Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

TUHH
Technische Universität Hamburg

Neural Network-Based Control of a Humanoid Robot for Grasping Tasks

Master Thesis
at the Knowledge Technology Group, WTM
Prof. Dr. Stefan Wermter

Department of Informatics
University of Hamburg

in cooperation with the
Institute for Reliability Engineering
Prof. Dr.-Ing. Uwe Weltin

School of Mechanical Engineering
Hamburg University of Technology

submitted by
Niklas Widulle
on
5th July 2016

Examiners:	Prof. Dr.-Ing. Uwe Weltin
	Prof. Dr. Stefan Wermter
Supervisor:	Francisco Cruz

Niklas Widulle
Matriculation no.: 20837931
Kasernenstraße 21
21073 Hamburg

Erklärung der Urheberschaft

Ich versichere an Eides statt, dass ich die vorliegende Master Thesis selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Abstract

A neural network based controller for a humanoid robot arm is designed. The basis of the controller is an inverse kinematics solution, which is learned by a multilayer perceptron network. Two learning strategies are implemented, optimized, and compared: goal and motor babbling, each of them for both a three-dimensional and a six-dimensional task space which includes the orientation of the end effector. Additionally, a solution for relative inverse kinematics is implemented and tested. Training of the network is done on the basis of an automatically generated forward model of the manipulator. To allow for linear and point-to-point movements, basic trajectory generation is implemented. The practical usability of each method is tested in simulation. This is verified on the real robot.

Three dimensional goal babbling has a consistent learning behavior, but struggles with the influence of redundant configurations. With goal babbling, a better accuracy in the center can be achieved, but the method has problems exploring the periphery of this very complex workspace. Using six task space dimensions, the accuracy of motor babbling is improved. Goal babbling also benefits from the additional orientation information. The relative inverse kinematics solution has a high error, likely due to the very large input space, and can only be used with additional corrections. Since defining achievable rotation coordinates for the under actuated manipulator proved very difficult, the three-dimensional methods are better in practical use.

This thesis contributes to current research as it provides a thorough comparison of motor and goal babbling in a practically relevant scenario. Additionally, it proves that using goal babbling is possible in a six-dimensional task space.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robots are already widely used in industrial applications such as logistics and production. In order to use robots in the home or office space new concepts are needed for both the shape and the control of robots. Because the environment of our daily lives is designed by humans and for humans, robots that mimic our human bodies, so called humanoid robots, are best at interacting with it.

To contribute to the fields of cognitive robotics and human-robot interaction University of Hamburg's Knowledge Technology Group (WTM) is developing a robot called Neural-inspired companion (NICO). It is based on the Nimbro-Op platform developed by the University of Bonn, but equipped with custom arms and the head from the I-Cub robot [48, 3, 30]. The robot arm has five degrees of freedom, three in the shoulder, one in the elbow and a rotating wrist. It is equipped with a gripper as its end effector.

Grasping items requires the ability to move the end effector of the arm to a predefined position and orientation. While calculating the position and orientation of the end effector from the joint angles is a simple forward transformation, computing the required joint angles for a desired position and orientation is a far more complex problem known as inverse kinematics (IK). For some special robot configurations used in industrial robots this problem can be analytically solved [34]. For the general case numerical solutions do exist. All of these methods require precise knowledge of the manipulator and cannot adapt dynamically to changes of the structure. These can occur, for example, when a motor is damaged or structural parts are bent. Soft computing methods based on artificial neural networks (ANNs) have been proposed and applied to the problem. Inspired by the way infants learn to control their body, random movements are performed to learn a model of the relation between the position in task space and the joint positions [42].

Two main methods exist for learning the IKs of a robotic manipulator without a priori knowledge abouts its kinematics. The older one is called motor babbling:

random movements are performed in the joint space and the resulting end effector positions are observed [46]. This mapping can then be learned by any regression method. A more recent development is goal babbling: random target directions are chosen, which the robot tries to reach, while already relying on the model in order to do so [40]. In order to prevent uncontrolled drift, movements are kept inside a sufficiently known region. This region expands with time as the robot is exploring the workspace. The main advantage of goal babbling is that it scales well with additional degrees of freedom in the manipulator. This is because, unlike motor babbling, it does not need to explore the whole joint space but rather explores the whole task space. So far, goal babbling has only been used to learn IK in two or three goal dimensions [41]. The orientation of the end effector has not yet been included. Since controlling the orientation of the end effector is useful for grasping tasks, the learning scheme is extended appropriately.

If some positions of the workspace can be reached by multiple joint configurations, it poses a serious problem to the learning method, since the examples seemingly contradict each other. Those positions are frequent if the manipulator has some form of redundancy, which is the case when the manipulator has more degrees of freedom than the task space. NICO's arm has five degrees of freedom, which means that for positioning the end effector it has redundancy, which can affect the accuracy if it is not dealt with. If the orientation of the end effector is included in the control scheme, the manipulator has less degrees of freedom than is needed to achieve motion in all directions and redundant positions do not exist. Goal babbling deals with redundancies by weighting the effectiveness of each movement. For methods including the orientation of the end effector the task space has six dimensions and redundancies are not an issue.

The data can be generated by either the forward kinematics solution, an appropriate simulation, or using the real robot and a measuring system for the end effector position. Using the forward kinematics is computationally cheaper than a simulation and is therefore the fastest method overall. A simulation can include additional effects such as collisions and, if all physical properties are modeled correctly, displacement. Using the real robot inherently includes all possible effects, but severely limits the amount of data that can be generated, as it requires real movements. Moving the end effector requires some form of trajectory generation. The positions of all required joints for each time interval needs to be generated according to the desired end effector motion. Which path should be taken is a problem known as path or motion planning and is additionally dependent on the task and the environment [49].This can be done by either a higher level controller or an operator.

In a previous work, a solution for the inverse kinematics of the arm based on goal babbling has been implemented. This solution focuses on the position of the end effector and therefore did not include movement of the wrist joint of the robot arm. No consideration was given to multiple joint configurations and their negative effect on the precision of the solution. The joints were restricted in such a way that only a small subset of the workspace is available [3].

The main objective of this thesis is to provide a controller which positions the robot's hand correctly at known coordinates in order to be able to successfully grasp objects. This requires a solution to the IK problem. As both motor and goal babbling can be applied to this problem, they are compared in their performance and applicability. Primary research questions which arise when solving the IK problem for this robotic manipulator are:

- How can the goal babbling approach to compute inverse kinematics be extended to the six-dimensional task space?
- How well does the goal babbling approach scale with additional dimensions and how does it compare to a motor babbling approach?

The main piece of the controller which is designed is the NN-based IK solution. For training the NN, data is generated by the forward kinematics that are automatically derived from NICO's design description. Motor and goal babbling in both three and six dimensions are implemented, optimized in their parameters and compared in their performance and accuracy. Additionally, a relative IK solution is implemented and compared the same way. A basic trajectory generation solution is implemented. The IK solutions generated by the different methods are compared in their practicality by using them to control the robot in simulation. Ultimately the solution is tested on the real robot to perform a grasping task.

This thesis contributes to current research as it provides a thorough comparison of the performance of motor and goal babbling in a real world scenario. Additionally it shows the possibility of incorporating the orientation in a goal babbling approach.

Chapter 2

Literature Review

2.1 Towards Humanoid Robots

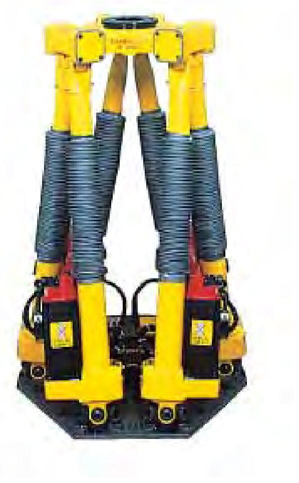
Even though robots are commonly used in industrial environments, referenced in various media and are now even starting to become relevant in our everyday lives no single definition for the term ”robot” exists. It was coined by Czech author Karel Čapek and derived from the term *rabota*, meaning ”forced labor” [12]. The lack of a universal definition is largely caused by the multitude of different robot types. They all have in common that the robot can perform tasks. The nature of these tasks, the complexity and level of independence vary widely. Common categories of robots include industrial robots (IRs), service robots and humanoid robots.

IRs serve to automate various tasks in the modern factory. Most commonly, they are used to handle workpieces, for welding, assembly and dispensing [21]. Advantages of robots in these applications are their high flexibility, low cost and high precision. While normal machines are build for a specific purpose, industrial robots are able to perform a multitude of functions and can therefore easily be reused and repurposed. Depending on the task a number of different kinematics are used. The most common are articulated, SCARA, parallel and delta (see fig. 2.1). New developments in the field of IR include human-robot-cooperation [18], robotic milling [56] and adaptive control [6]. Especially for human-robot-interaction and adaptive control many machine learning (ML) and NN based approaches can be used[36] [45].

Service robots perform tasks which would normally be done by humans. They are especially suited for tasks which are dangerous, repetitive or time-consuming. Service robots are used in the military, disaster recovery, nursing and household applications, mowing and vacuum cleaning. Currently available service robots can usually only perform a specific task. As most of these tasks are done in human vicinity, safety is a concern. This is often addressed by using small robots which

(a) Kuka Kr 60-3 [25]

(b) ABB IRB 360 [27]


(c) Fanuc f200-iB [13]

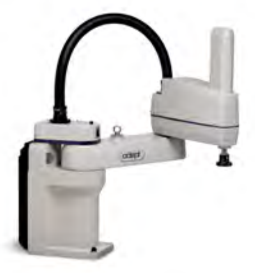

(d) Adept Cobra S 600 [1]

Figure 2.1: Different types of industrial robot manipulators: (a) articulated (b) delta (c) parallel/hexapod (d) SCARA

are intrinsically safe, though this limits uses. Since service robots are often mobile and act within unknown and dynamic environments, localization and mapping are necessary. ML has for example been proposed and used for how the robot performs the given task [32], for localization, mapping [31] and interaction [15].

A humanoid robot is built to emulate the body of a human being. This can enable it to make better use of environments and tools designed for human use. Unlike most service robots a humanoid robot can be multipurpose and handle a multitude of tasks and situations. A humanoid robot which mimics the appearance of a human being is also called an "android". This can help to improve acceptance by humans as we anthropomorphize targets of communication, as such androids are ideal for nursing and other activities which require communication with humans [22]. Human safety in this man machine collaboration is again an issue. Different methods of collision detection and force/torque limitation have been proposed and are being used. Limiting the size and power of the robot is a simple but constraining solution to the problem [54].

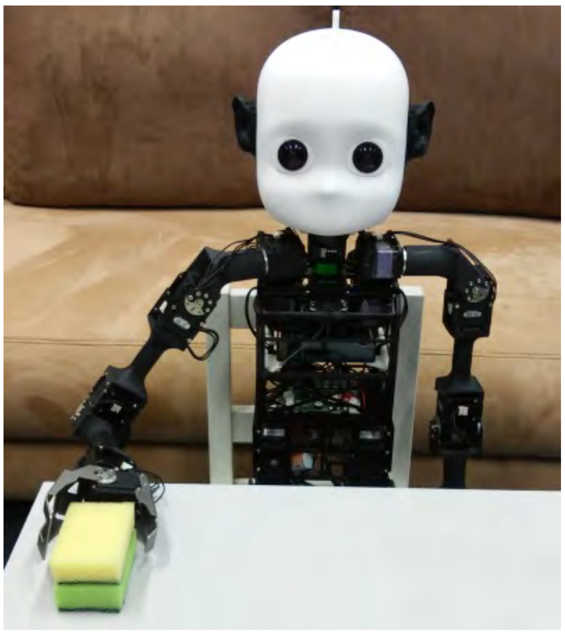


Figure 2.2: The NICO robot

Currently available humanoid robots are very constricted. This applies to both their mechanical dexterity and their cognitive capabilities, which limits their usefulness in all applications. Unlike their industrial counterparts humanoid robots are mainly subject of scientific interest than of practical use. Using neural network based control can help improve the dexterity of robots and can thereby provide a step towards humanoids supporting us in our everyday lives.

2.2 NICO Humanoid Platform

Neural-inspired companion (NICO) is a humanoid robot platform (see fig. 2.2) being developed by the WTM at the University of Hamburg. Its body is based on the Nimbro-Op platform developed by the Autonomous Intelligent Systems Institute (AIS) at the University of Bonn [48]. Nimbro-Op is designed as a soccer robot to take part in the RoboCup competition. It is teen-sized, allowing for greater physical capabilities than kid-sized robot platforms such as Nao [16] while remaining comparatively affordable. At its size, it can also still be considered inherently safe, the robot does not have enough power to do serious harm to humans. For actuation the platform uses Dynamixel servo motors manufactured by Robotis Limited. Because the primary focus is soccer, the robot's arms have a rather simple design, with only 3 degree of freedom (dof). Its hands are rigid, useful mainly for maintaining balance and to stand up. Instead of the regular head of the Nimbro-Op platform NICO uses the head of the iCub platform [30].

The arms are replaced with a five dof custom design [3]. This enables the robot to also reach in front of its torso and to rotate its end effector. As end effector a gripper is mounted to the robot arm, it is a simple one dof solution for grasping. It is limited since it does not include sensory feedback and has no fingers. The design of joints of the arm is based on the structure of a human arm. Unlike humans, it can also move its elbow in both directions. This structure has the problem that its IK and workspace are not easily defined (see section 2.3.1). With NICO being a research platform the aforementioned limitations provide challenges which can be overcome by the use of NN. Working with NICO can widen our knowledge and understanding of robotics and intelligent systems.

2.3 Fundamentals of Robot Control

Robot control is a rather new discipline which has many aspects and is still evolving. Because of the electro-mechanical nature of robots, control requires an interdisciplinary approach which covers many fields of research. With NN also being a rather recent discipline, the scientific community is conducting a wide range of experiments with NN to find its applications in the field robotics.

The topic of robot control consists of many different problems and layers. Depending on what type of robot is to be controlled, only some need to be considered. Since the focus of this work is to enable grasping with a humanoid arm only some are relevant. These include trajectory generation, forward and inverse kinematics and kinetics of manipulators.

Forward kinematics describe what position the end effector is in given the position of the manipulator joints. Inverting that problem is to ask for the needed joint positions in order to reach to a given end effector position. For a serial manipulator such as a humanoid robot arm the latter is a much harder problem. In order to find solutions a mathematical description of frames and transformations is needed. Detailed discussion of these topics is included in section 2.3.1.

The manipulator kinetics extend the kinematics by introducing the dynamics. For this reason the manipulators' mass and inertia need to be considered (see section 2.3.2).

In order to move the end effector of a robot manipulator, it is advisable to plan a trajectory consisting of the speed and acceleration of the end effector at every given time step. This planning however should consider the manipulators joint kinematics and kinetics as well as optimization goals such as minimal time or energy. This is detailed in section 2.3.3. A related problem is motion planning: this more high level approach aims to find a suitable or optimal motion given a certain task and is often considered in the field of mobile robotics.

2.3.1 Forward and Inverse Kinematics of Manipulators

This introduction follows roughly [9] in both notation and structure, because it can be considered the standard reference on this topic.

The position and orientation of an end effector can be described by a frame, which sets it in relation to a world or origin frame. A frame can be described by a transform T. This transform is a 4×4 matrix, containing a rotation matrix R and translation vector P. For example the frame $\{A\}$ could be defined by a transform from the origin frame to frame $\{A\}$.

$$ {}^{o}_{A}T = \begin{bmatrix} {}^{o}_{A}R & {}^{o}_{A}P \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

Additionally to defining a frame, a transform can also be used as a transform mapping which changes in which frame a point is defined and as a transform operator which creates a new point relative to a given point.

Transforms are highly useful because they can be linked together. For example, the transform from $\{A\}$ to $\{C\}$ is given by:

$$ {}^{A}_{C}T = {}^{B}_{C}T \, {}^{A}_{B}T \tag{2.2}$$

A serial manipulator structure, such as a humanoid robot arm, consists of a number of fixed links which are connected by joints. These fixed as well as variable parameters can be put into the structure of transforms with the help of the Denavit-Hartenberg notation [11]. Once the individual transformations are known the overall forward kinematics can be calculated by linking these transformations together. It is to be noted that this transformation only depends on the variable joint parameters θ.

The relationship between the speed of the joint parameters $\dot{\theta}$ and the speed of the end effector v_{ee} is defined by the *Jacobian* matrix J.

$$ v = J(\theta)\dot{\theta} \tag{2.3}$$

J is calculated using the knowledge that the position s_{ee} of the end effector is defined by the transform derived earlier. The partial derivatives of the position with respect to the joint parameters for the different linear and angular directions define the Jacobian matrix [5]. The Jacobian matrix is very important for iterative inverse kinematics as well as kinetics.

$$J(\theta) = (\frac{\delta s_{ee}}{\delta \theta}) \tag{2.4}$$

Because of eq. 2.3 it is clear that the Jacobian matrix defines in which directions a manipulator can or can not move. For example, a fully stretched out articulated robotic arm loses its ability to move further outwards, but a manipulator can also lose parts of its mobility in other configurations, when joint axes align they no longer offer unique movement directions. These positions are called *singularities*. They can be found by checking the determinant of the Jacobian matrix: if it is zero, the matrix loses rank and a movement direction is no longer available. When moving across the taskspace it is therefore advisable to avoid singular configurations. Otherwise it can result in erratic behavior of manipulator.

Even though the forward kinematics can be worked out easily and a single transform can be inverted without problems, general IK still poses a significant problem. This is due to the nonlinear nature of the IK function. A solution only exists if the robot can reach the given position and orientation. These positions together form the workspace of the robot. Depending on the kinematics of the manipulator, the workspace can be complicated especially when it comes to the achievable rotations. This, additionally to the easier IK and mechanical considerations, is one of the reasons why only a small number of different kinematic structures are used in IRs. For example, standard 6-dof IRs have a large dextrous workspace in which most orientations are reachable while delta robots have a simple three-dimensional Cartesian workspace which can then be augmented by adding joints for rotation [8]. A simple workspace makes task planning easier as the robots capabilities and limitations can be better understood.

There are a number of different approaches to solve IK. Traditionally there are the closed-form algebraic solutions. Under some additional assumptions concerning redundant configurations and only for some manipulator structures these solutions are available. Well known is Peipers' solution for 6-dof robots with a spherical wrist [35]. Most commercially available IRs use this manipulator structure.

Additionally to the algebraic solutions there are also iterative approaches. They are usually used when the closed-form solution is not available. Because of the iterative nature these approaches are problematic when a solution needs to be made available to a robot controller in real time. Another issue is that, unlike closed-form solutions, accuracy is a concern.

Iterative methods use forward kinematics to determine the current position s_{ee} and then apply a step to the joint parameters $\Delta\theta$ to get closer to the goal position t_{ee}.

$$\theta := \Delta\theta + \theta \tag{2.5}$$

Choosing an appropriate $\Delta\theta$ is the main objective of an iterative IK algorithm. The step to the joint angles can be applied to the actual robot, or it can be fed back into the algorithm until the error is below a certain threshold [5]. Going back to eq. 2.3 the movement after a change in the joint variables can be approximated by:

$$\Delta s_{ee} \approx J\Delta\theta \tag{2.6}$$

Trying to solve this approximation for $\Delta\theta$ is thereby a reasonable strategy. This is however problematic, as the Jacobian matrix needs to be invertible in order to solve the equation. Generally though, it is not invertible. The *Jacobian transpose method* substitutes the inverse with its transpose which is computationally cheap [53]. The *Jacobian pseudoinverse method* uses the Moore-Penrose inverse as the replacement. This approximation of the inverse guarantees that the error is minimized in the sense of least-squares optimality [24]. Other methods include a damping factor for stabilizing the movement near singularities [7], or use the singular value decomposition [29], [5].

2.3.2 Kinetics and Robot Control

Accurate and fast movement of a robotic manipulator requires consideration of not only its kinematics but also of its dynamics. Based on the principle of virtual work the relationship between the torques at the joints τ and the forces and torques at the end effector F_{ee} can be derived from eq. 2.3.

$$\tau = J(\theta)^T F_{ee} \tag{2.7}$$

Using the Lagrange's equation of motion the relationship between the joint torques and the joint velocities and accelerations can be described using eq. 2.8.

$$\tau = M(\theta)\dot{\theta} + V(\dot{\theta}, \dot{\theta}) + G(\theta) \tag{2.8}$$

Here M is the mass matrix containing the inertias of the manipulator, V contains the Coriolis- and centrifugal forces and G is the vector of the gravitational forces. With help of the Jacobian this can even be transferred into a task space notation. However, eq. 2.8 in itself makes clear why many robot control algorithms rely on direct access to the joint torques, as this allows to direct control over the manipulator dynamics [33]. NICO uses servo motors which include their own control loop, leaving only the joint position available as input. As it may be disadvantageous for advanced control schemes such as, for example, force control [55], it has a number of advantages. Positioning the robot arm is very simple. Movements do not necessitate complex trajectory generation, as the controllers inside the motors will automatically generate accelerations and velocities needed to achieve the given position. The proportianal-integral-derivative (PID) controllers inside the servomotors will operate more precise if their parameters are adapted to the weights they move. As long as the control loop of the servo motors is stable and no further feedback is applied, overall stability is guaranteed [2].

2.3.3 Trajectory Generation and Motion Planning

Trajectories need to be generated for a joint to describe a position at time step that allows to reach a goal position. Once the positions at each time step are calculated the needed velocities and accelerations to achieve these motions can be calculated by differentiation. A trajectory can then be described as a function of the form:

$$\begin{aligned} &\theta(t) \\ \theta(t_0) &= \theta_s \\ \theta(t_1) &= \theta_f \end{aligned} \tag{2.9}$$

Where $\theta(t)$ describes the joint position at time t, θ_s is the starting position and θ_f is the end position.

Typical robotic movements are either done in the joint space or require a certain, often linear, movement in the task space. A motion defined in joint space is considered if only the initial and final configuration is relevant, or if the joint space movements have been defined by previously teaching the robot. This is often done for industrial robots: a user guides it through a set of movements, it can then repeat these indefinitely [49]. If the movement is done in task space, the trajectory planning can also be done in task space. In this case, eq. 2.9 does not describe the movement of a joint but rather the end effector motion. Followed by this, the joint movements are derived using the inverse kinematics.

The trajectory should adhere to certain physical limitations of the device. Because the torque of the actuators is limited, the planned accelerations should also be. The speed of actuation is also often limited. For a smooth movement there should also be no discontinuity in acceleration to avoid jerk. This can be achieved by connecting polynomial functions. Each polynomial is calculated based on the required position, speed and acceleration at the start and end point. This requires a fifth-order polynomial. If third order polynomials are also used, the acceleration is not defined [9]. Another method is to connect linear functions with parabolic blends: intended linear motions are defined and connected via parabolic blends, which smoothen the transition. Because of this, the actual via points are often not reached exactly but merely approximated. When moving an industrial robot the user often has to specify how closely he wants the via positions to be approached, sacrificing move speed for precision [17].

Motion or path planning is used to search for a possible and ideally optimal way to perform a movement based task [26]. Because of the complexity of the problem, this is often done by the user. To achieve an autonomous robot however, it needs to be solved automatically. A part of it which is largely independent of the task is obstacle and collision avoidance. The difficulty of finding an appropriate path lies in the obstacles that need to be avoided. These can be objects, but also self collisions and the manipulators singularities. One proposed method is to define repelling potential fields around the obstacles and attracting ones at the goal. This changes the problem to finding a global minimum. Further difficulties are avoiding local minima [4] and constructing the potential fields such that only one global minimum exists. Another method is called ”probabilistic roadmap planner”. It generates a map of nodes across the workspace and then connects those that are known to be collision free. A movement is created as a path between nodes [23].

2.4 Machine Learning Approaches

Machine learning aims to teach computers without explicit programming. When machines are faced with either unexpected, unknown or too complex situations, explicit programming can fail. This is particularly relevant for autonomous agents or robots, as they have to adapt to unknown environments. Because animals and especially humans possess the ability to learn, approaches to machine learning are often biologically inspired.

2.4.1 Artificial Neural Networks

Artificial neural networks are models used for machine learning. They are inspired by biological neural network (NN) though they do not try to be perfect copies of them. Rather, they mimic certain aspects of biological neural networks and put them in framework which is useful for the application is question.

A simplified biological NN is shown in fig. 2.3. Its main components are:

- Soma or cell body, which perform the calculation
- Axon, the output channel of the neuron

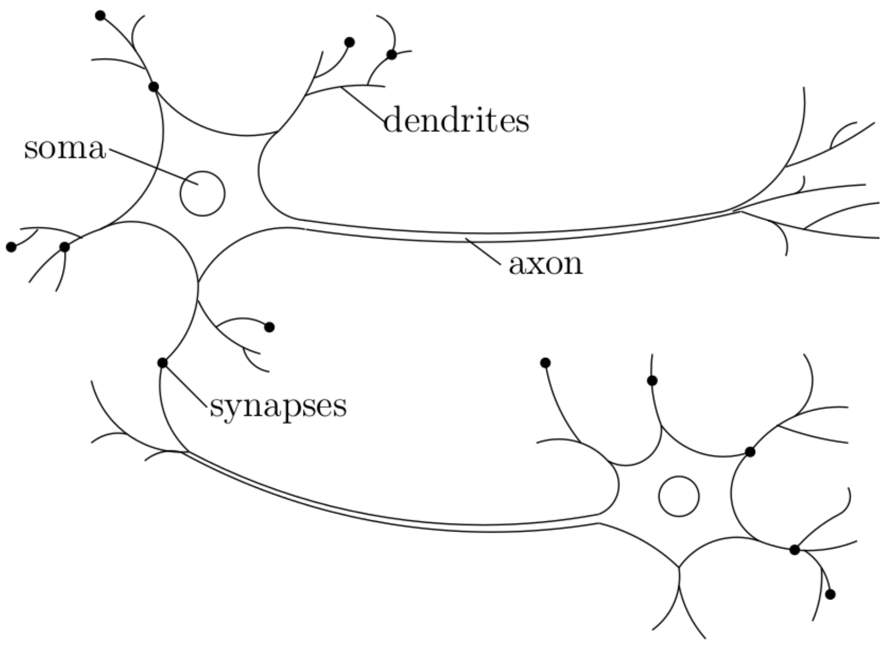


Figure 2.3: Biological neural network [51]

- Dendrites, which are the inputs
- Synapses, which connect the dendrites to another neurons axon

In artificial neural networks, the neurons a have similar structure. They are connected with a number of inputs and perform a calculation on that input with their *activation function*. This output can be connected to the inputs of other neurons [39].

2.4.2 Single-Layer Perceptrons

Perceptrons are amongst the oldest neural network inspired algorithms [44]. The word "perceptron" can refer to both a single neuron or a single layer feed-forward network.

A single neuron perceptron first multiplies each input $x_1..x_{n-1}$ with its associated weight $w_1..w_{n-1}$, then sums up its inputs. Additionally a constant input $x_n = 1$ serving as a bias term can be introduced, which also has a weight w_n associated with it. Based on this result the activation function f is performed and the output y is computed as:

$$y = f(\sum_{i=1}^{n} w_i x_i) \tag{2.10}$$

The simplest activation function is a threshold, which if the input is over a certain value output 1, and -1 otherwise.

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{2.11}$$

The perceptron learns by updating the weights. Because of the introduction of a weighted bias term the threshold value can always be chosen as zero. Learning is done by comparing the output of the perceptron with the real value t in a training set. The error, multiplied by a learning rate $\{\alpha | 0 < \alpha \leq 1\}$ and the associated input is subtracted from the current weight to give the new weight.

$$\hat{w_i} = w_i - \alpha(t - y)x_i \tag{2.12}$$

If multiple outputs are required, more neurons are needed. In the mathematical framework this makes the output a vector of size m. The weights can be stored in a matrix of size $n * m$. Each output can be computed with:

$$y_j = f(\sum_{i=1}^{n} W_{ij} x_i) \tag{2.13}$$

2.4.3 Multilayer Perceptrons

This configuration can learn any logical function as long as they are linearly separable [52]. If they are not, the model needs to be augmented with another layer of neurons. This is called a multi-layer perceptron (MLP). Because of the complex structure of the workspace for which the IK are to be computed, a MLP is used.

In a MLP the inputs to the last layer of neurons are the outputs of the layer before that, the so-called "hidden layer" (see fig. 2.4). This leads to the problem that the influence of the weights of the hidden layer is not direct, eq. 2.12 can no longer be used to update them. As a solution the backpropation algorithm is introduced [52]. It feeds the error backwards though the network to find the influence of each weight on the error. For this it needs to compute a gradient, which requires the activation function to be differentiable. As a replacement for the discrete step any continuously differentiable sigmoid function such as the logistic function or the hyperbolic tangent (tanh) can be used.

A MLP with one hidden layer with a finite number of neurons can learn any continuous function [19]. Because IK is generally not continuous, a MLP can not

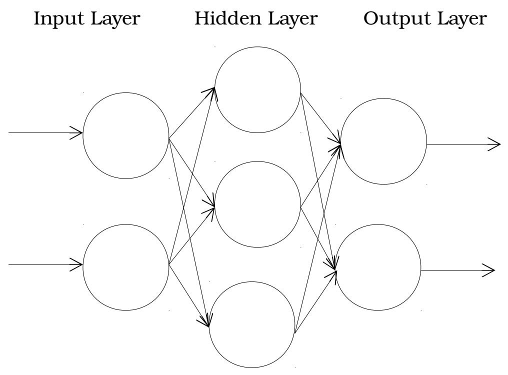


Figure 2.4: A multilayer network.

always find a precise approximation. As discussed in section 2.3.1 non continuities arise at singularities and near the workspace borders. Avoiding singularities is beneficial for robot control, but this cannot necessarily be said for the boundaries. Problems might arise if the errors of high gradients near these influence the performance in the other regions.

2.4.4 The Backpropagation Algoritm

The goal is to find a set of network parameters that minimize the squared error E of a given set of examples Z.

$$\hat{W} = \arg\min_{W}(E(W, Z)) \tag{2.14}$$

Locally around W it can be expanded into a Taylor series. This can be used to find the direction in which to move in order to minimize the error. The most common way is to only use the first element of the Taylor series, the gradient, in order to compute the *steepest descent* towards a minimum. Other methods also include the Hessian matrix, which is the second element of the Taylor expansion.

One problem of this method is that convergence is only guaranteed towards a local minimum. One method of avoiding local minima is to introduce additional noise into the the system by computing the gradient and updating the weights only based on single example. This method is called stochastic gradient descent (SGD) as it relies on random samples. While the computed gradient might point into a wrong direction for one sample, the high amount of samples will lead to convergence. A compromise between stochastic and regular gradient descent is offered by using mini batches. There the updates to the weights are computed from small batches.

Learning methods can also be distinguished by being performed online or offline. Offline methods use a finished dataset to perform their updates, while online methods generate their data while learning. Because it only needs one sample at a time SGD is referred to as an online learning method, though it can also be used on already completed datasets.

2.5 Learning Strategies for Inverse Kinematics

Traditional IK require knowledge of the structure of the robot. Humans initially do not have this and are still able to learn how to move. A robot may use a model of its own structure, but with time its configuration might change. For example, a motor might take damage or wear on the structure might bend its links. Learning IK requires data of the mapping from the task space to the joint space. Different strategies have been proposed on how to generate the needed data.

2.5.1 Motor Babbling for Learning Kinematics

One approach for data generation is motor babbling. Random motor commands are issued and the resulting end effector position is observed and recorded. In the case of servo motors where the motor commands are joint positions this is equivalent to issuing random motions in joint space and observing the resulting positions in task space. These methods have been applied with success to humanoid robots [10] [46].

The generation of the data can be done using a known forward kinematics, or in a simulation environment or on a real robot. Depending on which variant is chosen, different effects can be part of the learning procedure. A simulated environment makes it possible to include collisions, gravity and servo motors. The accuracy of these then depends on the simulation. Using a real robot for learning includes all possible effects and, additionally, also errors coming from the sensors used to measure the end effector position. The amount of data needed in order to generate the model limits use. If generation of a model requires years of real time movement, using the real robot is not an option. The simple forward model can instead be employed: It computes example data faster than the other approaches by magnitudes.

2.5.2 Goal Babbling for Learning Kinematics

Goal babbling is a newer approach for learning IK. Instead of giving random commands in joint space, these are issued in task space. This already relies on the model which is to be learned. The main advantage of this method is that it is largely independent of the number of joints the manipulator has. Motor babbling suffers from the *curse of dimensionality*, while goal babbling scales well with additional degrees of freedom [42].

While older goal babbling approaches still needed a-priori knowledge of the workspace, the newest can explore the workspace on its own[40]. That approach works as follows:

- A comfortable home position s_{home} is chosen
- The model is overfitted on the home position, initializing the network in the process.
- A random target space direction Δs
- The target position is then updated according to 2.15

$$s'(t) = s'(t-1) + \epsilon_X * (\frac{\Delta s}{||\Delta s||}) \tag{2.15}$$

The model guesses a move command which will put the end effector in the denoted position. The model has the parameters W, which in case of a NN refer to the weights of the network. A noise term e_θ is added in order to generate novel outcomes.

$$\theta(t) = g(s(t), W) + e_\theta(t) \tag{2.16}$$

The resulting real position is observed and used to update the model. The update is weighted in such a way that efficient movements get a higher impact [40]:

$$w_e = \frac{||s(t) - s(t-1)||}{||\theta(t) - \theta(t-1)||} \tag{2.17}$$

Some implementations also weight how much the intended move direction differs from the real one [43]. This gives consistent movements a higher weight and thereby help to deal with redundancies.

$$w_d = \frac{1}{2}(1 + \cos(s'(t) - s'(t-1), s(t) - s(t-1))) \tag{2.18}$$

The total weight of an example is then given by:

$$w_t = w_e * w_d \tag{2.19}$$

A new goal direction is chosen once a boundary of the workspace is detected. Given a perfect IK solution, the robot would only deviate from its path once it hits a boundary. Because the model is not perfect deviations will be common. To indicate boundaries the following rule is used:

$$0 < (s'(t) - s'(t-1))^T * (s(t) - s(t-1)) \tag{2.20}$$

Once the angle between intended and real movement is bigger than 90 the scalar product becomes negative and a new direction is chosen. This will also happen when the movement is further outside of its known region, keeping the explored region small at the beginning and then slowly expanding it. Additionally to the random movements, when a boundary is detected with a certain probability the robot returns to its home position. This movement is done in joint space.

$$\theta(t) = \theta(t-1) + \epsilon_\theta * (\frac{\theta_{home} - \theta(t-1)}{||\theta_{home} - \theta(t-1)||}) \tag{2.21}$$

This movement is meant to regularize the inverse, and stabilize the solution [37]. Goal babbling is an approach in which the next explored solution depends on the state of the model. This can lead to behavior in which the model becomes unstable, and tries to move the robot towards unreachable positions from which it cannot learn anymore. Oscillating between positions is possible, as well as getting stuck in a specific region of the workspace. The home movement is meant to help prevent that. The mentioned causes of instablity are of course additional to any in the learning algorithm: gradient descent can also become unstable or oscillate.

Successful implementations of goal babbling have so far been primarily in simulation. It has been shown that goal babbling scales very well with up to 50 dof in the manipulator [43]. The implementation uses a two dimensional task space. The robot is composed of links of the same length connected by rotary joints. This massively redundant setup makes finding a unique solution hard, goal babbling achieves that with the help of the efficiency weighting in eq. 2.17. In this two dimensional case, a precision in the millimeter range was achieved after $10^5 - 10^7$ movements, depending on the learning parameters. A movement here means not one data point, but one move direction, with 25 data points each. In the more recent approach of goal babbling with workspace discovery it is explicitly compared with motor babbling in terms of workspace discovery. Motor babbling is deemed significantly worse than goal babbling if there are more than 10 dof in the manipulator [40]. The underlying model is in all the mentioned work a local-linear map, which is trained using stochastic gradient descent.

The first implementation of goal babbling on a real world system learns the IK of robotic trunk [41]. This has the effect that the forward kinematics are now subject to noise because of the motors. Also the measurement of the end effector position does introduce additional noise to the system. The solution is still capable of controlling the system to a precision of around 2 cm in the central regions of the workspace and over 10 cm at some outer positions.

A previous implementation of goal babbling for the NICO robot has used it successfully in three dimensions to find a solution to the IK problem [3]. Using a neural network of 10 hidden neurons and a high learn rate it is possible to find a solution with an average error under 2 cm. However, only a portion of the actual workspace is considered, as the joint limits are reduced. This not only dramatically decreases the size of problem, but also linearizes it, as the influence of each joint declines. The positions of the test set are spread out across the workspace in an unknown pattern. It is also shown that a single layer perceptron can learn the IK with only slightly worse precision. This in conjunction with the low number of neurons in the best solution gives rise to the assumption that the model learned is rather linear. This is in line with [42], where a goal babbling approach was used on a limited portion of the workspace for the arm of a Honda humanoid robot.

Chapter 3

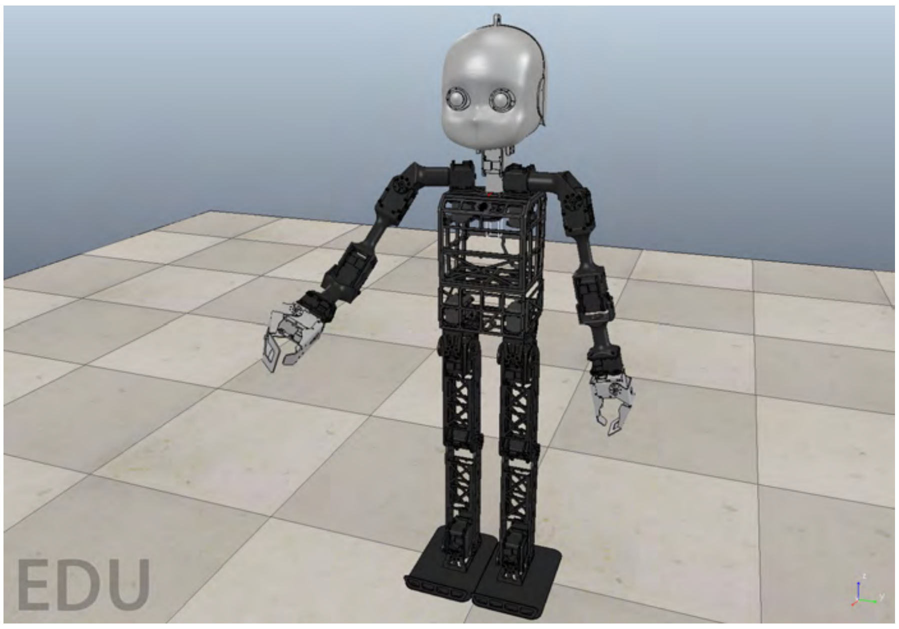
Proposed Approach and Setup

3.1 System Overview

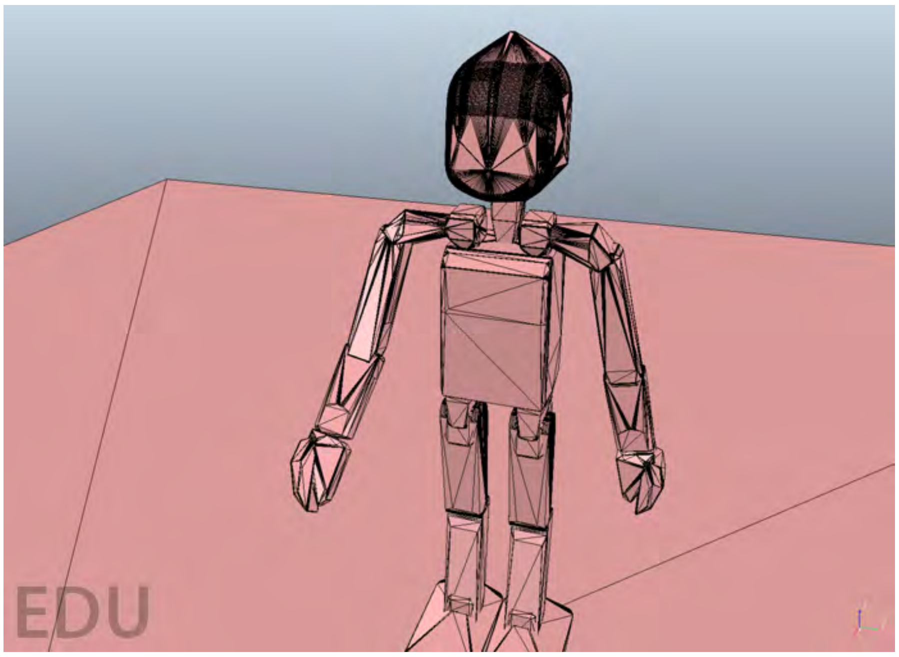
The most important part of the system is the NICO robot. The other parts are chosen to offer best possible compatibility with it. NICO is controlled by a Zotac Zbox Nano AQ02 computer mounted inside its chassis. All sensor inputs as well as control outputs need to be connected to it. Available sensors include stereo cameras in NICO's eyes and dual microphones that can be mounted in its ears. The motors are connected via a serial bus system to a controller board. The motors of one extremity share one bus. The controller board is connected to the computer by USB.

Robotis offers for its Dynamixel servo motors an application programming interface (API) written in C [28]. More high level control is offered by an external framework called "Pypot". It is originally part of the poppy project (see section 2.2) and is written in Python [14]. Simulation of the robot is performed in the software V-rep by Coppelia Robotics. It allows simulating robot dynamics, general mechanics, actuator control and camera sensors. It has built in algorithms for collision detection, iterative inverse kinematics, trajectory generation and motion planning [38]. The model of the NICO robot is imported into V-rep by using a unified robot description format (URDF) file, which is generated from its computer-aided design (CAD) files. The model is simplified to speed up the simulation (see fig. 3.1). Pypot has the option to control both the simulated robot in V-rep and the real robot using the same basic commands. Therefore, Python is chosen as the programming language for the implementation and Pypot as the interface.

As pointed out in section 2.2, the arms of the NICO robot have five dof. Three of the joints are located in the shoulder, one in the elbow and one is located at the wrist (see fig. 3.2). The axes of the first two joints in the shoulder almost intersect and together they can position the end effector in a sphere around the shoulder. The next two joints offer additional control of the position and the orientation.



(a) The NICO robot in V-rep simulation.



(b) The simplified version of the robot model.

Figure 3.1: The NICO robot in the V-rep simulation environment. (a) original (b) simplified

The elbow joint can, unlike a human elbow, be moved in both directions. This can lead to unnatural looking movements. The last joint of the robot arm is located at the robots wrist; it controls the orientation of the gripper. The gripper is closed or opened by an additional joint.

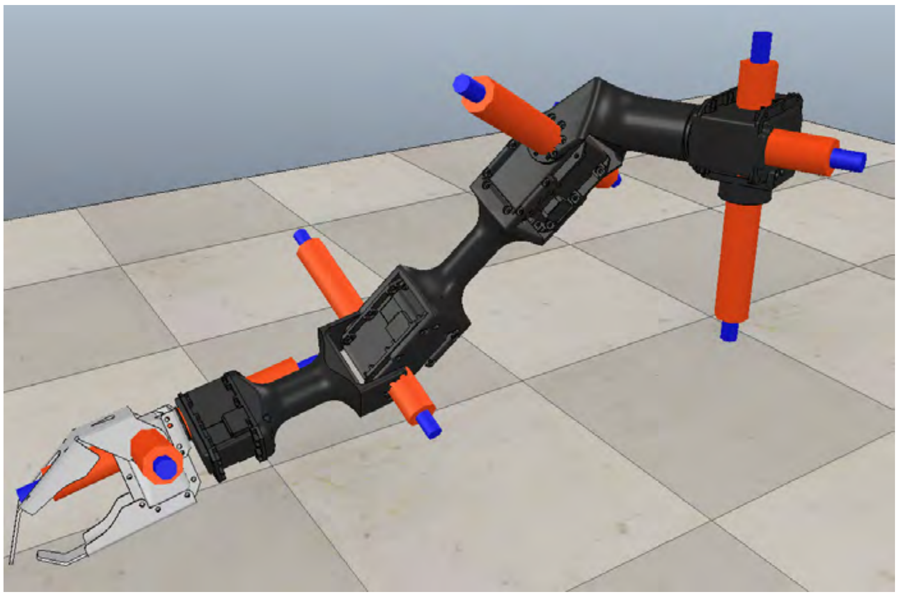


Figure 3.2: The joints of the right arm of NICO as seen in V-rep.

3.2 The Forward Model of the Robot Arm

In order to learn IK data of the forward transformation needs to be generated. As pointed out in 2.5.1, this data can be generated by a mathematical model, a simulation or the real system. Since it is expected that many samples need to be generated, a mathematical model is used.

NICO is a platform which is still under active development, which means that changes to its structure and body are to be expected. This makes it necessary that the model can be adapted without much effort to allow learning on the new model. In development of the NICO robot URDF files are used to define the kinematic structure. As the URDF file already contains most of the necessary information to generate the model, a parser makes it possible to easily update to the current state.

The forward model outputs the position and orientation of the end effector for given joint coordinates. For that it links together the joint transformations. It can output the position, the orientation as a rotation matrix, or Euler angles. Additionally to the information from the URDF file, it is possible to define an additional end effector. This adds an offset to the last link at the current orientation. This is useful when different positions of the robot gripper need to be used or when the robot is holding an item.

The origin of the coordinate system is always defined at the first joint. If the positions need to be used in another coordinate system they need to be transformed accordingly. This is done to match the coordinate system of the simulation.

3.3 Aproaches for Learning Inverse Kinematics

Learning the inverse kinematics is based on the two principal approaches of motor babbling and goal babbling. Implementation of the learning algorithms is done in Python. It needs to access the forward transformation of the robot kinematics to generate the data. Additionally it needs to be able to give IK solutions to other classes. The MLP, which acts as the learning model, is implemented in Pybrain, a NN framework which excels at ease of implementation and flexibility [47].

3.3.1 Learning Implications and Metrics

Depending on the approach, different problems arise during learning, as parameters must be chosen and performance measures considered.

The most important property of the learning approach is its stability. Depending on the parameters of the neural network and on the samples presented to it, it can become unstable and deviate more from the goal with each iteration. For the neural network the most important parameter influencing the stability is the learning rate, a high learning rate can achieve faster results but can cause unstable behavior. For high numbers of neurons, high learning rates can also cause numerical problems. In goal babbling, examples generated depend on the learning so far. That means, that if the neural network has learned from inconsistent examples so far it may not return to the region of consistent examples. The home position movement (see eq. 2.21) is implemented to prevent that.

Convergence additionally requires to the algorithm to get closer to a minimum with each iteration. Close to a minimum the gradient direction is often ill defined causing the network to oscillate around the solution. High learning rates amplify the problem. Because of the complex structure of the problem this oscillating behavior is to be expected, but the magnitude remains a relevant consideration, as it can prevent better approximation of the solution.

The accuracy of the solution is the main performance indicator. A 100% accurate solution would match a closed-form IK solution. This is, with the given method not only not attainable, it can also be argued that such a solution would lack the ability to adjust to other real world influences, such as motor positioning errors or the influence of gravity. It has been previously established (see section 2.4.3) that a MLP cannot learn the position at singularities. Because avoiding singularities can be considered beneficial, the induced positioning error might also be. A good NN solution should therefore be precise in order to be useful, generalizing to be able to include additional factors and should produce smooth trajectories, which might resemble human movements.

Workspace coverage is a major issue of goal babbling approaches. Because the goal directions are defined in task space and only positions with acceptable deviations are being moved to, the algorithm explores the workspace with time. How much of the workspace is actually reached and learned to a decent accuracy therefore depends on the speed of exploration and the number of iterations. Motor babbling on the other hand explores the workspace in joint coordinates. Because a priori knowledge of the joint limits is a reasonable assumption and the manipulator is not massively redundant, meaning full exploration of the joint space is required in either case, full workspace coverage will be reached rapidly.

How quickly the algorithm can come to a solution is very important for its usability. If the learning is to be performed on a real robot, generating millions of data points might be infeasible. Linear methods for example can converge quickly, but their overall performance might not be acceptable. Goal babbling as a method is introduced to allow fast learning even for highly redundant manipulators. Goal babbling is in general quicker the smaller the workspace is that should be discovered. It can also give very good results for the center of the workspace early on, but will only later give acceptable solutions in the periphery. The definition of how fast an approach is depends therefore on its objective: if only a low accuracy is needed, or only a small workspace is used, a solution can be found faster.

3.3.2 Motor Babbling

Motor babbling uses random motor commands to generate the needed data of the task space end effector positions. Since the joints are actuated by servos, the motor commands equal joint positions plus an unknown control error. Because the joints have different limits and actuation ranges, they have to be normalized to be used by the NN.

The example generation can be done with two different methods: either using continuous motions or randomly generating examples in joint space. The latter has the advantage that it can guarantee a more uniform distribution over the joint space, thereby optimizing learning. The other method is to use plausible motions, generated by picking random locations in joint space and connecting them. The generated positions are not uniform anymore; the middle of the joint space will have more associated data. But these movements translate better to both a simulated environment and the real world. The fact that more data is accumulated for the central locations is biologically highly plausible, as humans are more capable of controlling their arms in comfortable locations.

NICO's arm has five dof. The last joint controls the rotation of the gripper. If only the position of the end effector is to be controlled, this last joint can be ignored. This is not true if the robot is holding a tool in its gripper, but this case is not considered here. With three dimensions to be controlled and four dof this is a redundant manipulator. When using motor babbling without any additional methods, some end effector positions will be associated with multiple joint configurations. This will lead to an additional error. The experiments for three-dimensional motor babbling are carried out as random examples.

When additionally to the position the orientation of the end effector is controlled all of the five joints of the manipulator are used. Dual configurations are no longer an issue, because with the additional information the manipulator does not have any redundancy. In fact it is now under actuated, meaning it does not provide movement in all directions. Which these directions are is difficult to identify, as it depends on the current configuration (see section 2.3.1). The orientation can be described by Euler angles: three rotations around the primary axes. Because these are angles they have a cyclic nature. To provide information about that to the network the sine and cosines of each angle are passed to the network. The original angles can then be reconstructed using the arctangent. An interesting observation is that while the new network now has five outputs for the joint positions and nine inputs for the position and orientation, the problem for finding the joint angles associated with a position has not changed much. This is because the additional joint does barely affect the position. The orientation does only provide additional information. While methods are available to carry out this approach with random examples it is done with the more plausible joint movements.

3.3.3 Goal Babbling

Goal babbling relies on plausible, goal directed movements for data generation. It is implemented in both three and six dimensions.

The here implemented approach for three dimensions uses all of the available methods to deal with redundancies (see section 2.5.2). Compared with motor babbling, goal babbling has a lot more parameters. These include the step size for the movements, the random perturbation of the joint angles and the probability of home movements. The workspace of the robot's arm is assumed unknown and very complex, it largely resembles a spherical shell around the origin of the first joint. This means that the straight line movements generated by goal babbling cannot traverse all of it. Instead they need to bounce of the various borders. This is a difficult scenario for goal babbling and can induce additional problems, be it for the exploration of the outer regions or the stability of the algorithm. A parameter is introduced to help detect workspace borders: the joint limits. When a joint limit is reached, this is taken as indicative of a workspace boundary. This breaks the assumption of zero knowledge about the kinematics of the device, but is highly plausible and in line with the motor babbling approach.

Six-dimensional goal babbling is designed analogous with six-dimensional motor babbling. Again, the Euler angles are encoded using sines and cosines and fed to the network for training. The goal direction is this time harder to define, as straight movements do not include rotation. The positional part of the goal direction is generated the same way as for the three-dimensional goal babbling approach. The orientation direction is defined as a random vector of three Euler angles. This makes it easy to integrate the direction into the general framework. In order to still allow for straight movements the definition of the workspace boundary is unchanged. This has the disadvantage that the tried orientation may not match the real orientation. But since the regular movements already introduce many different directions, exploration still occurs. Additionally, movements that mostly affect rotation are a possibility. The method described in eq. 2.20 of detecting a border of the task space fails on such movements, but they will still reach a join limit at some point. Therefore, joint limits are again used as indicators for workspace boundaries.

3.3.4 Relative Inverse Kinematics

As a different approach relative inverse kinematics is introduced. Inspired by iterative IK algorithms this NN based relative approach does not try solve the global IK problem but rather give a joint movement given the current joint configuration and the intended direction. Iterative IK solve this by using approximations for the inverse of the Jacobian matrix. Additionally they rely on the only locally linear relationship between the joint angles and the task position (see section 2.3.1 and eq. 2.6). A NN is capable of providing a nonlinear mapping to approximate those problems. Because of the nature of the inverse of the Jacobian for certain joint configurations movement is not possible in all directions. In some other configurations, there are multiple solutions, which means that different joint movements will result in the same task space position. Both of these cases will result in inconsistent learning examples. But, as long as the proposed movement leads into the right direction, multiple iterations of the algorithm will still lead to a solution.

A major challenge in this approach is the very large input space. For each position in the workspace, each direction must be learned separately. The resulting input space is too big to be learned exhaustively. Using the generalization capabilities of a NN might still lead to a usable solution.

3.3.5 Performance Testing

As pointed out in section 3.3.1 the most important performance indicator is the accuracy of the solution. The obvious method of testing is to create a set of points and then use the average accuracy on those positions. The choice of those tests is important. Ideally they should not only give a simple performance metric but also additional insight into strengths and weaknesses of a solution.

Since the workspace of the manipulator is complex generating a set of points which are actually inside of it is not easy to do in task space coordinates. Instead joint space coordinates can be chosen and the resulting end effector positions are then added to the test set. Selecting the appropriate positions can be done at random. If enough points are created, they will cover the workspace. This solution has two major drawbacks: since the positions were created at random, recreating the results of the experiments is impossible, unless the whole set of points is known as well. This test set also does not provide additional insight into the performance of algorithm.

The solution chosen here is a test set which is spaced out in joint coordinates. The following procedure is used:

- A fraction ρ is chosen.
- Starting from the joint median, each joint is set to that fraction of its limit, its median, or that fraction of its negative limit. This results in three positions for each joint.
- All possible combinations of all joints are then used as test positions. For four joints this results in $3^4 = 81$ positions.

The chosen fractions are $\rho \in \{0.1, 0.5, 0.75, 1\}$. This test set can be easily recreated using only the information about the joint limits. It also contains a lot of additional information, especially for goal babbling: if the error close to the joint medians is very small, but further outside very large, then exploration only took place in the center. The error values at the joint limits represent an extreme case: as it is a discontinuous position the MLP cannot learn its exact value. As such it should be close to the maximum error. Because the four values of the error contain very distinct information all of them are used to judge a result. It is important to remember that the joint position is the output of the NN. The test inputs are therefore the task positions generated for these joint configurations.

The error of a position is calculated as the Euclidean distance between the intended and the actual position. The result has again the unit meter. The error of the orientation is not as easily defined. The Euclidean norm of the Euler angles is only useful for angles $< 90°$, because otherwise different representations can give different results, negating the definition of a norm [20]. Here the Euclidean norm on the six variables representing the position, the sine and cosine values of the Euler angles, is used. While this metric does not directly represent any physical unit it does give insight into the accuracy of the NN.

The relative NN cannot directly calculate global positions. Therefore the test set must be comprised of relative movements as well. The origins of these movements are spread throughout the task space by the same method as before. The goal positions of the movement are spread around it in task space, using a similar method. The distance can be adjusted. For each origin this results in $3^3 = 27$ goal positions. Because the absolute error will be small, it is divided by the length of the intended movement, resulting in a relative measure.

3.4 Trajectory Generation for the Robot Arm

To make the robot arm actually move, both in simulation and reality, the usage of some form of trajectory generation is needed. The more advanced schemes, detailed in section 2.3.3, can deliver smooth transitions and behavior. But, for their ideal implementation, they also need information about the robot kinetics, specifically maximum joint speeds and accelerations. This information is not available and implementing the algorithm based on this information breaks the assumption of a learning robot controller. Because the robot uses servo controllers, these strategies are not necessary.

3.4.1 Joint Space Move Commands

The simplest kinds of movements are defined in joint space. Two positions are specified in joint coordinates, the movement then connects those two points. While it is the simplest kind of movement it is still widely used, because the start and end positions can be taught by manually positioning them. For this movement the individual joint positions are split up in portions representing the sampling time of the robot controller. The Dynamixel motors get the goal position at each sampling interval. If instead the motors would just directly get the final position as input, their PID controller would create a movement at maximum speed which will likely have overshoot.

3.4.2 Task Space Move Commands

Task space move commands require the use of inverse kinematics. The two main categories are point to point movements, where only the start and end position is important, and movements with a defined end effector path. The most important of the latter are linear movements.

For a point to point movement the needed joint coordinates of the start and end positions are derived using IK, the path in between is carried out in the same way as a joint space movement.

Linear movements require IK solutions for all points along the path. They can be generated by splitting up the linear path in segments according to the sampling time of the controller and then looking up IK solutions for each position along the way. Because these do not always exist, a path based movement can get stuck in joint limits. For the three dimensional task space, both a method for linear movements is implemented and a convenience function for relative movements, which adds the input vector on top of the current position.

In the six-dimensional task space the definition of a linear movement is difficult. It is implemented as a linear path in three-dimensional space and a linear interpolation of the orientation. Because the NN requires all inputs to be defined a method that feeds the current orientation back as input is additionally implemented.

The coordinate system used for the orientation is the world frame and not a frame attached to the end effector. This makes defining global movements more consistent. The disadvantage is that local rotations are not as intuitive. However, the only local rotation which is always available is achieved using only the wrist joint, which does not need inverse kinematics. Generally the two representations can be transformed into each other using the forward kinematics.

Relative inverse kinematics require different functions for their movement. A simple linear movement is achieved by splitting up the goal direction into small steps and using these small movements and each current joint configuration as input to the NN. The disadvantage here is that errors from each individual movement add up. Another method is to use a constant move speed and update the goal direction for after each movement according to the current position. This currently makes use of the forward kinematics to assess the end effector position, but external referencing with a camera is also possible. Additionally to both these options each movement can be optimized internally with the following steps:

- A joint update is calculated by the NN.
- The resulting end effector position is calculated using forward kinematics.
- The error between the intended direction and the real direction is calculated.
- A fraction of that error is used as input to the NN.
- Repeat until the error is smaller than some margin.

With this process it could be possible to achieve a very high accuracy. The disadvantages include the following:

- This process requires internal knowledge of the forward kinematics, thereby breaking the assumption of zero a priori knowledge.
- It only works if the update moves into the right direction, which might not be the case for all configurations and positions.

- It uses more than one iteration and, depending on the implementation, an unknown compute time. This is a problem as the next update needs to be known at sample time.

All of the methods for moving mentioned above are implemented. Since they are implemented in Python and use the Pypot framework they can be used to control both the simulation in V-rep and the real robot.

3.5 Structure and Training of the Neural Network

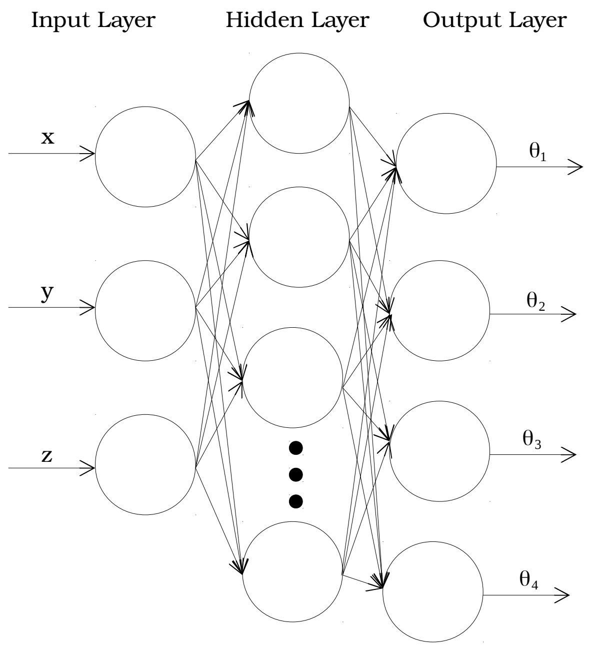


Figure 3.3: The structure of the neural network for learning in a three-dimensional task space.

In order to learn the IK some kind of regression method needs to be used. Because of its universal approximation capabilities, a MLP can be used. Since universal approximation can be achieved by using one hidden layer, this configuration is chosen. As stated in section 2.4.3, the fact that IK is generally non continuous means it cannot be precisely approximated at all positions. This can be helpful to avoid singularities, but a problem close to the workspace boundaries. The hyperbolic tangent is chosen as the activation function in the hidden layer. The activation function in the output layer is, unless otherwise stated, a linear function. The number of neurons in the hidden layer is one of the major optimization parameters: a high number of neurons allows the network to achieve more

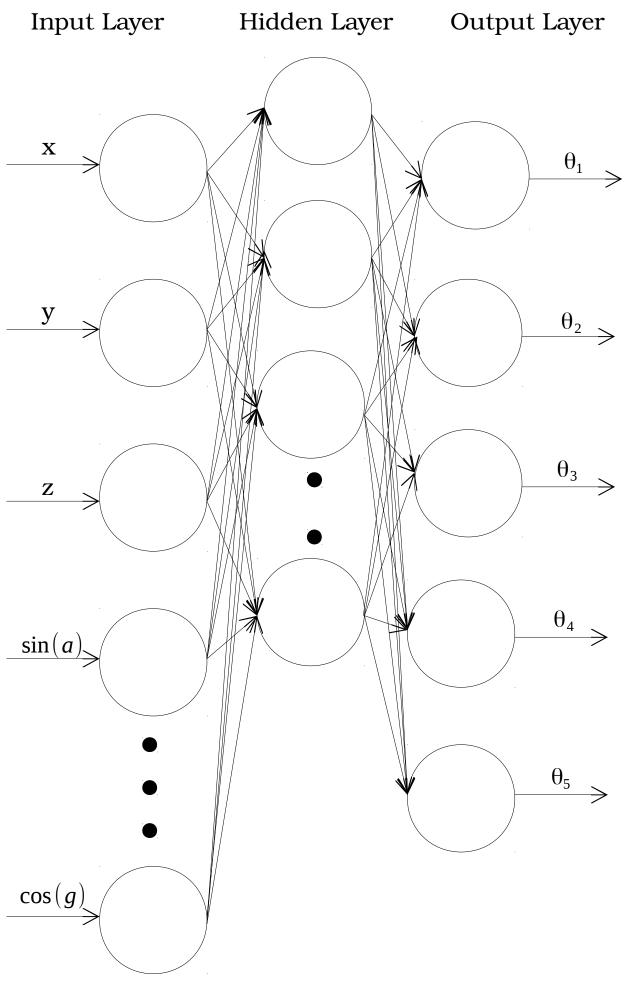


Figure 3.4: The structure of the neural network for learning in a six-dimensional task space.

complex behavior, but might learn slower and generalize worse. The number of input and output neurons is chosen according to the structure of the problem. For the IK solution in a three-dimensional task space, the inputs are the Cartesian coordinates and the outputs are the joint angles (see fig. 3.3). The joint angles are normalized according to their median position and their ranges. When the orientation is included in the six-dimensional case, the inputs also include the sine and cosine values of the current orientation in Euler angles (see section 3.3.2 and fig. 3.4). The relative IK approach has the current joint positions as input: it needs to know the current configuration in order to update it. The other inputs are a relative movement. The output is not the new joint position, but rather the update to the current joint configuration (see section 3.3.4 and fig. 3.5).

The NN is trained using backpropagation. Goal babbling is an online learning scheme and therefore only such training methods can be used. It works by training

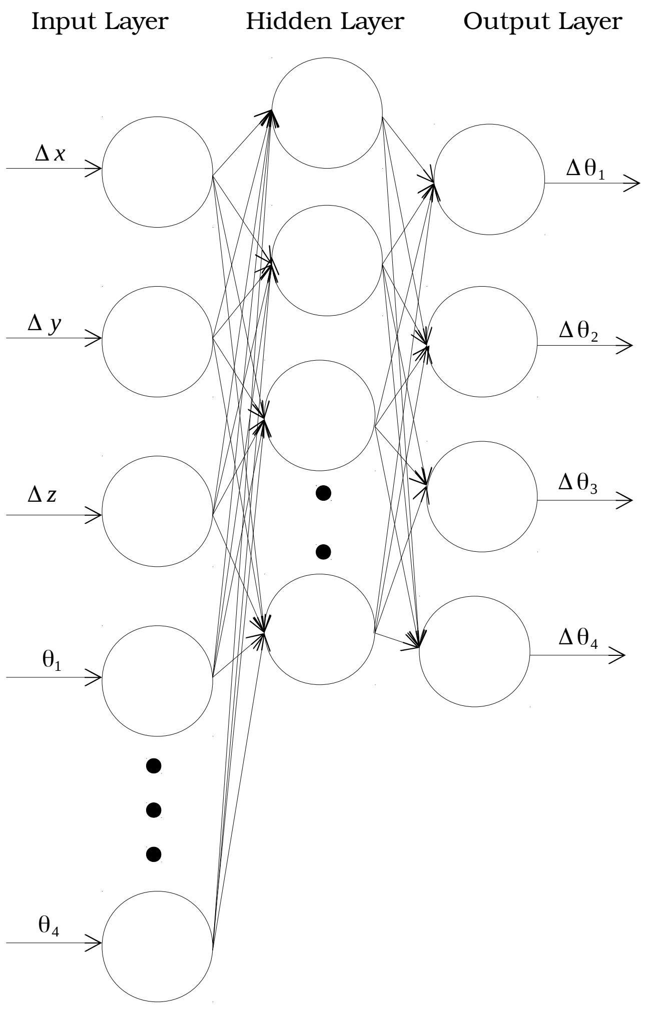


Figure 3.5: The structure of the neural network for learning the relative inverse kinematics in a three-dimensional task space.

on batches of data, because otherwise learning from singular positions in the outer regions of the workspace can destabilize the solution. The size of those batches is therefore chosen such that they cover a significant amount of the already explored workspace. The batches can then be trained as whole or using stochastic gradient descent. Following previous solutions (see section 2.5.2), stochastic gradient descent is chosen. While it would be possible to train on each data set for more epochs, it is not done. This is because goal babbling can already use the information gained from a single epoch to generate new examples and explore the workspace further. Fast exploration can thereby be more efficient without. In a scenario where data points are harder to generate, for example when training is done on a real robot, it might still be better to keep all data and use it to further optimize the solution. To keep motor babbling comparable, the same training method is used. In general motor babbling makes it possible to train the network independent of the data generation. This opens up the possibilities towards offline learning methods. When generating data on a real robot, this might enable training a sufficient model from less data points.

Chapter 4

Experimental Results

In the following section the results of the different approaches for learning IK are laid out. First the learning behavior of the NN in different configurations is analyzed. The practicality of using them to control the robot arm is tested. This is first done in simulation and afterwards verified on the real robot.

4.1 Learning Inverse Kinematics

4.1.1 Meta-Parameter Optimization

All of the approaches presented in section 3.3 have numerous parameters that influence learning behavior, not all of which can be explored exhaustively. Since all of the approaches use a MLP as the model, its parameters are present for all of them. The number of input and output neurons is specified by the application. The number of hidden neurons h is one of the parameters which is optimized. A high number of hidden neurons allows to learn more complex behavior but might learn slower and generalize worse. The other meta-parameter which is always present is the learning rate α. A high learning rate can speed the process up, but can also make it become unstable or oscillating. The other parameters depend on the chosen approach and are discussed when applicable.

Because the methods use different movements to create their data, the following does not count the number of movements, but rather the number of data points created. In the following they are called iterations, since each of these points is used to train the neural network once. Overfitting to the acquired dataset is therefore impossible. Unless otherwise stated, the number of iterations is fixed at 10^7, in order to keep the results comparable. This can favor some methods over others, but since for training on the real robot the amount of data that can be generated is limited, it is a useful comparison.

Table 4.1: The results of the experiments for three-dimensional motor babbling. Mean value and 95% confidence interval, five samples.

(a) Error at positions close to joint medians

$\rho = 0.1$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.0536 ± 0.013	0.0560 ± 0.017	0.0520 ± 0.026
$\alpha = 0.001$	0.0584 ± 0.027	**0.0368** ± 0.016	0.0483 ± 0.028

(b) Error at positions at $\rho = 0.5$

$\rho = 0.5$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.0641 ± 0.0044	0.0696 ± 0.018	0.0611 ± 0.0056
$\alpha = 0.001$	0.0595 ± 0.0061	**0.0496** ± 0.0016	0.0629 ± 0.026

(c) Error at positions at $\rho = 0.75$

$\rho = 0.75$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.0792 ± 0.0008	0.0729 ± 0.0094	0.0685 ± 0.0044
$\alpha = 0.001$	0.0755 ± 0.0030	**0.0628** ± 0.0024	0.0717 ± 0.026

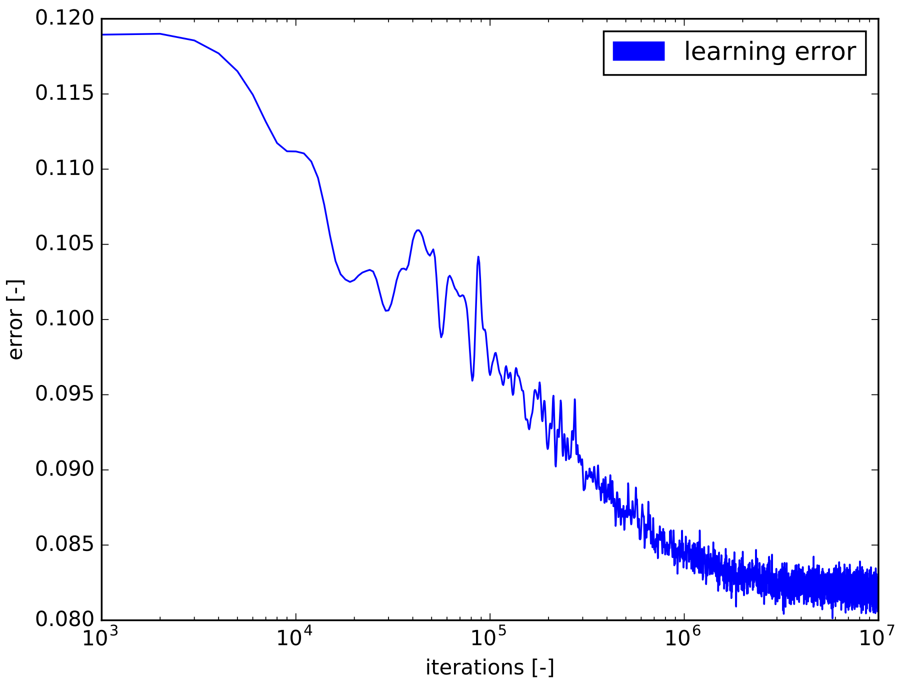
(d) Error at positions at $\rho = 1.0$, at the joint limits

$\rho = 1$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.150 ± 0.0036	**0.126** ± 0.0066	**0.126** ± 0.0052
$\alpha = 0.001$	0.145 ± 0.0064	**0.126** ± 0.0042	0.130 ± 0.015

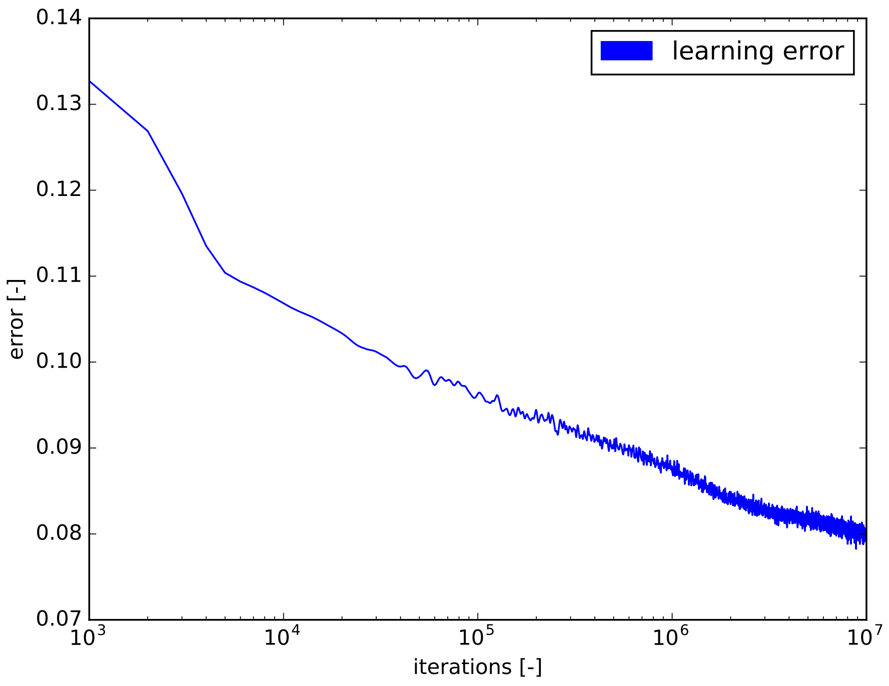
4.1.2 Performance Using 3D Motor Babbling

The motor babbling method used (see section 3.3.2) for the three-dimensional task space has no other meta-parameters than those of the neural network. The results for those are presented in table 4.1. As pointed out in section 3.3.5, the error is given separately for different regions in the joint space. Parameter ρ gives the relative distance of the position from the joint medians to the joint limits. At $\rho = 1$ the joints are at their limit.

The first interesting fact is that the values for each distance are close together, independent of the number of hidden neurons and learning rate. One hypothesis is that the errors are due to limitations of the model: it cannot represent the underlying process more precisely. In the joint centers this can be due to redundant joint positions. If more than one joint configuration results in the same task space position, averaging between them will cause errors. In the six-dimensional case this is not possible; section 4.1.4 will show the influence of those configurations. The joint limits represent discontinuities: they cannot be modeled accurately by a MLP. The errors go up considerably the closer they are approached.



(a) Using a learn rate of $\alpha = 0.01$



(b) Using a learn rate of $\alpha = 0.001$

Figure 4.1: The training errors for $h = 40$: (a) $\alpha = 0.01$, (b) $\alpha = 0.001$. Averaged other five samples, smoothed over 10 data points.

A closer look at the learning errors over the iterations in fig. 4.1 show that while a lower learning rate affects the learning speed only slightly, it continues to learn even after 10^6 iterations. This holds true even for the simplest tested configuration with only 10 hidden neurons. That the learning stops earlier for the higher learn rate can suggest oscillations around the optimal solution. The fact that the learning speed is only slightly reduced by the lower learning rate also suggests a high influence of random noise.

The test errors during learning (see fig 4.2) show a similar picture to the learning errors. For the higher learning rate, learning again comes to a halt at 10^6 iterations. With the lower learning rate it can be seen that the errors close to the joint limits are still being reduced while the errors in the center remain. This is consistent with the results of the other tested configurations. The problem of dual configurations can limit the precision in the center, so additional examples do not lead to any improvement. The errors at the joint limits can be reduced, since a better approximation is needed.

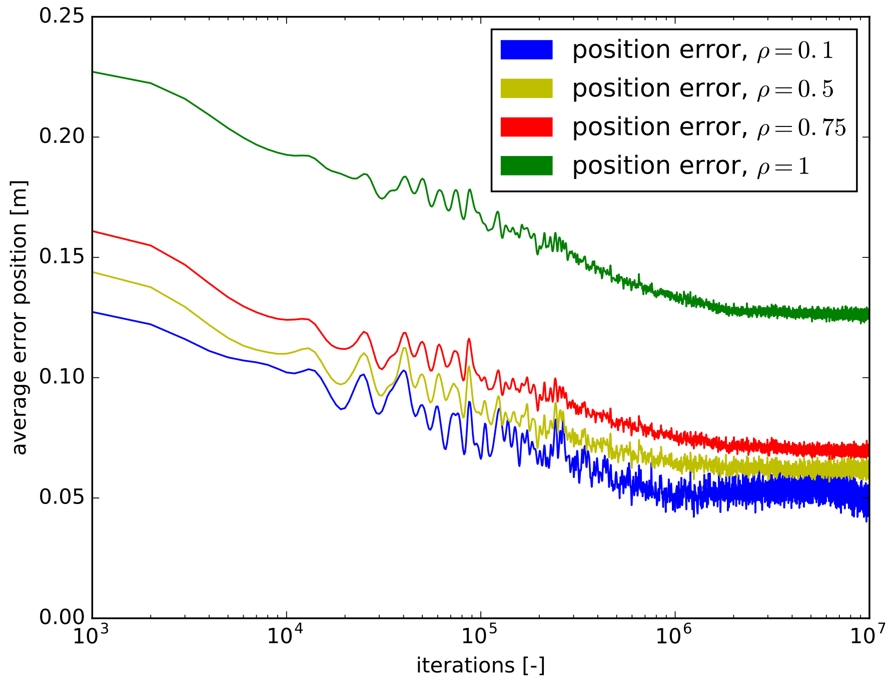
The errors for the test positions (see fig. 4.3a) are showing a systematic offset. The actual direction of this is different for each trained neural network. The directions of the errors further outside in the workspace seem more random. Some of the positions at the joint limits seem to be erratic, while others are close to the real value. Because most of these positions are at the border of the workspace, this shows that goal positions at the edge or outside of the workspace can lead to irregular movement. The fact that the error in the center of the workspace has the form of a constant offset has the advantage that it does affect relative movements less, meaning that the position can be easily adjusted.

4.1.3 Performance Using 3D Goal Babbling

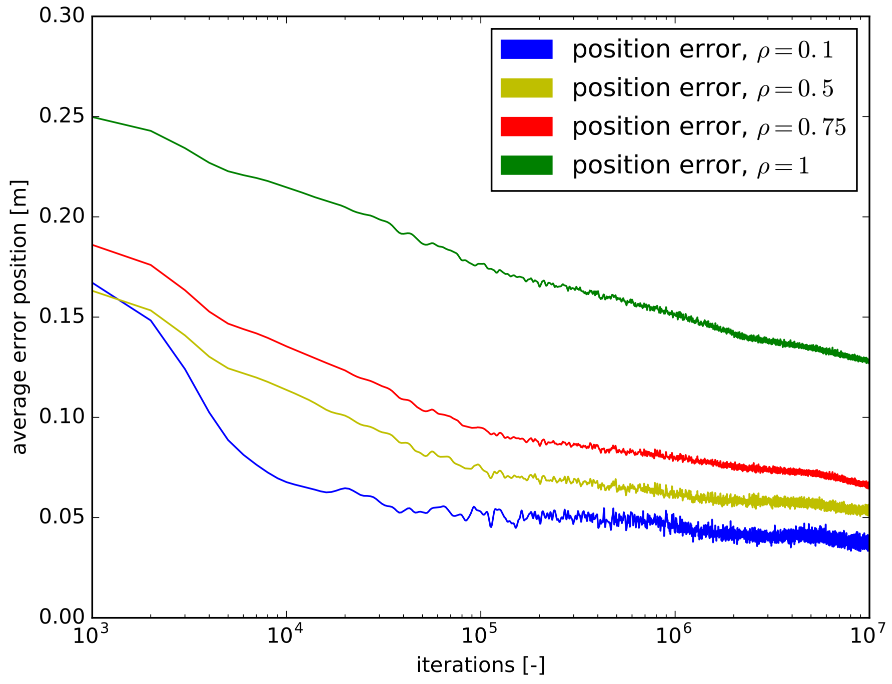
Unlike motor babbling, goal babbling has an excess of parameters that can be adjusted. These are:

- ϵ_X The intended step size
- e_θ the random noise of the joint positions
- p_{home} probability of a home movement, after a boundary is hit

Additionally to these parameters, the algorithm can be altered in many ways: The weight adjustments, which promote efficient movements (see eq. 2.17) or movements which move into the right direction(see eq. 2.18), can be included or ignored. The NN can be trained directly after each new positions, or after a certain number of them.

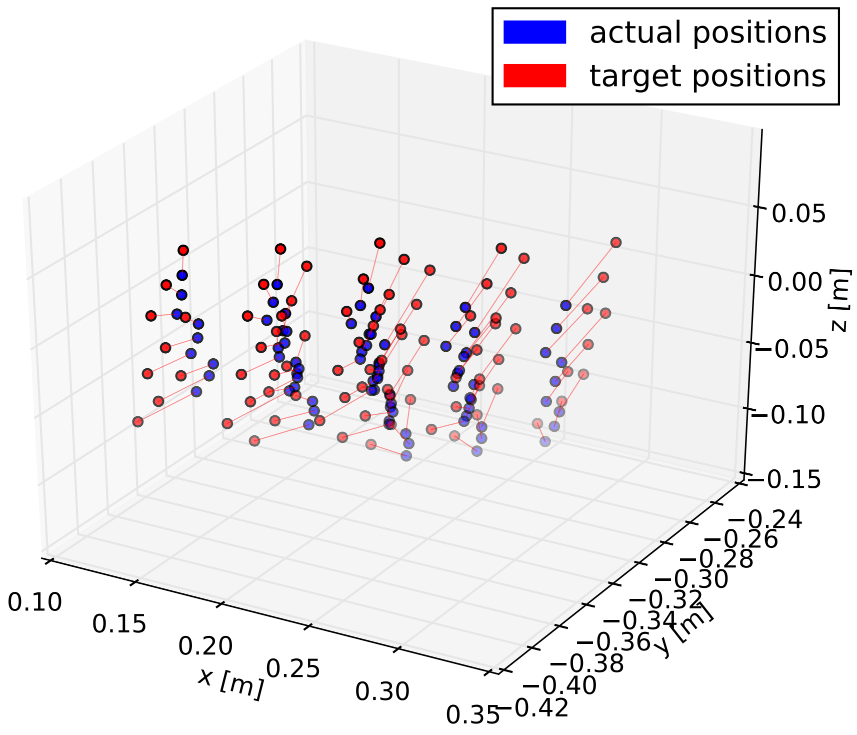


(a) Using a learn rate of $\alpha = 0.01$

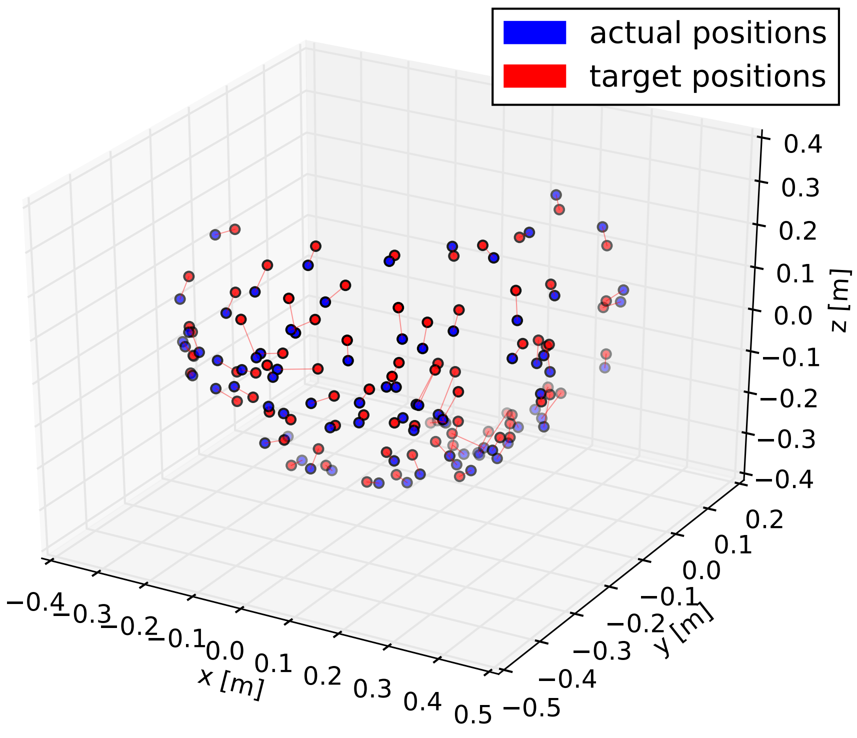


(b) Using a learn rate of $\alpha = 0.001$

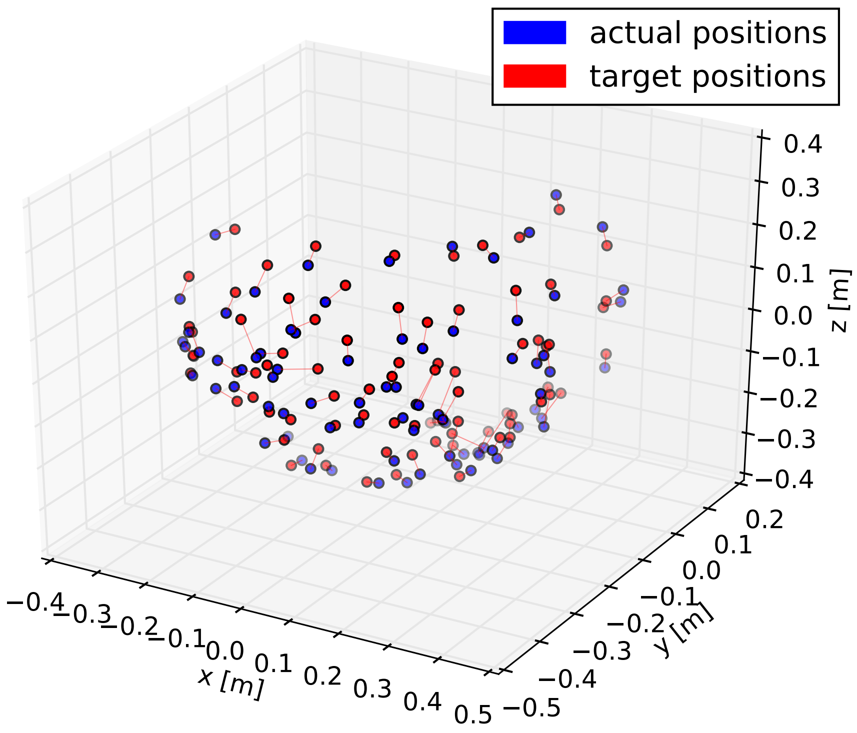
Figure 4.2: The test errors for $h = 40$: (a) $\alpha = 0.01$, (b) $\alpha = 0.001$. Averaged other five samples, smoothed over 10 data points.



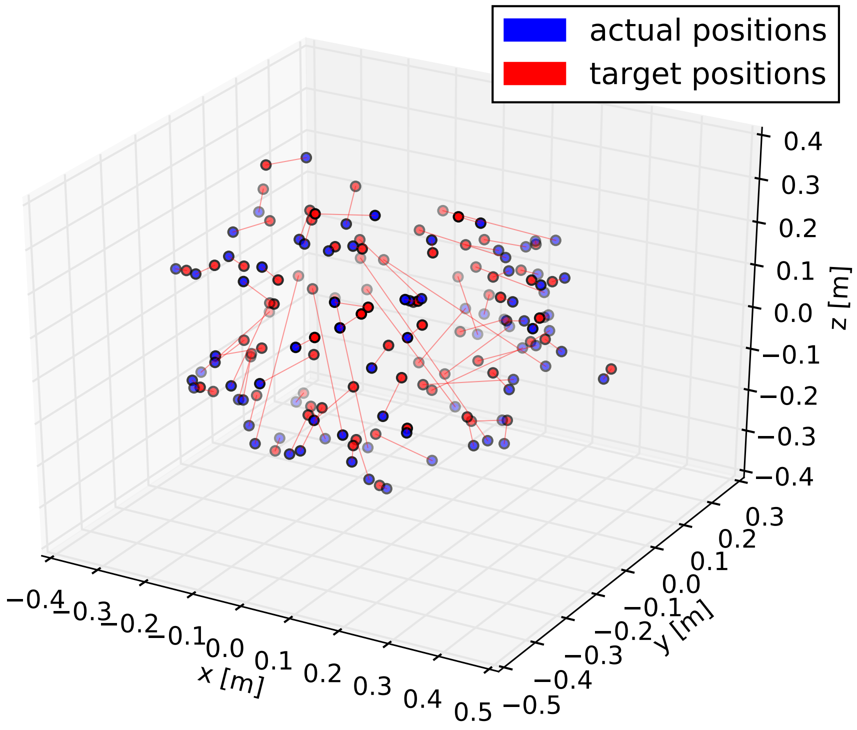
(a) The target and actual positions, $\rho = 0.1$



(b) The target and actual positions, $\rho = 0.5$



(c) The target and actual positions, $\rho = 0.75$



(d) The target and actual positions, $\rho = 1$

Figure 4.3: The test positions of a specific NN for the different values of ρ. The network parameters are: $h = 40, \alpha = 0.001$

Since not all of these can be explored fully, this section explains the influence each of them has. The parameters of the NN have again only a limited influence on the result. Table 4.2 shows the results for the different configurations. It is instantly apparent, that while the error at the center is low, in the more distant positions of the work space it becomes larger very quickly. The errors in the center are significantly lower than those achieved with motor babbling, which suggests that the weight adjustment mechanisms are successful in reducing the influence of redundant manipulator configurations.

Table 4.2: The results of the experiments for three-dimensional goal babbling. Mean value and 95% confidence interval of five samples. Activated weighting functions. $\epsilon_X = 0.02$, $e_\theta = 0.02$, $p_{home} = 0.1$

(a) Mean error at positions close to joint medians

$\rho = 0.1$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.0222 ± 0.011	**0.0147** ± 0.008	0.0226 ± 0.022
$\alpha = 0.001$	0.0536 ± 0.021	0.0202 ± 0.0175	0.0297 ± 0.035

(b) Mean error at positions at $\rho = 0.5$

$\rho = 0.5$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.184 ± 0.067	**0.137** ± 0.037	0.153 ± 0.084
$\alpha = 0.001$	0.237 ± 0.075	0.153 ± 0.058	0.220 ± 0.13

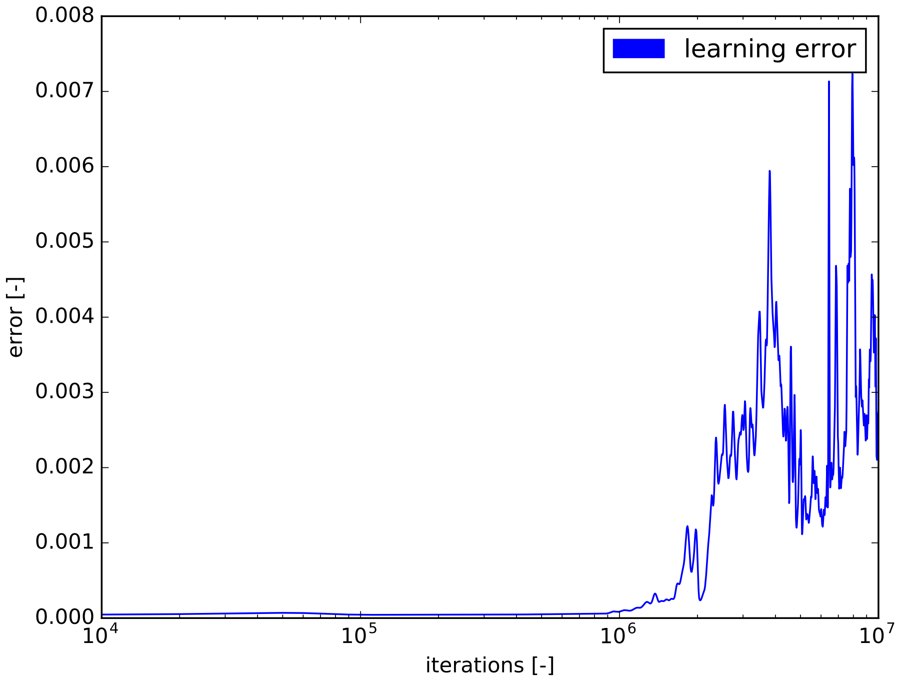
(c) Mean error at positions at $\rho = 0.75$

$\rho = 0.75$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.262 ± 0.056	0.221 ± 0.048	**0.220** ± 0.081
$\alpha = 0.001$	0.325 ± 0.069	0.230 ± 0.024	0.303 ± 0.11

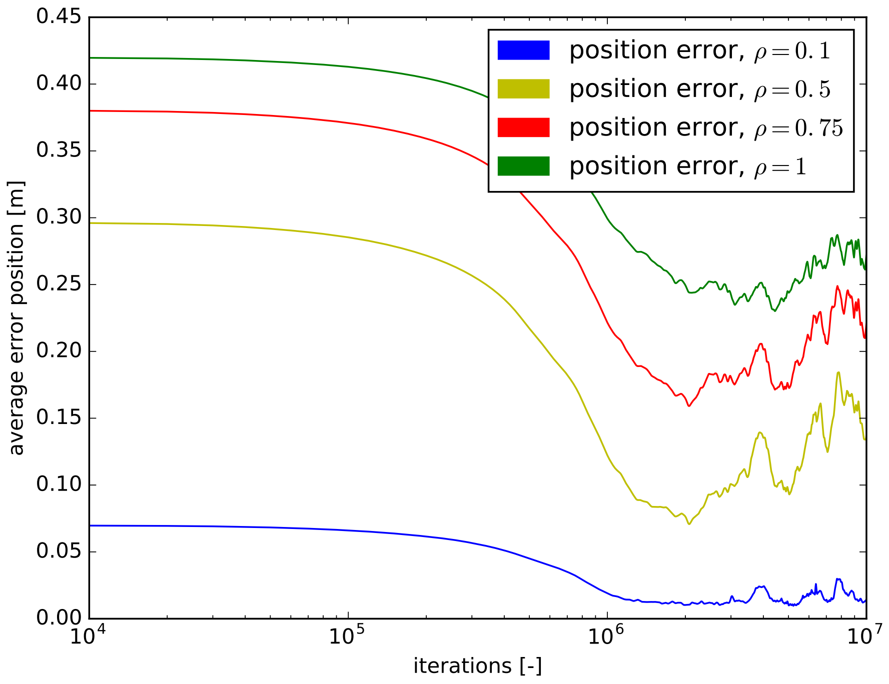
(d) Mean error at positions at $\rho = 1.0$, at the joint limits

$\rho = 1$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.304 ± 0.058	0.273 ± 0.058	**0.262** ± 0.073
$\alpha = 0.001$	0.379 ± 0.048	0.283 ± 0.016	0.343 ± 0.074

A closer look at the test and learning errors of one configuration in fig. 4.4 gives additional insight. As can be seen the learning is in the beginning very slow. The network is initialized such that it always outputs joint positions very close to the home position, independent of the goal position (see also fig. 4.5). The only new examples are therefore introduced by the random perturbation of the joint angles, defined by e_θ. The weights which are chosen to judge the effectiveness of such movements are also very low, since they will only randomly move into the correct direction. Choosing a high perturbation of the joint angles leads to another problem: The more random the movement is, the closer this solution is to just motor babbling, thereby defeating the purpose.



(a) Learning errors



(b) Testing errors

Figure 4.4: The test and learning errors for $h = 40$, $\alpha = 0.01$: (a) learning errors (b) test errors. Averaged other two samples, smoothed over 10 data points. $\epsilon_X = 0.02$, $e_\theta = 0.02$, $p_{home} = 0.1$

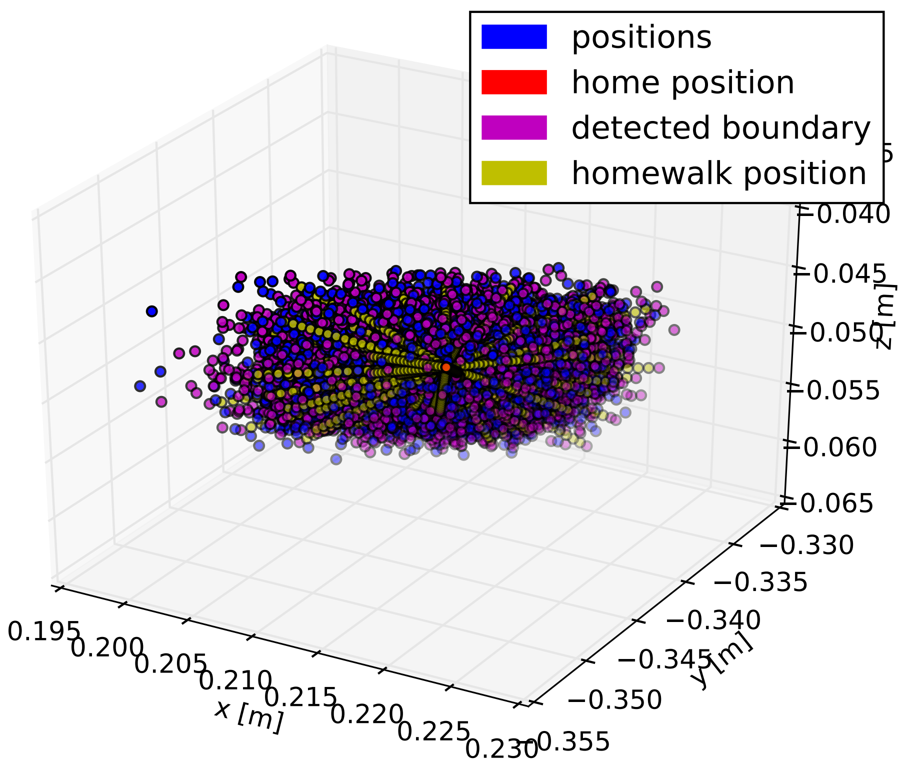
The second important observation is that when the error terms first go down the learning errors go up. This is contrary to not only the way it is in motor babbling, but also the intuition. The reason for this behavior is that the learning error only exists once new examples are shown to the network. While motor babbling starts learning instantly and with a high error, goal babbling explores the workspace while keeping the error constant. The error is kept in check by the mechanism that keeps the goal babbling inside its known regime. Because the speed of learning is dependent on the gradient, motor babbling can initially learn much faster. Since a high value for e_θ emulates motor babbling, this can explain why it was found advantageous for an implementation with few dof [3].

After the error terms first decline, they rise again and oscillate. While often times this would be a sign of overfitting, in this case it is due to another phenomenon: After the network has acquired some knowledge about how to move, it starts exploring the local surroundings. If the workspace is complex, as it is in this case, the algorithm can get lost in its more distant parts and forget about the space closer to the home position. Home movements are implemented to counteract this behavior. But, if the probability of home movements p_{home} is chosen too high, it hinders the algorithms potential to explore the outer regions. But if it is chosen too low it can become unstable, resulting in uncontrolled drift in the outer regions. This is also the primary reason why the network should only be updated after it has collected data from a number of positions: updating it instantly on each sample will introduce drift in the outer regions, since the home positions are not included in the update. The dilemma of exploration is caused primarily by the structure of the workspace. Unlike previous applications the workspace of the robot is very complex, resembling roughly a spherical shell. This means that single straight movements cannot cross the workspace. The outer regions are thereby very difficult to reach for the algorithm. A solution to this might be to use spherical coordinates instead of Cartesian. This would allow movements which are linear in the configuration space to traverse the workspace [41].

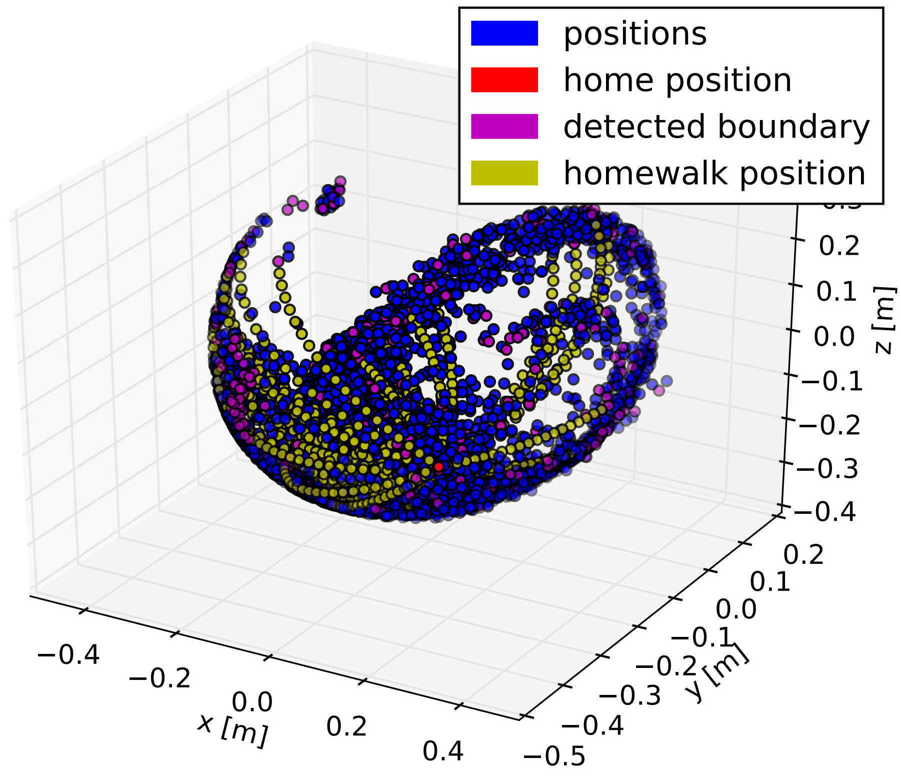
The way goal babbling explores the workspace is shown in fig. 4.5. As pointed out before, the exploration only starts after a significant amount of random movements. Once the networked learned how to move locally the exploration accelerates.

One network was trained for 10^8 examples. While the good result of an error of $\{0.0085, 0.0482, 0.114, 0.184\}$ meters suggests that the network continued to learn after 10^7 examples, its learning graph makes clear that these results were achieved multiple times during learning.

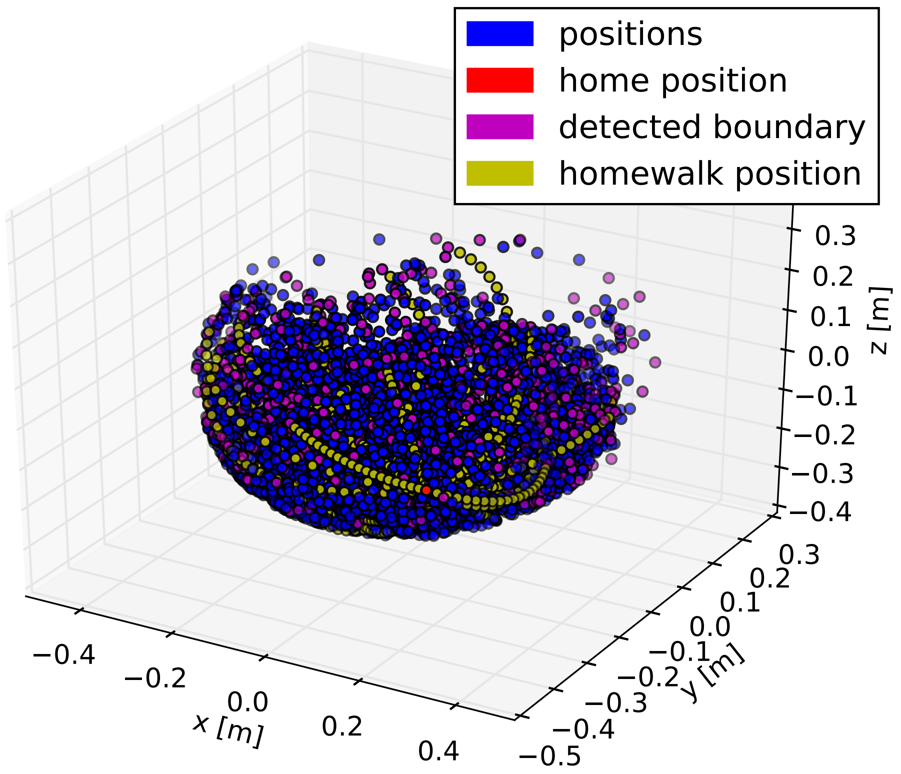
In conclusion it can be said that goal babbling does work even for this complex workspace, but its accuracy, while very good in the center, reduces quickly for positions outside of it. Especially problematic can be, that during learning of the periphery of the workspace, the overall accuracy of the other positions suffers. A possible solution might be to convert the workspace into spherical coordinates.



(a) The discovered workspace after 10^5 iterations.



(b) The discovered workspace after 10^6 iterations.



(c) The discovered workspace after 10^8 iterations.

Figure 4.5: Workspace discovery with goal babbling. While for 10^5 (a) discovery has not yet set in, at 10^6 (b) iterations a big portion of the workspace is covered. After 10^8 (c) iterations the workspace is mostly covered, though positions in the periphery are sparse. The amount of shown positions are limited to 10^5 each. Detected boundaries, the home position and home movements are illustrated. $\epsilon_X = 0.02$, $e_\theta = 0.02$, $p_{home} = 0.1$

4.1.4 Performance Using 6D Motor Babbling

Including the orientation in the results of motor babbling can achieve a higher accuracy, since redundant configurations do not exist anymore. While the input space has more dimensions, the size of the actual workspace is not increased. This means that each data point just contains more information to generate the IK solution. The downside of this is discussed in section 4.2. The results in table 4.3 reflect the higher positional accuracy in the center. More specifically, the positions close to the joint medians ($\rho = 0.1$ and $\rho = 0.5$) have a lower positioning error. The positions close to the joint limits show an even higher positioning error than those of the three-dimensional case. Redundant configurations are not likely to exist close to the joint limits, so there is no advantage to be gained by including the orientation. That the error is even higher at the joint limits can be the result of two factors:

- The NN needs to learn the orientation as well, which influences learning behavior.
- The way of acquiring data has changed: This implementation uses realistic joint movements. They are less likely to come close to the limits.

Unlike the three-dimensional case, this version shows a significantly changed error for some configurations. Especially the version with $h = 40$ falls off behind the configurations with more neurons in the hidden layer. This leads to the assumption that this more complex scenario can make proper use of the additional capabilities of such a network.

Table 4.3: The results of the experiments for six-dimensional motor babbling. Mean value and 95% confidence interval of ten samples. Linear output layer.

(a) Error at positions close to joint medians

$\rho = 0.1$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.0288 ± 0.011	0.0201 ± 0.0097	**0.0154** ± 0.0042
$\alpha = 0.001$	0.0223 ± 0.0047	0.0270 ± 0.0092	-

(b) Error at positions at $\rho = 0.5$

$\rho = 0.5$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.0395 ± 0.0063	0.0324 ± 0.0040	**0.0322** ± 0.0061
$\alpha = 0.001$	0.0370 ± 0.0027	0.0337 ± 0.0042	-

(c) Error at positions at $\rho = 0.75$

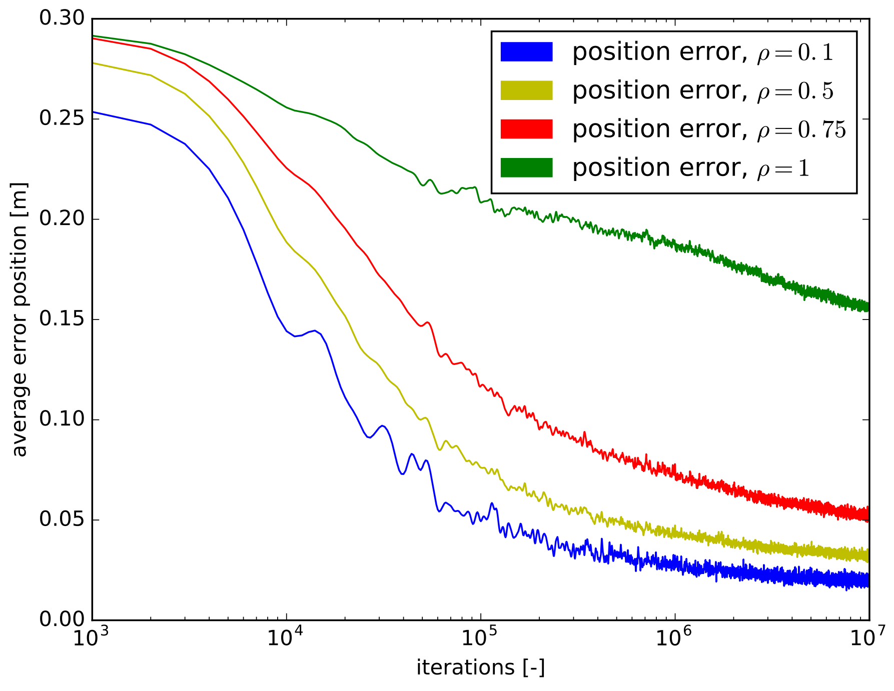
$\rho = 0.75$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.0669 ± 0.0043	**0.0503** ± 0.0047	0.0536 ± 0.0072
$\alpha = 0.001$	0.0663 ± 0.0023	0.0513 ± 0.0034	-

(d) Error at positions at $\rho = 1.0$, at the joint limits

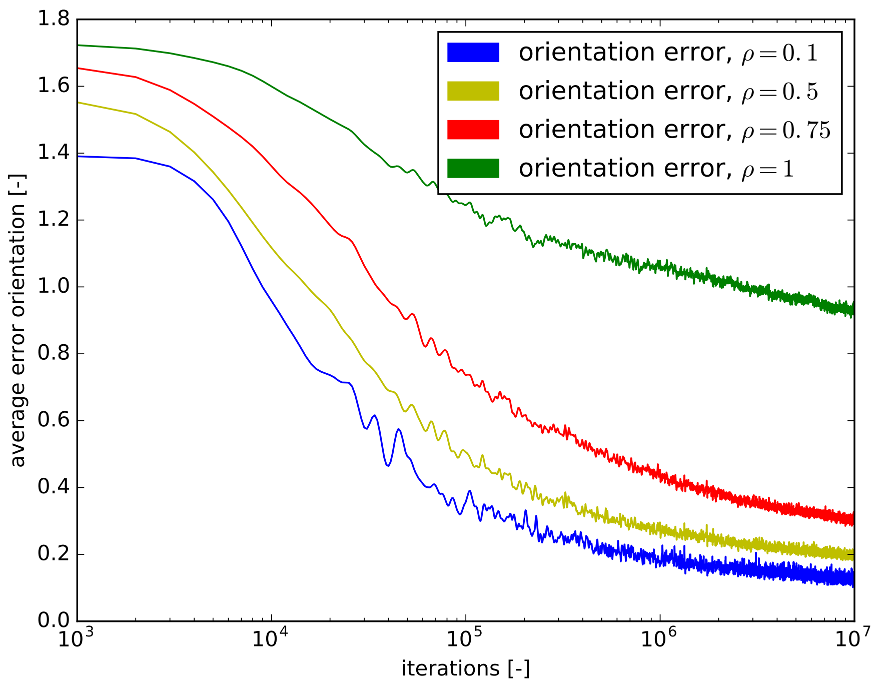
$\rho = 1$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.1739 ± 0.0067	0.1547 ± 0.0047	**0.1533** ± 0.0072
$\alpha = 0.001$	0.1835 ± 0.0058	0.1722 ± 0.0052	-

Additionally to the here shown results with a linear output layer, the tests were done with a *tanh* activation function in that layer. The results show no significant differences in performance.

The graphs of the test errors (see fig. 4.6) show that even higher learn rates still leave learning potential after 10^6 iterations. This is also a sign of the added complexity of including the orientations. The learning behavior of the orientations themselves is very similar to that of the positions. The errors close to the joint limits are higher than the errors in the center.



(a) Test error position



(b) Test error orientation

Figure 4.6: The test and learning errors for $h = 120$, $\alpha = 0.01$: (a) position errors (b) orientation errors. Averaged other ten samples, smoothed over 10 data points.

Table 4.4: Test error after 10^8 examples.

$h = 120$	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.75$	$\rho = 1$
error position [m]	0.0154	0.0255	0.0386	0.147
error orientation [-]	0.284	0.162	0.236	0.904

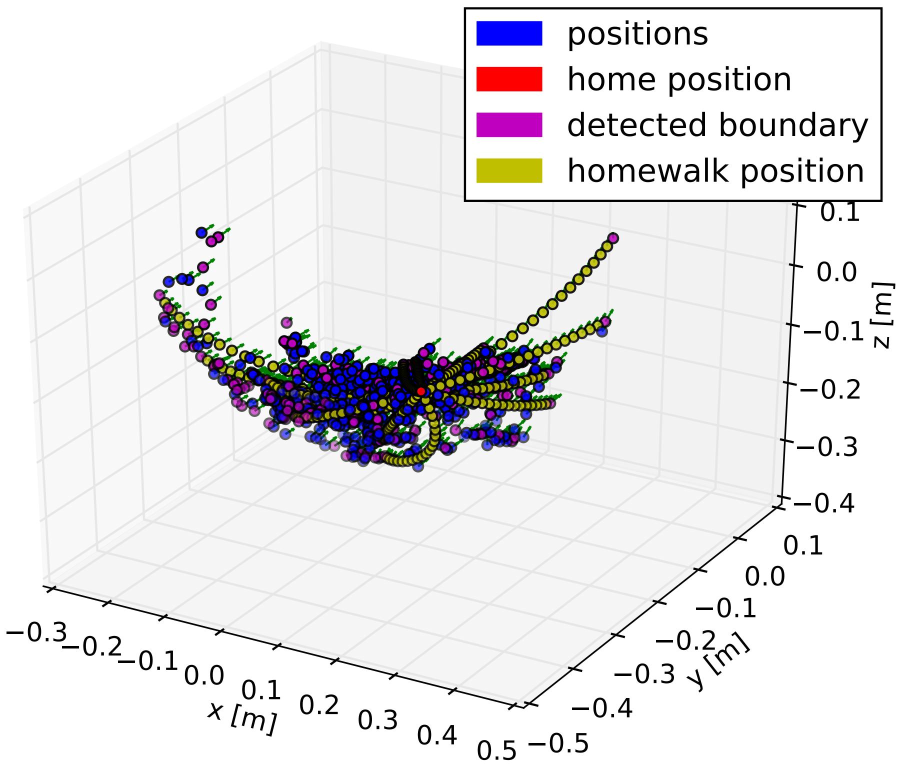
One configuration was trained for 10^8 samples and achieved a performance that matched the other versions at positions close to the joint medians, but outperformed them at the positions close to the joint limits (see table 4.4). While collecting that many data points is not be option when working with a real robot, repeated training on the same dataset could have the same effect.

4.1.5 Performance Using 6D Goal Babbling

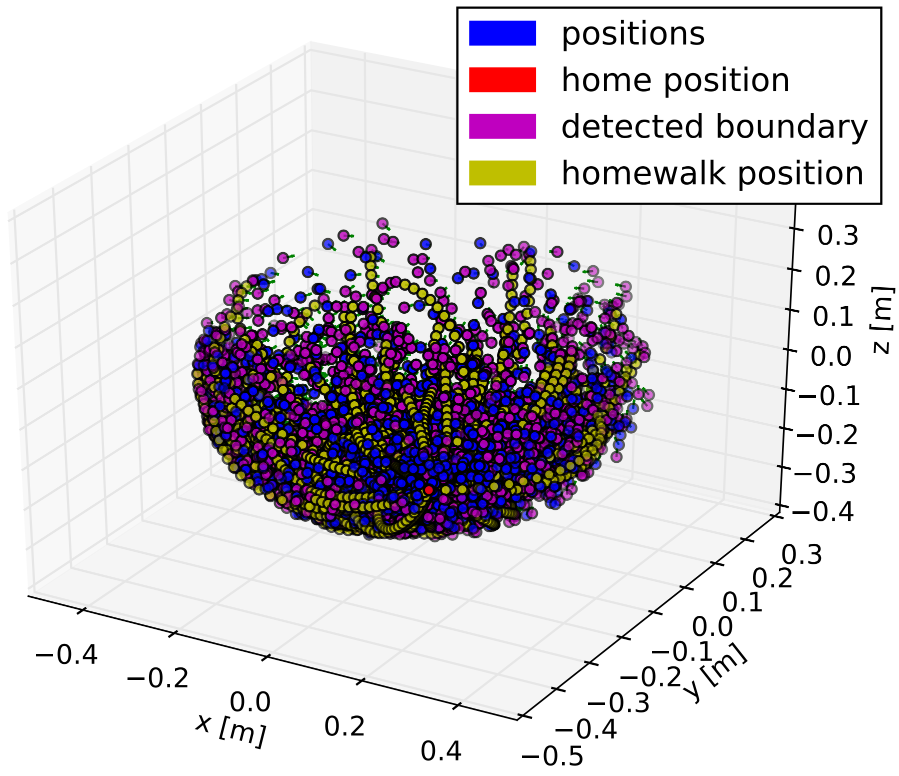
The position errors of six-dimensional goal babbling can be seen in table 4.5. The error close to the joint medians is very low for all configurations. This is due to the frequently visited home position. The errors at $\rho = 0.5$ are significantly lower for configurations with a higher amount of neurons in the hidden layer. This behavior is similar to six-dimensional motor babbling, only that it is even more pronounced here. While for motor babbling the better approximation capabilities of more hidden neurons results in just a lower error, the effect is more extreme for goal babbling: the algorithm can only move towards positions it knows, if a position is unknown it will detect a boundary and continue in a different direction. While this behavior keeps the majority of the solutions with few neurons in the hidden layer from exploring further outwards (see fig. 4.7), the large confidence interval suggests what can be verified when looking into individual results: some of the NNs did successfully learn these positions and had a significantly lower error. Further outside in the joint space the errors are going up for all configurations. This is the result of the same problem as in the three-dimensional case: those positions are visited less frequently since they cannot be reached by a single movement.

The errors in the orientation follow the pattern of the position errors: they are low in the center and grow considerably further outside. This is most pronounced when the network fails to explore the considered region.

The learning curves of the well-performing configuration (see fig. 4.8) show that the error decreases continuously for both the orientation and the position. This is in hard contrast to three-dimensional goal babbling, in which the error often increases over time. It can therefore be assumed that in case of complex workspaces the additional information given by the orientation helps to reduce the amount of inconsistent examples and thereby stabilizes learning. This solution is still significantly outperformed by six-dimensional motor babbling at all positions except the very center of the workspace.

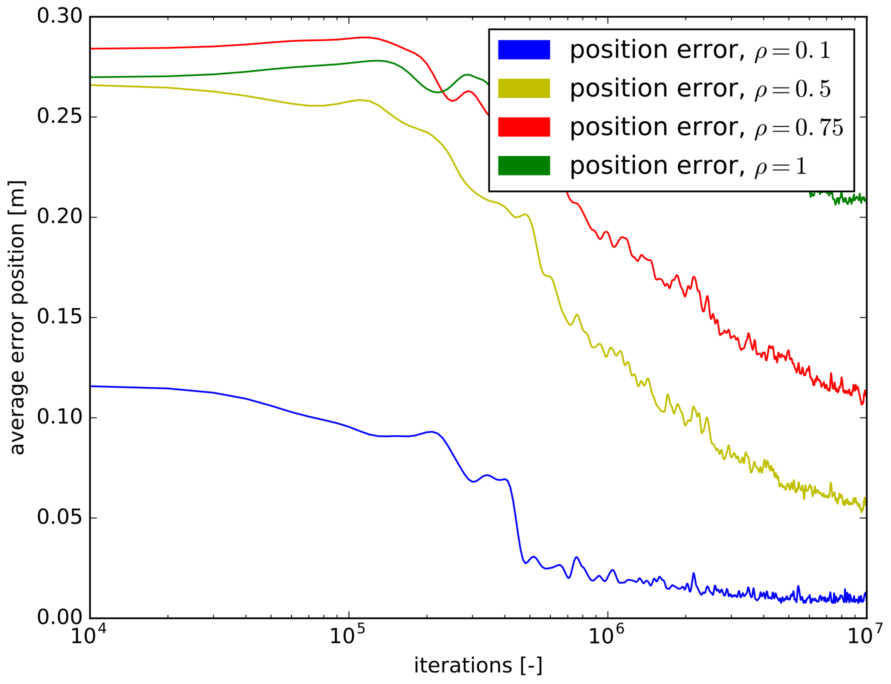


(a) Exploration sample, $h = 40$, $\alpha = 0.01$

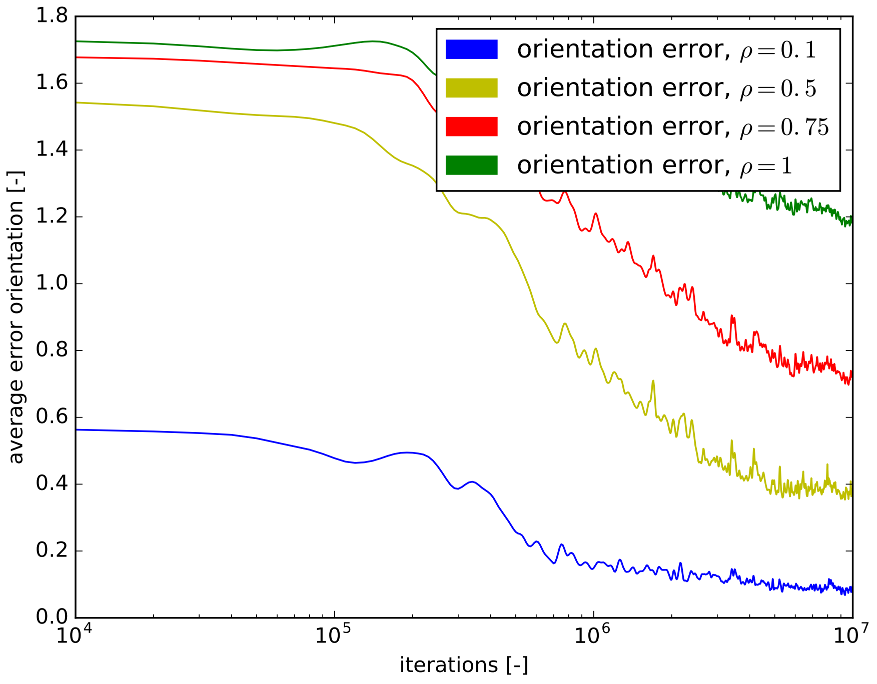


(b) Exploration sample, $h = 160$, $\alpha = 0.01$

Figure 4.7: Two examples of the exploration after 10^7 iterations: (a) $h = 40$, $\alpha = 0.01$ (b) $h = 160$, $\alpha = 0.01$.



(a) Test error position



(b) Test error orientation

Figure 4.8: The test errors for $h = 160$, $\alpha = 0.001$ (a) position (b) orientation. Average over 3 samples, smoothed over 10 data points.

Table 4.5: The results of the experiments for six-dimensional goal babbling. Mean value and 95% confidence interval, three samples. $\epsilon_X = 0.02$, $e_\theta = 0.02$, $p_{home} = 0.1$

(a) Error at positions close to joint medians

$\rho = 0.1$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.039 ± 0.0077	0.034 ± 0.062	0.016 ± 0.002
$\alpha = 0.001$	0.089 ± 0.043	0.0069 ± 0.0019	$\mathbf{0.007} \pm 0.001$

(b) Error at positions at $\rho = 0.5$

$\rho = 0.5$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.183 ± 0.076	0.123 ± 0.247	0.082 ± 0.026
$\alpha = 0.001$	0.296 ± 0.132	$\mathbf{0.047} \pm 0.002$	0.059 ± 0.020

(c) Error at positions at $\rho = 0.75$

$\rho = 0.75$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.221 ± 0.054	0.180 ± 0.202	0.155 ± 0.034
$\alpha = 0.001$	0.308 ± 0.103	0.125 ± 0.0238	$\mathbf{0.112} \pm 0.048$

(d) Error at positions at $\rho = 1.0$, at the joint limits

$\rho = 1$	$h = 40$	$h = 120$	$h = 160$
$\alpha = 0.01$	0.258 ± 0.065	0.253 ± 0.134	0.243 ± 0.042
$\alpha = 0.001$	0.297 ± 0.064	0.218 ± 0.037	$\mathbf{0.208} \pm 0.025$

In order to judge the effect of further learning, one configuration ($h = 120$, $\alpha = 0.01$) is trained for 10^8 iterations. The results of $\{0.0075, 0.0582, 0.137, 0.223\}$ meters for the corresponding values of ρ are better than the average of those with the same configuration but are not better than the overall best. The learning graph suggests that the results did not improve considerably after 10^7 iterations.

4.1.6 Performance of the Relative IK Solution

Table 4.6 shows the results for the training of a network for relative IK. The results are very different from all the previous since they represent relative errors: the network is given an intended movement of $0.01m$ in each direction and the result is divided by the length of the intended movement (see section 3.3.5) For the configurations at the joint limits this does mean that some target positions are unreachable since they are outside of the workspace. Therefore the results for $\rho = 1$ are omitted.

The relative errors for all positions are rather high. This can point towards a number of different problems. Even though the movements are relative, dual positions are still an issue. It can be possible to achieve the same move direction with

Table 4.6: The results of the experiments for three-dimensional motor babbling training a relative IK solution. $0.01m$ intended step size, relative error. Mean value and 95% confidence interval, three samples.

(a) Error at positions close to joint medians

$\rho = 0.1$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.775 ± 0.059	0.747 ± 0.036	0.729 ± 0.029
$\alpha = 0.001$	0.761 ± 0.025	0.713 ± 0.030	**0.695** ± 0.057

(b) Error at positions at $\rho = 0.5$

$\rho = 0.5$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	0.912 ± 0.042	0.777 ± 0.032	**0.720** ± 0.013
$\alpha = 0.001$	0.899 ± 0.021	0.780 ± 0.014	0.723 ± 0.034

(c) Error at positions at $\rho = 0.75$

$\rho = 0.75$	$h = 10$	$h = 40$	$h = 120$
$\alpha = 0.01$	1.05 ± 0.007	0.863 ± 0.024	**0.783** ± 0.027
$\alpha = 0.001$	1.05 ± 0.019	0.882 ± 0.028	0.796 ± 0.038

different joint movements. Since the relative movements are small the influence of this issue is smaller in comparison to the global IK solutions (see section 2.3.1). The other explanation lies in the big input space. Not only must each position be learned, but also each direction for each position. The result of this can be a very slow learning process. As for the different configurations: This approach does benefit from the added capabilities of more neurons in the hidden layer. This is especially true for positions closer to the joint limits.

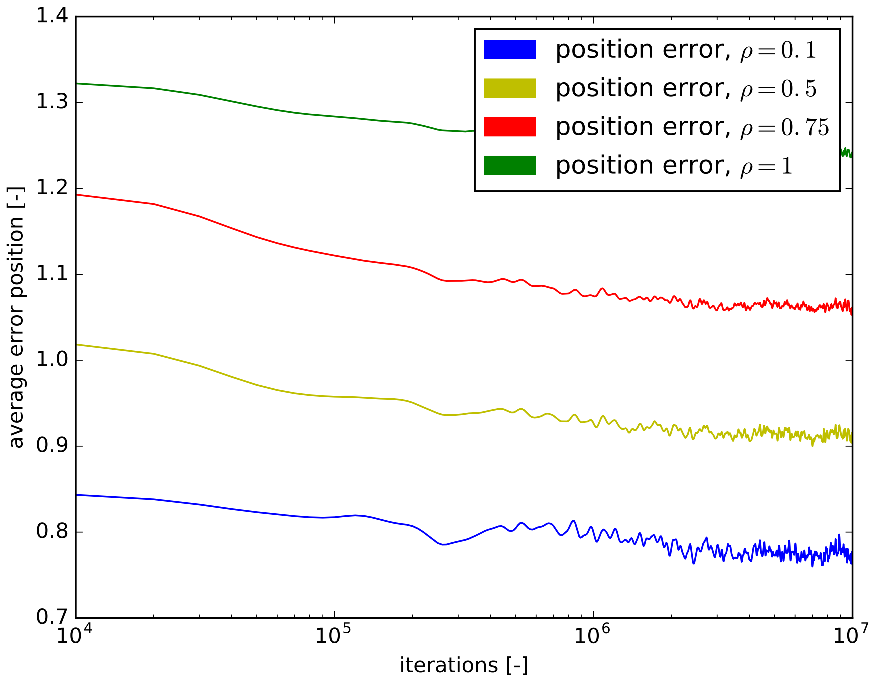
Fig. 4.9 shows the test errors over the iterations for the simplest NN configuration with ten hidden neurons and the most complex with $h = 120$. The simpler network learns initially faster, but is then overtaken by the more complex network. Learning is generally very slow for both, especially in the center configuration.

Because of the relative nature of the approach these results have only limited meaning towards their practical usability. This is analyzed in section 4.2.

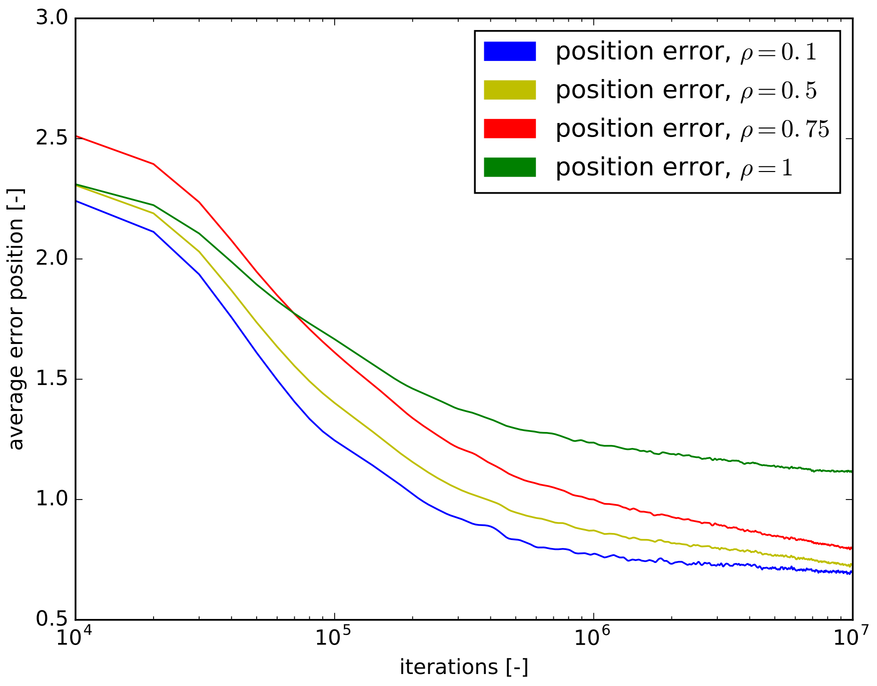
4.1.7 Performance in a Limited Workspace

Limiting the workspace by reducing the joint limits has numerous advantages:

- It can exclude positions with self-collisions
- It could improve the accuracy by reducing the search space



(a) Relative test error, $h = 10$, $\alpha = 0.01$



(b) Relative test error, $h = 120$, $\alpha = 0.001$

Figure 4.9: The relative errors for(a) $h = 10$, $\alpha = 0.01$ (b) $h = 120$, $\alpha = 0.001$. Average over 3 samples, smoothed over 10 data points.

- It linearizes the problem to some extent since the influence of each joint is reduced
- The workspace is less complex and can be traversed easier

The disadvantage is that it arbitrarily reduces the capabilities of the manipulator. Since self-collisions are depending on the current configuration of the robot, for example sitting or standing, avoiding them by reducing the joint limits leads to an overly conservative solution. It can be argued that self-collisions are best dealt with during motion planning, because in this step collisions with external objects also need to be considered and avoided. This is independent whether the motion planning is done by an operator or autonomous.

Due to time constraints, the following experiments are done only once and should be considered as mere indications and not as exact results. The reduced joint limits are chosen such that they resemble previous work [3]. Since the test set is different, and the experiments are only done once, the numbers can still not be directly compared. Additionally, here the number of data points is limited and not the number of movements, resulting in different sizes of acquired datasets.

Previous work suggests using a simple NN with $h = 10$. For goal babbling a very high random perturbation term combined with a high learning rate is recommended. Here a learning rate of $\alpha = 0.1$ and a random perturbation of $e_\theta = 0.1$ are used, both are smaller than the ones recommended. With these values, an error of $\{0.0204, 0.0190, 0.0338, 0.0613\}$ meters can be achieved after 10^6 iterations. This error term is considerably lower than all of those achieved for the full workspace. When using a lower random perturbation of $e_\theta = 0.02$ instead, the error becomes $\{0.0026, 0.0150, 0.0402, 0.0749\}$ meters. Here the error in the center of the workspace is considerably reduced, but the error outside of them is higher. Using pure motor babbling on with the same NN parameters leads to a result of $\{0.0124, 0.0155, 0.0215, 0.0388\}$ meters. When comparing this to the goal babbling solution with $e_\theta = 0.1$ it supports the assumption that it emulates motor babbling. Unfortunately, this also makes it more prone to the problem of redundant configurations, as is noticeable as the higher error in the joint center. The performance achieved with motor babbling does seem to be comparable with the one achieved in the previous work.

While these numbers are only indication, they do show that a smaller task space significantly reduces the positioning errors for both motor babbling and goal babbling. Even though it obviously limits the movements of the manipulator, this can be an option if a high accuracy is needed and offsets are not acceptable.

4.2 Evaluation of the IK Solutions in Simulation

With Pypot it is possible to use the same code to control the real robot and a simulated version in V-rep. V-rep includes four different physics engines, which can simulate dynamics, collisions and the joint servo motors. Accurate simulation requires precise modeling of the system. The CAD model of the NICO robot is an accurate kinematic description but does not possess all dynamical properties of the real robot. Additionally, the PID controllers of the simulated servo motors behaved inconsistent depending on the type of physics engine used. As a result, the simulation is used to judge how the different solutions can be used practically and not to get insight into the accuracy of the solution. In order to gain a certain amount of comparability between the approaches, one grasping movement was tested with all applicable solutions (see fig. 4.10). The main focus is still on general remarks towards the usability. It also has to be mentioned that the forward model used to train the NNs does seem to have some rotational offset which is currently unaccounted for. The reason could lie in a rotation of the basis frame of the forward model compared to the world coordinate system of V-rep. The influence of this issue in practical use is limited.

The three-dimensional motor babbling solution enables the robot to approach different positions throughout the task space. While the lower accuracy towards extreme positions is noticeable, most of them include self-collisions and are as such useless. The accuracy problems in the usable workspace are less of an issue, as long as relative movements are concerned. Locally the errors generated by the NN have the character of an offset, which has limited effect on the movements. Absolute movements on the other hand are more affected. When positions are closer to the corpus of NICO self-collisions are an issue and should be checked for in simulation. Rotation of the end effector can be easily implemented as movements of the last joint. The movements achieved are not as straight as those known from industrial robots. This does give them a more human look. Some of the arm configurations which the solution chooses are not possible for humans or are ineffective for us. The robot often makes use of its elbow joint, which bends in both sides, unlike the human elbow. Also the robot uses the shoulder joints to generate motions pointing towards and from his body, while humans would generate those generally more with the elbow. This behavior does reduce the human appeal of the robots movements. The standard grasping movement could be performed without issues.

The goal babbling solution does not behave much different from the motor babbling one. It is better suited for absolute movements near the joint centers, since there the accuracy is higher. The bigger errors in other positions can cause erratic movements. For most applications it seems that the motor babbling solution is the better approach since it is more consistent. The standard grasping movement could still be performed.

Using the six-dimensional solutions proves to be very difficult. Since it requires the orientation of the end effector in world coordinates, those need to be specified. Knowing them for a position is difficult, as is deducing whether or not a position is reachable. When rotating the end effector at a certain position, the robot tries to make use of all joints, and not just uses the wrist. A possible workaround when using the six-dimensional solution is to use the three-dimensional in order to reach for a position and use the six-dimensional for the rotation. The problem with that lies in the offset of each solution, switching the NNs will move the end effector. Additionally, the only rotation which is always available is easier and better achieved by simply rotating the wrist joint. In order to follow the standard grasping motion, another method to achieve three-dimensional movements is used: the current orientation is used as the intended orientation. The result of this is not the intended one, the errors add up quickly and the movement becomes unstable. As such, using the six-dimensional solutions does not provide benefits over the three-dimensional approaches.

The solution using relative motions has, as pointed out in section 2.6, a high error in relation to the intended movements. This does become easily obvious when using it for the standard grasping movement. Even small relative movements show a very significant deviation from the target position. This can be remedied when the target direction is updated each move step with the current direction to the target (see section 3.4.2). This does however lead to curved movements, as it first moves in the wrong direction and then slowly adapts it. Using the relative IK solution iteratively does not work as expected. The generated joint positions are seldom better than the original ones. Instead it often becomes unstable, and the error term grows. Another issue of this relative solution is that close to joint limits it can get stuck. This happens when the NN only outputs directions which lead further along the already limited joints. In its current form, this relative IK solution lacks the practicality of the three-dimensional approaches using absolute coordinates. The ability to update the position still might make it useful when coupled with external sensors such as the cameras in NICO's head.

All of the found solutions have specific shortcomings. Because of its general practicality and consistent learning behavior, the three-dimensional goal babbling solution is the best approach for most situations. When a higher accuracy in the center is needed, goal babbling can be the better choice. Alternatively the solution can be trained on a limited portion of the workspace which improves the accuracy greatly.

4.3 Evaluation Using the Real Robot

The main focus of this evaluation is to see how the planned movements behave on the real robot in comparison to the simulation. This is no longer dependent on the different approaches to generate the inverse kinematics. Therefore the following is tested using one network trained with motor babbling.

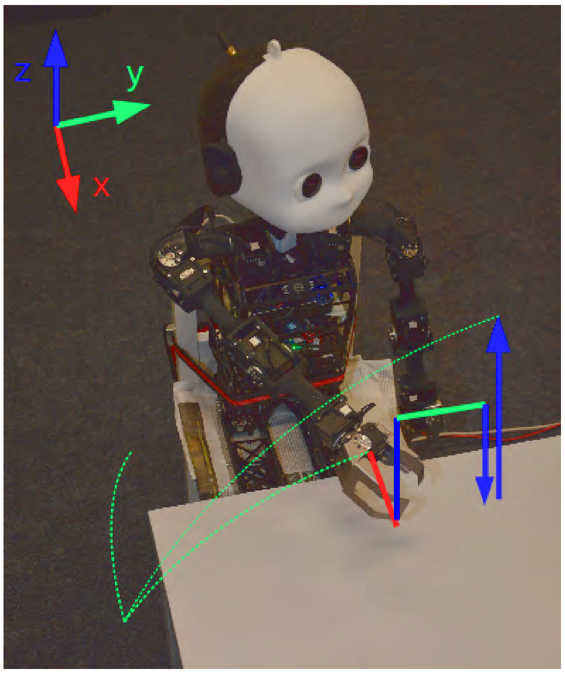


Figure 4.10: The tested grasping movement. Dashed lines indicate point to point movements, solid lines linear movements.

One obstacle in making the movements analogous to the simulation is lies in the definition of the movement direction and joint limits. For Pypot, those are defined in one file for both the simulation and the real robot. V-rep uses the URDF file to define those parameters. Therefore, the definitions in both of these files need to be identical and match the physical robot. The accuracy of the robot is not thoroughly tested here. In general such tests require sophisticated measuring tools and techniques, which lie outside of the scope of this work.

The operated movements are seemingly identical to those planned in simulation. This opens up the possibility to do task planning using the simulation. This can be especially useful to avoid collisions while using the real robot. The movements themselves are still sometimes jerky during linear elements. This seems to be the result of stick slip effects in the motors and can likely be fixed using different parameters for the servo motors. Using the implemented controller NICO is capable of performing grasping tasks (see fig. 4.11).

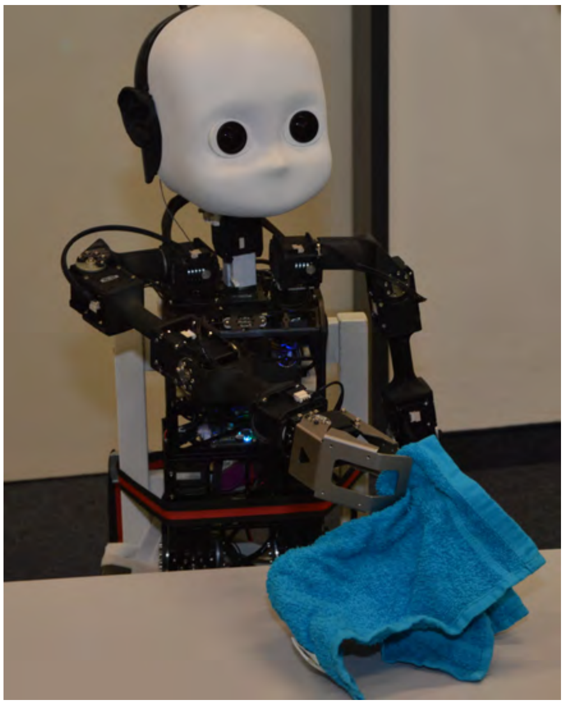


Figure 4.11: NICO grasping a towel.

Chapter 5

Discussion

Summery

The task is to design a neural network based controller for a humanoid robot arm which enables it to perform grasping tasks. The arm of the Neural-inspired companion (NICO) robot is a five degree of freedom (dof) manipulator equipped with a gripper as its end effector. In order to enable effective movements, a solution to the inverse kinematics (IK) problem needs to be found. Because closed-form solutions to this problem are not always available and cannot adapt to changes in the manipulators structure, a learning, neural network based approached is chosen. Current research suggests two principal biologically inspired methods of learning: goal babbling and motor babbling. Motor babbling relies on random motor commands in order to generate the examples from which a model is trained. In goal babbling, reaching movements are performed in task space. The model is trained from the so far performed movements, which in turn enable it to improve it reaching motions. As such, goal babbling is inherently an online learning method.

Since the amount of needed data can be high, the training is done on a, automatically from the model generated, mathematical forward kinematics solution. The wrist joint the robot has little effect of the position of the end effector, but it is very useful for controlling its orientation. As such, it needs to be incorporated into a grasping solution. One challenge faced when learning IK is that of redundant configurations achieving the same end effector positions. While learning, this leads to inconsistent examples. Incorporating the orientation into the learning approach can remedy this. It can also make proper use of the wrist joint. As using this six-dimensional approach has proven difficult in practice, an additional relative IK method is implemented. All mentioned methods are implemented, optimized in their parameters and tested for their accuracy. To use them to control the robot, both in simulation and reality, methods using the Pypot framework are implemented. These allow for linear and point to point movements in the task space.

The practicality of the different IK solutions is tested in simulation. It is then shown that these translate well into usage on the real robot.

Specifically, the following steps were undertaken:

- Automatic computation of the forward kinematics from the robot's design description files.
- Implementation of motor babbling and goal babbling methods using a multi-layer perceptron (MLP).
- Extension of these methods to include the orientation of the end effector.
- Implementation of an approach to learn relative IK.
- Parameter optimization and performance comparison of the methods.
- Implementation of basic trajectory generation for task space movements.
- Tests of the practical applicability of all the developed methods in simulation.
- Final testing on the real robot.

Conclusions

The results of the approaches to learning the IKs of the robot arm showed significant differences in their precision, workspace coverage, and most important their practical usability.

The precision of three-dimensional motor babbling suffers significantly in the center of the workspace due to the influence of redundant configurations. On the other hand, it learns reliably and shows the best performance at the periphery of the workspace. The errors in the middle have the form of an offset, this limits their influence on relative movements. Both the amount of hidden neurons and the learning rate have limited influence on the performance. This hints at the limitations in the model: the MLP is not capable of dealing with redundant configurations and the discontinuities at the joint limits, regardless of those parameters.

Three-dimensional goal babbling achieved a very high precision in the center. This is due to both the inclusion of a home position which is defined by its joint configuration, as well as the weighting scheme which favors efficient movements. Outside of the center the algorithm showed significant difficulty in further exploration. The reason is that, due to the complex structure of the workspace, straight movements cannot traverse it. This in conjunction with the fact that such positions are more difficult to learn for an MLP, as shown by the motor babbling approach, leads to high errors further away from the joint medians.

The inclusion of the orientation in six-dimensional motor babbling results in a significantly higher precision in the center of the workspace. The outer areas show slightly worse results. This can be due to the added complexity of including the orientation or the choice of data generation, since the six-dimensional method included realistic movements, rather than randomly generated positions. Unlike the three-dimensional method, this approach benefits from the added capabilities of a higher number of neurons in the hidden layer.

Goal babbling can also make use of the added orientation information. While the precision in the center does not benefit, the other areas are explored more reliable. This is true only for the more complex neural networks: as with six-dimensional motor babbling, goal babbling in higher dimensions does benefit from the added capabilities. The effect is even more pronounced for goal babbling: since further exploration depends on the achieved precision in the known areas, exploration can halt for suboptimal solutions. Six-dimensional goal babbling still has troubles reaching the outer regions of the workspace, the complexity of it remains a problem. This work has shown that goal babbling can be performed in a six-dimensional target space which includes the orientation. But, for this complex workspace and a comparatively low amount of degrees of freedom in the manipulator, it gets outperformed by motor babbling.

Confining the workspace to a certain region by reducing the joint limits improves the accuracy greatly. This is due to not only the overall reduced problem size, but also the simpler structure of the workspace and the lower influence of redundancy. The disadvantage lies in the arbitrary restriction of the capabilities of the manipulator.

The relative IK solution shows a high error in comparison to the attempted movements. The overall learning is slow regardless of the chosen NN parameters, but more complex networks do achieve a higher performance. The slow learn can be due to the very large input space: the network needs to learn about all possible move directions for each joint space position.

When using these solutions to move the robot arm in simulation, the results achieved by the three-dimensional motor babbling method prove to be the most viable. While the accuracy in the center is low, the more consistent movements outside of it outweigh it. When using relative movements, the position errors are less notable, since they have the structure of an offset. Using the current goal babbling solution can be an option if a higher accuracy in the workspace center is needed. The six-dimensional methods, while offering a higher precision, are unreasonably difficult to use. The NN always requires the orientation to be defined and generating achievable end effector orientations for given positions is not trivial. Trying to overcome this by always using the current orientation is not successful, the errors add up too quickly. Switching to a six-dimensional option after achieving a position just to control the orientation has two disadvantages: the position will be changed due to different offsets and the orientation can be better adjusted by directly controlling the wrist joint. The relative IK method has a high error when used without any corrections. When the goal direction is updated after each step the accuracy is improved, but the generated movements are often curved towards the goal.

The generated movements translate well to the real robot. While its dynamic behavior still needs improvement, mostly by optimizing the parameters of its joint servos, it follows the trajectories as expected. This enables the robot to perform grasping movements.

Future Work

While the overall task of enabling the robot to use its arms in a goal directed manner is achieved, each of the approaches used to learn the IK have specific shortcomings and opportunities for enhancements.

The goal babbling methods have troubles exploring the outer regions of the complex workspace. This might be alleviated by switching from Cartesian task space coordinates to spherical coordinates as it would allow the goal babbling movements to traverse the task space more easily [41].

Even when using six-dimensional descriptions of the end effector configuration, which eliminate redundancies, the MLP has significant difficulties in learning a reasonable solution. One influence can be the non-continuities at the workspace boundaries. Another problem can be that the learning process at a distinct position can affect the whole network. As such, a high gradient in distant positions can have a negative impact on the global learning process. Therefore, other regression techniques such as locally-linear maps might be better suited [42, 50].

Learning is currently done using the forward kinematics solution. Switching to the real robot can be advantageous as positioning errors and mechanical defects can be accounted for. This requires measurement of the end effector position, likely using to a camera system such as NICOs stereo vision. As this limits the amount of data points that can be created in reasonable time, the methods need to be optimized according to that. Using motor babbling to generate the data points and then performing offline learning can be a reasonable option, as it disjoints the data creation from the training. This would allow optimizing the regression without any restrictions.

Currently the robot still shows some jerky movements. This can be alleviated by optimizing the parameters of the joint servos, for example using the Ziegler-Nichols tuning rules [57]. The movements of the robot sometimes do not resemble those of human being. This is partially due to the elbow joint which can bend both ways, unlike the human elbow. Restricting this joint would reduce the workspace, but could make the movements look more akin to humans.

Appendix A

Nomenclature

Symbol	**Units**	**Description**
$\{A\}$	$-$	frame of reference
e_θ	$-$	random joint perturbation
F_{ee}	$-$	generalized forces and torques at the end effector
G	$-$	vector of gravity terms
M	$-$	mass matrix
J	$-$	robot Jacobian matrix
s_{ee}	m	end effector position
R	$-$	rotation matrix
T	$-$	transform
t	s	time
V	$-$	vector of centrifugal and Coriolis forces
v_{ee}	$\frac{m}{s}$	end effector speed
x_i	$-$	neural network input
y_i	$-$	neural network output
W	$-$	neural network weights
w_d	$-$	goal babbling weighting for movements in teh right direction
w_e	$-$	goal babbling weighting for efficient movements
w_t	$-$	total goal babbling weighting
ϵ_X	$-$	goal direction
θ	rad	joint position
$\dot{\theta}$	$\frac{rad}{s}$	joint speed
τ	Nm	joint torque

Appendix B

Instructions for the Source Code

Prerequisites

The code relies on the following software packages and frameworks:

- Python version 2.7.12
- Pybrain version 0.3
- V-REP version 3.2.3
- Numpy version 1.8.2
- Pypot version 2.10.0
- Matplotlib version 1.5.1
- transformations.py version 2015.07.18

Other versions may also work, but are untested.

Description of the Source Files

robot.py Contains the URDF parser and the forward kinematics solution.

neuralNetwork.py Wrapper for Pybrain. Contains the structure of the neural network.

inverseKinematics.py Contains classes to handle motor and goal babbling in three dimensions. Also used to to give solutions for the inverse kinematics.

inverseKinematics6D.py Different version used for 6D.

movingRobot.py Contains methods to handle moving the robot, both the simulation and the real robot. Contains trajectory generation for linear and point to point movements. Can use the standard 3D ik solution or the relative version.

movingRobot6D.py The same as for the 3D version, can additionally handle rotations.

trainingXX.py Script files automating learning across different configurations.

printLearningCurves.py Script to automatically generate learning and testing curves.

Usage Scenarios

Moving the Robot

At the beginning of the movingRobot.py file are a couple of bool variables that are used to configure the environment. important is isSimulation: if set to true, the simulation is controlled, false tries to control the real robot. Also the correct .json file and neural network need to be specified inside the code. A sample movement can look like this:

```
myNico = robotInPypot()
myNico.switchNN("absoluteIK")
time.sleep(2)
myNico.moveToPoint([0.0, -0.3, 0.8], 10)
time.sleep(2)
myNico.moveToPoint([0.30, -0.2, 0.8], 10)
myNico.openGripper()
time.sleep(2)
myNico.moveLinRel([0.05, 0.0, 0.0], 5)
myNico.closeGripper()
```

Training the Neural Network

The training can be done by either the trainingXX.py script files or directly by the inverseKinematics files. The training files handle using multiple configurations and saving the learning progress for later plotting. Training a neural network via motor babbling can for example be done like this:

```
myIK = motorBabblingIK(120, 0.01) #120 Hidden Neurons, 0.01 LR
myIK.learnIKBatched(10000000, 1000) #10M Iterations
print myIK.testIK(0.1)
myIK.plotTestPoints('TestPositionsClose.pdf')
print myIK.testIK(0.5)
myIK.plotTestPoints('TestPositionsMedium.pdf')
print myIK.testIK(0.75)
myIK.plotTestPoints('TestPositionsFar.pdf')
print myIK.testIK(1)
myIK.plotTestPoints('TestPositionsLimit.pdf')
myIK.plotDiscoveredPointsMB('ExploredPositions.pdf')
myIK.myNN.saveWeights("TrainedNetwork.xml") #saves the NN
with open("MotorBabbling_learning.json", 'w') as f:
        json.dump(myIK.testResults, f) #saves the learning progress
f.closed
```

Bibliography

[1] Adept Technology Inc. Adept cobra s 600 datasheet. URL: `www.adept.com/products/robots/scara/cobra-s600/`.

[2] Jose Alvarez-Ramirez, Ilse Cervantes, and Rafael Kelly. Pid regulation of robot manipulators: stability and performance. *Systems & control letters*, 41(2):73–83, 2000.

[3] Atif Mahboob. *Mechanical Design of the arms and Neural Arm control for the Humanoid Robot Platform Nimbro-OP*. M.sc. thesis, University of Hamburg, 2015.

[4] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[5] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods (april 2004). *Unpublished, available at http://www. math. ucsd. edu/˜ sbuss/ResearchWeb/ikmethods/ik-survey. pdf*, 2004.

[6] Fabrizio Caccavale, Ciro Natale, Bruno Siciliano, and Luigi Villan. Integration for the next generation: embedding force control into industrial robots. *Robotics & Automation Magazine, IEEE*, 12(3):53–64, 2005.

[7] Stephen K Chan and Peter D Lawrence. General inverse kinematics with the error damped pseudoinverse. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 834–839. IEEE, 1988.

[8] R Clavel. Device for the movement and positioning of an element in space, 1990. US 4976582.

[9] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[10] Yiannis Demiris and Anthony Dearden. From motor babbling to hierarchical learning by imitation: a robot developmental pathway. 2005.

[11] Jacques Denavit. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. of the ASME. Journal of Applied Mechanics*, 22:215–221, 1955.

[12] Diconary.com LLC. Dictionary.com unabridged, May 2016. URL: `http://www.dictionary.com/browse/robot`.

[13] Fanuc Robotics America Inc. F-200ib datasheet, 2005. URL: `www.fanucamerica.com/cmsmedia/datasheets/F-200iB%20Series_9.pdf`.

[14] Inria Flowers. Pypot sdk documentation, 2016. URL: `github.com/poppy-project/pypot`.

[15] Michael A Goodrich and Alan C Schultz. Human-robot interaction: a survey. *Foundations and trends in human-computer interaction*, 1(3):203–275, 2007.

[16] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jerome Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. The nao humanoid: a combination of performance and affordability. *CoRR abs/0807.3223*, 2008.

[17] H Heiss. Roboterbewegungen mit bahninterpolation und überschleifen. *VDI BERICHTE*, 1094:569–569, 1993.

[18] Clint Heyer. Human-robot interaction and future industrial robotics applications. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4749–4754. IEEE, 2010.

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[20] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[21] Trade Industrial Machinery Division Ministry of Economy and Japan Industry.

[22] Hiroshi Ishiguro. Interactive humanoids and androids as ideal interfaces for humans. In Ernest Edmonds, Doug Riecken, Cécile L. Paris, and Candace L. Sidner, editors, *the 11th international conference*, page 2, 2006. `doi:10.1145/1111449.1111451`.

[23] Lydia E Kavraki, Jean-Claude Latombe, and E Latombe. Probabilistic roadmaps for robot path planning. 1998.

[24] Charles A Klein and Ching-Hsiang Huang. Review of pseudoinverse control for use with kinematically redundant manipulators. *Systems, Man and Cybernetics, IEEE Transactions on*, (2):245–250, 1983.

[25] Kuka AG. Kuka kr 60-3 datasheet. URL: `www.kuka-robotics.com/usa/en/products/industrial_robots/medium/kr60_3/`.

[26] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[27] ABB Ltd. Datenblatt abb irb 360, 2013. URL: `new.abb.com/products/robotics/de/industrieroboter/irb-360`.

[28] Robotis Ltd. Dynamixel sdk documentation, 2010. URL: `support.robotis.com/en/techsupport_eng.htm#software/dynamixelsdk.htm`.

[29] Anthony A Maciejewski and Charles A Klein. The singular value decomposition: Computation and applications to robotics. *The International journal of robotics research*, 8(6):63–79, 1989.

[30] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM, 2008.

[31] Michael J Milford, Gordon F Wyeth, and DF Rasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE, 2004.

[32] Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 241–248. ACM, 2003.

[33] Romeo Ortega and Mark W Spong. Adaptive motion control of rigid robots: A tutorial. *Automatica*, 25(6):877–888, 1989.

[34] Donald L Peiper. The kinematics of manipulators under computer control. Technical report, DTIC Document, 1968.

[35] Donald L. Peiper. *The kinematics of manipulators under computer control.* Ph.d. dissertation, Stanford University of California, Department Of Computer Science, 1968.

[36] Pramila Rani, Changchun Liu, Nilanjan Sarkar, and Eric Vanman. An empirical study of machine learning techniques for affect recognition in human–robot interaction. *Pattern Analysis and Applications*, 9(1):58–69, 2006.

[37] René Felix Reinhart and Matthias Rolf. Learning versatile sensorimotor coordination with goal babbling and neural associative dynamics. In *Development and Learning and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference on*, pages 1–7. IEEE, 2013.

[38] Freese Rohmer, Singh. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[39] Raúl Rojas. *Neural networks: a systematic introduction.* Springer Science & Business Media, 2013.

[40] Matthias Rolf. Goal babbling with unknown ranges: A direction-sampling approach. In *2013 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–7, 2013. `doi:10.1109/DevLrn.2013.6652526`.

[41] Matthias Rolf and Jochen J Steil. Efficient exploratory learning of inverse kinematics on a bionic elephant trunk. *IEEE Transactions on Neural Networks and Learning Systems*, 25(6):1147–1160, 2014.

[42] Matthias Rolf, Jochen J. Steil, and Michael Gienger. Goal babbling permits direct learning of inverse kinematics. *IEEE Transactions on Autonomous Mental Development*, 2(3):216–229, 2010. `doi:10.1109/TAMD.2010.2062511`.

[43] Matthias Rolf, Jochen J. Steil, and Michael Gienger. Online goal babbling for rapid bootstrapping of inverse models in high dimensions. In *2011 IEEE International Conference on Development and Learning (ICDL)*, pages 1–8, 2011. `doi:10.1109/DEVLRN.2011.6037368`.

[44] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[45] George A Rovithakis and Manolis A Christodoulou. *Adaptive control with recurrent high-order neural networks: theory and industrial applications.* Springer Science & Business Media, 2012.

[46] Ryo Saegusa, Giorgio Metta, Giulio Sandini, and Sophie Sakka. Active motor babbling for sensorimotor learning. In *Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on*, pages 794–799. IEEE, 2009.

[47] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

[48] M. Schwarz, M. Schreiber, S. Schueller, M. Missura, and S. Behnke. Nimbro-op humanoid teensize open platform. In *Proceedings of 7th Workshop on Humanoid Soccer Robots*, volume 2012, 2012.

[49] Spong, Hutchinson, and Vidyasagar. *Robot modeling and control.* John Wiley and Sons, 2006.

[50] Jörg Walter and Helge Ritter. Rapid learning with parametrized self-organizing maps. *Neurocomputing*, 12(2):131–153, 1996.

[51] Herbert Werner. Lecture notes neural and genetic computing for control engineering, 2013.

[52] Bernard Widrow and Michael A Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

[53] William A Wolovich and H Elliott. A computational technique for inverse kinematics. In *Decision and Control, 1984. The 23rd IEEE Conference on*, pages 1359–1363. IEEE, 1984.

[54] Keenan A Wyrobek, Eric H Berger, HF Van der Loos, and J Kenneth Salisbury. Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2165–2170. IEEE, 2008.

[55] Ganwen Zeng and Ahmad Hemami. An overview of robot force control. *Robotica*, 15(05):473–482, 1997.

[56] Hui Zhang, Jianjun Wang, George Zhang, Zhongxue Gan, Zengxi Pan, Hongliang Cui, and Zhenqi Zhu. Machining with flexible manipulator: toward improving robotic machining performance. In *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pages 1127–1132. IEEE, 2005.

[57] John G Ziegler and Nathaniel B Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

Acronyms

AIS Autonomous Intelligent Systems Institute

ANN artificial neural network

API application programming interface

CAD computer-aided design

dof degree of freedom

IK inverse kinematics

IR industrial robot

ML machine learning

MLP multi-layer perceptron

NICO Neural-inspired companion

NN neural network

PID proportianal-integral-derivative

ROK Republik of Korea

ROS robot operating system

SCARA selective compliance assembly robot arm

SGD stochastic gradient descent

URDF unified robot description format

USB universal serial bus

WTM Knowledge Technology Group