Hierarchical Policy Search Algorithms

Algorithmen zur hierarchischen Strategieoptimierung
Bachelor-Thesis von Kim Werner Berninger aus Miltenberg
Tag der Einreichung:

1. Gutachten: Christian Daniel, MSc.
2. Gutachten: Prof. Dr. Gerhard Neumann
3. Gutachten: Prof. Dr. Jan Peters

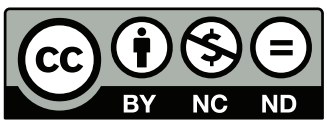
TECHNISCHE
UNIVERSITÄT
DARMSTADT

Hierarchical Policy Search Algorithms
Algorithmen zur hierarchischen Strategieoptimierung

Vorgelegte Bachelor-Thesis von Kim Werner Berninger aus Miltenberg

1. Gutachten: Christian Daniel, MSc.
2. Gutachten: Prof. Dr. Gerhard Neumann
3. Gutachten: Prof. Dr. Jan Peters

Tag der Einreichung:



Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 1. Oktober 2015

(Kim Werner Berninger)

Abstract

Policy gradient algorithms have taken on an important role in reinforcement learning since, compared to value function methods, they can cope better with continuous state and action spaces. However they also have their drawbacks for in general they tend to suffer from a vast amount of parameters as well as high variance in their gradient estimates. Furthermore, they are prone to getting stuck in local optima while ascending the estimated gradient.
This situation can be improved by applying a divide-and-conquer strategy as it already has been used in the value function field since its early days. Hereby, a complex problem can not only be split up in smaller subtasks in order to accelerate overall learning by introducing a hierarchical structure. The resulting smaller problems can be solved using several different kinds of learning algorithms.
In this thesis, we are going to compare performances of a conventional policy gradient algorithm with one employing the hierarchical approach by applying them to an example problem. The latter is going to utilize a value function based algorithm as well as policy gradient reinforcement learning. Those are composed to yield a so called hybrid policy gradient algorithm.

Zusammenfassung

Algorithmen zur numerischen Optimierung parametrisierter Strategiefunktionen haben im Bereich des bestärkenden Lernens einen hohen Stellenwert eingenommen, da sie, im Gegensatz zu Methoden, die zunächst den erwarteten Gesamtgewinn annähern müssen, besser mit kontinuierlichen Zustands- und Aktionsräumen umgehen können. Jedoch müssen auch hier Abstriche gemacht werden, da diese Verfahren im Allgemeinen mit großen Mengen an Parametern und hoher Stichprobenvarianz zu kämpfen haben und daher oft nur langsam konvergieren. Außerdem ist es möglich, dass sich die Optimierung in einem lokalen Maximum verfängt.
Abhilfe kann hier ein Ansatz nach dem Grundsatz „Teile und herrsche“ schaffen, wie er in den übrigen Verfahren schon seit geraumer Zeit eingesetzt wird. Auf diese Weise lässt sich ein großes zu lösendes Problem nämlich nicht nur in kleinere Teilprobleme zerlegen und somit durch den Aufbau einer hierarchischen Struktur effizienter erlernen, sondern es können auch verschiedene Arten von Lernalgorithmen für jeweilige Aufgaben benutzt werden.
Einen solchen hybriden hierarchischen Algorithmus möchten wir in dieser Arbeit näher betrachten und auf ein Beispielproblem anwenden. Weiterhin werden wir die Performanz dieser Lösung mit der eines klassischen Optimierungsalgorithmus vergleichen.

Acknowledgments

I want to thank my supervisors Christian Daniel and Gerhard Neumann for their support, their patience and for introducing me to such an interesting topic as well as keeping me motivated to approach it.
Furthermore I want to thank my friends and family for cheering me up whenever I was exhausted from spending several hours a day just starring at the plot of a simulated ship.

Contents

List of Figures

1 Introduction

When dealing with reinforcement learning problems comprising continuous state or action spaces, the usage of *value function methods*, in general, does not lead to a sufficient solution. For that reason *policy gradient reinforcement learning* is often used as an alternative in those domains. Instead of deducing the optimal policy from the value function, it is rather modeled in terms of parameters which are adjusted using gradient ascent in order to optimize the reward accomplished by following the resulting policy.

However, that kind of algorithms suffers from low performance in high-dimensional spaces incorporating huge sets of parameters to be learned. There have already been several approaches to applying a divide-and-conquer principle to those problems in order to accelerate their convergence.

Those hierarchical methods have been around since the dawn of reinforcement learning itself and have already been used in several value function contexts. However, they are still a topic of recent research.

A simple example of using that strategy involves applying domain knowledge about some problem in order to formulate smaller subtasks which are easier to solve using policy gradient. Then, a high-level planner can learn how to solve the original task by building a sequence of those low-level problems and using their performed actions as a solution to the high-level problem. If the number of subtasks is finite, it is then appropriate to use a value function method instead of policy search for this planner. In that case the overall algorithm is called a *hierarchical hybrid algorithm*.

1.1 A short example

As an introductory example we want to consider a home cleaning robot powered by a rechargeable battery which shall use a policy gradient algorithm in order to learn how to clean multiple rooms and arrive at its docking station whenever necessary without running out of power. In this setting the learning algorithm is confronted with a vast amount of parameters in a high dimensional continuous state space and would therefore take very long to reach its optimal policy.

The robot's task could be divided into two subproblems: cleaning just one room at a time and just driving back to the charging station. It could then learn solving those separately. Having done so the robot should learn how to sequence those actions so that it is able to use them for solving the whole problem.

This approach comes with several advantages. Firstly, often a specific sub-policy can be re-used for solving multiple similar problems, for example using the gained knowledge from one room for also cleaning other rooms. Secondly, on some levels of the resulting hierarchy some parameters of the state can be ignored completely and therefore the complexity can be reduced even more than just by splitting up the task space. For example the amount of dust in each room is indeed crucial for the cleaning subtask, however it is of no significance to the charging subtask.

1.2 Related work

The work with the greatest impact on this thesis is the work of Ghavamzadeh et al. [1]. It introduced hierarchical approaches to the world of policy gradient algorithms and proposed a framework for hierarchical reinforcement learning with arbitrarily chosen learning algorithms.

Many years before that, in the late 80s, hierarchical reinforcement learning has already been mentioned by Watkins when he introduced Q-Learning [2]. Also, he mentioned a ship steering problem similar to the one used by Ghavamzadeh as an abstract use case for hierarchical learning.

As already mentioned hierarchical methods have already successfully been applied to value function learning [3, 4].

A more recent approach to hierarchical policy gradient can be found in the work of Kroemer et al. [5] where it is used in order to learn robotic manipulation tasks as a sequence of separately learned movement primitives which are used in order to transition between the phases the robot's action depend on. That way the robot can learn some primitives by itself while still imitating some other primitives from a demonstrator. Also they can be reused for different kinds of manipulations.

Another use of an hierarchical approach is shown in the work of Daniel et al. [6] where it is used to augment Relative Entropy Search so that it becomes able to better adapt to different options an agent can follow in order to solve its task.

1.3 Outline

In this thesis we are first going to have a look at some foundations of reinforcement learning which we are going to need in order to inspect the hierarchical policy search. Among those are *Markov decision processes* as well as two example learning algorithms which we are going to use in order to compose the hierarchical hybrid algorithm.
Chapter 3 builds upon those foundations by introducing the concepts used by Ghavamzadeh et al. in order to formulate the hierarchical policy gradient. So we are going discuss task graphs, subtasks and hybrid policy gradient algorithms, among others.
In chapter 4 we are going to tackle the ship steering problem first introduced by Miller et al. [7] as a specific example problem. It will also be used in order to compare a flat policy gradient algorithm to a hierarchical one in chapter 5.
Finally we are going to reflect on the gathered insight and discuss how the approach in this paper could be improved using the latest advances in hierarchical policy gradient in chapter 6.

2 Foundations

In order to decompose a learning problem into a set of subtasks we formalize it as a special Markov decision process called *task graph*. We are first going to examine general MDPs and, subsequently, look at the hierarchical case. Since we are going to evaluate a hierarchical hybrid policy gradient algorithm we will also need at least one classical policy gradient algorithm for solving the subproblems as well as a value function based algorithm which learns when to execute which subtask. Those algorithms are going to be REINFORCE and classical Q-Learning.

2.1 Markov decision processes

A *Markov decision process* (MDP) is a mathematical model used to describe state transition systems inhabited by an agent. That agent interacts with its environment by performing different actions which can lead the state of the system to change. The agent can also receive a reward or punishment for each of its actions. Its objective is to optimize the reward it collects over time. Here, we are going to use a definition of MDPs which is similar to the one from the work of Sutton et al. [8].
An MDP consists of a set of states S and each of those states $\mathbf{s} \in S$ comes with a set of available actions A_s and a transition distribution $P_\mathbf{s} : A_\mathbf{s} \times S \to [0,1]$ with $\sum_{\mathbf{s}' \in S} P_\mathbf{s}\left(\mathbf{s}' \middle| \mathbf{a}\right) = 1$ for each $\mathbf{a} \in A_\mathbf{s}$. Given a state $\mathbf{s}_t$ at some discrete point in time t the agent can perform an action $\mathbf{a}_t \in A_{\mathbf{s}_t}$ in order to reach the next state $\mathbf{s}_{t+1}$ with probability $P_\mathbf{s}\left(\mathbf{s}_{t+1} \middle| \mathbf{a}_t\right)$. Furthermore we define a distribution $\mu : S \to [0,1]$ with $\sum_{\mathbf{s} \in S} \mu(\mathbf{s}) = 1$ where $\mu(\mathbf{s})$ is the probability of $\mathbf{s}$ being the initial state $\mathbf{s}_0$.
In order to be able to tell how good an agent is performing we define a score $R_{\mathbf{s},\mathbf{a},\mathbf{s}'} \in \mathbb{R}$ for each state $\mathbf{s} \in S$, each action $\mathbf{a} \in A_\mathbf{s}$ and each resulting state $\mathbf{s}' \in S$ with $P_\mathbf{s}\left(\mathbf{s}' \middle| \mathbf{a}\right) > 0$. Using those scores the reward function $r : S \times A \to \mathbb{R}$ can now be defined as

$$r(\mathbf{s},\mathbf{a}) = \mathbb{E}_{P_\mathbf{s}}\left[R_{\mathbf{s},\mathbf{a}',\mathbf{s}'} \middle| \mathbf{a}' = \mathbf{a}\right], \tag{2.1}$$

which describes the expected reward observed when performing action $\mathbf{a} \in A_{\mathbf{s}_t}$ in state $\mathbf{s} \in S$.
An MDP can be viewed as a bipartite graph with the states and actions being nodes and the transitions being the edges connecting them. An example is depicted in figure 2.1.

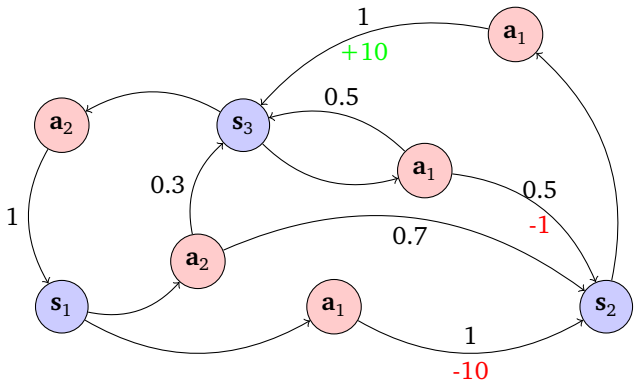


Figure 2.1.: An example MDP with three states and two actions. The labels on the edges from actions to states show the probabilities of the action resulting in the respective state. The red and green labels show the according scores R.

One important aspect of all MDPs is that they fulfill the *Markov property* which states that the probability of being in some state $\mathbf{s}_{t+1}$ in the system may only depend on its preceding state $\mathbf{s}_t$ and the action $\mathbf{a}_t$ between them, not on any other state or action prior to those. Let S_t and A_t be random variables which denote the state and action at a point in time t. Then the Markov property can be formalized as

$$p\left(S_{t+1} = \mathbf{s}_{t+1} \middle| S_t = \mathbf{s}_t, A_t = \mathbf{a}_t, \ldots, S_0 = \mathbf{s}_0, A_0 = \mathbf{a}_0\right) = p\left(S_{t+1} = \mathbf{s}_{t+1} \middle| S_t = \mathbf{s}_t, A_t = \mathbf{a}_t\right). \tag{2.2}$$

A distribution $\pi(\cdot|\mathbf{s}) : A_s \to [0,1]$ with $\sum_{\mathbf{a} \in A_s} \pi(\mathbf{a}|\mathbf{s}) = 1$ which returns the probability that a given action is executed in a given state $\mathbf{s} \in S$ is called a *policy*.

The solution to an MDP is a policy $\pi^\star = \text{argmax}_\pi J_\pi$ which optimizes the *expected long term reward*

$$J_\pi = \mathbb{E}_{\mu,\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]. \tag{2.3}$$

Here $\mathbf{s}_0$ is sampled according to $\mu(\cdot)$, $\mathbf{a}_t$ according to the policy $\pi\left(\cdot\middle|\mathbf{s}_t\right)$ and $\mathbf{s}_{t+1}$ according to the transition distribution $P_{\mathbf{s}_t}\left(\cdot\middle|\mathbf{a}_t\right)$ for each $t \in \mathbb{N}$. The formulation used here assumes that the underlying learning problem has an *infinite horizon*, i.e. there is no terminal state and it can run infinitely without having to be restarted. This assumption suffices for now, however finite horizon problems are also going to be discussed in section 2.3.
The value γ is a so called *discount factor* between 0 and 1 which gives more weight to the immediate rather than to the long-term reward. Hence, the infinite sum also is guaranteed to converge if r is bounded.
The optimal policy for the example MDP in figure 2.1 is

$$\pi(\mathbf{a}|\mathbf{s}) = \begin{cases} 1 & \text{if } \mathbf{s} = \mathbf{s}_1 \text{ and } \mathbf{a} = \mathbf{a}_2, \\ 1 & \text{if } \mathbf{s} = \mathbf{s}_2 \text{ and } \mathbf{a} = \mathbf{a}_1, \\ 1 & \text{if } \mathbf{s} = \mathbf{s}_3 \text{ and } \mathbf{a} = \mathbf{a}_1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

A way to compute this solution will be presented in section 2.2.

2.2 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm first introduced by Watkins [2] and one possible way of solving an MDP with a finite number of states and actions. It uses the so called *action-values* $Q(\mathbf{s}, \mathbf{a})$ for each state-action-pair $\mathbf{s} \in S$ and $\mathbf{a} \in A_\mathbf{s}$ for the deduction of an optimal policy.
In order to be able to correctly define those values, we first have to introduce the concept of *value functions*. They are one of the most fundamental concepts of reinforcement learning and are used in several different approaches of policy search., Many of those can be found in Sutton's an Barto's introductory book [9]. In this thesis, we are going to stick to their notations.
For a given policy π, a value function $V^\pi : S \to \mathbb{R}$ depends on an initial state s and describes the expected reward when following the policy starting from this state. A formal definition is given by

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\middle|\mathbf{s}_0 = \mathbf{s}\right], \tag{2.5}$$

with $\mathbf{a}_t$ and $\mathbf{s}_{t+1}$ each sampled as in equation 2.3 for each $t \in \mathbb{N}$. Since the first term of the sum inside the expectation does not depend on the transition probability we can reformulate the equation in a recursive form

$$V^\pi(\mathbf{s}) = \mathbb{E}_\pi\left[r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_P\left[V^\pi\left(\mathbf{s}'\right)\right]\middle|\mathbf{s}\right]. \tag{2.6}$$

A formal proof of that equality can be found in Appendix A.
Using the value function we can now define the action-value function $Q^\pi(\mathbf{s}, \cdot) : A_s \to \mathbb{R}$ for each state $\mathbf{s} \in S$ as

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_P\left[V^\pi\left(\mathbf{s}'\right)\middle|\mathbf{s}, \mathbf{a}\right]. \tag{2.7}$$

It yields the expected long term reward achieved by taking an action $\mathbf{a}$ in a state $\mathbf{s}$ and afterwards following the policy π. Now we consider a deterministic optimal policy $\pi^\star$, i.e. a policy which always certainly chooses the action $\mathbf{a}$ in a state $\mathbf{s}$ so that the long term reward becomes maximized. Such a policy yields the following optimal value function

$$V^\star(\mathbf{s}) = \max_{\mathbf{a}}\left(r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_P\left[V^\star\left(\mathbf{s}'\right)\middle|\mathbf{s}, \mathbf{a}\right]\right). \tag{2.8}$$

This equation is also known as the *Bellman equation* and could already be used in order to compute $V^\star$ given that the behavior of the environment P is known. The optimal policy $\pi^\star$ could then be inferred using $V^\star$ so that it certainly chooses the action for which the action-value is maximal.
Using that function we can further define the optimal state action function

$$Q^\star(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{P_\mathbf{s}}\left[V^\star\left(\mathbf{s}'\right)\middle|\mathbf{s}, \mathbf{a}\right]. \tag{2.9}$$

Using equations 2.8 and 2.9 the optimal value function can directly be computed from the optimal action-value function:

$$V^\star(\mathbf{s}) = \max_{\mathbf{a}} \left(Q^\star(\mathbf{s}, \mathbf{a})\right). \tag{2.10}$$

Now we assume to already have obtained estimates $\hat{V}$ and $\hat{Q}$ for the optimal value and state-action functions and observe a sample state transition $\left(\mathbf{s}, \mathbf{a}, \mathbf{s}', r\right)$ along with the gained reward from an experiment. Then the approximation

$$\hat{Q}(\mathbf{s}, \mathbf{a}) \approx r + \gamma \hat{V}\left(\mathbf{s}'\right), \tag{2.11}$$

should hold. The error experienced with this approximation is called the *temporal difference error* δ and can be used to improve the estimate of the state action function:

$$\delta = r + \gamma \hat{V}\left(\mathbf{s}'\right) - \hat{Q}(\mathbf{s}, \mathbf{a}) = r + \gamma \max_{\mathbf{a}} \left(\hat{Q}\left(\mathbf{s}', \mathbf{a}\right)\right) - \hat{Q}(\mathbf{s}, \mathbf{a}). \tag{2.12}$$

If δ is greater than zero the current estimate $\hat{Q}$ is too small, otherwise if it is less than zero, $\hat{Q}$ is too big. So $\hat{Q}$ can be improved by adding the temporal difference error with each experienced sample. For that a *learning rate* $alpha_t$ is used for each $t \in \mathbb{N}$ in order to control the step size of the improvement and not to execute too huge leaps.
With a stream of samples $\left(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_t\right)$ for $t \in \mathbb{N}$ that leads to a sequence

$$Q_{t+1}\left(\mathbf{s}_t, \mathbf{a}_t\right) = \begin{cases} Q_t\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha_t \left(r_t + \gamma \max_{\mathbf{a}} \left(Q_t\left(\mathbf{s}_{t+1}, \mathbf{a}\right)\right) - Q_t\left(\mathbf{s}_t, \mathbf{a}_t\right)\right) & \text{if } \mathbf{s} = \mathbf{s}_t \text{ and } \mathbf{a} = \mathbf{a}_t, \\ Q_{t+1}(\mathbf{s}, \mathbf{a}) & \text{otherwise,} \end{cases} \tag{2.13}$$

for each $\mathbf{s} \in S$ and $\mathbf{a} \in A_\mathbf{s}$. Watkins has proven [10] that for $t \to \infty$ this sequence converges to $Q^\star(\mathbf{s}, \mathbf{a})$ for each $\mathbf{s} \in S$ and $\mathbf{a} \in A_s$.
In contrast to the *value iteration* from equation 2.8 Q-Learning needs to keep track of a non-deterministic policy pi in order to produce the needed samples and enforce exploration of the state and action space. This policy should be formulated in a way that yields greater probabilities for actions with higher action-values.
An example for that is the *softmax policy*

$$\pi(\mathbf{a}|\mathbf{s}) = \frac{\exp\left(Q(\mathbf{s}, \mathbf{a})/\tau\right)}{\sum_{\mathbf{a}' \in A} \exp\left(Q(\mathbf{s}, \mathbf{a}')/\tau\right)}, \tag{2.14}$$

for each $\mathbf{s} \in S$ and $\mathbf{a} \in A_s$. The term τ is called the *temperature* and is used for dampening the difference in probabilities for each action.

2.3 The REINFORCE learning algorithm

From now on we are going to focus exclusively on *episodic learning problems*, i.e. problems which include a set of terminal states and eventually have to be restarted instead of being able to run infinitely.
In contrast to Q-Learning which has to rely on finite as well as discrete sets of actions and states, the next kind of learning algorithms being discussed is also able to cope with continuous spaces natively.
Policy gradient methods do not use the value function in order to deduce the optimal policy. Instead the policy is modeled using a set of parameters $\boldsymbol{\theta}$ which is fine-tuned by means of optimization.
An extensive introduction to policy gradients can be found in the work of Deisenroth et al. [11]. The short overview given in this paper reproduces their explanations and slightly adapts them to the conventions used here.

Consider an episode of an agent interacting with its environment. It is represented by a trajectory

$$\boldsymbol{\tau} = \left(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\right), \tag{2.15}$$

consisting of a finite sequence of T states and the actions connecting those.
Traditionally continuous states and actions are often denoted with different characters than discrete ones. However we are going to stick to the already established notation in order to avoid ambiguities as soon as we are working with coordinates in chapter 4.
Using the trajectory notation from equation 2.15 the reward for one episode can then be defined as

$$R(\boldsymbol{\tau}) = r\left(\mathbf{s}_T\right) + \sum_{t=0}^{T-1} r\left(\mathbf{s}_t, \mathbf{a}_t\right). \tag{2.16}$$

Again $r(\mathbf{s}, \mathbf{a})$ is used to denote the expected reward for executing an action $\mathbf{a}$ in a state $\mathbf{s}$. Additionally $r(\mathbf{s}_T)$ is defined as the reward observed for reaching the terminal state $\mathbf{s}_T$. A discount factor is not necessary since the finite sum is guaranteed to converge regardless of it.
Using the same probabilities μ and P as above as well as the parameterized policy π_θ the probability $p_\theta(\tau)$ of observing a trajectory τ can be defined as

$$p_\theta(\tau) = \mu(\mathbf{s}_0) \prod_{t=0}^{T-1} \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) P_{\mathbf{s}_t}(\mathbf{s}_{t+1} | \mathbf{a}_t). \tag{2.17}$$

Just as in the infinite horizon case the objective is to find a policy which maximizes the expected reward

$$J_\theta = \mathbb{E}[R(\tau)] = \int R(\tau) p_\theta(\tau) \mathrm{d}\tau. \tag{2.18}$$

However in the policy gradient setting this is done by adjusting θ so that J_θ becomes maximal. Therefore its gradient $\nabla_\theta J_\theta$ is computed and used in order to perform a gradient ascent by adding it to the current parameter

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_{\theta_t} J_{\theta_t}. \tag{2.19}$$

α_t is again used as the learning rate.
There are several ways of computing that gradient. One well known set of methods are the so called *likelyhood-ratio* or *REINFORCE* algorithms which have been introduced by Williams [12]. Those use an equation called the *likelyhood-ratio-trick* which can be easily reconstructed using the chain rule of derivation

$$\nabla_x \log(f(x)) = \frac{\nabla_x f(x)}{f(x)} \Leftrightarrow \nabla_x f(x) = \nabla_x \log(f(x)) f(x). \tag{2.20}$$

Thus, $\nabla_\theta J_\theta$ can be expressed as

$$\begin{aligned} \nabla_\theta J_\theta =& \nabla_\theta \int R(\tau) p_\theta(\tau) \mathrm{d}\tau \\ =& \int R(\tau) \nabla_\theta p_\theta(\tau) \mathrm{d}\tau \\ =& \int R(\tau) \nabla_\theta \log(p_\theta(\tau)) p_\theta(\tau) \mathrm{d}\tau \\ =& \mathbb{E}_{p_\theta}[R(\tau) \nabla_\theta \log(p_\theta(\tau))]. \end{aligned} \tag{2.21}$$

Since the logarithm transforms the product from equation 2.17 into a sum it can be further simplified to

$$\begin{aligned} \nabla_\theta \log(p_\theta(\tau)) =& \nabla_\theta \left(\log(\mu(\mathbf{s}_0)) + \sum_{t=0}^{T-1} \left(\log(\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) + \log(P_{\mathbf{s}_t}(\mathbf{s}_{t+1} | \mathbf{a}_t)) \right) \right) \\ =& \sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)). \end{aligned} \tag{2.22}$$

Combining the last two equations the gradient $\nabla_\theta J_\theta$ can eventually be expressed as following expected value

$$\nabla_\theta J_\theta = \mathbb{E}_{p_\theta} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)) \right) R(\tau) \right]. \tag{2.23}$$

Since computing that expected value would be unfeasible because of the integral a better approach would be to draw sample trajectories from the distribution p_θ and then use a *Monte Carlo method* in order to estimate it.

That procedure however would lead to a huge variance due to possibly noisy samples. In order to keep it at least as little as possible it is usual to subtract a term b from the rewards which is used to minimize the variance without introducing a bias to the estimation. That term is called a baseline. This is possible since

$$
\begin{aligned}
&\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)\left(R(\boldsymbol{\tau})-b\right)\right] \\
=&\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)R(\boldsymbol{\tau})-\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)b\right] \\
=&\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)R(\boldsymbol{\tau})\right]-\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)b\right] \\
=&\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}-b\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}}\log\left(\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)\right] \\
=&\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}-b\int\nabla_{\boldsymbol{\theta}}\log\left(p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right)p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\,\mathrm{d}\boldsymbol{\tau} \\
=&\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}-b\int\nabla_{\boldsymbol{\theta}}p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\,\mathrm{d}\boldsymbol{\tau} \\
=&\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}-b\nabla_{\boldsymbol{\theta}}\int p_{\boldsymbol{\theta}}(\boldsymbol{\tau})\,\mathrm{d}\boldsymbol{\tau} \\
=&\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}-b\nabla_{\boldsymbol{\theta}}1 \\
=&\nabla_{\boldsymbol{\theta}}J_{\boldsymbol{\theta}}.
\end{aligned}
\tag{2.24}
$$

The value for b is chosen so that it minimizes the variance of the estimate. In order to compute it, the derivative of the gradient's variance is set to 0 and the resulting equation is solved for b. This yields a baseline

$$
b_h=\frac{\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}_h}\log\left(\pi_{\boldsymbol{\theta}_h}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)^2R(\boldsymbol{\tau})\right]}{\mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}_h}\log\left(\pi_{\boldsymbol{\theta}_h}\left(\mathbf{a}_t\middle|\mathbf{s}_t\right)\right)\right)^2\right]},
\tag{2.25}
$$

for the gradient of the expected reward with respect to the h^{th} component of $\boldsymbol{\theta}$.
The Monte Carlo estimates of the expected values in the equations 2.23 and 2.25 can be computed by taking the average over the sum of sampled values. That leads to the following estimates $\nabla_{\boldsymbol{\theta}_h}J_{\boldsymbol{\theta}}^{\text{RF}}$ and b_h^{RF} for the gradient and the baseline:

$$
\nabla_{\boldsymbol{\theta}_h}J_{\boldsymbol{\theta}}^{\text{RF}}=\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}_h}\log\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{[i]}\middle|\mathbf{s}_t^{[i]},t\right)\left(R^{[i]}-b_h^{\text{RF}}\right),
\tag{2.26}
$$

$$
b_h^{\text{RF}}=\frac{\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}_h}\log\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{[i]}\middle|\mathbf{s}_t^{[i]},t\right)\right)^2R^{[i]}}{\sum_{i=1}^{N}\left(\sum_{t=0}^{T-1}\nabla_{\boldsymbol{\theta}_h}\log\pi_{\boldsymbol{\theta}}\left(\mathbf{a}_t^{[i]}\middle|\mathbf{s}_t^{[i]},t\right)\right)^2}.
\tag{2.27}
$$

3 Hierarchical Task Decomposition

In this chapter, the task decomposition used in order to introduce a hierarchical structure to a learning problem shall be discussed. We are going to reflect on the propositions made by Ghavamzadeh et al. [1] as well as Dietterich et al. [3] and slightly adapt them so that they work for the problems addressed in this thesis. However we are not going to need all of Ghavamzadeh's assumptions since the learning algorithms used in this thesis are already established and, thus, there is no need for the step based policy gradient algorithm described alongside the hierarchical learning in their paper.

3.1 Task Graphs

A *task graph* is a special form of an MDP which offers a hierarchical decomposition to the problem which is modeled by it. It consists of a finite set of subtask MDPs $\left\{M^0, \ldots, M^n\right\}$ which are interconnected and form a hierarchy with the root MDP M^0 at the highest level. The lowest level MDPs make up the primitive subtasks which directly perform a primitive action without moving deeper in the hierarchy.
Each MDP M^i has its own policy π^i, a set of states S^i and each state $\mathbf{s}$ a set of actions $A^i_{\mathbf{s}}$ as well as a transition probability P^i_s. Furthermore there is a set of initial states I^i as well as the terminal states T^i and a reward function r^i.
Whenever a non-primitive task in the hierarchy performs an action it executes the respective subtask in a manner similar to subroutines in a computer program, i.e. when the subtask terminates the execution returns to the original task and reports the new state.
The combination of all policies $\pi = \left\{\pi^0, \ldots, \pi^n\right\}$ forms the *hierarchical policy* and is optimal iff it solves the root MDP.
As in Ghavamzadeh's work we require all of the subproblems als well as the root problem to be episodic, so that eventually every subtask call terminates and returns to its respective caller.
The decomposition of a regular MDP for a given problem requires domain knowledge in order to identify its underlying structure and using it for the formulation of the task graph.
A heavily simplified example task graph for the cleaning robot problem described in 1.1 is depicted in figure 3.1. The *drive* subtask is an example for a reusable task since it is shared by both the *charge* and the *clean* tasks.

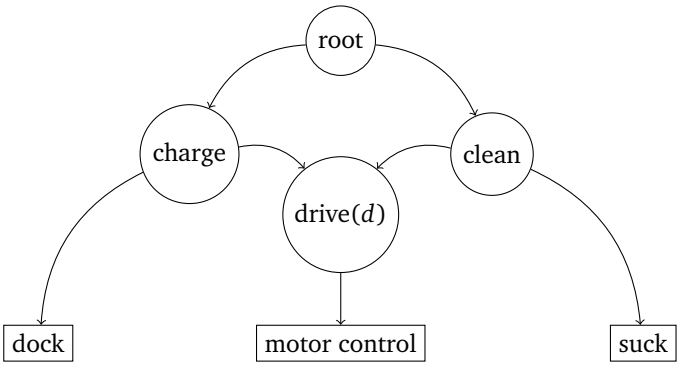


Figure 3.1.: An example taskgraph. Non-primitive actions are depicted by circular nodes, primitive ones by rectangular nodes.

3.2 Parametrized Subtasks

In addition to the parameters which are learned by a non-primitive subtask its policy can also incorporate further parameters which can be set by its caller rather than by the subtask itself. This approach is only used implicitly in [1] but it is specifically made use of in Dietterich's work [3] and provides even better chances for a subtask to be re-used in different occasions. In the above example the *drive* subtask could for example come equipped with a choosable destination d as it implied by its notation.
Also the caller of a parametrized subtask can add the parameters used to call it to its set of actions and learn which one to apply to a subtask execution in a certain state.
How this approach is going to be used is exemplified in section 4.5.

4 Experiments

4.1 The ship steering problem

In order to evaluate the performance of an HPG algorithm in comparison to the flat approach we are going to consider the ship steering problem introduced by Miller et al. [7].
The given scenario consists of a ship which is located at coordinates x_t and y_t in a continuous two-dimensional space and moving at a constant speed $V = 3\frac{m}{s}$ in some direction θ_t at some point in time t. The ship is able to adjust its turning rate $\dot{\theta}_t$ and therefore change the direction in which it is moving. However due to the water's inertia this turning rate does not change immediately but rather converges to its desired value u_t in some time frame T. The ship's state is updated every time the sampling interval $\Delta = 0.2s$ is passed and it can therefore be described using the following formulas:

$$x_{t+1} = x_t + \Delta V \sin\left(\theta_t\right), \tag{4.1}$$

$$y_{t+1} = y_t + \Delta V \sin\left(\theta_t\right), \tag{4.2}$$

$$\theta_{t+1} = \theta_t + \Delta\dot{\theta}_t, \tag{4.3}$$

$$\dot{\theta}_{t+1} = \dot{\theta}_t + \Delta\frac{1}{T}\left(u_t - \dot{\theta}_t\right). \tag{4.4}$$

Now our goal is to maneuver the ship through a given target line or near a specific point on the surface. We are first going to find a solution to the second problem and then use the first one in order to compare our results to those of Ghavamzadeh et al. in [1].
The original problem is formulated for a coordinate system with its x- and y-axes reaching from 0 to 1000. Thus we are going to try to solve it for this space using the flat REINFORCE algorithm as well as an HPG algorithm which utilizes REINFORCE for solving a set of simplified subtasks and Q-Learning for learning how to construct a sequence of those which eventually solves the top-level task.
As subtasks we consider movements of constant length in eight different directions, four of them diagonal and the other four horizontal/vertical. Also we are going to simplify the learning of those by combining the horizontal/vertical as well as the diagonal movements each as one subtask and rotating the space in order to apply them in four different directions. For those subtasks we map the problem onto a coordinate system bounded from 0 to 150 in both dimensions.

4.2 Policy formulation

Given a current state $\mathbf{s}_t = \left(x_t, y_t, \theta_t, \dot{\theta}_t\right)^T$ we use a stochastic policy $\pi_{\mathbf{w},\sigma}\left(a_t \middle| \mathbf{s}_t\right)$ for the flat and the low-level tasks in order to sample the desired turning rate a_t which should be applied in this state. This distribution is modeled as a Gaussian distribution with its mode being linearly dependent on some N-dimensional feature vector $\boldsymbol{\phi}\left(\mathbf{s}_t\right)$ using weights $\mathbf{w} = \left(w_1, \ldots, w_N\right)^T$. So altogether the parameters which are going to be learned by the policy gradient are $\beta = \{\mathbf{w}, \sigma\}$ with σ^2 being the variance of the stochastic policy

$$\pi_{\mathbf{w},\sigma}\left(a_t \middle| \mathbf{s}_t\right) = \mathcal{N}\left(a_t \middle| \mathbf{w}^T\boldsymbol{\phi}\left(\mathbf{s}_t\right), \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\left(a_t - \mathbf{w}^T\boldsymbol{\phi}\left(\mathbf{s}_t\right)\right)^2}{2\sigma^2}\right). \tag{4.5}$$

After being drawn from the policy the turning rate is narrowed down to the space between -15 and 15 in order to avoid too large control signals using a variant of the sigmoid function

$$a_t' = \frac{30}{1 + e^{-a_t}} - 15. \tag{4.6}$$

For the high-level planner we use an ϵ-greedy policy which is defined using the action-values $Q(s,a)$ for each state $s \in S$ and subtask $a \in A_s$

$$\pi(a|s) = \begin{cases} 1 - \epsilon & \text{if } a = \underset{a'}{\operatorname{argmax}}\, Q\left(s, a'\right) \\ \frac{\epsilon}{|A|} & \text{otherwise.} \end{cases} \tag{4.7}$$

4.3 Feature extraction

For the features we use a *CMAC function approximator* [13, 14]. In order to do so the state space is discretized using a grid of the same dimension so that this results in a countable number of tiles. Of course the state space itself has to be bounded. Now we can define the feature vector $\boldsymbol{\phi}(\mathbf{s}_t) = (\phi_1(\mathbf{s}_t), \ldots, \phi_N(\mathbf{s}_t))^T$ with an activation function $\phi_j(\mathbf{s}_t)$ being defined for each tile $j \in \{1, \cdots, N\}$ as

$$\phi_j(\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{s}_t \text{ lies in tile } j \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

In addition the number of parameters can be increased by using multiple tilings of the same size and translating each by a different offset. Since there are now actions which can be expressed as the sum of tile weights the feature vector gets more expressive. With each added offset tiling the number of actions which can be expressed grows polynomially.

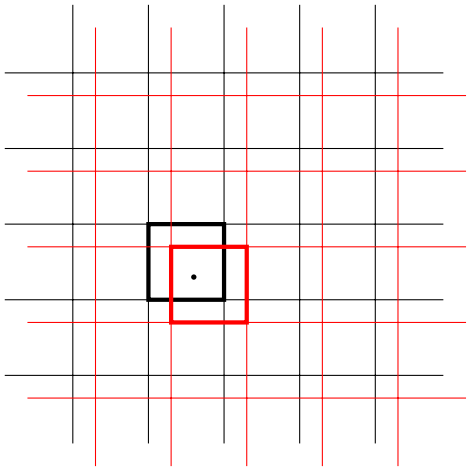


Figure 4.1.: An example tiling in a two-dimensional space using two different offsets. The black dot depicts a state and the thick tiles are the ones being activated by it.

4.4 The reward function

As reward function a punishment according to the squared Euclidian distance of the ship to its respective destination $\mathbf{t}$ in the two-dimensional coordinate system is used at the top as well as at the bottom level. Since the outcome of performing an action in a given state is deterministic the expected reward per step

$$r(\mathbf{s}, a) = R_{\mathbf{s}a\mathbf{s}'} = \left\| (x', y')^T - \mathbf{t} \right\|^2, \tag{4.9}$$

can directly be defined as the score $R_{\mathbf{s}a\mathbf{s}'}$ achieved for reaching a state $\mathbf{s}' = \left(x', y', \theta', \dot{\theta}'\right)^T$ by performing the according action a in an original state $\mathbf{s}$.
The overall reward

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} r(\mathbf{s}_t, a_t), \tag{4.10}$$

for one episode $\boldsymbol{\tau}$ defined as in equation 2.15 of the task is the sum of the rewards for each step with T being the number of steps performed in order to reach a terminal state.
However we have to consider that the state space is bounded and an episode has to terminate when the ship moves out of those bounds. Without further adjustment the learning algorithm would now prefer actions which move the ship out of bounds as fast as possible in order to avoid further punishment.

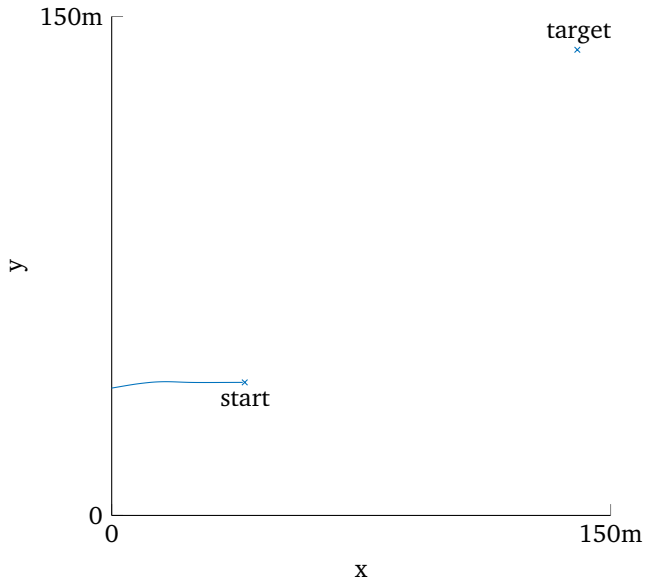


Figure 4.2.: A local optimum under the reward model without a fixed finite horizon. The blue line depicts the ship's trajectory.

This problem can be solved by considering a finite horizon formulation for this task. We assume a maximum amount of steps $T_{\max}$ and abort an episode as soon as it has reached it. When an episode has already terminated after $T < T_{\max}$ steps we also add the last squared distance multiplied with the missed number of steps to the accumulative reward which eventually has the form

$$R(\boldsymbol{\tau}) = -\sum_{t=1}^{\min\left(T,T_{\max}\right)} \left\| \left(x_t, y_t\right)^T - \mathbf{t} \right\|^2, \tag{4.11}$$

with $\left(x_t, y_t\right)^T = \left(x_T, y_T\right)^T$ for all $t \geq T$.

4.5 Subtasks used

Analogous to Ghavamzadeh's approach we also use two subtasks both taking place in a smaller coordinate system which reaches from 0 to 150 on both axes.
In the first subtask the ship's initial position is located at $(40, 40)$ and its target lies at $(140, 140)$. We call that one the *diagonal subtask*. In the second one the ship has to navigate from $(40, 75)$ to $(100, 75)$. This task is called the *horizontal subtask*. Figure 4.3 shows a sample drawn from each policy learned in order to solve those subtasks.

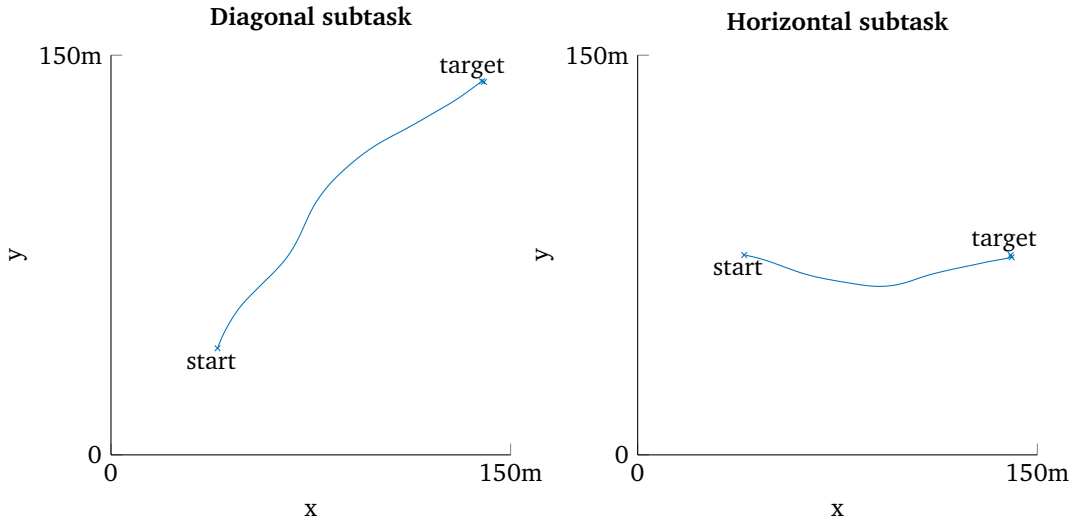


Figure 4.3.: Example trajectories for the diagonal (left) and horizontal (right) subtasks.

Just like in [1] we are also going to reuse each subtask four times, each time rotated by 90 degrees. That angle is passed to the subtask as a parameter upon execution by the top level planner. That makes a total of eight subtasks for it to choose from.
However since the original objective of simply passing through a gate shall also be exchanged by reaching a small area around a specific point in the big coordinate system those eight subtasks are not sufficient as the distances covered by each subtask are just too big for hitting a small target. A result of that problem is shown in figure 4.4. So for this experiment we add one further parameter besides the rotation angle to the subtasks which is going to be the distance to the subtarget which the substate should be initialized with. Since the ship will still be mapped to a state on the line between the original initial substate and the target and is therefore likely to pass tiles for which it already gained knowledge, this procedure is not going to cause that much overhead at least for the low level learner.

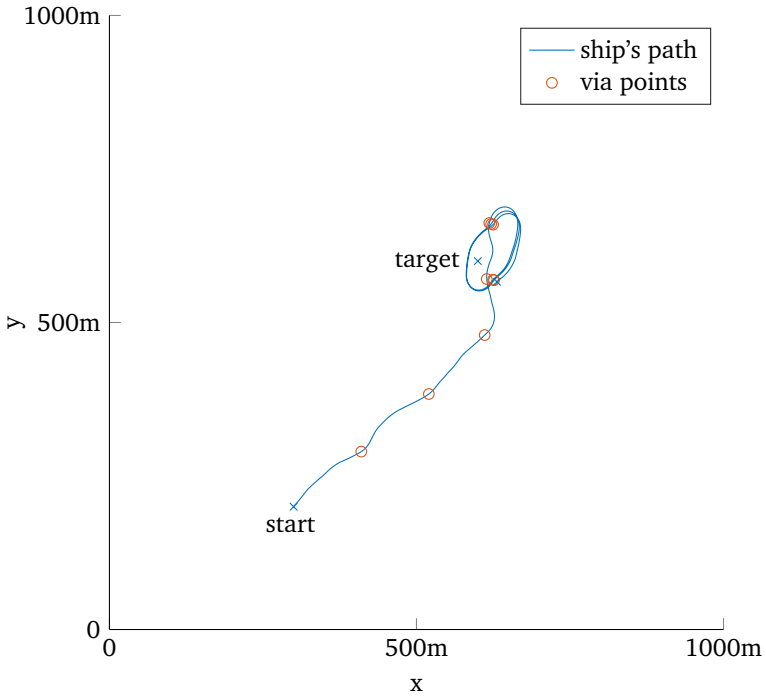


Figure 4.4.: With a fixed length traversed with each subtask execution it is almost impossible to exactly reach a small region. Therefore the ship sails in circles around it.

4.6 Transforming the states

Whenever a subtask is called by the top level planner the state of the ship has to be mapped to the according substate in order to properly being able to compute the actions to be performed. Let $\mathbf{s}'_{\text{sub}}$ be the x- and y-coordinates of the original starting state for the called subtask and $\mathbf{t}'_{\text{sub}}$ be the respective subtarget in the subtask's own coordinate system. Furthermore let $\mathbf{s}_{\text{sub}}$ and $\mathbf{t}_{\text{sub}}$ be their respective coordinates with respect to the top level coordinate system.
Upon calling a subtask its low level coordinate system is rotated by an angle α using a rotation matrix $R(\alpha)$ and then translated by a vector $\mathbf{o}$ so that the current state of the ship $\mathbf{s}$ gets mapped to its low level representation $\mathbf{s}'$ like

$$\mathbf{s} = R(\alpha)\mathbf{s}' + \mathbf{o}. \tag{4.12}$$

In order to obtain the states low level representation we have to find $\mathbf{o}$.
Let l be the distance $\mathbf{s}'$ is away from $\mathbf{t}'_{\text{sub}}$ in the direction of $\mathbf{s}'_{\text{sub}}$. Then it holds that

$$\mathbf{s}' = \mathbf{t}' + l\frac{\mathbf{s}'_{\text{sub}} - \mathbf{t}'_{\text{sub}}}{\left\|\mathbf{s}'_{\text{sub}} - \mathbf{t}'_{\text{sub}}\right\|}. \tag{4.13}$$

Combining equations 4.12 and 4.13 yields

$$\mathbf{o} = \mathbf{s} - R(\alpha)\left(\mathbf{t}' + l\frac{\mathbf{s}'_{\text{sub}} - \mathbf{t}'_{\text{sub}}}{\left\|\mathbf{s}'_{\text{sub}} - \mathbf{t}'_{\text{sub}}\right\|}\right). \tag{4.14}$$

In addition to transforming the ship's x- and y-coordinate its course also has to be adapted to its orientation in the low level space. The turning rate stays the same.

5 Results

We have conducted three major experiments. First the ship learned solving the original problem of passing a gate of a certain width using only a flat REINFORCE algorithm. Then this experiment has been repeated using the hierarchical approach described in the previous chapter. And finally the ship's objective has been made a bit more complex by requiring it to exactly hit a small region around a target point.
Each of those experiments generated 20000 episodes and produced at least a graph showing the number of successful episodes in 1000 samples. Those can be compared to those by Ghavamzadeh et al. in [1] since they used the same kind of performance measure.

5.1 Passing through the gate

For this problem the ship is placed at a uniformly random point in a 1000-by-1000 coordinate system. Also its course and turning rate are picked randomly. The ship's objective is to pass a horizontal gate with its center positioned at $(600, 600)$ and a width of 100m.

5.1.1 Flat

For the flat learner we use a plain REINFORCE gradient ascent with a learning rate $\alpha_t = 10 \cdot (0.9)^t$ and a normalized gradient estimate. The tiling used divides the four-dimensional coordinate system in a $10 \times 10 \times 10 \times 5$ board with no further offsets in order to keep the algorithm performant. An episode terminates as soon as a step crosses the target line, the ship leaves the bounds of the coordinate system or it reaches 6000 steps. The gradient is estimated using 1000 samples at a time.
The learner's performance is depicted in figure 5.1. The ship steering problem does not seem unsolvable using a flat approach, however it does not even achieve a success rate of 50 percent after 20000 episodes.

5.1.2 Hierarchical

The hierarchical approach to the ship steering problem uses a 20×20 tiling for the x- and y-coordinate and ignores the ship's course and turning rate at the top level. All four state parameters are propagated to the low level whenever a subtask is called. Those use a tiling size of $5 \times 5 \times 10 \times 5$ with a total of three equidistant offsets. Both tilings a consistent with the ones used for the same problem in [1].
The gradient estimates of the low level learners are computed from 500 samples each, normalized and applied with a constant learning rate of $\alpha_t = 1$. The Q-Learning algorithm at the top level uses an ϵ-greedy policy with $\epsilon = 0.2$.
A low level episode gets terminated when the ship leaves the bounds, hits a circular area of radius 10 around the low level target or reaches 600 steps. At the top level an episode lasts a maximum of ten subtask calls or until one subtask has successfully crossed the gate.
Figure 5.2 shows the performance of the hierarchical learner and compares it to the results from the previous section. Although it starts without knowledge of how to solve the subtasks and therefore first has to improve those before even being able to perform meaningful learning at the top level, it manages to reach a decent performance very quickly. This graph also confirms that our algorithm is quite close to its inspiration since the corresponding figure from [1] looks almost the same. Figures 5.3 and 5.4 depict the performance of the subtask learners in the same way as the previous graphs as well as using the mean reward of each iteration. Those are also relatively similar to the ones of [1] despite using a different algorithm.

5.2 Hitting the region

Except for some minor changes this modification of the original learning task uses the same setup. The main difference is that each top as well as low level episode terminates as soon as it reaches a circular area with a radius of 10 around the point (600, 600). That does not only mean that the top level goal is harder to achieve. In addition a low level episode which reaches the top level target gets terminated right away without being able to adapt its reward to the situation. That circumstance could lead to an increase of the gradient estimator's variance.

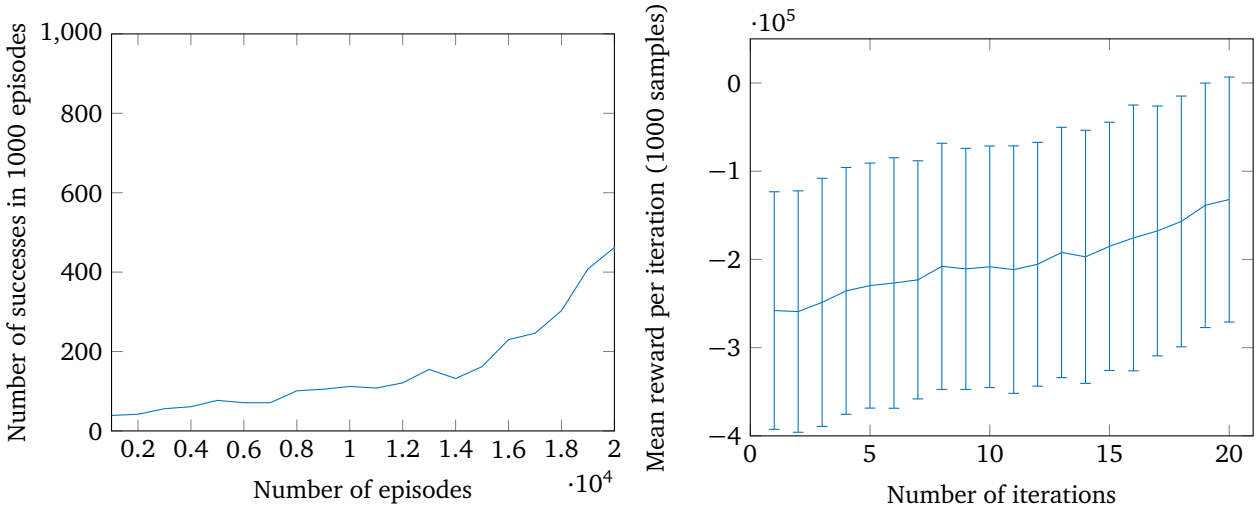


Figure 5.1.: The performance of the flat learner depicted as the number of successful episodes in 1000 samples (left) and as the mean reward of 1000 samples (right). The error bars in the plot on the right represent two times the standard deviation of each mean.

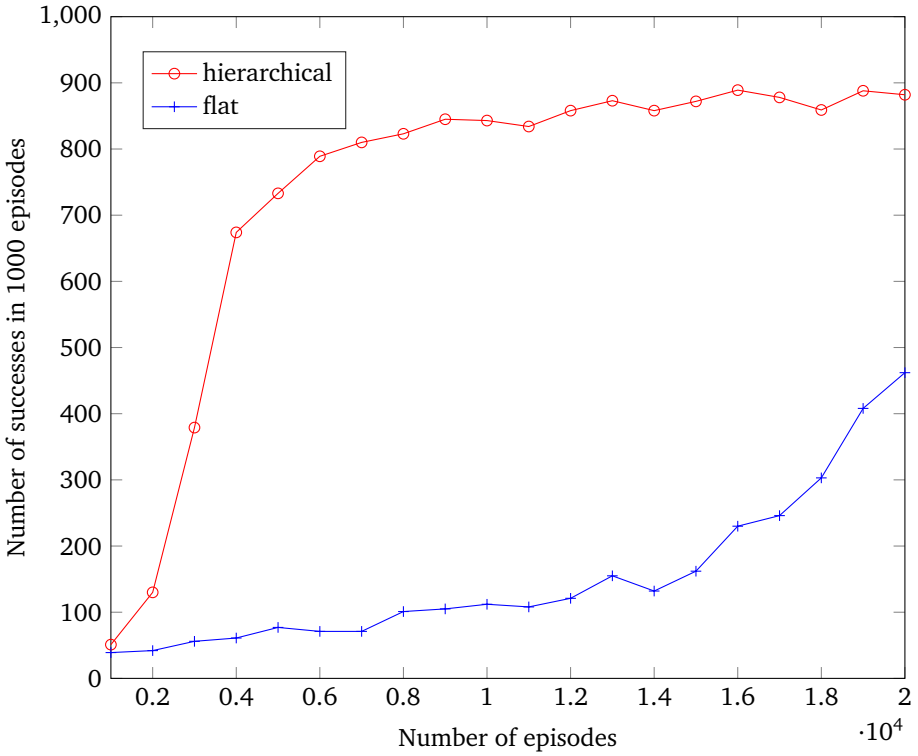


Figure 5.2.: The performance of the hierarchical learner depicted as the number of successful episodes in 1000 samples compared to the one of the flat learner.

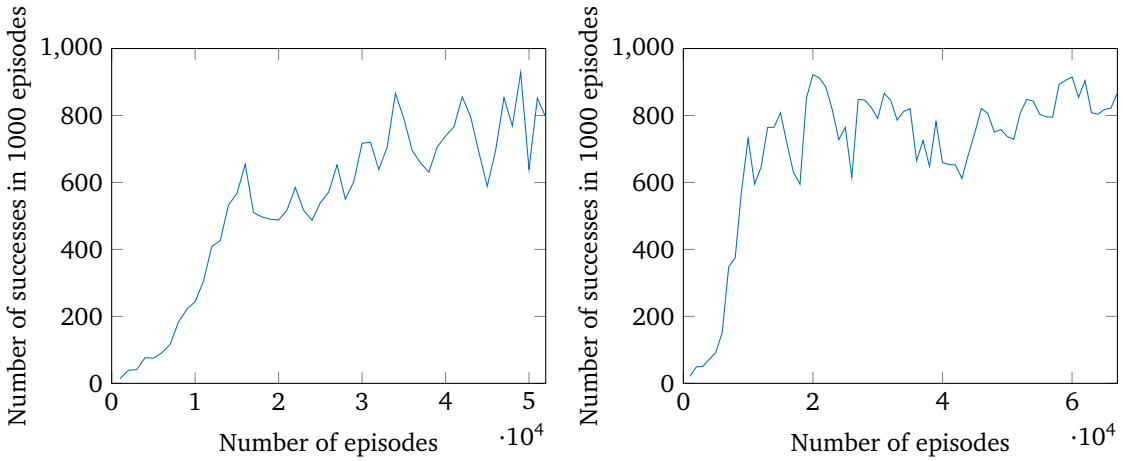


Figure 5.3.: The success rate of the diagonal (left) and horizontal (right) subtask per 1000 samples.

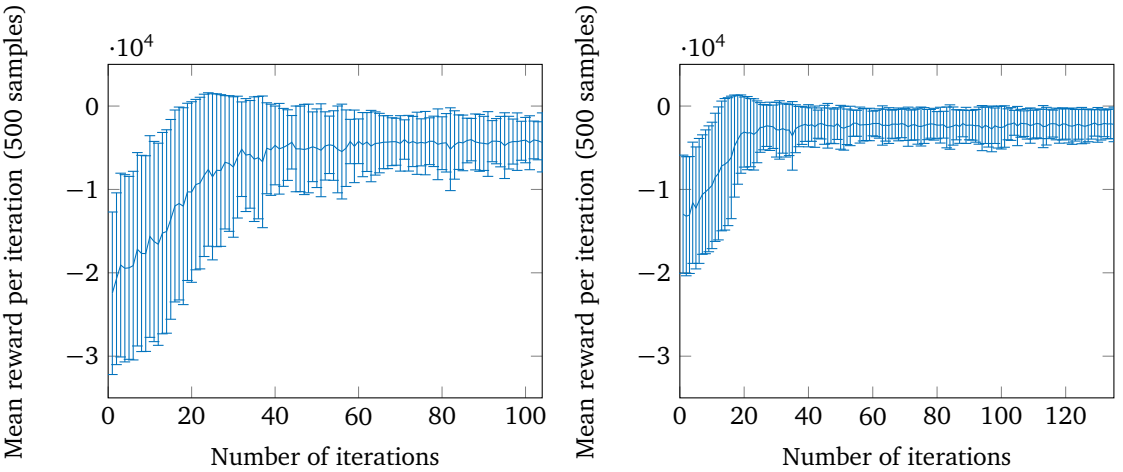


Figure 5.4.: The mean reward of the diagonal (left) and horizontal (right) subtask per 500 samples drawn each policy iteration. The bars show two times the standard deviation.

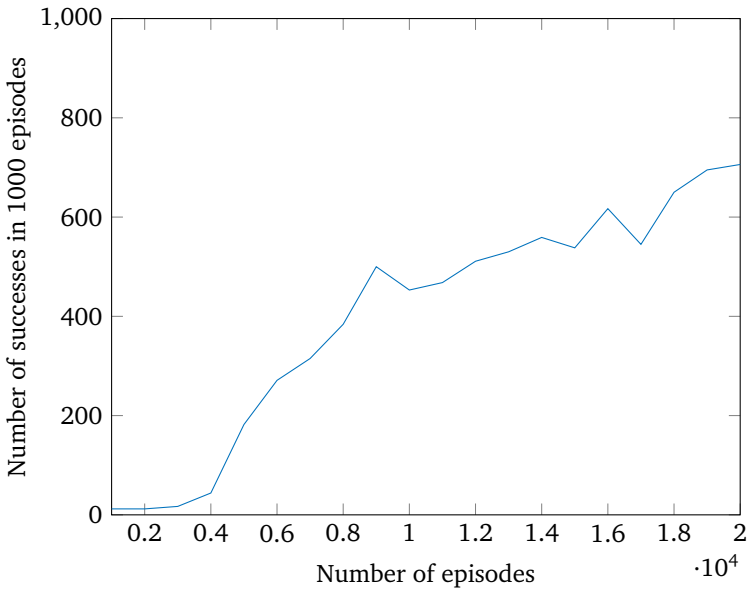


Figure 5.5.: The performance of the hierarchical learner solving the more challenging task depicted as the number of successful episodes in 1000 samples.

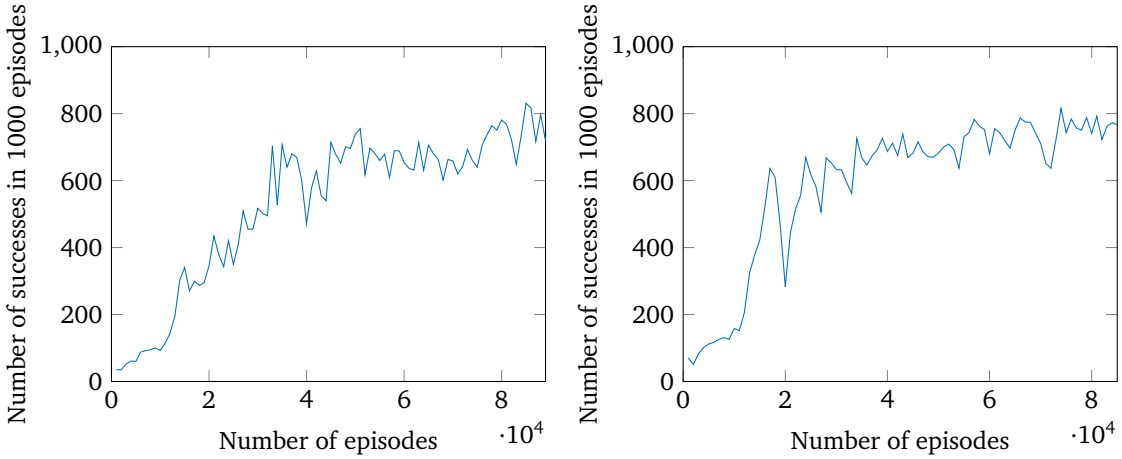


Figure 5.6.: The success rate of the low level learners in the third experiment. Again the diagonal subtask is on the left and the horizontal one on the right.

As already mentioned in section 4.5 we also have to compensate for the target being much smaller than the ship's movements by having the top level learner adapt the distance traversed by a subtask call. Therefore the distance between the original initial substate and the subtarget is split into five discrete equidistant segments and the state is mapped to one of those.
The performance achieved in this task can be seen in figure 5.5. A maximal success rate of 70 percent is relatively high since it must be considered, that the smallest distance the ship can travel in one step is still at least 20m and the tiles all have a width and height of 10m. Also the ship's direction of movement is still constrained to eight directions on the top level. Overcoming those problems by using a finer-grained tiling in an area around the top level goal or propagating the exact target position to a subtask which is called in its proximity could however increase the performance significantly.
Again we tracked the performance of the subtask learners as it can be seen in figures 5.6 and 5.7. The latter shows a slightly higher maximal reward for each subtask than figure 5.4. That is a result of the states often being mapped to a point which is already closer to the respective subtarget.

In order to better understand how the top level policy works there is an additional plot in figure 5.8 which uses a color coding of the tiles to show using the learned policy which action is most probable to be performed in which state. The images shown there each depict the 20×20 tiling which is laid on top of the state space. The two colors in the picture on the right denote the subtask which is to be called when the state lies in the respective tile. Gray stands for the

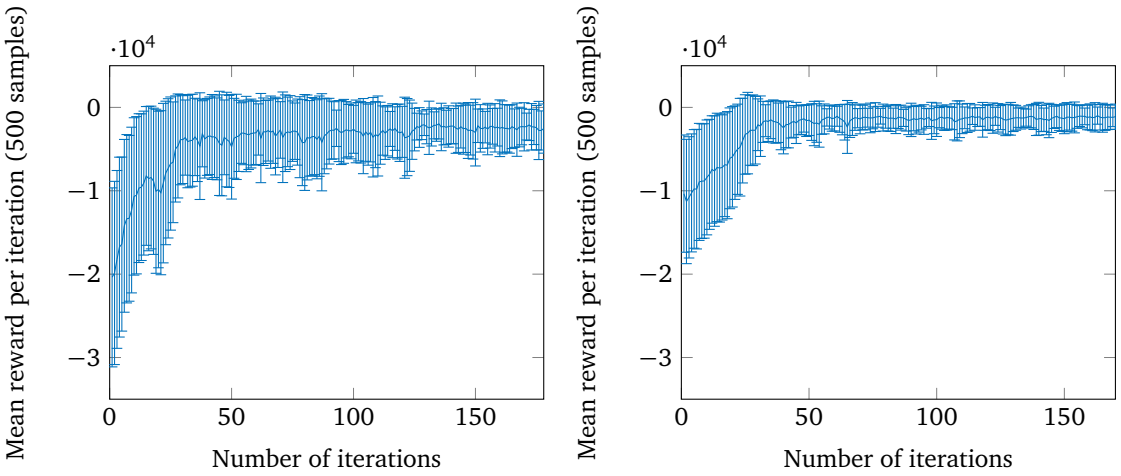


Figure 5.7.: The mean reward of the diagonal (left) and horizontal (right) subtask per 500 samples drawn each policy iteration. The bars show two times the standard deviation.

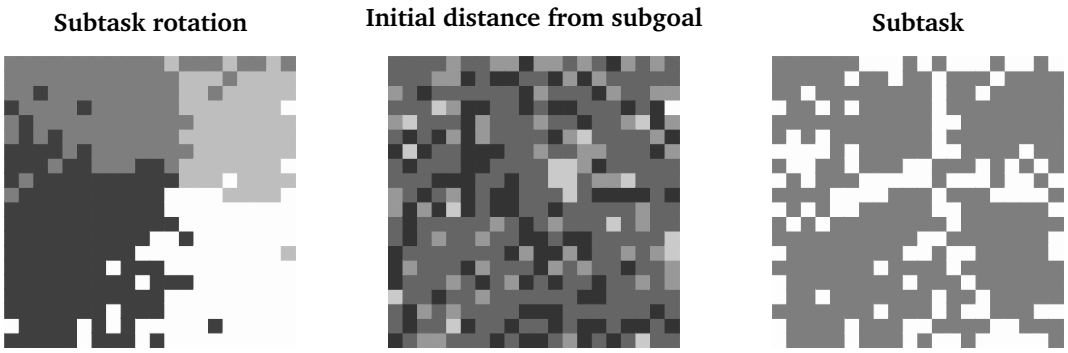


Figure 5.8.: The color coded policies learned by Q-Learning in the third experiment.

diagonal and white for the horizontal subtask. On the left there is the rotation of the chosen subtask. Going from dark to bright they depict rotations by 0, 90, 180 and 270 degrees clockwise respectively. The plot in the middle shows the distance from the respective subtarget the substate is initialized with. Black is the original distance and the brighter it gets the nearer it starts to its goal.

Finally a sample trajectory drawn using the learned policies can be seen in figure 5.9.

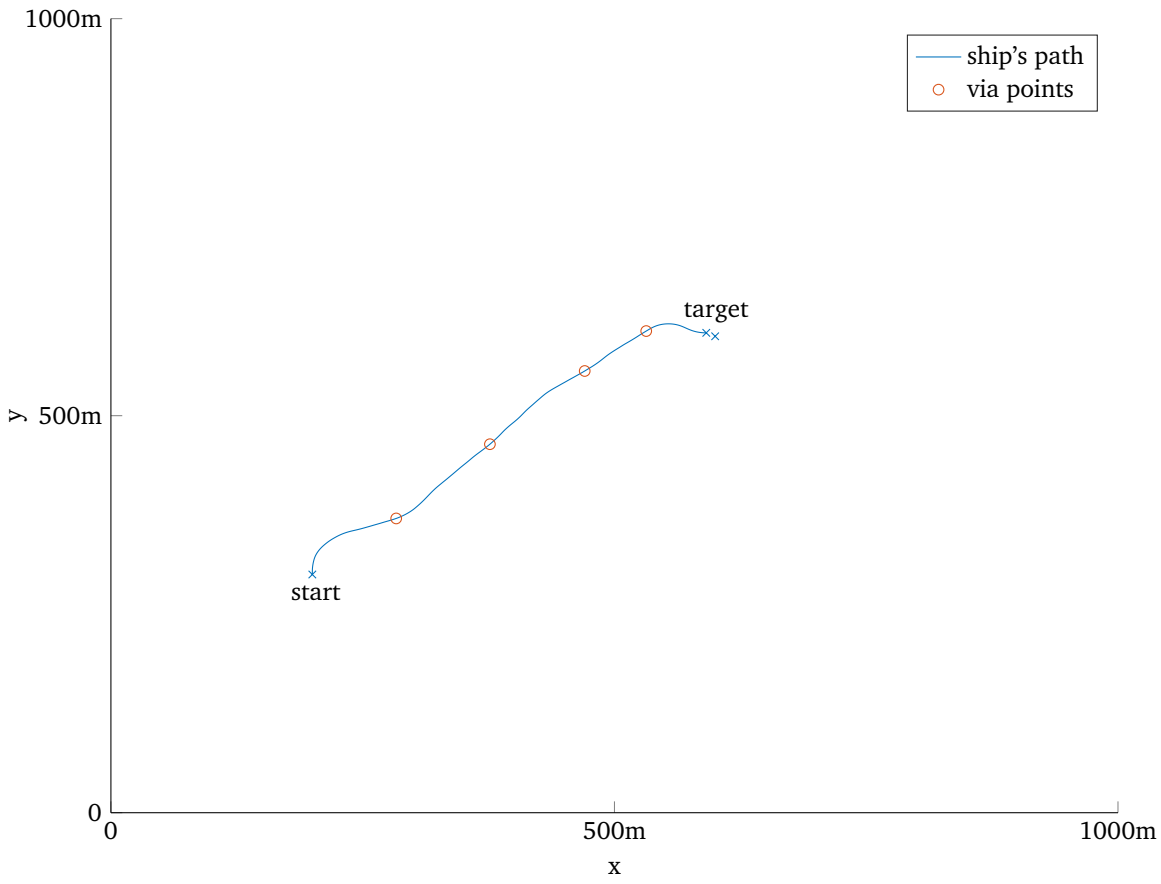


Figure 5.9.: A trajectory sampled from the policy learned in the third experiment.

6 Outlook

In this thesis, the foundations for the application of a hierarchical hybrid algorithm have been expounded. Then it was used in order to solve a non-trivial learning problem and the results have been compared to those of a traditional flat approach. Finally we modified the problem in order to reassess whether the hierarchical algorithm was able to adapt to the changed requirements.

Judging by the examples examined in this thesis hierarchical methods can improve some reinforcement learning problems radically as it was the case with the original ship steering problem. However not all problems seem suitable for a hierarchical decomposition. That became clear when we observed drawbacks as soon as we slightly modified the given task. Also not all problems provide us with such an apparent and convenient way of decomposing its transition system. So it becomes obvious that a hierarchical learning algorithm would greatly benefit from an automated way of achieving that.

As already hinted at in the introduction, hierarchical methods have, by now, found a lot more echo in the field of policy gradient reinforcement learning. Some more modern approaches [5, 6], for example, use latent variable models in order to automate the search for subtasks. A similar treatment could be applied to the ship steering problem in order to find a greater amount of suitable ones.

Another simple way of improving both of the presented solutions would be to use learning algorithms which require less samples to be observed in order to compute a meaningful update. After all the learners on higher levels of the hierarchy can only take advantage of their subtasks after those are already able to solve their own problem at least in some degree. That circumstance can be seen quite clearly when examining the performance of the last experiment which starts improving after a delay of 4000 episodes.

It can be concluded that hierarchical methods provide a meaningful improvement to reinforcement learning and should be taken into consideration at least as soon as a task comes with an exploitable hierarchical structure.

Bibliography

[1] M. Ghavamzadeh and S. Mahadevan, "Hierarchical policy gradient algorithms," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)* (T. Fawcett and N. Mishra, eds.), pp. 226–233, 2003.

[2] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[3] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning.," in *ICML*, pp. 118–126, 1998.

[4] R. E. Parr, "Hierarchical control and learning for markov decision processes," 1998.

[5] O. Kroemer, C. Daniel, G. Neumann, H. van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2015.

[6] C. Daniel, G. Neumann, and J. R. Peters, "Hierarchical relative entropy policy search," in *International Conference on Artificial Intelligence and Statistics*, pp. 273–281, 2012.

[7] W. Miller, P. Werbos, and R. Sutton, *Neural Networks for Control*. A Bradford book, MIT Press, 1995.

[8] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, pp. 181–211, Aug. 1999.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[10] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[11] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," pp. 388–403, 2013.

[12] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[13] M. Taylor, *Transfer in Reinforcement Learning Domains*. Studies in Computational Intelligence, Springer, 2009.

[14] A. Bredenfeld, *RoboCup 2005: Robot Soccer World Cup IX*. LNCS sublibrary: Artificial intelligence, Springer, 2006.

A Derivation of the recursive value function

In this section we are going to present the steps which lead from the definition of the value function in equation 2.5 to its recursive form shown in equation 2.6.
We denote a trajectory which has been sampled using μ, π and P as an infinite sequence

$$\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \mathbf{s}_3, \mathbf{a}_3, \ldots). \tag{A.1}$$

Also we denote trajectories which begin n states or actions ahead as $\tau^{[n]}$. A few examples can be seen in equations A.2 and A.3. According to that notation τ could also be written as $\tau^{[0]}$.

$$\tau^{[1]} = (\mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \mathbf{s}_3, \mathbf{a}_3, \ldots) \tag{A.2}$$

$$\tau^{[2]} = (\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \mathbf{s}_3, \mathbf{a}_3, \ldots) \tag{A.3}$$

We call the set of all trajectories T and define a random Variable $R : T \to \mathbb{R}$ as

$$R\left(\tau^{[0]}\right) = \sum_{t=0}^{\infty} \gamma^t r\left(\mathbf{s}_t, \mathbf{a}_t\right). \tag{A.4}$$

Finally we define the probability of sampling a specific trajectory $p\left(\tau^{[0]}\right)$ as well as the conditional probabilities $p\left(\tau^{[1]} \middle| \mathbf{s}_0 = \mathbf{s}\right)$ and $p\left(\tau^{[2]} \middle| \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right)$ of sampling the remaining trajectory given we already know the first state, or the first state and the following action, respectively:

$$p\left(\tau^{[0]}\right) = \mu\left(\mathbf{s}_0\right) \prod_{t=0}^{\infty} \pi\left(\mathbf{a}_t \middle| \mathbf{s}_t\right) P_{\mathbf{s}_t}\left(\mathbf{s}_{t+1} \middle| \mathbf{a}_t\right), \tag{A.5}$$

$$p\left(\tau^{[1]} \middle| \mathbf{s}_0 = \mathbf{s}_0\right) = \prod_{t=0}^{\infty} \pi\left(\mathbf{a}_t \middle| \mathbf{s}_t\right) P_{\mathbf{s}_t}\left(\mathbf{s}_{t+1} \middle| \mathbf{a}_t\right), \tag{A.6}$$

$$p\left(\tau^{[2]} \middle| \mathbf{s}_0 = \mathbf{s}_0, \mathbf{a}_0 = \mathbf{a}_0\right) = \prod_{t=0}^{\infty} P_{\mathbf{s}_t}\left(\mathbf{s}_{t+1} \middle| \mathbf{a}_t\right) \pi\left(\mathbf{a}_{t+1} \middle| \mathbf{s}_{t+1}\right). \tag{A.7}$$

Furthermore we are going to need the following equation

$$p\left(\tau^{[3]} \middle| \mathbf{s}_0 = \mathbf{s}_0, \mathbf{a}_0 = \mathbf{a}_0, \mathbf{s}_1 = \mathbf{s}_1\right) = p\left(\tau^{[3]} \middle| \mathbf{s}_1 = \mathbf{s}_1\right), \tag{A.8}$$

which holds since observing some further part of a trajectory is conditionally independent from its previous states and actions given a known state in between those two parts.
Then we can formulate the value function as

$$V^{\pi}(\mathbf{s}) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty} \gamma^t r\left(\mathbf{s}_t, \mathbf{a}_t\right) \middle| \mathbf{s}_0 = \mathbf{s}\right] = \mathbb{E}_{\pi,P}\left[R\left(\tau^{[0]}\right) \middle| \mathbf{s}_0 = \mathbf{s}\right] = \int_T p\left(\tau^{[1]} \middle| \mathbf{s}_0 = \mathbf{s}\right) R\left(\tau^{[0]}\right) \mathrm{d}\tau^{[1]}. \tag{A.9}$$

Finally following steps can be performed in order to complete the proof

$$
\begin{aligned}
V^{\pi}(\mathbf{s}) &= \int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) R\left(\boldsymbol{\tau}^{[0]}\right) \mathrm{d}\boldsymbol{\tau}^{[1]} \\
&= \int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) \left(\sum_{t=0}^{\infty} \gamma^t r\left(\mathbf{s}_t,\mathbf{a}_t\right)\right) \mathrm{d}\boldsymbol{\tau}^{[1]} \\
&= \int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) \left(\gamma^0 r\left(\mathbf{s}_0,\mathbf{a}_0\right) + \sum_{t=1}^{\infty} \gamma^t r\left(\mathbf{s}_t,\mathbf{a}_t\right)\right) \mathrm{d}\boldsymbol{\tau}^{[1]} \\
&= \int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) \left(r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma \sum_{t=0}^{\infty} \gamma^t r\left(\mathbf{s}_{t+1},\mathbf{a}_{t+1}\right)\right) \mathrm{d}\boldsymbol{\tau}^{[1]} \\
&= \int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) \left(r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma R\left(\boldsymbol{\tau}^{[2]}\right)\right) \mathrm{d}\boldsymbol{\tau}^{[1]} \\
&= \int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) R\left(\boldsymbol{\tau}^{[2]}\right) \mathrm{d}\boldsymbol{\tau}^{[1]} \\
&= \int_{\mathbf{a}_s} \left(\int_T p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) r\left(s,\mathbf{a}_0\right) \mathrm{d}\boldsymbol{\tau}^{[2]}\right) + \left(\int_S \int_T \gamma p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) R\left(\boldsymbol{\tau}^{[2]}\right) \mathrm{d}\boldsymbol{\tau}^{[3]} \mathrm{d}\mathbf{s}_1\right) \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \left(\int_T \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) p\left(\boldsymbol{\tau}^{[2]}\middle|\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a}_0\right) r\left(\mathbf{s},\mathbf{a}_0\right) \mathrm{d}\boldsymbol{\tau}^{[2]}\right) + \left(\int_S \int_T \gamma p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) R\left(\boldsymbol{\tau}^{[2]}\right) \mathrm{d}\boldsymbol{\tau}^{[3]} \mathrm{d}\mathbf{s}_1\right) \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) r\left(\mathbf{s},\mathbf{a}_0\right) \left(\int_T p\left(\boldsymbol{\tau}^{[2]}\middle|\mathbf{s}_0=\mathbf{s},\mathbf{a}_0=\mathbf{a}_0\right) \mathrm{d}\boldsymbol{\tau}^{[2]}\right) + \left(\int_S \int_T \gamma p\left(\boldsymbol{\tau}^{[1]}\middle|\mathbf{s}_0=\mathbf{s}\right) R\left(\boldsymbol{\tau}^{[2]}\right) \mathrm{d}\boldsymbol{\tau}^{[3]} \mathrm{d}\mathbf{s}_1\right) \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) r\left(\mathbf{s},\mathbf{a}_0\right) + \left(\int_S \int_T \gamma \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) P_{\mathbf{s}}\left(\mathbf{s}_1\middle|\mathbf{a}_0\right) p\left(\boldsymbol{\tau}^{[3]}\middle|\mathbf{s}_0=s,\mathbf{a}_0=\mathbf{a}_0,\mathbf{s}_1=\mathbf{s}_1\right) R\left(\boldsymbol{\tau}^{[2]}\right) \mathrm{d}\boldsymbol{\tau}^{[3]} \mathrm{d}\mathbf{s}_1\right) \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma \pi\left(\mathbf{a}_0\middle|s\right) \left(\int_S P_{\mathbf{s}}\left(\mathbf{s}_1\middle|\mathbf{a}_0\right) \int_T p\left(\boldsymbol{\tau}^{[3]}\middle|\mathbf{s}_1=\mathbf{s}_1\right) R\left(\boldsymbol{\tau}^{[2]}\right) \mathrm{d}\boldsymbol{\tau}^{[3]} \mathrm{d}\mathbf{s}_1\right) \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) \left(\int_S P_{\mathbf{s}}\left(\mathbf{s}_1\middle|\mathbf{a}_0\right) V^{\pi}\left(\mathbf{s}_1\right) \mathrm{d}\mathbf{s}_1\right) \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) \mathbb{E}_P\left[V^{\pi}\left(\mathbf{s}'\right)\middle|\mathbf{s},\mathbf{a}_0\right] \mathrm{d}\mathbf{a}_0 \\
&= \int_{\mathbf{a}_s} \pi\left(\mathbf{a}_0\middle|\mathbf{s}\right) \left(r\left(\mathbf{s},\mathbf{a}_0\right) + \gamma \mathbb{E}_P\left[V^{\pi}\left(\mathbf{s}'\right)\middle|\mathbf{s},\mathbf{a}_0\right]\right) \mathrm{d}\mathbf{a}_0 \\
&= \mathbb{E}_{\pi}\left[r\left(\mathbf{s},\mathbf{a}\right) + \gamma \mathbb{E}_P\left[V^{\pi}\left(\mathbf{s}'\right)\right]\middle|\mathbf{s}\right].
\end{aligned}
$$

The action $\mathbf{a}$ seen in the last term is sampled using π and the resulting state $\mathbf{s}'$ is drawn from $P_{\mathbf{s}}$.
This equation can also be proven for discrete sets of states and actions completely analogously by exchanging the integrals with sums.