KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Machine Learning

Second cycle, 30 credits

AI for Networked Robotics

Towards Adaptive Reinforcement Learning for Network-Aware Robotics via Quantization Techniques

MICHAŁ ANDRZEJ SITARZ

Stockholm, Sweden, 2026

AI for Networked Robotics

Towards Adaptive Reinforcement Learning for Network-Aware Robotics via Quantization Techniques

MICHAŁ ANDRZEJ SITARZ

Master's Programme, Machine Learning, 120 credits
Date: January 1, 2026

Supervisors: Christos N. Mavridis, Fernando Dos Santos Barbosa
Examiner: Karl Henrik Johansson

Host company: Ericsson Research
Swedish title: AI för nätverksansluten robotik
Swedish subtitle: Mot adaptiv förstärkningsinlärning för nätverksmedveten robotik via kvantiseringstekniker

Abstract

Motion planning algorithms have been at the center of research focus in the past few decades, aiming to find efficient solutions to the problem of controlling single- and multi-agent systems under different environments and specifications. With the advent of fast high-band wireless technologies and smart largely connected autonomous devices, however, there is an emergent need for a paradigm shift in the control design of networked cyber-physical systems over wireless channels. Towards this direction, this thesis investigates new reinforcement learning methods as adaptive, memory-efficient learning-based control schemes that can achieve optimal robot control under wireless network constraints. In particular, quantization methods are explored in the context of adaptive state aggregation, aiming to solve the communication-aware planning problem using a memory-efficient Q-learning algorithm. Three main state aggregation Q-learning methods were developed based on three quantization methods: Stochastic Vector Quantization (SVQ), Self-Organizing Maps (SOM), and Online Deterministic Annealing (ODA). Experiments were conducted in simulated environments with real 5G network observations to assess the impact of each method on policy optimality, training efficiency, and memory/information compression. The results indicate that optimal task-agnostic state aggregation can be progressively generated using annealing optimization solved with gradient-free two-timescale stochastic approximation. This enables localized Q-function approximation with increasing detail, as needed by the task at hand. Several extensions to the original framework were investigated to enhance learning performance and better align the method with communication-aware robot control requirements. The results of this work show potential in the advancement of adaptive Q-learning techniques and the development of new network-aware robot control methods.

Keywords

Learning-based Control, Reinforcement Learning, Communication-aware Motion Planning, Network Control, Vector Quantization, Online Deterministic Annealing

Sammanfattning

Rörelseplaneringsalgoritmer har stått i centrum för forskningsfokus under de senaste årtiondena, med målet att hitta effektiva lösningar på problemet att styra en- och fleragentsystem i olika miljöer och under olika specifikationer. Med framväxten av snabba trådlösa teknologier med hög bandbredd och smarta, starkt uppkopplade autonoma enheter finns det emellertid ett växande behov av ett paradigmskifte i styrdesignen för nätverksbaserade cyberfysiska system över trådlösa kanaler. I denna riktning undersöker denna avhandling nya förstärkningsinlärningsmetoder som adaptiva, minneseffektiva inlärningsbaserade styrsystem som kan uppnå optimal robotstyrning under begränsningar i trådlösa nätverk. Särskilt utforskas kvantiseringsmetoder i samband med adaptiv tillståndsaggregering, med målet att lösa det kommunikationsmedvetna planeringsproblemet med en minneseffektiv Q-inlärningsalgoritm. Tre huvudsakliga tillståndsaggregerande Q-inlärningsmetoder utvecklades baserat på tre kvantiseringsmetoder: stokastisk vektorkvantisering (SVQ), självorganiserande kartor (SOM) och online deterministisk anlöpning (ODA). Experiment utfördes i simulerade miljöer med verkliga observationer från 5G-nätverk för att utvärdera effekten av varje metod på policyoptimalitet, träningseffektivitet samt minnes- och informationskomprimering. Resultaten visar att optimal, uppgiftsoberoende tillståndsaggregering kan genereras successivt med hjälp av anlöpningsoptimering som löses med gradientfri tvåskalig stokastisk approximation. Detta möjliggör lokaliserad Q-funktionsapproximation med ökande detaljnivå, beroende på den aktuella uppgiften. Flera utvidgningar av det ursprungliga ramverket undersöktes för att förbättra inlärningsprestandan och bättre anpassa metoden till krav inom kommunikationsmedveten robotstyrning. Resultaten av detta arbete visar på potentialen för vidareutveckling av adaptiva Q-inlärningstekniker och utvecklingen av nya nätverksmedvetna robotstyrningsmetoder.

Nyckelord

Inlärningsbaserad styrning, Förstärkningsinlärning, Kommunikationsmedveten rörelseplanering, Nätverksstyrning, Vektorkvantisering, Online deterministisk anlöpning

Contents

1 Introduction **1**
1.1 Background . . . 2
1.2 Problem . . . 3
1.2.1 Original Problem and Definition . . . 4
1.2.2 Scientific and Engineering Issues . . . 4
1.3 Purpose . . . 5
1.4 Goals . . . 5
1.5 Research Methodology . . . 6
1.6 Delimitations . . . 6
1.7 Report Structure . . . 7

2 Background **9**
2.1 Communication-Aware Motion Planning . . . 9
2.1.1 Motion Planning . . . 9
2.1.2 Wireless Communication Basics . . . 10
2.2 Reinforcement Learning . . . 11
2.2.1 Markov Decision Process . . . 12
2.2.2 Bellman's Principle of Optimality . . . 12
2.2.3 Methods . . . 13
2.2.4 Q-Learning . . . 14
2.3 Vector Quantization . . . 15
2.3.1 Hybrid RL . . . 16
2.3.2 Methods . . . 17
2.4 Related Work . . . 22
2.4.1 Network-Aware Robotics . . . 22
2.4.2 Machine Learning Classifiers for Communication-Aware Robotics . . . 23
2.4.3 Q-learning with Online Deterministic Annealing . . . 24
2.5 Summary . . . 25

3 Methods **27**
3.1 Research Process and Paradigm . . . 27
3.2 Data Collection . . . 29
3.2.1 Sampling . . . 29
3.2.2 Sample Size . . . 29
3.2.3 Target Population . . . 29
3.3 Experimental design and Planned Measurements . . . 30
3.3.1 Test environments . . . 30
3.3.2 Hardware/Software . . . 31
3.4 Assessing Reliability and Validity . . . 31
3.4.1 Validity . . . 31
3.4.2 Reliability . . . 32
3.5 Planned Evaluation Framework . . . 33
3.5.1 Evaluation framework . . . 34
3.5.2 Software Tools . . . 35
3.6 System documentation . . . 35

4 Methodology and Implementation **39**
4.1 CartPole Simulation . . . 39
4.1.1 CartPole Environment . . . 39
4.1.2 Hybrid Algorithm . . . 40
4.2 Communication-Aware Robotics Application . . . 46
4.2.1 Motion Planning . . . 46
4.2.2 Communication-Aware Motion Planning . . . 48

5 Results and Analysis **49**
5.1 Results . . . 49
5.1.1 CartPole . . . 49
5.1.2 Communication-Aware Motion Planning . . . 57
5.1.3 CAMP . . . 59
5.2 Analysis and Discussion . . . 62
5.2.1 Reliability Analysis . . . 62
5.2.2 Validity Analysis . . . 63

6 Conclusions and Future Work **65**
6.1 Conclusions . . . 65
6.2 Limitations . . . 67
6.3 Future work . . . 68
6.4 Reflections . . . 69

References **73**

List of Figures

1.1 Network Digital Twin (NDT) vs. Reinforcement Learning (RL) Motion Planning . . . 3
1.2 Standard vs. Communication-Aware Motion Planning . . . 3

2.1 Q-learning training loop . . . 14
2.2 State Aggregation . . . 16
2.3 Bifurcation process as temperature drops in ODA [19]* . . . 21

3.1 System Structure . . . 37

4.1 CartPole simulation environment [21]* . . . 40
4.2 Adapted from [20]* . . . 41
4.3 Stochastic Vector Quantization update algorithm . . . 42
4.4 Diverse starting codevector configurations . . . 43
4.5 Self-Organizing Maps update algorithm . . . 43
4.6 Self-Organizing Maps update algorithm . . . 44
4.7 Self-Organizing Maps update algorithm . . . 44
4.8 Online Deterministic Annealing algorithm adapted from [20]* 45
4.9 Motion Planning environment [left]; Communication-Aware Motion Planning (CAMP) environment [right] . . . 47

5.1 Different Stochastic Vector Quantization (SVQ) codevector initializations and their resulting locations after convergence (indicated by an arrow). Red indicates if they have not moved from their original position ("dead neurons"). Uniform 3D grid *[left]*; 2D dimensional manifold *[right]* . . . 50
5.2 Resulting training curves (red) with different SVQ learning rates (orange), Q-learning rate (blue), SVQ learning rate (orange) . . . 50

5.3 Resulting training curves (red) with initialization from Figure 5.1. Resulting training curves (red) with initialization from Figure 5.1, SVQ learning rates (orange), Q-learning rate (blue), and SVQ learning rate (orange) 51
5.4 Short pre-training of an SOM, and then switching to SVQ updates 52
5.5 SOM performance and parameters using a replay buffer and online normalization 53
5.6 Online Deterministic Annealing (ODA) algorithm trained on CartPole with and without normalization 54
5.7 Different update frequencies for ODA CartPole training with different seeds 55
5.8 Number of converged codevectors as the training length increases (using batch updates) 55
5.9 Average number of timesteps for different numbers of state codevectors 56
5.10 Converged codevectors on CartPole 57
5.11 Comparison of resulting trajectories (blue) and optimal actions (arrows) at different codevectors, between ad hoc discretization and ODA on traditional Motion Planning 58
5.12 Training curves for ad hoc discretization (low and high resolution) and ODA on traditional Motion Planning 59
5.13 Comparison of the effect of reward functions with different weight parameters on the robot trajectories (evaluated using the method explained in Chapter 4) 60
5.14 More in-depth comparison of behaviors between agents trained on different reward functions 61
5.15 Performance distribution for different reward functions (evaluated by the ideal reward function where each component is evaluated equally) 61

List of acronyms and abbreviations

CAMP	Communication-Aware Motion Planning
DA	Deterministic Annealing
EM	Electromagnetic
MAE	Mean Absolute Error
MDP	Markov Decision Process
ML	Machine Learning
MSE	Mean Squared Error
NDT	Network Digital Twin
NN	Neural Network
ODA	Online Deterministic Annealing
QoS	Quality of Service
RL	Reinforcement Learning
RSRP	Reference Signal Recieved Power
RSSI	Received Signal Strength Indicator
SA	Stochastic Approximation
SINR	Signal-to-Interference-plus-Noise Ratio
SOM	Self-Organizing Maps
SVM	Support Vector Machine
SVQ	Stochastic Vector Quantization
TD	Temporal Difference
UE	User Equipment
VQ	Vector Quantization

List of Symbols Used

The following symbols will be later used within the body of the thesis.

$\mathcal{P}$ State transition probabilities

$\mathcal{U}$ Set of plausible actions the agent can take

$\mathcal{X}$ Set of plausible states the agent can be in

π Policy assigning the best action to take in a given state

Chapter 1

Introduction

In recent years, advances in Machine Learning (ML) research have led to the transformation of numerous fields in a wide range of industries. These developments are mainly attributed to increases in computational power, the widespread availability of data, and significant algorithmic innovations. As a result, ML has emerged as a powerful and practical tool to address complex problems that were previously difficult or infeasible to solve using traditional approaches. One area that ML has significantly influenced is robotics, enabling a boost in control, decision-making, and perception; pushing the boundaries of what these autonomous systems are capable of.

Among many challenges in robotics, *motion planning* is a central and active area of research. It involves computing safe and feasible trajectories efficiently for an agent to operate under various environmental constraints and task specifications. A particularly promising, yet still underexplored, area is Communication-Aware Motion Planning (CAMP). CAMP extends the motion planning problem by incorporating network constraints. This approach focuses on enhancing the performance and robustness of a robot that communicates over a 5G wireless network. Unlike the traditional motion planning problem, CAMP must consider communication quality, which can vary based on network conditions, robot position, and task demands, all of which can impact the overall system performance.

This thesis will examine the application of Reinforcement Learning (RL) — a family of ML algorithms focused on sequential decision-making — to the CAMP problem. Specifically, the focus lies on controlling a single-agent system operating in real-time, where network requirements are both partially unknown and coupled with the robot's control behavior. The aim is to explore how RL methods can enable the agent to learn control policies that are not only

task-optimal but also communication aware, thereby improving reliability and responsiveness in practical robotic systems.

1.1 Background

CAMP is an emerging research area that has recently attracted growing attention from both the robotics and communications communities. The central goal of CAMP is to design robot trajectories that not only fulfill task-specific objectives — such as moving to the goal with minimum delay without collisions — but also maintain and optimize connectivity to a communication infrastructure throughout the robot's operation [1, 2, 3]. It inherently involves the interplay between two key components that are distinct, yet closely related: *modeling the communication environment* and *trajectory motion planning*. The existing work in the field can be categorized into: (1) the types of network properties being modeled, including whether the communication model is known a priori or inferred, and (2) whether the robot trajectories are fixed in advance or actively optimized with respect to the communication constraints.

From the perspective of *network modeling*, most of the existing literature assumes complete knowledge of the spatial variation in wireless channel characteristics. Several studies employ stochastic models that capture basic signal metrics [1] such as path loss, shadowing, or fading, while neglecting higher-level metrics related to Quality of Service (QoS) [4]. Firstly, *handover modeling* in the presence of multiple base stations is especially critical in wireless networks that utilize high-frequency 5G networks and beamforming, where maintaining seamless connectivity across base stations poses a significant challenge. Furthermore, *throughput* and *latency* are key indicators of communication performance, directly affecting the robot's ability to receive timely commands and transmit sensor data — factors that are essential for safe and effective real-time operation.

Moving on to when a complete wireless model is not available, the works commonly adopt: (a) using Electromagnetic (EM) propagation models without any additional data, or (b) using limited data to estimate network properties. In the latter, data-driven models — namely Support Vector Machines (SVMs) and Gaussian process regression models — from ML have been leveraged [5]. A more recent approach involves the use of a Network Digital Twin (NDT) to model the communication environment [6]. An NDT is a virtual replica of the physical communication network that integrates real-time or simulated data to predict and analyze the network performance. For CAMP, an NDT can be used to anticipate the network changes as the

robot moves through the environment, which can be utilized to enable more informed and adaptive motion planning (top of Figure 1.1).

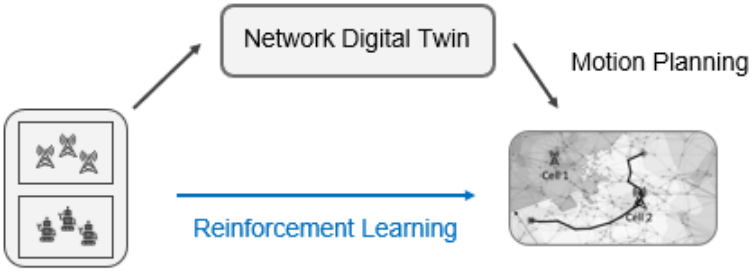

Figure 1.1: NDT vs. RL Motion Planning

As for *motion planning*, the problem is typically framed as a multi-objective optimization task that balances efficiency and safety. This is often approached by either using searching algorithms on a discretized space [7], or by using optimal control methods [2, 8, 9]. Despite these advances, relatively few works explicitly address the challenge of jointly optimizing robot trajectories given the high-level network properties [2, 9], marking it as a key area for further exploration. The difference between ordinary motion planning and its communication-aware counterpart is presented in Figure 1.2.

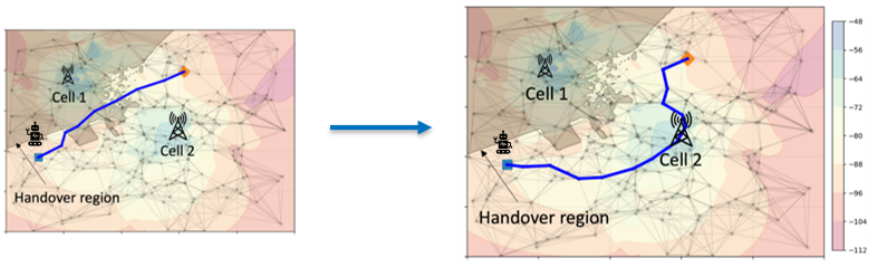

Figure 1.2: Standard vs. Communication-Aware Motion Planning

1.2 Problem

The given problem is that of CAMP, and the central question of this thesis can be summarized as:

How can a robotic agent learn a control policy that simultaneously achieves the task objective and maintains reliable communication when access to the network model is not available and the network properties are uncertain?

1.2.1 Original Problem and Definition

The problem definition involves an autonomous agent that aims to move efficiently and safely from one location to another, while adhering to spatial constraints and considering network communication constraints. Network properties are dynamic, and QoS properties — including handovers, latency, and throughput — can change as network demand changes, which is vital in CAMP, as communication constraints are closely tied to control decisions.

Despite significant advancements in motion planning, most approaches rely on unrealistic design assumptions. They either fully know the network models or use predefined motion plans. Those approaches have a limited adaptability in real-time and uncertain environments, and cannot handle changes in network quality. Therefore, optimizing motion under the QoS properties presents a complex challenge that is not sufficiently addressed in the current literature. This motivates the use of data-driven and adaptive control strategies, where both motion plan and communication behavior can co-evolve based on interaction with the environment.

1.2.2 Scientific and Engineering Issues

The problem definition gives rise to several scientific and engineering issues:

- *Network Accessibility*: In real-life applications, the robot might not have access to an accurate or any network model. Instead, it has to infer network properties.
- *High-Dimensional and Nonlinear Dynamics*: Robotic motion combined with network behavior is often complex and nonlinear, making traditional analytical solutions intractable.
- *Uncertainty*: Learning under uncertainty can lead to issues with convergence, stability, and exploration of the environments, making learning of optimal controls difficult.
- *Intertwined Control-Communication Space*: Network properties are dynamic, and change over time, which in turn would affect the robot's

motion under communication constraints. This creates a feedback loop between the two domains.

- *Real-Time Decisions*: Control policies must operate in real-time, and thus, computational and sample efficiency are key factors.
- *Transfer from Simulation to Reality*: Bridging the gap between the model trained in simulation and its deployment to a real robot with a real, dynamic network is nontrivial.

1.3 Purpose

The purpose of this thesis is to explore how CAMP can enhance the performance, reliability, and intelligence of autonomous systems operating in real-time, wireless environments that utilize 5G networks.

To gain a better understanding of the purpose, it can be helpful to view this project through the lens of an autonomous delivery robot. If the robot needs to move to a specific location (the goal), the shortest path that A* provides may take the robot away from the base station. This might be problematic if the robot requires specific latency to operate, or if the base station provides motion planning, and delays can render the robotic control unsafe. Incorporating network constraints into the motion planning can mitigate these issues.

Therefore, the core aim can be summarized as creating a method that enables mobile robots to plan their movements in a way that considers and adapts to the quality of the wireless network, rather than treating it passively or as a fixed parameter. This is especially relevant to communications companies such as *Ericsson*. As 5G and next-generation networks are deployed to support intelligent industries, connected automation, and mission-critical applications, autonomous agents must operate efficiently, reliably, and safely under any conditions. CAMP directly affects better resource allocation, improves the QoS, and supports EdgeAI, all of which are crucial to Ericsson's vision of industry digitalization and network-aware automation. If the goals of this thesis are met, the results will have a direct impact on communications companies, robotic developers, and industrial users.

1.4 Goals

The goal of this project is to explore, develop, and evaluate hierarchical learning algorithms for describing an NDT and design a RL-based control

framework for autonomous robots to find optimal trajectories under communication constraints. The project has been divided into the following sub-goals:

1. Integrate the motion control objectives and communication constraints, and formulate the problem as an Markov Decision Process (MDP);
2. Conduct an exploration into suitable RL algorithms which would be suitable for real-time decision-making in the CAMP setting; and
3. Build a simulated environment where network properties vary spatially and temporally, where the algorithms will be compared and evaluated.

1.5 Research Methodology

This thesis employs a mixed-methods approach, combining both quantitative and qualitative analysis. The former is mainly used to evaluate and compare the performance of various ML algorithms using objective metrics. At the same time, the latter would primarily be used to assess how effective those algorithms are when applied to the CAMP problem by analyzing the behavior of the learned agent. Therefore, it will follow a pragmatic research philosophy, meaning that the choice of methods will be based on what best helps to answer the research question. This aligns with the project's approach, which involves a bottom-up strategy, starting with RL algorithms and then selecting appropriate ones to address the CAMP problem.

Focusing solely on a quantitative approach could result in capturing performance metrics but overlooking the reasons why certain behaviors emerge or how well the system generalizes, both of which are crucial for safe and reliable autonomous systems. Similarly, relying purely on qualitative methods would lack reproduction and measurable evaluation. These considerations are significant for CAMP, where optimal controls emerge from learning through interaction with the environment, and robot-network interaction is not easily measurable or modelable.

1.6 Delimitations

There are a couple of delimitations in this thesis project that need to be outlined:

- *Multi-Agent Experiments*: There is a possibility to extend the problem to multiple agents, which makes the situation much harder as each robot affects the networks, and it requires multi-agent motion planning.

- *Simulation to Real-life*: Deploying the algorithm and conducting tests on a real-life robot is desirable, but it was decided to be done only if there is time. Likewise, the low-level controls, robot dynamics, and physical motion execution are assumed to be handled by other libraries.
- *Comparison to other methods*: As CAMP research is relatively new, with not a lot of comparable methods, and original datasets provided by Ericsson, doing a proper method comparison might be infeasible.

All of the points mentioned above are valuable future research thrusts; however, they currently lie outside the scope of the project due to complexity and time constraints.

1.7 Report Structure

Chapter 2 presents relevant background information about motion planning, RL algorithms, and Vector Quantization (VQ). Likewise, it summarizes the relevant works conducted in network-aware robotics and hybrid RL. Chapter 3 gives an overview of the research methods used throughout this research project. Chapter 4 discusses the methodology and implementation of the hybrid Q-learning with VQ algorithm. Chapter 5 summarizes the major results from the experiments and analyzes them in the context of reliability and validity. Finally, Chapter 6 concludes the project by evaluating the results in terms of the original objectives, provides an overview of future work, and reflects on sustainability, ethics, and societal impacts.

Chapter 2

Background

This chapter provides basic background information about three major topics related to this thesis: (1) CAMP, (2) RL, and (3) VQ methods. In (1), background on the communication-aware robotics will be given, along with an understanding of motion planning. Afterward, (2) will cover the major theory behind RL, the major methods used, and open problems related to this thesis. Understanding (2) will help understand the importance of (3), as combining the two will lead to a hybrid RL approach. After covering those major areas, the most relevant works will be summarized.

2.1 Communication-Aware Motion Planning

This section will set the stage for understanding CAMP by explaining the *motion planning* problem and providing an overview of wireless communication basics. Afterward, in Section 2.4.1, relevant works will be analyzed on communication-aware robotics.

2.1.1 Motion Planning

Traditionally, **motion planning** is a computational process of finding a feasible path (trajectory) for an autonomous agent from an initial state to a specific goal, while adhering to a set of constraints. These can include *collision avoidance*, *kinematics*, *computational complexity*, and *uncertainty*. At its core, the goal of motion planning is to make the operation safe, efficient, and feasible.

Formally, the motion planning problem can be defined as follows: Given a movable autonomous agent with an initial state q_{init} and a goal state q_{end} in a configuration space $\mathcal{C}$ (manifold representing all possible robot

configurations), along with the description of the environment, the goal is to find a continuous path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ with $\tau(0) = q_{init}$ and $\tau(1) = q_{end}$. [10] The major constraint is for the entire path to lie within the free configuration space (C_{free}), where $\mathcal{C}_{free} \subset \mathcal{C}$ and it captures the space without obstacles. The resulting path is therefore a geometric curve in C_{free}, and the trajectory also incorporates time, velocity, and acceleration.

The two key motion planning variants are global and local. The first is done when there is full environmental knowledge, whereas the other is reactive. The most common approaches for solving the global motion planning problem are: (1) *grid-based*, (2) *sampling-based*, and (3) *optimization-based* planners. (1) discretizes $\mathcal{C}$ into a graph and uses shortest-path searches (like Dijkstra or A*). Under certain conditions, optimality is guaranteed, but it suffers from the curse of dimensionality. (2) randomly samples configurations - with approaches like Rapidly-exploring Random Trees or Probabilistic Roadmaps - and avoids explicit discretization. When compared to (1), it scales better with higher dimensions, but does not provide optimality and often requires smoothing. Finally, (3) formulates the problem as a constrained optimization problem, with the goal of minimizing a cost function subject to the given spatial constraints. This approach typically yields optimal local trajectories, but it is computationally expensive for complex environments.

When motion planning must also incorporate **communication constraints**, the feasible configuration space C_{free} becomes dependent on both the *obstacles* and *communication quality maps*. This leads to an intertwined problem of balancing optimal travel with connectivity maintenance. Section 2.4 will summarize and analyze the works that delve into solving this problem.

2.1.2 Wireless Communication Basics

Within the configuration space, electromagnetic signals are transmitted carrying information communicated through the network. The network is responsible for immediate and fast connectivity and the maximization of the communication quality of the users. Among the various communication quality metrics that can be measured in a wireless 5G/6G network, the most important in communication-aware cyber-physical systems and IoT applications are signal strength, user latency, channel throughput, and communication delay due to handovers between cells [11].

Signal Strength

Signal strength comprises mainly three components: *path loss* (low-

frequency decay in the channel power with distance), *shadowing* (high-frequency fading mainly due to obstructing structures), and *multi-path fading* (very high-frequency attenuation due to scattering and reflection), given the locations of the base stations [1]. Signal strength can be measured by numerous metrics, including the Reference Signal Recieved Power (RSRP) and the Signal-to-Interference-plus-Noise Ratio (SINR) observed by the user [12].

Latency and Channel Throughput:

Throughput represents the actual rate at which user data is successfully transmitted over a wireless channel and depends on SINR, allocated bandwidth, and the radio access technology (e.g., code rate, modulation, and waveform). Latency is the time delay experienced for a data packet to travel from the source to the destination across the network, including transmission delay, processing delay, propagation delay, and queuing delay. Latency is also affected by SINR, network congestion, and radio access technology (e.g., coding, modulation, beamforming, etc.) [12].

Handover Delays

A handover is the process of transferring an active connection (voice or data) from one base station to another, maintaining service continuity as the user moves or radio conditions change. A handover is triggered when the User Equipment (UE) detects that the signal quality (e.g., RSRP or SINR) from a neighboring cell becomes better than that of the serving cell over a time window, or when network conditions require transferring the connection to maintain service quality, or based on the predictions of the signal quality (see e.g. [13]). Handovers can increase packet latency and delay due to the time required for signaling, context transfer, and resource reallocation between cells during the connection switch.

2.2 Reinforcement Learning

RL is a framework within ML which aims to teach an agent to make decisions through interaction with the environment. In comparison to supervised learning, where the input-label pairs are provided a priori, RL relies on the feedback from the environment in the form of costs (or rewards) to guide the learning process. The agent tries to learn a policy (π), that maps states ($x \in \mathcal{X}$)

to actions ($u \in \mathcal{U}$):

$$\pi : \mathcal{X} \to \mathcal{U},$$

such that it performs optimally over time in accordance with a specific objective.

This framework is closely connected to the field of optimal control, where the aim is to find control strategies that minimize the cumulative cost (or maximize cumulative reward) in dynamical systems. This is especially true when the environment is modeled as a MDP.

2.2.1 Markov Decision Process

MDP $(\mathcal{X}, \mathcal{U}, \mathcal{P}, C)$ is a mathematical framework used to capture the notion of sequential decision-making under uncertainty formally. The underlying Markov property, which specifies that the future depends only on the current state and action, opens up the possibility for RL algorithms to use value functions and principles from dynamic programming. Specifically, they can leverage Bellman's optimality equations to learn optimal behaviors.

2.2.2 Bellman's Principle of Optimality

Bellman's Principle of Optimality lies at the heart of RL and optimal control. It states that the optimal solution is built from optimal solutions of smaller subproblems, and thus, guarantees that the best resulting path includes the best paths to any intermediate points [14].

As a result, this leads to the Bellman's equation for the value function $V^*(x)$, which provides the minimum expected cost from the current state x_i and by taking optimal actions thereafter:

$$V^*(x_k) = \min_{u \in \mathcal{U}} \Big[C(x_k, u) + \gamma \mathbb{E}_{x_{k+1}} [V^* | x_k, u] \Big], \tag{2.1}$$

where $\gamma \in [0, 1)$ is a discount factor used to encourage the agent to prioritize immediate rewards rather than future rewards. Given the $V^*(x)$, the optimal policy can be derived as:

$$\begin{aligned} \pi^*(x_k) &= \arg\min_{u \in \mathcal{U}} \Big[C(x_k, u) + \gamma \mathbb{E}_{x_{k+1}} [V^* | x_k, u] \Big] \\ &= \arg\min_{u \in \mathcal{U}} \Big[V^*(x_k) \Big] \end{aligned} \tag{2.2}$$

Unlike classical optimal control, where to solve this problem full

knowledge of dynamics and cost function are required, RL method allows the agent to learn what the value (or action-value) function is by gathering experience through interaction with the environment.

2.2.3 Methods

For this thesis, we will consider *Model-Free* RL, specifically making a distinction between *Value*- and *Policy*-based methods.

Value-based methods usually store the value estimates (either $V(x)$ or $Q(x, u)$) in a tabular form and then use them to find the optimal actions via value minimization. The most common methods are SARSA and Q-learning. They are commonly used for discrete state-action spaces, and do not work well with large or continuous spaces. However, as they use Bellman updates, they can guarantee global convergence for discrete spaces. This stems from the Stochastic Approximation (SA) theorem [15], which states that the update sequence (produced by algorithms like Q-learning or SARSA):

$$x_{n+1} = x_n + \alpha(n)[h(x_n) + M_{n+1}], n \geq 0, \tag{2.3}$$

almost surely converge to the solution of the invariant set of the Ordinary Differential Equations (ODEs):

$$\dot{x}(t) = h(x(t)), t \geq 0, \tag{2.4}$$

provided specific conditions are met. In other words, despite having noisy updates, the algorithm will still be guided by the underlying dynamics of the value function. With enough time, it will stabilize around an optimal solution (characterized by the invariant set of the ODEs).

On the other hand, policy-based methods generally store policy parameters, which are then adjusted via gradient descent; that is, the policy is directly learned and optimized. The most common approaches include REINFORCE, Actor-Critic methods, and Deep Q-learning. These approaches are better suited for continuous spaces, but they have some limitations. Firstly, they get stuck in local minima, thus no longer giving the global convergence guarantees we saw before. Secondly, the approaches utilize parametrized models, which have lower explainability and are more computationally intensive. Finally, they depend heavily on initialization, hyperparameters, and exploration, making training them more complex.

2.2.4 Q-Learning

Q-learning is one of the most famous value-based RL algorithms. It is used to find optimal policies by learning the action-value function $Q^*(x, u)$, and to solve the MDP. At every step of the training process, the agent observes a state x_k within the environment, and decides on which action u to take. Thereafter, the environment returns the cost $C(x_k, u)$ and the new state x_{k+1}. Given all the components, Bellman's optimality equation is used to update the action-value function:

$$Q(x_k, u) \leftarrow Q(x_k, u) + \alpha\big[C(x_k, u) + \gamma \min_{u'} Q(x_{k+1}, u') - Q(x, u)\big], \quad (2.5)$$

where α is the learning rate. Over time, the algorithm learns to approximate $Q^*(x, u)$, and Equation 2.2 can be used to get the optimal policy. The update function belongs to the family of Temporal Difference (TD) learning algorithms, where updates are performed using the current value estimated to update itself, making training more data-efficient, as there is no need to rely on the entire episode.

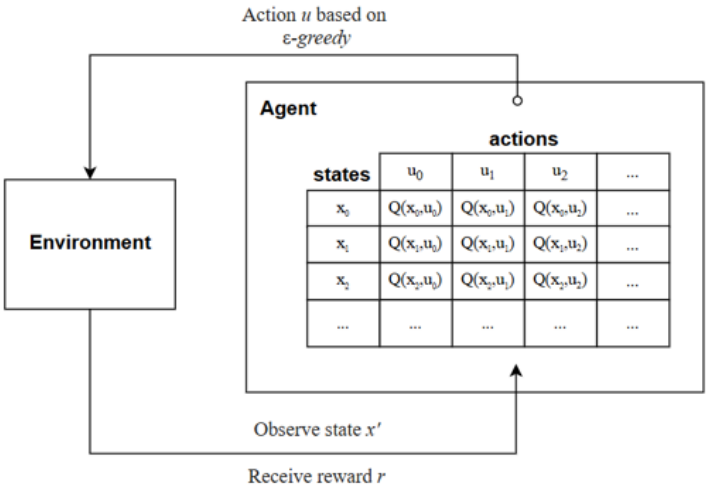

Figure 2.1: Q-learning training loop

Furthermore, it is an *off-policy* algorithm, meaning it learns the optimal policy independently of the agent's current behavior. This allows for effective exploration strategies, notably the ϵ-greedy approach, where the agent starts with a high probability of taking random actions. Still, over time, it will begin to focus on selecting the best actions. This balances the concepts of exploration

(testing new actions) and exploitation (taking optimal actions) throughout the training.

Finally, it performs iterative updates in the form of Equation 2.3, and thus, under suitable conditions (including decreasing α and sufficient exploration), it is a SA algorithm. Therefore, it converges to a fixed point of the Bellman operator, which gives strong theoretical convergence properties for discrete data. However, extending it to continuous data poses a major open problem to value-based algorithms, due to the curse of dimensionality. The question then becomes, *is it possible to retain the properties that Q-learning provides, and apply it to continuous data*? Q-learning has served as a basis for other, more advanced algorithms, such as Deep Q-learning (DQN), which work with higher-dimensional and continuous data. However, in those settings, we no longer get convergence guarantees. Another approach is to discretize (bin) the space, and if done in a "smart" way, this could help alleviate the issues of high dimensionality. The following section will provide a detailed discussion of this concept.

2.3 Vector Quantization

The most primitive type of clustering in space is through *ad hoc discretization* - breaking the space into a set of uniform bins. Essentially, the space is split into a grid of *codevectors*, where each state will be represented by the nearest codevector. This simplifies the complexity of the original MDP, making it more computationally tractable. However, to get a good representation of a continuous space, very high resolution is often required, which leads to the curse of dimensionality. Furthermore, in various tasks, certain states require more resolution than others. We will see this in action when analyzing the results in Chapter 5. Consequently, we will look into more adaptive forms of binning the space by utilizing the data.

VQ is a form of clustering that is used to reduce the complexity of large or continuous state spaces using a finite set of representative prototypes, referred to as *codewords* ($\mu_i \in \mathcal{X}$). This is a data-driven approach that abstractly bins similar states together (see Figure 2.2). The states in which we are interested are the *latent system modes*. Each codevector creates its own Voronoi partition ($S_i \in \mathcal{X}$), which divides the space into regions based on proximity to that codevector.

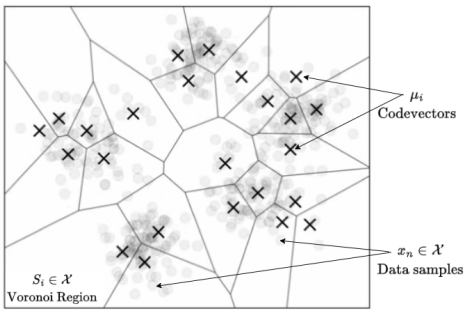

Figure 2.2: State Aggregation

2.3.1 Hybrid RL

A *hybrid RL* approach combines model-free control optimization with unsupervised clustering. In a complex decision-making system, it allows the agent to learn the control policy (*control actions*) in parallel with a representation of the environment's underlying structure (*system modes*).

In theory, running two independent SA learning processes can lead to convergence of both algorithms, if they evolve at different rates [15]. The idea behind it is to update the "fast" component more aggressively by treating the other component as nearly static, and update the "slow" component gradually by treating the other component as close to equilibrium. This allows the RL agent to adapt rapidly to a temporarily fixed mode partition, and the clustering adapts slowly based on long-term statistics of state visits. The separation is crucial when there is a variable dependency between the two processes. SA in two-timescales can be summarized mathematically as (see Section 2.2.3 for explanation about SA):

$$\begin{aligned} \mu_{t+1} &= \mu_t + \beta(t)\left[g(\theta_t, \mu_t) + M_{t+1}^{(\mu)}\right] \\ \theta_{t+1} &= \theta_t + \alpha(t)\left[f(\theta_t, \mu_t) + M_{t+1}^{(\theta)}\right] \end{aligned} \tag{2.6}$$

where the step sizes balance the process with $\frac{\beta(t)}{\alpha(t)} \to 0$ as $t \to \infty$[16].

In summary, hybrid RL, which utilizes VQ on two timescales, provides a method for learning complex, multimodal environments where the mode boundaries are not known a priori. The synergy between Q-learning and VQ methods ensures stability, convergence, interpretability, and adaptability.

2.3.2 Methods

This section provides an overview of the most common VQ methods, along with their benefits and limitations in relation to the Hybrid RL approach.

K-Means

K-means is a method that groups data points into K clusters by placing each point into the group with the nearest center. Formally, it aims to find a set of K codevectors $\{\mu_1, ..., \mu_K\}$ that minimizes the within-cluster variance using the following objective:

$$\min_{M,V} J(M) := \mathbb{E}\big[\min_i d(\mathcal{X}, \mu_i)\big], \tag{2.7}$$

where d is the divergence metric, $M = \{\mu_i\}_{i=1}^K$ is the set of codevectors, and $V = \{S_i\}_{i=1}^K$ is the set of Voronoi partitions [17]. For K-means, d is most commonly the Euclidean distance, and so the objective can be written as:

$$J = \sum_{n=1}^{N} \min_i ||x_n - \mu_i||^2. \tag{2.8}$$

The algorithm works with batch updates, where firstly all of the points are assigned an index (s_n) of the closest centroid to x_n:

$$s_n = \arg\min_i ||x_n - \mu_k||^2. \tag{2.9}$$

After all of the points are assigned, the codevectors are updated:

$$\mu_k \leftarrow \frac{1}{|S_k|} \sum_{x_n \in S_k} x_n, \tag{2.10}$$

where S_k is the set of all points assigned to the cluster k during the assignment step. The updates are done until the algorithm converges.

K-means is efficient; however, it is an offline algorithm (batch-oriented) and it assumes a fixed distribution, which makes it inherently unsuitable for continuous and incremental updates. Furthermore, it requires a pre-defined K (number of codevectors) with sensitivity to initialization. This is a problem that we will encounter repeatedly, and it is imperative because the number of clusters is task- and data-dependent (see Chapter 5). Finally, it is prone to getting stuck in local minima due to its nonconvex nature.

Stochastic Vector Quantization

Stochastic Vector Quantization (SVQ) is an iterative SA method that randomly adjusts the codevectors using the data. For a new observation $\boldsymbol{x}$, the nearest centroid is $k^* = \arg\min_k ||\boldsymbol{x} - \mu_k||^2$ and the codevector update is:

$$\mu_{k^*} \leftarrow \mu_{k^*} - \eta(t)\nabla_{\mu_{k^*}} d_\phi(\boldsymbol{x}, \mu_{k^*}), \tag{2.11}$$

where $\eta(t) > 0$ is the decreasing learning rate, and $d_\phi(\cdot, \cdot)$ is the Bregman divergence. The Bregman divergence is defined as a set of divergences which can be summarized as: $d_\phi(x, y) = \phi(x) - \phi(y) - \langle \nabla\phi(y), x - y \rangle$, where ϕ is a strictly convex function, and the update can be derived from minimizing this divergence [18]. Squared Euclidean distance is the most basic form of Bregman divergence. The gradient of the divergence is used to provide the direction of the steepest descent, guiding the update of codevectors where the divergence is minimized between a point $\boldsymbol{x}$ and the winning codevector μ_{k^*}.

In comparison to K-means, this algorithm has incremental stochastic updates that integrate naturally into the Hybrid RL framework. Furthermore, it is also not constrained to only Euclidean distance, but the convergence is guaranteed for all Bregman divergences. The importance of this will become apparent later when we discuss the use of divergences other than the Euclidean distance. However, because we focus on updating a single point at a time, convergence is very slow, the points oscillate significantly, and the training is sensitive to the learning rate η. Finally, just like with K-means, it also requires a predefined set of codevectors with good initialization.

Self-Organizing Maps

Self-Organizing Maps (SOM) is a method that iteratively groups the data by moving nearby neighbors together, instead of a single point. The neighborhood is defined as a set of codevectors arranged in a predefined low-dimensional lattice, commonly a grid, and updates not only the winner but also its neighbors.

The update process is formalized as:

$$\mu_i^{t+1} = \mu_i^t - \eta(t)G\big(h^t, i, \sigma(t)\big)\nabla_{\mu_i} d_\phi\big(\boldsymbol{x}, \mu_i^t\big), \tag{2.12}$$

where t defines the timestep, η is the learning rate, and h^t is the winning index

at timestep t. G is the Gaussian neighborhood function defined as:

$$G(x, m, \sigma) = \frac{1}{\sqrt{(2\pi)^k |\sigma|}} e^{-\frac{1}{2}(x-m)^T \sigma^{-1}(x-m)}, \tag{2.13}$$

with m and σ being the mean and covariance of the neighborhood, respectively. As a result, the mechanism ensures that the winner moves the most towards the new sample point. Based on the location of other codevectors in the predefined lattice, the closer neighbors will move more, while the further ones will move less and less.

The SOM is an extension of SVQ, which is also a SA algorithm, but additionally, it captures the topological relationships in the data. As this is no longer a winner-takes-all scenario, the result is smoother transitions between modes, greater robustness to noise, and faster convergence. Some problems are that it is challenging to find an optimal lattice shape, as small maps can underfit the data, and large maps can overfit the data, creating "dead neurons" (codevectors that are never used) or topological distortions. Moreover, adding the neighborhood function with σ introduces an additional hyperparameter that needs to be fine-tuned, thereby making training more challenging. If the parameter shrinks too quickly, the map will fail to organize globally, whereas if it shrinks too slowly, the local fine-tuning will be poorly done.

Online Deterministic Annealing

Online Deterministic Annealing (ODA) is also an online, prototype-based learning algorithm that can be used for clustering. It is an extension of the traditional Deterministic Annealing (DA), where SA techniques are employed to facilitate incremental updates and prevent local minima. The algorithm can be viewed as a progressively growing competitive learning Neural Network (NN) architecture, where the codevectors are dynamically added and refined through a controlled annealing process. The annealing process starts with a high-entropy state that is robust to initialization and over time transitions to a low-entropy state that aims to minimize distortion; thus, it simulates the physical annealing process of gradual "cooling" of the system.

ODA takes the objective of the traditional hard-clustering VQ from Equation 2.7, and relaxes it into a soft-clustering framework. The former assigns each of the data points to an exclusive cluster, whereas the latter allows the data point to belong to several clusters with varying association probabilities $p(\mu_i|X)$. With that, the *distortion* can be redefined as the

expected value over those probabilities:

$$D := \mathbb{E}\big[\sum_i p(\mu_i|X) d_\phi(X, \mu_i)\big], \tag{2.14}$$

with a random data point X.

The key innovation of the method, and the main objective function that the algorithm optimizes, is the following Lagrangian function:

$$F_T = D - TH, \tag{2.15}$$

with temperature $T > 0$ (acts as a Lagrange multiplier), and H is the Shannon entropy:

$$H = \mathbb{E}\big[-\log p(X, Q)\big] = H(X) - \int p(x) \sum_i p(\mu_i|x) \log p(\mu_i, x) dx, \tag{2.16}$$

with a random variable Q associated with the assignment of data points to codevectors.

When T is high, entropy maximization dominates the optimization objective, leading to uniform associations and a convex optimization landscape with a unique global minimum—the mean of the dataset. As T is decreased during training, distortion minimization begins to take priority, leading to phase transitions through the process of bifurcation - the splitting of codevectors. This annealing process helps create adaptive resolution based on the data, and escape local minima, as the codevectors converged on previous temperatures, which leads to good initialization for the lower temperatures.

Just like SVQ and SOM, this method uses Bregman divergences as the distortion measure, which incorporates common distance metrics and enables gradient-free updates. Solving for the optimal association probabilities yields the Gibbs distribution:

$$p(\mu_i|x) = \frac{p(\mu_i) e^{-d_\phi(x, \mu_i)/T}}{\sum_j p(\mu_j) e^{-d_\phi(x, \mu_j)/T}}, \tag{2.17}$$

where $p(\mu_i)$ is the marginal probability of the codevector. On the other hand, solving for optimal codevector locations is conditional on expectations (assuming Bregman divergence is used):

$$\mu_i^* = \mathbb{E}\big[X|\mu_i\big] = \frac{\int x p(x) p(\mu_i|x) dx}{p(\mu_i)}. \tag{2.18}$$

These can be approximated stochastically as the algorithm works in an online setting, following **Theorem 4** from [17]. It can be proved that the random variable $m_n = \frac{\sigma_n}{\rho_n}$ almost surely converges to $\mathbb{E}\big[X|\mu\big]$ with:

$$\sigma_{n+1} = \sigma_n + \alpha(n)\big[x_n p(\mu|x_n) - \rho_n\big], \tag{2.19}$$
$$\rho_{n+1} = \rho_n + \alpha(n)\big[p(\mu|x_n) - \rho_n\big], \tag{2.20}$$

given that $\sum_n \alpha(n) = \infty$ and $\sum_n \alpha(n)^2 < \infty$.

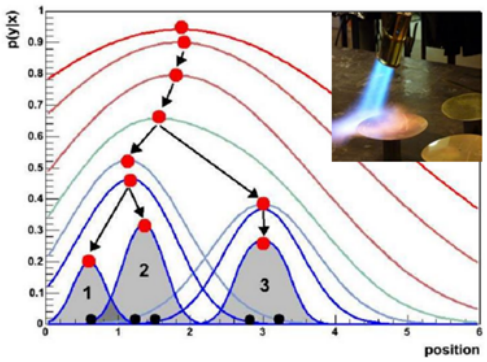

Figure 2.3: Bifurcation process as temperature drops in ODA [19]*

The algorithm begins with a single codevector ($K = 1$), and some initial temperature $T = T_{max}$. The annealing loop resets when the algorithm with the current codevector(s) converges, i.e., the codevector updates are too small or non-existent. At the start of each loop, the codevectors are split and randomly perturbed as $\mu \sim \{\mu + \delta, \mu - \delta\}$. As the data samples arrive, the association probability for each codevector will be computed, and they will be updated given Equations 2.17 and 2.19. At the end of the annealing loop, if the critical temperature is reached, where there is a phase shift, the codevectors will have moved away from each other - imitating a bifurcation process (see Figure 2.3). Otherwise, they will be merged back together. At the very end, the temperature will be decreased (cooling), and the loop will be reset. The whole algorithm will terminate globally if a predefined temperature threshold (T_{min}) or the maximum number of codevectors (K_{max}) is reached.

This process ensures convergence to a true density in clustering. The online nature of the algorithm, combined with the annealing process, provides explainability, adaptive and localized resolution, limited hyperparameter

* used with permission from the author

tuning, and robustness. Section 2.4.3 will show how all of these methods work on simulated RL tasks, and how ODA outperforms them all.

2.4 Related Work

Having done an overview of the background in the previous section, the most relevant works can be discussed. The first work will provide an overview of the advances made in communication-aware robotics, which will help in understanding the history of approaches used in the field. The second examines data-driven methods from ML that are employed when the full knowledge of the wireless model is not available. The final work will investigate a method that enables optimal action control and mode estimation to operate in parallel.

2.4.1 Network-Aware Robotics

To recap, CAMP is an extension of ordinary motion planning (see Section 2.1.1), where the objective integrates communication constraints into the path planning process. The field has advanced significantly, starting with simple models and evolving into more sophisticated frameworks. The work by [1] conducts a comprehensive overview of the evolution of this field, highlighting how communication-aware planning methods have progressed over time.

Early research in multi-robot systems and autonomous navigation prioritized path planning, sensing, and distributed decision-making. However, it either assumed that the network was always available or could be modeled using simplistic abstractions. Common assumptions included disk-based connectivity models or path-loss-only channel models. They are computationally efficient, but they fail to capture spatial variations (caused by shadowing or multi-path fading), which has been demonstrated through empirical wireless channel measurements in realistic environments.

More recently, these limitations led to the emergence of communication-aware robotics, which considers communication quality explicitly during planning. Several works tackling this problem assume a wireless model is available a priori and plan robot trajectories accordingly. Meanwhile, other works include probabilistic channel models, utilizing techniques like the Gaussian processes or stochastic channel maps. Those are used to predict expected link quality at previously unseen locations. For example, path planning to establish connectivity can be formulated as an optimization problem like the Min-Exp-Dist-Path, to minimize the expected distance

traveled until a connected spot is discovered (subject to graph-based constraints). These are NP-hard problems, prompting the use of methods from game theory, such as log-linear learning on potential games. Those yield asymptotically ϵ-suboptimal solutions by converging to Nash equilibria.

Despite the advances in the field, a lot of these approaches have to make strong assumptions: either (1) the network model is known a priori; (2) it can be accurately learned offline; or (3) the optimization is decoupled, meaning the trajectories are not included in learning or refining of the communication model. In reality, acquiring dense and accurate measurements to create an accurate network model is costly, time-consuming, and can be simply infeasible. Therefore, network models based solely on limited data are often not accurate, resulting in a degraded optimization process. Moreover, there is not a lot of research into explicitly incorporating meaningful wireless QoS constraints to optimize robot trajectories, which are crucial for this project (see Section 1.1). Most of the existing methods rely only on proxy metrics, such as the signal strength or connectivity indicators; however, those do not directly or accurately reflect end-to-end communication performance.

Consequently, the joint problem of optimizing robot trajectories while actively estimating and refining the network model using QoS constraints remains largely unexplored. This project addresses this gap by coupling motion planning and network information into a single problem, enabling robots to produce trajectories that are communication-informed. In turn, the proposed method could advance beyond the assumption of static or fully known network models, creating a more adaptive and practical method.

2.4.2 Machine Learning Classifiers for Communication-Aware Robotics

Other recent works have investigated the use of data-driven approaches from ML for CAMP. Specifically, [5] predicted and optimized wireless signal propagation, namely Received Signal Strength Indicator (RSSI) collected from several base transceiver stations, and applied several ML classifiers to construct an initial propagation model and incrementally update it over time. The paper evaluated those methods based on their Root Mean Squared Error (MSE), MSE, Mean Absolute Error (MAE), and R-squared.

The following list is a summary of the methods that the authors evaluated:

1. *Linear Regression*: Fits a linear model intending to minimize Root MSE; suitable for doing basic predictions.

2. *SVM*: Uses hyperplanes and kernels to perform non-linear regression.

3. *Ensemble Boosted Trees*: Employs weighted ensembles and applies boosting for improved accuracy.

4. *Gaussian Process Regression model*: Utilizes the Bayesian non-parametric method to estimate a probabilistic function.

The results showed that the *Gaussian Process* achieved the best performance, with the lowest Root-Mean MSE, MSE, and MAE, along with the highest R-squared value. Nevertheless, this study relies solely on a single networking constraint and does not incorporate some critical QoS metrics, such as handovers. Furthermore, these models are still limited by the prior information they can utilize, and they do not scale well with incremental updates. Therefore, creating a **resource-efficient** network model adaptation scheme remains an open problem with significant implications.

2.4.3 Q-learning with Online Deterministic Annealing

To create a hierarchical learning algorithm to describe an NDT and design a RL algorithm to find optimal trajectories (see Section 1.4), a highly relevant work to consider is [20]. The work proposes using ODA (as an adaptive state aggregation scheme described in Section 2.3.2) in conjunction with Q-learning (a TD RL algorithm described in Section 2.2). The resulting hybrid algorithm forms a SA algorithm with two components running at different timescales: (1) RL running as the fast component, and (2) VQ running as the slow component (see Section 2.3 for more details). The study employs Bregman divergences as the dissimilarity measure for (2) to increase efficiency and reduce computational cost.

The method was evaluated in *CartPole*, which is a classic simulated benchmark problem used to test RL algorithms. The goal is to apply discrete forces (right or left) to a cart moving along the tracks without dropping a pole. The algorithm must take into account factors such as the position and velocity of the cart, as well as the angle and angular velocity of the pole. The number of episodes quantifies the agent's performance in balancing the pole. The authors quantified the performance of ODA with other algorithms from Section 2.3.2, along with a simple binning of the state space (no state aggregation) [20, 21].

In addition to the benefits that ODA provides, such as the adaptive number of codevectors and adaptive resolution, the results showed that it significantly outperformed any other method, yielding very positive results. However, one

downside of this research is that it only focused on comparing different state aggregation methods. Still, it would be interesting to see, performance-wise, how it compares to other RL algorithms, such as Actor-Critic methods or Deep Q-learning.

2.5 Summary

This chapter provides a background overview of the central concepts covered in this thesis, including motion planning, RL, and VQ. As shown throughout Sections 2.4.1 and 2.4.2, the current research lacks a significant amount of work that addresses the joint problem of motion planning and network model estimation. Moreover, different works utilize different network data, resulting in varied measurements and metrics. When the wireless model of the network is not fully available, most works utilize either probability density models (Section 2.4.1) or Gaussian Process regression models (Section 2.4.2). Finally, Section 2.4.3 is highly relevant because this project will be a natural extension of that research, where it will be applied to motion planning and CAMP. Furthermore, it contains the baseline performances of different algorithms on CartPole, which will be helpful for evaluation.

Chapter 3

Methods

This chapter provides an overview of the research methods used throughout this thesis project. Section 3.1 summarizes the research process and explains the research paradigm used in this project. Section 3.2 explains the pre-existing dataset provided by Ericsson. Section 3.3 dives into the experimental design, along with hypotheses, test environments, and software/hardware requirements. Section 3.4 looks at the reliability and validity of both the data and the method for the problem of CAMP. Section 3.5 summarizes how the dataset will be evaluated and processed during the experiments. Finally, Section 3.6 provides an overview of the system, which is followed by the implementation phase in Chapter 4.

3.1 Research Process and Paradigm

The research process is split into several steps. Most commonly, the first step in the research process is to identify the problem; however, this is an industrial thesis, and the problem definition has already been provided. Therefore, after understanding the given situation, the next step is to evaluate the literature. In particular, this involved an in-depth understanding of the methods which will be used (see Chapter 2). This step is crucial because, during the implementation stage, it is necessary to understand all these methods to implement and compare them. Using the knowledge from the literature review, hypotheses could be drawn about how this approach can work and what the best way to approach the given problem is. The implementation stage itself was done using the *bottom-up approach*, i.e., starting with the creation of the most basic components and then utilizing them to build up a method. In the context of this thesis, this meant building up the hybrid RL

method with different VQ methods. The aim is to experiment and test the methods in a simple simulation first (like in [20, 21]), before applying them to the problem of CAMP. The collected results—quantitative for CartPole and primarily qualitative for CAMP—can be analyzed and evaluated. Afterward, having verified the performance in a simulated CAMP environment, there is an option to deploy it on a real robot for testing and comparison with other methods. Finally, the results and findings are to be reported in the thesis project, and, additionally, in a research paper if interesting or essential findings are discovered.

This project adopts a pragmatic research paradigm, which entails that, rather than being tied to a single ontology, different approaches may be helpful depending on the context. It prioritizes practical problem-solving over strict adherence to a single method. Therefore, it accepts that a research question may require a combination of qualitative and quantitative approaches to yield the most appropriate insights; it is concerned with what works. This leads to selecting the most suitable method for different phases of the research process, in comparison to strategies that focus on method purity.

In the context of the thesis, the initial phase will be quantitatively testing the methods in a simulated environment, CartPole. Specifically, the agent's performance will be measured by how long it can balance a pole, and the resource efficiency by how well the model performs given the number of state codevectors. The reason for using the CartPole environment is two-fold. Firstly, it will help to objectively test and compare various methods in a controlled, replicable environment. This will yield quantitative results for the methods' performance that cannot be as easily obtained through evaluating trajectories in the motion planning environment. Secondly, this project also leans into the exploration of state aggregation algorithms, and thus, using an environment that is consistent with the literature will help evaluate the algorithms and will allow us to make direct comparisons to the literature. This is a positivist-inspired element that aims to help build up the methods and facilitate data-driven selection of the most promising method. Subsequently, the application to CAMP shifts the process towards the mix of qualitative and quantitative methods. The majority of the analysis will be done by visually inspecting the resulting trajectories from motion planning. Nevertheless, a method will be used to quantify the differences between trajectories relatively. These will help to capture the emergent phenomena that focusing on just a single method could overlook. By combining different approaches, the pragmatic paradigm will ensure a holistic evaluation, helping to bridge the gap between theory and practical implementations while enhancing the validity

and usefulness of the findings. To summarize, this paradigm aligns well with the evolving nature of the research process and the bottom-up approach.

3.2 Data Collection

This study utilizes pre-existing datasets provided by the hosting company, Ericsson, rather than conducting its own data collection. The company collected the datasets for research purposes related to communication-aware robotics. Access to this data was granted under a non-disclosure agreement (NDA) and in compliance with ethical guidelines. The data encompasses a map area with spatial signal strength and handover regions, collected via logging tools on a robot. The data was preprocessed by the researchers for use within Python's Gymnasium library. Utilizing proprietary data aligns with the industrial context of this thesis and allows for efficient analysis without the need for intensive data-gathering fieldwork.

3.2.1 Sampling

To sample the data, the robot would drive around the map, with the ability to connect to two base stations. The two base stations were only connected to the robot collecting the data to employ more deterministic data collection. The data collection was conducted using a Husky robot, as it is the most common robot with which the Ericsson Research team works in their research. After running across the map for each point in space, an average of both signal strength and handover regions was computed. This sampling approach is non-probabilistic, ensuring depth and applicability over random selection.

3.2.2 Sample Size

There are approximately 6 pre-processed scenarios, comprising handover and signal strength datasets, where each has 200×200 points spread across a map to create a continuous vector field.

3.2.3 Target Population

The target population encompasses autonomous robotic systems deployed in areas where a wireless connection is essential for operation. This encompasses industrial warehouses or urban delivery robots, where wireless conditions can vary. The provided dataset provides a representation of this population by

using a simplified, static environment. Training on the given data should result in good generalization to the target population in terms of spatial network variability. However, it might not extend very well to real-world, dynamic cases where there are additional temporal variations.

3.3 Experimental design and Planned Measurements

The experimental design for this project is split into two phases to address the objectives of CAMP: a simulation phase followed by a real-world application phase. The first phase is a controlled experimental setup to quantitatively evaluate multiple algorithms, using a factorial design to test various independent variables. The dependent variable is the episode duration (how long the pole was balanced). The environment and Q-learning algorithm are the control variables, as they remain fixed throughout this process. Additionally, Q-learning might be substituted with a more thorough comparison to other methods. The second phase applies the best-performing algorithm from the first phase to the provided dataset (see Section 3.2), focusing on qualitative and quantitative results. The significant difference between experiments will be the number of codevectors, the map, and the constraints. The resulting trajectories will be visually evaluated, and their relative performance will be compared across different constraints.

The main hypotheses of this project are:

1. RL will provide a **direct link from the agent and network observations to motion planning**, surpassing an NDT;
2. encoding QoS communication constraints of interest into an appropriate cost function will help **prioritize network performance**; and
3. Utilizing a hybrid of Q-learning and adaptive state aggregation will yield a **resource-efficient model**.

3.3.1 Test environments

Both of the phases mentioned above utilize OpenAI's Gym library. The first phase uses the default CartPole-v1 environment, with each episode lasting 500 steps. The same seed was set for the environment and all supporting libraries (such as NumPy) at the start for initialization and randomization of actions,

to ensure replicability. The second phase involves a custom gymnasium environment, utilizing a 2D map that imports the pre-processed, pickled data from Section 3.2. The training is run for 1000 steps. The starting position is randomized at the start of each episode, but the goal is set in place at coordinates $(0.6, 0.8)$.

3.3.2 Hardware/Software

The methods used do not require a GPU to run, as they are incredibly lightweight. The development was done and tested only on Windows 11 machines. Regarding software requirements, the significant things required are Git, Python, and Python's virtual environment (as all of the libraries are saved within the *requirements.txt* file). If the robots were to be deployed, then ROS would have been crucial, and a robot (such as a Husky) would have been available for use.

3.4 Assessing Reliability and Validity

Given that the data was provided by the hosting company, Ericsson, rather than collected directly, the assessment of the reliability and validity of the data focuses on evaluating its quality and usefulness with respect to CAMP research. The following two sections establish the *validity* - how the dataset and method capture the intended outcome - and *reliability* - the consistency of their application and result.

3.4.1 Validity

The validity of the method was assessed to ensure that the approaches applied accurately measure the intention of researching CAMP.

To ensure construct validity, the first phase is done to test the effectiveness and efficiency of the method before applying it to the overarching problem. Effectiveness is operationalized through finding optimal actions via a cumulative reward. Since CartPole is a widely accepted RL testbed/benchmark, using it to evaluate the performance of the algorithm is valid, and it opens the possibility to compare with other methods. Efficiency is ensured by comparing the performance with the number of codevectors. Measuring CAMP is a more complex problem, and relying solely on an objective function that requires manual tuning (at the designer's discretion) is not the most appropriate metric. Therefore, qualitative measures, including

visualizations, are employed to ensure better construct validity by comparing paths with and without communication constraints. The given dataset, which includes RSRP for measuring the reference signal from a cell tower to the robot and handover regions for measuring the probability of switching to another tower, represents QoS metrics for 5G wireless networks, which were of interest to Ericsson. Hence, the generalization may be limited to its operational environments.

Ensuring high external validity with just the first phase alone is impossible. Results from CartPole can provide insights into how well vector quantization methods support RL convergence; however, they have a very weak generalization to complex robotic planning. Nonetheless, the abstraction is essential to discern if and how it works. In theory, the reward function between the simulation and the real world should remain unchanged, so once an appropriate one is constructed, it should exhibit high external validity. However, two things could lead to lower external validity: (1) static data and (2) a simple environment. Due to (1), it is hard to verify what would happen with temporal variations. Using a method like Q-learning, which records Q-tables for all states, could be helpful, as they are easily updated. Whereas RL methods using black-box NNs do not pair states with actions, so they might not adapt as well. (2) was required to collect high-quality data for training and testing, leading to a perfect environment that might not represent the real world very well. However, that should not matter much to RL algorithms, for which the most crucial part is the reward function. A final problem, which is a combination of (1) and (2), is that there are no dynamic interferences, such as other robots or humans. However, in these cases, the path would still be the same, and the low-level motion controller would ensure safe operation.

Internal validity was strengthened by using seeds to eliminate randomness and ensure standardization across runs. Similarly, hyperparameters will be held constant throughout runs and only varied systematically one at a time in a factorial design to minimize interactions from unaccounted parameters. Splitting the research into two phases and using the combination of qualitative and quantitative measures in the latter ensures triangulation of the method. Furthermore, the resulting paths will be reviewed by both the author and the supervisor for verification.

3.4.2 Reliability

The method can be considered reliable if, after running the experiment multiple times with the same hyperparameters and random seeds, it produces

consistent performance and convergence trends. The sources of randomness in both the method (such as taking random actions) and in the environment (such as initial states) must be controlled, and the variance across runs should be reported. Overall, the CartPole experiments are highly reliable, as they can be easily replicated under the same conditions.

Likewise, in the CAMP environment, the observed outcomes are reproducible under the same seed. Moreover, the objective function should lead to reliable behaviors that balance feasible motion and communication quality across multiple trials. Since the spatial data is static, repeated experiments will produce consistent results. On the other hand, because the dataset lacks temporal variability, it is reliable in a more limited set of static scenarios.

Overall, RL algorithms are inherently stochastic; thus, without sufficient runs and statistical analysis, it may be challenging to determine if the method produces reliable patterns or random fluctuations. However, Q-learning and VQ methods should lead to a convergent solution, and they are highly explainable, so it should be easier to find a reliable solution. Furthermore, analyzing the technique in the CartPole environment first can help identify dependable patterns in a benchmark environment. The CAMP dataset may have measurement biases, which could impact the reliability of any conclusions regarding spatial signal strength. That is something that cannot be controlled, and the only way to mitigate it would be to redo the data collection. Therefore, it is essential to keep that in mind when deploying to a real-world scenario.

3.5 Planned Evaluation Framework

To address the research objectives of CAMP, a data analysis plan is employed to interpret the existing dataset systematically. As previously stated, the analysis is divided into two phases: a quantitative experiment in a CartPole simulated environment to evaluate efficiency and effectiveness, and a mixed-methods approach for CAMP that assesses both qualitative and quantitative metrics. Robust, reproducible, and accurate results are needed to optimize motion planning while adhering to communication constraints. This section outlines the techniques and tools to ensure that the goal is met.

3.5.1 Evaluation framework

In the first stage, a RL algorithm (Q-learning) will be integrated with state aggregation algorithms (SVQ, SOM, ODA, None). The analysis in this phase will focus on comparing and analyzing various metrics:

1. **Convergence (Time to Stabilize)**: the number of training episodes required for the agent's performance to asymptote.

2. **Learning stability (Curve Smoothness)**: the absence of large and erratic fluctuations, or sharp drops in performance, indicating a smooth and reliable learning process.

3. **Performance (Average Cumulative Reward)**: expected total reward collected by the agent by executing actions according to the optimal policy.

4. **Model efficiency (Performance vs. Complexity)**: the cumulative reward relative to the number of codevectors used, quantifying the trade-off between model complexity and performance.

The results will be statistically compared across multiple independent runs with different seeds to identify the most effective quantization approach. Furthermore, the results from CartPole can be compared to other RL algorithms from the literature.

In the second stage, the best-performing state aggregation method will be applied to design a communication-aware objective function that incorporates the required QoS metrics. The analysis will focus on motion planning and communication quality trade-offs, using metrics such as:

1. **Path Efficiency**: measures the physical optimality of a collision-free path, calculated by the time taken and distance traveled to reach the goal.

2. **Communication Quality**: quantified by the cumulative reward integrating key QoS metrics: minimum delay, maximum signal strength (RSRP), and minimum handover switching.

These two steps will provide a controlled evaluation of both the method and the domain-specific assessment of their use in CAMP.

3.5.2 Software Tools

The entire analysis will be carried out using open-source simulation environments. All of the experiments leverage Python 3 because it provides a set of robust, openly available libraries crucial to working on the provided dataset and RL algorithms. The OpenAI Gymnasium (v1.1) will be utilized for the creation of the environments, and their visualizations - with the assistance of Matplotlib (v3.9), Seaborn (v0.13), and OpenCV-Python (v4.11). For statistical analysis, NumPy (v2.0) and pandas (v2.2.3) will handle data processing and numerical computations. The former will also be heavily utilized for creating all of the methods from scratch. GitLab will be used for version control, and Python's virtual environment will be utilized to package all the necessary libraries. These tools were selected for their reliability, availability, and alignment with the RL goals of this project.

3.6 System documentation

Due to the proprietary nature of the research conducted in collaboration with Ericsson Research, the code repository is private and accessible only to authorized personnel under the NDA. This section will provide a record of the components and protocols for this research to ensure reproducibility within the constraints of the industrial partnership.

All of the requirements are located in the $requirements.txt$, and the most important ones were summarized in Section 3.5.2. To set up the virtual environment, in the base directory, the following commands need to be executed:

Listing 3.1: Setup

```
# if it is the first time
(python3 –m venv .venv)

source .venv/Scripts/activate
pip install –r requirements.txt
(pip install –r requirements–dev.txt) # if developer

python –m ipykernel install ––user ––name=.venv
––display–name "venv" # (requires ipykernel)
```

The resulting codebase is located in */hybrid-rl*, and the directory structure is summarized in Figure 3.1. Most importantly, the core file with all of the

training is the **train.py**. The first thing it does is use an argument parser to load all of the passed-on variables, and sets most of the default ones. Other than some variables for saving, testing, and logging, here is a comprehensive list of the most important (excluding extensions and the comparison methods) variables/hyperparameters from the experiments (default values recorded in parentheses):

1. env: CartPole-v1 or custom CAMP environment
2. $seed$: Randomness seed for all the libraries (42)
3. $n\_episodes$: Number of training episodes (2000)
4. max_iter: Number of steps per episode (1000)
5. $updates$: Vector Quantization method {ODA, None} *Note:* This included {SOM, SVQ} in previous versions, and they are located in the $legacy$ directory.
6. $alpha$: Initial learning rate for Q-learning (0.5)
7. $alpha\_rate$: Decay rate for $alpha$ (0.999)
8. $epsilon$: Initial exploration probability (1.0)
9. $epsilon_rate$: Decay rate for $epsilon$ (0.995)
10. $epsilon_min$: Minimum value of $epsilon$ (0.01)
11. $gamma$: Discount factor for Q-learning (0.99)
12. $reward\_params$: List of the weights for the CAMP objective function $\{\alpha$: motion planning, β : signal strength, γ : handover$\}$
13. T_min: Minimum temperature for ODA (0.0001)
14. T_max: Maximum temperature for ODA (0.9)
15. $beta$: Initial learning rate for ODA (0.9)
16. $beta\_step$: Decay rate for $beta$ (0.9)
17. K_max: Maximum number of codevectors generated
18. $gamma\_steps\_1$: How quickly to update the temperature $([0.1, 0.5, 0.8])$

19. $normalize$: Normalization of the state space ($True$)
20. $replay\_batch$: Batch size of the replay buffer (0)

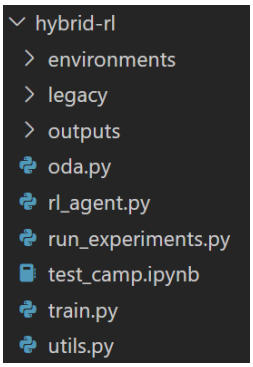

Figure 3.1: System Structure

Subsequently, the training script loads in relevant files to create the desired methods. For Q-learning, the *Agent* class is located in $rl\_agent.py$, and it includes all of the functions required to train and control the agent: $init$, $update$, eps_greedy_action, $follow\_policy$, and $decay\_params$. Furthermore, it is related to the vector quantization method. The most important functions that a VQ class has are: $init$, $update$, $winner$, and get_params. After creating the hybrid agent, the environment is created. The custom environment is loaded from the $environments$ directory, which contains the Gymnasium implementation and the pickle datasets. Finally, the file runs a standard RL training loop and logs all important parts into the $outputs$ folder. To run the training script, the following is an example, running with most default values:

Listing 3.2: Script Run Command

```
python hybrid-rl/train.py --env "camp" --K_max 50
--updates "oda" --seed 42

# for a list of available params
(python hybrid-rl/train.py --help)
```

Chapter 4

Methodology and Implementation

This chapter presents the methodology and implementation of the proposed hybrid Q-learning algorithm with vector quantization. The process was divided into two main phases; Section 4.1 describes the initial evaluation and experiments on the CartPole environment, and Section 4.2 outlines the application of the method to normal (4.2.1), and communication-aware (4.2.2), motion planning. This dual split was done to build an effective algorithm from the bottom up, and to conduct more in-depth research on the hybrid algorithms (for more information, see Chapter 3). The results of the experiments mentioned in this chapter will be presented in Chapter 5.

4.1 CartPole Simulation

This stage encompasses an expansive study into RL with VQ methods for adaptive state aggregation.

4.1.1 CartPole Environment

The efficacy of the proposed methods was evaluated using the classic *CartPole* problem (see Figure 4.1), which is a benchmark environment provided by the Gymnasium framework. The system models a cart that moves along a horizontal track with a pole at its center. The environment state is defined by four variables, represented by the vector $(x, \dot{x}, \theta, \dot{\theta})$, corresponding to the cart position, cart velocity, pole angle, and pole angular velocity, respectively. At each time step, the agent selects an action consisting of applying a discrete

force to the cart, either in the "right" or "left" direction. The objective is to learn a policy that balances the pole for as long as possible, and thus, the reward is measured as the number of timesteps before the pole falls. To be exact, either $|\theta| > 12°$ or $|x| > 2.4m$ result in failure.

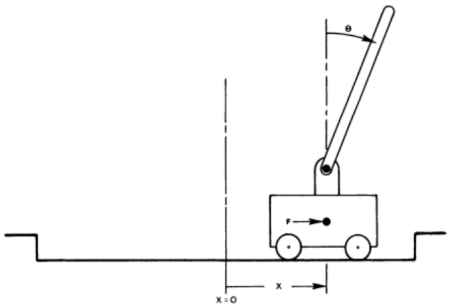

Figure 4.1: CartPole simulation environment [21]*

4.1.2 Hybrid Algorithm

The core component of learning the optimal policy is Q-learning, which enables the system to iteratively refine its policy by estimating the expected future rewards for state-action pairs (see Section 2.2 for more details). Fundamentally, the aim is to approximate the Q function via a parametric model that is piece-wise constant. The model's parameters, which specify the partition structure in the data space, are optimized through vector quantization algorithms. Nevertheless, the system dynamically processes its states and actions in real-time, employing TD RL approach to refine its policy. Consequently, there is a need for a two-timescale SA algorithm with a fast component that optimizes *control actions* in $Q := \{Q(k,u)\}_{k=1}^{K}$ via TD-learning, and a slow component which updates the *system modes* $\mu := \{\mu_k\}_{k=1}^{K}$.

The framework above is summarized in Figure 4.2, where the general algorithm operates similarly regardless of the VQ algorithm used. This means that when the code vectors are being updated, the update paradigms from the respective method are used. The methods are summarized below. To simplify visualizations and explanations, the state vector was simplified to $(\dot{x}, \theta, \dot{\theta})$.

Implementing the algorithms from scratch paved the way for experimenting with the methods and comparing them to existing literature. Likewise,

* used with permission from the author
* used with permission from the author

Algorithm 1 Hybrid Reinforcement Learning with Vector Quantization

Require: $\{\mu_k\}_{k=1}^{K}$, learning rate α, Bregman divergence d_ϕ, cost function C

1: **while** Convergence not reached **do**

2: Observe sample x_t and find cluster

3:
$$k_t = \arg \min_{\tau=1,..,K} d_\phi(x_t, \mu_\tau)$$

4: Choose optimal action
$$u_{t+1} = \pi(k_t)$$

5: Observe $x_{t+1} = f(x_t, \mu_{t+1})$, compute k_{t+1}, and update
$$Q(k_t, u_{t+1}) \leftarrow (1-\alpha)Q(k_t, u_{t+1}) + \alpha\big[C(x_t, u_{t+1}) + \gamma \min_{u'} Q(k_{t+1}, u')\big]$$

6: **if** Updating codevectors **then**

7: Update μ using Vector Quantization

8: **end if**

9: **end while**

Figure 4.2: Adapted from [20]*

understanding the processes and identifying the best ones will help determine how to correctly tackle the problem of CAMP in Section 4.2. Ultimately, it assists with finding an answer to the first part of the research question; i.e., understanding how a robotic agent can learn a control policy that simultaneously achieves the task objective.

Ad Hoc Discretization

Ad hoc discretization is a common approach used in conjunction with Q-learning when the environment is continuous. This approach divides the continuous state space into a set of equally distributed bins. The bounds and the spacings between the bins are hyperparameters, and thus, the localization and resolution of the codevectors are user-defined. An example from CartPole is shown in Figure 4.4. In terms of Algorithm 1, lines 6-8 are skipped completely as there are no codevector updates.

Stochastic Vector Quantization

SVQ begins with a simple discretization, similar to ad hoc discretization, and throughout training, the winning codevectors are updated. As shown in Algorithm 1, the closest codevector is found, and that codevector is updated using Equation 2.11, and the resulting algorithm is summarized in Figure 4.3.

Algorithm 2 Stochastic Vector Quantization update

Require: Winning cluster k, Sample x, Codevectors μ, learning rate β

1: Stochastic update: $\mu_k \leftarrow \mu_k + \beta(x - \mu_k)$

2: **Return** updated μ_k

Figure 4.3: Stochastic Vector Quantization update algorithm

This method provides a data-driven approach for updating the codevectors one by one. However, this brings an extra layer of complexity, as we no longer need to train a Q-learning algorithm; concurrently, we need to update the states in such a way that Q-learning can still learn optimal actions. Therefore, there are several directions for the experiments, and the major ones include looking into: (1) the hyperparameters, and (2) the effects of initialization. For (1), it is crucial to do an extensive study into hyperparameters that balance state exploration and discovering optimal actions. The experiments will test the following:

(a) Learning rates α (Q-learning) and β (VQ),

(b) Exploration rate ϵ,

(c) Codevector update rate.

(a) and (b) are tested by comparing the performance graphs over time. (c) is done to strike a good balance between the fast and slow components of the two-timescale SA framework, which is an essential factor as pointed out in [20, 21]. (2) is also very important as good initialization is crucial for the performance of this algorithm, as otherwise one can end up with a lot of "dead" neurons or the training can take extremely long before convergence is reached. An example of different starting initialization configurations is shown in Figure 4.4. This can help us understand the underlying nature of the data, and possibly find some hidden structures. These experiments also demonstrate the difficulty of good initialization, and thus lead to experiments with more adaptive algorithms.

Self-Organizing Maps

SOMs works on a similar principle to SVQ, but are slightly more involved. Once again, the algorithm finds the closest codevector, but instead of moving only the "best matching unit", the whole neighborhood is moved via the Gaussian falloff function. The algorithm implementation is summarized in 4.5.

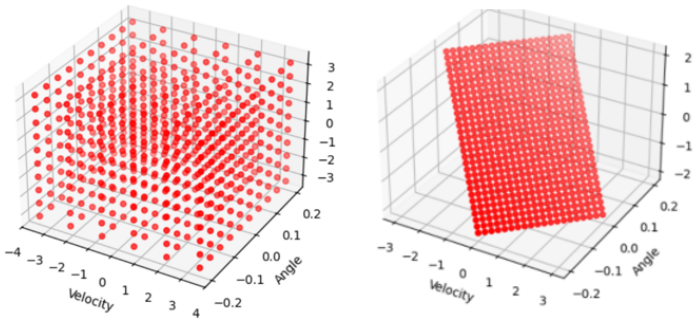

Figure 4.4: Diverse starting codevector configurations

Algorithm 3 Self-Organizing Maps Update

Require: Winning cluster k, Sample x, Codevectors μ, Learning rate β, σ, grid
Ensure: $\sigma > 0$; otherwise use Algorithm 2

1: Compute c_k by unraveling k in the grid
2: Generate all grid coordinates $\{g_i\}$ for grid dimensions $[n_1, n_2, \ldots, n_d]$
3: Compute distances: $d_i \leftarrow \|g_i - c_k\|_2$ for all grid coordinates g_i
4: Compute weights: $w_i \leftarrow \exp(-d_i^2/(2\sigma^2))$ ▷ Gaussian neighborhood
5: Stochastic update: $\mu \leftarrow \mu + \beta(x - \mu) \cdot w$
6: **return** updated μ

Figure 4.5: Self-Organizing Maps update algorithm

The experiments for SOM are similar to those conducted for SVQ, with both leveraging the two-timescale SA framework to update codevectors for Q-function approximation, as detailed in [20]. However, in this case, a greater emphasis was placed on extending and improving the method presented in [21] through two key enhancements: (1) the integration of a replay when doing codevector updates, and (2) the application of online normalization for the states. (1) is used to store a history of states, enabling the algorithm to sample past experiences during the updates, which aims to stabilize learning by reducing the impact of noisy or sparse data. Furthermore, it helps stabilize the two-timescale SA training, as it slows down the component responsible for the system's modes. Additionally, for (2), Welford's algorithm [22] was used to dynamically scale the states using running estimates of mean and variance. The major problem with Euclidean distance is that it assumes all dimensions are independent and equally scaled. Normalization ensures equal contributions from heterogeneous variables when performing distance

calculations for selecting the winning codevector and for neighborhood weighting. Importantly, this is also a SA algorithm, which means that over time, we will be moving towards the true expected mean of the dataset. Figure 4.6 summarizes the algorithm for updating the running mean and variance, whereas Figure 4.7 contains the normalization steps.

Algorithm 4 Welford's Online Update

Require: New state sample x, current count n, mean $\bar{x}$, variance σ^2, and aggregate squared distance from the mean M_2

if $n = 0$ **then**
    Initialize: $n \leftarrow 0$, $\bar{x} \leftarrow 0$ $\sigma^2 \leftarrow 0$, $M_2 \leftarrow 0$
end if
$n \leftarrow n + 1$
$\delta \leftarrow x - \bar{x}$
$\bar{x} \leftarrow \bar{x} + \frac{\delta}{n}$
$\delta_2 \leftarrow x - \bar{x}$
$M_2 \leftarrow M_2 + \delta \cdot \delta_2$
if $n > 1$ **then**
    $\sigma^2 \leftarrow M_2/(n-1)$
end if

Figure 4.6: Self-Organizing Maps update algorithm

Algorithm 5 State Normalization

Require: State x, running mean $\bar{x}$, variance σ^2, small ϵ

for each feature $f = 1$ to d **do**
    $\hat{x}_f \leftarrow (x_f - \bar{x}_f)/\sqrt{\sigma_f^2 + \epsilon}$
end for
return $\hat{x}$

Figure 4.7: Self-Organizing Maps update algorithm

Online Deterministic Annealing

The SOM experiments follow the pattern of previous experiments using the algorithm implementation in Figure 4.8, and aiming to reproduce the results detailed in [20]. The major extensions applied to the original method were: (1) normalization, (2) replay buffer, and (3) batch updates.

* used with permission from the author

Algorithm 6 Online Deterministic Annealing Updates

Require: Sample x, Codevectors $\{\mu_k\}_{k=1}^{K}$, Learning rate β, Perturbation parameter δ, Temperature T, Number of times for convergence n_c, Convergence threshold ϵ_c

while $K < K_{max}$ or $T > T_{min}$ **do**
  Perturb $\mu_k \leftarrow \{\mu_k + \delta, \mu_k - \delta\}, \forall k$
  Set converged $\leftarrow False$, $n \leftarrow 0$
  while not converged **do**
    Duplicate codevectors $\mu' \leftarrow \mu$
    for $k = 1$ to K **do**
      Update:

$$p(\mu_k|x) \leftarrow \frac{p(\mu_k)\exp(-\frac{d_\phi(x,\mu_x)}{T})}{\sum_{k'} p(\mu_{k'})\exp(-\frac{d_\phi(x,\mu_{k'})}{T})}$$

$$p(\mu_k) \leftarrow p(\mu_k) + \beta_n\big[p(\mu_k|x) - p(\mu_k)\big]$$

$$\sigma(\mu_k) \leftarrow \sigma(\mu_k) + \beta_n\big[xp(\mu_k|x) - \sigma(\mu_k)\big]$$

$$\mu_k \leftarrow \frac{\sigma(\mu_k)}{p(\mu_k)}$$

    end for
    Set $n \leftarrow n + 1$
    if $d_\phi(\mu, \mu') < \frac{\epsilon_c}{T}\frac{(1+\beta_0)}{1+\beta_n}$ satisfied n_c times **then**
      Set converged $\leftarrow True$
      Clean overlapping codevectors
      Update Temperature T
    end if
  end while
end while
return updated $\{\mu_k\}_{k=1}^{K}$

Figure 4.8: Online Deterministic Annealing algorithm adapted from [20]*

(1) and (2) follow the same logic as described in the previous section. The only difference is that instead of using Welford's algorithm for normalization, *Mahalanobis distance* was used. Mahalanobis distance is a multivariate measure that incorporates the correlation and scale between the variables. For a vector x relative to a mean vector μ, with covariance matrix S, the distance metric is defined as:

$$d_m(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}.$$

It is a Bregmann divergence, which means that it can be used with ODA with theoretical convergence guarantees. The reason for the change in the distance metric is to encode the normalization explicitly into the distance metric, specifically using S^{-1}; where large-variance directions will be down-weighted, and correlated features will be decorrelated. This approach avoids constant transformations of the states between the original and normalized frames, but simplifies the implementation and builds on the theoretical foundations of the VQ methods. Over time, as more samples are observed and the S matrix is built up, the algorithm will move from using basic Euclidean distance at the start to normalizing the state space. For (3), the update steps themselves were altered from using a for-loop (lines 6-8 of Figure 4.8) to using NumPy vector computations. The reasons for changing the logic are two-fold. Firstly, omitting a for-loop results in a considerable speed-up of the algorithm, as this is the biggest overhead. However, updating all of the codevectors in parallel changes the logic of the original implementation, as the for-loop updates each codevector separately, changes the probabilities, and then updates the next ones iteratively. Whereas, the "vector" implementation computes the probabilities a priori and uses them to update all the codevectors. Therefore, it is interesting to explore how the original theory can be implemented in different ways and examine the effects on the results.

4.2 Communication-Aware Robotics Application

The second stage of the research process moves towards the desired application goal of CAMP by first applying the algorithm to a standard *motion planning* problem and then adding the desired *communication constraints*.

4.2.1 Motion Planning

Moving from CartPole to motion planning changes the state/action space, but the algorithms themselves stay the same. The testing environment, and the state space) is composed of (x, y) locations on a 2D map. The robot can take an action of moving in those two directions. Therefore, there are two continuous states and two continuous actions. The objective is to reach a pre-determined goal on the map, regardless of the starting position. The simple environment is displayed in Figure 4.9. To solve this problem, a hybrid of Q-learning and ODA was used. Overall, the reward function guides the agent, which in turn

also affects the state aggregation algorithm that discovers the optimal modes of the system. For this problem, the optimal states will be located in the most important areas of the map. In contrast, in sparsely visited areas, there will be limited representation, resulting in localized resolution. Moreover, the algorithm's results can be visualized as points on the map, each with an optimal action assigned to it via the Q-table, which aids in explainability and debugging.

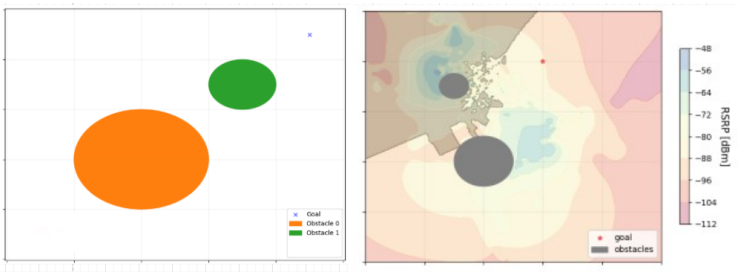

Figure 4.9: Motion Planning environment [left]; CAMP environment [right]

Before diving into the experiments, it is essential to note that the continuous action space was discretized into eight directions around the agent. The reasons being that the discretization works well enough, unlike the much more complex state space, and training state and action space discretization concurrently in ODA is still an open problem*.

Firstly, the reward function that was ultimately used to motivate minimum time delay was:

$$R_{delay} = \begin{cases} +100 & \text{if goal reached,} \\ -100 & \text{if collided,} \\ -r_{dist} - \epsilon & \text{otherwise,} \end{cases} \tag{4.1}$$

where r_{dist} is the progress to the goal, and ϵ is a small constant penalty. This reward function was designed to motivate the agent to reach the goal (due to a high reward), avoid collisions (as it results in a significant penalty), and move towards the goal efficiently (as penalties will accumulate over time).

For the experiments, a mix of qualitative and quantitative analysis was conducted on the results of using Q-learning with either *ad hoc discretization*

*An attempt was made to aggregate the states and actions, but that would result in the Q-learning framework breaking, so a different approach would be needed, which was outside of the scope of this project.

or *ODA*. The former is the traditional method of implementing Q-learning in a continuous state space, making it a valuable comparison point. The training lasted approximately 2,000 episodes, with each episode consisting of 1,000 steps, allowing the robot sufficient time to reach the goal. Furthermore, an Actor-Critic network was attempted, as it is the traditional approach to training a RL on a continuous state space.

4.2.2 Communication-Aware Motion Planning

The final stage of this research was extending the traditional motion planning to include QoS communication constraints. The map of interest is slightly different as it incorporates the signal strength and handover regions obtained through real-life data collection. There are various datasets, but the one in Figure 4.9 was used most extensively for testing and visualization.

As discussed previously, the reward function drives both the Q-learning agent and, in turn, the state aggregation. Therefore, more emphasis was placed on testing how different reward functions affect the algorithm and how they can be used to reach the desired results. The reward function of interest is:

$$R = \alpha R_{delay} + \beta R_{signal} + \gamma R_{handover}, \tag{4.2}$$

where $\{\alpha, \beta, \gamma\}$ are trade-off parameters (weights) of each component of interest. Those need to be set manually and are thus a designer's choice. R_{delay} is the time delay component (see Equation 4.1), R_{signal} is the normalized, squared, and scaled down (by 0.1), strength of network connection in decibels (RSRP), and finally, $R_{handover}$ returns a penalty (of 10) if a handover region is crossed. The results were compared by creating a small dataset of random starting locations, both randomized and specifically chosen to test the algorithm in specific scenarios. This is a comparison of a more qualitative nature. Still, a quantitative performance evaluation was also conducted to assess how well each algorithm performs in comparison to the desired application. That is, the algorithm can be trained with a different reward function, but then evaluated using the "full" reward function from Equation 4.2.

Chapter 5

Results and Analysis

This chapter focuses on the findings based on experiments outlined in Chapter 4. Afterward, an analysis is done to verify their reliability and validity.

5.1 Results

Section 5.1.1 showcases the most important findings from the CartPole experiments, whereas Section 5.1.2 summarizes the effects of the hybrid algorithm on traditional and communication-aware motion planning.

5.1.1 CartPole

As previously described, the state aggregation method within the hybrid algorithm varies throughout the experiments; therefore, the following sections will summarize the significant results from those experiments independently. Ultimately, all the algorithms will be compared and evaluated against each other before proceeding to motion planning. The following sections will present the most important findings for each method, and the final section will compare the best results across all methods.

SVQ

When experimenting with SVQ, two major things were taken into account: (1) hyperparameters, and (2) codevector initialization. Regarding (1), it was concluded that as long as the parameters gradually decrease over time, the performance won't change significantly, and the model will continue to learn. Therefore, for the experiments, once good parameters were discovered, an

exponential decay function was applied (see Figure 5.3). Following the two-timescale SA theory, the VQ component has a small learning rate to stabilize the training over time. Initially, the learning rate for the VQ was kept constant, and then halfway through, it was dropped to zero, for Q-learning to learn on converged codevectors. Still, it was discovered that slowly decreasing it throughout the *whole* training yielded slightly better results. Most likely, the reason it works better is that when the actions are mostly random, it is better to make larger changes to the codevectors. However, over time, as the agent aims to take more optimal actions, the codevectors should not change as much.

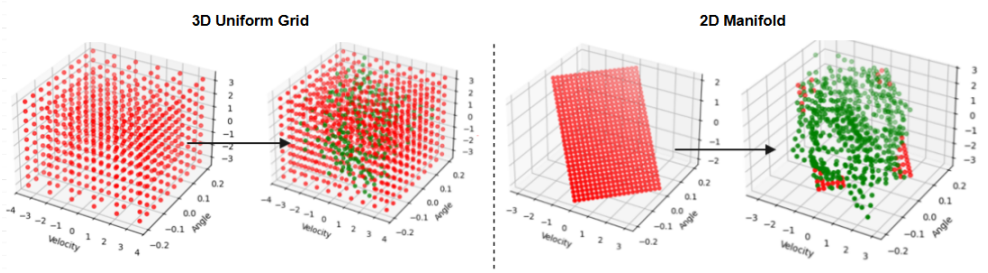

Figure 5.1: Different SVQ codevector initializations and their resulting locations after convergence (indicated by an arrow). Red indicates if they have not moved from their original position ("dead neurons"). Uniform 3D grid *[left]*; 2D dimensional manifold *[right]*

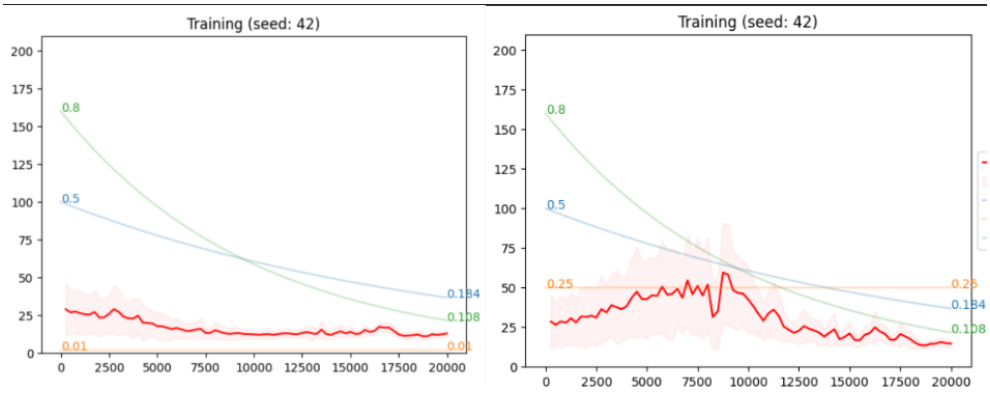

Figure 5.2: Resulting training curves (red) with different SVQ learning rates (orange), Q-learning rate (blue), SVQ learning rate (orange)

However, it was concluded through visualizations of the state space and training curves that (2) makes a drastic change to the results, and is much more

critical than simple hyperparameter tuning (see Figure 5.2 for comparison). Specifically, Figure 5.1 shows the difference between initializations, and Figure 5.3 displays the resulting performance. This entails that, to achieve decent results within the training time, it is crucial to find a suitable initialization. The reason it boosts performance is that there are significantly fewer "dead neurons", allowing for a higher resolution of codevectors within the area of interest, which in turn makes it more accurate. However, this process is not automated; instead, the initial results were analyzed and manually re-initialized in the area of interest. This makes the process tedious, inflexible for other environments, and not applicable to higher dimensions.

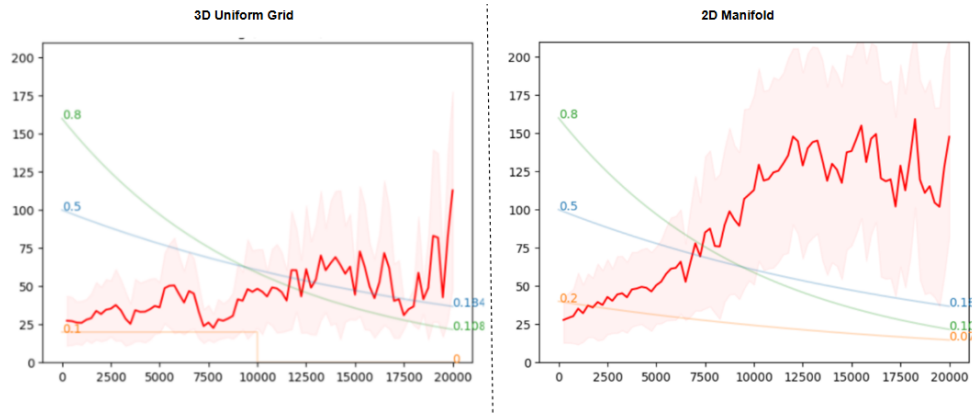

Figure 5.3: Resulting training curves (red) with initialization from Figure 5.1. Resulting training curves (red) with initialization from Figure 5.1, SVQ learning rates (orange), Q-learning rate (blue), and SVQ learning rate (orange)

Regardless, analyzing the results shows a possible lower-dimensional manifold in the state-space of CartPole (when excluding the position of the cart). Finding such a manifold is valuable because it reveals underlying constraints and correlations in the system's dynamics, possibly enabling efficient computation through dimensionality reduction. Regardless, it is essential to consider that manifolds can be non-linear and approximating them can lead to a loss of information. For this reason, vector quantization algorithms are potent, as they can learn a better approximation of the state space given a good starting position (the manifold) through interaction with the data. This topic of initialization will be revisited in the upcoming sections; however, over time, the problems above will be addressed through more refined methods.

SOM

Training the algorithm with SOM proved to be more challenging than initially expected and replicating exact results and behaviors from [20] failed, although, different findings were discovered.

A framework was designed to run the SOMs updates on a uniform grid of codevectors for a couple of episodes, while decreasing the neighborhood size until only the winning codevector is updated (i.e., mimicking SVQ updates). This resulted in a very similar training performance to the previous experiments, but it eliminates the required hand-tuned initialization, addressing the problem discussed with the method earlier. The framework and performance are summarized in Figure 5.4, and similar results were obtained across multiple seeds.

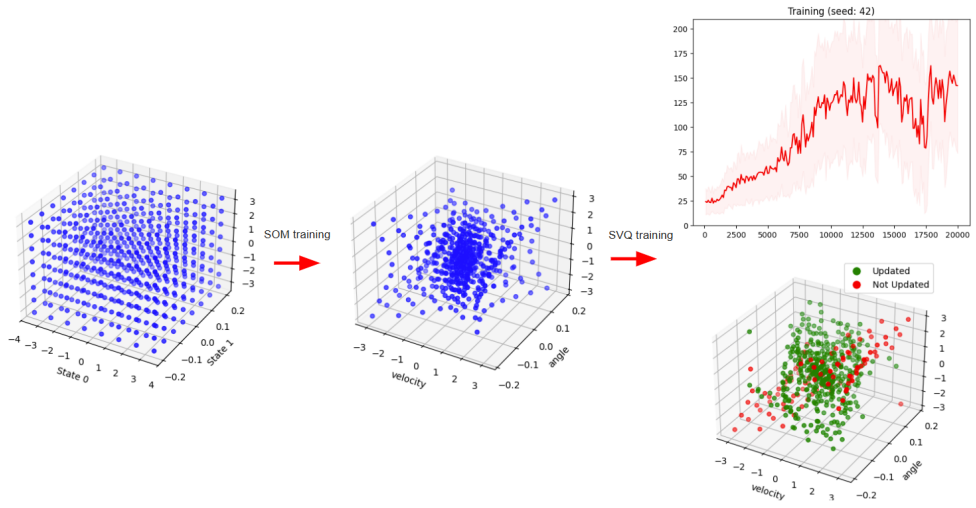

Figure 5.4: Short pre-training of an SOM, and then switching to SVQ updates

Later on, that framework was abandoned, and instead, specific extensions were implemented to improve the training. Firstly, a replay buffer was added with a size of 32 for both the Q-table and codevector updates (every 10 steps). Secondly, online normalization was added (explained in Section 4.1). Finally, the weight computations were altered to use the grid locations instead. As a result, performance improved significantly, allowing for the use of much smaller grids, such as 8x8 in this case, which resulted in 64 codevectors. The previous method required a couple of hundred codevectors, and prior work by [20] required 625 clusters to get good results. The parameters

and training are showcased in Figure 5.5, which suggests that normalization is a crucial component when dealing with variables of different scales. Furthermore, collecting data across the episodes and randomly drawing them breaks temporal correlations, stabilizes the training, and improves sample efficiency.

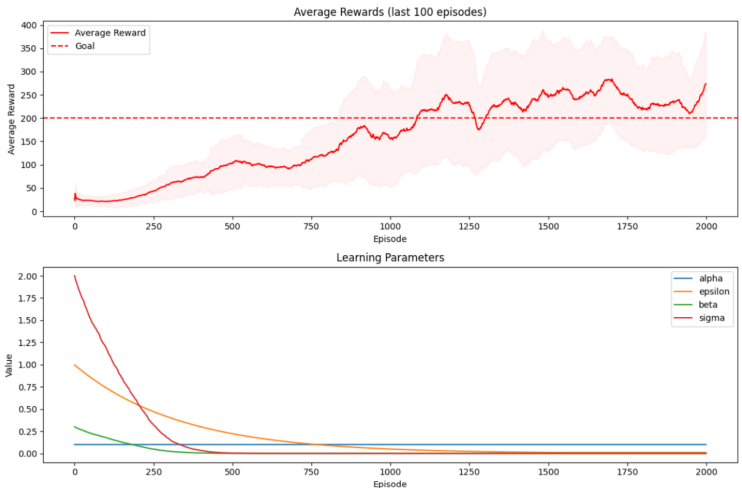

Figure 5.5: SOM performance and parameters using a replay buffer and online normalization

Despite getting decent performance, it still poses some challenges. For starters, determining the number of aggregate states is typically domain-specific, and selecting an appropriate number can be challenging. Moreover, deciding on initial positions can also be problematic, as seen before. Ultimately, there is no way to determine the optimal performance versus complexity trade-off. The following method aims to mitigate all of the problems, and the results indicate that that's indeed the case.

ODA

Following the failed attempt to directly replicate the SOM results, firstly, the current ODA implementation was compared to the original from [20] using a simple 2D toy dataset, which requires clustering 5 Gaussian islands.

Firstly, following the observations from previous experiments, normalization was added. The comparison between normalized and un-normalized training is visualized in Figure 5.6. All of the other experiments also incorporated normalization, as it boosted performance significantly and stabilized the training.

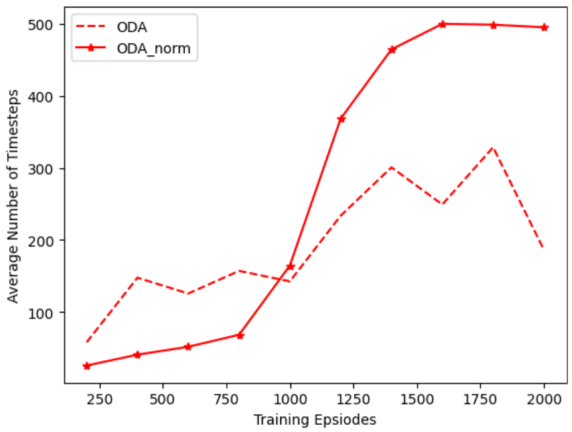

Figure 5.6: ODA algorithm trained on CartPole with and without normalization

Testing a wide range of hyperparameters yielded varied results, but overall, training a decent model that outperforms the other methods was not particularly challenging (as will become apparent in the next section). One example of varied results was obtained when training with a low temperature threshold ($T_{min} = 1e - 3$) and varying the frequency of ODA updates (Figure 5.7). Using a replay buffer did not yield as good results as it did for SOM, but updating at every step was also not ideal. It turned out that updating every other step worked out the best.

Finally, one exciting finding was the consideration of the batch update to the codevectors (see Chapter 4 for an explanation). When using that approach, the performance is very similar to that of the traditional update, with a slight caveat. As the training length increases and the temperature decreases, the expectation is to obtain more codevectors. However, due to the slight changes in the update, a point is eventually reached where the codevectors stop increasing and instead start to merge back together, significantly reducing the number of codevectors, as shown in Figure 5.8 (which occurs for all seeds). Usually, it would be expected that it would also drop performance, but that is not the case. In the CartPole training, the only drawback was that it resulted in somewhat cyclic learning, where the performance would oscillate; however, this is a reasonable consequence of having a tiny number of codevectors to represent the state space.

Additional experiments were conducted in the Pendulum environment to test this behavior. Unfortunately, in this environment, using this method was

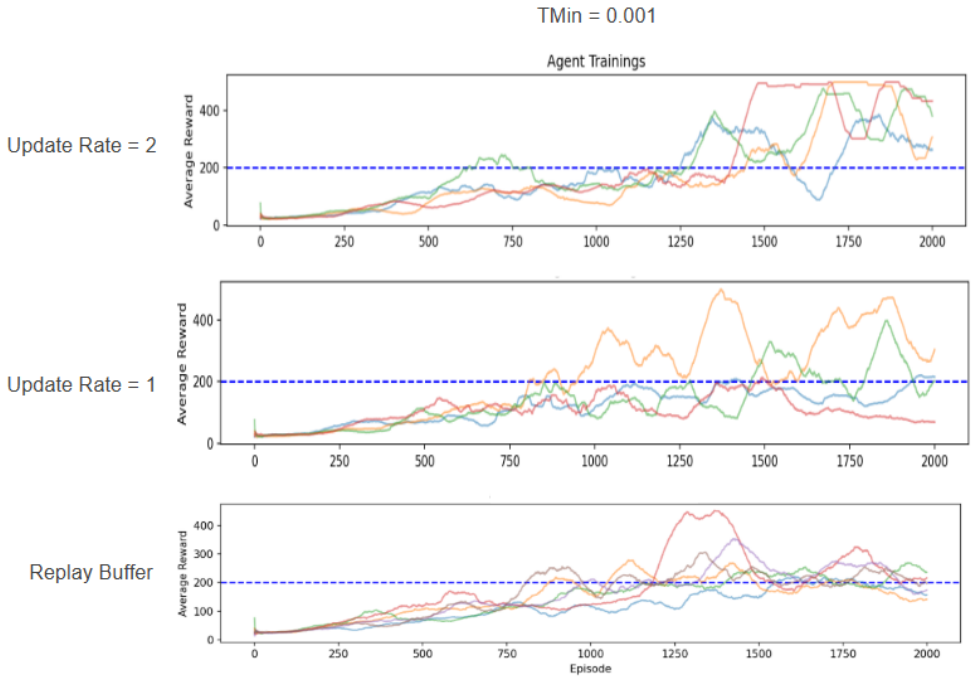

Figure 5.7: Different update frequencies for ODA CartPole training with different seeds

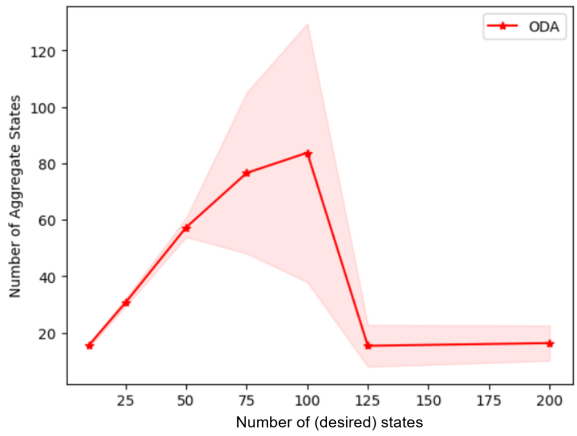

Figure 5.8: Number of converged codevectors as the training length increases (using batch updates)

not successful, as it resulted in dropping most of the code vectors down to a single one. It is unclear why that happened, but switching to the traditional

updates resolved the issue. As working on ODA was not the main topic of the research, and satisfying results were obtained, this research thrust was not pursued further. The most likely explanation for this merging behavior is that updating all of the codevectors concurrently without updating the new distances and probabilities can result in the codevectors overlapping more likely. However, further research would be needed to explain why there is no drop-off in performance.

Summary

To summarize the results from the CartPole experiments, Figure 5.9 compares the performances across the methods. Despite some differences in the implementations, the overall ranking is as expected, with ODA outperforming the other methods significantly; meanwhile, SOM performs well and better than ad-hoc discretization, which cannot compete with the other ones.

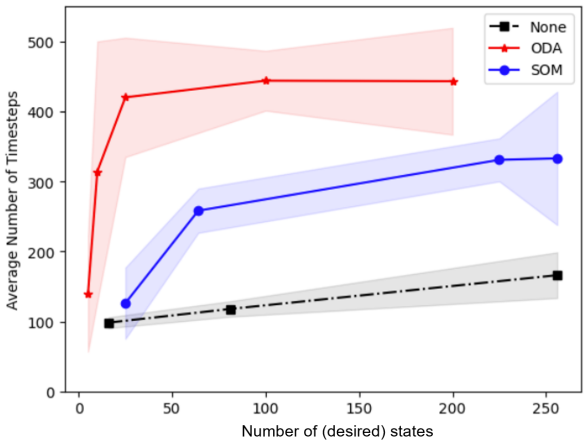

Figure 5.9: Average number of timesteps for different numbers of state codevectors

These results circle back to the motivation behind these methods. They provide an adaptive and local resolution, which, in theory, offers strong convergence guarantees. Additionally, ODA takes it a step further by incorporating hierarchical clustering, which does not require any initialization, and gives a performance-complexity trade-off via its hyperparameters. Finally, a very good approximation of the state space can be estimated with a minimal number of codevectors, making the model highly time- and memory-efficient (see Figure 5.9).

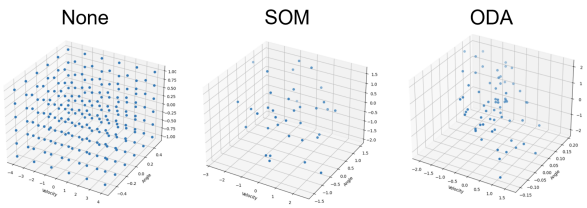

Figure 5.10: Converged codevectors on CartPole

Given the satisfactory results on the CartPole environment, the hybrid Q-learning algorithm with ODA was chosen for the Motion Planning problem

5.1.2 Communication-Aware Motion Planning

The following experiments primarily focused on applying ODA to motion planning, as described in Section 4.2. This ties directly to the motivation behind this research, namely, implementing an algorithm for a robot to simultaneously achieve the task objective and maintain reliable communication when access to the network model is unavailable. Having implemented and conducted thorough experiments on the CartPole environment, switching to the motion planning case primarily involves adjusting the reward function and performing some minor hyperparameter tuning, which demonstrates the method's adaptability.

Motion Planning

Firstly, starting with the *task objective* itself, the algorithm focuses on optimizing the reward function from Equation 4.1 (ignoring the communication constraints).

Results presented in Figure 5.11 showcase the difference between ODA and ad hoc discretization, both of which have around 50 codevectors. Both algorithms successfully discover valid and safe trajectories for the robot; let's analyze the results. Firstly, the difference between the codevectors representing the state space and the corresponding Voronoi partitions differs significantly. With the vector quantization method, it is clear that many more codevectors are present around the obstacles and the goal, where they matter the most. Whereas, there is much lower resolution in the corners of the map, where the agent doesn't visit very often. In comparison to the ad hoc discretization, the codevectors are utilized in a "smarter" way. This is further proven to be the case in Figure 5.12, where the ad hoc discretization with the

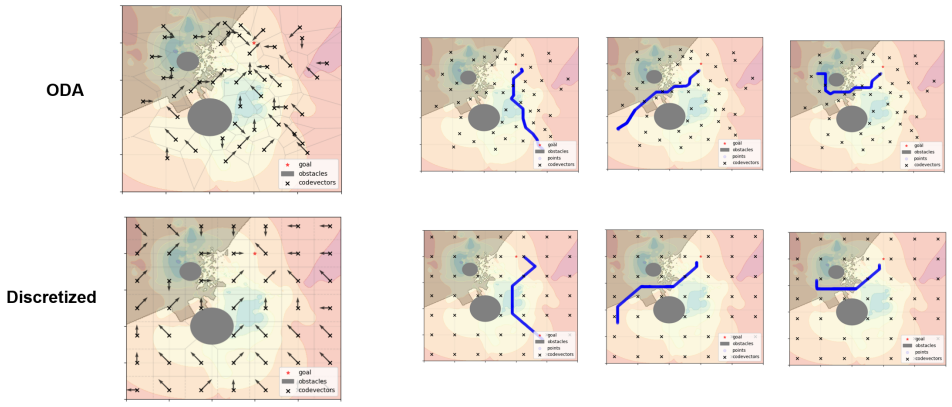

Figure 5.11: Comparison of resulting trajectories (blue) and optimal actions (arrows) at different codevectors, between ad hoc discretization and ODA on traditional Motion Planning

same amount of codevectors cannot reach the performance of ODA, despite it being a simple environment. The only way to match the performance is by increasing the resolution of the grid, but this results in slower computation and a significant amount of wasted code vectors. One important observation from the graph is that lower resolution do result in a bit of a periodic learning behavior, which is mostly the case that depending on the actions near the goal, there might just missing the goal, but that would be mitigated in real-world scenario by switching to a local controller around the goal (whereas this would serve as more of a global one). Furthermore, increasing the complexity and the size of the map would make the ad hoc case work even worse. Another necessary consequence of using discretization is that, in extreme cases or around obstacles, depending on the state's location and frequency of visitation, the final action may not be optimal, which can be hazardous for the robot. This problem is mitigated with vector quantization as all the codevectors are data-driven.

It is important to note that an Actor-Critic model was attempted on this task to make a comparison with a prominent Deep Learning RL method. However, despite training an architecture successful on the Pendulum environment, training it on this environment has failed. This highlights the major problem with black-box algorithms, namely, the limited interpretability of what the agent has learnt directly. This can be a major drawback when dealing with safety-critical decision-making. In contrast, the proposed methods allow explicit inspection of state-action values, which can aid in understanding

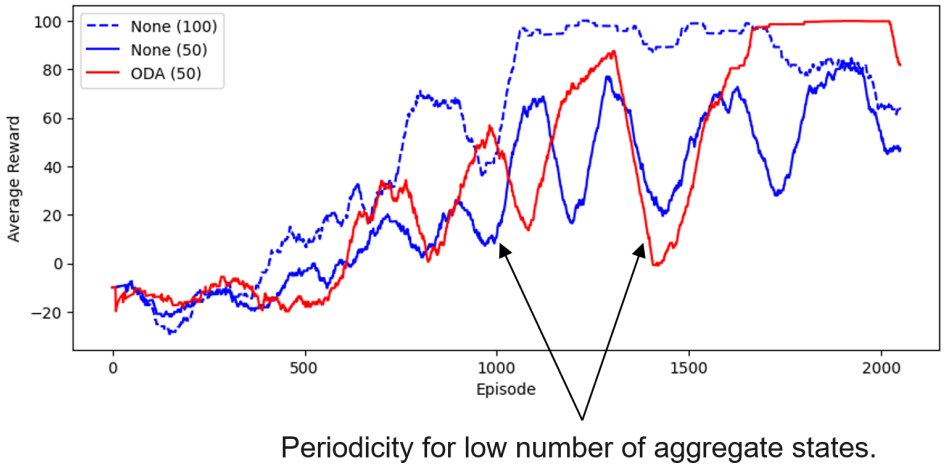

Figure 5.12: Training curves for ad hoc discretization (low and high resolution) and ODA on traditional Motion Planning

the learned behavior. While deep RL is well-suited for continuous or very high-dimensional state-action spaces, there is a lot of additional complexity involved in terms of model design, stable training, or hyperparameter selection. Furthermore, adapting a deep Actor–Critic network to even small changes in the environment could require substantial retraining, as the network only stores parameter weights. By comparison, the proposed method could, in principle, handle such changes by updating or refining the corresponding Q-table, rather than re-learning the entire policy from scratch (this aspect is further discussed in Chapter 6).

5.1.3 CAMP

Finally, the following results apply the hybrid ODA and Q-learning algorithm to all the networking data. The first set of experiments tested how the reward function changes the trajectories. Figure 5.13 shows how the resulting trajectories from the same starting point differ drastically.

The left column showcases ordinary motion planning (as seen in the previous section), and there is clearly no regard for either signal strength or avoiding handover regions. This is particularly clear when evaluating the resulting reward, as shown when the networking constraints are implemented (top of each image). Once the signal strength is included (in the middle column), the robot attempts to stay in regions with better signal strength. It prioritizes them over reaching the goal as quickly as possible. Finally, the

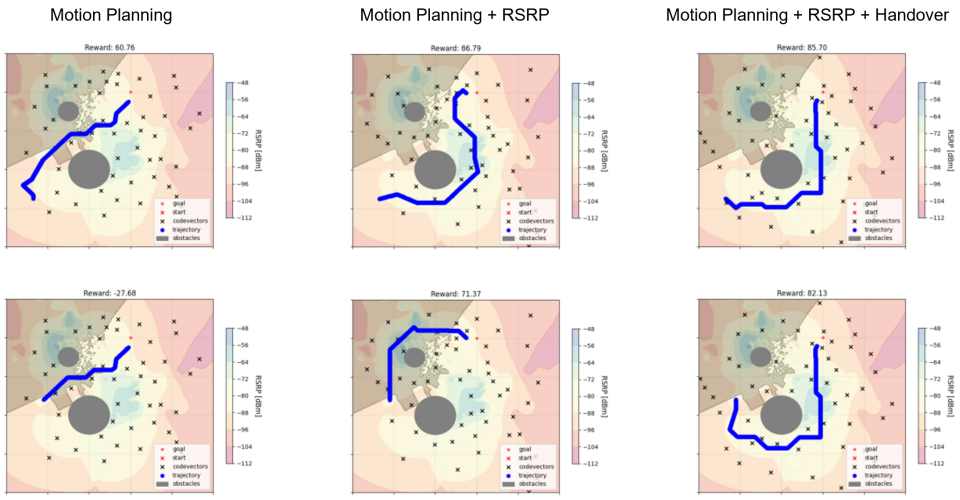

Figure 5.13: Comparison of the effect of reward functions with different weight parameters on the robot trajectories (evaluated using the method explained in Chapter 4)

last column shows the addition of handover regions. With this setup, there is a stark difference in how the algorithm behaves, as it avoids crossing into handover regions altogether, mainly because entering one means that it will have to leave it once again to reach the goal. Figure 5.14 illustrates the subtle differences that the algorithm learns, as mentioned above.

These results in particular show the power of using RL algorithms, as by simply changing the reward function, the entire behavior of the algorithm changes, and the reward function (as presented in Equation 4.2), only needs changes to the parameter weights to prioritize one objective over another. Moreover, the vector quantization method is also influenced by the reward. For instance, in the case of incorporating signal strength, more codevectors will be generated around areas with better signal strength, as that is where the robot will be heading more often during training. Whereas, when only the motion planning is being run, the codevectors are more equally spread out. Yet again, this demonstrates the adaptability of the ODA algorithm, a feature that would be very difficult to replicate using other vector quantization methods.

However, using this approach also has its consequences. Figure 5.15 illustrates the quantitative differences between the methods, assuming the objective is to train an agent on all communication constraints. This is valuable information because it helps analyze the algorithms as they should behave (given the ideal reward function). As expected, the motion planning algorithm

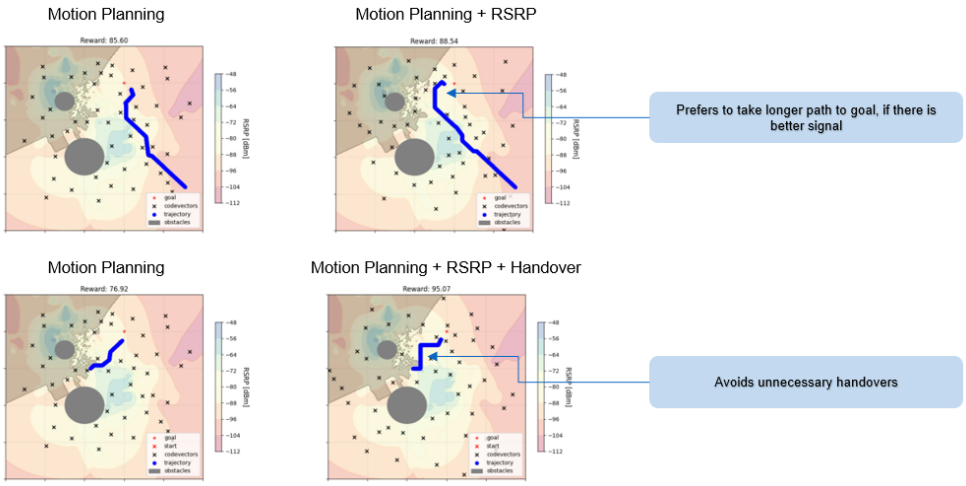

Figure 5.14: More in-depth comparison of behaviors between agents trained on different reward functions

Figure 5.15: Performance distribution for different reward functions (evaluated by the ideal reward function where each component is evaluated equally)

performs much worse overall because it lacks knowledge of the network. Overall, the algorithm incorporating all of the constraints works very well, but there are some outliers where it fails. Firstly, adding more objectives may lead to the other objectives being undervalued, and choosing the correct weights can be a challenging problem in itself. For instance, the handover penalties might undervalue the signal strength and time delay, resulting in much longer

paths or ones that may not reach it in the first place. This is a general problem in RL, where the agent may sometimes evaluate a penalty along the path as too great and thus prefer to either stay dormant or even hit an obstacle to end the episode early. The former is presented in Figure 5.15, which results from being placed in a challenging position that requires crossing the non-convex handover region. Thankfully, during the experiments, the latter was not observed, which means that the agent was well-trained to avoid obstacles. Therefore, more fine-tuning of the reward function should be done to prevent these issues, or perhaps a different approach specifically for the handovers, as sometimes they cannot be avoided altogether. Furthermore, when computing the rewards, the values are currently normalized, and thus, the changes in the entire path can be minuscule and often complex for the algorithm to choose between.

To conclude, incorporating communication constraints into the reward function shows promising results with the current hybrid algorithm, and, except for a few edge cases, it performs very well. Some improvements and extensions will be presented as future work in Chapter 5.

5.2 Analysis and Discussion

The following two sections discuss and analyze the reliability and validity of the experiments. Section 5.2.1 evaluates the consistency and stability of the method and data across the experiments, ensuring that results are reproducible. Section 5.2.2 assesses whether the technique and data accurately measure the intended construct in the context of the research objective.

5.2.1 Reliability Analysis

The data used for CAMP experiments was collected and provided by Ericsson, and pre-processed for the environment by this project's supervisor. The data was assumed to be reliable, as Ericsson's Research team had previously used it for their own experiments.

Throughout the experiments, random seeds were utilized to ensure that they are standardized, and thus, ensure repeatable outcomes. The goal is to ensure consistency of experimental procedures. Firstly, both environments (CartPole and CAMP) were formed as OpenAI Gym environments, which allow complete control over the randomness of the data; for instance, it can include consistent random starting positions. Secondly, the random processes governing the methods were controlled for, such as the codevector

initialization or random actions in the Q-learning algorithm. To ensure everything is controlled for, it was verified that re-running an experiment with the same seed results in the same training curve. Furthermore, the algorithms were tested with different seeds, and most of the graphs in Section 5.1 include either the variance across multiple runs (shaded area) or separate training curves for each seed. This demonstrates the algorithm's stability, as its performance may vary depending on randomization. However, the results show that the method works consistently, especially the ODA hybrid algorithm. To ensure the validity and reliability of the method, triangulation was employed, first using the CartPole simulation and then conducting quantitative and qualitative analyses on the CAMP simulation. To identify the exact differences in the algorithm's outcomes, the same starting positions were used, and the resulting trajectories were inspected. The starting positions for testing were a combination of both hand-picked and randomly selected locations on the map, to minimize experimenter bias.

Finally, to ensure that conclusions can be drawn reliably, the hyperparameters and methods were carefully changed, whilst keeping the majority of other parameters constant. Specifically, once suitable learning parameters for Q-learning and vector quantization were identified, they were largely left unchanged. This was especially important to check how the different parts affect the training, without any confounding effects from other parameters. For example, when testing for normalization, only the normalization would be enabled to compare before and after.

5.2.2 Validity Analysis

Construct validity was ensured by aligning the evaluation metrics, such as the episode duration in CartPole before the pole drops and path trajectory in CAMP, with the theoretical constructs of Hybrid RL and CAMP. While testing the algorithm in the CartPole environment provides insights into basic control performance, it lacks the complexity of the motion planning and communication constraints, making it insufficient for high construct validity. However, the CAMP simulation, incorporating agent dynamics and communication data from Ericsson's datasets, comprehensively captures the intended phenomena, thereby enhancing the validity of the results.

Internal validity was strengthened by controlling for confounding variables in both experimental phases and by using simulations with fixed parameters to isolate algorithm performance. This ensures that the observed outcomes accurately reflect the impact of the Hybrid RL algorithm, rather than external

factors. Using the CartPole environment limits the applicability to real-world scenarios. Therefore, to ensure high *external validity*, the CAMP simulation environment was designed to closely mirror the desired industrial settings, such as warehouse or city navigation with obstacles and communication constraints, ensuring that results are generalizable to similar operational contexts. On the other hand, the simulations are a simplified model, constrained by the use of a single robot and static obstacles, which limits their generalization to Ericsson's applications, where multiple robots, changing network conditions, or moving obstacles (such as humans) may be present. Although these simplifications help isolate the algorithm's performance under controlled communication conditions, they reduce its applicability to real-world settings that require multi-agent coordination and real-time adaptability. These issues will be further discussed and evaluated in Chapter 6, which will also outline future work.

Confirming the pre-existing dataset accurately reflects the phenomena of CAMP for a single robot in a static environment ensures good *data validity*. The dataset covers variables of interest to Ericsson, such as RSRP and handover regions. The original problem involves staying in strong signal ranges (measured by the former metric) and avoiding handover regions. Hence, the data should contain the most critical aspects to ensure high content validity.

Chapter 6

Conclusions and Future Work

This chapter concludes the thesis project by first summarizing the main results and discoveries in Section 6.1. Section 6.2 outlines the main drawbacks of the study in terms of the method and the dataset. Section 6.3 builds on these limitations to construct a plan for future work. At the end, Section 6.4 will reflect on the ethical, societal, and sustainability concerns that this project affects.

6.1 Conclusions

This thesis addresses the central research question of how a robotic agent can learn a control policy that simultaneously achieves task objectives and maintains reliable communication, without access to a network model or knowledge of uncertain network properties, as outlined in Chapter 1. By integrating Q-learning with VQ methods, the study achieved significant positive effects. Initial experiments integrated various methods, including SVQ, SOM, and ODA, and tested them on the CartPole example. Although the exact findings from prior work (see Chapter 2) were not directly replicated, the performance trends were consistent, with notable extensions enhancing the proposed methods. Specifically, the incorporation of normalization, batch updates, and a replay buffer significantly improved the stability and efficiency of the Q-learning policy. In particular, the ODA technique achieved robust and high-performing results on the CartPole task, as evidenced by stable convergence and effective state-space partitioning, as indicated by the minimal number of codevectors required. These experiments demonstrated how the reward function guides the RL algorithm to perform specific actions, which in turn influence how the state space is explored, ultimately affecting the VQ

method.

The experiments on the hybrid algorithm revealed several key benefits for autonomous systems, as outlined below:

1. Adaptability,
2. Localized resolution,
3. Explainability,
4. Convergence.

The algorithm demonstrated robust adaptability to diverse state spaces, requiring only a well-defined reward function to achieve effective performance, with minimal hyperparameter tuning compared to alternative methods. Specifically, ODA eliminated the need for predefining the number of codevectors or specifying initial conditions, as adjusting the temperature parameter enabled dynamic refinement of state-space partitions. This facilitated higher resolution in critical regions, such as near goals or obstacles, optimizing codevector placement for efficient policy learning. Consequently, the algorithm achieved stable and computationally efficient performance, with rapid convergence to effective control policies. Moreover, the discrete codevectors enabled the visualization of the Q-table in lower-dimensional projections (e.g., 2D maps for robot positions), thereby enhancing the explainability of the robot's actions compared to opaque black-box methods. This supports safer operation and streamlined debugging in robotic systems. As a two-timescale SA framework, the algorithm theoretically converges to an optimal solution given sufficient data and time, as evidenced by stable performance trends in the CartPole environment. These advancements, particularly with ODA, underscore the algorithm's efficacy in achieving robust, interpretable, and efficient control, aligning with the objectives of CAMP.

The evaluation of results, conducted through quantitative metrics (e.g., reward for path efficiency with connectivity constraints) and qualitative trajectory analysis using Ericsson's dataset, confirmed that constructing a good reward function and using the hybrid algorithm resulted in efficient, collision-free, and communication-aware motion planning. Encoding the constraints into the reward function allowed the agent to adapt to QoS properties, such as handovers, and maintain a reliable network connection (measured using RSRP). These outcomes fully met the goal of enabling safe and efficient

navigation under uncertain network conditions without the need for a network model, as evident by the results presented in Chapter 5.

The method still has some drawbacks, namely the complexities of balancing multiple objectives or reliance on a single-robot setup. Similarly, the dataset currently only includes a static environment, lacking dynamic variations, which limits its real-world applicability. The limitations of this project are further detailed in Section 6.2. Reflecting on the project, the only thing I would strive to do differently would be to try harder to incorporate real robot experiments. Moving outside of the simulation would have resulted in higher external validity, and it would have paved the way to collect more data for the simulation. Section 6.3 outlines other possible directions that extend this project.

6.2 Limitations

The results of this project are promising, but that being said, they are not perfect. Several limitations need to be considered to accurately reflect on this project, and deduce what valid future work could be done to extend these findings.

Firstly, it is essential to note that during the experiments, a static environment with a single agent was used. This affects the external validity of the results, as in reality, the environments in which robots operate are dynamic, both in terms of obstacles and temporal network variations. For the former, this thesis aimed to design a global planner, and the actual low-level controller, which would be used to avoid dynamic obstacles (such as humans or other agents), is outside the scope (see Chapter 1). For the latter, the method was only tested on static network properties as it was limited to the provided data. Unfortunately, data with temporal variations was not available, and there was not enough time to collect it. In terms of the initial plan, this is a significant limitation; however, as discussed in Section 6.3, the given method has the potential to address temporal network variations.

Building on what was just said, the environment itself is also a minor limitation. For starters, using simulation is great for testing the method in a controlled environment; however, to fully evaluate its merits and drawbacks, it would need to be tested on the actual robot. Furthermore, this project utilized a single dataset, but at the time of the thesis, it was the only one that had been pre-processed and was available. Therefore, data availability and quality remain significant limitations; however, since the project concluded, more datasets have been created, allowing the method to be tested in the future.

In the CartPole experiments, a thorough comparison and evaluation of different methods was conducted. However, all of these methods were VQ methods, which limits the comparison. For the CartPole example, this is not a significant problem because there is an extensive literature on it; however, it would have been beneficial to try different methods for the CAMP experiments. An attempt was made to use an Actor-Critic model; however, it would have taken significantly longer to train it properly, and thus, it was deemed outside the scope of the project due to time constraints.

Finally, the performance of the method on simulated data was satisfactory, but it could still be refined. The weight parameters for the different constraints are currently at the designer's discretion. In theory, this should be a good thing because, for operation, it can be selected which constraints to focus on. However, in practice, this is not as simple as adding more constraints, as it can cause the agent to ignore the other constraints, depending on the weights assigned and scaling. Overall, as shown in the results (Chapter 5), finding appropriate weight parameters for the reward function can be hard, and thus, more experiments on the actual CAMP problem are required.

6.3 Future work

This project covered a significant portion of the initial plan; however, as identified in Section 6.2, there are still some remaining issues, improvements, and extensions to consider for possible future work.

The first, and most logical, next step is to test the method in either a realistic (Gazebo) simulation or by physically deploying it onto a robot. These would extend the simulation to model real-world constraints, such as nonholonomic motion, sensor noise, and bandwidth limits, while also incorporating temporal variations. This step would involve adapting the method to work with the ROS framework, creating a reward function based on the networking data, and adding a low-level controller. This is a crucial future step as it directly ties into Ericsson's goals of autonomous robotic navigation, and it can extend the current method to dynamic environments and possibly, multi-agent scenarios.

Alternatively, a robot could be used to collect a dataset with temporal variations in networking and utilize that data in the current Gymnasium simulation, thereby enabling a new dimension of experiments. The data could be collected by having multiple agents connect and switch between the base stations, and add extra devices connected to these base stations, as that would create fluctuations in the network. The current approach is likely to perform poorly under such reward conditions; however, there is

potential to mitigate this effect. In particular, targeted adaptation to time-dependent reward variation could be used without modifying the underlying state representation, due to decision-making directly relying on Q-values in response to changing rewards. Parametrized models, such as deep RL methods, can also adapt to changing reward functions through continued training; however, such adaptations are less transparent and may require more careful tuning. Otherwise, the training could lead to instability or even forgetting. Otherwise, incorporating more information into the state space to capture temporal or contextual variations is another viable approach. If encoded appropriately, the current method should extend to higher dimensions without any problems, as demonstrated with the CartPole example.

Furthermore, additional experiments are needed in the handover regions, as there are cases where the process fails. The best course of action is to adjust the weights and scaling of the corresponding reward parameters. Another option is to design adaptive or learned reward functions that dynamically balance the different constraints, rather than relying on fixed weightings. A multi-objective RL framework could explicitly model these trade-offs, producing Pareto-optimal policies. Ultimately, it is crucial to comprehend how this method differs from others in terms of performance, adaptability, and efficiency. This can include using other RL methods or a different set of methods altogether, as mentioned in Chapter 2.

6.4 Reflections

The development of communication-aware robotics contributes indirectly to the *sustainability* of autonomous systems by improving their resource utilization and operational efficiency. This project did not directly measure environmental impact, as it is constrained to a simulation environment; however, motion coordination and efficient communication align with sustainable system design. From an economic standpoint, optimizing motion with network awareness can lead to reduced energy consumption, transmission costs, and network congestion in connected robotic systems. In particular, this is particularly relevant for industries that deploy many autonomous agents. From an ecological perspective, creating energy-efficient path planning can lead to lower power requirements for networked robots, particularly when they operate on limited battery or in remote environments. The importance of sustainability is reflected in potential applications such as smart infrastructure, robot deliveries, and disaster response, where improved connectivity and coordination can lead to safer and more efficient operations.

This thesis project has very low *ethical risks* as it is entirely computational and does not involve any human participation, personal, or sensitive information. However, as outlined in Section 6.3, there is a possibility of deploying this method in autonomous decision-making systems in real-world settings, which raises ethical considerations. Hence, it is vital to ensure that future extensions of this project adhere to moral principles of fairness, responsibility, and explainability. Given that the backbone of this method is a RL algorithm, it should be designed with safeguards to ensure safe operation and prevent unexpected behaviors, which the low-level controller is intended to mitigate.

Finally, this research addresses the challenges that are increasingly *socially relevant*, as society moves towards pervasive connectivity (especially with 5G networks) and autonomous systems. The proposed CAMP framework can benefit various societal applications, including intelligent transportation and delivery, search and rescue operations, environmental monitoring, and warehouse management. These applications require a stable and reliable communication link, which is essential for safe, coordinated, and reliable operation. Therefore, this work contributes to a broader goal of integrating AI into autonomous systems to enhance their efficiency and reliability. Furthermore, the methodology of this project is highly relevant in both the industry and research, as it boosts an understanding of how learning-based systems can be both adaptable and explainable, while balancing multiple objectives in parallel. To summarize, in a broader sense, this work reflects the societal shift towards data-driven, adaptive, and connected technologies that support intelligence automation.

References

[1] A. Muralidharan and Y. Mostofi, "Communication-aware robotics: Exploiting motion for communication," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 115–139, 2021. [Pages 2, 11, and 22.]

[2] D. B. Licea, M. Bonilla, M. Ghogho, S. Lasaulce, and V. S. Varma, "Communication-aware energy efficient trajectory planning with limited channel knowledge," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 431–442, 2020. doi: 10.1109/TRO.2019.2948801 [Pages 2 and 3.]

[3] M. Lindhe and K. H. Johansson, "Communication-aware trajectory tracking," in *2008 IEEE International Conference on Robotics and Automation*, 2008. doi: 10.1109/ROBOT.2008.4543417 pp. 1519–1524. [Page 2.]

[4] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE wireless communications*, vol. 11, no. 6, pp. 6–28, 2004. [Page 2.]

[5] A. Gupta, K. Ghanshala, and R. C. Joshi, "Machine learning classifier approach with gaussian process, ensemble boosted trees, svm, and linear regression for 5g signal coverage mapping," *IJIMAI*, vol. 6, no. 6, pp. 156–163, 2021. [Pages 2 and 23.]

[6] L. U. Khan, W. Saad, D. Niyato, Z. Han, and C. S. Hong, "Digital-twin-enabled 6g: Vision, architectural trends, and future directions," *IEEE Communications Magazine*, vol. 60, no. 1, pp. 74–80, 2022. [Page 2.]

[7] S. De Bast, E. Vinogradov, and S. Pollin, "Cellular coverage-aware path planning for uavs," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2019. doi: 10.1109/SPAWC.2019.8815469 pp. 1–5. [Page 3.]

[8] A. Ghaffarkhah and Y. Mostofi, "Communication-aware motion planning in mobile networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2478–2485, 2011. [Page 3.]

[9] U. Ali, H. Cai, Y. Mostofi, and Y. Wardi, "Motion-communication co-optimization with cooperative load transfer in mobile robotics: An optimal control perspective," *IEEE Transactions on Control of Network Systems*, vol. 6, no. 2, pp. 621–632, 2018. [Page 3.]

[10] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Page 10.]

[11] M. Z. Chowdhury, M. Shahjalal, S. Ahmed, and Y. M. Jang, "6G wireless communication systems: Applications, requirements, technologies, challenges, and research directions," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 957–975, 2020. [Page 10.]

[12] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The Next Generation Wireless Access Technology*, 2nd ed. Amsterdam: Elsevier, 2020. ISBN 978-0-12-822320-8 [Page 11.]

[13] H. Rydén, H. Farhadi, A. Palaios, L. Hévizi, D. Sandberg, and T. Kvernvik, "Next generation mobile networks' enablers: Machine learning-assisted mobility, traffic, and radio channel prediction," *IEEE Communications Magazine*, vol. 61, no. 10, pp. 94–98, 2023. doi: 10.1109/MCOM.001.2200592 [Page 11.]

[14] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957. [Page 12.]

[15] V. S. Borkar and V. S. Borkar, *Stochastic approximation: a dynamical systems viewpoint*. Springer, 2008, vol. 100. [Pages 13 and 16.]

[16] C. N. Mavridis and J. S. Baras, "Annealing optimization for progressive learning with stochastic approximation," *IEEE Transactions on Automatic Control*, vol. 68, no. 5, pp. 2862–2874, 2022. [Page 16.]

[17] ——, "Online deterministic annealing for classification and clustering," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 10, pp. 7125–7134, 2022. [Pages 17 and 21.]

[18] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *Journal of machine learning research*, vol. 6, no. Oct, pp. 1705–1749, 2005. [Page 18.]

[19] C. N. Mavridis and J. S. Baras, "Convergence of stochastic vector quantization and learning vector quantization with bregman divergences," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 2214–2219, 2020. [Pages vii and 21.]

[20] C. N. Mavridis, N. Suriyarachchi, and J. S. Baras, "Maximum-entropy progressive state aggregation for reinforcement learning," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 5144–5149. [Pages vii, 24, 28, 41, 42, 43, 44, 45, 52, and 53.]

[21] C. N. Mavridis and J. S. Baras, "Vector quantization for adaptive state aggregation in reinforcement learning," in *2021 American Control Conference (ACC)*. IEEE, 2021, pp. 2187–2192. [Pages vii, 24, 28, 40, 42, and 43.]

[22] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962. doi: 10.2307/1266577 [Page 43.]