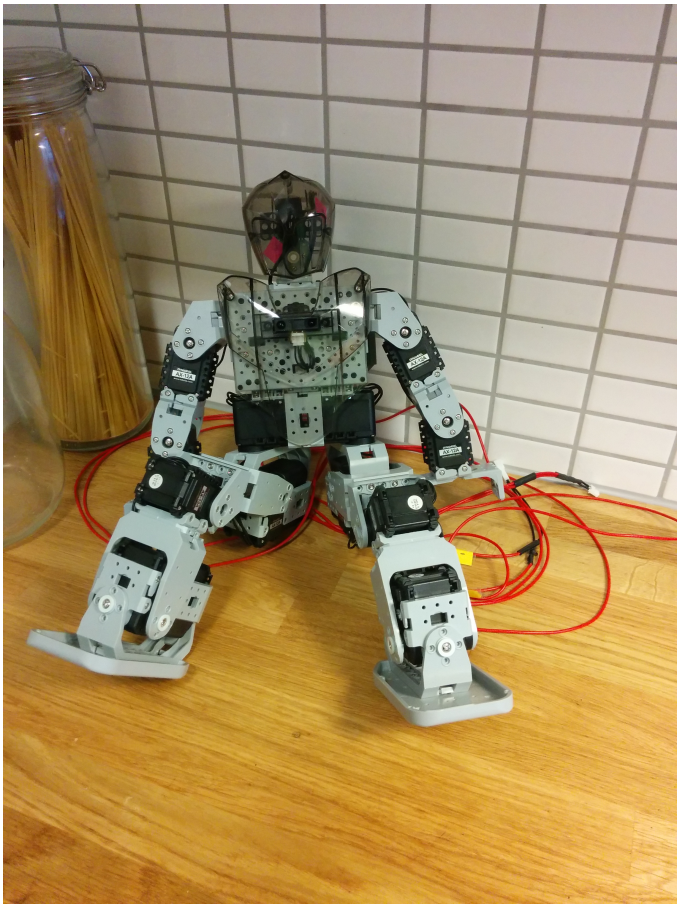


Curiosity based Self-Organization of Humanoid Robot

Master's thesis in Complex Adaptive Systems

PONTUS LOVIKEN

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

MASTER'S THESIS IN COMPLEX ADAPTIVE SYSTEMS

Curiosity based Self-Organization of Humanoid Robot

PONTUS LOVIKEN

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2015

Curiosity based Self-Organization of Humanoid Robot
PONTUS LOVIKEN

Master's thesis 2015:63
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
The Humanoid Robot chillin' out.

Chalmers Reproservice
Göteborg, Sweden 2015

Curiosity based Self-Organization of Humanoid Robot
Master's thesis in Complex Adaptive Systems
PONTUS LOVIKEN
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

This thesis presents a novel approach to how a high dimensional humanoid robot of 18 dimensions can learn within a few hours to control its body so that it is able to perform simple tasks such as rolling around or to sit up. The method is robust and works equally well when an arm is removed, and in a case where the robot was trained to use two arms and one was removed it quickly adapted to its new body. The robot is equipped with an accelerometer that measures the tilt of the torso in 2 dimensions. This "tilt"-space is divided into a discrete set of states, and the way in which the dimensionality of the servo-space is made irrelevant is to only allow one servo-configuration per state. These configurations are evolved using a Self-Organizing Map, while an Artificial Curiosity-driven Reinforcement Learner chooses what state to state transitions to attempt. An additional parameter is added in a final experiment, to see if the agent can even learn to stand. This experiment was however unsuccessful.

Keywords: Self-organized robotics, Reinforcement Learning, High DoF, Physical Environment, Humanoid Robot, Bioloid, Embodiment, Artificial Curiosity, Kulback-Liebler Divergence, Self-organizing map, HyperSOM.

Preface

This work is my master thesis and even though the subject of self-organizing robotics has interested me for some time I first got to work with it as a minor task of another course. During that work I first got to work with the Bioloid Premium [12] and also came across the concept of curiosity driven learning, which intrigued me and gave me the urge to do something more thoroughly within the field. In search for a feasible subject me and some friends therefore attended the Seventh International Workshop of Guided Self-Organization, which was that year (2014) held in Freiburg, Germany. There I got to talk with many interesting people, but also get familiar with concepts such as the Homeokinetic Principle [11] and embodiment, which later helped me view the problem of self-organizing robotics from new angles. With these new concepts in mind I started to more seriously consider different ways in which to implement it on the Bioloid, but it wasn't really until the end of my literature studies that the method I would use started to appear. The hardest problem when creating the method must have been to figure out how to reduce the vast state space of a robot of 18 servos and a two dimensional sensor. The key to finally handle this came first from the idea that the more useful body postures of the robot would lie on a hyperspace manifold. These postures would then act as an attractor of the robot. This idea was more of a gut feeling, but if there actually was such an manifold it would mean that one didn't have to look through all the robot's postures, but simply just search for the attractors. But how would one do that? By dividing up the state-space solely by the sensor-space the answer to that would be to find the most stable postures for the different tilts of the body. Since I have previously worked quite a lot with Kohonen Networks [6] I quickly came to think about self-organizing maps, which felt suitable if these more stable states really were on a manifold. Finally a method was needed in order for the robot to know how to navigate within its state-space of different sensor readings, and for that curiosity driven reinforcement learning seemed to be an excellent choice.

Acknowledgements

I would like to thank my supervisor and examiner Krister Wolff who allowed me to borrow home not only one but two Bioloids (as servos of the initial one broke down due to extensive usage) and with whom I've had a constructive dialog throughout this project. Furthermore would I like to thank all people around me who never locked me up due to my never ending rabble about self-aware robots together with my pronounced desire to create one them.

Nomenclature

RL	-	*Reinforcement Learning*, a method to learn how to navigate within a state-space.	(1.1.1)
GA	-	*Genetic Algorithm*, a stochastic optimization method that *evolves* solutions in a fashion similar to that of natural evolution.	(1.1.1)
NN	-	*(Artificial) Neural Network*, an artificial network of nodes and edges that (often) computes problems in a distributed manner.	(1.1.1)
HK	-	*Homeokinetic principle*, a method for moving embodied agents in a dynamic manner.	(1.1.1)
DoF	-	*Degrees of Freedom*, the number of dimensions in an agents state- or action-space.	(1.1.3)
AC	-	*Artificial Curiosity*, a method a RL can use in order to choose actions.	(1.2.6)
RA	-	*Random Action*, when a RL chooses between it's available actions in a uniformly random fashion.	(4.4)
$AC_{w=0}$	-	*Artificial Curiosity (original setting)*, the original implementation of AC, where only the rewards of actions are accounted for.	(2.1.4)
MDP	-	*Markov Decision Process*, a mathematical framework for modeling decision making.	(2.1.1)
KL-Divergence	-	*Kullback-Leibler Divergence* or *Relative Entropy*, expresses the information gain when improving a probability distribution with more data.	(2.1.3)
hyperSOM	-	*Hyperspace Self-Organizing Map* is a self-organizing map that predicts coordinates of nodes after some unknown transformation, if the nodes were in a grid formation before the transformation and some coordinates are known after the transformation.	(2.2)
DS	-	*Direct Space*, the space of directly controllable settings.	(2.2.1)
GS	-	*Global Space*, the space of sensor readings that can't be directly controlled.	(2.2.1)
GB	-	*Generalization Based*, a feature of a self-organizing map, meaning that unknown direct coordinates of nodes are approximated by extrapolating the known coordinates of other nodes.	(4.4)
NGB	-	*Non-Generalization Based*, means that GB isn't used.	(4.4)
PT	-	*Pre-Trained*, when a RL and a hyperSOM has previously been trained when started.	(4.5)
NPT	-	*No Pre-Training*, when a RL and a hyperSOM starts out blank.	(4.5)

Contents

Abstract i

Preface iii

Acknowledgements iii

Nomenclature v

Contents vii

1 Introduction 1
1.1 Background and Purpose . . . 1
1.1.1 Method for Adaptability . . . 1
1.1.2 Learning Speed . . . 1
1.1.3 High Degrees of Freedom . . . 1
1.1.4 Intrinsic Motivation . . . 1
1.2 Previous Work . . . 1
1.2.1 Evolved Virtual Creatures . . . 1
1.2.2 Gaits evolved in simulation, used by physical robots . . . 3
1.2.3 The Homeokinetic Principle . . . 3
1.2.4 Crawling Humanoid with Reinforcement Learning . . . 4
1.2.5 The Playground Experiment . . . 5
1.2.6 Curiosity Driven Reinforcement Learning . . . 6
1.3 Scope and Limitations . . . 7
1.4 Contributions . . . 7
1.5 Reading Guidance . . . 7

2 Theory 8
2.1 Reinforcement Learning . . . 8
2.1.1 Markov Decision Process . . . 8
2.1.2 Q-learning . . . 8
2.1.3 Artificial Curiosity . . . 9
2.1.4 Adaptation . . . 9
2.1.5 Reducing available actions . . . 10
2.1.6 Initiation . . . 10
2.1.7 Upper limit of T . . . 10
2.2 Hyperspace Self-Organizing Map . . . 10
2.2.1 Definitions . . . 10
2.2.2 Procedure . . . 12
2.2.3 The algorithm . . . 18
2.3 Derivation of Coordinates . . . 20
2.3.1 Direct Coordinates . . . 20
2.3.2 Global Coordinates . . . 20

3 Method 24
3.1 Hardware . . . 24
3.1.1 Components . . . 24
3.1.2 Connection . . . 25
3.2 Software . . . 27
3.2.1 Parameters . . . 27
3.2.2 Initialization . . . 28
3.2.3 Monitor tools . . . 28
3.2.4 Data to collect . . . 29
3.2.5 Other tests . . . 30

3.2.6 Execution . . . 31
3.3 Summary: How are the general problems tackled? . . . 31
3.3.1 Method of Adaptability . . . 32
3.3.2 High Degrees of Freedom . . . 32
3.3.3 Learning Speed . . . 32
3.3.4 Intrinsic Motivation . . . 32

4 Experiments **33**
4.1 Initial test of Curiosity Based RL . . . 33
4.2 Initial test of HyperSOM . . . 34
4.3 Experiment 1 - Cartesian vs Spherical Coordinates . . . 35
4.3.1 Description . . . 35
4.3.2 Set up . . . 35
4.3.3 Parameter choice . . . 35
4.4 Experiment 2 - Importance of Curiosity Based RL and hyperSOM generalization . . . 37
4.4.1 Description . . . 37
4.4.2 Set up . . . 37
4.5 Experiment 3 - Effect of damaged morphology . . . 38
4.5.1 Description . . . 38
4.5.2 Set up . . . 38
4.6 Experiment 4 - Effect of higher resolution . . . 39
4.6.1 Description . . . 39
4.6.2 Set up . . . 39
4.7 Experiment 5 - The Ultimate Challenge . . . 39
4.7.1 Description . . . 39
4.7.2 Set up . . . 39

5 Results **40**
5.1 Initial test of Curiosity Based RL . . . 40
5.2 Initial test of HyperSOM . . . 40
5.3 Experiment 1 - Cartesian vs Spherical Coordinates . . . 41
5.3.1 The typical run . . . 41
5.3.2 Data . . . 42
5.3.3 Evolved behaviors . . . 43
5.3.4 Evaluation . . . 43
5.4 Experiment 2 - Importance of Curiosity Based RL and hyper-SOM generalization . . . 47
5.4.1 Data . . . 47
5.4.2 Evaluation of the different settings . . . 47
5.5 Experiment 3 - Effect of damaged morphology . . . 49
5.5.1 Data . . . 49
5.5.2 Evaluation of different settings . . . 49
5.6 Experiment 4 - Effect of higher resolution . . . 52
5.6.1 Data . . . 52
5.6.2 Evaluation . . . 52
5.7 Experiment 5 - The Ultimate Challenge . . . 53
5.7.1 Data . . . 53
5.7.2 Evaluation of the two runs . . . 53

6 Discussion **57**
6.1 Interpretation of outcome . . . 57
6.1.1 Capabilities . . . 57
6.1.2 Limitations . . . 58
6.1.3 Other effects . . . 60
6.1.4 Shortcomings in experimental method . . . 61
6.2 Variations . . . 62
6.2.1 Global Parameters . . . 62
6.2.2 Coordinate system variations . . . 62

6.2.3 Action chooser . . . 62
6.2.4 State searcher . . . 63
6.2.5 Transition Record . . . 63
6.2.6 Number of direct states per node . . . 63
6.2.7 Build/decay mass . . . 64
6.3 The bigger picture . . . 64

7 Conclusion and Future Work **65**
7.1 Conclusion . . . 65
7.2 Future Work . . . 65

References **67**

1 Introduction

1.1 Background and Purpose

A robot is classically a machine built and programmed to perform a very specific task in a number of steps. Outside of this scope the robot can't do anything, unless manually reprogrammed for its new task which could be very similar to the original, without the robot being able to generalize. To address this, intense research is being put into robots that self-organize. Self-organized behavior in contrast to designed behavior has the advantage of being much more robust in the sense that it is often able to keep functionality even when components malfunction, the environment is new to the robot or even when body parts are missing. There are many different approaches to how this self-organization is made, and a few of them will be presented in section 1.2. These methods differ more or less from each other, but generally there are a few obstacles that every method needs to solve when creating self-organized robots, supposed to act in a real environment.

1.1.1 Method for Adaptability

First of all the robot needs some method that allows it to adapt. Examples of such includes: Reinforcement Learning (RL), Genetic Algorithms (GA), Neural Networks (NN) and the Homeokinetic principle (HK). How these has previously been implemented will be explained in more detail under section 1.2.

1.1.2 Learning Speed

In the case of a physical robot it is crucial that the agent is able to do the most of the information given from its interaction with the environment, since the speed at which these interactions can occur is almost always the limiting factor of its learning speed. This rules out many options that work well in simulation since the training in reality would simply take too much time, unless the simulated results can be transferred to the physical robot in some reliable way.

1.1.3 High Degrees of Freedom

One difficulty with more complex robots such as humanoids is their high degrees of freedom (DoF). Generally there are one degree of freedom per actuator and the state space is then further spanned by every sensor reading available to the robot. As the state space typically grows exponentially with the DoF some generalizations has to be made for a learning speed to be applicable.

1.1.4 Intrinsic Motivation

Why should the robot try to do anything at all? In some cases the robot is just given a fitness function and its goal is simply to optimize this measure. This is convenient as it gives an absolute measure of a solution but it also brings a number of problems. To begin with how to set the fitness function. What number would capture the intended outcome the best? And what if some other task is later required of the robot? A different approach is that of Artificial Curiosity where the agent rather tries to build an as extensive picture of its capabilities as possible, by focusing on the activities in which the learning progress is the greatest.

1.2 Previous Work

In this section follows a number of different work that has been done within the field.

1.2.1 Evolved Virtual Creatures

In Karl Sims classical work from 1995 [14] he showed that both morphology and complex behaviors can be developed in a simulated environment by the use of evolution. The creatures were evolved to perform a number of distinct task such as swimming or running, to following a moving point or competing for a cube. A video of them in action can be seen at `https://www.youtube.com/watch?v=JBgG_VSP7f8`, and an example of the

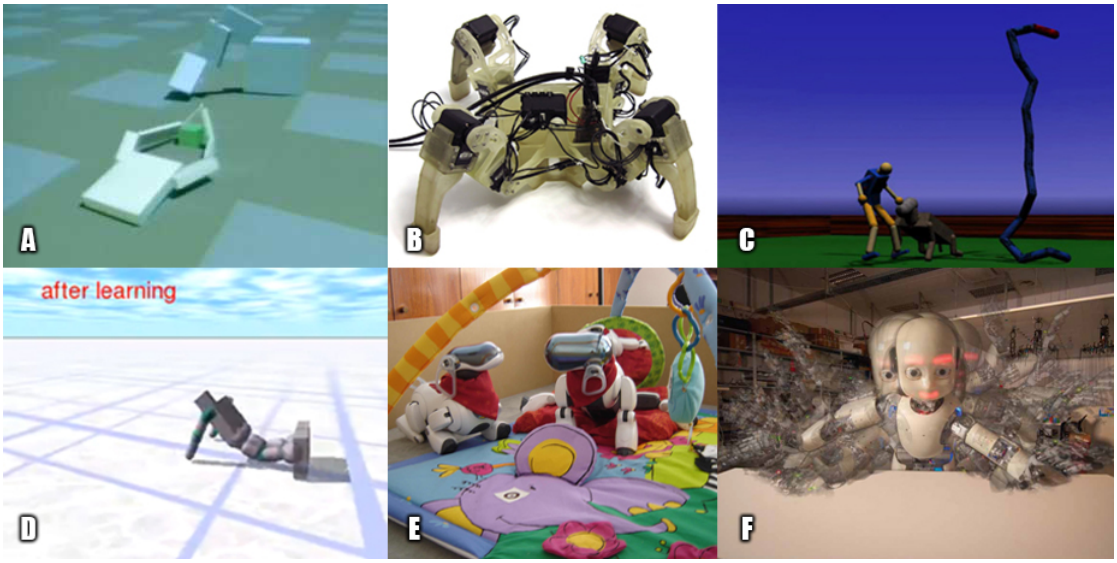


Figure 1.1: **A:** Evolved virtual creatures fighting for dominance over the green cube..**B:** The robot for which gates where evolved using hyperNEAT. **C:** An image for a run where three different creatures were controlled by the same basic homeokinetic driven NNs. **D:** A crawling behavior evolved using DCOB. **E:** The Playground Experiment. The dog in the middle is able to interact with the objects in its surrounding using some basic actions. **F:** An iCub humanoid robot where each arm and the torso are independent agents that are using AC in order to investigate their state spaces as efficiently as possible.

cube competing behavior can be seen in Figure 1.1 A. Karl Sims work is often considered a milestone in the science of self-organized behaviors.

Method for Adaptability

An GA was used with which both the morphology of the creatures and the recurrent NNs controlling them were evolved.

Learning Speed

This work was made for simulated creatures first hand so this work doesn't really handle the problem of how to achieve a learning speed applicable to physical agents.

High Degrees of Freedom

In his work he allowed every body-segment to have its own simple recurrent NN, together with a global brain for synchronization. By doing so the dimensionality is drastically reduced since every actuator is more or less controlled by an independent NN.

Intrinsic Motivation

As often is the case in GAs the "motivation" of the agents was basically to score as high as possible on some fitness function, such as swimming, walking, jumping and following.

Result

His work was highly influential since it showed beyond any doubt that an evolutionary process could be used in order to develop behaviors and bodies adapted for various kinds of tasks.

1.2.2 Gaits evolved in simulation, used by physical robots

In a more recent paper [2] evolution was once again used in order to develop behaviors, but now i order to produce gaits in physical robots. The gaits was however not evolved using the physical robot, but rather simulations of it. Gaits developed through evolution has been done multiple times, but what distinguishes this work from previous ones is mainly the choice of NN controlling the agent. The hyperNEAT network [5], with Tod Lipson and Jeff Clune as two of its more visible proponents, is special in the sense that the edges of the network are not directly coded. Instead they are coded indirectly by an underlying evolving function, made in such a way that symmetries and regularities appear naturally. An analogue that is often used is that nature doesn't code for each of our fingers individually (which would be a direct encoding), but instead has a general plan for how to make a finger and then just use that one instruction over and over again for every finger but with smaller variation (indirect encoding). The hyperNEAT is made to mimic this ability which thus allows the network to reuse useful structures it finds for one area to other areas, with modification. This allows creatures evolved by hyperNEAT to use their bodies in a more symmetric and natural way than with the classical direct encoding, where every weight is evolved independently. A video and more information can be found at `http://creativemachines.cornell.edu/evolved-quadruped-gaits`.

Method for Adaptability

In this work they evolved gates in a simulation using hyperNEAT. Once evolved the resulting gates were used by physical robots. The robot used can be seen in Figure 1.1 B. The adaptability of the method is thus limited, since a simulated robot was needed to begin with. In an earlier work by Hod Lipson and others [4] this was solved by first letting the robot create a simulation of itself by observing its interaction with the surrounding in order to create a model of itself. A TED talk about this can be seen at: `http://www.ted.com/talks/hod_lipson_builds_self_aware_robots?language=en`.

Learning Speed

Due to the simulation step it was possible to produce gaits in a reasonable amount of time. Since the gates were the result of 40'000 evaluated runs it is clear that this step was necessary to have an acceptable learning speed using this method.

High Degrees of Freedom

By being able to utilize symmetries and repetition hyperNEAT is able to generalize to a high degree, which decreases the complexity a lot. It also helps in this case that the robot itself had a high degree of symmetries, which makes such generalizations easier.

Intrinsic Motivation

The motivation is to optimize a fitness function measuring the speed of the robot, or more precisely the distance traveled by the robot within some time.

Result

The evolved gates are claimed to be the fastest ones ever observed for the robot, even though it must be mentioned that the speed of the real robot was only 57% the speed of the simulated robot.

1.2.3 The Homeokinetic Principle

The Homeokinetic Principle, HK, first presented in the book "The Playful Machine" [11] by Ralf Der and Georg Martius presents a rather different approach to the subject of self-organizing robotics. HK is a method that rapidly adapts to a given body and starts to control that body in a dynamic way. It has mostly been tested in simulated environments since the dynamic behaviors produced might damage components of physical robots, even though some physical implementations have been made. It is closely connected to the homeostatic principle from which it originated, which is also described in the same book. Videos of experiments using HK may be seen at `http://www.playfulmachines.com/videos.html`, and Figure 1.1 C shows a frame from one such experiment.

Method for Adaptability

In this approach the robot is controlled by two feed forward NNs that are constantly updated using back-propagation. The first network uses the agent's action and tries to predict the resulting state, while the other networks uses the state and tries to pick the action which is the hardest for the first network to predict the outcome of, using a gradient. This constantly leads the agent to move towards its more dynamic areas of the state-space which often results in interesting behaviors that are tightly connected to the shape of the agent's body.

Learning Speed

Due to the simple nature of the two networks, training is almost instant.

High Degrees of Freedom

The degrees of freedom doesn't seem to be a problem since there is no goal of exploring the full space, but just to follow the gradient towards the states of highest uncertainty.

Intrinsic Motivation

The intrinsic motivation is to do the locally most dynamic thing possible, at all times.

Result

It shows some pretty interesting results in many systems, typically more interesting the more inherently unstable the systems are. In the case of humanoids however it seems like a lying humanoid is too stable to be able to display any interesting movements. For humanoids to do more interesting things they need to be suspended into the air by a bungee line, put in a low gravity environment, or similar.

1.2.4 Crawling Humanoid with Reinforcement Learning

In this work [1], both simulated robots and real robots were trained using RL with a discrete action space, called DCOB (Directed to Center Of a target Base-function). Most emphasis is put on a simulated humanoid robot that is supposed to learn how to crawl as fast as possible, and the aim of the work is to find a way to implement RL on a system of as high DoF as a humanoid robot. It was never implemented on a physical humanoid robot, but the same method was also used on a physical spider robot.

Method for Adaptability

Reinforcement Learning together with Peng's $Q(\lambda)$-learning [3] where used in this case.

High Degrees of Freedom

RL will be more thoroughly explained in section 3.2.2 but in short one needs to divide an agents state space into a discrete set of states. This basically means that every state corresponds to a specific body-posture, and then the RL figures out the best way to move between these body postures (states) in order to accomplish some goal. In this case forward crawling motion. The typical way to divide up the state space is by a grid, where every dimension has at least a couple divisions. In practice this means that a robot in one state will move to another state if for example one arm is sufficiently moved. This would mean that the number of states scales exponentially with the DoF, which would mean that their robot of 16 joints would at least have $2^{16} = 65'536$ different states, if every joint were just able to have two discrete settings. This isn't feasible so what they basically did was to couple symmetric joints to move the same way, and enforced other similar restrictions on their robots to create systems between 3 and 7 DoF. In the cases of higher degree was the states not in a grid formation but rather was a preset number of states more or less randomly spread out in its state space. Using these methods they got a system small enough to be trained in simulation.

Learning Speed

Their most important effort in increasing the learning speed was to reduce the search space by the restrictions enforced on the system to reducing the DoF. In the end they had 125 states in the 3 DoF case, and 600 states in the 7 DoF case. The training of the humanoid were performed in simulations where the systems needed between 8000 runs in the 3 DoF case to 16000 runs in the 7 DoF case before any kind of significant progress began to appear. This corresponded roughly their training time. Every run lasted for up to 20 seconds in simulated time, meaning that one would need between 40 and 80 hours respectively in order to acquire some progress in an entirely physical system, and twice as much time to get their results.

Intrinsic Motivation

The motivation of this crawling task was to crawl as far as possible during 20 seconds if starting standing, and to not hit the head in the ground.

Result

The robots learned to crawl with varying quality depending on the given DoF. An image from a resulting behavior may be seen in Figure 1.1 D, and an animation of the same behavior may be seen at `https://www.youtube.com/watch?v=bvbf9ly8Ep4`.

1.2.5 The Playground Experiment

In this experiment by Pierre-Yves Oudeyer and others in their work "Intrinsic Motivation Systems for Autonomous Mental Development" [10] a dog robot was put on a playground mat allowed a limited number of actions to do, such as punch or bite objects in its surrounding. The emphasis wasn't to have the robot do any advanced things, but rather to see if it could be trained in a way more similar to that of human infants, where the driving force behind actions where curiosity. The set-up of the experiment can be seen in Figure 1.1 E.

Method for Adaptability

An important issue regarding self-organizing robots is that of *Intrinsic Motivation*: What is the robot's motivation do anything at all? As we have earlier seen the motivation has previously often been of the form to perform a specific task as well as possible. In this case a framework for the method "Intelligent Adaptive Curiosity" is defined. It is inspired by the cognitive development of children and states that the agent should focus on the tasks in which it thinks it will learn the most by doing. That is: Avoid to do things you already know the outcome of, and don't do things that you don't comprehend at all. Focus on what is in your reach, but not yet fully understood. Practically this was done by letting the agent chose between a number of high level actions, such as trying to hit or bite objects in its surrounding. It had different "experts" for the different kinds of tasks it could choose to perform and the role of these experts were to predict the outcome. Then the quality of each expert was monitored by a "meta-learner" to see what expert learned the most. 65% of the time the agent did the actions of the expert that learned the fastest and the rest of the time it did random actions.

High Degrees of Freedom

Since high level actions were predefined the action-space was quite limited.

Learning Speed

The small action space ensured a rapid training speed, but also the curiosity based learning system helped it to learn quicker since it payed more attention to the areas of greater progress.

Intrinsic Motivation

Intelligent Adaptive Curiosity (Not the same as Artificial Curiosity). The motivation was to learn as much as possible which led the robot to begin with the easier tasks to then lose interest in them as they became better understood, to then switch its attention to more complex tasks.

Result

The agent wasn't able to learn very much to begin with since its number of actions was very limited, but the important feature of this experiment was to practically show how curiosity based learning could be implemented on a real system, and that it indeed made the agent choose tasks to try in an intelligent and adaptable manner.

1.2.6 Curiosity Driven Reinforcement Learning

In the paper "Curiosity driven reinforcement learning for motion planning on humanoids" by Mikhail Frank and others [13] the upper body of a humanoid robot is allowed to move freely above a table. Restrictions are put so that its movements are stopped if it is about to touch the table or stress one of its cables. The goal of it is to use RL in order to learn how to get between general states of its state-space, and do the training using Artificial Curiosity (AC), a method presented in the paper that is supposed to optimize the amount learned.

Method for Adaptability

RL by AC. AC will be more thoroughly explained in section 2.1.3, but in short does the agent measure the information gained by any attempted transition between two states. This information acts as the reward in the method, meaning that the agent always tries to do the actions giving it the most new information, and because of that it is always driven towards the actions giving it the greatest learning curve. This method is inspired but not identical to the method of the playground experiment.

High Degrees of Freedom

The state space is in this task divided in a regular grid-based way which is often typical in RL applications. To avoid the dimensionality of all the actuators the robot is therefore divided into smaller independent parts, so that every arm is an independent agent and so is the torso. Each arm is then a 4-dimensional agent and the torso is a 3-dimensional agent.

Learning Speed

Since dimensionality is low is there only around 80 states of any agent. Then transitions where only allowed to the closest neighbors of any state leading to a limited number of actions to try out.

Intrinsic Motivation

The intrinsic motivation was to maximize information gain, which was done using AC.

Result

Comparisons to two other methods, random exploration and to always do the least tried action, showed that the agent found the states at a rate only slightly faster than the alternatives, but when it came to try all possible actions it was able to find all actions more than twice as fast as the second best. Anyhow was it able to visit half of all the states after around 60 performed actions, all states around 600 actions, and all possible actions after 1'600 actions. Visually the agent moved its arms around in the air, as seen in Figure 1.1, and especially became interested of the transitions where the low-level controller couldn't steer it properly, since the outcomes in those cases were the most surprising. When the torso moved freely the table became especially interesting to the arms, since the table then seemed to move around.

1.3 Scope and Limitations

The purpose of this thesis is to create a framework with which a physical generally shaped robot of high DoF is able to learn independently how to use its body in a sensible way. Furthermore should the robot be able to notice and react to changes in its body, and adapt to those changes. Training should be rapid enough so that significant results can be seen after at most a couple of hours. Different settings will be tried out, but since the robot will operate in a real environment the running time will be limited and often only one or a few tests will be allowed to any setting. To simplify the learning progress will the environment be kept as static as possible. Finally this work should first and foremost be seen as a proof of concept. This thesis shows a novel approach to the field of self-organizing robotics, but no claims are made that the exact implementation in this work is the ultimate one within that approach.

1.4 Contributions

No previous work seems to be able to handle high DoF in a sensible way. The contribution of this work will be to show a framework for which high DoF no longer cause a problem, and where additional DoF only allows for more efficient and easily found solutions, without increasing the search space in the process. Furthermore does the method allow a user to later control the agent within the state-space examined and understood by the robot.

1.5 Reading Guidance

This concludes Chapter 1 containing the *Introduction*.

Chapter 2, *Theory*, will present the frameworks of the main methods used in this work. It begins with the classical but jet very much alive field of Reinforcement Learning, to then go on to introduce the Hyperspace Self-Organizing Map which, inspired by other self organizing maps is specifically created for this work. In a final section will the derivation of the agents coordinates be presented.

Chapter 3, *Method*, handles the actual implementation of this work. To begin with hardware wise and then software wise. Finally is there a summation of the method all in all, in the same way methods of previous work have been analyzed, to simplify for the reader to compare the method of this work with the earlier ones.

Chapter 4, *Experiments* presents the different experiments that have been performed. It starts with two smaller test performed in order to very that both the Reinforcement Learner and the Self-Organizing Map work as intended, followed by 5 different experiments on the whole physical system.

Chapter 5, *Results*, looks at the outcomes of these experiments, while Chapter 6, *Discussion*, analyzes these results but also the effects of different possible variations to the method. In a final section the wider implications of the method are discussed.

Chapter 7, *Conclusion and Future Work* concludes the work, and mention possible future work that can be done using the method.

2 Theory

2.1 Reinforcement Learning

The field of Reinforcement Learning (RL) is a vast one, and in this thesis will only the parts most relevant to this work be covered. If the reader wants a more extensive coverage of the field the introductory book [15] is recommended.
RL is in its core, as an agent, to learn what action $a \in \mathcal{A}$ to take in a state $s \in \mathcal{S}$ in order to maximize some future reward $r \in \mathbb{R}$. The agent doesn't know in advance what the reward of an action done at a particular state will be, and must therefore find it out by a trial-and-error search. Furthermore it is often the case in reinforcement learning that a reward of an action isn't immediate. In the game of chess e.g, the first move will never give an immediate reward, but nonetheless it highly affects the chances to later score rewards by capturing enemy pieces or winning the game.

2.1.1 Markov Decision Process

Reinforcement Learning is often taking place within a Markov Decision Process (MDP). It is defined by the following features:

- A state-space $\mathcal{S}$ with a discrete number of *states* s_i.
- A set of action-spaces $\{\mathcal{A}_i\}_{i=1}^{|\mathcal{S}|}$ with *actions* $\{a_{ij}\}_{j=1}^{|\mathcal{A}_i|}$ that can be performed while in state s_i.
- A *probability function* $P_a(s, s')$, telling the probability to transition to state s' if performing action a when in state s.
- A *reward function* $R_a(s, s')$ telling the yielded reward for doing doing the transition $s \xrightarrow{a} s'$. Rewards can be both positive and negative in order to encourage or discourage behaviors.
- A *policy* $\pi(s) \in \mathcal{A}$ of what action to do in some state s.

The core problem of a MDP is to find an *optimal policy*, or equivalently to figure out the way in which to tell the agent what to do in every situation, in order to optimize the expected future reward. With a policy (optimal or not) there is thus exactly one instruction of what to do for every state the agent might find itself in. Since $\pi(s)$ makes the action of every state deterministic a value $V^\pi(s)$ can be set for every state s which represents the expected future reward if strictly following the policy π from a start point of s. We call this function $V^\pi(s)$ for the *state-value function (for policy π)*. With this, the optimal policy function and the state-value-function $V(s)$ under this policy (superscript π is dropped when using optimal policy) can be recursively defined as:

$$\begin{aligned} \pi(s) &\equiv \arg\max_a \{\textstyle\sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V(s'))\} \\ V(s) &\equiv \textstyle\sum_{s'} P_{\pi(s)}(s, s')(R_{\pi(s)}(s, s') + \gamma V(s')) \end{aligned} \tag{2.1}$$

where $0 \leq \gamma < 1$ is the *discount factor*, which weights the importance of immediate rewards ($\gamma \to 0$) contra future rewards ($\gamma \to 1$).

2.1.2 Q-learning

In the case of RL, P_a and R_a are not known and must be built simultaneously with the policy search. From here on, P_a, R_a, and V will no longer represent the true functions, but rather the agents best approximations of them. A function Q is defined

$$Q(s, a) \equiv \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V(s')). \tag{2.2}$$

The Q-value expresses the expected future reward of the *state-action pair* (s, a). Basically it computes the mean reward of doing an action a when in state s, where every possible outcome of the state-action is weighed against how likely that outcome is. Since the Q-value will always tell us the expected future reward for any

action, an optimal policy would therefore be to always do the action with the highest Q-value. Following this policy it is thus possible to rewrite V to

$$V(s) = \arg\max_a \{Q(s,a)\}. \tag{2.3}$$

since we are counting on the agent to follow the same policy in every other state as well. Note however that this is only the optimal policy if P_a and R_a are the true functions, which they are usually not. In most applications that means that some exploration is needed while not following the Q-values.

2.1.3 Artificial Curiosity

One of the most efficient ways to build up at least the P_a matrix, is to actually use a new independent set of Q-values, but where the reward is now based on the improvement of P_a by doing the action. This can actually be done without an initial "goal" of the agent, so that no initial reward-function is needed at all. Instead one would get an agent which main focus is to learn how to operate within its state-space as efficiently as possible. Learning then becomes the goal in itself, which makes it an *intrinsic motivation*. This method is called Artificial Curiosity (AC) and was first proposed in [13]. It works by defining the reward-function so that the reward $R_a(s,s')$ is equal to the improvement of $P_a(s,s')$ by doing so. This is measured by the Kullback-Leibler Divergence [7] (also called relative entropy) defined

$$D_{KL}(\mathbf{P}||\mathbf{P'}) = \sum_i P'_i \ln\left(\frac{P'_i}{P_i}\right). \tag{2.4}$$

where

$$\mathbf{P} = [P_1 \ P_2 \ \ldots] \tag{2.5}$$

is the old probability distribution and $\mathbf{P'}$ is the updated one. P_i would thus be the the old probability estimate that event i would occur. This probability is computed by dividing the number of times the state-action lead to the outcome i. In its essence the KL-divergence measures the information gained by improving the probability distribution by an additional transition. The interesting thing with such a measure is that the Q-learning can be used at all times since the reward numbers really are the true ones, and also because state-actions becomes less interesting as their probability functions converges. This way will the agent always look for the state-actions that surprises it the most.

2.1.4 Adaptation

When using this in a RL some adaptations are useful. First of all are often the actions denoted as states. The action s' then means that an agent should attempt to do a transition to the state s'. The recorded transitions can be denoted so that $T(s_i, s_g, s_f)$ represents the number of times an attempted transition from s_i (i: initial) to s_g (g: goal) has ended up in state s_f (f: final). The probability $P(s_i, s_g, s_f)$ is then

$$P(s_i, s_g, s_f) = \frac{T(s_i, s_g, s_f)}{\sum_{j=1}^{N} T(s_i, s_g, s_j)} \tag{2.6}$$

One problem with this is that if there are many states these matrices might become very large. One way to make them smaller is to instead just look at the probability that a transition will reach its goal or not, so that $T(s_i, s_g, success/fail)$ denotes the times an attempted transition $s_i \to s_g$ has either failed or succeeded. $P(s_i, s_g)$ is then the probability that the transition is going to succeed from passed experience. More on this topic can be found under Discussion. A final adaptation to this work is to also compute the probability that a state can be reached at all, regardless of from which state the action is performed. This probability is simply denoted $P(s_g)$, and using this, the final reward of a state-action from s_i to s_g is:

$$R_{s_g}(s_i, s_g) = D_{KL}(\mathbf{P}(s_g)||\mathbf{P'}(s_g)) \times D_{KL}(\mathbf{P}(s_i, s_g)||\mathbf{P'}(s_i, s_g)) \tag{2.7}$$

Where

$$\mathbf{P}(s_g) = [P(s_g) \ \ (1 - P(s_g))] \tag{2.8}$$

etc. This setting will be referred to as AC throughout this report, but observe that the original setting suggested in [13] only used the KL-divergence associated with state actions, and not the more general one that computed the probabilities to reach states at all. This original setting will however be tested, but will then be referred to as $\text{AC}_{w=0}$.

2.1.5 Reducing available actions

If any state was allowed to attempt a transition to any other state then the search space would easily become too vast for any practical purpose. Therefore are often the number of allowed actions at any state limited in some way. A common way to do this is by the use of an action radius r_a. As action s_g is then only allowed if the distance between s_g and the current state s_i is less than r_a in some measuring system, often neighborhood distance is the states are ordered in some sort of grid. Another way to limit the number of actions is to only allow actions that the agent has already earlier performed from that state, either by initiation or by failure to reach some other state.

2.1.6 Initiation

To be able to compute D_{KL} an initial probability-distribution is needed to avoid division by zero. This is done by recording one successful and one failed transition to every direct neighbor of a state and also with itself. If a new allowed action is seen, that is if the agent misses the state it aimed for and instead went to a state that is not its direct neighbor but still within r_a, then that transition is also initiated the same way before it is updated with the actual transition to it and the reward is measured.

2.1.7 Upper limit of T

If the underlying probability function isn't static, that is if the probability of a state-action success changes over time, then it might be wise to weigh later transitions higher than earlier ones. One way to do this is to not allow the sum $T(s_i, s_g, success) + T(s_i, s_g, fail)$ to surpass some predefined value. This limit is called $max_{T_{SA}}$ for state-action transitions, and max_{T_S} for the total number of failed or successful transitions to some state. If the number of transitions then surpasses this value the matrices are rescaled so that:

$$T \leftarrow \left(\frac{T}{\sum T}\right) max_T \tag{2.9}$$

This way will the probabilities stay unchanged, while later measures are given greater importance than older ones.

2.2 Hyperspace Self-Organizing Map

The goal of a hyperSOM (Hyperspace Self-Organizing Map) is to find out how to use controllable settings in order to control other settings indirectly, and vice versa: what settings to choose in order to get a desired outcome of indirect settings. HyperSOM is inspired by Kohonen Networks [6] but shouldn't be confused with it since there are some fundamental differences.

2.2.1 Definitions

Direct and Global Space

To formalize the hyperSOM some definition might be useful:

- The *Direct Space* (DS) spans all possible settings that can be directly controlled. Example: A speaker is in direct control over what frequency to play. The frequency is thus part of DS.
- The *Global Space* (GS) spans sensor readings that can't be directly controlled, but still are indirectly affected by the choice of controllable settings. Example: A speaker can cause oscillations to an empty glass of wine, but only as an indirect consequence of the chosen frequency. The oscillation is therefore indirectly controllable and thus part of GS.
- A *Direct Coordinate* $\mathbf{x} \in$ DS, $\mathbf{x} = [x_1\ x_2\ \dots\ x_m]^T$ correspond to a specific setting in DS, where x_i is the setting of the i:th controllable parameter and m is the dimension of DS.
- A *Global Coordinate* $\mathbf{g} \in$ GS, $\mathbf{g} = [g_1\ g_2\ \dots\ g_n]^T$ corresponds to a specific setting in GS, where g_i is the value of the i:th parameter that one wants to learn how to control and n is the dimension of GS.

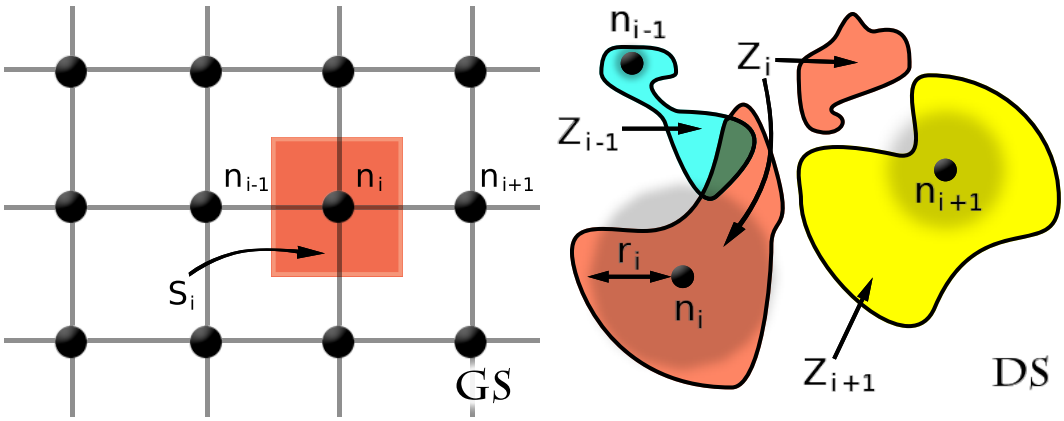


Figure 2.1: *Left*: The nodes are spread out in GS in a grid. A global coordinate $\mathbf{g}$ belongs to the state for which node it is closest to. The red square indicates the coordinates that consequently belongs to state $\mathcal{S}_i$. *Right*: How the nodes might be spreaded out in DS. The darkened area around the nodes corresponds to the certainty of a nodes coordinate, with r_i inversely proportional to the mass m_i of the node. The different areas corresponds to different Sets.

Nodes

In its essence, hyperSOM consists of a set of nodes $\{n_i\}_{i=1}^{N}$ where every node maps a fixed global coordinate $\mathbf{g}_i$ to an evolving direct coordinate $\mathbf{x}_i$. The nodes global coordinates are arranged in a uniform grid in GS, where the dimension of the grid is the same as the dimension of GS. The relative position of the node n_i in the grid is $\mathbf{k}_i$, and finally every node has a "mass" m_i that roughly corresponds to how reliable the correspondence

$$\mathbf{x} \to \mathbf{x}_i \Rightarrow \mathbf{g} \to \mathbf{g}_i \tag{2.10}$$

appears to be. In other words: If the agent moves to the direct coordinates $\mathbf{x}_i$, how likely is it that the global coordinates will follow to $\mathbf{g}_i$? This means that the more certain a mapping is, the more massive the node will be.

A node n_i is thus defined by:

- A fixed global coordinate $\mathbf{g}_i \in$ GS.
- A grid-position $\mathbf{k}_i \in \mathbb{Z}^{dim(\mathrm{GS})}$.
- A direct coordinate $\mathbf{x}_i \in$ DS that evolves over time.
- A mass $m_i \geq 0$ that is related to the degree by which: $\mathbf{x} \to \mathbf{x}_i \Rightarrow \mathbf{g} \to \mathbf{g}_i$.

States

Every node represents a state so that state s_i is represented by node n_i. For this reason states and nodes will often be used interchangeably. An agent with coordinates $(\mathbf{g}', \mathbf{x}')$ is in s_i if

$$\|\mathbf{g}' - \mathbf{g}_i\| < \|\mathbf{g}' - \mathbf{g}_j\|\,, \ \forall\ j \neq i \tag{2.11}$$

That is, the state of the agent corresponds to the node that is closest to the agent in GS. Figure 2.1 (left) displays the layout of the nodes in GS in a two-dimensional case.

Sets

Since the global coordinate $\mathbf{x}'$ was possible in s_i we say that $\mathbf{x}'$ belongs to the *Set* i, or equally: $\mathbf{x}' \in \mathcal{Z}_i$. Formally:

$$\mathbf{x}' \in \mathcal{Z}_i \text{ if,}$$
$$\text{there exists a } \textit{stable} \text{ coordinate pair } (\mathbf{g}', \mathbf{x}'),$$
$$\text{where } \mathbf{g}' \in s_i.$$

Observe that:

1. The sets are open.
2. The sets are not convex.
3. A coordinate $\mathbf{x} \in$ DS may belong to multiple sets.

Explanations of these statements follows:

1: If $\mathbf{x} \in \mathcal{Z}$, then $\mathbf{x} + \mathbf{\Delta x} \in \mathcal{Z}$ too, for sufficiently small deviations $\mathbf{\Delta x}$.

2: Two coordinates in DS can belong to the same set, while a coordinate on a line inbetween may not.
Example: Frequencies $f_1, f_2 \in$ DS makes a glass vibrate. Therefore $f_1, f_2 \in \mathcal{Z}_{vibrate}$. Create a new frequency f_k:

$$f_k = kf_1 + (1-k)f_2, \tag{2.12}$$

with $k \in [0, 1]$. It is in this case possible that $f_k \notin \mathcal{Z}_{vibrate}$, and $\mathcal{Z}_{vibrate}$ is thus not a convex set.

3: The same direct coordinate $\mathbf{x}$ can keep an agent stable in GS, for more than one state.
Example: $l \in$ DS determines the length of a cylinder with a fixed radius. The state of the cylinder can either be to stand on one of the two faces, or to lie on the side: $\mathcal{S} = \{s_{face}, s_{side}\}$. If $l \to 0$ then only s_{face} is stable (like a coin), and when $l \to \infty$ only s_{side} is stable (like a straw). In between there are however choices for which both states are stable. Thus $l \in (\mathcal{Z}_{face}$ AND $\mathcal{Z}_{side})$, if $0 < l < \infty$.

2.2.2 Procedure

The goal is to find the "best" direct coordinate $\mathbf{x}_i$ for every node n_i. A "good" coordinate $\mathbf{x}_i \in \mathcal{Z}_i$ is recognized by that:

1. The coordinate $\mathbf{x}_i$ is close to $\mathbf{x}_j$, if $\mathbf{g}_i$ is close to $\mathbf{g}_j$.
2. There are many states s_j, for which $\mathbf{x}_j \to \mathbf{x}_i$ reliably leads to the transition $\mathbf{g} \to \mathbf{g}_i$.
3. $\mathbf{x}_i' = \mathbf{x}_i + \mathbf{\Delta x}$ is also a good coordinate, for reasonably small deviations $\mathbf{\Delta x}$.

To sum it up: We want an agent that goes between close global coordinates with minimal change in direct coordinates (1), where every state can be reliably reached from a multitude of other states (2) and where smaller deviations from the direct coordinate doesn't matter too much (3).

Exploration & Exploitation

To find suitable values $\mathbf{x}_i$ some kind of search needs to be done. When an agent wants to reach a state s_i it therefore don't just sets its direct coordinates to $\mathbf{x}_i$, but rather explores the vicinity of it by going to a coordinate $\mathbf{x}_{try}$, defined by:

$$\mathbf{x}_{try} = \mathbf{x}_i + r_i \mathbf{\Delta x} \tag{2.13}$$

where

$$r_i = \frac{1}{1 + m_i} \tag{2.14}$$

and

$$\mathbf{\Delta x} = w[\Delta x_1 \ \Delta x_2 \ \ldots \ \Delta x_n]^T \tag{2.15}$$

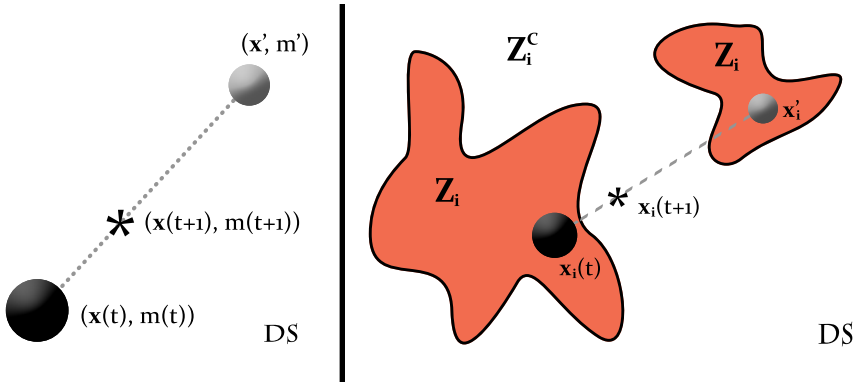


Figure 2.2: Left figure shows how a now direct coordinate $\mathbf{x}(t+1)$ is created from the old position $\mathbf{x}(t)$ and the new reading $\mathbf{x}'$. Right figure shows that this operation may sometimes move the next direct coordinate to a position outside of the given set, even though both the points used are within the set. This is a consequence of the sets not being convex.

where w and $\{\Delta x_i\}_{i=1}^n$ are random numbers with w uniformly distributed $[0,1]$ and Δx_i normal distributed with mean 0 and variance 1. The effect of this is that a goal state with small mass will result in a large search radius, as indicated by Figure 2.1 (right), while a state with a large mass will be highly deterministic, e.g mostly just exploiting what is already known with a small search radius. Considering that mass is connected to how good a direct coordinate is this means that the agent chances more the less sure it is of a goal, and plays it safe when a good goal is known. This way the mass helps balancing the exploration/exploitation in an elegant way.

Mass increase

The mass of a node is an important measure that basically approximates the quality of a certain choice $\mathbf{x}_i$. To increase the mass of a node we thus need some data that is useful for the overall quality. Suppose an agent (at rest) is at the direct coordinate $\mathbf{x}'$ and at the global coordinate $\mathbf{g}'$. This is useful information since it reveals a possible direct coordinate to the state of the agent (determined by 2.11). How useful the datapoint $(\mathbf{x}', \mathbf{g}')$ is, is (among other factors) related to how centered $\mathbf{g}'$ is in the state. A global position that is almost equally close to two different nodes is for example less valuable since it is then more likely that a small perturbation in $\mathbf{\Delta x}$ would make the agent miss the state, meaning that that coordinate is less valuable by earlier definition. In Figure 2.1 (left) this is basically a question on how centered the datapoint is within the red square.
The mass of a datapoint will be our measure of how valuable the datapoint is. If $\mathbf{g}_{1^{st}}$ and $\mathbf{g}_{2^{nd}}$ are the global coordinates of the two nodes closest to $\mathbf{g}'$ in GS, then the mass of the datapoint is:

$$m' = 1 - \frac{\|\mathbf{g}_{1^{st}} - \mathbf{g}'\|}{\|\mathbf{g}_{2^{nd}} - \mathbf{g}'\|} \tag{2.16}$$

Notice that this datapoint belongs to the state $s_{1^{st}}$ (from definition 2.11). This means that $m \in [0,1]$ with $m = 1$ if the datapoint is spot on the state, and 0 if it is equally close to the two closest nodes. Using this the mass and the direct coordinate of the node can be updated as following:

$$\begin{aligned} \mathbf{x}_i(t+1) &= \frac{m_i(t)\mathbf{x}_i(t) + m'\mathbf{x}'}{m_i(t) + m'} \\ m_i(t+1) &= m_i(t) + m' \end{aligned} \tag{2.17}$$

The new coordinate $\mathbf{x}_i(t+1)$ is thus the center of mass of the previous coordinate $\mathbf{x}_i(t)$ and the new datapoint $\mathbf{x}'$, se Figure 2.2 (left).

Mass decay

In previous section we described positive data. That is data that told us what worked in some cases. Not all coordinates within a set are however equally good, and in order to find the better ones we also need to

communicate negative feedback to the agent. If the agent tries to reach a state unsuccessfully that is such an occasion. There are multiple reasons why a transition might fail.

- The goal coordinate $\mathbf{x}_{try}$ might actually be outside of the set. Either because the radius r_i allows it to be, or because the center of mass actually ends up outside of the set since the sets are not convex, see Figure 2.2 (right).
- $\mathbf{x}_{try}$ belongs to multiple sets, and for the attempted transition some other state was more likely of being the outcome.
- There are physical limitations stopping the agent from actually reaching $\mathbf{x}_{try}$. Such limitations might be anything from psychical obstacles or not enough time, to the agent getting stuck in itself or some kind of malfunction in the agent.

When a transition fails, that is proof that the current coordinate isn't good enough. The way this is communicated is by decreasing the mass,

$$m_i(t+1) = \lambda m_i(t) \tag{2.18}$$

with $\lambda \in [0, 1]$. This operation have two major consequences for the agent:

1. It tells the agent to increase its search radius.
2. It tells the agent to increase the importance of new findings.

Another way to look at this is that the agent thinks it knows how to get to a state, but when it fails it becomes more unsure and if it finds another way to do it it would be more interested in that way.

The combined effect

As explained there are two mechanics controlling the mass of nodes, one that builds them up, and one that tears them down. With this in mind, let's see what kinds of direct coordinates and masses the hyperSOM will move towards. If we start with the coordinates they will be the center of mass of multiple samples. The samples for $\mathbf{x}_i$ can come from two sources:

1. Successful attempts to reach the state s_i. $\mathbf{x}' = \mathbf{x}_i + r_i\mathbf{\Delta x}$.
2. Unsuccessful attempts to reach other states. $\mathbf{x}' = \mathbf{x}_j + r_j\mathbf{\Delta x}$, $j \neq i$, that ended in state s_i by accident.

This means that the different direct coordinates will be pulled towards each other, fulfilling the first criteria under "Method". Secondly, failed attempts to reach a state with a coordinate $\mathbf{x}_i$ will decay the mass, making the coordinate more likely to move to new areas. This means that the more previous states that can safely be transitioned to this state with some coordinate $\mathbf{x}_i$, the less will the mass of that choice decay, fulfilling the second criteria. Finally a coordinate centered in a set will allow a larger radius without risking failure, meaning that the coordinates will move towards the center of the part of the set where it is, fulfilling the third criteria. Thus, the assumption is that following these rules of decay and building up will lead to "good" direct coordinates.
With this in mind, let's look at what masses nodes will move toward when taking both the build up and the decay in mind. If ρ is the probability that a transition to a state will be a failure, and one gets $m' \in [0, 1]$ on average in mass for every time it does succeed, then equilibrium for a mass is reached when the build up equals the decay rate. That is, we look for a mass m, that when updated on average stays the same. There is a $1 - \rho$ probability that the next mass will be $m + m'$, and a ρ probability that the next mass will be λm. Therefore the mass won't increase or decrease on average if,

$$m = (1 - \rho)(m + m') + \rho\lambda m \tag{2.19}$$

giving

$$m \to \frac{1 - \rho}{(1 - \lambda)\rho} m' \tag{2.20}$$

Examples of these masses as functions of the probabilities can be seen in Figure 2.3. These plots can be useful when choosing a λ-value.

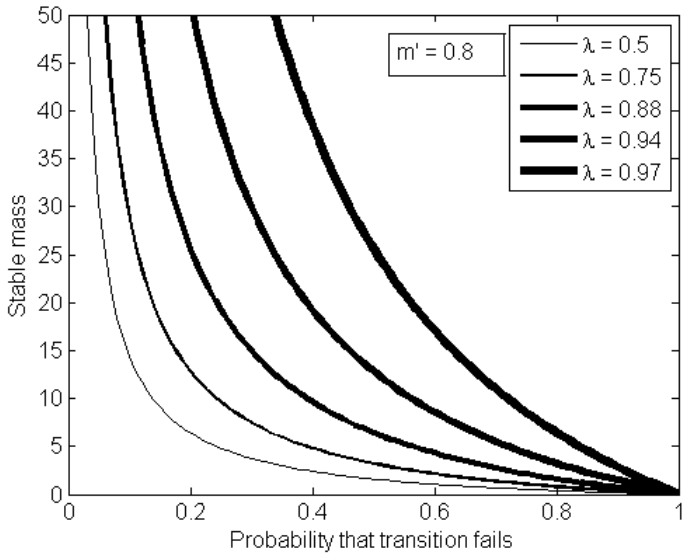


Figure 2.3: What value the mass of a node would converge to, depending on the precision with which the state can be reached. Or in this case, the probability that a transition to the state would fail. In this case it is assumed that the mass of new data-points is 0.8 on average.

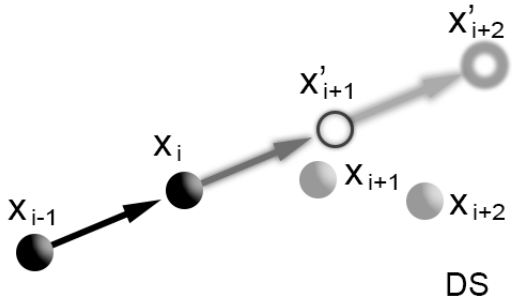


Figure 2.4: A simplification of how the coordinates of the so far unknown states $i+1$ and $i+2$ can be estimated, on basis of the relation of the two known states $i-1$ and i using linearization. $\mathbf{x_k}$ denotes a "good" direct state that belongs to the set Z_k and $\mathbf{x'_k}$ is an approximation, thought to belong to the set Z_k. Observe that the quality of the guesses decreases the further away one gets from the known data.

Generalization to unvisited states

A final feature of the hyperSOM is that it can generalize the direct coordinates of the states it has visited previously, to states for which it lacks data so far, and thus don't know where to find in DS. This allows the agent to find new states more efficiently in the cases where the relation between neighboring states are somewhat linear. As an example: If pushing an object with a little bit of force moves the object by a little bit, then a fair first guess for how to move the object by a lot would be to push the object with even more force, instead of just assume that the needed action could be anything since one has never observed the object to move by a lot previously.
The approximations of the hyperSOM are performed as follows:

1. Receive a new data point $\mathbf{x}'$ of mass m', and conclude what node/state it belongs to. Let's call this state i. We are thus looking at the node with the relative position $\mathbf{k}_i = [k_{i,1}\ k_{i,2}\ \dots\ k_{i,D}]$ in the grid.

2. What we want to do is to create approximate coordinates of all nodes on basis of the vicinity of the new data point $\mathbf{x}'$ and its closest neighbors. To do this we look for vectors $\{\mathbf{v}_h\}_{h=1}^{D}$ with one vector per dimension to span the grid. In a two dimensional case this means that a node j would get the direct coordinate:
$$\mathbf{x}'_j = (k_{j,1} - k_{i,1})\mathbf{v}_1 + (k_{j,2} - k_{i,2})\mathbf{v}_2 \tag{2.21}$$
To find these vectors we look at the adjacent nodes in the grid. If $\mathbf{x}_+$ and $\mathbf{x}_-$ are the closest neighbors in some dimension h, then
$$\begin{aligned} \mathbf{v}_+ &= \mathbf{x}_+ - \mathbf{x}' \\ \mathbf{v}_- &= \mathbf{x}' - \mathbf{x}_- \end{aligned} \tag{2.22}$$
Only one vector per dimension is however needed. To reduce them to one we observe that $\mathbf{x}_+$ and $\mathbf{x}_-$ may not be equally significant. It could be the case that we have robust data for one of them while the other is just an approximation. This relation would be reflected by the mass, and thus we can set
$$\mathbf{v} = \frac{m_+\mathbf{v}_+ + m_-\mathbf{v}_-}{m_+ + m_-} \tag{2.23}$$
So if $m_- \ll m_+$ then $\mathbf{v} \approx \mathbf{v}_+$, but if $m_- = m_+$ then $\mathbf{v}$ is the mean of the two.

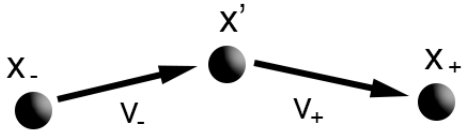


Figure 2.5: How $\mathbf{v}_+$ and $\mathbf{v}_-$ is related to $\mathbf{x}'$, $\mathbf{x}_-$ and $\mathbf{x}_+$.

3. Create an approximation for every node, so that
$$\mathbf{x}'_j = \mathbf{x}'_i + \sum_{h=1}^{D} \Delta k_h \mathbf{v}_h \tag{2.24}$$
where
$$\Delta k_h = k_{h,j} - k_{h,i} \tag{2.25}$$
that is, how far away node j is from node i in dimension h. Give every approximation a mass
$$m'_j = m' \mathrm{e}^{-\xi \|\mathbf{\Delta k}\|} \tag{2.26}$$
where
$$\|\mathbf{\Delta k}\| = [\Delta k_1\ \Delta k_2\ \dots\ \Delta k_D] \tag{2.27}$$
and $\xi \approx \ln(100)$.

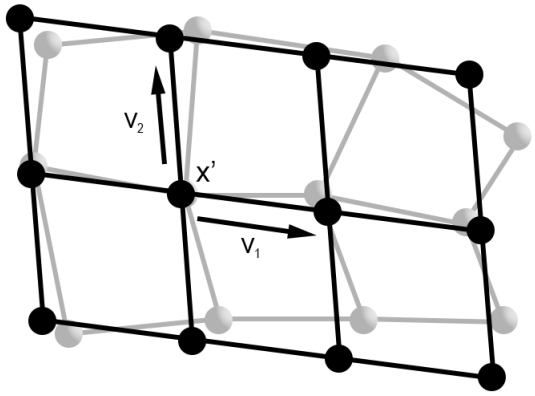


Figure 2.6: How hyperSOM predicts the positions of the different nodes based on the vicinity of $\mathbf{x}'$.

4. Update every node j with its approximation $(\mathbf{x}'_j, m'_j)$. Notice that $(\mathbf{x}'_i, m'_i) = (\mathbf{x}', m')$. Since $\xi \approx \ln(100)$, the mass of the approximations $j \neq i$ are very small which means that nodes with solid data will hardly move at all, while the so far unseen nodes will be almost entirely influenced by the closest nodes for which data exists. Even though the search radius around the unseen nodes then becomes quite large, the presence of the random weight w yet makes the agent look close to the approximation in a lot of cases.

What is basically being done is that whenever a coordinate $\mathbf{x}'$ is found a linearization is made on basis of the nodes closest neighbors which is extended for the full hyperSOM, but with an exponentially decreasing mass the further away from the known state one gets. An example of this mechanism in action can be seen in Figure 2.7.

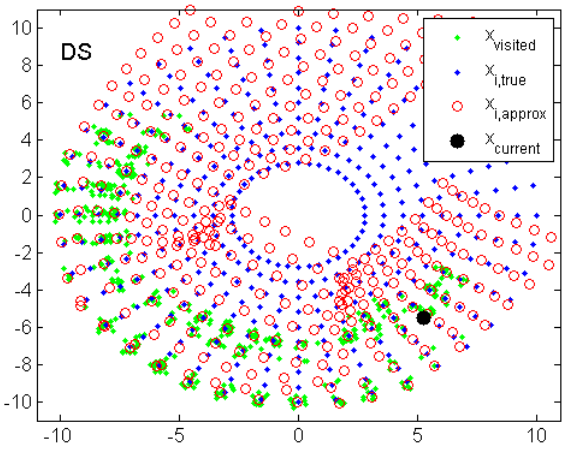


Figure 2.7: An example of the hyperSOM at work, using "Generalization for unvisited states". The goal of this hyperSOM is to place the circles $x_{i,approx}$ over the blue dots $x_{i,true}$, using the data found $x_{visited}$. In other words, green spots tells what regions the agent has been visiting, and the red rings are the agents estimates of where the blue dots are. Observe that if the blue dots are at least locally continuously spread out the agent is often able to make good guesses of the positions of the closest unvisited blue dots based on the positions of the visited ones.

2.2.3 The algorithm

The full algorithm may be found in Algorithm 1. The functions used by the algorithm are:

ReadDirectCoordinates()	Returns the current direct coordinate $\mathbf{x}$ of the agent.
ReadGlobalCoordinates()	Returns the current global coordinate $\mathbf{g}$ of the agent.
GetGoalState(i)	Returns the next state to try to get to, given current state i.
GetTarget($\mathbf{x}, m$)	See Algorithm 3.
GoTo($\mathbf{x}$)	Tells the agent to move into given direct coordinate $\mathbf{x}$.

Algorithm 1 HyperSOM

DEF: $K = \{\mathbf{k}_i\}_{i=1}^{N}$, so that $K(i) = \mathbf{k}_i$ ▷ The relative positions in the grid
DEF: $G = \{\mathbf{g}_i\}_{i=1}^{N}$, so that $G(i) = \mathbf{g}_i$ ▷ The global coordinates
DEF: $X = \{\mathbf{x}_i\}_{i=1}^{N}$, so that $X(i) = \mathbf{x}_i$ ▷ The direct coordinates
DEF: $M = \{m_i\}_{i=1}^{N}$, so that $M(i) = m_i$ ▷ The nodes' masses

Initiate: G as an orthogonal grid in GS
Initiate: $X(i) = \mathbf{0}$, and $M(i) = 0$, for all nodes i
Set: $\lambda \in [0, 1]$ ▷ Mass decay for failing
Set: $\xi \approx \ln(100)$ ▷ Generalization decay. $\ln(a)$ adds k:th closest neigbour mass $m\frac{1}{a^k}$
Set: $b_{gen} \in \{True, False\}]$ ▷ Wheather to make generalizations or not.

while running **do**
    $\mathbf{x}' \leftarrow$ ReadDirectCoordinates()
    $\mathbf{g}' \leftarrow$ ReadGlobalCoordinates()
    $i_1 \leftarrow \arg\min_i \{||G(i) - \mathbf{g}'||\}$ ▷ Get current state
    $i_2 \leftarrow \arg\min_{i \neq i_1} \{||G(i) - \mathbf{g}'||\}$ ▷ Get second closest state

$$m' \leftarrow 1 - \frac{\|G(i_1) - \mathbf{g}'\|}{\|G(i_2) - \mathbf{g}'\|}$$

    if i_g is defined **then**
        $[X, M] \leftarrow$ Update($\mathbf{x}', m', i_1, i_g, X, M, K, \lambda, \xi, b_{gen}$)

    $i_g \leftarrow$ GetGoalState(i_1)
    $\mathbf{x}_t \leftarrow$ GetTarget($X(i_g), M(i_g)$)
    GoTo($\mathbf{x}_t$)

Algorithm 2 Update($\mathbf{x}, m, i_c, i_g, X, M, K, \lambda, \xi, b_{gen}$)

if $i_c \neq i_g$ **then**
    $M(i_g) \leftarrow \lambda M(i_g)$

if not b_{gen} **then**
    $X(i_c) \leftarrow \dfrac{M(i_c)X(i_c) + m\mathbf{x}}{M(i_c) + m}$
    $M(i_c) \leftarrow M(i_c) + m$

else
    for every dimension d **do**
        $\mathbf{k}_{\pm} \leftarrow K(i_c) \pm \widehat{e}_d$ ▷ Get grid positions of adjecent nodes
        $i_{\pm} \leftarrow \text{GetState}(\mathbf{k}_{\pm})$

        $\mathbf{v}_+ \leftarrow X(i_+) - \mathbf{x}, m,$
        $\mathbf{v}_- \leftarrow \mathbf{x}, m, -X(i_-)$

        $\mathbf{v}_d \leftarrow \dfrac{M(i_+)\mathbf{v}_+ + M(i_-)\mathbf{v}_-}{M(i_+) + M(i_-)}$

    for all states i **do**
        $\Delta\mathbf{k} \leftarrow K(i) - K(i_c)$ ▷ Distance vector from measured state

        $\mathbf{x}' \leftarrow \mathbf{x} + [\mathbf{v}_1\ \mathbf{v}_2\ ...]\Delta\mathbf{k}$
        $m' \leftarrow m\text{e}^{-\xi\|\Delta\mathbf{k}\|}$

        $[X, M] \leftarrow \text{Update}(\mathbf{x}', m', i, i, X, M, K, \lambda, \xi, False)$

return X, M

Algorithm 3 GetTarget($\mathbf{x}, m$)

$r \leftarrow \dfrac{1}{1+m}$

$s \leftarrow ru$ ▷ $u \in \mathcal{U}(0,1)$ is a uniform random number in the span $[0,1]$
$\mathbf{\Delta x} \leftarrow [n_1\ n_2\ n_3\ \ldots]^T$ ▷ $n_i \in \mathcal{N}(0,1)$ is a gaussian random number with mean 0 and variance 1

$\mathbf{x} \leftarrow \mathbf{x} + s\mathbf{\Delta x}$
for all elements $x_i \in \mathbf{x}$ **do**
    $x_i \leftarrow \max(x_i, -1)$
    $x_i \leftarrow \min(x_i, +1)$

return $\mathbf{x}$

2.3 Derivation of Coordinates

2.3.1 Direct Coordinates

The direct coordinates of the robot is composed of the servo settings. This means that any direct coordinate $\mathbf{x}$ corresponds to a distinct body posture of the robot. There are three properties of the servos' original settings that makes them somewhat awkward to work with:

1. Their position-span of $[0, 1023]$.
2. The fact that they don't obey left-right symmetry. For an opposite servo to mirror a servo of setting x it needs a setting $1023 - x$. To realize this think that you rotate both your arms clockwise. Then they will move in opposite directions.
3. The original position-span often allows positions that are physically impossible or potentially harmful to reach. Especially there is a risk of damaging cables by stretching them to far or getting them stuck somewhere.

To treat these problems the following preprocessing is done:

1. Right-side servo settings are redefined: $x \leftarrow 1023 - x$.
2. Define a x_{min} and a x_{max} for every symmetric pair of servos, so that all values $x \in [x_{min}, x_{max}]$ are relatively safe.
3. Rescale every x proportionally so that $[x_{min}, x_{max}] \rightarrow [-1, 1]$.

By doing this every direct coordinate $\mathbf{x}$ becomes a vector of elements in the range $[-1, 1]$.

2.3.2 Global Coordinates

There are three global coordinates:

- Two expressing the tilt of the robot in two dimensions.
- One optional expressing how "stretched out" the robot is.

The tilt parameters are given by an accelerometer, and the third is calculated from the servo-readings.

The tilt parameters

The tilt parameters are given by a 2D accelerometer, expressing the gravitational pull in two different directions, in absence of other forces. They are preprocessed in two steps, first while continuously read by the *Arduino*, and secondly by *Matlab* when used by the agent, which is done at a slower pace. The original values sent to *Arduino* are integers where the value 1000 roughly corresponds to $1g$. Absent of other forces, we have

$$a_x^2 + a_y^2 + a_z^2 = g^2 \tag{2.28}$$

Only two of these accelerations are measured directly by the accelerometer since it is 2D, but if a_x and a_y are the directions measured then the value of a_z can be deduced from this expression with the only exception to its sign. This is however only in theory. In reality there's a lot of noise involved, both due to the precision of the sensor, but also due to other forces than gravity affecting the outcome. In *Arduino* the following steps are performed when processing the data:

1. For every new value, take a number of measurements (~ 4) and only accept the measurement closest to the last accepted measurement. This filters out sudden reading peaks caused by impulses acting on the agent. They can be very high in magnitude, but are also very short in time.
2. Rescale the data to span $[-1, 1]$.

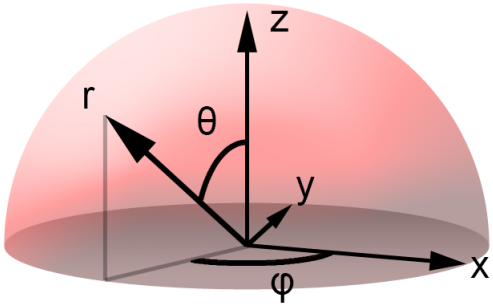


Figure 2.8: In rest and absent of other forces is the robot constantly subjected to a total acceleration of $1g$ pointed towards the center of the Earth. If the accelerometer measures the acceleration within the xy-plane in a coordinate system relative to the robot, and the accelerations in the z direction is assumed to be positive, then every measured tilt can be put on the surface of a half-sphere of radius $1g$. These positions can be unambiguously defined either in their projections on the xy-plane (since its a half-sphere), or in the spherical coordinates φ, θ.

Since the *Arduino* will read values at a much higher pace than *Matlab*, there will always be a stack of old values each time *Matlab* require a reading. These older values can be used to create a mean for the current value, but it's wise to let the newer values have a bigger influence since the agent might have actually moved in the time when the stack was collected. One way to do this is to use a weighted mean, where the simplest form is:

$$v_{i+1} = \lambda v_i + (1 - \lambda) s_{i+1} \tag{2.29}$$

where v_n is the weighted mean of the n oldest samples $\{s_i\}_{i=1}^{n}$ and $\lambda \in [0, 1]$ is the decay parameter with which old data loses its significance for the final result. Thus the k:th *newest* sample in the stack get the final weight:

$$w_k = (1 - \lambda)\lambda^{k-1} \tag{2.30}$$

Every value is now created as a mean in the span $[-1, 1]$ with sudden fluctuations filtered out. This could be sufficient, however we will also try out the option to change the values into spherical coordinates.

Spherical coordinates

Considering equation 2.28 we see that the system has a constant radius which effectively reduces the 3 dimensional system into a 2 dimensional one, and a spherical geometry perfectly captures that, see Figure 2.8. The transformation from cartesian coordinates (x, y, z) to spherical coordinates (r, θ, φ) with $g = 1$ is:

$$\begin{aligned} r &= 1 \\ \theta &= \cos^{-1} z \\ \varphi &= \tan^{-1} \tfrac{y}{x} \end{aligned} \tag{2.31}$$

In the case of φ it is important to take the correct quadrant into account. This is practically done by using the $atan2(y, x)$ - function. In the case of θ we have another problem. We don't know the actual value of z. Assuming 2.28 we know that:

$$z = \pm\sqrt{1 - x^2 - y^2} \tag{2.32}$$

but whether the sign is positive or negative is impossible to tell. For this work we will simply define all z to be positive, but it's easy to see that if we could separate them, then a spherical representation would be the obvious choice since the cartesian representation, which is only the projection of r onto the x, y-plane, could never be able to differentiate between positive or negative z.
By doing this transformation we now have a global position expressed in (θ, φ), with $\theta \in [0, \pi/2]$ (since $z \geq 0$) and $\varphi \in [0, 2\pi]$. We want to eventually change those ranges into $[-1, 1]$, but before we do there is one last issue to address. The idea is to divide the θ, φ-space into a grid of states. If we however do this in discrete steps $\Delta\theta, \Delta\varphi$ this would mean, considering Figure 2.8, that the states closer to the "north pole" of the sphere will

cover a much smaller solid angle/area of the sphere, than those at the equator. The sphere cap solid angle from the north-pole to θ is

$$A(\theta) = 2\pi(1 - \cos\theta) \tag{2.33}$$

If we have n_θ states in the θ-direction then

$$\Delta\theta = \frac{\pi}{2n_\theta} \tag{2.34}$$

This means that the states will be divided into rings of constant width θ. Ring k is delimited by the span $[\theta_{k-1}, \theta_k]$ where

$$\theta_k = k\Delta\theta \tag{2.35}$$

With this we can conclude the solid angle Ω_k of the k:th ring is

$$\begin{aligned} \Omega_k &= A(\theta_k) - A(\theta_{k-1}) \\ \Omega_k &= 2\pi(\cos\left[(k-1)\Delta\theta\right] - \cos\left[k\Delta\theta\right]) \end{aligned} \tag{2.36}$$

The ring of largest solid angle will thus be the one at the equator, with Ω_{n_θ}. If we want n_φ states in the φ -direction then that would mean that the solid angle of a state in this ring is

$$\Delta\Omega = \frac{\Omega_{n_\theta}}{n_\varphi} \tag{2.37}$$

This is the trick: In order to keep the solid angle of all states more or less constant, we must allow fewer states in the other rings. Explicitly the numbers of states in ring k becomes

$$n_{\varphi,k} = \text{Round}\left[\frac{\Omega_k}{\Delta\Omega}\right] \tag{2.38}$$

$$n_{\varphi,k} = \text{Round}\left[\frac{\Omega_k}{\Omega_{n_\theta}} n_\varphi\right] \tag{2.39}$$

This is practically implemented in the following steps:

1. Create a grid $n_\theta \times n_\varphi$. This grid will have enough states for the model, even though some of the states won't be used. Explicitly there will be N states:

$$N = \sum_k^{n_\varphi} n_{\varphi,k} \tag{2.40}$$

2. Given a measured θ we get that

$$k = \text{Ceil}\left[\frac{2\theta}{\pi} n_\theta\right] \tag{2.41}$$

3. Rescale φ to φ' so that only the $n_{\varphi,k}$ first states of the row can be reached. Thus:

$$\varphi' = \frac{n_{\varphi,k}}{n_\varphi}\varphi \tag{2.42}$$

This way only the states (in the grid) belonging to the model can be visited. Figure 2.9 shows what this mapping looks like. In a final step $[\varphi', \theta]$ are assigned to g_1, g_2 so that

$$\begin{aligned} g_1 &= 1 - \frac{4\theta}{\pi} \\ g_2 &= \frac{\varphi'}{\pi} - 1 \end{aligned} \tag{2.43}$$

and $g_1, g_2 \in [-1, 1]$.

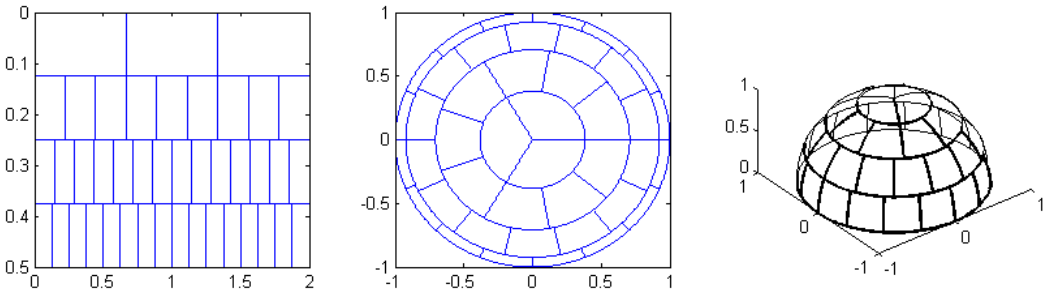


Figure 2.9: How the global space of the robot can be divided using spherical coordinates, if one priorities that the states should be as evenly sized as possible. Leftmost we see how many states there will be for every segment of the sphere, in the middle what this would look like in the xy-plane and to the right how the states are distributed in 3D.

The "stretched-out" parameter

The stretched out parameter is a somewhat constructed one, made solely to allow a humanoid robot to have states only reachable if standing up. A better measure would be a measure of the accelerometers height above the ground, but since such a measure would be to hard to implement the idea is that a "stretched out" parameter could do an equally good job, even though it will be specially designed for this specific robot, and is thus not applicable for other morphologies. The reasoning goes that if the agent have states that are only visitable with stretched out legs, then the only way to reach a state with $\theta = 0$, is to be standing. Without this measure the robot wouldn't be able to separate between a sitting pose and a standing one.

The parameter values are constructed so that two direct coordinates are taken from the robot, one where the legs are completely stretched out, $\mathbf{y}_1$, and one where the robot has its legs the most bundled up, $\mathbf{y}_0$. These vectors $\mathbf{y}_1, \mathbf{y}_0$ are not in the full DS, but only consider the servos in the legs and the hips of the robot so that arm-settings does not count. A new vector $\mathbf{y}$ is then constructed so that each element is in the range $[-1, 1]$, where 1 means that the current setting of the robot is exactly the same as in $\mathbf{y}_1$, and -1 means that it is as far or more from the element in $\mathbf{y}_1$, as the element in $\mathbf{y}_0$ is. This means that there often will be multiple ways to score -1, but only one to score 1. Finally the last "stretched-out" parameter g_3 is constructed so that:

$$g_3 = \text{mean}(\mathbf{y}) \tag{2.44}$$

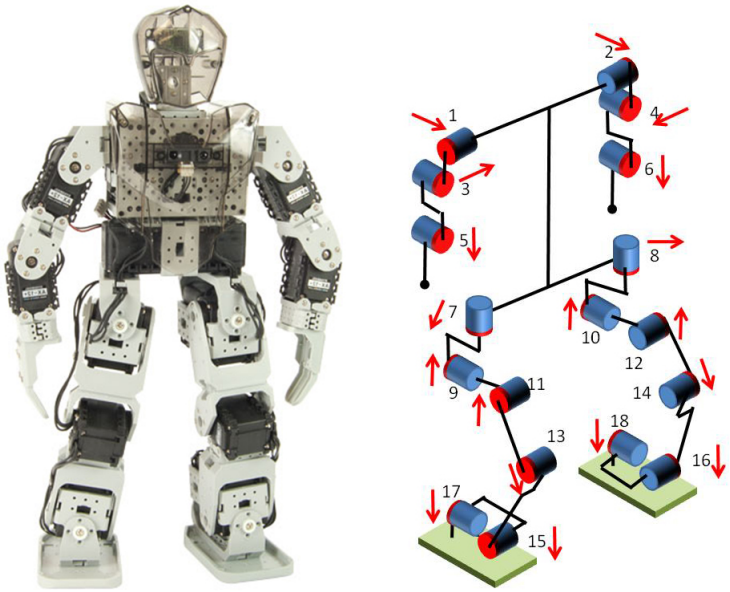


Figure 3.1: A Bioloid Premium and the layout of its servos.

3 Method

3.1 Hardware

3.1.1 Components

For this work a Bioloid Premium [12] is used, together with a Memsic Dual-Axis Accelerometer [9]. The Bioloid is in its design very general and can be assembled in a multitude of ways. In principle the method presented in this work should be able to cope with any shape, but in this work will the standard humanoid form be used, as seen to the left in Figure 3.1. This shape is as new to the agent as any other shape would be, so there is really no reason not to use it also considering that the humanoid form is more interesting to us as humans. Many things in the Bioloid Premium set, such as the battery, IR-sensors, a gyro and a distance sensor, are not used in this work. The only thing that will be used from the original Bioloid platform is its 18 servos. An accelerometer from Parallax is however added and positioned within the chest of the robot, in order to measure the tilt of the robot.

Servos

The servos consists of 18 Dynamixel AX-12A, with a layout as seen in Figure 3.1 to the right. Some features of the servos are:

- ID - Every servo is given an ID number $\in [0, 252]$.
- Max Torque - The maximum output torque $\in [0, 1023]$.
- Present Position - The current angle of the servo $\in [0, 1023]$.
- Goal Position - The angle the servo should move to $\in [0, 1023]$.
- Moving Speed - How fast the servo should move to the Goal Position $\in [0, 1023]$.

There are more features which can all be found at [12], but these are the most important for this work.

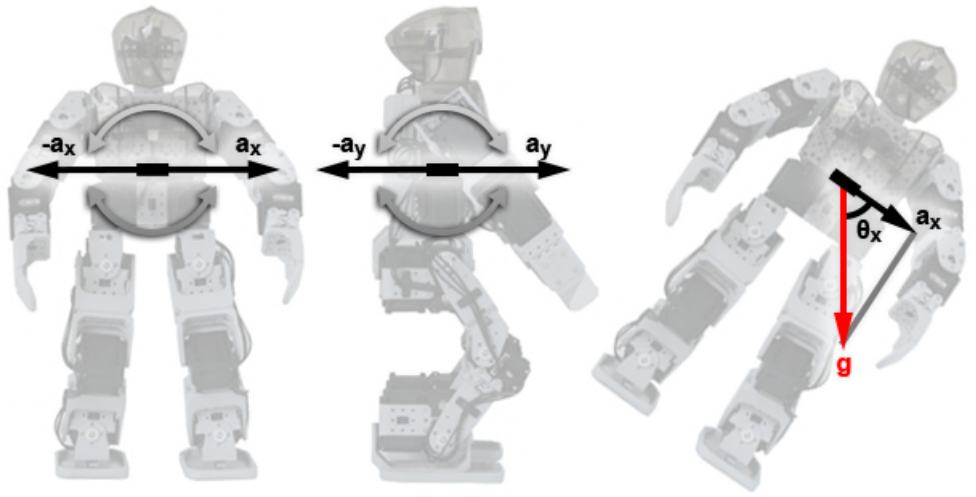


Figure 3.2: The relation between the accelerometer readings and the tilt of the robot.

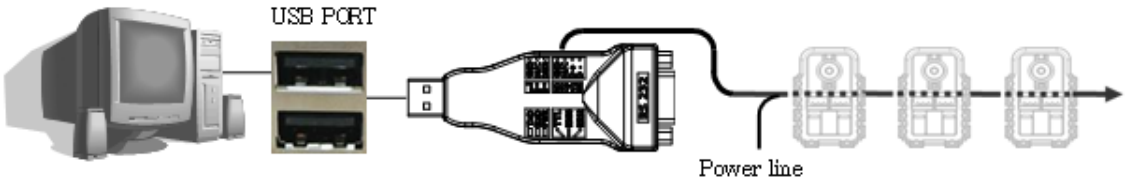


Figure 3.3: The connection between the servos and the computer controlling them.

Accelerometer

To measure the tilt of the robot a Memsic Dual-Axis Accelerometer [9] is positioned within the chest of the robot, according to Figure 3.2. It measures the acceleration in two directions x and y. It is positioned so that no acceleration is measured at all when the chest is upright, acceleration is measured in the x-direction when the robot is tilted sideways, within the frontal plane (leftmost), and acceleration is measured in the in the y-direction when robot is tilted forward-backward, within the sagittal plane (center). That means that if the angles towards the direction of gravity are $[\theta_x, \theta_y]$, then the corresponding forces are:

$$\begin{aligned} a_x &= g\cos(\theta_x) \\ a_y &= g\cos(\theta_y) \end{aligned} \tag{3.1}$$

if no other forces are acting on the robot. The accelerometer then returns, for every direction, an integer acceleration value $\in [-3000, 3000]$ corresponding to $[-3g, 3g]$, but in practice would the real span be $[-g, g]$. It could nevertheless be good to look for forces larger than g, since that could indicate a hit, like when falling and hitting the ground.

3.1.2 Connection

Servos

The servos can be connected directly to each other using Daisy Chains, which means that not all servos needs to be directly connected to the controller. As long as one servo in a chain is connected then every servo is. The controller may then send and receive data from any servo by using that servos unique ID number. The controller in this case is a computer, which is connected to the servos through a USB-device *USB2Dynamixel* [12]. This set-up can be seen in Figure 3.3.

Accelerometer

The accelerometer have four pins that needs to be connected to the computer. Two signal-pins for the the accelerations, one for voltage and one for ground. These are connected by cable to an Arduino Uno, outside of the robot, where the signals are preprocessed before sent to a computer by USB.

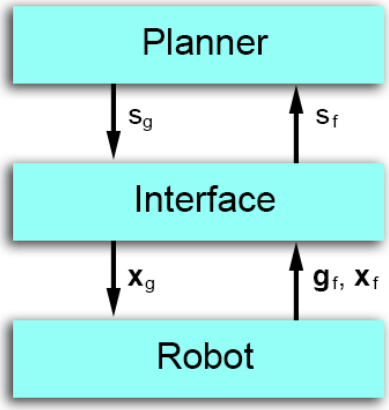
Power

The servos needs 12V for optimal performance, so to guarantee this power is supplied to the robot directly from the power-grid through an adapter of 12V, 5A.

3.2 Software

In its simplest form this is a solution in three layers. The three layers consists of the *Planner*, the *Interface* and the *Robot*. It basically works like this:

1. The Planner decides a new state s_g to try to go to, given current state s_f.
2. Given the goal state s_g, the Interface suggest a direct coordinate $\mathbf{x}_g$ that should move the robot to the wanted state.
3. The Robot tries to move to the direct coordinate $\mathbf{x}_g$. When that is done the robot returns the real direct coordinate $\mathbf{x}_f$ that it managed to reach together with the measured global coordinate $\mathbf{g}_f$ to the Interface.
4. The Interface concludes what state corresponds to the global coordinate. It uses this information together with the real direct coordinate to improve its future guesses. It returns the new state s_f to the planner.



The planner was in this implementation set to a AC-driven RL, and as an interface was a hyperSOM used.

3.2.1 Parameters

Before going into the exact procedure lets take a moment and review all parameters involved:

Name	Type	Explanation
I	Integer	The number of iterations for training.
$[n_1,\ n_2,\ n_3]$	Integers	The size of the grid, where n_i is the number of layers in the g_i-direction. The two parameter choices indicates cartesian/spherical coordinates.
N	Integer	The number of reachable states. In the cartesian case this means that states that are completely outside of the unit-circle are not counted, if such states exist.
r_a	Integer	The action radius. The agent is not allowed to attempt to reach a state that is further away than this number, in any dimension. In other words: The max norm needs to be smaller than this number. In the spherical case the longitudinal distance is measured at the longitude closest to the pole of the sphere.
M	Integer	Total number of state-actions, or in other words, the total number of transition between states that are allowed by r_a.
$max_{T_S},\ max_{T_{SA}}$	Integers	Maximum sum of recorded transitions. After limit is breached renormalization is performed to these numbers.
t_{move}, t_{rest}	Doubles	Time for the agent to reach a goal-coordinate, and time for the agent to settle down at a new state in order to stabilize before taking a measurement.
λ	Double	The mass decay rate of a state when the agent fails to reach that state.
ξ	Double	SOM decay ln(100)
γ	Double	RL-importance of future reward (=0.9)
spherical	Boolean	Whether or not to use spherical coordinates.

The values only reflects the values chosen for the tests. It might not be the best possible parameter choices, but that is not the point of this work. The numbers where chosen because they work, and since this is more of

a *proof of concept* that is sufficient. When more than one value is given comparisons will be made between the different choices. For further details, read Discussion, section6.2.

3.2.2 Initialization

Nodes

Since the grid is 3-dimensional there are

$$N = n_1 \times n_2 \times n_3 \tag{3.2}$$

nodes (and states) in total. In the cases where the third global parameter is not used (and the grid is 2D) n_3 is set to 1. Every node is assigned a position $\mathbf{k} = \{k_1, k_2, k_3\}$, $k_i \in \{1, 2, \ldots, n_i\}$, within the node-grid, and that position corresponds to the state s_i so that

$$s_i = k_1 + (k_2 - 1)n_1 + (k_3 - 1)n_1 n_2 \tag{3.3}$$

Furthermore there are

$$M = N \times N \tag{3.4}$$

possible *state-actions* in total, that is the numbers of possible state-action combinations since every state can potentially be tried to be reached from any state (including itself). If a state-action $(sa)_i$ is defined to be an attempt to go from a state s (state), to a state s' (action), then

$$(sa)_i = s + (s' - 1)N \tag{3.5}$$

When placing the nodes in GS the grid is basically just rescaled but otherwise kept intact, so that all nodes of a fixed layer k_i are also in the same layer g_i. Since we want all nodes to cover the same volume in GS and $g_i \in [-1, 1]$ we get:

$$\mathbf{k} = \{k_1, k_2, k_3\} \;\Rightarrow\; \begin{cases} \mathbf{g} = \{g_1, g_2, g_3\} \\ g_i = \dfrac{2k_i - n_i - 1}{n_i} \end{cases} \tag{3.6}$$

The direct coordinates and the mass on the other hand are to be determined by time, and are initiated as

$$\begin{matrix} \mathbf{x}_i = \mathbf{0} \\ m_i = 0 \end{matrix}, \; \forall \; i \tag{3.7}$$

so that the initial search of the agent is as general as possible.

Markov map

In order for the RL to initially be able to attempt actions, it needs a record of earlier tried ones since it is only allowed to try state-actions that have previously been successful, see section for details. In order to initialize the Markov map matrices one failed and one successful transition is therefore added from every state to itself and everyone of it's neighbors. That is, if i and j are two different nodes, and

$$\|\mathbf{k}_i - \mathbf{k}_j\| \leq 1 \tag{3.8}$$

then a transition between state i and j in both directions should be observed. In the case of spherical coordinates note that the φ-boundaries are cyclic.

3.2.3 Monitor tools

In order to get an overview of what the agent is doing and thinking, two maps are used, a mass map and a V_S-map. Figure 3.4 show what they might look like during a run, and Figure 3.5 show what areas of the map corresponds to what global coordinates of the agent in both a cartesian and a spherical coordinate system.

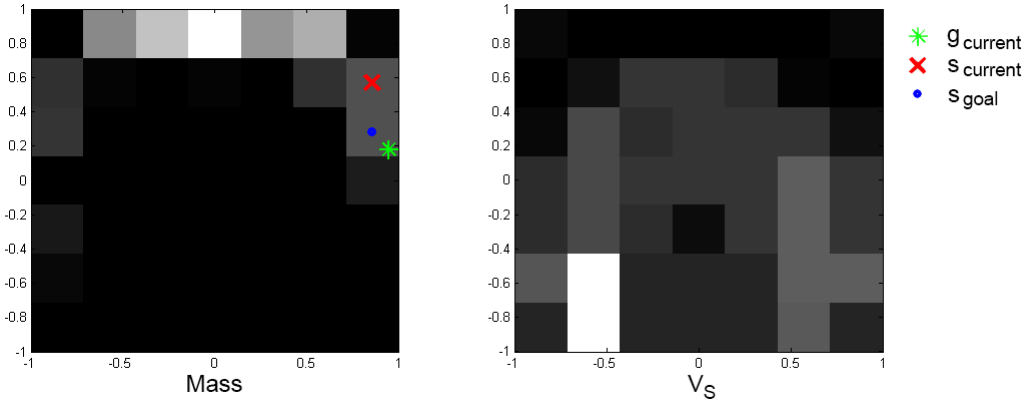


Figure 3.4: The mass-map and V_S-during training. Lighter colors indicate higher number which means greater mass in the left case, and more interesting states, in the right case.

Mass map

A mass map displays GS divided in all its states, where every state is given a color depending of the mass of that state. In addition three cursors can be added:

- One displaying the true global coordinate $\mathbf{g(t)}$ of the agent, updating in real-time.
- One marking the last visited state. (state)
- One marking the aimed new state. (action)

This gives a powerful oversight of the learning process in real-time, for GS of dimension ≤ 3. The mass map first of all shows what areas of the state-space that the agent has so far visited, since unvisited states have mass 0, but it also gives information of its certainty of different states since it needs good coordinates $\mathbf{x_i}$ for a state's mass to increase. Finally it also gives a hint of what the RL wants to do, since the independent actions also becomes visible. Overall this is a good way to monitor in real time what the robot is doing and where it is in it's learning process.

V_S-map

As explained in section 3.2.2 the V_S matrix is computed to assign an expected future reward if reaching a state. Similar to the mass-chart in layout this map instead displays how interested the agent is in different states. Observe however that it is not possible to predict what actions an agent will attempt just from watching this map, since the agent also accounts for the probability of actually getting there.

3.2.4 Data to collect

When running a learning sequence there are a couple of things that can be measured in order to evaluate the learning progress. In this section follows some that will be used throughout the results.

Probability of transition failure

The probability matrices acquired by the RL can't be used directly because of the initiation that says that any transition between neighboring states have succeed once and failed once. This would mean that unseen states would automatically be assumed to have a $50-50\%$ chance of being reached, which is most certainly not the case. Another way to approximate the probabilities is to use the masses of the hyperSOM. The relation between the mass of a state and the probability to reach that state successfully was examined in section 2.2.2. Using equation 2.20 we may conclude that

$$\frac{m'}{(1-\lambda)m+m'} \to \rho \tag{3.9}$$

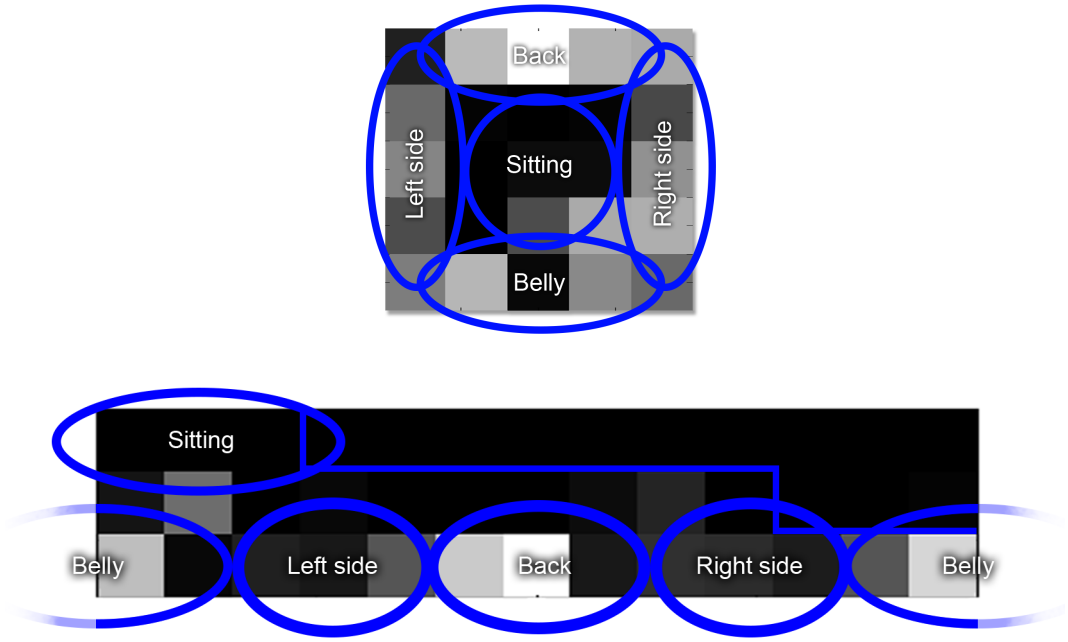


Figure 3.5: How the different states of the mass-maps are related to the postures of the robot.

as the masses converges, where ρ is the probability of a transmission failure. Note that a state without mass will now be assumed to have a 100% chance of failure. The exact value of m' is not that important since the probabilities will only be compared to each other, but in order to have some value the assumption $m' = 0.6$ is made.

KL chart

To measure the information-gain of the RL can the KL-rewards can be measured. Since this is the actual information gain regarding the transition-probabilities it would perfectly suited for the task.

Found states and state-actions

By watching how many states an agent have visited or tried out gives a clue of how efficiently it explores its state-space and state-action-space.

3.2.5 Other tests

The go-to-command

In order to be able to test what an agent has learned a feature is added to allow a person to stop the agents curiosity-driven search and instead give it a specific state to reach. When the curiosity driven learning is turned off the hyperSOM is also frozen, so that its capabilities can be checked without it training on the tasks for which it is checked. The KL-based reward matrix is then switched to a more classical RL reward function where the only reward is to arrive at the wanted state. The agent would then strictly use the policy of the highest Q-values which would bring it to the wanted state in the most efficient way possible, on basis of the agent's probability function P_{SA}. This is an excellent tool to use when one wants to utilize what the agent has learned, and as a such it could be used in order to collect information of what kinds of task the agent has come to master. Such standard tests are

- Go between the back and the left side.
- Go between the back and the right side.

- Go between the back and the belly.
- Roll around one one revolution, cw or ccw, without sitting up.
- Go from the back to a sitting pose.
- Go from the back to a standing pose. (if $n_3 > 1$).

Using these tests an average feeling of the effectiveness of a specific setting should be given.

3.2.6 Execution

Procedure of learning

The approach while running is pretty straightforward:

1. Get current coordinates $(\mathbf{x}, \mathbf{g})$.
2. Train hyperSOM with the coordinates.
3. Conclude current state s_c based on $\mathbf{g}$.
4. (Not first turn) Train RL on the fact that s_c was the outcome of being in previous state s_p trying to reach the goal-state s_g.
5. (Not first turn) If transition $s_p \rightarrow s_g$ failed, let the mass of s_g decay so that $m_g = \lambda m_g$.
6. Ask RL for the next state s_g to attempt to reach.
7. Get a goal-coordinate $\mathbf{x}_g$ from hyperSOM in order to reach s_g.
8. Tell the agent to move to $\mathbf{x}_g$.
9. Let it move for t_{move} seconds.
10. Tell it to hold current position.
11. Wait for t_{rest} seconds to let the robot stabilize.
12. Go to 1.

Set up of experiments

The different experiments have all somewhat different settings but the overall procedure is apart from that constant.

1. The robot starts lying on it's back, with no previous knowledge at all.
2. The robot is then allowed to train freely, a number of iterations defined in the experiment. During this training a human assistant may only intervene in order to remove obstacles (such as cables) that the robot is at risk of interacting with, or gently push the robot back to its assigned training area if it's getting too far away.
3. After training is done, use the go-to-command and test its capabilities.

3.3 Summary: How are the general problems tackled?

In the introduction we discussed some general challenges associated to self-learning agents and how different models have previously handled them. Let's now as a summary of this method see how this approach handles the same issues.

3.3.1 Method of Adaptability

The method of this thesis is a three layered structure with the robot and the environment at the bottom, a hyperSOM in the middle learning to associate different states with different servo-settings, and a curiosity based RL on the top that learns the probabilities of the hyperSOM to bring the agent between different states. Especially the RL tries to find the areas for which hyperSOM is doing the most progress, and make the agent focus on those areas.

3.3.2 High Degrees of Freedom

In order to decrease the degrees of freedom this work only allows one direct vector $\mathbf{x}$ per state. What this does is that the number of servos in principle stops being relevant while the associated states can be search in a continuous space through the semi-stochastic process defined for the hyperSOM. Left is only the global-space that is of much lower dimension, in this case 2D or 3D.

3.3.3 Learning Speed

Especially these two factors helps allowing for a high learning speed:

- The fact that the hyperSOM only needs to find one direct coordinate for every state means that its search space is also small. The fact that every possible coordinate $\mathbf{x}$ is also in at least one set $\mathcal{Z}$ also ensures that something is learned.
- Every experience is used in order to learn something new. Even if the agent misses its target it has also found a new way to find the state where it ended up instead.

3.3.4 Intrinsic Motivation

The intrinsic motivation is artificial curiosity. That is, it tries to maximize its learning-process when the ultimate goal is to learn how to control the global coordinate of the robot with the direct coordinates.

4 Experiments

4.1 Initial test of Curiosity Based RL

The initial test of AC was performed like this:

Algorithm 4 AC test

Create a grid of dimension 20×20.
Assign each square/state of the grid a uniformly random number in the span $[-0.4, 0.6]$.
Place the agent at position $(10, 10)$.

for Every iteration **do**
    Add $+1$ to the assigned number of the current state.
    Choose to go up, down, left or right. The boundaries are periodic.
    $n \leftarrow$ the assigned number of the state that the agent tries to reach.
    $r \leftarrow$ uniformly random number in span $[0, 1]$.

    if $r < 1 - \dfrac{1}{1+n}$ **then**
        Go to the attempted state. (Success)
    else
        Stay in the current state. (Failure)

Observe that a state with a start number < 0 can never be visited by the agent (if its not the starting position), and that the more time a state has been visited the easier it gets. This is meant to simulate the fact that things often gets easier and easier to do the more time they have been previously performed. The test is now to in one case allow the agent to pick its actions using AC, and in the other case to just pick actions randomly. All in all this creates a maze of reachable and impossible states, since the states of an initial value less than 0 can never be reached, while the other states can be reached with varying quality. Figure 4.1 shows the maze used, where all positions initiated > 0 are painted white, and the states that can never be reached are painted black.

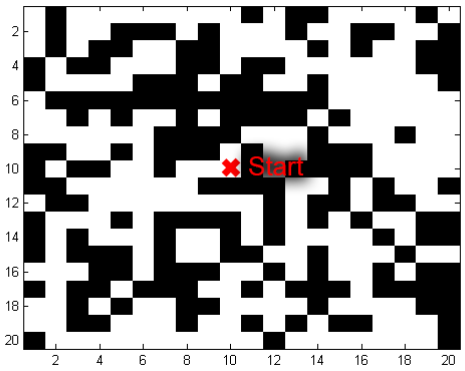


Figure 4.1: A maze used in the test of Curiosity Based RL. The agent starts at position $(10, 10)$ and can only visit the white squares, even though many of them will require multiple attempts before successfully reached. Note that the boundary conditions are cyclic.

4.2 Initial test of HyperSOM

In this test of the hyperSOM, every state s_i is given a coordinate $\mathbf{x}_{i,true}$ in direct space so that the neighborhoods of states are locally continuous but globally curved, see Figure 4.2 for the layout. The agent (the bigger dot) starts at a random position $\mathbf{x}_{current}$ and is told which state it is closest to. It is then, for every iteration moved randomly to the a new coordinate that is in the vicinity of neighboring state. By doing this repeatedly the hyperSOM is supposed to place its nodes $\mathbf{x}_{i,approx}$ as close to the real states as possible, including the states that has never actually been visited by the agent.

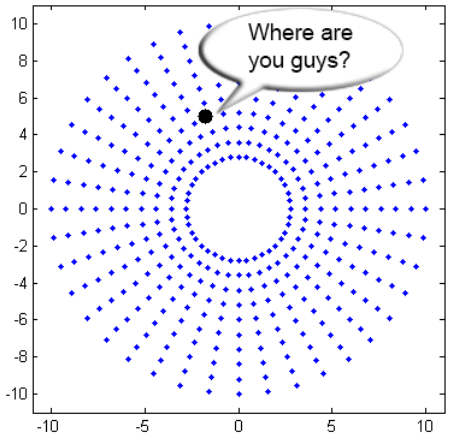


Figure 4.2: The task of the hyperSOM in this experiment is to place one node over each blue dot, where every dot corresponds to the direct position of a specific state.

4.3 Experiment 1 - Cartesian vs Spherical Coordinates

4.3.1 Description

As a first experiment we are interested to see the differences we get depending on the coordinate choice. Spherical Coordinates should be more "natural" to the system, but how important is this choice? Is it worthwhile to investigate the "best" coordinate system for the system, or is it adaptable enough to work efficiently even for the naive choice?

4.3.2 Set up

First of all we set $n_3 = 1$, and the reason for that is that this measure is the same for the two systems and by only having one layer the number of states can be reduced, leading to faster training. Furthermore we like the states to be as uniform as possible. In the cartesian case this means that

$$n_x = n_y, \tag{4.1}$$

and in the spherical case, where the height of a segment is $\frac{\pi}{2n_\theta}$ and the width (at the bottom segment) is $\frac{2\pi}{n_\varphi}$, it means that

$$n_\varphi \approx 4n_\theta. \tag{4.2}$$

There is however a good idea to allow n_φ to be odd since that would mean that the robot is in the middle of a state when lying on its back compared to being exactly on the edge between two states. Finally the two coordinate systems should have as similar amount of states as possible, and not too many states either. A smaller number of states should translate into a shorter training time, and furthermore, since the Markov Map only keeps track of if a transition is successful or not, and not how likely the agent is to end up in a specific state if a transition fails, the states needs to be fairly few so that the states are large enough to be reached with some certainty. On the other hand it needs to be enough states to allow interesting positions of the robots, such as sitting, back, belly, etc, and intermediate states that allows the robot to move between them.

4.3.3 Parameter choice

With previous considerations in mind it seems like a good parameter choice could be:

$$\begin{aligned}[n_x, n_y] &= [5, 5] \\ [n_\varphi, n_\theta] &= [13, 3]\end{aligned} \tag{4.3}$$

See Figure 4.3. All parameters that will be used for this experiment are:

Name	**Cartesian**	**Spherical**
I	2000	2000
$[n_1,\ n_2,\ n_3]$	[5, 5, 1]	[13, 3, 1]
N	25	26
r_a	1	1
M	169	183
M/N	6.76	7.04
$[max_{T_S},\ max_{T_{SA}}]$	[200, 20]	[200, 20]
$[t_{move},\ t_{rest}]$	[1.0, 1.0]	[1.0, 1.0]
λ	0.8	0.8
ξ	ln(100)	ln(100)
γ	0.9	0.9
spherical	False	True

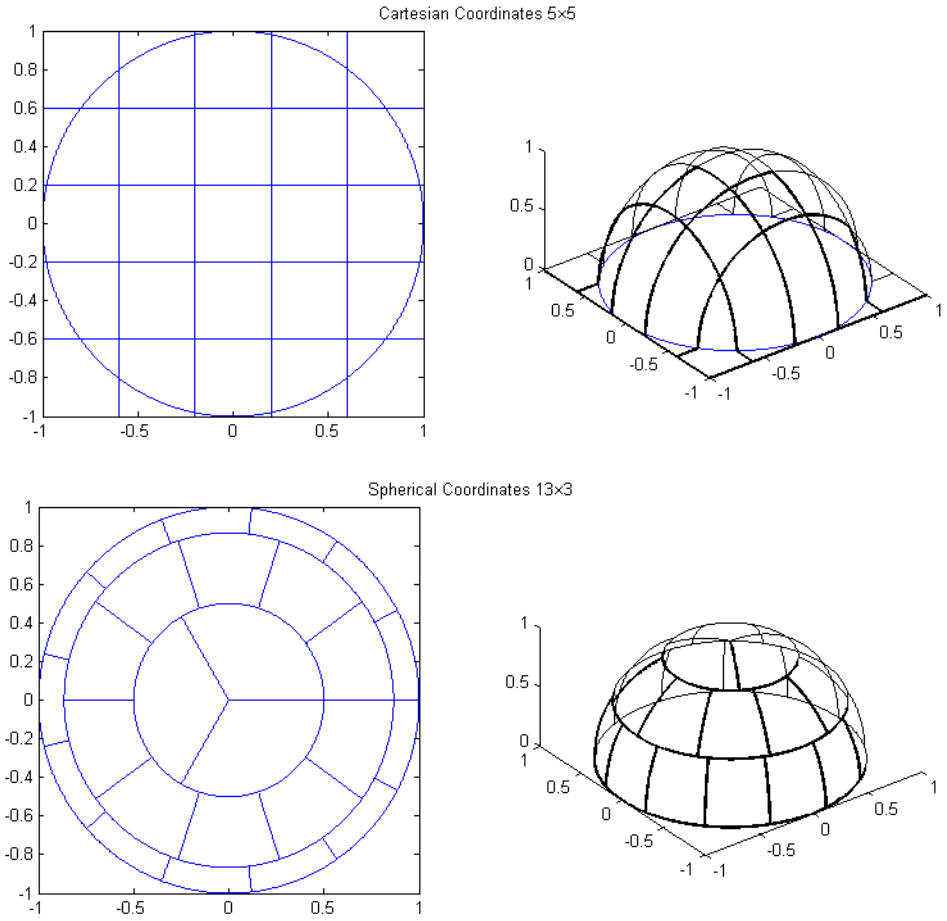


Figure 4.3: Experiment 1. The geometries of the different coordinate systems.

4.4 Experiment 2 - Importance of Curiosity Based RL and hyperSOM generalization

4.4.1 Description

The method of this thesis is based on a two layer solution with a generalization-based hyperSOM at the bottom, trying to figure out what direct coordinates to associate with what state, and a curiosity based RL at the top, trying to observe the progress of the hyperSOM to decide what actions that will develop it the most. But how beneficial is it to let the actions be picked by a AC-driven RL, or direct coordinates be picked by a generalization based hyperSOM? These matters will be examined in this experiment.

4.4.2 Set up

Parameters

The settings that are considered best in the Experiment 1 are the ones that will be used in this one.

RL

For the RL three different settings will be tried,

- Artificial Curiosity where rewards of states are counted as well as rewards of state-actions (AC).
- Artificial Curiosity where only reward of state-actions are considered ($AC_{w=0}$).
- Random Actions (RA).

It may be noted that $AC_{w=0}$ was the setting originally used in [13].

hyperSOM

In the hyperSOM case two settings will be tried,

- Generalization Based (GB)
- Non-Generalization Based (NGB)

See Theory for definitions.

The different cases

All in all four different settings will be tried:

1. AC + GB
2. $AC_{w=0}$ + GB
3. RA + GB
4. AC + NGB

4.5 Experiment 3 - Effect of damaged morphology

4.5.1 Description

In the heart of the method of this thesis lies generality, the notion that the self-organization of the agent should be independent of the shape of the agent and learn how to deal with whatever body it is given. Not only that, due to the plasticity of both the RL and hyperSOM it should also be able to overwrite older structures if the morphology of the agent's body changes. This should in theory mean that the robot is able to adapt to and restore its functionality in case of damage. In this experiment the effect of such a damage or change in morphology will be studied. More specifically how the robot will cope with the loss of an arm.

4.5.2 Set up

Three different cases will be compared in this experiment:

1. A reference run to see how the robot evolves with two arms. (2 Arms (ref))
2. How a robot trained with two arms reacts to the loss of an arm. That is, the one armed robot is Pre-Trained (PT).
3. How a robot evolves with one arm with No Pre-Training (NPT).

Beside this, the same settings are used as the best of Experiment 1, where the better run of that experiment will be the 2 armed reference.

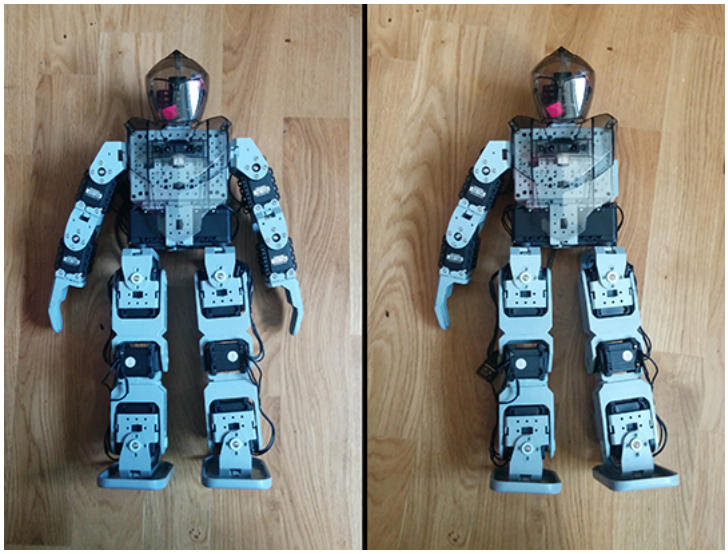


Figure 4.4: Experiment 3. Morphology before and after.

4.6 Experiment 4 - Effect of higher resolution

4.6.1 Description

What is the effect of a higher resolution in the state-space? On one hand the search space increases, on the other hand transitions between states might get easier due to more intermediate states and new states might also be more easily found since they are then closer to each other.

4.6.2 Set up

Use a cartesian grid 7×7, with $r_a = 2$. See Figure 4.5 for layout.

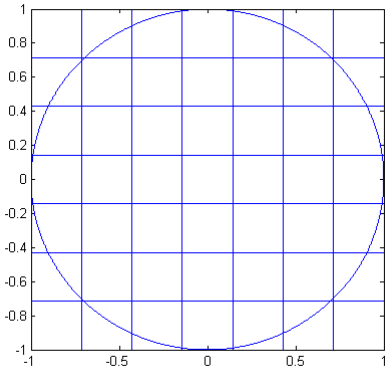


Figure 4.5: Experiment 5. Grid layout. Note that corner states can never be reached.

4.7 Experiment 5 - The Ultimate Challenge

4.7.1 Description

In this final experiment is the effects of longer periods of training examined, to see what this system is ultimately able to learn given time. This will be done in two runs, first in continuation of the better run of Experiment 1, and then using the "stretched out" parameter g_3 to see the effect of one added dimension and also if the robot is able to find the upright state with stretched legs, which can only be reached if the robot is standing.

4.7.2 Set up

- 2D: $I = 5'500$ iterations, $[n_x \ n_y] = 5 \times 5$.
- 3D: $I = 12'000$ iterations, $[n_x \ n_y \ n_3] = 5 \times 5 \times 3$.

In the 3D case $M/A = 15.77$ actions/state but besides that all settings will be in accordance with previous experiments.

5 Results

5.1 Initial test of Curiosity Based RL

Figure 5.1 clearly shows that exploration through AC is more efficient than random exploration.

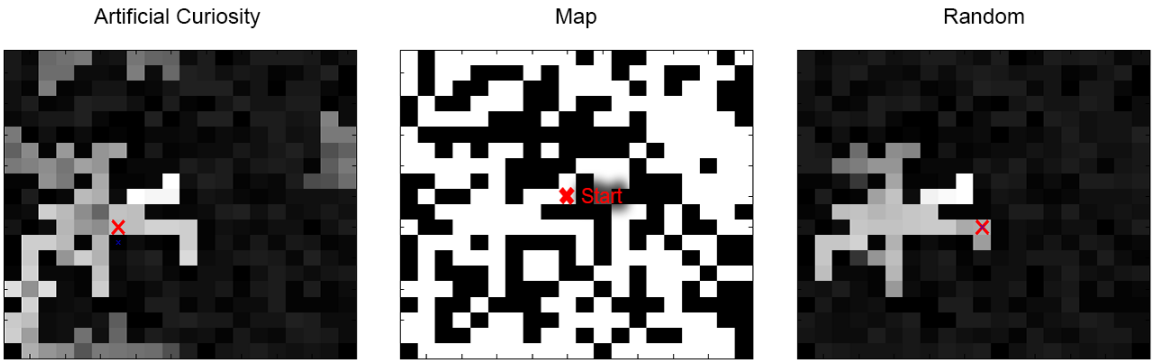


Figure 5.1: The different tones in the darker colors are due to their different initiation values, while lighter colors represents states that have been visited. The map in the middle shows what states were initiated > 0 so that they can be visited successfully at all. The red cross was given 10'000 iterations for exploration. Observe that the boundary conditions are periodic.

5.2 Initial test of HyperSOM

Figure 5.2 shows that the hyperSOM indeed is able to generalize the states it has found in order to predict the positions of states still unvisited, if they are spread out in an at least locally continuous fashion.

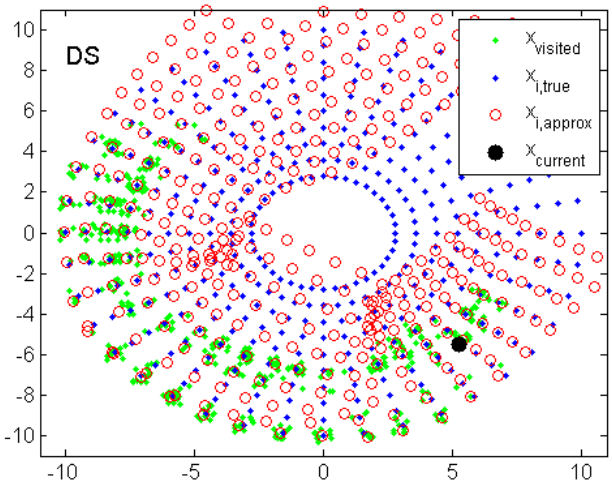


Figure 5.2: The hyperSOM at work. The goal of this hyperSOM is to place the circles $x_{i,approx}$ over the blue dots $x_{i,true}$, using the data found $x_{visited}$. In other words, green spots tells what regions the agent has been visiting, and the red rings are the agents estimates of where the blue dots are. Observe that if the blue dots are at least locally continuously spread out the agent is often able to make good guesses of the positions of the closest unvisited blue dots based on the positions of the visited ones.

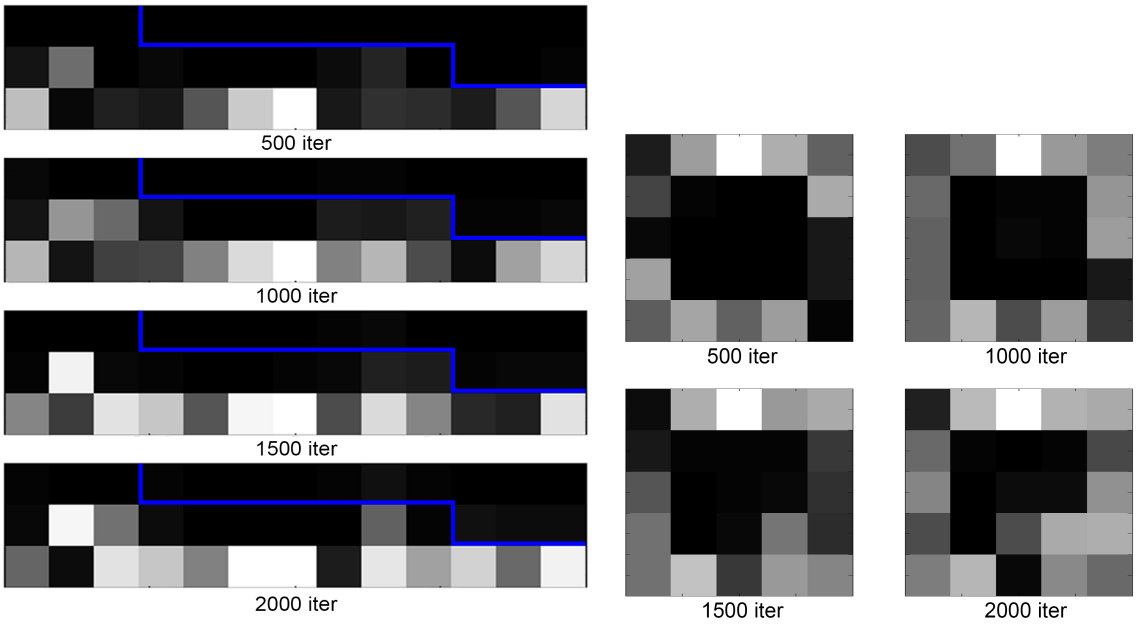


Figure 5.3: Experiment 1: This shows the mass of the different states during the different iterations. To the left is the spherical coordinate system where the states to the left of the blue lines are the ones in the sphere. To the right is the cartesian layout. Brighter colors means higher mass and thus higher certainty, but note that even darker states may be reachable, but since the robot doesn't have as much data on it yet the robot behaves more stochastic in those areas. One such example is that sitting states were reachable in the 1500 iteration in the spherical case, and the 2000 iteration in the cartesian case, however those states are not enough explored in any of the charts in order to accumulate any mass.

5.3 Experiment 1 - Cartesian vs Spherical Coordinates

5.3.1 The typical run

There were no visible major difference to the naked eye between the two systems, here follows a typical run.

1. Iteration $1 - 20$: Typically the robot began to move randomly in order to find the adjacent states to its original position on its back. Soon, within 10-20 the closest neighbors are found.

2. Iteration $20 - 100$: Now the robot's behavior starts to become more determent as it works to rotate even more. It isn't unusual for the robot to then soon manage to rotate and fall flat on its belly. Especially for cartesian symmetry the robot tends to get stuck on the belly for a while.

3. Iteration 500: At this point the robot have usually gone through all states while lying down, even though it can't go between them with any certainty.

4. Iteration 1000: The robot has now usually been in a sitting position at some point. It still can't get up by determination, but since the robot is doing progress within the field it finds it interesting and typically switches between back and sitting.

5. Iteration $1000 - 2000$: At this point the robot basically tries to puzzle together all the different areas it has an understanding of. This might mean that it changes some states that was originally used to get between belly and back to instead help the robot get from back to sitting. This in turns might mean that if it falls on its belly now it often have huge problems to get back since the side states are now more optimized for getting the robot to a sitting position than to move between back and belly. Sometimes however one side is used for getting to and away from belly, while the other side can be used in order to get to a sitting pose.

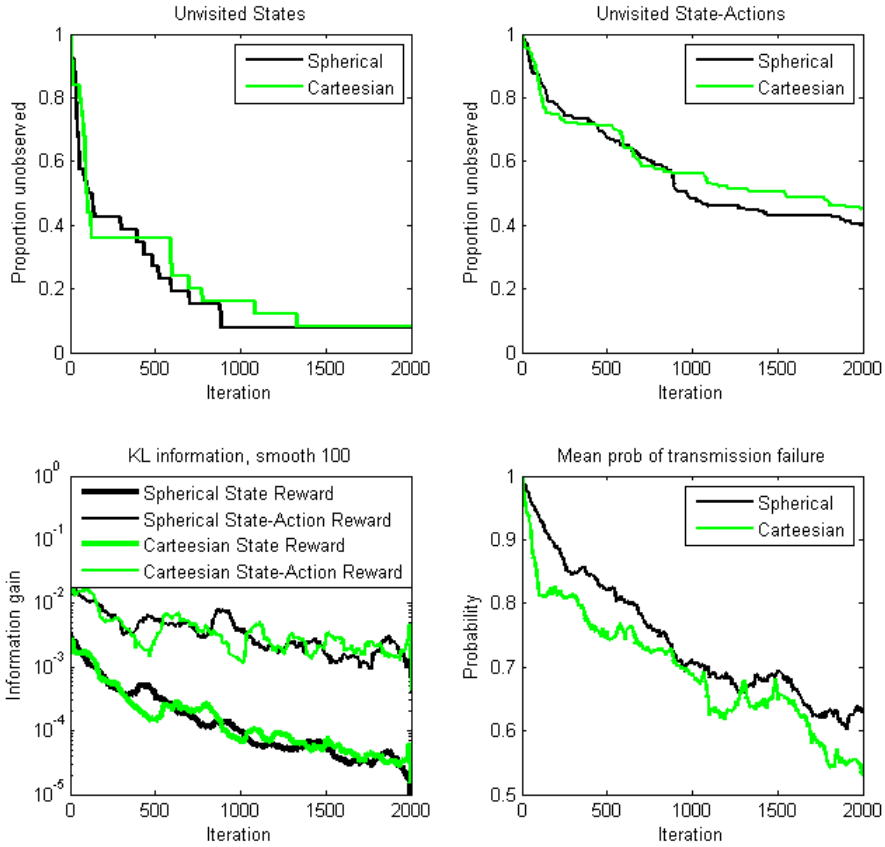


Figure 5.4: Experiment 1: Time evolution of different properties between a spherical coordinate system and a cartesian one based on one run each. In the case of KL information, the thicker lines denotes the state reward, while the thinner denotes the state-action reward.

6. Iteration 2000: At this point the robot can often be told to get from back to any of its sides, and sometimes to its belly and most often to a sitting position.

Figure 5.3 shows the mass of the different states during the different iterations. All in all 1000 iterations took approximately 1 hour, where the robot ran in periods of 500 iterations ($\sim$ 20 min) followed by a 10 minutes rest in order for the servos to cool down.

5.3.2 Data

Figure 5.4 shows the time-evolution for some different properties in the two different systems, and Figure 5.5 shows the distribution of how probable it is that a transition to that state will fail, see Section 3.2.4. This is data from just one run each, but together with previous test runs it is hard to find any major differences between the two systems at all. One general difference however seems to be that the spherical system discovers new states at a more even pace while the cartesian more often finds them in chunks. It makes sense since the spherical states are more evenly distributed over the sphere, while cartesian states are denser in some areas and less dense in others. In this example the cartesian system seemed to do slightly better but from previous test runs it seems to be as common that the opposite is true.

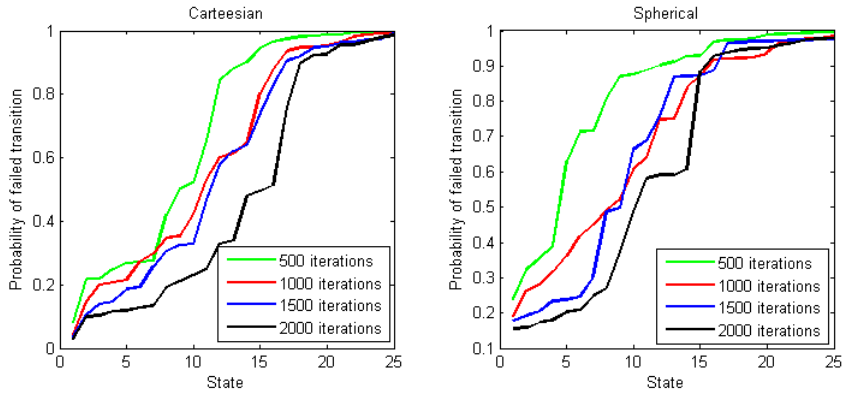


Figure 5.5: Experiment 1: This shows the distribution of how probable it is that a state will be reached successfully for the two different systems, at different points in its evolution.

5.3.3 Evolved behaviors

In this evaluation of a test $a \leftrightarrow b$, "yes" means that the agent is able to freely between the two states a and b, "$\rightarrow$" means that the agent is able to move from a to b but not the other way around, "$\leftarrow$" the opposite and "-" that the robot is unable to move between the states at all. In the case of sitting we are only interested if it is able to get up at all, so that is either "yes" or "-".

During the training the robot builds up a knowledge of how to move around in its global space. The results of some specific tasks are for the cartesian system

Iteration	**back $\leftrightarrow$ r-side**	**back $\leftrightarrow$ l-side**	**back $\leftrightarrow$ belly**	**rotate cw $\leftrightarrow$ ccw**	**back $\leftrightarrow$ sitting**
500	$\rightarrow$	-	-	-	-
1000	yes	$\leftarrow$	$\leftarrow$	-	-
1500	yes	$\leftarrow$	yes (barely)	-	-
2000	yes(barely)	yes	yes	ccw	yes (barely)

and for the spheric system

Iteration	**back $\leftrightarrow$ r-side**	**back $\leftrightarrow$ l-side**	**back $\leftrightarrow$ belly**	**rotate cw $\leftrightarrow$ ccw**	**back $\leftrightarrow$ sitting**
500	yes	$\rightarrow$	$\leftarrow$ (barely)	-	-
1000	yes	yes	yes	cw	yes (barely)
1500	yes	$\rightarrow$	yes	cw	yes (barely)
2000	yes	yes ($\leftarrow$)	yes	yes	-

Figure 5.6 shows how a rotation between back and belly would be performed by the spherical system, figure 5.7 shows how the robot would get into a sitting position after being on its back, and figure 5.8 shows an example of how the robot got to a sitting position in the spherical system in one of the test runs. Even though it was a test run this behavior is included to show the similarities between the ways in which the robot gets up. It seems like, independent of the imposed geometry, the best way to get up is to roll to the side and get up from there using one arm.

5.3.4 Evaluation

It is impossible from these two trials together with the experience from previous test runs to conclude that one system is better than the other. If the robot would be able to separate up from down the spherical system would be obvious, but since that is not the case here I conclude that the subsequent experiments could be performed in the cartesian systems as well, and since that system is more basic and easy to interpret, together with the fact that the underlying hyperSOM also makes its predictions based on a cartesian grid (see Discussion section 6.1.2), that is the system that will be used henceforth.

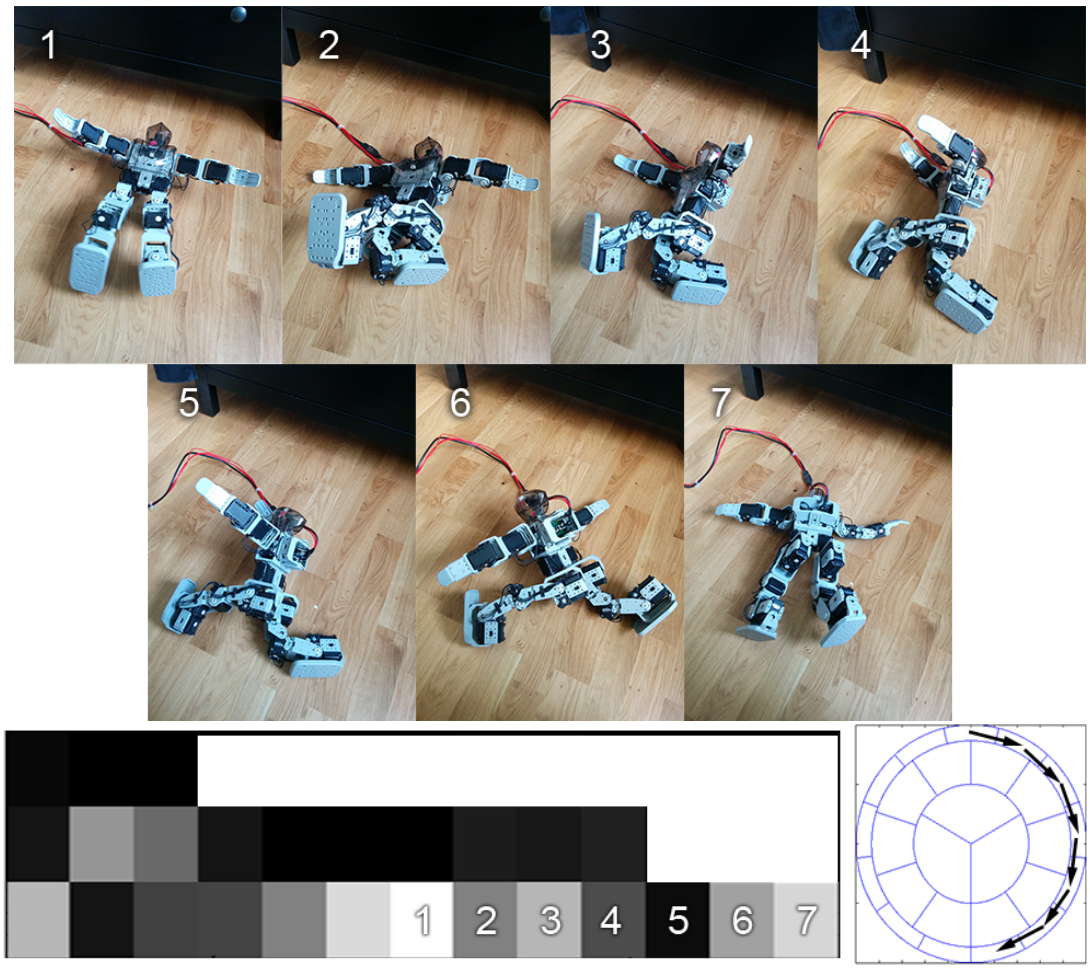


Figure 5.6: Experiment 1: How the robot performs a roll over if asked. Note that it can get back to it's back again by doing the poses in the opposite direction.

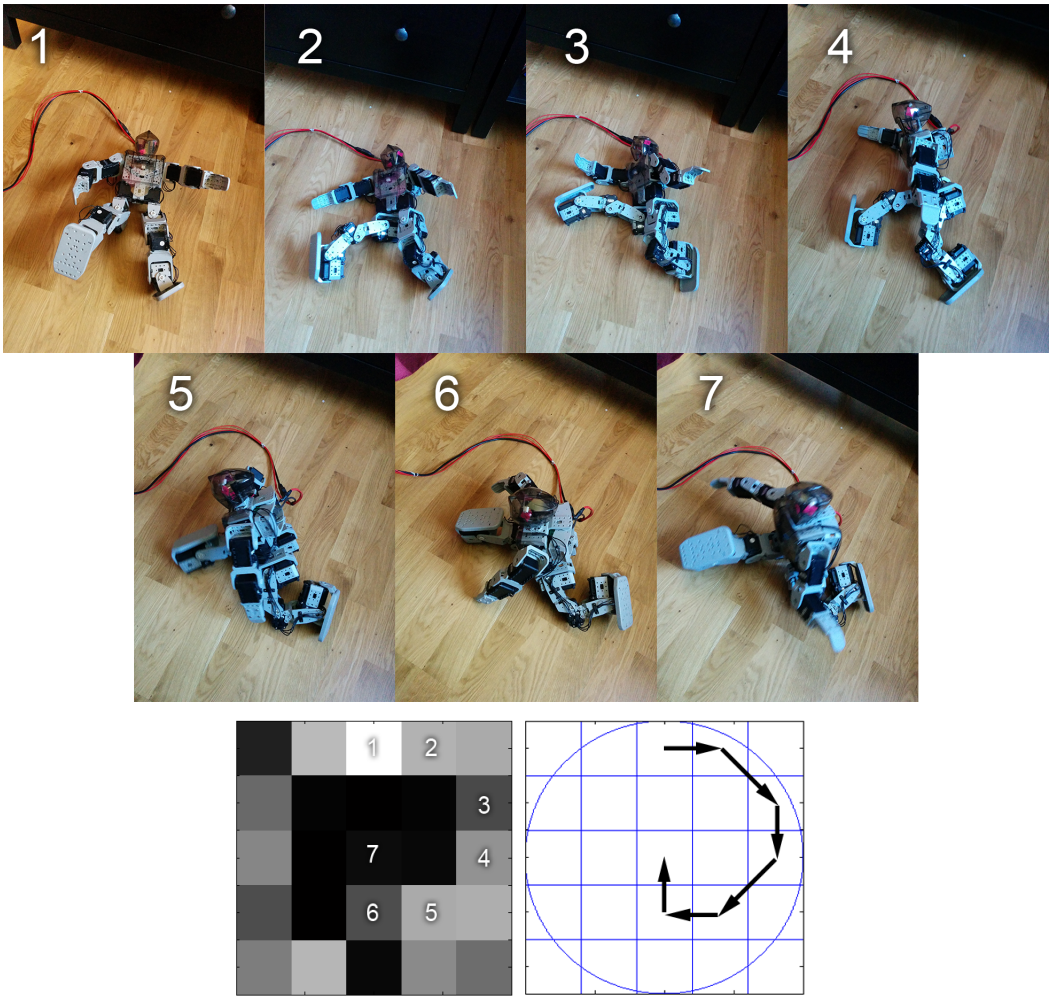


Figure 5.7: Experiment 1: The way in which the robot sat up in the cartesian system.

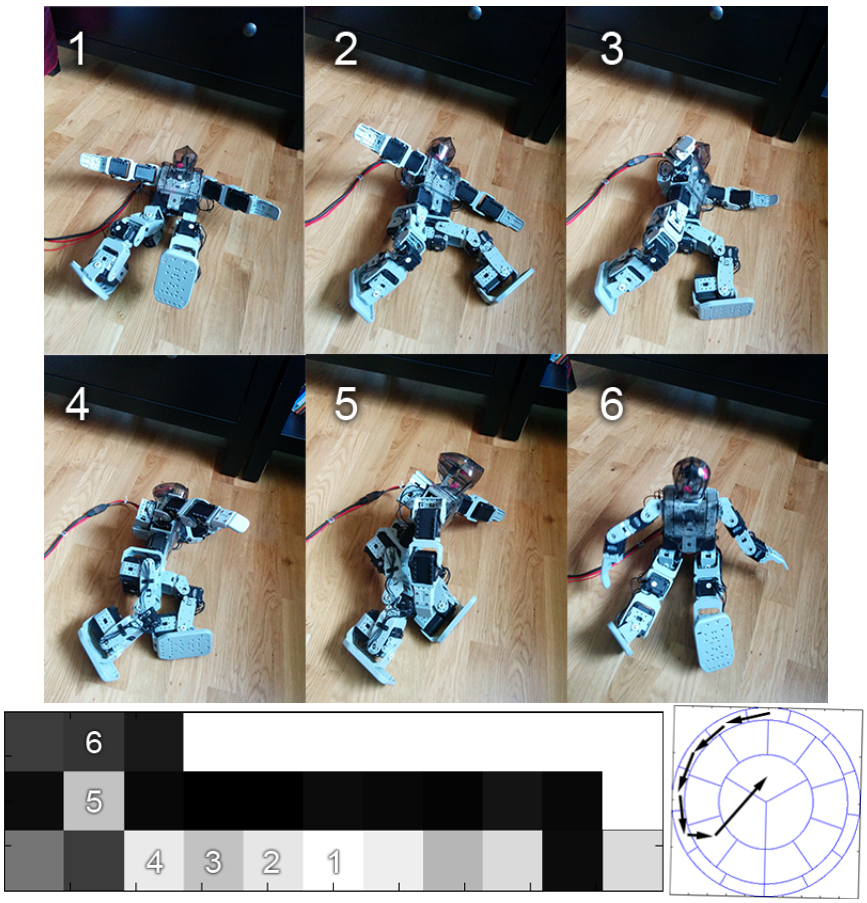


Figure 5.8: Experiment 1: The way the robot sat up in one of the test runs, in a spherical system of 11×3 (not 13×3 as in the example in the graphs). Notice the similarities in the pattern of the arrows compared to Figure 5.7.

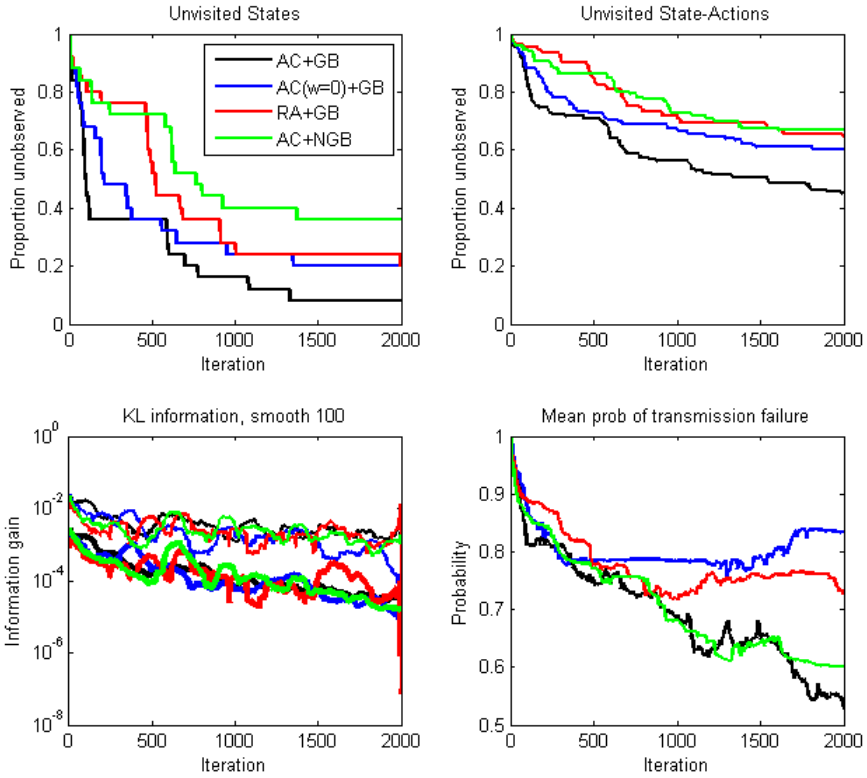


Figure 5.9: Experiment 2: Time evolution of different measured properties. In the case of KL information, the thicker lines denotes the state reward, while the thinner denotes the state-action reward.

5.4 Experiment 2 - Importance of Curiosity Based RL and hyper-SOM generalization

5.4.1 Data

The performance may be seen in Figure 5.9 and 5.10.

5.4.2 Evaluation of the different settings

AC+GB

This is the regular setting. It performed the best in all categories, except perhaps for the KL-graph that is difficult to read anything out from really.

AC(w=0)+GB

This is the setting that was used in [13]. Except for the original settings this setting performed the best in the categories of finding the different states and state-actions, but the worst in how probable it would be to successfully do a transition to any state, on average. Figure 5.10 shows that this may be due to that this method lets the agent reach a few states with a high precision, while the other may almost never be reached.

RA+GB

This agent mostly stayed around the more stable area of the global space, such as the back or belly positions. It sometimes got close to states such as sitting up, but then when it fell over back to its back it wouldn't go

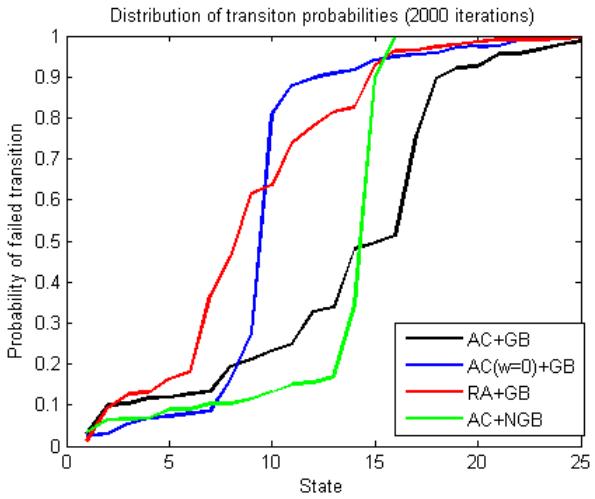


Figure 5.10: Experiment 2: The distribution of how probable it is that a state will be reached successfully in the different options.

back to where it just had been for a while.

AC+NGB

Of all methods this method finds the least number of states and state-actions. On the other hand it is able to reach the states it has found with high accuracy.

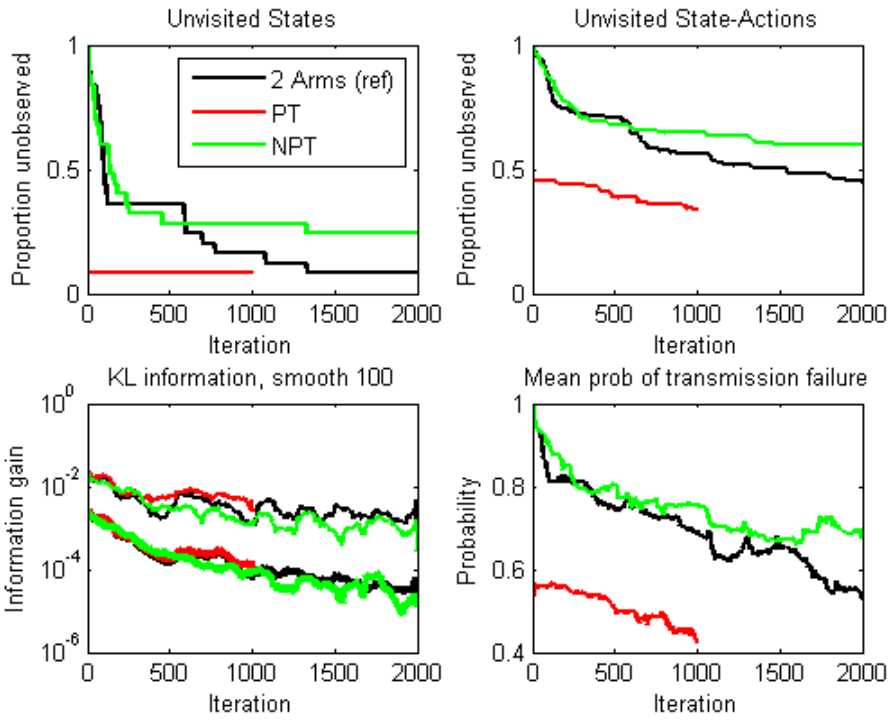


Figure 5.11: Experiment 3: Time evolution of different measured properties. In the case of KL information, the thicker lines denotes the state reward, while the thinner denotes the state-action reward.

5.5 Experiment 3 - Effect of damaged morphology

5.5.1 Data

The evolution of the different parameters may be seen in Figure 5.11 and 5.12, while charts of how the mass accumulated during the runs may be seen in Figure 5.14 and 5.13.

5.5.2 Evaluation of different settings

With pre-training (PT)

In the case where a pre-trained brain was used from a 2 armed robot, the robot almost immediately noticed that something was strange on the side of the lost arm. It immediately became obsessed by that side and soon (less than 100 iterations), it had figured out a new way to roll over at that side. The initial idea was to run it for 2000 iterations but since it so soon reached the same performance as the pre-trained 2-armed robot only 1000 new iterations were made.

Without pre-training (NPT)

Just from the look, the robot did quite well and found the side states to a good precision. It however had a hard time to reach the sitting states, mostly because it tried to get up in an almost standing pose, making the whole thing very unstable, see Figure 5.15. From the Figures 5.12 it looks as if this robot made worse than the original one trained with two arms, but contrary to this, this was actually one of the best robots of all the test, when it came to controlling it. It could basically reach all its important states if asked to, except to sit up stable as explained earlier. Thus it seems like the measures of the graphs are not telling the whole story. If one look at Figure 5.11 one striking thing is that the states it do know, are very well known. This was however also the case of the AC(w=0) setting of the previous task, but that robot was not among the better at all.

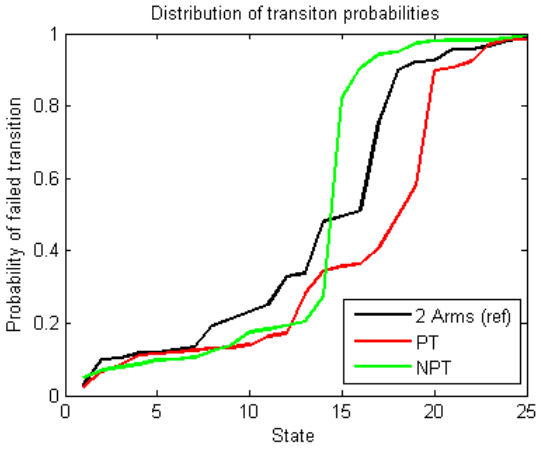


Figure 5.12: Experiment 3: This shows the distribution of how probable it is that a state will be reached successfully after 2000 iterations in the NPT and reference case, and 1000 iteration in the PT case, not counting the 2000 iterations before with two arms.

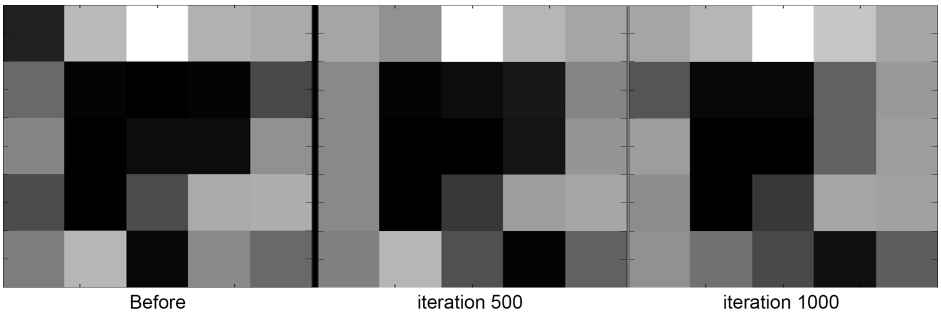


Figure 5.13: Experiment 3: PT. How the mass differed in the states during training. Lighter colors means more mass.

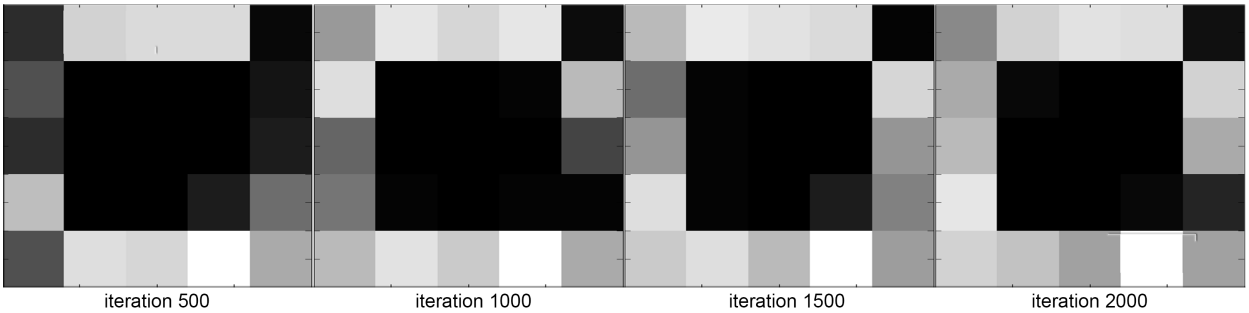


Figure 5.14: Experiment 3: NPT. How the mass differed in the states during training. Lighter colors means more mass.

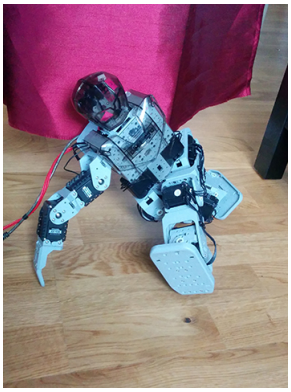


Figure 5.15: Experiment 3: The robot that started without any pre-knowledge developed an upright pose where it didn't sit but rather tried to stand in this fashion. This made the center states really hard to find for the robot and they thus never managed to gain much in mass.

Evolved behaviors

Figure 5.16 shows the same basic behavior: the robot turning from its back to its belly, in three different settings. First in the way evolved with two arms, and then in the new way evolved from the previous one when the loss of one arm changed the morphology of the robot, and lastly in the way found if the robot had one arm all the time.

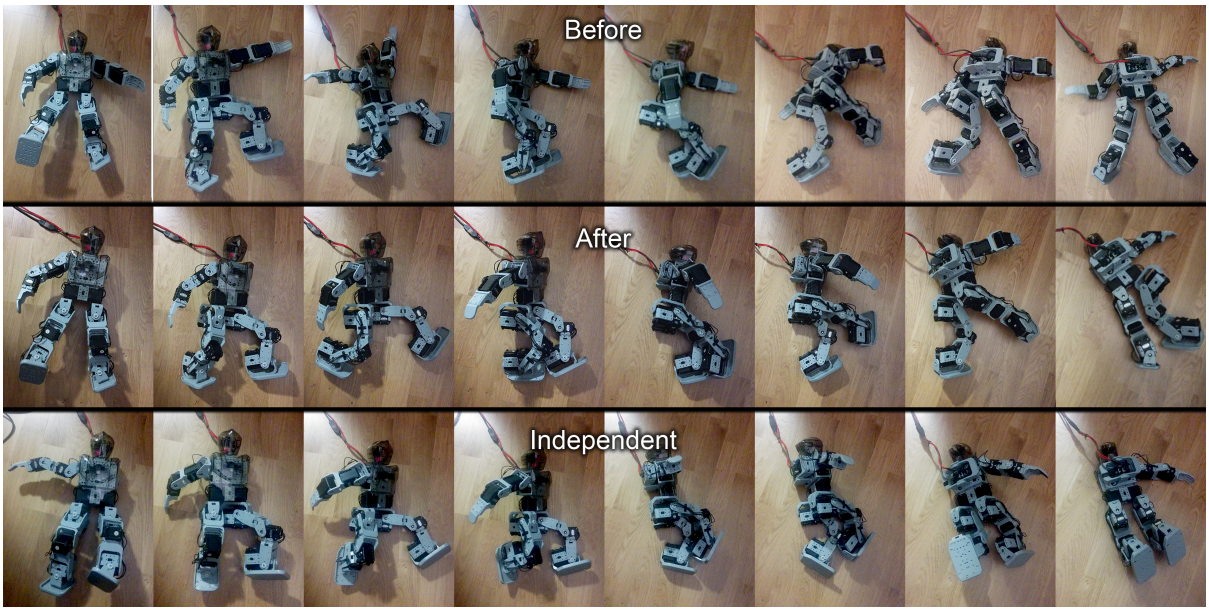


Figure 5.16: Experiment 3: Turning comparison. First the evolved turning for two arms, then how it changed when one arm was removed and finally a turning evolved with no pre-training. Each column represents a specific state, meaning that all robots in one column are in the same position in global space.

5.6 Experiment 4 - Effect of higher resolution

5.6.1 Data

The evolution of the different parameters may be seen in Figure 5.17 and 5.18, while charts of how the mass accumulated during the runs may be seen in Figure 5.19.

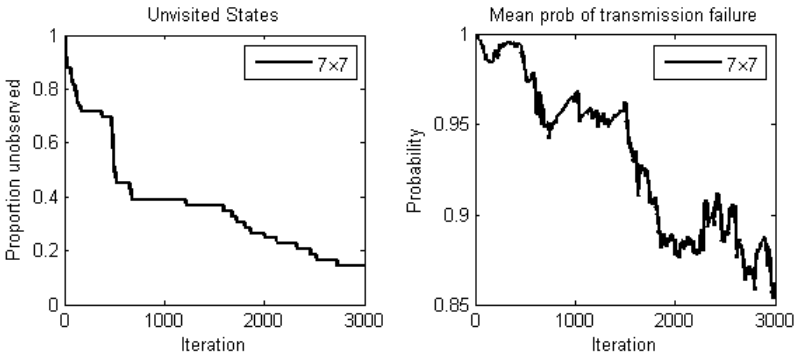


Figure 5.17: Experiment 4: How the number of unvisited states and mean probability that a transition to a state will fail, change over time. Number of unvisited state-actions are excluded since it was unclear how to count their exact number since new state-actions could be added if found by accident within the action radius, but outside of the initiated state-actions.

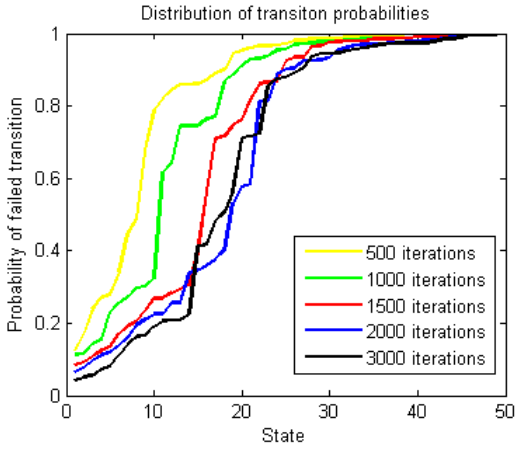


Figure 5.18: Experiment 4: The distribution of how reliably the different states can be reached.

5.6.2 Evaluation

The capabilities were at the last iteration the following:

Iteration	**back ↔ r-side**	**back ↔ l-side**	**back ↔ belly**	**rotate cw ↔ ccw**	**back ↔ sitting**
3000 iter	yes	yes	→	-	yes (some states)

It actually had the physical capability to get from belly to back, but didn't have good enough statistics on which way to take so therefore failed too many times. Since there were so many sitting states there were only a few the agent mastered, the center-most not one of them. Within the areas well known to the agent, it moved in the most smooth way yet seen.

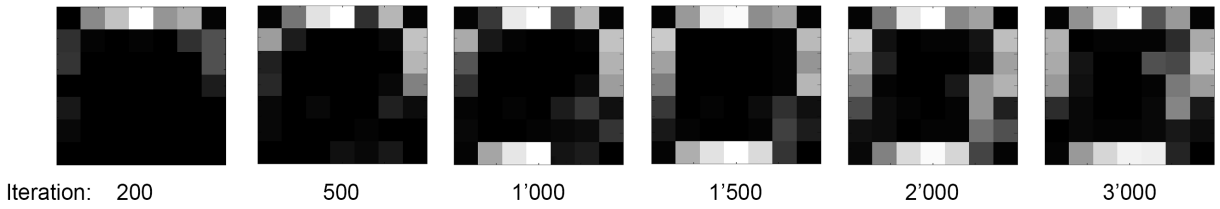


Figure 5.19: Experiment 4: How the mass of the states changed over time, light colors means more mass.

5.7 Experiment 5 - The Ultimate Challenge

5.7.1 Data

The evolution of the different parameters may be seen in Figure 5.20 and 5.21, while charts of how the mass accumulated during the runs may be seen in Figure 5.22 and 5.23. Figure 5.24 shows how a posture (direct coordinate) evolved over time in the 3D cases. Figure 5.25 shows the impact of the three g_3 layers and what they correspond to in terms of how stretched the legs are.

5.7.2 Evaluation of the two runs

2 Dimensions

This was a continuation of the agent in the cartesian case of Experiment 1, so up until iteration 2000, they are the same. After that the most interesting thing was that it started to try to reach the belly positions from a sitting position, see Figure 5.26 (left) around iteration 5'000. Another change was that it after a while concentrated on smaller areas of its state space. In the beginning it seemed more interested in finding all states there were, but after a while it rather tried to figure out the discontinuities of its state-space. That is, an exploratory behavior became more exploitative. When doing so it often happened that when the robot noticed and fixed a broken transition A to B, it then often destroyed other transitions in the process so that for example transition B to C didn't work anymore. More specifically the robot often swished uses of side states between leading to sitting coordinates or belly coordinates, while at the same time excluding the other transition. All in all the charts show that the success-rate increased most of the time, yet it's not clear that the final robot would be considered "better" after the 5'500th iteration than after the 2'000th by someone who would try to control it, since it's lost many of the important transitions mentioned earlier. The overall performance after the final iteration where:

Iteration	**back ↔ r-side**	**back ↔ l-side**	**back ↔ belly**	**rotate cw ↔ ccw**	**back ↔ sitting**
5500 iter	yes	yes	←	-	yes

3 Dimensions

This agent behaved more or less as the two dimensional one, even though progress were somewhat slower due to the three times as many states. It soon became clear that this agent would never be able to find a standing posture because it had too few or no useful intermediate postures that could lead to such a state, and there were far to many other easier states that it couldn't solve that kept its attention. Neither seemed the additional number of layers help it to for example use some layers in order to get around to the belly, and some others to go to a sitting position. This conflict became very clear around 7'000 iterations when it had found a very efficient way to get to a sitting position with very high accuracy and from both sides too, and then fell to its belly and had no idea of how to get back to it's back since all its side states were now used to get between the back and sitting. This spurred an intense interest in a transitions out from the belly states, leading to a rapid decay of the side states. This in turn meant that that side state lost its functionality as a state to use in order to reach sitting states. The new transition between back and belly was however not perfect since the legs were too close together, meaning that its feet often got stuck in one another, see Figure 5.26 (right).

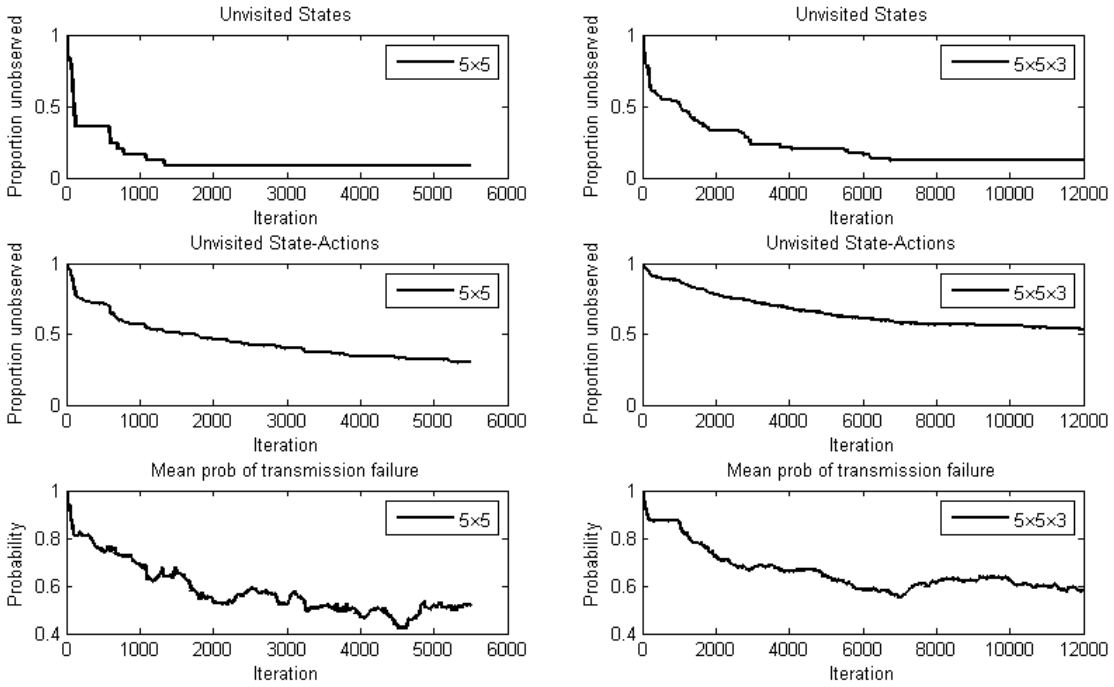


Figure 5.20: *To the left:* The progress of the 2 dimensional agent. *To the right:* The progress of the 32 dimensional agent. *Both sides:* The first plateau of the charts ($\sim 50 - 500$ left side, $\sim 100 - 1000$ right side) where when the robot first fell over to the belly and got stuck.

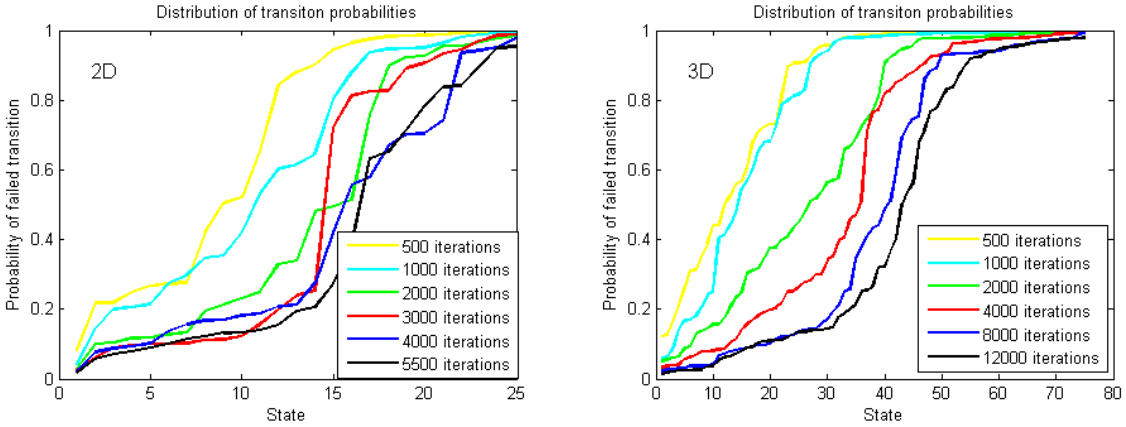


Figure 5.21: The distribution of how reliably the different states can be reached. To the left in the 2D case, and to the right if the g_3 dimension is introduced.

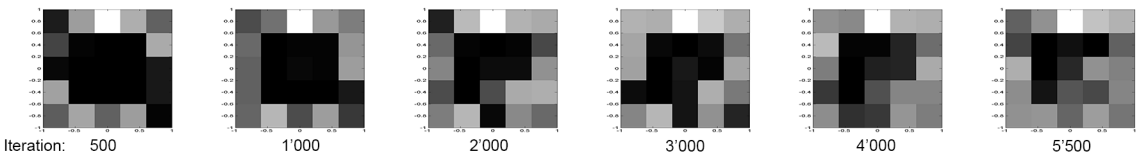


Figure 5.22: How the state-space is explored in the 2D case.

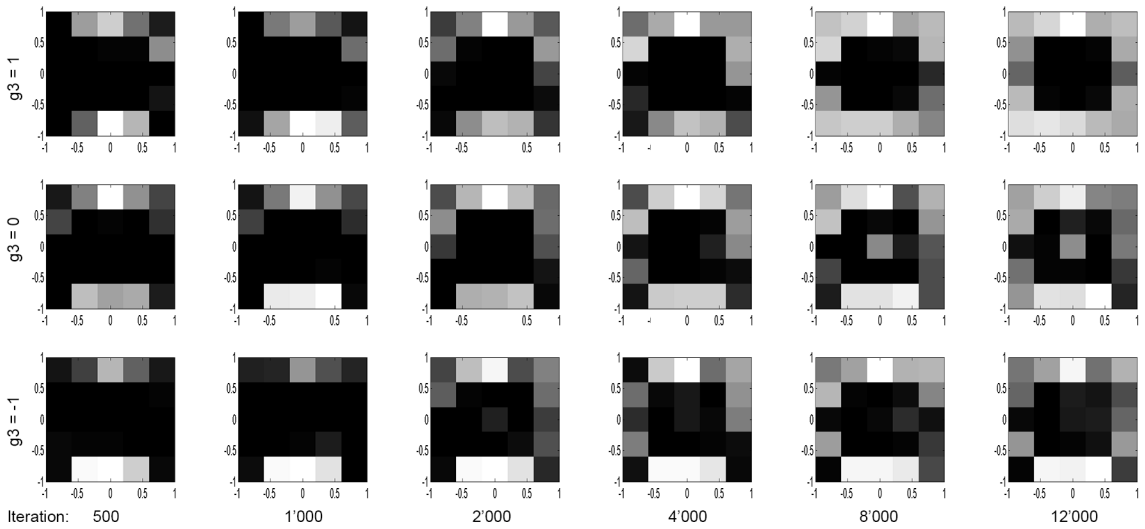


Figure 5.23: How the state-space is explored in the3D case.

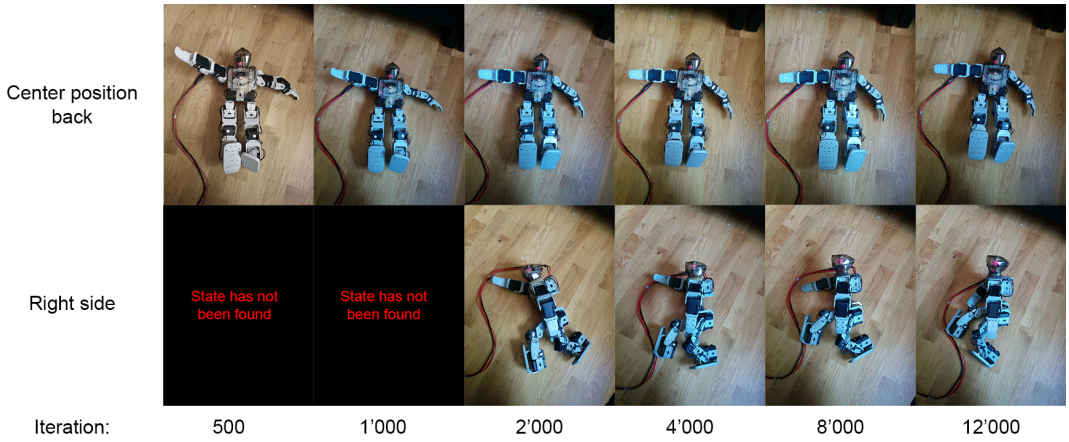


Figure 5.24: How direct coordinates (postures) evolves with time. A trend seems to be that the states evolves to allow for transitions to the adjacent states with higher accuracy, since the "center position back"-posture lifts its legs up slightly and separates them somewhat, allowing it to more easily turn over to any of its sides, and the "right side"-posture brings its arms in, to make it easier to rollover to any side.

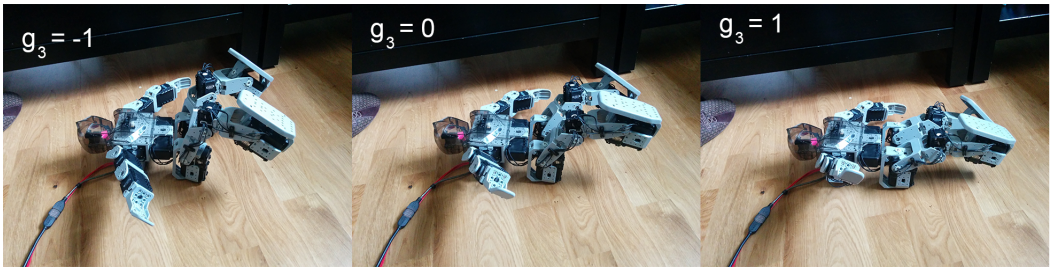


Figure 5.25: Impact of the third global parameter in terms of how stretched the legs are.

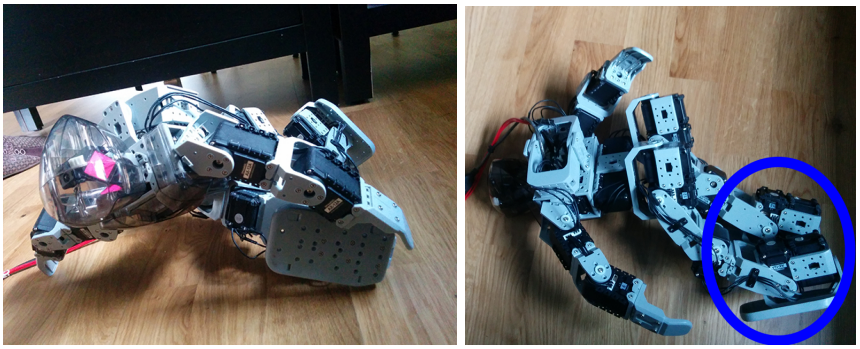


Figure 5.26: Left: 2D: Tried to learn how to reach the belly while still sitting, and since state was so wide it almost succeeded. Right: (3D) Feet got stuck in each other so had problem rolling back from belly. Wants to pull left leg back.

Iteration	**back ↔ r-side**	**back ↔ l-side**	**back ↔ belly**	**rotate cw ↔ ccw**	**back ↔ sitting**
12000 iter	yes	(lost $\sim 8'000$)	yes	-	(lost $\sim 8'000$)

6 Discussion

6.1 Interpretation of outcome

6.1.1 Capabilities

Finds new states efficiently

This method allowed the agent to quickly visit most of the states given, regardless of the coordinate system. In both the spherical case (13×3) and the cartesian case (5×5) more than half the states were visited after only 100 iterations, and around 1000 iterations only a couple (or none in some test runs) were left. When the state-space was twice as large, as in the 7×7 run, half the states were visited around the 500th iteration, so it didn't scale in a linear way in this instance. This is probably related to the fact that so many more state-actions where possible in this experiment, due to the increased action radius of 2. It seems generally for all cases as if the agent started with the easier states to then work itself up to the more difficult ones to reach, such as the sitting states, which is exactly what one would expect from a curious agent that by definition should focus on where the most progress was made.

Finds relevant postures despite high DoF

The robot really found good and relevant postures using this method, why? One idea is about embodiment. Since the direct coordinates of the physics of the body they automatically cluster around the coordinates that resonates with the given body. In a way the relevant postures of the body thus work as attractors. This is evidently much more efficient than a brute-force search of the full 18-dimensional servo-space would be.

Can be smoothly controlled in well understood areas

The robot was easily controlled in the parts of the state-space where the agent had a good understanding, such as from the back, to the sides and sometimes to a sitting posture and other times to the belly.

Restructures its mind if necessary

The robot showed in numerous occasions that it was able to rewrite direct coordinates that didn't work as intended. It could be that it was able to recreate a lost way from its belly back to its back, or to rewrite a rotation learned with two arms when one arm was lost. In the evaluations of what an agent had learned and it was told to perform certain tasks, hyperSOM was frozen so that testing of a skill wouldn't affect its ability to perform that task. In other applications this would however probably be desirable since it would mean that even if the agent didn't know how to do a task when asked, it would when asked restructure its brain in order to be able to perform that task.

Searches efficiently around the found states

One thing that however worked really well was the way in which mass was connected to the stochasticity of a state. Since a random number each time scaled the radius of what direct coordinate to try (see equation 2.15) then even if a node had a good direct coordinate but low mass the agent would sooner or later still try states very close to that one. This meant that if a good direct coordinate were found but the node was then decayed by some reason, the robot would still eventually find that coordinate again.

Coordinate system independence

The coordinate system didn't seem to be very important to the performance of the agent. This is nice since it means that not too much effort needs to be invested in how to divide its state-space as long as the states are somewhat evenly spread out and not too oddly shaped. If however the agent was able to separate up from down a spherical system would probably be better.

Morphology independence

Experiment 3 showed that the method worked equally well to a slightly different body, one with just 1 arm, and that a previously 2 armed robot could learn how to function after a loss of one arm. It would however be interesting to test the method on a completely different body, and see if it really is as general as it seems to be.

6.1.2 Limitations

Figure 6.1: Can't go directly to the wanted belly pose since uppermost leg is behind the lower leg. Instead falls flat on the back.

Transition conflicts

Even though some charts are hinting towards that the progress continues as more iterations are done, practical experiences hints to that the agent isn't able to enhance its functionality much by additional iterations after around 2000 iterations. It seems to be some inherent limit to how big part of its state-space it is able to control smoothly at any single point in time, since increased functionality in one area leads to a decreased functionality in another. This became clearest in the side states that had to be used both to allow the agent to get to and from its belly, but also in order to get up to a sitting position. The conflict is this: In its belly position the robot often has its legs and arms spread out for maximum stability. To go directly to such a state from a state lying on the side it is important that the uppermost leg is in front of the other. If it isn't and the robot spreads the legs out for the belly pose, then the robot flips itself back to its back as in Figure 6.1. This pose, however, is a really good pose if the agent instead wants to sit up in the next step, and this is the source of the problem. If the state is used for transition to the belly it can't be efficiently used in order to get to a sitting position, and vice versa. The result of such a conflict were often to find something in between, but in this case that meant that its legs came closer and closer to each other which in turn increased the risk of them getting stuck

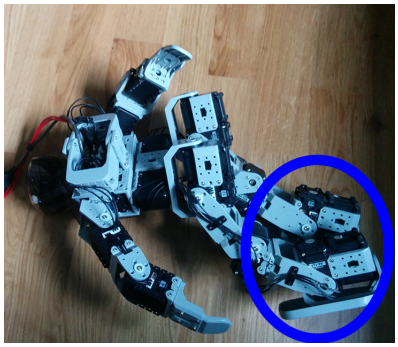


Figure 6.2: The robot can't decide weather to use this state as a transition to the belly or to be sitting which forced the legs to come closer and closer to each other. To have the legs close to each other on the other hand has its own risks since the legs might then get stuck in each other. This will decay the states that it is trying to reach, but strengthen the state with the feet stuck in one another since it will observe that it is stable in the current state when doing so.

in each other, which can be seen in Figure 6.2. This will in turn decay the states which it's trying to reach, while strengthen the state with the feet to close to each other since it was stable that way.

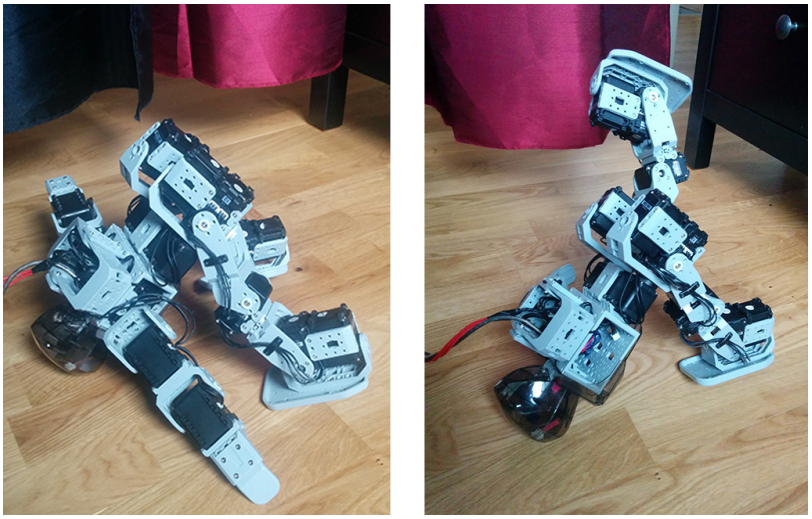


Figure 6.3: Two examples of how the robot wrongly thought it came come to sitting states due to its inability to separate up from down.

Up/down blindness

The inability to separate up from down had some negative consequences. The first one was that the agent often tried to reach sitting poses by trying to stand on its head as much as possible. These states were especially common when lying on the belly. The problem with this was that it then often thought it had reached the sitting poses which made it surprised when it then couldn't do the previously possible transitions, which in turn punished the states it could no longer reach. These upside-down states often became less frequent as training proceeded, probably because the regular states where more stable which in turn allowed the agent to spend more time there. Figure 6.3 shows one such occurrence. Another effect was that the lowermost states of the robot became twice as large. If for example a state stretched between $[0, \theta]$ from the equator of the half-sphere of Figure 2.8, then in reality all angles $[-\theta, \theta]$ would belong to that state.

Spherical neighborhood problem

In this implementation the nodes of the hyperSOM were ordered into a cartesian grid. This might have created some problems for the spherical coordinate systems where the states where not in a cartesian grid. Rather was the states in this implementation ordered so that the kth column of the nth row represented the kth state on the nth ring. Since different rings had different numbers of states this made the states more and more skewed in comparison to their closest neighbors in other rings. As an example: The top of the sphere had only 3 states, while the equator had 13. But in the grid the three states of the top would be directly above the three first states of the equator, even though the third state of the top was in reality closest to the 11th or 12th state in the bottom. See Figure 6.4.

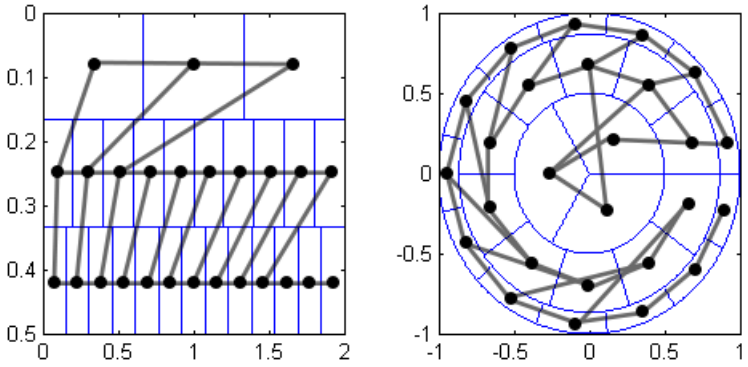


Figure 6.4: How the hyperSOM grid gets distorted in a spherical coordinate system, and why generalizations in the y-direction gets shifted.

Didn't handle larger action spaces well

In the case of resolution 7×7 where an action radius of 2 was allowed there where about twice as many states as in the 5×5 case, and yet was the exploration of the states more than two times slower. The reason was probably the much larger action spaces since every state now had up to 25 different states to move to, compared to 9 possible states in the 5×5 case. In reality the number of actual actions that could be attempted was less, since only states at a distance of 2 could only be attempted if that transition had been confirmed earlier, but nonetheless did the agent get stuck in an endless effort of trying all those actions out, most of which were fairly uninteresting, and sometimes lead to decay of states that could easily be reached from its immediate neighbors. An idea that was tested in order to decrease this was to put a roof of the number of possible actions in any state, and pick out them through roulette wheel selection [16] after how probable such a transition would be to succeed, but I never got this to work properly.

6.1.3 Other effects

Convergent evolution

Figure 5.16 shows nice examples of how different runs often led to similar postures, strengthening the hypotheses of embodiment of section 6.1.1. Other states allowed for more variation as can be seen in Figure 6.5 where three different ways to sit where discovered by the agent independently.

Left-Right division/specialization

The agent seldom behaved symmetrically and often used one side in order to try to sit and the other one to roll from and to the belly. One curious thing is that it in almost all cases used its right side in order to get up and its left to get to and from the belly. It might just be a consequence of too few runs, but it can't be excluded that there might be some mechanical difference between the two sides behind it, since the body was the constant factor between the separate runs.

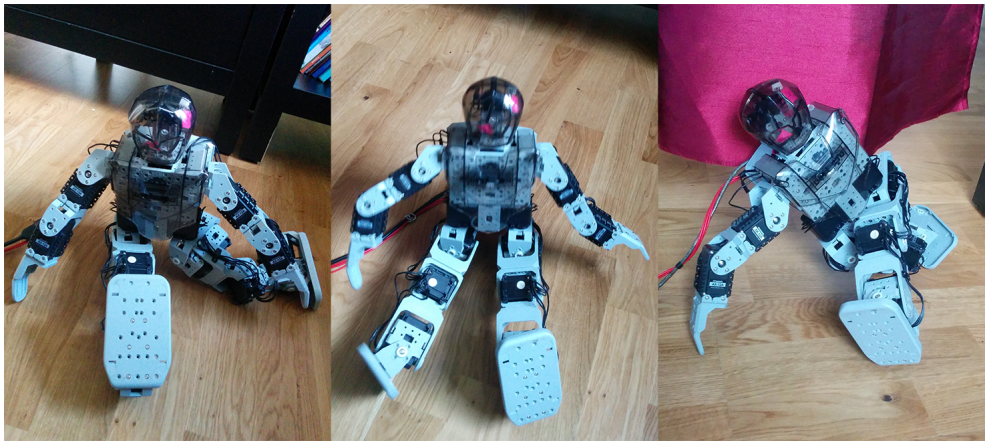


Figure 6.5: An example of a non-convergent evolution. Three different ways to sit, all found in different and independent runs.

6.1.4 Shortcomings in experimental method

Some shortcomings of this work includes

- Few test samples. Every setting was generally only allowed one run which might gives random variations large impact.
- Short training time. Even though one experiment allowed for a longer training time the general training time was relative short. It can't be excluded that some tested methods would have performed better, given more time.
- Cable obstacle. To communicate with the robot and supply it with power cables were inserted directly into it. These sometimes acted as obstacles so that some postures suddenly didn't work in some cases. On the other hand could this be beneficial too since it might have forced the robot to use more robust solutions in these areas.
- Training discontinuity. In order not to stress the servos too much the robot was only allowed to run in periods of 500 iterations, to then shut down and be able to cool down. When powered of it often changed its posture and when it woke up it would notice an unexpected movement.
- Noise from accelerometer readings. Partly because of limited precision, but sometimes also because the robot was in motion at the time of measure, even though the robot would freeze its position for a second before every reading. Both these events might lead the robot to make a false conclusion of what state it is in.
- Error in code. Since the code consisted of many interacting parts that were altered at numerous occasions as work progressed, it can't be excluded that some errors might still occur.
- It is unclear what significance the different charts measured during the experiments have. Number of visited states and state-actions are straightforward but doesn't tell much about how easily they can be reached again, mean probability of a failed transition seems in some ways be the best one, but there are still instances where agents with lesser values perform better on given tasks since the states they do know well are better connected with each other. Finally was the KL-divergence measure just plain weird. To begin with did the random actions produce as good KL-numbers as the AC-chosen ones. Could there be a problem with the implementation of it, or is this an effect of that the underlying probabilities of the Markov Map is also changing as the hyperSOM evolves? The measure of KL-divergence didn't add anything to the evaluation of different runs anyhow, even though the methods choosing action based on them performed significantly better.

6.2 Variations

6.2.1 Global Parameters

In this example the global parameters where the tilt of the torso and how stretched out the legs were. The tilt was really useful information that gave rise to many relevant behaviors, while the stretch out measure, not so much. In the future it should probably be worthwhile to get an accelerometer that could measure in 3 dimensions instead. Other global parameters that might be interesting to use in future applications includes:

- A ir-sensor measuring distance from the chest of the robot. Then would the robot be able to separate between lying on the belly and standing on all four.
- Sensors to indicate how big the loads are on its feet and hands.
- How high above the ground some point of the robot is.
- Angular speed.
- Distance moved forward in transition.
- Texture of ground.
- Proximity to some object.
- Inner compass.

6.2.2 Coordinate system variations

In this implementation the states were placed in a grids, both a cartesian and a spherical. In the spherical there might have been a conflict by the imposed structure and underlying hyperSOM grid. The motivation of the underlying hyperSOM grid is mostly to allow for the generalizations that helped finding the so far unseen states, but if one could find efficient ways to search for so far unseen states in a non-grid dependent fashion, then it would in theory be possible to place the nodes freely in GS. This would allow finer resolution in the more interesting area, or even free floating states that self organize to where they are the most needed. In such systems, and probably even grid-based systems such as the spherical, it could be a good idea to define the neighbors in a more general way. Two ways to do that could be to:

1. Define a number k of neighbors to every node. Transitions are then only allowed to the k closest nodes in GS.
2. Define a neighbor radius r_a. Transitions are then only allowed to nodes within that radius.

Whatever system one choses it is important to ensure that all nodes are connected into one cluster, so that the agent can attempt to reach any node from any other node, using the allowed edges.

6.2.3 Action chooser

Even though this RL was controlled by AC there are many other possible options, of which some was tested in this work.

AC(w=0)

This is the AC used in [13] which was the inspiration of the version of this thesis. A possible explanation why this method performs worse could be that it doesn't account for that a transition to a state changes that state, meaning that it could be relevant to look into other transitions to that state too. This wasn't the case in the original implementation so in that case it wasn't strange that this version were used.

RA

Random exploration was not a total disaster, probably since the underlying hyperSOM still evolved in an efficient way to preserve what the random explorer happened to stumble upon. Since some states are naturally more stable they seem to act as attractors to the robot, and since this agent didn't try to go back to the less stable areas it naturally had a hard time to find states such as sitting.

Other alternatives

Some other alternatives that have crossed my mind would be:

- A mass related method. Since mass reflects the probability of reaching states maybe changes in mass could be used in a similar fashion as the changes in P_S and P_{SA}?
- Connectivity based action choosers. The problem with the CA implementation is that it often gained control over some areas but then lost the links between the different areas of the state space that it did control. If one instead found a value of how likely the agent was to on average reach any state from any state, and then gave a reward proportional to how much that transition increased this number, that might help the agent become able to maintain control over larger parts of its state space when tested.

6.2.4 State searcher

As suggested in section 6.2.2 it could be good to have a state searcher that is not dependent on a grid structure. This was technically the case for the NGB hyperSOM, which looked for new states by deviating randomly from neighboring ones, but that method clearly found less states than the original GB hyperSOM. It could really be worthwhile to look for other methods for this important task.

6.2.5 Transition Record

In this implementation only the probability to reach a state was considered for the Markov Decision Process, giving a probability function $P(s_i, s_g)$ with i=initial, and g=goal. $P(s_i, s_g)$ thus gives the approximated probability that an attempted transition $s_i \rightarrow s_g$ is going to succeed. In the original implementation of MDPs this is not the function typically used, but rather one uses $P(s_i, s_g, s_f)$ that instead tells the probability that an attempt to go $s_i \rightarrow s_g$ would actually end in a state s_f. The advantage of this approach is that the agent is then able to evaluate the effects of failing to reach a state, so that states where failures leads to uninteresting areas of the state space gets less interesting than states where failures often leads to other states that are also interesting. Another advantage is that a finer division of the state-space can be made, since smaller states leads to more failures but it is then able to account how interesting an area of states are as a whole since the states closest to the goal states are the states where it is most likely to end up in. The reason why $P(s_i, s_g, s_f)$ wasn't used in this work was that it was thought that this matrix would be too large since it would have a dimensionality of N^3. However, if one uses sparse matrices and only allows for goal-states within some action-radius of the initial states, then the number of nonzero-elements of the matrix will be of dimensionality N^2. From the experience of this work, that should absolutely be feasible.

6.2.6 Number of direct states per node

Each state was only allowed to be represented by one direct coordinate $\mathbf{x}_i$, and this was the most important trick of reducing the dimensionality of the problem down to a feasible level. It definitely worked at some level, since the postures found by the robot seemed natural and effective. Problems however occurred when a state needed to serve as a bridge between many different types of postures, such as that an efficient posture of sitting was far away from an effective posture of rolling over to the belly, creating a conflict of how to best use the node. A solution to this dilemma could be to allow multiple states for every node, and introduce an additional global parameter that denote which layer of the solutions the agent wants to reach. The agent would then always reach the chosen layer, and could for example have one layer in order to reach a sitting pose and one to roll over with. This was however almost the case in the last experiment when the robot had three layers of how stretched out its legs were, and it didn't really help. A final idea, which at first glance seems bulletproof, is to allow one direct coordinate for every state-action, or transition. If the robot then stumbles over a way to go from A to B, then this posture would only be altered by other postures that do the same thing, so that a transition from C to B wouldn't spoil it. In this case an action A to B would probably be performed in two steps:

1. Go to the direct coordinate $\mathbf{x}_{A \rightarrow B}$.
2. If in B: Go to the direct coordinate of $\mathbf{x}_B$.

Then the agent would never forget how to do a transition, and the second step assures that the agent always starts in the same posture in state B, so that the Markov Decision isn't dependent of the previous state in the next transition. This would basically mean that every state-action would get its own node, and with such a change the search of new states must be performed in some other way than a GB hyperSOM.

6.2.7 Build/decay mass

To build and decay masses seems to a large extent have worked as intended but there are some variations of this that could be considered. One such is whether perhaps both the node the agent tries to reach, and the state from which it leaves should have their masses decreased when a transition fails. This would weaken direct coordinates such as the one where the feet got stuck in each other (Figure 6.2). Another thing to consider is what level of stochasticity should mass lead to. The current level seemed to work fine, but it is unclear what it dependence is on how small the states are, and to what degree the stochasticity might stop a direct coordinate of o node to converge, since the stochasticity made it miss often enough so that the mass couldn't increase enough for it to stop missing that much. One could also consider to freeze the mass and direct coordinates in periods, in order for the RL to update its probability distribution of what the hyperSOM is capable of at any certain point.

6.3 The bigger picture

What we basically have here is the first steps to a general controller of unknown systems, where general values can be measured and control is given over some basic controllers. It could be, as in this case, to teach a robot how to control its body. Within the same field it could even be implemented in robot building kits, such as Lego, so that users can build robots whatever way they like and then just turn them on and let them figure out by themselves how to operate their bodies. Once they learned that the user could tell them what to do, and they would do it in a way suitable to their given bodies. And if something breaks, no problem, they will then notice that some areas of their state space are not working as supposed and thus decay those areas and builds them up again, in a way suitable to their new morphology. It however goes beyond robots. All sorts of systems could be controlled using this method. Give the controller access to some of the electronics in the house and then access to the current temperature and temperature derivative, and let it learn how to use the electronics to create the wanted temperature at a later point. Just make sure it isn't given access to something dangerous such as the stove... It could even learn how to make wine glasses vibrate by being able to play frequencies and given how long time a sound resides once it stopped playing as a global variable. More practically maybe would be to let an agent figure out how to control an unknown servo using different pulses as direct coordinates, and the servo output as global coordinates. The servo could then in a future application be controlled by this agent, and work as a direct coordinate to an even higher agent.

7 Conclusion and Future Work

7.1 Conclusion

The method of this work shows a powerful new way to think about self-organizing high dimensional systems such as robots. As [1] admitted: "RL for high degree-of-freedom robot control is still an open issue."
In this work many of the problems that has faced previous works are completely gone, with a state-space of only 25 states and 2 hours of training a robot was able to perform tasks in a natural fashion that could in many aspects compete with the previous work where thousands of states were required together with days of training. The power of this solution appears to be in its abstraction of a Planner, an Interface and a Robot with a sensor of global coordinates (see section 3.2). The exact implementation of these parts worked good enough to prove the concept and show many desirable features, both when tested alone but also as a whole. There are however reasons to in the future look for other solutions for these. All in all the method displayed the following qualities:

- It is able to handle high dimensional robots as easy or more easy than low dimensional ones.
- It adapts to the body a robot is given, both in matter of shape but also in matter of number of servos.
- A robot is able to relearn behaviors if it is somehow damaged so that its functionality changes.
- It allows a physical robot to start displaying skills within hours.
- It is more or less coordinate independent. It learns to work with whatever it is given.

7.2 Future Work

Ideas for future work that would be interesting includes:

- Run simulations of the training to see what happens in the real long run. Will it eventually solve the whole state space, or is their an inherent limit for how much it is able to learn.
- Implement a solution where each state-action has its own mass, and the mass of a node is the sum of the masses of the state-actions leading to it. If a transition fails then only the mass of that state-action is decreased, so that a node that works well for some transition will keep that functionality and if it can't be used for some other transition, then the RL just have to learn that and try to find another way.
- Implement a solution where every state-action is represented by a node, so that the direct coordinate of a transition is only created as a mean of earlier successful transitions for that state action.
- Use the full probability matrix $P(s_i, s_g, s_f)$ and compare it to the reduced matrix of this work, that only remembered if a transition were successful or not.
- Implement the method on robots of completely different morphologies.
- Look for non-grid alternatives to hyperSOM.
- Examine the effect of different global parameters on the resulting behaviors of an agent.
- Train the agent to control some basic global coordinates. Then use these global coordinates as direct coordinates in order to control some even higher global coordinates. As an example: This robot could learn how to control the angel φ. Then use φ as a direct coordinate in order to control the global variable of angular speed.

References

[1] T. O. Akihiko Yamaguchi Jun Takamatsu. DCOB: Action space for reinforcement learning of high DoF robots. *DOI* **10** (2013), 327–346.

[2] J. Clune. Evolving Gaits for Physical Robots with the HyperNEAT Generative Encoding: The Benefits of Simulation. *EvoApplications 2013* LNCS 7835 (2013), 540–549.

[3] R. J. W. Jing Peng. Incremental multi-step Q-learning. *Machine Learning* **22**.1–3 (1996), 283–290. DOI: 10.1007/BF00114731.

[4] V. Z. Josh Bongard and H. Lipson. Robotic Introspection: Self Modeling (2006).

[5] D. B. D. Kenneth O. Stanley and J. Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life* **15**.2 (2009), 185–212.

[6] D. T. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. John Wiley & Sons, 2004, pp. 163–179. ISBN: 9780471666578.

[7] D. J. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003, pp. 34–35. ISBN: 0521642981.

[8] *MATLAB manual. Ordinary Differential Equations*. Version 7.8. Mathworks, 2008. URL: http://www.mathworks.com/access/helpdesk/help/techdoc/ref/ode45.html.

[9] *Memsic 2125 Dual-Axis Accelerometer (#28017)*. Version 2.0. Parallax Inc, 2009. URL: http://www.robotshop.com/media/files/pdf/memsic-2125-datasheet-28017.pdf.

[10] V. V. H. Pierre-Yves Oudeyer Frédéric Kaplan. Intrinsic Motivation Systems for Autonomous Mental Development. *Transactions on Evolutionary Computation* **11**.2 (2007), 265–286.

[11] G. M. Ralf Der. *The Playful Machine - Theoretical Foundation and Practical Realization of Self-Organizing Robots*. Springer, 2011. ISBN: 78-3-642-20252-0.

[12] *ROBOTIS e-Manual*. Version 1.25.00. ROBOTIS, 2010. URL: http://support.robotis.com/en/product_manual.htm.

[13] J. Schmidhuber. Curiosity driven reinforcement learning for motion planning on humanoids. *Frontiers in neurorobotics* **7**.25 (2014). DOI: 10.3389/fnbot.2013.00025.

[14] K. Sims. Evolving Virtual Creatures. *SIGGRAPH '94 Proceedings* (1994), 15–22.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998. ISBN: 9780262193986.

[16] M. Wahde. *Biologically Inspired Optimization Methods - An Introduction*. WIT Press, 2008, pp. 48–49. ISBN: 978-1-84564-148-1.