Inherently Constraint-Aware Control of Many-Joint Robot Arms with Inverse Recurrent Models

Sebastian Otte[1], Adrian Zwiener[2], and Martin V. Butz[1]

[1] Cognitive Modeling Group, University of Tübingen
Sand 14, 72076 Tübingen, Germany
[2] Cognitive Systems Group, University of Tübingen
Sand 1, 72076 Tübingen, Germany

Abstract. In a recent study, it was demonstrated that Recurrent Neural Networks (RNNs) can be used to effectively control snake-like, many-joint robot arms in a particular way: The inverse kinematics for control are generated using back-propagation through time (BPTT) on recurrent forward models that learned to predict the end-effector pose of a robot arm, whereby each joint is associated with a certain computation time step of the RNN. This paper further investigates this approach in terms of constraint-aware control. Our contribution is twofold: First, we show that an RNN can be trained to also predict the poses of intermediate joints within such an arm, and that these can consequently be included in the control-optimization objective as well, giving full control over the entire arm. Second, we show that particular components of the arm's target can be selectively switched on and off by means of "don't care" signals. This enables us to handle constraints inherently and on-the-fly, without the need of any outer constraint mechanisms, such as additional penalty terms. The experiments demonstrating the effectiveness of our methodology are carried out on a simulated three dimensional 40-joint robot arm with 80 articulated degrees of freedom.

Keywords: Recurrent Neural Networks, Long Short-Term Memory, Neurorobotics, Robot Control, Robot Arm, Constraint Handling

1 Introduction

Handling many-joint robot arms is usually challenging in terms of control and planning. Recently, it was shown that recurrent neural forward models can be used to compute the inverse kinematics of many-joint robot arms [14]. Specifically, variants of *Long Short-Term Memory* (LSTM) [8, 12] were trained to estimate end-effector poses given specified arm configurations. The forward computation unfolds in a sequential manner, whereby the projection through each joint of the arm is computed via one recurrent iteration in the RNN. Thus, the recurrences match the sequential nature of computing kinematic forward-chains and the LSTM structure provides highly accurate estimates. *Back-propagation* *through time* (BPTT) was used to iteratively optimize the goal-oriented inverse mapping, which induces the goal-directed movement of the robot arm; essentially enacting the unfolding goal-directed optimization process. From a computational neuroscience perspective this is closely linked to *active inference* in that action control is inversely inferred by the imagination of the future goal state [3–5].

Previous related approaches have implemented distributed mathematical models of the arm's local relative kinematics and induced control by means of the derivatives of the model [15, 2]. In contrast, our approach learns the local relative kinematics by an RNN. Particularly, the combination of LSTMs and the application of BPTT during inference time allows flexible goal-directed arm control facing a much larger number of redundant joints. As shown previously [14], our RNN-based arm control approach scales well even for arms with up to 120 articulated degrees of freedom (DoF), training the forward model based on *stochastic gradient descent* (SGD) with momentum term.

Here, we focus on constraint-aware control. Originally, the recurrent forward model was trained to predict the pose of the end-effector only [14]. As a result, the output of intermediate computation steps reflects the internal representation of the developing end-effector pose estimate. An according analysis has shown that the intermediate outputs did not even approximately match the respective joint poses. Moreover, it appeared to be surprisingly difficult to learn when the network was forced to develop an internal representation that allows the prediction of all intermediate joint poses. For full control over the entire arm, however, it is essential to provide these intermediate poses – referred to as *pose chain* in the following – to enable the induction of joint-specific constraints and optimizations.

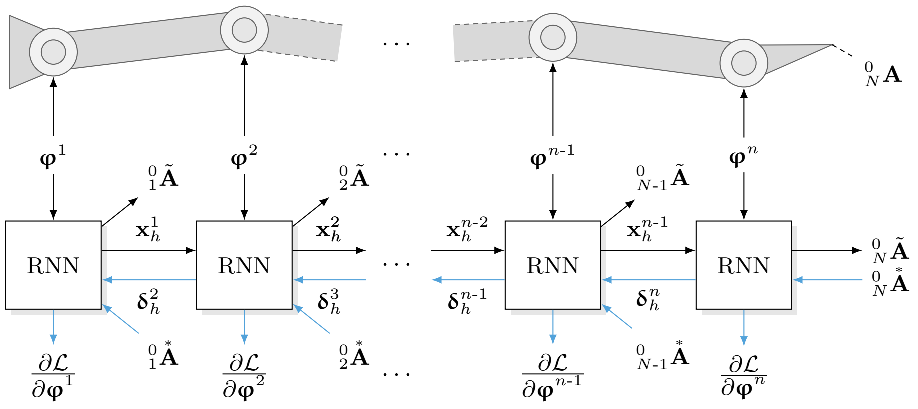
This paper tackles this issue by means of a more fine-granular learning setup as well as by using *Adam* [9] for training. Moreover, it is shown that all particular components of the entire pose chain can be selectively included or excluded (left free) from the optimization objective, where we call the latter a "*don't care*" signal. As a result, arm specific constraints can be formulated in work space easily and are inherently considered by the system on-the-fly and in an biological plausible, active-inference-like manner [3–5], without the need for preplanned trajectories or additional constraint-specific penalizations. The effectiveness of our modifications is demonstrated in several scenarios with a simulated 40-joint robot arm with 80 DoF.

2 Inverse Recurrent Model

To enable control via BPTT, an RNN is trained to approximate a forward model M, which maps a robot arm configuration state, that is, a sequence of angle vectors $\boldsymbol{\varphi}^j$, onto the corresponding pose chain:

$$\boldsymbol{\Phi} = \left(\boldsymbol{\varphi}^1, \ldots, \boldsymbol{\varphi}^n\right) \xmapsto{\mathrm{M}} \left({}^0_1\mathbf{A}, \ldots, {}^0_N\mathbf{A}\right), \tag{1}$$

where ${}^0_j\mathbf{A} \in \mathbb{R}^{4\times 4}$ refers to the reference frame transformation of the j-th joint and N denotes the end-effector frame. Each ${}^0_j\mathbf{A}$ can be decomposed into the



$^{0}_{N}\mathbf{A}$
$\boldsymbol{\varphi}^1$
$\boldsymbol{\varphi}^2$
$\boldsymbol{\varphi}^{n\text{-}1}$
$\boldsymbol{\varphi}^n$
$^{0}_{1}\tilde{\mathbf{A}}$
$^{0}_{2}\tilde{\mathbf{A}}$
$^{0}_{N\text{-}1}\tilde{\mathbf{A}}$
$\mathbf{x}_h^1$
$\mathbf{x}_h^2$
$\mathbf{x}_h^{n\text{-}2}$
$\mathbf{x}_h^{n\text{-}1}$
RNN
RNN
RNN
RNN
$^{0}_{N}\tilde{\mathbf{A}}$
$^{0}_{N}\overset{*}{\mathbf{A}}$
$\boldsymbol{\delta}_h^2$
$\boldsymbol{\delta}_h^3$
$\boldsymbol{\delta}_h^{n\text{-}1}$
$\boldsymbol{\delta}_h^n$
$^{0}_{1}\overset{*}{\mathbf{A}}$
$^{0}_{2}\overset{*}{\mathbf{A}}$
$^{0}_{N\text{-}1}\overset{*}{\mathbf{A}}$
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\varphi}^1}$
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\varphi}^2}$
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\varphi}^{n\text{-}1}}$
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\varphi}^n}$

Fig. 1. Computing the inverse mapping using BPTT. An input sequence (the current state of the arm) is presented to RNN in a sequential manner, which produces and estimate of the pose chain. The discrepancy between the output and the desired pose chain is back-propagated through the network (blue lines) and mapped onto the input sequence.

joint's orientation, which is given by the orthonormal base $^{0}_{j}\mathbf{R} \in SO(3) \subset \mathbb{R}^{3\times 3}$, and its translation, that is, its relative position, given by $^{0}_{j}\mathbf{p} \in \mathbb{R}^3$. Note that it is not necessary to explicitly model the lengths of the limbs as they can be inherently learned by means of trainable biases.

To calculate such a mapping with an RNN, each joint transformation is considered as a "computing time-step" in the RNN. Accordingly, the RNN requires only k input neurons, where k is the number of angles per joint – two in this paper. Thus, the computation is fully independent from the number of joints. The angle vectors $\boldsymbol{\varphi}^j$ are presented to the network in a sequential manner. As a result, the RNN is forced to use its recurrences to handle the repetitive character of computing chains of mostly very similar transformations [14]. After the RNN is trained on a sufficiently rich pool of training pairs, it is able to predict the pose chain of the arm given a sequence of angle vectors.

To control the arm, it is necessary to compute the inverse mapping, that is, an appropriate angle sequence given a desired pose chain. How this is achieved can best be explained by considering Fig. 1. First, the current arm configuration $\mathbf{\Phi}$ is processed by the RNN sequentially, producing corresponding pose chain estimates $(^{0}_{1}\tilde{\mathbf{A}}, \ldots, ^{0}_{N}\tilde{\mathbf{A}})$. The discrepancies (loss) $\mathcal{L}$ between this estimated and the desired pose chain $(^{0}_{1}\overset{*}{\mathbf{A}}, \ldots, ^{0}_{N}\overset{*}{\mathbf{A}})$ are back-propagated reversely through the unfolded RNN. The resulting input gradients are thus computed via

$$\frac{\partial\mathcal{L}}{\partial\varphi_i^j} = \sum_{h=1}^{H}\left[\frac{\partial net_h^j}{\partial\varphi_i^j}\frac{\partial\mathcal{L}}{\partial net_h^j}\right] = \sum_{h=1}^{H} w_{ih}\delta_h^j, \tag{2}$$

projecting the loss back onto the input sequence, where h indexes the hidden units and net_h^j denotes the weighted sum of inputs (or *net input*) into unit h at computation step j. Starting from any possible arm configuration, by following the negative gradient through the joint space in an iterative manner, a possible solution to the inverse mapping is generated. We thus update the joint angles in the following manner, which is essentially SGD with momentum:

$$\mathbf{\Phi}(\tau+1) \longleftarrow \mathbf{\Phi}(\tau) - \eta \nabla_{\mathbf{\Phi}(\tau)} \mathcal{L} + \mu \left[\mathbf{\Phi}(\tau) - \mathbf{\Phi}(\tau-1) \right], \tag{3}$$

where τ denotes the current iteration step, $\eta \in \mathbb{R}$ is a gradient scale factor (cf. learning rate in gradient descent learning), and the momentum is scaled with the rate $\mu \in \mathbb{R}$ (i.e., $\mu \approx 0.5$), which accelerates convergence when the gradient signal is weak. Independently of this tuning parameters, we also restrict the maximum update step size to regularize relatively high gradients, which results in a more uniform motion behavior.

As proposed previously [14], we also apply a target correction step, compensating the error of the forward model. This can be done when the real forward model is accessible during the optimization – for instance, by means of a mathematical formulation or a (visual) feedback mechanism. Instead of presenting the desired targets, encoded as vectors $\mathbf{z}^j \in \mathbb{R}^9$, we present "modified" versions $\tilde{\mathbf{z}}^j$ to the network when computing the loss. Let $\mathbf{u}^j \in \mathbb{R}^9$ be the true current pose and $\mathbf{y}^j \in \mathbb{R}^9$ the pose prediction of the RNN. We thus compute $\tilde{\mathbf{z}}^j$ with respect to a given $\mathbf{\Phi}$ as follows:

$$\tilde{\mathbf{z}}^j = \begin{bmatrix} \left[y_i^j + \gamma_1 (z_i^j - u_i^j) \right]_{1 \leq i \leq 3} \\ \left[y_k^j + \gamma_2 (z_k^j - u_k^j) \right]_{4 \leq k \leq 9} \end{bmatrix}, \tag{4}$$

where $\gamma_1, \gamma_2 \in [0, 1]$ are additional scaling factors, which scale the influence of the positional and the orientation discrepancy, respectively. This modification causes the RNN to converge towards the real target pose with high precision, effectively compensating for remaining forward model errors.

3 Selective Component Constraining

As formulated in the upper formalization, controlling the arm requires the presence of a full chain of desired poses with all associated components. Clearly this is impractical – especially for arms with lots of joints. It would be better if only particular components could be selectively included in the optimization process, while all other components should be optimized automatically. For instance, when moving a cup with a fluid in it, it is important to maintain a horizontal end-effector orientation, while the direction in the horizontal plain is not directly relevant. Previous studies have shown that such constraints can, for instance, be inferred by a programming by demonstration paradigm [1] or by means event boundary signal-oriented inference [7].

To enable the selective induction of constraints, we propose to use "don't care" signals, which we define as respective zero gradients in the unconstrained components. That is, don't cares do not induce any additional gradient signals to the backward pass, regardless of their forward pass estimates. As a result, full and arbitrarily selective control of the robot arm's behavior becomes possible.

4 Experimental Results

In this paper we focused on a simulated three dimensional 40-joint robot arm. Each joint can rotate along the x and the y axis, which is physically realizable easily. The entire arm thus has $2 \cdot 40 = 80$ DoF. For control, we used $\gamma_1 = 1.0, \gamma_2 = 0.1$ to equalize the magnitude of the position and orientation-induced gradients, as otherwise the orientation gradient would be numerically dominant.

Preliminary studies have shown that when training with SGD, it is difficult – for arms with 20+ joints even impossible – to learn to predict the poses of all intermediate joints (cf. Section 1). The following modifications ensured learning success: First, we dropped SGD and, instead, applied Adam [9], which is effectively more robust to gradient fluctuations and local minima, using the parameters $\beta_1 = 0.9, \beta_2 = 0.999$ (smoothing factors of the first two moment estimates) and a cautious learning rate of $\eta = 10^{-4}$. Second, we remodeled the training, into ten training episodes, which consisted of respective, randomly generated arm configurations, where the joint angle ranges were limited to 10 %, 20 %,30 % and so forth of the full range. The first nine sets contained 2 000 training examples, each, whereas the tenth set – in which the full angle ranges (here $\pm 45°$) are covered – contains 20 000 examples. In each training episode, 50 epochs were performed. The smallest possible amount of training data was not investigated.

The used RNN architecture consisted of two hidden layers with 24 LSTM blocks with intra-block connected gates [12]. This LSTM type is advantageous in regression tasks [13]. Each hidden block contained three inner cells and has variable biases for cells and gates, which is helpful when the computation involves spatial mappings [14]. Additionally, each hidden layer was not recurrently connected to itself, but both hidden layers were mutually fully connected. This was the best architecture discovered previously [14]. Our experiments have shown that this architecture, in combination with the training procedure detailed above, produces well performing RNNs reliably. All experiments were performed using the JANNLab neural network framework [11].

Fig. 2 shows results when the task was to keep the end-effector pose horizontal, while approaching a certain target position. The was realized by presenting only the goal position and the upwards-orientation as targets, while the other components of the end-effector had assigned don't care signals. Fig. 3 shows a related scenario, in which the goal was to follow an elliptic trajectory on a 2D vertical plane with the end-effector, while the orientation of the end-effector should remain horizontal. Again, the remaining components of the end-effector orientation are left free, as are all other components.

Fig. 4 shows that it is also easily possible to fully fix the end-effector pose, while introducing targets to intermediate parts of the arm. In this example, the (x,y) position of the 20th joint was optimized, by following points on a virtual circle around the main axis of the arm.

Finally, Fig. 5 demonstrates further possibilities of selective constraining joint components. Note that these kinds of constraints, and also those shown above, can be flexibly switched on and off on-the-fly, depending on the requirements of the current scenario or the current task.

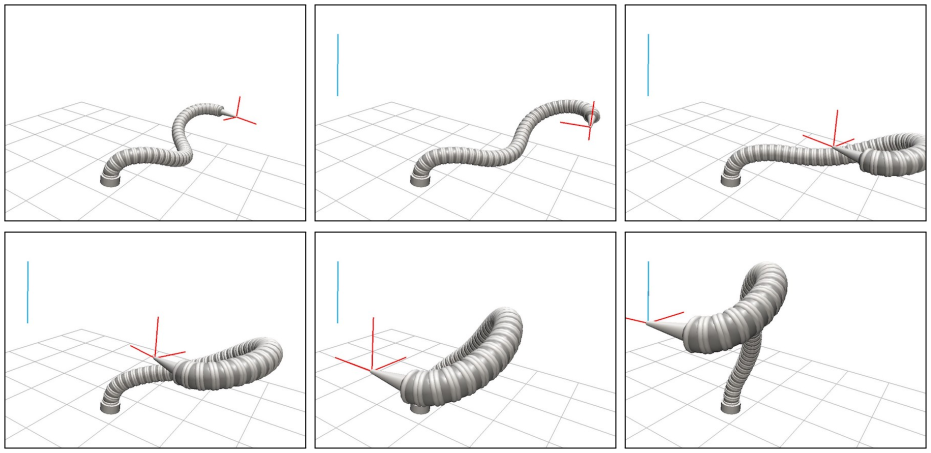


Fig. 2. This images sequence depicts the movement towards the target while an upright end-effector orientation is maintained (e.g for handling fluids). The forward direction of the end-effector is assigned with a "don't care" signal, thus it is effectively ignored during the optimization, that is, the movement.

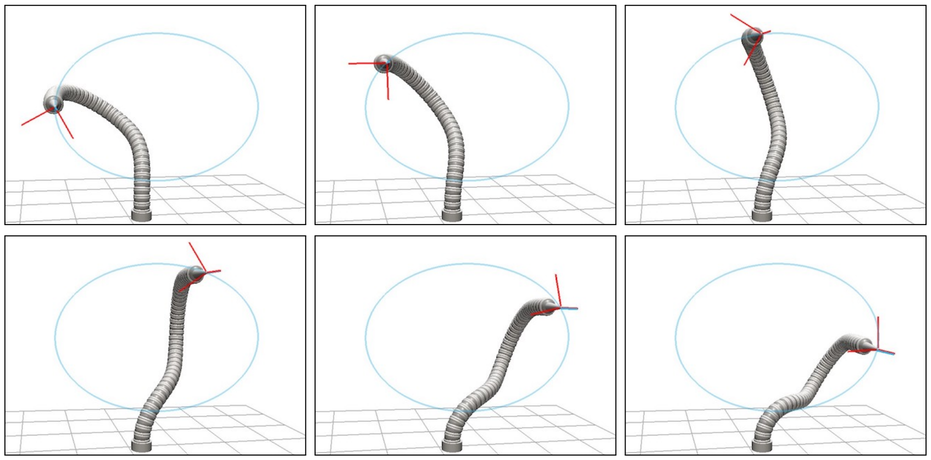


Fig. 3. Image sequence of a drawing/welding scenario. The target position for the end-effector tip moves along an elliptic trajectory on a 2D plane nearby the robot arm. The "forward" orientation of the target is fixed to the negative normal vector of the plane, while the "up" orientation is left free.

5 Summary and Conclusion

In this paper, we investigated and extended a procedure for RNN-based robot arm control [14]. Specifically, we showed that an RNN can be trained to predict the poses of intermediate joints within such an arm and that these can consequently be included in the controlling objective, enabling full control over

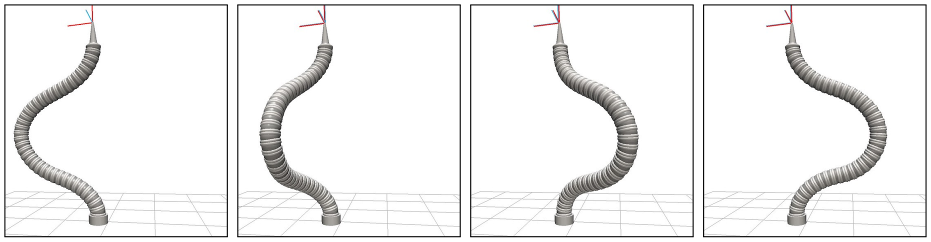


Fig. 4. Selective intermediate joint control. The x,y position of the 20th joint is defined following a circle around the main axis, while the end-effector pose is fixed. All other components of the arm (including the z position of the 20th joint) are left free ("don't care" signal).

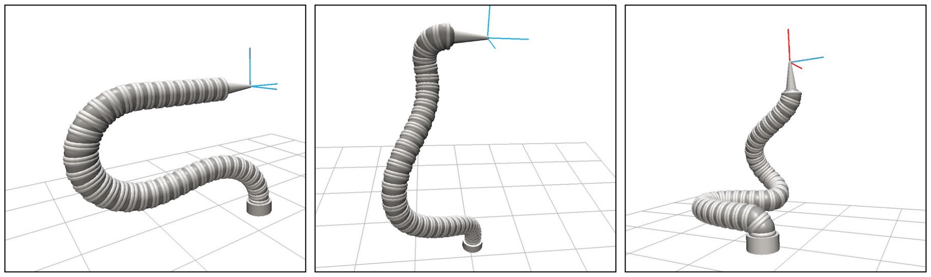


Fig. 5. Examples demonstrating intermediate joint control. In the left image the last 10 joints are constrained to have to same forward orientation as the end-effector target. In the center image the target of the "forward" direction of the sixth-last joint is set to the "up" direction of the end-effector target. In the right image, the "up" targets for the first 20 joints are set to the world's up direction (z-axis).

all joints of an arm. Moreover, we showed that particular components of the arm's target can be selectively switched on and off by means of a "don't care" signal. This enabled us to handle constraints inherently and on-the-fly, without the need of any outer constraint mechanism, such as additional penalty terms. In several scenarios with a simulated three dimensional 40-joint robot arm, we demonstrated the effectiveness of our procedure.

The presented results underline the potential of this RNN-based approach for robot arm control. The approach is essentially inspired by an active inference [3–5] perspective on robot control, inferring motor commands by back-projecting error gradients between current and desired system states. We believe that similar techniques could be of practical interest for the novel field of continuum robots (c.f. e.g. [10]), for which precise and goal-directed control is still a major challenge.

Our future research will consider the addition of higher-level constraints, such as curvature optimization, by means of adding "constraint neurons", which will allow tuning the associated properties of the arm. Furthermore, we plan to include the dynamics of the arm as well, which requires extending the architecture in a multi-dimensional fashion [6]. Finally, we plan to combine this approach with options to project trajectory imaginations into the future, thus enabling trajectory optimization besides the current goal-directed arm state optimizations addressed in this paper.

References

1. Calinon, S., Guenter, F., Billard, A.: On learning, representing, and generalizing a task in a humanoid robot. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 37(2), 286–298 (2007)
2. Ehrenfeld, S., Butz, M.V.: The modular modality frame model: Continuous body state estimation and plausibility-weighted information fusion. Biological Cybernetics 107, 61–82 (2013)
3. Friston, K.: The free-energy principle: a rough guide to the brain? Trends in cognitive sciences 13(7), 293–301 (2009)
4. Friston, K.: The free-energy principle: a unified brain theory? Nature Reviews Neuroscience 11(2), 127–138 (2010)
5. Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., Pezzulo, G.: Active Inference: A Process Theory. Neural Computation 29(1), 1–49 (Nov 2016)
6. Graves, A., Fernández, S., Schmidhuber, J.: Multi-dimensional recurrent neural networks. In: Sá, J.M.d., Alexandre, L.A., Duch, W., Mandic, D. (eds.) Artificial Neural Networks – ICANN 2007, pp. 549–558. No. 4668 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Sep 2007)
7. Gumbsch, C., Kneissler, J., Butz, M.V.: Learning behavior-grounded event segmentations. In: Papafragou, A., Grodner, D., Mirman, D., Trueswell, J.C. (eds.) Proceedings of the 38th Annual Meeting of the Cognitive Science Society. pp. 1787–1792. Cognitive Science Society, Austin, TX (2016)
8. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput. 9(8), 1735–1780 (Nov 1997)
9. Kingma, D.P., Ba, J.L.: Adam: A method for stochastic optimization. 3rd International Conference for Learning Representations abs/1412.6980 (2015)
10. Neumann, M., Burgner-Kahrs, J.: Considerations for follow-the-leader motion of extensible tendon-driven continuum robots. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). pp. 917–923 (May 2016)
11. Otte, S., Krechel, D., Liwicki, M.: JANNLab Neural Network Framework for Java. In: Poster Proc. MLDM 2013. pp. 39–46. ibai-publishing, New York, USA (2013)
12. Otte, S., Liwicki, M., Zell, A.: Dynamic Cortex Memory: Enhancing Recurrent Neural Networks for Gradient-Based Sequence Learning. In: Artificial Neural Networks and Machine Learning – ICANN 2014, pp. 1–8. No. 8681 in LNCS, Springer Int. (Sep 2014)
13. Otte, S., Liwicki, M., Zell, A.: An Analysis of Dynamic Cortex Memory Networks. In: International Joint Conference on Neural Networks (IJCNN). pp. 3338–3345. Killarney, Ireland (Jul 2015)
14. Otte, S., Zwiener, A., Hanten, R., Zell, A.: Inverse Recurrent Models – An Application Scenario for Many-Joint Robot Arm Control. In: Artificial Neural Networks and Machine Learning – ICANN 2016. pp. 149–157. No. 9886 in LNCS, Springer Int. (Sep 2016)
15. Schilling, M.: Universally manipulable body models – dual quaternion representations in layered and dynamic MMCs. Autonomous Robots 30, 399–425 (2011)