Towards Learning Inverse Kinematics with a Neural Network based Tracking Controller

Tim Waegeman, Benjamin Schrauwen

Department of Electronics and Information Systems
Ghent University, Ghent Belgium
http://reslab.elis.ugent.be

Abstract. Learning an inverse kinematic model of a robot is a well studied subject. However, achieving this without information about the geometric characteristics of the robot is less investigated. In this work, a novel control approach is presented based on a recurrent neural network. Without any prior knowledge about the robot, this control strategy learns to control the iCub's robot arm online by solving the inverse kinematic problem in its control region. Because of its exploration strategy the robot starts to learn by generating and observing random motor behavior. The modulation and generalization capabilities of this approach are investigated as well.

Keywords: Adaptive control, Feedback control, Inverse kinematics, Neural network (NN), Reservoir computing (RC)

1 Introduction

Drawing a figure on a blackboard is a task which humans perform without consciously thinking about how each joint of their arm should be positioned. For robots like the iCub [1], this task is much more difficult. A robot needs to be able to map a position from task-space to joint-space, which is called inverse kinematics. There are multiple positions in joint-space to reach a given task-space target. On the other hand, mapping positions from joint-space to task-space, called forward kinematics, is a unique transformation. Solving the inverse kinematics problem has been investigated extensively. Some approaches use analytical and numerical methods to solve this problem [2, 3]. More advanced techniques learn inverse kinematics in different sub-regions of the task-space and use a weighting approach to approximate the inverse kinematic over the entire task-space [4, 5]. Other techniques use a Recurrent Neural Network (RNN), to train on forward kinematic data containing the positions in joint and task-space [6–8]. This removes the redundancy in the inverse transformation, because there are no redundant examples given during training. However, these techniques learn an attractor that is limited to the regions described by the training data and although they posses generalization capabilities, learning the entire inverse kinematic model can not be claimed. The proposed controller learns to control the

iCub's arm online by creating a model based on previous actions and observations of the robot. It uses a Reservoir Computing (RC) network to acquire an inverse kinematic model, but only in the visited regions of the arm. By doing both the control an the updating of the model simultaneously, the desired control behavior can be acquired without the need of having any prior knowledge about the robot.

The remainder of this paper is structured as follows: first, in Section 2, we give a short introduction on Reservoir Computing and explain the training algorithm. Next, in Section 3, the design steps of the controller are explained. To demonstrate the controllers performance, we apply it on a kinematic control task in Section 4. Finally, we draw our conclusions in Section 5.

2 Reservoir Computing

The RC network model used in this paper follows the Echo State Network (ESN) approach [9]. An ESN is composed of a discrete-time recurrent neural network (i.e., the reservoir) and a linear readout output layer which maps the reservoir states to the desired output. For many applications, the dynamics of the reservoir need to be tuned to match the intrinsic time scale of the input data. The system's dynamics can effectively be tuned by using leaky integrator neurons [9]. Their states and the readout output are updated as follows:

$$\mathbf{x}[k+1] = (1-\gamma)\mathbf{x}[k] + \gamma \tanh\left(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}[k] + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}}\right) \tag{1}$$

$$\mathbf{y}[k+1] = \mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{x}[k+1] + \mathbf{W}^{\mathrm{o}}_{\mathrm{b}}, \tag{2}$$

where $\mathbf{u}[k]$ denotes the input at time k, $\mathbf{x}[k]$ represents the reservoir state and $\mathbf{y}[k]$ is the output. The weight matrices $\mathbf{W}^{\Delta}_{*}$ represent the connections from $*$ to Δ between the nodes of the network (where r, i, o, b denote *reservoir*, *input*, *output*, and *bias*, respectively). All weight matrices $\mathbf{W}^{\mathrm{r}}_{*}$ to the reservoir are initialized randomly, while all connections to the output $\mathbf{W}^{\mathrm{o}}_{*}$ are trained using standard linear regression techniques. As non-linearity a hyperbolic tangent function is used. After initialization, $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$ is normalized by dividing it with its largest absolute eigenvalue (spectral radius). For linear neurons the spectral radius should typically be close, but smaller than one. Because of this spectral radius the system is operating at the edge of stability [9]. The leak rate γ in (1) controls the time scale of the network [9, 10]. Usually training $\mathbf{W}^{\mathrm{o}}_{*}$ is done offline, in batch mode. In this work, $\mathbf{W}^{\mathrm{o}}_{*}$ is trained online using Recursive Least Squares (RLS). With each iteration the output weights are adjusted so that the network converges to the desired output:

$$\mathbf{P}[k] = \mathbf{P}[k-1] - \frac{\mathbf{P}[k-1]\mathbf{x}[k]\mathbf{x}^T[k]\mathbf{P}[k-1]}{(1+\mathbf{x}^T[k]\mathbf{P}[k-1]\mathbf{x}[k])}, \tag{3}$$

with $\mathbf{P}[0] = \frac{\mathbf{I}}{\alpha}$, $\mathbf{x}[k]$ the current states and α constant. $\mathbf{P}[k]$ is a running estimate of the Moore-Penrose pseudo inverse $(\mathbf{x}^T\mathbf{x} + \rho\mathbf{I})^{-1}$. The used training error is

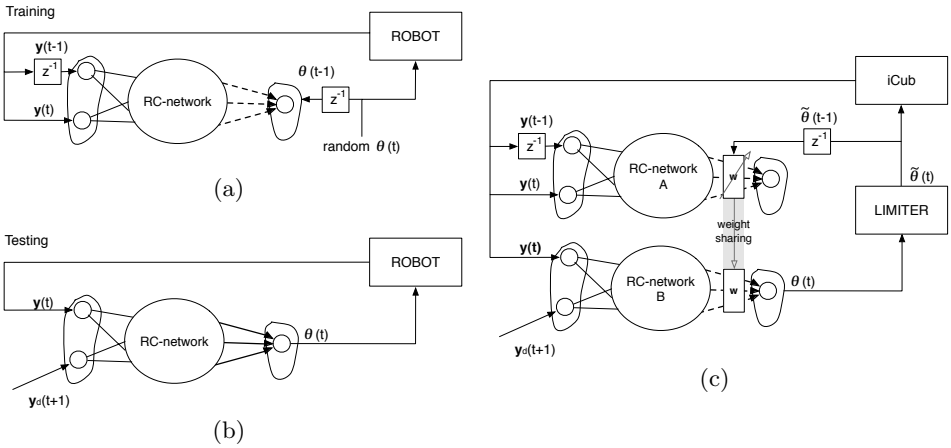


Fig. 1. (a) Illustration of a controller method described in the work of Jaeger [12]. During training, random $\boldsymbol{\theta}(t)$ values are used to train the output weights of the network based on the corresponding robot response $\mathbf{y}(t)$. (b) During testing, the trained network is used to control the robot according to to the desired trajectory $\mathbf{y}_d(t+1)$. (c) Schematic representation of the proposed controller. The dashed arrows represent the output weights $\mathbf{w}$ which are trained. These are the same for both networks (weight sharing). The optional limiter bounds $\boldsymbol{\theta}(t)$ to a certain range. Afterwards, the bounded values $\tilde{\boldsymbol{\theta}}(t)$ drive the robot. The values $\tilde{\boldsymbol{\theta}}(t-1)$ are used as desired network output for RC-network A which are used to train the weights $\mathbf{w}$.

defined as the difference between the generated and desired output $d[k]$:

$$e = \mathbf{w}[k-1]\mathbf{x}[k] - d[k] \tag{4}$$

$$\mathbf{w}[k] = \mathbf{w}[k-1] - e\mathbf{P}[k]\mathbf{x}[k]. \tag{5}$$

When using RLS these output weights are rapidly and effectively modified. This behavior satisfies the conditions necessary for the FORCE approach by Sussillo and Abbott [11]. This approach allows learning with feedback of the actual output (small errors included) instead of clamping the feedback to the correct output (no errors) during training.

3 Design of The Controller

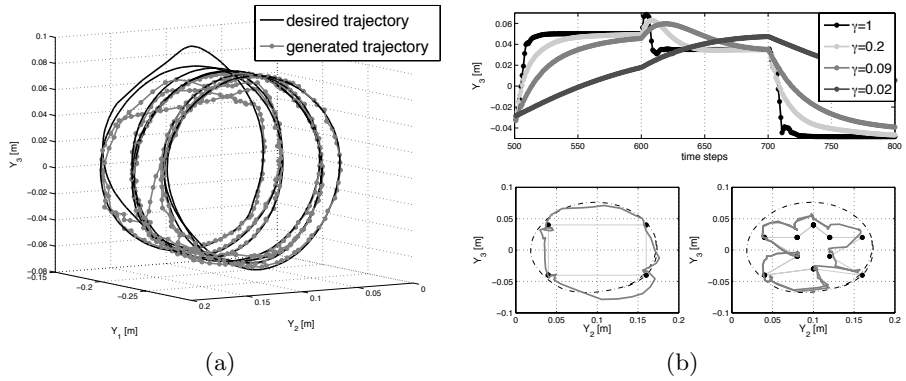
The goal of this work is to design a controller which can learn the inverse kinematics in the vicinity of the desired trajectory and without any prior knowledge about the robot system. For most supervised learning techniques, training examples are generated by a teacher controller, observations in joint and task-space or an actual inverse kinematic model of the robot. However, in this work no prior knowledge or model is assumed.

Another approach to learn a good kinematic representation is to use a model exploration strategy where random motor commands are generated and the corresponding robot response is observed. In the work by Jaeger [12], such a strategy

is taken. Here, an RC-network is used which is trained offline by using random values as training output and the plant response to these values as training input. In this example, the feedback information $\mathbf{y}$ is presented to the RC-network in 2 versions: the current feedback $\mathbf{y}(t)$ and a delayed version $\mathbf{y}(t-1)$. During training, also the desired output, which are the random motor commands $\boldsymbol{\theta}(t)$, are delayed one time step before given to the RC-network. As a result, the network learns to generate the previous output $\boldsymbol{\theta}(t-1)$, given the previous ($\mathbf{y}(t-1)$) and current input ($\mathbf{y}(t)$). After training the output weights (dashed lines in Fig. 1(a)), the desired end-effector position $\mathbf{y}_d(t+1)$ is presented to the input which was connected to $\mathbf{y}(t)$ during training. The actual end-effector position on the other hand, is given to the reservoir input which was connected with $\mathbf{y}(t-1)$ during training. The output of the network $\boldsymbol{\theta}(t)$ commands the robot.

In this work we want to achieve similar results but in an online manner. The advantage here is the ability of the controller to readjust the internal model of the robot to unforeseen changes in the robot or its environment during control. As shown in Fig. 1(c), a similar RC-network to the one described above (Fig. 1(a)) is used. This network, which we will call RC-network A, is trained online in a supervised manner by using RLS. Below RC-network A we have a duplicate network, RC-network B, with the same input, reservoir and output weights (weight-sharing) as RC-network A. This network is connected to the robot in a similar manner as described by Jaeger in Fig. 1(b). The output of this network is not only connected to the robot but is also used (delayed by one time step) as the desired output for training the output weights. The reservoir states are initially the same for both networks and are randomly chosen according to a normal distribution ($\mathcal{N}(0,1)$). This random initialization causes the robot to move its arm and generating examples which are spread more evenly over the solution space. Without this so called motor babbling the samples, used to model the kinematics, would be clustered which leads to poor generalization. Because the inputs are not the same for both networks, the corresponding states will evolve differently. However, as RC-network A is converging to a more accurate model, the inputs of both networks will become the same with a difference of 1 time step. Because of the desired trajectory in task-space and the current robot feedback as input, RC-network B starts generating values which are in turn used to command the robot. Such commands are limited to a certain range according to the actuators specifications. For instance, when controlling an actuator the amount of torque that it can deliver is limited. In Fig. 1(c) such limiting is represented by a limiter which bounds $\boldsymbol{\theta}(t)$ to $\tilde{\boldsymbol{\theta}}(t)$. Delayed by one time step, these values $\tilde{\boldsymbol{\theta}}(t-1)$ are given to RC-network A as the desired output. With each iteration, the resulting output weights are used for RC-network B.

By applying this topology, RC-network A is learning the controller solely on the generated examples during actual control. On the contrary, RC-network B uses the trained parameters to improve the control of the robot based on both the desired and actual robot response.



(a) (b)

Fig. 2. (a) This plot illustrates a generated trajectory after training (gray) together with the desired trajectory (black). The dots on the generated trajectory are the sample points. (b, top) Demonstrates the velocity modulation on target point reaching (100 time steps per target point) . (b, bottom) Generalization (dark gray) to different target points (black dots). The circular dashed line represents the projection of the spiral on the plain of the target points.

4 Robot Experiments

To evaluate the proposed control strategy we learn a kinematic model of the 7 degree-of-freedom (DOF) iCub robot arm. A "Webots" simulation model of the iCub robot is used to do the experiments. Here Y_1, Y_2 and Y_3 correspond to respectively the Z-, X- and Y-axes of the robot's frame of reference. Encoders in each joint measure the angular positions. When a joint position is given to the robot, an internal PID controller will generate the torque necessary to move the joint to the desired position. Therefore, due to the dynamics of the robot, a delay between the commanded and recorded position is observed.

4.1 RC-network Setup

RC-network A and B are, except for their input, identical. For choosing the number of neurons, a trade-off between execution speed and performance has to be made. In our experiments we used 400 neurons. All the following parameters are hand tuned. The connection matrix from input to the reservoir ($\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$) has elements which are drawn from a normal distribution ($\mathcal{N}(0, 1.5)$). The reservoir has a connection matrix with values that are drawn from $\mathcal{N}(0, 1)$. Other properties of the reservoir are a spectral radius and a leak rate of 1. In this work the leak rate will be used to modulate the velocity of the generated trajectory. The connection of the bias to the readout layer is trained. The connection matrix $\mathbf{W}_b^r$ from (1) is drawn from a standard normal distribution and scaled by a bias-term of 0.5. The robot model is commanded by angular positions in degree. The network will however, explore this range (within the boundaries set by the

limiter) to find the correct angles. The introduced RLS-parameter α, defined in Section 2, is set to 1. The initial output weights $\mathbf{w}(0)$ are normalized random values ($\mathcal{N}(0,1)$).

4.2 Learning Kinematic Model

The desired spiral trajectory in task-space is similar to the one described in [7]. By connecting the proposed controller to the iCub simulation model, the controller will initially drive the robot arm randomly. Thanks to the RLS learning rule, fast adaptation of the output weights is achieved. As a results, fast learning of the kinematic model is acquired. After only 1000 time steps (time step = 270 ms) the robot starts following this spiral trajectory. Most feedback controllers use an error defined on the task-space to achieve the desired behavior. Although the proposed controller is not designed to minimize this error in task-space, it converges to the desired trajectory because the internal trained model corresponds to that of the iCub's arm. When such a model is achieved we could choose to continue the training online, learning the inverse kinematics in newly visited task-space regions. However, to evaluate the trained model at a certain point in time we will stop the training by setting $\Delta\mathbf{w} = \mathbf{0}$ in (5) and evaluate how well it continuous to follow a desired trajectory without learning. In Fig. 2(a) we show such generated trajectory of the iCub's arm (gray), which needs to follow a spiral trajectory similar (not the same) to the desired trajectory during learning. As demonstrated, the learned inverse kinematic model corresponds well with the iCub's arm for the desired end effector positions. However, because of the physical limitations of the robot some desired trajectory points, especially the ones closer to the robot ($Y_1 < -0.2$), are unreachable by the robot.

4.3 Generalization

Next, we investigate the transient and generalization behavior of the learned kinematic model. Instead of following a spiral trajectory we define some target corner points that form a specific shape (e.g.: a square or a star). So 4 target points for a square and 10 for a star. These points are all located on a plane perpendicular to the direction of the spiral. Each target point excites the network for 100 time steps. Afterwards, the next target point is given to the network. In other words, it is not necessary to follow a square or star trajectory, but the task is to reach the target corner points of each shape. The target points forming a square are located on the projection of the learned spiral data (dashed lines). As shown in Fig. 2(b) (bottom, left) the desired target points are reached. However, the generated movement between two different target points is demonstrating transient behavior, that is, it does not follow a straight line (shortest path between target points) but rather according to an arc (dark gray). The acquired kinematic model was learned by following a spiral trajectory, never reaching other regions of the task-space, which explains the transient arc behavior of the generated motion. The target points forming a star shape are, except for two,

not located on the projection of the spiral trajectory. Although the learned kinematic model is based on the data seen while following the spiral trajectory, the model is generalizing well to the other target points. Fig. 2(b) (bottom, right) shows small deviations in the reached target points (black dots), but it illustrates that the learned model generalizes well.

4.4 Modulation

The velocity at which each trajectory point is reached can be modulated after or while training. We achieve this by changing the leak rate γ of the reservoir states (1). As described before, changing $\gamma \in [0, 1]$ effectively changes the time scale of the system. Fig. 2(b) (top) demonstrates the effect of such modulation for multiple γ's after learning ($\Delta\mathbf{w} = \mathbf{0}$) and for different target points (e.g.: the target points forming the star shape). By decreasing γ, the distance between each sample point will decrease as well. As shown in the top plot of Fig. 2(b), the robot is unable to reach the target positions within 100 time steps when using $\gamma = 0.02$.

5 Conclusion

We presented a novel controller based on a recurrent neural network, which is able to learn the inverse kinematics of the iCub's robot arm fast, without any prior knowledge about the robot. By using an internal exploration approach, the proposed system starts learning a model of the arm. Although there is no error defined on the actual trajectory, the robot will eventually generate the desired motion. The system only uses observations of forward kinematic motion generation to learn a kinematic model of the robot arm. Consequently, the controller only observes a unique mapping of joint to task-space positions. Bad examples due to redundancy in the inverse kinematics, are not observed and thus eliminated in the actual control. Both generalization experiments demonstrate that when online learning is discontinued, the learned inverse kinematic model is restricted to the previously visited regions. Although generalizing well to unseen task-space regions, the claim of learning a full kinematic model is only valid when all possible end effector positions are visited during the online learning. From our experiments transient behavior of the network emerged as arced motions between the different presented target points. Finally, we demonstrated the velocity modulation of the controller and its effect on the resulting trajectory. In future work the applicability of this control approach to more advanced control tasks will be investigated and compared to classical control approaches.

Acknowledgment

This work was partially funded by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) and the FP7 funded AMARSi EU project under grant agreement FP7-248311.

References

1. Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F.: The iCub humanoid robot: an open platform for research in embodied cognition. In: Proc. of the 8th Workshop on Performance Metrics for Intelligent Systems, pp. 50-56, ACM (2008)
2. Tolani, D., Goswami, A., Badler, N.: Real-time inverse kinematics techniques for anthropomorphic limbs. Graphical models, vol. 62, 353-388, Elsevier (2000)
3. Grochow, K., Martin, S., Hertzmann, A., Popović, Z.: Style-based inverse kinematics. Transactions on Graphics (TOG), vol. 23, 522-531, ACM (2004)
4. D'Souza, A., Vijayakumar, S., Schaal, S.: Learning inverse kinematics. In: Intelligent Robots and Systems, vol. 1, pp. 298-303, IEEE (2001)
5. Peters, J., Schaal, S.: Learning operational space control. Proceedings of Robotics: Science and Systems (RSS), MIT Press, Philadelphia (2006)
6. Guez, A., Ahmad, Z.: Solution to the inverse kinematics problem in robotics by neural networks. In: International Joint Conference on Neural Networks, pp. 617-624, IEEE (1988)
7. Reinhart, R., Steil, J.: Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot iCub. In: 9th IEEE-RAS International Conference on Humanoids, pp. 323-330, IEEE (2010)
8. Neumann, K., Rolf, M., Steil, J., Gienger, M.: Learning Inverse Kinematics for Pose-Constraint Bi-Manual Movements. From Animals to Animats 11, 478-488, Springer (2010)
9. Jaeger, H., Lukosevicius, M., Popovici, D., Siewert, U.: Optimization and applications of echo state networks with leaky-integrator neurons. Neural Networks, vol. 20, 335-352, Elsevier (2007)
10. Schrauwen, B., Verstraeten, D., Van Campenhout, J.: An overview of reservoir computing: theory, applications and implementations. In: Proc. of the 15th European Symposium on Artificial Neural Networks, pp. 471-482, D-side (2007)
11. Sussillo, D., Abbott, L.: Generating coherent patterns of activity from chaotic neural networks. Neuron, vol. 63, 544-557, Elsevier (2009)
12. Jaeger, H.: A method for supervised teaching of a recurrent artificial neural network. Patent Application, WO 2002/031764 A2 (2002)