target, executes a complete survey at the commanded standoff using the hierarchical *SurveyTarget* skill as a single task-level block, and returns along a path that nearly retraces the outbound transit. Dark markers indicate the vehicle heading along the executed trajectory.

Table 4.13: Per-skill execution errors for the integrated *SurveyTarget* mission, reported as means over 20 independent executions of the same run with their standard deviations. Completion errors are the residual offset from the goal at skill termination; the station-keeping error is the mean position error averaged over the two station-holding phases; and the standoff error is the mean absolute deviation from the commanded standoff distance. The complete mission spans a mean of 1410.05 steps (282.01 s).

Task	Skill	Δx (m)	Δy (m)	Magnitude (m)
SurveyTarget	GoToXY (approach)	0.90 ± 0.04	-0.36 ± 0.02	0.97 ± 0.03
Transit	GoToXY (return)	-0.84 ± 0.05	-0.42 ± 0.03	0.94 ± 0.05
SearchAndDetect / Report	StationKeep	0.29 ± 0.01	0.02 ± 0.01	0.29 ± 0.01
SurveyTarget	SurveyTarget	–	–	0.53 ± 0.01

relative to the standalone skill. The architecture thus admits two interchangeable realisations of the same inspection task, a Task Agent expansion into primitive skills and a self-contained high-level skill, demonstrating that the blocks defined at its interfaces may be exchanged according to which realisation is better suited to the objective at hand.

4.5 Field Validation

The preceding sections reported results obtained in simulation, both for each contribution of this dissertation evaluated in isolation and for the full integrated pipeline. This section reports results obtained outside that controlled setting: the real-world field deployment of the Agentic Mission Planner on the Nautilus ASV, which exercises the full end-to-end pipeline rather than a single contribution in isolation.

Following the simulation experiments, the target-centric inspection mission was selected for physical deployment on the Nautilus ASV at the ATLANTIS Test Centre, as the more operationally demanding of the two scenarios evaluated in simulation. The deployment was carried out under the mild sea-state conditions described in Section 4.1.3, which subjected the vehicle to the wind, wave, and current disturbances characteristic of real unstructured marine environments and are representative of a typical operational inspection window.

The trials were initiated with the operator prompt: *"Find Durius and inspect the splash zone in the north face. Summarize it, then return home."* From this command, the Mission Agent reproduced the four-task decomposition structure evaluated in simulation: a *SearchAndDetect* task to locate the target structure, a *SurveyTarget* task to inspect the north-face splash zone, an *AutoGenerateReportSummary* task to consolidate the collected data, and a final *Transit* task to return the vehicle to its origin. Following the protocol established in simulation, the coordinates of the target structure were supplied directly to the planner, with onboard autonomous target detection deferred to future work. Continuous vehicle manoeuvring was governed by the same RL navigation policies deployed in simulation, with frozen neural network weights, requiring no platform-specific retraining.

To account for the positioning uncertainty inherent to real GPS-based localisation, compounded by the environmental disturbances of real marine operation, the *GoToXY* termination tolerance was relaxed from 1 m to 2 m relative to the simulation configuration. All other skill parameters and the operator review procedure remained identical to the simulation trials. The complete pipeline was executed eight times from different starting positions, with all four task termination conditions met within the relaxed tolerance across every trial.

The complete four-task pipeline achieved a mean completion time of 653.88 steps, corresponding to 130.8 s of execution time, averaged across the eight trials. Quantitatively, the *GoToXY* policy achieved a mean waypoint tracking error of 1.691 m, and *StationKeep* a mean error of 1.471 m during the structure acquirement and report generation phase. As anticipated, these residuals exceed their simulated counterparts of 0.932 m and 0.425 m respectively, an expected consequence of the relaxed termination tolerance and the environmental disturbances present during field deployment. Despite this degradation relative to simulation, the vehicle maintained coherent navigation throughout every trial and completed the full inspection sequence within the prescribed tolerance, confirming that the learned policies transferred successfully to the physical platform without any additional training. The resulting trajectory is shown in Figure 4.12, and an example of the collected data in Figure 4.13.

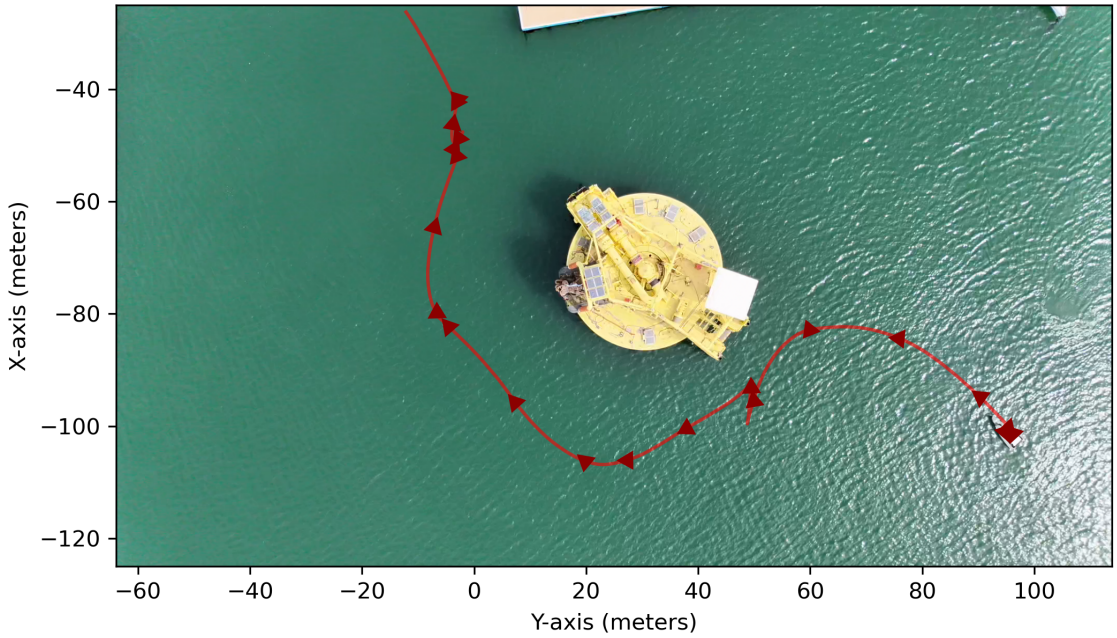


Figure 4.12: Field trajectory for Mission: Target-centric survey, demonstrating the ASV inspecting one of the faces of the target.

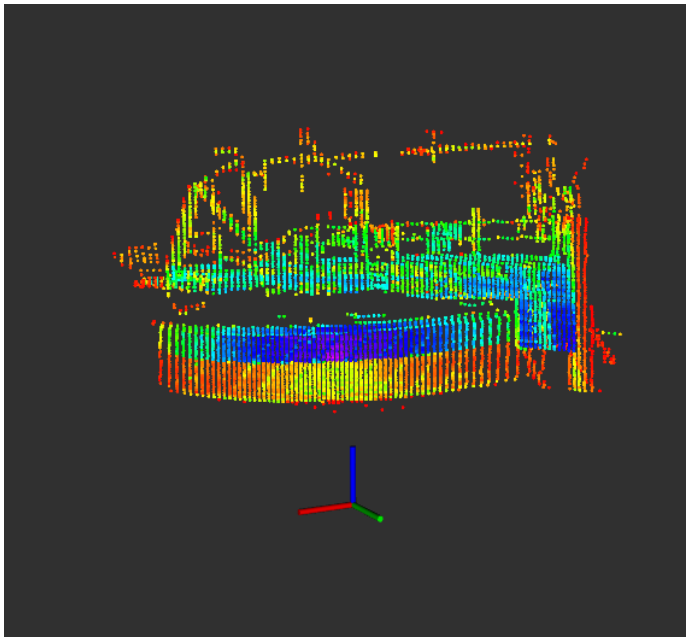


Figure 4.13: LiDAR point cloud of the target acquired during the survey mission.

The field results validate the end-to-end semantic-to-physical pipeline under real operational conditions. A free-form natural language command was reliably translated into a structured mission plan by the LLM agents and subsequently executed as continuous physical actuation on the Nautilus ASV operating in an unstructured marine environment. The successful completion of all task termination conditions across all eight trials, combined with coherent navigation throughout

each deployment, demonstrates that the decoupled architecture preserves robust execution when subjected to the disturbances characteristic of real maritime operations.

The decision to process LLM inference entirely onboard the vehicle, rather than relying on external cloud-hosted models, introduces an inherent latency trade-off at planning time. However, it ensures network independence and eliminates the connectivity dependencies identified in prior autonomous maritime systems [9], which relied on proprietary models accessed via external networks. This design guarantees reliable operation in remote marine environments where connectivity cannot be assumed, and ensures that the complete semantic-to-physical pipeline remains operational within the onboard compute constraints of the Nautilus ASV.

4.6 Chapter Conclusion

This chapter evaluated the two contributions of the dissertation, first characterising each within a simulated environment, then exercising the Agentic Mission Planner in field deployment and the two contributions together in simulation. For the Agentic Mission Planner, the comparative benchmarks established two principal findings. Supervised fine-tuning through LoRA proved to be a necessary condition for reliable structured decomposition at both abstraction levels, with base models, including instruction-tuned variants, unable to produce schema-valid, vocabulary-grounded output without task-specific adaptation, irrespective of scale. Once fine-tuned, compact models matched the decomposition quality of larger counterparts, allowing the complete two-agent hierarchy to run within the onboard compute budget of the vehicle. The selected agents, Llama-3.2-3B-Instruct at the Mission level and Llama-3.2-1B-Instruct at the Task level, attained F1 scores of 0.89 and 0.97 respectively, with near-zero hallucination and schema validity of 100% and 98%, while remaining within the latency constraints imposed by the deployment hardware. The qualitative robustness analysis further showed that the residual failures of the Mission Agent remained confined within the output schema and the reference vocabulary, so that every incorrect decomposition stayed detectable and operator-reviewable through the mission control interface, preserving the safety guarantee of the operator review gate.

For the hierarchical survey skill, the simulation study quantified both the behaviour of the policy and the contribution of each reward component. Under uniform shape sampling the policy completed a full orbit in 87.0% of VMAS episodes, with the entirety of the collision count concentrated on the L-shape, where the skill behaviour struggles most with the concave corner. Oversampling the concave class during training raised the overall success rate to 92.0% and reduced collisions to 2.0%, at the cost of a roughly doubled standoff error and a small loss-of-target rate, a trade favourable for a task whose primary criterion is a collision-free, completed survey. The reward ablation showed the three shaping terms to be complementary and largely non-overlapping in function: the progress term is the main component that produces the survey, the standoff term sets its radius, and the visibility term protects sensor coverage, with only the progress term strictly necessary for any successful survey to emerge. Deployment of the trained policy in Gazebo without further adaptation reproduced the VMAS behaviour faithfully on the convex structures, which

were surveyed at a 100% success rate with sub-metre standoff error, while the concave structures remained the sole difficulty. This transfer identifies local boundary curvature, rather than the simulator gap, as the principal limitation of the current policy.

Brought together, the two contributions were integrated and evaluated as a single pipeline in Gazebo simulation, exercising the exchangeable-block property of the architecture. The inspection stage of the mission, expanded by the Task Agent into a sequence of primitive navigation skills, was instead fulfilled directly by the hierarchical *SurveyTarget* skill occupying the same task slot, with no modification to the surrounding layers. From a single free-form command the Mission Agent produced the expected four-task decomposition, and the embedded skill tracked the commanded standoff to a mean absolute error of 0.53 m, effectively identical to the 0.54 m obtained for the same structure in standalone evaluation. The survey skill therefore retains its accuracy when embedded as a task-level block rather than executed in isolation, closing the loop between the two contributions and substantiating the interchangeability of the blocks defined at the architecture's interfaces.

The field deployment, by contrast, exercised the Agentic Mission Planner together with its primitive RL navigation skills rather than the hierarchical survey skill, and constitutes the culminating result of the chapter. Having first been validated end-to-end in Gazebo across both representative missions, in which the navigation policies achieved mean waypoint tracking errors of 0.971 m and 0.932 m for the area-survey and target-inspection scenarios, the planner was deployed on the Nautilus ASV at the ATLANTIS Test Centre, under the wind, wave, and current disturbances of a real marine environment. From a single free-form operator command, the planner reproduced the four-task decomposition evaluated in simulation, and the mission was executed eight times from different starting positions, with all four task termination conditions met within the relaxed tolerance on every trial. The learned navigation policies were transferred with frozen weights and required no platform-specific retraining; the resulting tracking residuals of 1.691 m and 1.471 m exceeded their simulated counterparts, as expected under the relaxed tolerance and the environmental forcing, yet the vehicle maintained coherent navigation throughout and completed the full inspection sequence in each trial.

Taken as a whole, the results reported in this chapter substantiate the central proposition of the framework. Flexible, language-based mission specification and robust, learned skill execution can be decoupled and then recombined into a pipeline that operates end-to-end, from a free-form natural language command to physical actuation, and that sustains this operation not only in physics-based simulation but on a physical platform deployed in an unstructured marine environment.

Chapter 5

Conclusions and Future Work

This dissertation set out to address the problem of mission-level autonomy for ASV inspection of maritime infrastructure, a problem in which a human operator must be able to express what the vehicle is to accomplish without composing low-level command sequences by hand, while the vehicle must execute the resulting plan robustly across extended time horizons and under the wind, wave, and current disturbances characteristic of unstructured marine environments. The survey of existing work showed that these two demands, flexible high-level specification on the one hand and robust long-horizon execution on the other, are difficult to satisfy together, since no single paradigm simultaneously provides a flexible interface for specifying missions and an execution layer that remains dependable over long horizons without recourse to continual replanning or human intervention. The framework developed in this dissertation responded to that gap by coupling LLM-based mission planning with RL-based skill execution, organised around a single principle: the separation of semantic reasoning, performed at planning time, from physical actuation, delegated to a fixed library of validated skills. Realised through a hierarchical four-layer translation engine spanning the Mission, Task, Skill, and Action layers, and through a mission ontology and a mandatory operator review gate, this organisation ensures that each executed manoeuvre remains traceable to an approved task, and that the language, perception, and control components can be developed and replaced independently.

This framework was instantiated through two complementary contributions, and the experimental evaluation substantiated each in turn before exercising them together. The first, the Agentic Mission Planner, addressed the specification side of the problem through two fine-tuned language models deployed entirely onboard. The comparative benchmarks established that supervised fine-tuning through LoRA is a necessary condition for reliable structured decomposition at both abstraction levels, and that, once fine-tuned, compact models match the decomposition quality of considerably larger counterparts, allowing the complete two-agent hierarchy to run within the onboard compute budget of the vehicle. The selected agents attained F1 scores of 0.89 and 0.97 at the Mission and Task levels, with near-zero hallucination and schema validity of 100% and 98%, while the residual failures of the Mission Agent remained confined within the output schema and the reference vocabulary, so that every incorrect decomposition stayed detectable and operator-

reviewable, and the safety guarantee of the review gate was preserved. The second contribution, the hierarchical RL survey skill, addressed the execution side, equipping a high-level policy that reasons over a LiDAR-centric observation, free of any privileged knowledge of the object's geometry, and composes a low-level navigation primitive to orbit an *a priori* unknown structure at a commanded standoff distance. In simulation the policy completed a full orbit in the large majority of episodes, with its residual difficulty concentrated on concave geometry, and it transferred to Gazebo without further adaptation, reproducing the learned behaviour faithfully on convex structures at a 100% success rate with sub-metre standoff error.

Their integration was then demonstrated in simulation. The planner had already been shown to translate both representative missions across the full abstraction hierarchy and execute them through its primitive navigation skills; building on this, a single inspection mission was repeated with the hierarchical survey skill substituted into the task slot that would otherwise be expanded into primitives. The chapter then culminated in the framework's first deployment beyond the simulator, with the Agentic Mission Planner run on the Nautilus ASV at the ATLANTIS Test Centre under genuine wind, wave, and current forcing. There a single free-form command was again decomposed into the four tasks established in simulation, and the vehicle carried out the mission from a range of starting positions, sustaining coherent navigation throughout and completing the inspection on each attempt. Through these results the objectives set out at the start of the dissertation were met: a hierarchical, four-layer mission execution architecture that separates semantic reasoning from physical execution and preserves an operator review gate; a hierarchical RL skill that infers the geometry of an unknown target from onboard sensing and sustains a stable orbit under disturbance; the realisation of the upper layers through two fine-tuned, onboard language models selected by comparative benchmarking; and an end-to-end validation that carried the framework from physics-based simulation to a physical platform in the field.

Beyond technical performance, these properties reflect the core design logic of the architecture and support its alignment with governance expectations for safety-critical autonomous systems. The mandatory approval gate instantiates the meaningful human oversight required of high-risk systems, while the task-to-manoeuvre traceability supports the record-keeping and auditability that the same regime demands, positioning the framework favourably with respect to forthcoming regulatory requirements. Taken as a whole, the results reported in this dissertation substantiate its central proposition: flexible, language-based mission specification and robust, learned skill execution can be decoupled and then recombined into a pipeline that operates end-to-end, from a free-form natural language command to physical actuation, and that sustains this operation not only in physics-based simulation but on a physical platform deployed in an unstructured marine environment. The combination of LLM-based mission specification and RL-based skill execution thus constitutes a viable approach to adaptive yet trustworthy mission planning and execution for ASV inspection, particularly in settings where natural-language specification, modular skill execution, robustness to disturbance, and onboard autonomy are essential design requirements.

5.1 Future Work

The results presented in this dissertation also delineate several directions in which the framework can be extended.

The most immediate concerns the field validation of the hierarchical survey skill. Whereas the field deployment exercised the Agentic Mission Planner with the standard navigation policies, the survey skill itself was validated only in simulation, across both VMAS and Gazebo. The natural next step is therefore its evaluation on a physical platform against real structures. The simulation results already show reliable behaviour on convex and straight boundaries, which characterise the majority of the maritime structures targeted for inspection, so the skill is expected to transfer to the field much as the navigation policies did; the concave case, where re-entrant corners remain the sole residual difficulty, represents a narrower and longer-term refinement rather than an obstacle to deployment.

Secondly, the Mission-level training data, constructed here to cover the templates expressible through the proposed ontology, could be expanded to a wider range of mission phrasings as the operational vocabulary grows, further strengthening the generalisation of the Mission Agent.

A final direction concerns the extension of the framework to other classes of autonomous vehicles. The value of the layered, modular design is that semantic reasoning is separated from physical execution and the layers communicate through well-defined interfaces, so that the architecture itself, comprising the Mission and Task agents, the mission ontology, and the operator review gate, is not specific to a ASV. Porting the framework to a different platform, such as an aerial vehicle, would leave this structure intact: the underlying skill library would be replaced by one of aerial skills suited to the new platform, and the agents would be re-fine-tuned on data reflecting that new skill set. Pursuing this direction would test the generalisation capabilities of the architecture and broaden its applicability across inspection domains.

References

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The Option-Critic Architecture". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2017, pp. 1726–1734. DOI: 10.1609/aaai.v31i1.10916.

[2] Andrew G. Barto and Sridhar Mahadevan. "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete Event Dynamic Systems* 13.4 (Oct. 2003), pp. 341–379. ISSN: 1573-7594. DOI: 10.1023/A:1025696116075.

[3] Mehdi Belabyad et al. "Skills and competencies for operating maritime autonomous surface ships (MASS): a systematic review and bibliometric analysis". In: *Maritime Policy and Management* 53.1 (2026), pp. 1–26. DOI: 10.1080/03088839.2025.2475177.

[4] Matteo Bettini et al. "VMAS: A Vectorized Multi-agent Simulator for Collective Robot Learning". In: *Distributed Autonomous Robotic Systems*. Springer Nature Switzerland, 2024, pp. 42–56. ISBN: 978-3-031-51497-5. DOI: 10.1007/978-3-031-51497-5_4.

[5] brian ichter brian et al. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *Proceedings of The 6th Conference on Robot Learning*. Ed. by Karen Liu, Dana Kulic, and Jeff Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2023, pp. 287–318. URL: https://proceedings.mlr.press/v205/ichter23a.html.

[6] Yuji Cao et al. "Survey on Large Language Model-Enhanced Reinforcement Learning: Concept, Taxonomy, and Methods". In: *IEEE Transactions on Neural Networks and Learning Systems* 36.6 (2025), pp. 9737–9757. DOI: 10.1109/TNNLS.2024.3497992.

[7] Thomas Carta et al. "Grounding Large Language Models in Interactive Environments with Online Reinforcement Learning". In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 3676–3713. URL: https://proceedings.mlr.press/v202/carta23a.html.

[8] Maricruz F. S. Cepeda et al. "Exploring Autonomous and Remotely Operated Vehicles in Offshore Structure Inspections". In: *Journal of Marine Science and Engineering* 11.11 (2023), p. 2172. DOI: 10.3390/jmse11112172.

[9] Kim Alexander Christensen et al. "AI Captain: Conversational mission planning and execution system for autonomous surface vehicles". In: *Ocean Engineering* 338 (2025), p. 121988. ISSN: 0029-8018. DOI: 10.1016/j.oceaneng.2025.121988.

[10] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton: CRC Press, July 2018. ISBN: 978-0-429-48910-5. DOI: 10.1201/9780429489105.

[11] Thomas G. Dietterich. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition". In: *Journal of Artificial Intelligence Research* 13 (2000), pp. 227–303. DOI: 10.1613/jair.639.

[12] Benjamin Eysenbach et al. *Diversity is All You Need: Learning Skills without a Reward Function*. 2018. arXiv: 1802.06070 [cs.AI]. URL: https://arxiv.org/abs/1802.06070.

[13] Aleksandra Faust et al. "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 5113–5120. DOI: 10.1109/ICRA.2018.8461096.

[14] Richard E. Fikes and Nils J. Nilsson. "Strips: A new approach to the application of theorem proving to problem solving". In: *Artificial Intelligence* 2.3 (1971), pp. 189–208. ISSN: 0004-3702. DOI: 10.1016/0004-3702(71)90010-5.

[15] Carlos Florensa, Yan Duan, and Pieter Abbeel. *Stochastic Neural Networks for Hierarchical Reinforcement Learning*. 2017. arXiv: 1704.03012 [cs.AI]. URL: https://arxiv.org/abs/1704.03012.

[16] Maria Fox and Derek Long. "PDDL2.1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research* 20 (2003), pp. 61–124. DOI: 10.1613/jair.1129.

[17] Kevin Frans et al. *Meta Learning Shared Hierarchies*. 2017. arXiv: 1710.09767 [cs.LG]. URL: https://arxiv.org/abs/1710.09767.

[18] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1587–1596. URL: https://proceedings.mlr.press/v80/fujimoto18a.html.

[19] Javier García, Fern, and o Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research* 16.42 (2015), pp. 1437–1480. URL: http://jmlr.org/papers/v16/garcia15a.html.

[20] Global Wind Energy Council. *Global Offshore Wind Report 2026*. https://www.gwec.net/reports/globaloffshorewindreport. Accessed: 2026-06-27. 2026.

[21] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. "Variational Intrinsic Control". In: *arXiv preprint arXiv:1611.07507* (2016). DOI: 10.48550/arXiv.1611.07507. arXiv: 1611.07507 [cs.LG].

[22] Shixiang Gu et al. "Continuous deep Q-learning with model-based acceleration". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 2829–2838.

[23] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

[24] Karol Hausman et al. "Learning an Embedding Space for Transferable Robot Skills". In: *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=rk07ZXZRb.

[25] Wenlong Huang et al. “Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents”. In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 9118–9147. URL: https://proceedings.mlr.press/v162/huang22a.html.

[26] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. “Hierarchical Reinforcement Learning: A Survey and Open Research Challenges”. In: *Machine Learning and Knowledge Extraction* 4.1 (2022), pp. 172–221. DOI: 10.3390/make4010009.

[27] Auke Jan Ijspeert et al. “Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors”. In: *Neural Computation* 25.2 (2013), pp. 328–373. DOI: 10.1162/NECO_a_00393.

[28] Félix Ingrand and Malik Ghallab. “Deliberation for autonomous robots: A survey”. In: *Artificial Intelligence* 247 (2017), pp. 10–44. ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.11.003.

[29] Michael Janner et al. “When to Trust Your Model: Model-Based Policy Optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. DOI: 10.48550/arXiv.1906.08253.

[30] YiDing Jiang et al. “Language as an Abstraction for Hierarchical Deep Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. DOI: 10.48550/arXiv.1906.07343.

[31] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial Intelligence* 101.1 (1998), pp. 99–134. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(98)00023-X.

[32] Leslie Pack Kaelbling and Tomás Lozano-Pérez. “Hierarchical task and motion planning in the now”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1470–1477. DOI: 10.1109/ICRA.2011.5980391.

[33] Samarth Kalluraya, George J. Pappas, and Yiannis Kantaros. “Multi-Robot Mission Planning in Dynamic Semantic Environments”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 1630–1637. DOI: 10.1109/ICRA48891.2023.10160344.

[34] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[35] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[36] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[37] George Konidaris and Andrew Barto. "Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper_files/paper/2009/file/e0cf1f47118daebc5b16269099ad7347-Paper.pdf.

[38] George Konidaris et al. "Robot learning from demonstration by constructing skill trees". In: *The International Journal of Robotics Research* 31.3 (2012), pp. 360–375.

[39] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (2013), pp. 122–145. DOI: 10.1109/TEVC.2012.2185849.

[40] Nathan Lambert, Kristofer Pister, and Roberto Calandra. "Investigating Compounding Prediction Errors in Learned Dynamics Models". In: *arXiv preprint arXiv:2203.09637* (2022). DOI: 10.48550/arXiv.2203.09637. arXiv: 2203.09637 [cs.LG].

[41] Timothy P. Lillicrap et al. "Continuous Control with Deep Reinforcement Learning". In: *Proceedings of the 4th International Conference on Learning Representations (ICLR)*. 2016. DOI: 10.48550/arXiv.1509.02971. arXiv: 1509.02971 [cs.LG].

[42] Bo Liu et al. "LLM+P: Empowering Large Language Models with Optimal Planning Proficiency". In: *arXiv preprint arXiv:2304.11477* (2023). DOI: 10.48550/arXiv.2304.11477. arXiv: 2304.11477 [cs.RO].

[43] Zhixiang Liu et al. "Unmanned surface vehicles: An overview of developments and challenges". In: *Annual Reviews in Control* 41 (2016), pp. 71–93. ISSN: 1367-5788. DOI: 10.1016/j.arcontrol.2016.04.018.

[44] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021. arXiv: 2108.10470 [cs.RO]. URL: https://arxiv.org/abs/2108.10470.

[45] Justin E. Manley. "Unmanned surface vehicles, 15 years of development". In: *OCEANS 2008*. 2008, pp. 1–4. DOI: 10.1109/OCEANS.2008.5152052.

[46] Alejandro Marzinotto et al. "Towards a unified behavior trees framework for robot control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 5420–5427. DOI: 10.1109/ICRA.2014.6907656.

[47] Cynthia Matuszek et al. "Learning to Parse Natural Language Commands to a Robot Control System". In: *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Ed. by Jaydev P. Desai et al. Heidelberg: Springer International Publishing, 2013, pp. 403–415. ISBN: 978-3-319-00065-7. DOI: 10.1007/978-3-319-00065-7_28.

[48] Dipendra Misra et al. "Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2667–2678. DOI: 10.18653/v1/D18-1287.

[49] Volodymyr Mnih et al. "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[50] Thomas M. Moerland et al. "Model-based Reinforcement Learning: A Survey". In: *Found. Trends Mach. Learn.* 16.1 (Jan. 2023), pp. 1–118. ISSN: 1935-8237. DOI: 10.1561/2200000086.

[51] Ziaul Haque Munim and Hercules Haralambides. "Advances in maritime autonomous surface ships (MASS) in merchant shipping". In: *Maritime Economics and Logistics* 24 (2022), pp. 181–188. DOI: 10.1057/s41278-022-00232-y.

[52] Ofir Nachum et al. "Data-Efficient Hierarchical Reinforcement Learning". In: *Advances in Neural Information Processing Systems 31 (NeurIPS)*. 2018, pp. 3303–3313. DOI: 10.48550/arXiv.1805.08296. arXiv: 1805.08296 [cs.LG].

[53] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 1558608567.

[54] Shubham Pateria et al. "Hierarchical Reinforcement Learning: A Comprehensive Survey". In: *ACM Comput. Surv.* 54.5 (June 2021). ISSN: 0360-0300. DOI: 10.1145/3453160.

[55] Andry Maykol Pinto et al. "ATLANTIS - The Atlantic Testing Platform for Maritime Robotics". In: *OCEANS 2021: San Diego - Porto*. 2021, pp. 1–5. DOI: 10.23919/OCEANS44145.2021.9706059.

[56] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[57] Nikita Rudin et al. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning". In: *Proceedings of the 5th Conference on Robot Learning (CoRL)*. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2022, pp. 91–100. URL: https://proceedings.mlr.press/v164/rudin22a.html.

[58] Andrei A. Rusu et al. "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 262–270. URL: https://proceedings.mlr.press/v78/rusu17a.html.

[59] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (2017). DOI: 10.48550/arXiv.1707.06347. arXiv: 1707.06347 [cs.LG].

[60] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. PMLR, 2015, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

[61] Dhruv Shah et al. "LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action". In: *Proceedings of The 6th Conference on Robot Learning*. Ed. by Karen Liu, Dana Kulic, and Jeff Ichnowski. Vol. 205. Proceedings of Machine Learning Research. PMLR, 2023, pp. 492–504. URL: https://proceedings.mlr.press/v205/shah23b.html.

[62] Archit Sharma et al. "Dynamics-Aware Unsupervised Discovery of Skills". In: *Proceedings of the 8th International Conference on Learning Representations (ICLR)*. 2020. DOI: 10.48550/arXiv.1907.01657. arXiv: 1907.01657 [cs.LG].

[63] Bharat Singh, Rajesh Kumar, and Vinay Pratap Singh. "Reinforcement learning in robotic applications: a comprehensive survey". In: *Artificial Intelligence Review* 55.2 (Feb. 2022), pp. 945–990. ISSN: 1573-7462. DOI: 10.1007/s10462-021-09997-9.

[64] Ishika Singh et al. "ProgPrompt: Generating Situated Robot Task Plans using Large Language Models". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 11523–11530. DOI: 10.1109/ICRA48891.2023.10161317.

[65] Simon Stepputtis et al. “Language-conditioned imitation learning for robot manipulation tasks”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546. DOI: 10.5555/3495724.3496826.

[66] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[67] Richard S Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[68] Richard S. Sutton, Doina Precup, and Satinder Singh. “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In: *Artificial Intelligence* 112.1–2 (1999), pp. 181–211. DOI: 10.1016/S0004-3702(99)00052-1.

[69] Jie Tan et al. *Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*. 2018. arXiv: 1804.10332 [cs.RO]. URL: https://arxiv.org/abs/1804.10332.

[70] Stefanie Tellex et al. “Robots That Use Language”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 25–55. DOI: 10.1146/annurev-control-101119-071628.

[71] Stefanie Tellex et al. “Understanding natural language commands for robotic navigation and mobile manipulation”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. 1. 2011, pp. 1507–1514. DOI: 10.1609/aaai.v25i1.7979.

[72] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[73] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[74] Anete Vagale et al. “Path planning and collision avoidance for autonomous surface vehicles I: a review”. In: *Journal of Marine Science and Technology* 26 (2021), pp. 1292–1306. DOI: 10.1007/s00773-020-00787-6.

[75] Alexander Sasha Vezhnevets et al. “FeUdal Networks for Hierarchical Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 3540–3549. URL: https://proceedings.mlr.press/v70/vezhnevets17a.html.

[76] Christopher J. C. H. Watkins and Peter Dayan. “Q-Learning”. In: *Machine Learning* 8.3–4 (1992), pp. 279–292. DOI: 10.1007/BF00992698.

[77] Joohyun Woo, Chanwoo Yu, and Nakwan Kim. “Deep reinforcement learning-based controller for path following of an unmanned surface vehicle”. In: *Ocean Engineering* 183 (2019), pp. 155–166. ISSN: 0029-8018. DOI: 10.1016/j.oceaneng.2019.04.099.

[78] Yao Zhang. “Intelligent Robot Control and Uncertainty Analysis Integrating Reinforcement Learning and Large Language Models”. In: *Journal of Computing and Electronic Information Management* 17.1 (2025), pp. 1–5. DOI: 10.54097/r6xtmv46.

[79] Chao Zhou et al. “Autonomous Surface Vessel Obstacle Avoidance Based on Hierarchical Reinforcement Learning”. In: *Proceedings of the ASME 2020 39th International Conference on Ocean, Offshore and Arctic Engineering (OMAE), Volume 1: Offshore Technology*. ASME, 2020, V001T01A005. DOI: 10.1115/OMAE2020-18454.