Hierarchical Reinforcement Learning in Robot Control

Florian Schnell
Student
Technische Universitaet Darmstadt
64289 Darmstadt
yg63sixe@rbg.informatik.tu-darmstadt.de

Abstract

The use of reinforcement learning in robotics suffers greatly from the curse of dimensionality, stemming from the necessity of a high DOF in a robot to solve even the simplest tasks. This paper gives an overview over several attempts to decrease this problem using a hierarchical structure of reinforcement learning, an evaluation of the problems that arise with such approaches and possible solutions.

1 Introduction

The field of reinforcement learning stems from behavioral studies of animals and provides us with a highly independent form of learning. Unlike optimal control methods, reinforcement learning does not rely on supervision or training examples, which makes it very suitable for the use in robotics. Further details on the history of reinforcement learning and it's differences and connections to other forms of learning are documented in [1].

1.1 What is reinforcement learning?

Reinforcement learning (RL) is an area of machine learning, concerned with how an *agent* ought to take *actions* in an *environment* so as to maximize some notion of cumulative *reward* [2]. The agent remembers it's actions and outcomes from previous trials and uses that knowledge (exploitation), but sometimes chooses a random action to see it's outcome (exploration). Slowly, the agent will learn how to maximize his reward and due to exploitation phases does not simply strive for the quickest way to a reward, but the highest cumulative reward.
One famous example in behavioral studies is that of a mouse in a maze. We have a fixed maze and a mouse gets placed inside that maze. Each time, a small amount of cheese is planted at the same position inside the maze. The mouse will explore the maze and find the cheese, ending the experiment. In subsequent trials, the mouse knows which path it took and what reward is at it's end, yet sometimes tries different paths to find a shorter route. After several trials it has learned the shortest path from starting point to the cheese. In this case, the mouse is our agent, it's movements the actions, the maze the environment and the reward is the cheese. The first few trials with the mouse not knowing the shortest path are the exploration phase, the ones following the shortest path the exploitation phase. In order to verify that the mouse does not simply smell the cheese, there is a small variation to the experiment. After the mouse learned the path, another repetition is done without cheese in the maze. The mouse will still take the learned route, but after finding nothing, will move on to search the rest of the maze (exploration).
The field of RL encompasses several algorithms and strategies that all follow the common rule of the agent taking actions to gain a reward. Popular examples of Algorithms are Temporal Difference Learning, Q-Learning and Monte Carlo Methods. Further detail concerning RL can be found in [1].

1.2 What is the motivation for hierarchical reinforcement learning?

While RL is a sound learning strategy, it suffers greatly from the curse of dimensionality [1]. This means, the required computation time increases supra-linear to the complexity of the problem. A real-life application in robotics usually uses a robot with several joints resulting in a high Degree-of-Freedom (DOF). Even the simplest models need to represent the position or alignment of those joints, quickly resulting in a 40 or more dimensional space. Using RL in such a high dimensional space is computational expensive, slowing the learning time down or even making it not feasible. Even with our ever increasing computational abilities, it would take a long time to make the application of reinforcement learning in real life robotics beyond basic tasks a viable option. Since the goal is to find a way of independent learning in the field of robotics, this leads to the necessity to find either another approach or decrease the effect of the curse of dimensionality.

1.3 What is hierarchical reinforcement learning?

Hierarchical reinforcement learning (HRL) focuses on the latter by dividing the task and the learners into smaller parts and creating an hierarchical structure. The hierarchical structure consists of two or more layers in which RL is applied. The layers can be divided into two subgroups, control and action layers. The control layers try to solve their given goal by issuing sub-goals to the layer below - controlling it. Each additional control layer represents a further abstraction of the task. The amount of control layers is only capped by how often the original task can be divided into subtasks. The example given in the MAXQ [3] framework uses three to four different layers of abstraction, while the case studies later on evaluated in this thesis only use one. The action layer tries to solve the given goal by issuing commands to their actuators. It works in the same space as an RL application of the problem. The action layer is always the lowest layer in each branch of the hierarchy. On different branches, different action layers can exist. In general, it can be seen as one small part of a standard RL approach, learning only the best actions from it's start to subgoal as opposed from begging to the end of the task. By issuing different subgoals to the action layer the whole task is completed.

The improvements over traditional RL can come from different facts stem from the fact, that the learning is not done on the complete task, but sequences of sub-tasks. This reduced search depth can result in several improvements. One possible improvement is, if the task can be divided into sub-tasks that can be performed independently of each other. Using a low level learner to learn each of the sub-tasks and a high level learner to learn the order of the sub-tasks can decrease the learning time. While a traditional RL approach would see each different order of sub-tasks as something different, the HRL approach still uses any knowledge gained in the sub-tasks for subsequent trials, no matter which order is used. If I want to make breakfast, it has no effect towards learning of each sub-task, if I first brew coffee or prepare food. Another possible improvement is reducing the complexity of the task. If the complete task can be divided into a sequence of sub-tasks, failure in any of those does not affect previous sub-tasks. If I have to finish the complete task of dressing, I still gain knowledge on how to put on socks, even if I later on fail to put on my trousers. A third possible improvement comes from re-usability of the gained knowledge. If I want to expand my breakfast into having some orange juice, only the high level learned would need to adapt and a few trials are necessary. Similar, if we remove a sub-task, only the order needs to be learned, since knowledge of how to perform each sub-task still exists. Re-usability does not only come in handy if the task is change, but also if a sub-task is needed to be done several times. If I need to clean three exact identical rooms, I can use any knowledge gained from cleaning the first when i clean the second room. Which kind of improvements, or even if any improvements at all are gained when applying HRL is dependent on the task and which kind of HRL is used. While HRL can bring improvements, it has it's downsides as well. Compared to plain RL, a HRL implementation is more complex to set up. This increased complexity does not only mean, that more time is needed, but sometimes also more knowledge of the task in order to decompose it. The implications of this are discussed in the case studies as well as in the conclusion at the end of this thesis.

2 Case studies of HRL

The case studies are presented in the first two subsections, with an evaluation of both following in the last subsection. The setup of the case studies will only briefly described, focusing on which algorithms and methods were used, why they where used, but not how exactly they were used. This is due to the fact, that a full description would go beyond the scope of this overview.
All figures used in the case-studies are taken from their respective papers, as are all formulae.

2.1 Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning

In this paper *Jun Morimoto & Kenji Doya* propose the application of hierarchical reinforcement learning for the acquisition of stand up behavior by a robot. The learning is done in two phases, first a simulation phase to acquire the theoretical skills and then a practical phase using a real robot.

Setup. A three link, two joint robot (Fig. 2.1.1) with no connection to the ground should stand up from a lying position and keep an upright posture for a set time. The robots only actions are controlling the actuators in it's joints. It has to use it's own body as a lever as well as it's momentum to realize the task. To faciliate this, HRL is used with a two layer strategy, one control and one action layer.

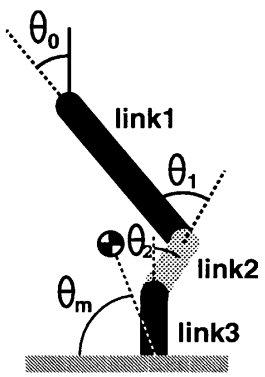

Figure 2.1.1: Robot configuration. θ_0: pitch angle, θ_1: hip joint angle, θ_2: knee joint angle, θ_m: the angle of the line from the center of mass to the center of the foot. *Picture and description are taken from the original paper* [2.1]

The *control layer* works in a low dimensional, abstract state space and sets the sub-goals for the action layer. Each state represents one of the sub-goals. The learner uses the Q(λ)-learning method to learn the sub-goal sequence to finish the task. As representation of the state variables the posture of the robot (Fig. 2.1.1) $X = (\theta_m, \theta_1, \theta_2)$ was chosen. The reward $R(T)$ is defined as follows.

$$R(T) = R_{main} + R_{sub} \tag{1}$$

$$R_{main} = \begin{cases} 1, & \text{on successful stand-up} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$R_{sub} = \begin{cases} 1, & \text{final goal achieved} \\ 0.25\left(\frac{Y}{L}+1\right), & \text{sub-goal achieved} \\ 0, & \text{on failure} \end{cases} \tag{3}$$

With Y representing the height of the robot and L the total length of the robot. The final goal is an upright posture, meaning the height is equal to the length. Reaching the goal is necessary, but not sufficient, since the robot may still fall down if he has a too high momentum. Each time the robot reaches the neighborhood (detailed in low level learner) of a sub-goal, a new one is chosen, the action-value function in the upper learner is updated and the reward given. The task was considered successful, if the robot stood upright for at least $2(T+1)$ seconds. If the robot fell down and hit his head or hip or a time limit expired, the task was considered a failure.

The *action layer* works directly in the high-dimensional state space without discretization. Pitch and joint angles $\theta = (\theta_0, \theta_1, \theta_2)$ (Fig 2.1.1) and the corresponding angular velocities $\dot{\theta} = (\dot{\theta_0}, \dot{\theta_1}, \dot{\theta_2})$ were used as the state variables $x(t) = (\theta, \dot{\theta})$. Due to this dimensional difference from the high level learner to the low level learner, any sub-goal is a hyperplane for the low level learner. The action layer uses continuous $TD(\lambda)$-learning with the actor-critic method specified by INGnet. Similar to the top level, the reward is divided into two parts. The first is given during control according to the distance from the current posture θ to the sub-goal posture $\hat{\theta}$. $r(\theta, \hat{\theta}) = \exp(- \parallel \theta - \hat{\theta} \parallel / s_\theta) - 1$ where $s_\theta = 30$ [deg] gives the width of the reward function.

$$r(t) = \begin{cases} \exp\left(-\frac{\| \dot{\theta} - \hat{\dot{\theta}} \|}{s_{\dot{\theta}}}\right), & \text{if sub-goal achieved} \\ -1.5, & \text{if fall down} \end{cases} \quad (4)$$

The second reward is given at the end of the control depending on the distance from the current pitch and joint angular velocity $\dot{\theta}$ to the desired values $\hat{\dot{\theta}}$. This reward is only possible to gain after the first successful stand-up, during which $\hat{\dot{\theta}}$ is initially set. $\hat{\dot{\theta}}$ is updated in each successful trial $\hat{\dot{\theta}} \leftarrow \eta\hat{\dot{\theta}} + (1-\eta)\dot{\theta}$ with $\eta = 0.9$. The given reward r(t) is defined as in equation (4) where $s_{\dot{\theta}} = 60$ [deg] giving the width of the reward function. If the time limit is reached prior to the end of control, no reward is given.

Results. First the stand-up task was simulated. The states for the high level learner was set to $X = (\theta_m, \theta_1, \theta_2)$, where $0 \leqslant \theta_m \leqslant 90$, $-150 \leqslant \theta_1 \leqslant 0$, $0 \leqslant \theta_2 \leqslant 25$ [deg] and the action step as $\Delta\theta_m = 30$, $\Delta\theta_1 = 50$, $\Delta\theta_2 = 25$ [deg]. The lower level has a higher freedom with $-150 \leqslant \theta_1 \leqslant 150$, $-150 \leqslant \theta_2 \leqslant 150$ [deg]. Each trial was started with the robot lying on the ground, $x = (90, 0, 0, 0, 0, 0)$ [deg] and continued for $t < 2(T+1)$ seconds in simulated time, where T is the discrete time in the upper level. If the robot fell down and hit its hip or head on the ground, the trial was counted as a failure and restarted. Each simulation was continued up to 1000 trials. Furthermore a noise term was added to the simulation. Figure 2.1.2 shows one successful run of the simulation.

As measure of the learning speed, the amount of trials before achieving 10 successful trials was used. With this configuration the robot needed on average 749 trials, which took 30 minutes simulated time, stood up successfully in 7 out of 10 runs and used 4.3 sub-goals for a successful stand up (all averaged over 7 successful runs). During the learning the robots knowledge was observed. After about 300 trials the upper learner knew the first two appropriate sub-goals and the lower level learner achieved them, while after 750 trials the complete goals were known and could be achieved by the lower learner.

To investigate the action step size, a *second simulation* was started with θ_2 fixed to 0 [deg] by servo control. The as state variables the old ones, pruned of θ_2 and $\dot{\theta}_2$ were used. As different step sizes for $\Delta\theta_1 = 25, 30, 50$ [deg] and $\Delta\theta_m = 30$ [deg] were used. Each simulation was continued up to 1000 trials. The step sizes 25 and 30 received similar success rates of 90% respective 100% after 408 respective 375 trials, meaning the via points for a successful stand up were in their vicinity. The step size 50 on the other hand seemed to lack this via point and had only a 20% success rate after 463 trials. A *third simulation* with plain RL was used as a comparison. For this, $TD(\lambda)$-learning was used. After 1685 trials, which took 65 minutes the robot achieved 10 successful stand-ups. The robot succeeded in standing up in 5 out of 10 runs. A *fourth set of simulations* was started with the $\theta_2 = 0$ [deg] setting. In this set, the upper learner did not receive the R_{sub} reward. After 1000 trials, the robot never learned to stand up. To verify the effectiveness, the results of the original first set of the simulation were used in a real robot (Fig 2.1.3). After an additional 164 trials, the robot learned to stand up in 6 out of 7 experiments (all averaged over 6 successful runs).

Conclusion. Comparing the first set of the simulations (original HRL approach) to the third set (plain RL approach) clearly showed the usefulness of the HRL. The robot learned the task in about a fourth of the trials and in a more robust way. The second set of simulations showed, that the discretization affects the results, but is not a strict point but a certain range of action steps that allow the upper learner to successfully acquire appropriate sub-goals. The fourth set of simulations showed, that the upper level learner needs to gain a reward for setting achievable sub-goals. Otherwise it chooses sub-goals, that are unfulfillable for the low level learner and result in near-constant failure.

2.2 Hierarchical Reinforcement Learning with Movement Primitives

In this paper, *Freek Stulp & Stefan Schaal* propose a decomposition of tasks into Dynamic Movement Primitives, whose sequences are then learned by their HRL algorithm PI. This approach is implemented on an 11-DOF arm and hand concerning a pick-and-place task.

Setup. In this paper four different learning methods are compared in three different experiments. The first experiment is a theoretical example to emphasize the differences between the learning

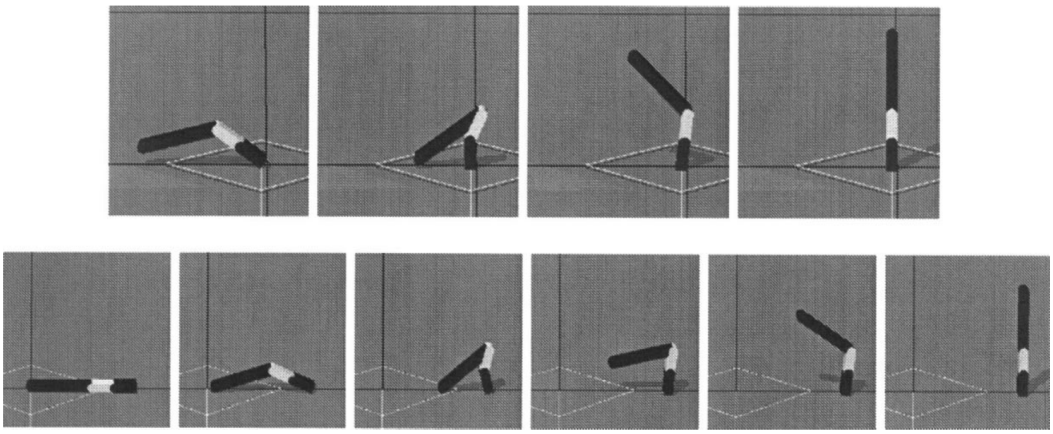

Figure 2.1.2: Example of a successful sub-goal sequence (top); example of a successful stand-up trajectory (bottom). *Picture and description are taken from the original paper* [2.1]

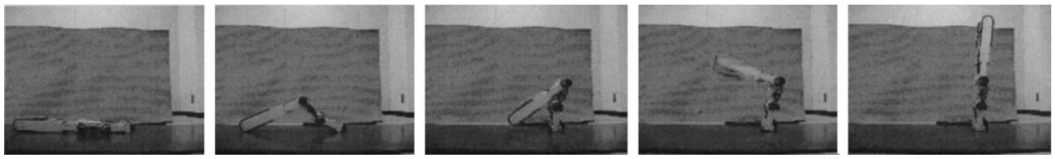

Figure 2.1.3: Example of a stand-up trajectory using the real robot. *Picture is taken from the original paper* [2.1]

methods, while the second and third are real-life robot applications of grasping and moving an object. Each of the experiments is composed of a sequence of two Dynamic Movement Primitives (DMP).

A DMP is a set of dynamic system equations (Fig 2.2.1). They describe a motion from the starting point x_0 with shape θ towards the goal g. Changing any of the three parameters x_0, θ or g results in a different motion, thus making it a flexible representation. DMPs can be chained together by using the goal as the starting point for the next, resulting in sequences of DMPs to model more complex motions. A DMP as shown in Fig 2.2.1 only specifies a 1-dimensional motion. In order to gain multi-dimensional movement, several sets of system equations are used, which share the same phase variable. Each of those sets describes the motion in one dimension with it's own goal and shape parameters.

$$\frac{1}{\tau}\ddot{x}_t = \alpha(\beta(g - \dot{x}_t) - \dot{x}_t) + g_t^T\theta \qquad \text{Transform. system} \qquad (5)$$

$$[g_t]_j = \frac{w_j(s_t) \times s_t}{\sum_{k=1}^{p} w_k(s_t)}(g - x_0) \qquad \text{Basis functions} \qquad (6)$$

$$w_j = \exp(-0.5h_j(s_t - c_j)) \qquad \text{Gaussian kernel} \qquad (7)$$

$$\frac{1}{\tau}\dot{s}_t = -\alpha s_t \qquad \text{Canonical system} \qquad (8)$$

Figure 2.2.1: The core idea behind DMPs is to perturb a simple linear dynamical system (the first part of Eq. 1) with a non-linear component ($g_t\theta$) to acquire smooth movements of arbitrary shape. The non-linear component consists of basis functions g_t , multiplied with a parameter vector θ. *Picture and Description taken from the original paper* [2.2]

The four different learning methods all use the PI (Path Integrals) algorithm. In each experiment, the shape of the motion θ is first learned via imitation learning. Each learning method tries to improve this initial motion, reducing the total cost of the movement described by the cost function.The four learning methods and the original motion are:

I Original motion

II Only shape learning

III Only goal learning

IV Shape and goal learning with a greedy approach

V Shape and goal learning with a HRL approach

They are further detailed in the first experiment, which is set up to highlight their differences.

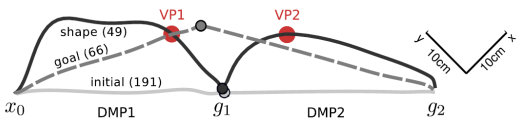

Figure 2.2.2: Trajectories for the only-goal and only-shape approach. The detours done by the shape learning are clearly visible. *Picture is taken from the original paper* [2.2]

First experiment: — The DMPs are initialized to generate straight motions with a bell-shaped velocity profile, going through the first goal point. Two via-points are added to the motion, with one being in each DMP. These points do not lie on the original motion. The robots task is to optimize this motion to pass through both via points whilst minimizing the velocity of the end-effector.

I is the original motion learned by imitation learning. It is not optimzed towards the cost function, resulting in a high distance error to the via points, but uses a rather low velocity. The cost is 191. Fig 2.2.2 and 2.2.3

II keeps the original goals from the initial motion. The robot learns to move through both via-points, decreasing the cost by reducing the distance error. Reaching both via-points and the goal point separating the DMP needs a longer path, which results in a higher velocity. The total cost decreases to 49. Figure 2.2.2

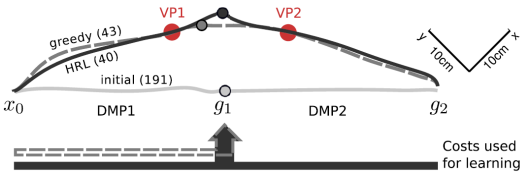

Figure 2.2.3: Trajectories for the greedy and HRL approach. The lower bar shows which costs are used during the learning of the first goal, with greedy not taking the the cost of the second DMP into account. *Picture is taken from the original paper* [2.2]

III changes the first goal, but does not change the shape. Since the motion always moves through the starting point, the first and the final goal according to the shape, this results in a different motion. This new goal results in a slightly longer path, resulting in an increased velocity. Additionally the cost is increased to the distance error to the via points, resulting in a total cost of 66. Figure 2.2.2

IV changes both shape and goal greedily concerning each DMP. The resulting motion goes through both via-points. Since it optimizes towards each DMP greedily, a longer path is needed to reach the second DMP, resulting in a higher velocity penalty. The overall costs are reduced to 43. Figure 2.2.3

V changes both shape and goal parameters, but on different levels. It uses a more complex approach than the others. Unlike them, HRL is used with two layers of learners. The upper learners explores the space of each goal-parameter g and tries to minimize the cost regarding the current and all following DMPs. This is due to the fact, that each goal influences the current and subsequent DMPs, as seen from the results of *IV*. The lower learner explores the space of the shape parameter θ with the current goal parameter set by the top level. The lower level is similar to the approach of *II*. The total cost of this approach is 40. Figure 2.2.3

Second experiment: — This task was chosen to mirror an experiment from the field of psychology in which humans are asked to grasp a bottle from a shelf and place it on a higher or lower shelf. The way a person grasps the bottle in the first place is influenced by the second motion. If tasked to place it on a higher shelf, the bottle is grasped significantly lower and vice versa. This makes the second motion of placing the bottle easier to perform.
In the same way as the humans, a robot is placed in front of a cupboard. It's task is to grasp a cylindrical tube in the center of the lower shelf and place it on the upper shelf. To solve the task a sequence of two 7-DOF motion primitives are used. The motion primitives represent the position of the end-effector (3-DOF) and the posture of the hand (4-DOF). During the whole task the orientation of the hand is fixed towards the shelves (Fig 2.2.4).

The first motion reaches from the initial rest position x_0 to the object position g_1 and grasps it. The second motion transports the object from g_1 to a position on the upper shelf g_2. The shape parameters are again trained through imitation learning. This task is more complex than the initial experiment due to interaction with the environment, which requires a collision avoidance controller. This controller triggers if the motion vector would bring it too close to the shelf and starts an acceleration away from the shelf. The cost of the motion consists of four components:

- The first component $G(t)$ concerns the grasping and transporting of the object. If the gripping is successful, the value is 0, otherwise 1 (a high penalty for failing to grasp the object or dropping it).
- The second component is the obstacle avoidance module. It's acceleration $\|\ddot{x_{avoid}}\|$ is added to the cost during each time step. If the module is not triggered this results in no additional cost.
- The third component $\|\ddot{x}\|/250$ is the overall acceleration.
- The fourth component is the distance of the object to the shelve when it gets released.

The first three components are immediate costs and are divided by the number of time steps N, to make the cost independent from the duration of the movement. The fourth cost is only added once. This results in the following cost function:

$$J(\tau_i) = \phi_{t_N} + \frac{1}{N}\int_{t_i}^{t_N} (4G(t) + \|\ddot{x}_{avoid}\| + \|\ddot{x}\|/250)dt \tag{9}$$

This experiment is first performed as a simulation. Afterwards the results are used in an implementation using a real robot to verify it's applicability.

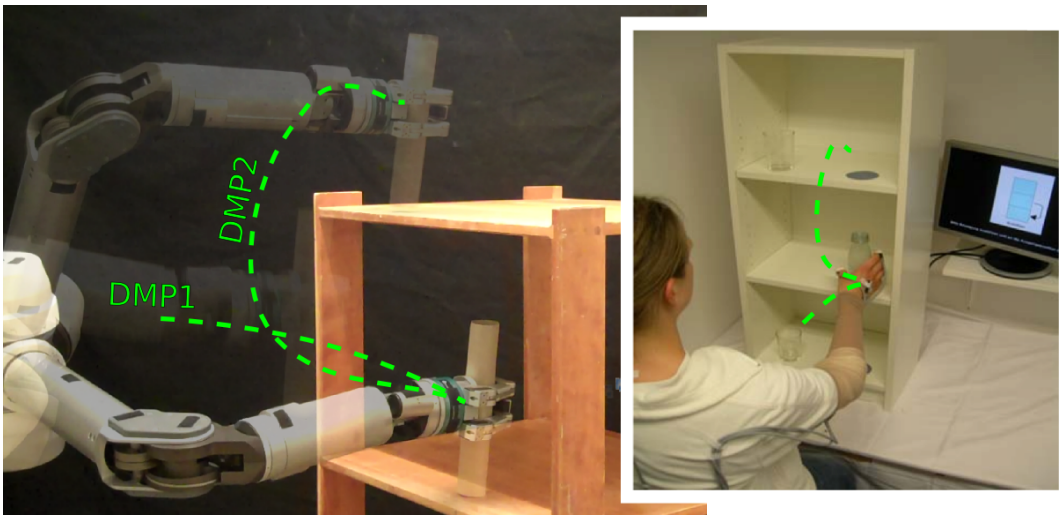

Figure 2.2.4: The pick and place task as performed by the robot and the original experiment which inspired the authors. *Picture is taken from the original paper* [2.2]

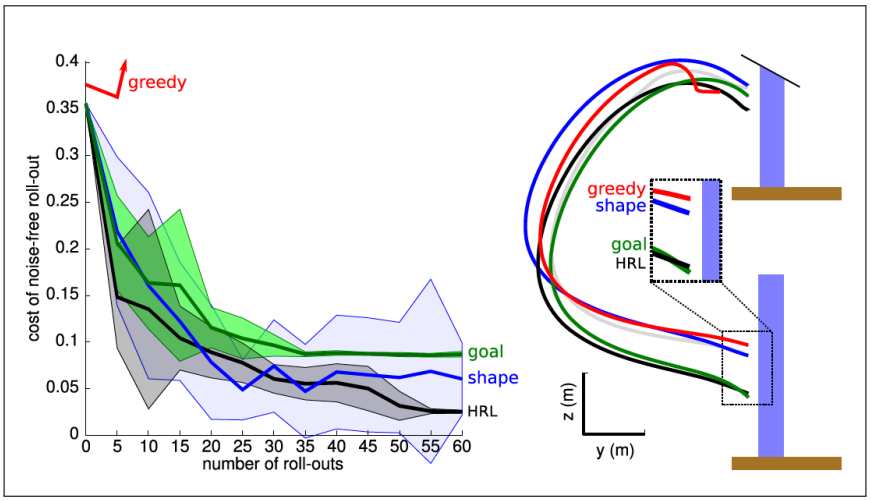

Figure 2.2.5: Left: Learning curves ($\mu \pm \sigma$ over five learning sessions) for each of the learning strategies. Note that the greedy strategy soon jumps to a cost of over 1.0, and does not achieve a lower value afterwards. Hence, only its first two values are plotted. Right: End-effector paths for the different strategies after 60 roll-outs, averaged over the five sessions. The initial trajectory before learning is light gray. For clarity, only the path generated by the second DMP is depicted. *Picture and description are taken from the original paper* [2.2]

Third experiment: — The experiment uses the setup from the second experiment with only a slight change - the object is placed initially on the top shelve and the robot is tasked with grasping and moving it to the lower shelve. Accordingly, the DMPs are trained by imitation learning with a human performing this task.

Results. The second experiments results (Fig 2.2.5) are in line with the observations of the first experiment. The initial learning gives a non-perfect motion, meaning the object is grasped close to the top. The shape only approach (*II*) does not change the goal points of grasping and dropping the object, but changes the shape of the movement to minimize the activation of the collision avoidance module. The averaged result is close to 0.75. The goal only approach (*III*) grasps the bottle lower and drops the object from a lower height. The first part minimizes the necessary movement in the second DMP, while the second part reduces the terminal cost of dropping the object. Due to the fact, that the shape is not optimized towards these new goals the resulting motion is not optimal, resulting in an unnecessary high cost. The averaged cost is close to 0.1. The greedy approach (*IV*) optimizes for each DMP independently. This results in the object getting grasped high to minimize the movement in the first DMP, making the second motion more difficult. This greatly impacts the averaged overall cost, which is over 1.0 after 60 roll-outs. Finally the HRL approach (*V*) grasps the object lower and drops it lower to reduce the costs like seen in *III*. Additionally the shape is optimized to the new goals. This leads to an average cost of 0.025.
The third experiments results not detailed since they mirror the second experiment. The only difference is, that the height at which the object is grasped is reversed. *III* and *V* grab the bottle higher, while *IV* grabs it lower.

Conclusion. The first experiment showed, that there can be quite some differences in the task up to which strategy is learned. While the greedy approach (*IV*) gave good results on the first DMP, the second was suboptimal. In longer sequences this may create an increasing problem since the choice of each DMP's parameters effect all subsequent, yet the greedy approach does not care for this fact. In both experiment two and three the problems of each learning method resurfaced. The only-shape and the only-goal methods could not reach good results due to the fact, that both optimized only concerning one parameter. Resulting from this this at least one part of the cost function still lead to comparably high costs (between two and four times the cost of the HRL approach). The greedy method produced worse results than the original motion, due to the fact it learned each DMP separately. This led to very low costs in the first DMP, but an almost impossibly task in the

second DMP. Either the velocities needed to be high or the collision avoidance module was triggered, resulting in a very high overall cost (more than 40 times the cost of th HRL approach) and the worst of all results. The HRL approach on the other hand behaved similar to humans, as observed in the original psychological experiment. The object is grabbed low and dropped from a low height and the shape is adapted to use minimal acceleration while still not triggering the collision avoidance module.

2.3 Evaluation of the case studies

In both case studies we saw implementations of HRL using first simulations to learn the task and second a real robot to prove it's applicability. Both case studies can be seen as succesful in the way, that both implementations were able to solve their task using a real robot. Yet in both case studies several problems arise. First and most important is the prior knowledge problem. This problem comes from the fact, that plain RL approaches need next to no knowledge about the system. They need to know the goal and the possible agents, everything else gets explored. Yet in both case studies we saw additional knowledge being used. In the fist case study the authors showed, that the action step does not need to be a discrete value, but anything in a certain range proves to be viable. Still, the fact that the state space of the top learner only uses a subset of the dimension and of those even a part with quantification is a use of prior knowledge. Similar in the second second case study the DMPs were already identified before the experiment. Another problem in the second case study is the comparison between the different methods. The HRL approach is compared to a set of three, four if counting the initial imitation learning, methods. Since the experiment is mimicking the one done in experimental psychology, the authors most likely knew that optimization towards the complete sequence of DMPs yields the best results. Yet none of the used RL methods used this fact. Additionally a comparison to a RL approach that does not use two separate DMPs but a single, high dimensional motion would have further detailed not only the usefulness of HRL, but also DMPs. Since the authors stated that they would employ the PI algorithm using DMPs, this is not necessary and might not have fitted the scope of the research. Generalizing we can say that both implementations were successful, but used prior knowledge. This prior knowledge problem is currently unsolved, but several approaches exist. Some of them will be explained in short in the next Section.

3 Solving the prior knowledge problem

The prior knowledge problem can be summarized as the fact, that a HRL should not need more knowledge about the system than a RL approach. To solve this problem it would be necessary to gain a method to divide a RL implementation of a task into a HRL one, while still improving the performance. In order to facilitate such a transformation, several parts of the original RL structure need to be explored.

- The reward function needs to be split into at least two parts, rewarding achieved sub-goals in the lower learner and the task in the higher learner. Some approaches, like shown in Subsection 2.1 additionally need a reward for picking achievable sub-goals for the high level learner. Solving this in any way is trivial and can be done by giving the high level learner a high reward upon completion of the task and the low level learners a reward upon completion of the subtask. Finding a reward function that values sub-goals close to the goal more highly can result in less needed trials since the progress gets more streamlined, but finding such functions is far more complex.
- The dimensional space needs to be discretized for the high level learner. This task can be achieved by clustering states together that are close to each other by a certain measure. Several methods using unsupervised learning already exist like Self Organizing Maps ([4], [5]), different Vector Quantification and Clustering methods ([6], [7], [8]) or k-means-clustering ([9], [10]).
- The original policies utilized by RL need to be split or adapted into different ones fitting the high and low level learns.

In the following subsections, two approaches are shown. The first one utilizes clustering methods and a basic reward function to show a sub-goal and policy exploration method. The second ap-

proach focuses on finding better subgoals by not clustering them according to location, but relative frequency.

3.1 Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization

In this paper *Bakker, Bram & Jürgen Schmidhuber* present an algorithm targeted at solving the prior knowledge problem by automatically discovering subgoals and subpolicies with their algorithm, HASSLE (**H**ierarchical **A**ssignment of **S**ubgoals to **S**ubpolicies **LE**arning algorithm). Prior to the usage of the algorithm the the possible states of the system are quantized with any clustering algorithm. In their experiment ARAVQ ([11]), a unsupervised learning vector quantization technique was used. They used this clustering as the input for their high level learner and the unclustered for their low level learner. In the first phase of learning the algorithm explored it's possibilities, repeatedly assigning a sub-goal and a policy to the lower level learner. This in turn utilizes the policy until it either reaches the sub-goal or it's time runs out.The result is then used to update several things. If the policy reached the sub-goal, the triple (policy, starting point, end-point) was used to update it's C-value. C-values denote how useful a policy is in reaching an end point from a given starting point. If the policy did not reach the sub-goal, but another state of the high level learner, the C-value for the sub-goal was lowered, while the C-value for the reached state was increased. If it did not reach any new state, the C-value was only lowered. If the sub-goal was reached by the low level learner it updated it's policy according to standard RL methods to improve reaching it on subsequent trials. In the second phase of learning the algorithm tried to improve it's possibilities and to solve the task. It will now choose only sub-goals it deems useful for solving the task. Those sub- goals are to be solved by the policy having the highest c-value given current starting point and desired sub-goal. During subsequent trials the high level learner will improve it's chosen sub-goals, the C-values will better denote the most useful policy and the low level learner will improve it's ability to fulfill the given sub-task. In an experiment, the authors showed compared the result of their algorithm to four RL systems first in a discrete and then in a stochastic system. In both HASSLE vastly outperformed the other algorithms in effectiveness (time steps needed to reach the goal) over learning time (time steps needed to current knowledge).

3.2 Connect-based subgoal discovery for options in hierarchical reinforcement learning

Chen, Fei; Gao, Yang; Chen, Shifu; Ma, Zhenduo — In this paper the authors present a method to discover sub-goals to be used in a HRL implementation. They used the given policies combined with all different starting points of the implementation to gain the frequency $C(s)$ of visits for any given state s. This is similar to other methods, which then used states with a high frequency as sub-goals, which can lead to loops and an unnecessary high amount of sub-goals. In order to circumvent this, they calculated the relative exceeding frequency to neighboring states and chose several of the ones with the highest values. Calculating the relative exceeding frequency is done in three steps (Fig 3.2.1). First, the non-absolute difference $\delta(t_s)$ between the current s and each neighboring state t is calculated. If this value is negative, 0 is used. Second, if any of the values is 0 (meaning any neighboring state is visited more frequently), the relative exceeding frequency $\Delta(s)$ is 0, else it is the sum of all differences $\delta(t_s)$. Third, the states are ordered according to $\Delta(s)$ and as many as wanted are picked in descending value.

$$\delta(t_s) = \begin{cases} C(s) - C(t), & \text{if } C(s) > C(t) \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$$\Delta(s) = \begin{cases} \sum_{t \in neighbors(s)} \delta(t_s), & \text{if all } \delta(t_s) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Figure 3.2.1: The definitions for $\delta(t_s)$ and $\Delta(s)$.

4 Discussion and Conclusion

The presented case studies showed that using HRL to lessen the impact of the curse of dimensionality is feasible. While the central structure of both HRL approaches was similar, major differences

Part	Case study 2.1	Case study 2.2
Representation of the environment in the low level learner	Joint angles and velocities with continuous values	DMPs via goal and shape parameter
Representation in the high level learner	Discrete states of the joint angles and center of mass	Goal parameters in a continuous 2 dimensional plane
Goal for the low level learner	State of the high level learner	Point in the 2 dimensional plane
Measurement of success	Fulfillment of the task	Fulfillment of the task and how it was achieved
Abstract task	Learning of a motion	Improving a motion initially learned by imitation learning

Figure 4.1: Differences in the HRL implementations of both case studies.

existed (Fig 4.1) . In the first one a rather small amount of states for the upper learner existed and it tried to make transitions between those until the final one was reached. In the second we saw an unchangeable amount of two goals, but their values could be freely chosen by the upper learner to maximize the reward. Additionally the influence of the low level learner differed greatly. While in the first approach the low level learner only had a minor impact on the reward - it either failed and a different path was chosen or it succeeded - in the second approach it was also important how it succeded. A better path not only gave a higher chance of success, but also a lower cost function and therefor better result. These are only examples of how big the differences in HRL approaches can be, each one is tailored to efficiently solve it's given task. While each approach would most likely work on the others task after some adaption, the result should be worse or equal.

What would happen if we could judge how good a approach is to solve a certain task? If we had a metric that could specify that using PI with DMPs as in 2.2 is the best way to go to solve the given task? We could create an algorithm around this metric that uses the unchangeable facts of the system - environmental input, actions, reward/goal - and chooses which approach is the best. Afterwards we can use auto-generation methods as presented in Section 3 to actually create this approach, i.e. tailor the general variant to the task. For some approaches we cluster the basic representation to get ones for the higher layers, for others we sample some paths, weight them and cut the dimensions yielding the least information. We basically could start with a first step towards completely solving the prior knowledge problem that plagues a lot of HRL approaches. New HRL approaches could be worked into the metric, as well as improvements in the abstraction of the representation. Due to this such a metric would not be limited to the current knowledge about HRL, but could encompass any gains made in the field.
While any work in the field to improve HRL is important, such an algorithm that independently of a developers input can find a HRL solution for each task would be the most important in my view.

4.1 Further work in the field of HRL

A. Barto and S. Mahadevan give a quite detailed introduction into HRL and a possible subcharacterization of HRL in *Recent advances in hierarchical reinforcement learning*. Discrete event systems, 13(1-2):41-77, 2003.

S. Schlötzer gives a more detailed evaluation concerning 2.1 in her report *Hierarchical Reinforcement Learning*. Joint Advanced Student School 2008, St. Petersburg

5 Citations

[1] Richard S. Sutton & Andrew G. Barto (1998), *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, A Bradford Book
[2] Wikipedia, *Reinforcement Learning*
[3] Dietterich, Thomas G. *The MAXQ method for hierarchical reinforcement learning*, Proceedings of the fifteenth international conference on machine learning. Vol. 8. 1998.
[4] Vesanto, Juha, and Esa Alhoniemi. *Clustering of the self-organizing map*, Neural Networks, IEEE Transactions on 11.3 (2000): 586-600.
[5] Kohonen, Teuvo. *The self-organizing map*, Proceedings of the IEEE 78.9 (1990): 1464-1480.
[6] Equitz, William H. *A new vector quantization clustering algorithm*, Acoustics, Speech and Signal Processing, IEEE Transactions on 37.10 (1989): 1568-1575.
[7] Gray, Robert. *Vector quantization*, ASSP Magazine, IEEE 1.2 (1984): 4-29.
[8] Wikipedia, *Vector Quantization*
[9] Wikipedia, *k-means-clustering*
[10] Hartigan, John A., and Manchek A. Wong. *Algorithm AS 136: A k-means clustering algorithm*, Journal of the Royal Statistical Society. Series C (Applied Statistics) 28.1 (1979): 100-108.
[11] F. Linåker and H. Jacobsson. *Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond*, In Proc. of IJCAI'2001, 2001.

5.1 Evaluated papers

[2.1] J. Morimoto and K. Doya. *Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning*. Robotics and Autonomous Systems, 36(1):37-51, 2001.
[2.2] F. Stulp, S. Schaal. *Hierarchical Reinforcement Learning with Movement Primitives*. In 11th IEEE-RAS International Conference on Humanoid Robots, 2011.
[4.1] Bakker, Bram, and Jrgen Schmidhuber. *Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization*, Proc. of the 8-th Conf. on Intelligent Autonomous Systems. 2004.
[4.2] Chen, Fei, et al. *Connect-based subgoal discovery for options in hierarchical reinforcement learning* Natural Computation, 2007. ICNC 2007. Third International Conference on. Vol. 4. IEEE, 2007.