Learning robot reaching motions by demonstration using nonlinear autoregressive models

Rafael F. Santos a,b,*, Guilherme A.S. Pereira a, L.A. Aguirre a

a Graduate Program in Electrical Engineering, Federal University of Minas Gerais, Av. Antônio Carlos 6627, 31270-901 Belo Horizonte, MG, Brazil
b Federal University of Itajubá - Advanced Campus of Itabira, Irmã Ivone Drumond 200 - Distrito Industrial II, 35903-087 Itabira, MG, Brazil

HIGHLIGHTS

• A new method for learning robot reaching motions from demonstrations.
• Dynamic system identified using Nonlinear Autoregressive (NAR) models.
• Good learning performance with low control efforts.
• Evaluated with mobile and manipulator robots.

ARTICLE INFO

Article history:
Received 8 November 2017
Received in revised form 24 May 2018
Accepted 12 June 2018
Available online 26 June 2018

Keywords:
Learning by demonstration
Nonlinear autoregressive models
Dynamical systems
Fixed point

ABSTRACT

This paper presents NAR-RM, a method for learning robot reaching motions from a set of demonstrations using Nonlinear AutoRegressive (NAR) polynomial models. Reaching motions are modeled as solutions to autonomous discrete-time nonlinear dynamical systems, so that the movements start near the data of the demonstrations follow the trained trajectories and always reach and stop at the target. Since NAR models obtained using standard system identification techniques do not always adequately model the reaching motions, in this paper we present a method that uses a least-squares estimator with constraints to impose the location of fixed points in the model. With the imposition of new fixed points it is possible to change the location of the original fixed points of the model, thus allowing the learning of stable reaching motions. We evaluate our method using a library of human handwriting motions, a mobile robot and an industrial manipulator.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

In the last decade, learning by demonstration has been used to teach a robot how to perform point-to-point movements, also known as reaching motions. Point-to-point movements provide basic components, called movement primitives, which can be used to compose more complex movements [1]. These movements may be learned through a set of demonstrations, at the kinematic level, where the most relevant characteristics are extracted and codified. Movements encoded using nonlinear dynamical systems (DS) have shown good results [2–4].

A widely known method for learning reaching motion with DS is called Dynamic Movement Primitives (DMP) [3]. The DMP estimates a nonlinear DS while ensuring stability to the target. Stability is obtained by suppressing the nonlinear terms at the end of the motion, when a smooth switch to a stable linear system is controlled by a phase variable. As the learned DS is non-autonomous, the use of a heuristic to reinitialize the phase variable is always necessary in case of temporal perturbations, which may occur due to delays during execution of the model.

Autonomous DS are good alternatives for modeling movements robust to temporal perturbations. Binary Merging (BM) [5], for example, is a method that uses a Gaussian mixture model to estimate an autonomous DS with local asymptotic stability to the target. The stability assurance is obtained during the learning phase, where constraints, some based on Lyapunov functions, are imposed. The stability check is done numerically and the application of the method is limited to regions close to the demonstrations.

Another well-known method that generate autonomous DS models is the Stable Estimator of Dynamical Systems (SEDS) [4]. SEDS uses a Gaussian mixture model to estimate autonomous DS with global asymptotic stability to the target. Although SEDS is able to learn several types of movements, the set of movements is limited to a contractive dynamics corresponding to a quadratic
Lyapunov function. The same authors of SEDS proposed another method, called Control Lyapunov function-based Dynamic Movements (CLF-DM) [6], to estimate autonomous DS that guarantee global asymptotic stability to the target in the execution phase. The method uses Lyapunov functions, learned using the data from the demonstrations, to stabilize nonlinear DS in the execution phase. The Lyapunov function learned from the data is less restrictive than the one used in SEDS, allowing learning a larger set of movements. However, the method uses a on-line correction signal that potentially interferes with the DS, requiring a very careful selection of some parameters [7].

Methods using neural networks were also proposed. The Neu- rally Imprinted stable Vector Fields (NIVF) [8], uses a Lyapunov candidate function to shape and guarantee the local asymptotic stability of the autonomous DS during the learning phase. The method allows to define the stability region a priori, but depends on a costly process that is executed after the learning phase to attest the stability guarantee [7]. Initially, the NIVF used only a quadratic Lyapunov candidate function (NIVF-QLG), allowing the method to learn the same set of movements as SEDS. Subsequently, a method to learn Lyapunov candidate functions from the demonstration data, called Neuraly Imprinted Lyapunov candidate (NILC) [9], was presented and incorporated into the NIVF (NIVF-LL), which allowed to expand the set of movements learned.

A method that is capable of learning autonomous DS, guar- anteed global asymptotic stability to the target, in the learning phase, and that had the capacity to learn a larger set of movements is proposed in [7]. The method called r-SEDS extends the SEDS application domain by means of a diffeomorphic transformations r. The main idea of the method is to transform to a quadratic form, using r, a Lyapunov candidate function L, learned using the data, and the demonstrations data D, generating respectively L and D. Then, SEDS is applied to estimate a dynamic system y using D, which is stable according to L. Finally, the inverse transformation of the diffeomorphism is applied to obtain the stable DS y.

System identification techniques using autoregressive models were applied before to teach wheeled mobile robots by demonstration. Autoregressive moving average model with exogenous inputs (ARMAX) and nonlinear ARMAX (NARMAX) were used in [10] and [11] to model sensor–motor mappings, where the sensory information of the demonstrations are used to obtain a model that is used directly to control the robot. The learned models, which requires a considerable amount of data and demonstrations to be identified, were used in reactive tasks such as wall following, corridor following, door traversal and route learning. Following papers explore different training techniques, stability analyzes and the sensors that must compose the model in [12–14]. Notice that the objective of these works is different from what was proposed by the papers surveyed before, such as [3,4] and [9], which model robot trajectories instead of sensor–motor mappings.

This paper presents NAR-RM, a novel approach to identify, using a small set of demonstrations, autonomous dynamic systems (DS) that models reaching motions using nonlinear autoregressive (NAR) polynomial models [15]. Because previous work, such as [10] and [14], were not used to model robot trajectories, by the authors knowledge, this is the first work in the literature that uses polynomial NAR models to program reaching motions by demonstrations. Although the application of such a tool seems to be straightforward for those that are familiar with system identification, we show that standard techniques do not always model robot reaching motions adequately. Therefore, this paper also contributes with a method, based on a constrained least-squares estimator, to impose the location of the model’s fixed points, thus guaranteeing some desirable properties. A Matlab implementation of our contribution is freely available online [16]. Fig. 1 shows the phase map of the typical 2D model obtained with the method proposed in this paper. We use a small set of demonstrations, shown as the red trajectories, to generate a model with a stable fixed point at the target and a large basin of attraction, shown in green, that covers all the demonstrations. In this paper, we numerically compare our method with state-of-the-art methods and show that the proposed methodology may be considered the best alternative to generates precise models that have, at the same time, a low effort for the low level controller.

The rest of this paper is organized as follows. In Section 2 we present the problem statement. A review of the mathematical representation used in the paper is presented in Section 3. Section 4 describes the method to model reaching motions using NAR, which may be implemented by the algorithms proposed in Section 5. In Section 6, we present the experimental validation of the method using a library of human handwriting motions and experiments with mobile and manipulator robots. Discussions are presented in Section 7, and finally, we summarize the work in Section 8.

2. Problem statement

The problem we are dealing with in this paper is to learn robot-reaching motions from a set of demonstrations. These demonstrations are point-to-point trajectories in the robot workspace. They are initially executed by the robot when it is controlled by a human operator, which may operate the robot using a force/motion sensor or a joystick. After training, the robot will be able to reproduce the trajectory without human interference. In the ideal situation, the motion learned by the robot must be robust to temporal and spatial perturbations. Temporal perturbations may be caused by communication delays and the dynamics of low level controllers. Spatial perturbations may occur due to sensing and actuation errors, which can introduce small deviations in the robot trajectories during its execution. Spatially, it is also desirable that the correct motion is executed by the robot even if it starts from different positions and the target position changes.

Following successful ideas in the literature, the aim here is to model the robot as a nonlinear dynamical system (DS). Solutions to the estimated DS model must be close to the data collected during demonstrations, and should always reach and end at the target. Hence the main problem requirements are:

- The DS model must be autonomous. This ensures that the model is robust to temporal perturbations;
- The target must be a fixed point attractor and the target basin of attraction should cover all demonstration data points. In this way, any motion initiated near the demonstration region will end at the target;
• The distance between the demonstration data points and the boundary of the target basin of attraction should be the largest possible to increase the spatial robustness.

In this work, nonlinear autoregressive polynomial models (see Section 3) are used to represent the DS and are estimated using system identification techniques.

3. Background

A multivariable nonlinear autoregressive model can be written as [17]

\[ y(k) = F[y(k-1), \ldots, y(k-n_y)] + e(k), \]

where

\[ y(k) = [y_1(k), y_2(k) \ldots y_p(k)]^T, \]

where \( y_j(k), i = 1, 2, \ldots, p \) are output signals and \( e(k) \in \mathbb{R}^p \) accounts for uncertainties and possible noise at time \( k \). Also, \( n_y \) is the maximum lag, which determines the model order. In this paper, \( F[\cdot] \) is assumed to be polynomial with nonlinearity degree up to \( \ell \). Other alternatives for \( F \) are possible [18]. Model (1) is a special case of a nonlinear autoregressive moving average model with exogenous inputs (NARMAX) model [19], which has been shown to be adequate to represent many nonlinear dynamical systems [15].

Hence, (1) represents a set of \( p \) nonlinear autoregressive polynomial models, which do not need to be of the same order nor of the same degree, that is, \( n_y \) and \( \ell \) can be different for each equation, but to keep nomenclature simple they will be used in general.

For any equation in (1) the outputs of all other equations can be seen as inputs. Therefore (1) can be dealt with as \( p \) multi-input single-output (MISO) problems. In what follows the formulae relates to the MISO case.

Having defined which representation to use, the identification of a model consists roughly of two steps, namely structure selection and parameter estimation.

Assuming a dataset of size \( N \), the \( i \) th equation in (1) can be expressed in matrix form as

\[ \mathbf{y}_i = \psi_i \hat{\theta}_i + \xi_i, \]

where \( \psi_i \in \mathbb{R}^{N \times n_y} \) is the matrix of regressors, \( \hat{\theta}_i \in \mathbb{R}^{n_{\theta}} \) is the vector of estimated model parameters, \( \mathbf{y}_i = [y_i(k)]_{k=1}^N \) is the vector of model outputs and \( \xi_i = [\xi_i(k)]_{k=1}^N \) is the vector of residuals, which in some cases can be interpreted as an estimate of the unknown noise \( e \).

The regressors of model (1) may contain any combination of lagged outputs up to degree \( \ell \). The number of such combinations is determined by the values of \( \ell \) and \( n_y \) and can easily include hundreds of candidate regressors. This huge amount of terms is a major impediment to the usefulness of the estimated model and some kind of mechanism is called for in order to automatically choose the best \( n_y \) regressors to compose the \( i \)th equation of the model. This problem is often referred to as model structure selection and must be judiciously accomplished regardless of the mathematical representation being used.

Having decided which regressors to use (more on this later), the \( n_{\theta} \)-dimensional parameter vector \( \hat{\theta}_i \) can be estimated by minimizing the following cost function \( J_i(\hat{\theta}_i) = \xi_i^T \xi_i \) using orthogonal techniques [20]. The orthogonal algorithm described in [17] effectively overcomes two major difficulties in nonlinear model identification, namely (i) numerical ill-conditioning and (ii) structure selection. This amounts to selecting the columns of the regressors matrix \( \psi_i \). A criterion for structure selection that has proved helpful in many situations involving both real and simulated data is the error reduction ratio (ERR) [17]. This criterion is a welcome by-product of the orthogonal least squares procedure outlined above. Other, sometimes complementary, procedures have also been proposed in the literature [21–23]. See [18,24,25] for a comparison of some structure selection methods. The ERR criterion is briefly reviewed in the next section.

3.1. The ERR criterion

A widely used criterion in the structure selection is the error reduction ratio (ERR) [17]. Some of the interesting features of this criterion are: (i) it is computationally inexpensive because it is based on the minimization of one-step-ahead prediction errors. As a matter of fact, it can be obtained as a by-product of the orthogonal least-squares, (ii) it has a simple interpretation: the ERR evaluates the importance of the model terms according to their ability to explain the data variance one step ahead, and (iii) it has been found effective in a number of practical situations (see references).

The reduction in the variance of the residuals, that occurs as new terms are included in the model, can be normalized in relation to the output variance. Then, the error reduction ratio due to the inclusion of the \( i \)th regressor in the model can be written as [26]:

\[ \text{ERR}_i = \frac{\text{MS1PE}(\hat{\mathbf{M}}_{i-1}) - \text{MS1PE}(\hat{\mathbf{M}}_i)}{\text{MS1PE}(\hat{\mathbf{M}}_i)}, \quad i = 1, \ldots, n, \]

where \( \langle \cdot, \cdot \rangle \) indicates the internal product. MS1PE(\( \hat{\mathbf{M}}_i \)) stands for the mean square one-step-ahead prediction error of the model with \( i \) terms (regressors); \( n \) is the number of candidate terms tested for; and \( \mathbf{M}_i \) represents a family of models with nested structures, thus \( \mathbf{M}_{i-1} \subset \mathbf{M}_i \). In (3) the numerator equals the reduction in variance of the residuals due to the inclusion of the ith regressor. The denominator of (3) is proportional to the (zero-mean) data variance. One of the advantages of the ERR is that, after orthogonalization, it can be represented in compact form as [17]:

\[ \text{ERR}_i = \frac{\hat{\mathbf{y}}_i^T \{\mathbf{w}_i, \mathbf{w}_i\}}{\langle \hat{\mathbf{y}}_i, \hat{\mathbf{y}}_i \rangle}, \quad i = 1, 2, \ldots, n, \]

where \( \mathbf{w}_i \) is the ith regressor in the orthogonal representation and \( \hat{\mathbf{y}}_i \) is the corresponding estimated parameter. Thus, at each step, the term with the largest ERR is added to the model.

3.2. Parameter estimation

One way of determining \( \hat{\theta}_i \) in \( \mathbf{y}_i = \psi_i \hat{\theta}_i + \xi_i \) is by the Least-Squares (LS) estimator, that minimizes \( \langle \xi_i, \xi_i \rangle \) and is given by:

\[ \hat{\theta}_i = (\psi_i^T \psi_i)^{-1} \psi_i^T \mathbf{y}_i. \]

The inversion of matrix \( \psi_i^T \psi_i \) is guaranteed when the data are persistently exciting, when the model structure is not badly over-parametrized and when the data are not grossly oversampled [15].

A possible use of constraints in the LS estimator (CLS) was presented in [27] and allows to estimate models using a closed form. Consider a set of \( n_c \) constraints on the parameter vector written as \( \mathbf{c} = \mathbf{S} \theta, \) where \( \mathbf{c} \in \mathbb{R}^{n_c} \) is a known constant vector, and \( \mathbf{S} \in \mathbb{R}^{n_c \times n_{\theta}} \) is a known constant matrix. The solution to the problem

\[ \hat{\theta}_{\text{CLS}} = \arg \min_{\theta} \langle \xi_i, \xi_i \rangle, \]

is given by

\[ \hat{\theta}_{\text{CLS}} = \hat{\theta}_i - A(SA)^{-1}(S\hat{\theta}_i - \mathbf{c}), \]

where \( A = (\psi_i^T \psi_i)^{-1}S^T \). For \( SA \) to be invertible, the constraints (encoded in \( S \)) should not induce singularities. This would happen, for instance, if two equations of the model (see Eq. (1)) were identical, which is not a likely situation. A more plausible situation which would pose numerical problems is if the number of constraints...
becomes large compared to the number of parameters. This would hinder fitting dynamical data due to the lack of sufficient free parameters.

Next section describes the methodology proposed in this paper to model reaching motions using multivariate NAR models and the techniques presented in this section.

4. Methodology

In this paper we look for first order models, where the motion of the system is determined exclusively by its current state. To identify these models, the data used are a set of \( n \) demonstrations \( Y(n) \), which are interpreted as trajectories in a state space \( \mathbb{R}^p \). Further, it is assumed that \( Y(n) \) are integral curves of an underlying vector field. The aim is to find NAR models that will approximate the vector field. In this case, \( \mathbf{y}(k) = [y_1(k) \ldots y_p(k)]^T \) (see Eq. (1)), where \( y_1(k), \ldots, y_p(k) \) are the coordinates of the trajectory. Hence we search for models composed of \( p \) first-order (\( n_{y_1} = \cdots = n_{y_p} = 1 \)) difference equations, to produce \( \hat{y}_1(k), \ldots, \hat{y}_p(k) \).

The pool of candidate regressors is composed by all possible combinations up to degree 1 of such variables plus a constant term. Assuming, without loss of generality, that the target is the origin of the state space, in order to ensure that the target is a fixed point of the model it is sufficient to remove the constant term of the pool of candidate regressors [28]. Thus, all the model regressors will depend on \( y_1(k), \ldots, y_p(k) \). The regressors of each model equation are automatically chosen from the pool of candidates independently, using the ERR criterion described in Section 3.1, so that the terms in each equation can be different. This is done in two steps, first the regressors are ranked in descending order of relevance in terms of the ERR criterion [17]; second, the number of terms of each model, \( \dim(\theta_1) = n_{y_1}, \ldots, \dim(\theta_p) = n_{y_p} \), is chosen using Akaike’s criterion [29]. Hence a typical model has the general form:

\[
y_1(k) = F_1[y_1(k-1), \ldots, y_p(k-1)] + e_1(k)
\]

\[
y_2(k) = F_2[y_1(k-1), \ldots, y_p(k-1)] + e_2(k)
\]

\[
\vdots
\]

\[
y_p(k) = F_p[y_1(k-1), \ldots, y_p(k-1)] + e_p(k).
\]

(8)

Steady-state analysis of model (8) will play an important role in this work. Assuming the model is stable, in steady-state \( y_1(k) = y_1(k-1) = \hat{y}_1, \ldots, y_p(k) = y_p(k-1) = \hat{y}_p \). Hence, dropping the noise, the equilibria of model (8) are given by \([\hat{y}_1, \ldots, \hat{y}_p] \) that are the solutions to the set of equations:

\[
\hat{y}_1 = F_1[\hat{y}_1, \ldots, \hat{y}_p]
\]

\[
\hat{y}_2 = F_2[\hat{y}_1, \ldots, \hat{y}_p]
\]

\[
\vdots
\]

\[
\hat{y}_p = F_p[\hat{y}_1, \ldots, \hat{y}_p].
\]

(9)

and the stability of such equilibria can be determined by the eigenvalues \( \lambda_1, \ldots, \lambda_p \) of the Jacobian matrix

\[
DF(y) = \begin{bmatrix}
\frac{\partial F_1}{\partial y_1(k-1)} & \cdots & \frac{\partial F_1}{\partial y_p(k-1)} \\
\frac{\partial F_2}{\partial y_1(k-1)} & \cdots & \frac{\partial F_2}{\partial y_p(k-1)} \\
\vdots & \ddots & \vdots \\
\frac{\partial F_p}{\partial y_1(k-1)} & \cdots & \frac{\partial F_p}{\partial y_p(k-1)}
\end{bmatrix}
\]

(10)
evaluated at such equilibrium. According to Hartman–Grobow theorem for maps [30], the hyperbolic fixed point can be classified as: an attractor if \(|\lambda_i| < 1, \forall i = 1, 2, \ldots, p\); a repellor if \(|\lambda_i| > 1, \forall i = 1, 2, \ldots, p\); and a saddle if there are eigenvalues inside and outside the unit circle.

If a fixed point \( \mathbf{y}^* \) is an attractor of the system, then it has a basin of attraction. The basin of attraction of \( \mathbf{y}^* \) is the set of all initial conditions such that, when the system starts from one of these initial conditions, the system settles to \( \mathbf{y}^* \).

4.1. Fixed point allocation

An important contribution of this work is the imposition of fixed points during parameter estimation to attain some desirable model properties. Let \( \mathbf{y}^* = [y_1^*, y_2^*, \ldots, y_p^*] \) be the coordinates of a fixed point. In order to impose that the model in (8) should have such an equilibrium, a constraint of the form \( c = S \theta \) with

\[
c = \begin{bmatrix}
y_1^* \\
y_2^* \\
\vdots \\
y_p^*
\end{bmatrix}, \quad S = \begin{bmatrix}
F_1[y_1^*, y_2^*, \ldots, y_p^*] \\
F_2[y_1^*, y_2^*, \ldots, y_p^*] \\
\vdots \\
F_p[y_1^*, y_2^*, \ldots, y_p^*]
\end{bmatrix}
\]

(11)
should be satisfied. This can be achieved using the estimator (7). It must be remembered that to impose the trivial fixed point \( \mathbf{y}^* = \mathbf{0} \) to the model it suffices to remove the constant term from the pool of candidate regressors.

By using standard system identification techniques, it is possible to obtain models in the form of Eq. (8) that are able to produce integral curves in the demonstration region that resemble the demonstrated trajectories. These models will also have a fixed point at the target position, but it is not guaranteed that this point is always an attractor. This happens because the LS estimator does not impose any constraint as to the stability of a fixed point. Fig. 2(a) shows examples of fixed points of a two-dimensional model \( M_1 \), where point A is an attractor and point 51 and 52 are saddles.

One of the contributions of this work is to investigate how to modify the location of some fixed points of a model in order to obtain or improve on some desired characteristics with a view to learning reaching motions. On this line, we call attention to the following fact: Consider an unconstrained model \( M_1 \) with a fixed point at \( \mathbf{y}^* \) that has been estimated from the data \( Y(n) \). The parameters can be reestimated from the same data using (7) with (11) in order to impose a new fixed point at \( \mathbf{y}^* \). The constrained model is referred to as \( M_2 \). Now the following conjecture is put forward:

**Conjecture 1 (Fixed Point Stability Type Retained Under Small Perturbations).** The fixed point \( \mathbf{y}^* \) of \( M_2 \) will be of the same type as that of \( \mathbf{y}^*_1 \) of \( M_1 \) for sufficiently small \( \| \mathbf{y}^*_1 - \mathbf{y}^* \| \).

A justification of Conjecture 1 is as follows. The estimator in (7) is in the form

\[
\hat{\theta}_{LS} = \hat{\theta}_{LS} - A[S,A]^{-1}e,
\]

where \( e = \mathbf{S} - \mathbf{c} \) is the “error” by which the unconstrained estimated parameter vector misses the constraint. If \( \hat{\theta}_{LS} \) satisfies the constraint, then \( e = \mathbf{0} \) and both the constrained and unconstrained estimated vectors would be the same. Hence \( \lim_{\epsilon \to 0} \hat{\theta}_{CLE} = \hat{\theta}_{LS} \). In words, provided \( \epsilon \) is not too large, \( M_2 \) and its fixed points can be interpreted as a perturbed version of \( M_1 \) and its fixed points. It should be noticed that because the constraints implemented using (11) refer to location of fixed points, \( e \) is directly related to \( \| \mathbf{y}^*_1 - \mathbf{y}^* \| \). Then, for sufficiently small \( \| \mathbf{y}^*_1 - \mathbf{y}^* \| \) the types of fixed points of \( M_1 \) and \( M_2 \) will be the same, that is, saddles will remain saddles and the attractor will remain attractor. This has indeed been our experience in a vast number of cases, and is illustrated schematically in Fig. 2, where the yellow points 51 and 52 represent saddles points and the green point A is a attractor point. Statistical results of the application of Conjecture 1 in practice are presented in Section 6.

The basin of attraction of the target (attractor A) is shown in Fig. 3(a) in green and does not include some demonstration points. The basin of attraction often has as a border the stable manifold of a saddle (fixed point). This suggest a very simple procedure to increase the “size of the basin: “push away” the saddle. In fact, what is done is to impose a fixed point outside the basin (to the left in Fig. 3(a)) as described before. If the new fixed point is
Fig. 2. Two-dimensional model identified using LS estimator (a), and CLS estimator (b). In $M_2$, the location of $S_1$ was imposed by means of constrained estimation, that is $S_1 = \hat{y}^* = [y_1^*, y_2^*]^T$ is the new position of $S_1$, and the origin remains a fixed point — because the constant term was removed from the pool of candidates. The stability type of $S_1$ and $A$ did not change, see Conjecture 1.

Fig. 3. Possible two-dimensional model with the basin of attraction estimated in green color. The saddle $S_2$ in (b) was imposed to the left of its counterpart in (a) in order to increased the basin of attraction and to include all demonstration data. $M_3$ is a reaching motions model. (For interpretation of the reference to color in this figure legend, the reader is referred to the web version of this article.)

not too far, based on Conjecture 1, it is expected that the new fixed point will remain a saddle and as a consequence the basin is automatically increased in that direction, as illustrated in Fig. 3(b).

In this paper, Reaching Motions (RM) models will refer to models with an attractor at the target which has a basin of attraction that covers all demonstration points.

The choice of the new location for the saddle $S_2$ is somewhat subjective. On the one hand one would like it to be quite far (to the left in the case of Fig. 3(b)) such that small actuation and sensing errors during the execution of the model will still result in trajectories within the basin of attraction and, consequently, the robot will be driven to the target. On the other hand, if $S_2$ is placed too far, then there is the chance that it will no longer be a saddle and that the basin of attraction will be altogether different. A possible practical procedure to further extend the basin of attraction is to impose the fixed point $S_2$ at several locations and chose one based on the resulting basin. A simple suggestion is to take $m$ points evenly located around $S_2$ at a distance $d$, as illustrated in Fig. 3(c).

In synthesis, the methodology proposed in this section is, based on a set of demonstrations, to select a model structure and identify its parameters, using a standard LS estimator and, using a constrained estimator, slightly changes this model to transform it in Reaching Motions (RM) model, i.e. a model with an attractor at the target and a basin of attraction that covers all demonstration points. Next section will propose a set of algorithms that implement such a methodology.

5. Numerical implementation

Algorithm 1 is a generic algorithm that can be used (i) to make the target an attractor, (ii) to increase the basin of attraction, and (iii) to optimize other characteristics of the model. It returns a constrained NAR model, receives as input the demonstrations $Y_{[n]}$, the nonlinearity degree $\ell$, the maximum number of iterations $\text{MaxIt}$, the distance $d$, the number of points to try $m$ and the unconstrained model. Hence the algorithm performs changes in the unconstrained model until a specific characteristic (e.g. target attractor and size of the basin of attraction) is obtained or optimized.

Since Algorithm 1 has general purpose, some functions must be defined according to the application. The most important function is $\text{bestModel}$ (line 11), because it defines how the models will be evaluated, being responsible for guiding the whole searching process of the algorithm. It can be, for example, a function that computes the accuracy in the execution of a learned trajectory, as detailed in Section 5.1. The $\text{hasCharacteristic}$ function (line 5) is used to check if the model has an expected characteristic, for example the attractor target. Finally, the $\text{acceptable}$ function (line 9) is used to verify if the model has the minimum characteristics to continue evolving. If the model is not acceptable, $d$ is modified and searching continues.

The next subsections shows how Algorithm 1 can be used to change specific characteristics of the model.

5.1. Obtaining an attractor at the target

In our experiments the models usually had an attractor at the target position or, in rare cases, the target was not an attractor but had one or more attractors close the target position. These two situations happens because the demonstrations used in the estimation of models always describe trajectories that end around
the same target, defining one or more attractive regions close to the target. In the situation where the target is not an attractor, by applying the constraints based on Conjecture 1, it is possible to reestimate the parameters and obtain a model with a fixed point attractor at the target, as illustrated schematically in Fig. 2.

Based on Algorithm 1, it is then possible to implement a function called getTargetAttractor, which is called to check if a model has an attractor at the target and, if not, to obtain another model with this characteristic. The idea is to obtain m models with a fixed point imposed at distance d around the target using the imposeFP function and check if the best model has an attractor at the target. If no model has an attractor at the target, distance d is increased.

To evaluate and choose the best model we use the bestModel function. This function analyzes all models by checking if any of them has a fixed point attractor at the target, if none of the models has this feature the function returns any model. If there is one or more models that have an attractor on the target, a metric that evaluates the accuracy in the reproduction of the motions learned is used to choose the best model. A possible metric is called Swept Error Area (SEA) [6], which quantifies the model accuracy in two dimensions. This metric compares trajectories obtained by the model with the set of demonstrations $Y_{(m)}$ and is computed by

$$
\varepsilon = \frac{1}{n} \sum_{m=1}^{n} \sum_{t=1}^{n} A (\hat{Y}_{(m,t)}, \tilde{Y}_{(m,t+1)}, \hat{Y}_{(m,t)}, Y_{(m,t+1)}) ,
$$

where $A(p^1, p^2, p^3, p^4)$ corresponds to the quadrilateral area generated by the four points $p^1$ to $p^4$, $T^m$ is the number of points in the $m$ th demonstration and $Y_{(m,t)}$ is the point $t$ of trajectory $Y_{(m)}$ that starts in the same starting point of the demonstration $Y_{(m)}$ and is generated by the model.

At first, it cannot be guaranteed that all trajectories leaving the initial points will reach the target, since some of these points may not belong to the basin of attraction of the target. Therefore, one may use a partial calculation of SEA to choose the model, as follows:

1. determine, for each model, the number of trajectories that reach the target leaving the initial points of the demonstrations;
2. chose the models that have more trajectories that reach the target and calculate the SEA for these trajectories;
3. chose the model that presented the lowest SEA.

Other needed functions are acceptable and hasCharacteristic, which in this case are similar, verifying if the model has a fixed point attractor at the target.

### 5.2. Analyzing the target basin of attraction

As stated in Section 2, we are looking for models where the target basin of attraction should cover all demonstration data points. To compute such a basin, the region $R$ where the basin will be evaluated is defined. $R$ should represent the expected domain of the system and contain all demonstration data points. Since the number of points of $R$ is infinite, we choose to estimate the basin of attraction in a discrete way. Given a continuous region $R$, we define a discrete map $M$ of $R$, which is composed by a set of regular cells forming a grid, and apply the algorithm proposed in Appendix to estimate the basin of attraction.

To increase the target basin of attraction and turn the current model into a RM model, two functions based on Algorithm 1, namely getRM_Model1 and getRM_Model2, may be implemented. getRM_Model1 function uses partial SEA in the bestModel function and implement the hasCharacteristic function for checking if all trajectories that start at the initial points will reach the target. However, because the model does not reproduce the trajectories perfectly, not always the use of partial SEA will be sufficient to guarantee that the demonstrations will be in the target basin of attraction. Thus, function getRM_Model2, where function bestModel chooses models that have the smallest number of cells on $M$ that contain points of the demonstrations that do not belong the target’s basin of attraction, will also be necessary. In this function, the hasCharacteristic function check if the model is a RM model. Both, getRM_Model1 and getRM_Model2 use acceptable functions that accept only models that have the target attractor. If no acceptable models are found, $d$ is decreased and all fixed points apart from the target will be imposed at slightly different locations in the attempt to improve the model.

### 5.3. Improving a reaching motions model

In order to quantitatively evaluate RM models, in 2D, we use SEA to measure the reproduction accuracy. We also propose a metric, called $\alpha$-distance, to evaluate the robustness of the basin of attraction in relation to the spatial perturbations. This metric is defined below:

**Definition 1.** The $\alpha$-distance is the shortest Euclidean distance between the demonstration trajectories and the boundary of the target basin of attraction.
5.4. Complete procedure

The $\alpha$-distance indicates the largest deviation that can occur in the trajectory before it leaves the basin of attraction. Hence the larger the $\alpha$-distance, the greater the robustness with respect to spatial perturbations. Globally stable methods would have an infinite $\alpha$-distance.

In this work the $\alpha$-distance is approximated by the smallest distance between the center points of the cells at the border of the basin of attraction and the trajectories of the demonstrations.

To optimize $\text{SEA}$ and $\alpha$-distance values, two other functions based on Algorithm 1 may be implemented. Functions optimizeAlphaDistance and optimizeAccuracy do not use any hasCharacteristic function (since the model already has all desirable characteristics), they allow for all points except the target to change location, the acceptable function only allows RM models and if no acceptable model is found, distance $d$ is decreased. The optimizeAlphaDistance function uses the $\alpha$-distance in the bestModel function with the constraint that the chosen model cannot have the $\text{SEA}$ value higher than the $\text{SEAMax}$. On the other hand, the optimizeAccuracy function uses the $\text{SEA}$ in the best-Model function with the constraint that the chosen models cannot have the $\alpha$-distance value less than the $\text{AlphaMin}$.

5.4. Complete procedure

Once each function is defined, a complete procedure to obtain a Reaching Motions model (RM) is presented in Algorithm 2, where the functions are called in sequence. The set of input parameters $I$ is the same of Algorithm 1, i.e. $I = \{Y_{(n)}, \ell, \text{MaxIt}, r, d, \text{Model}\}$. In this paper, we name the complete procedure by NAR-RM.

Next section shows experiments with a 2D library of motions that numerically illustrate and validate the proposed NAR-RM algorithm. We also tested NAR-RM with two actual robots, a planar mobile base and an industrial manipulator that operates in 3D.

6. Experiments

6.1. LASA handwriting dataset

The proposed methodology was implemented using Matlab. Our code is freely available at [16]. We initially tested the method with the LASA Handwriting Dataset library,1 also used in [46–9,31]. The library consists of 30 calligraphy movement patterns in 2D. Each movement pattern consists of 7 demonstrations with 1000 points each. Each demonstration has different starting points and the same end point at $x = 0$ and $y = 0$.

1 Available in: https://bitbucket.org/khansari/lasahandwritingdataset.

Table 1 Percentage of fixed points that do not change their type when new points are imposed using distance $d$.

<table>
<thead>
<tr>
<th>$d$ (mm)</th>
<th>0.01</th>
<th>0.05</th>
<th>0.1</th>
<th>0.2</th>
<th>0.5</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>$N^o$ of points imposed with $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unchanged type (%)</td>
<td>$\ell = 3$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>98</td>
<td>98</td>
<td>97</td>
<td>93</td>
<td>86</td>
<td>80</td>
<td>75</td>
</tr>
<tr>
<td>Unchanged type (%)</td>
<td>$\ell = 4$</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>98</td>
<td>95</td>
<td>93</td>
<td>86</td>
<td>78</td>
<td>70</td>
</tr>
<tr>
<td>Unchanged type (%)</td>
<td>$\ell = 5$</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>97</td>
<td>92</td>
<td>78</td>
<td>65</td>
<td>57</td>
<td>52</td>
</tr>
</tbody>
</table>

Table 2 Results obtained using an unconstrained LS estimator.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>Total</th>
<th>$\ell = 3$</th>
<th>$\ell = 4$</th>
<th>$\ell = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of models</td>
<td>90</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Target is not attractor</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>RM models</td>
<td>41</td>
<td>18</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>

In our experiments, the degree of nonlinearity $\ell$ was varied from 3 to 5 in the construction of NAR models. The 7 trajectories of the demonstrations were decimated at a rate of 20, resulting in 50 points each. Point $(0, 0)$ has been added to the end of all decimated demonstrations resulting in 51 points. The equilibria are found by solving numerically the set of nonlinear equations in (9) using the Matlab function solve.

To verify numerically Conjecture 1, the percentage of fixed points that retained their original type when other points were imposed were computed (Table 1). This was done for 11 different distances ($d$) from the original fixed points for 90 models, 30 for $\ell = 3, 4, 5$. The number of fixed points was 94, 143 and 198 for $\ell = 3, 4, 5$, respectively. For each fixed point, except the origin, 8 were imposed for each value of $d$, as shown in Fig. 3(c), resulting in a total 38,016 imposed fixed points. As shown in Table 1, the smaller the $d$, the greater the percentage of fixed points that do not change type, as expected from Conjecture 1. Also, for higher values of $\ell$ the percentage of fixed points that do not change type is smaller because of the increase of degrees of freedom.

For the rest of this section, NAR-RM was applied using the evaluation region $R$, defined by four times the size of the smallest rectangular area of dimensions $dx$ and $dy$ that contains the demonstrations. The demonstrated trajectories were kept in the center of this region. However, for the sake of a better visualization, the figures of this paper will be limited to only half of $R$. The discrete map $M$ of $R$ was defined with 100 rows and 100 columns. To impose the fixed points, we tried $m = 8$ points as shown in Fig. 3(c) and the distance $d$ was defined as $5 \times \text{dist}$, where $\text{dist}$ is the largest value between the width and length of a cell of $M$. For the models of this paper, $d \approx 7.5$ mm. Also, $\text{MaxIt}$ in all functions was set to 15 and $\text{SEAMax}$ has been set to be 20% larger than the current $\text{SEA}$ and $\text{AlphaMin}$ has been set to be 20% smaller than $\alpha$-distance.

Table 2 shows results of 90 models identified using the unconstrained LS estimator. Notice that 9 models do not present an attractor target and 41 are already RM models.

After applying the proposed NAR-RM procedure, it was possible to transform the 90 initial models into RM models. Fig. 4 shows typical results obtained after the execution of each function of the NAR-RM procedure. For this specific figure, the identified model is given by:

$$x(k) = +0.983506x(k-1) + 0.096590y(k-1) - 0.000078x(k-1)^2 + 0.005253y(k-1)^2 - 0.000538y(k-1)^3 - 0.016513x(k-1)y(k-1) - 0.000300x(k-1)^2 y(k-1) - 0.004126x(k-1)^2$$

$$y(k) = +0.779775y(k-1) - 0.000042x(k-1)^2 - 0.015285x(k-1)y(k-1) - 0.002493y(k-1)^2 - 0.000216x(k-1)^2 y(k-1) - 0.004130x(k-1)^2 - 0.00102y(k-1)^3 - 0.001130x(k-1)^2 + 0.000001x(k-1)y(k-1)^2.$$
Fig. 4. Execution the NAR-RM procedure. The figures show the phase portrait of the identified models in blue, the fixed points in yellow (saddle), green (attractor) and red (repellor), and the basin of attraction in green. Demonstrated reaching motions are shown as red trajectories and executed motions are in black. All the axis are shown in millimeters. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Number of models (out of the 41 RM models) with improved performance after applying the NAR-RM procedure.

<table>
<thead>
<tr>
<th>RM models</th>
<th>Total</th>
<th>$\ell = 3$</th>
<th>$\ell = 4$</th>
<th>$\ell = 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improved SEA</td>
<td>30</td>
<td>14</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Improved $\alpha$-distance</td>
<td>39</td>
<td>17</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>Improved both</td>
<td>28</td>
<td>13</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

It is important to mention that in our tests with the 9 models that initially did not have a fixed point attractor at the target, the `getTargetAttractor` function was able to find models with a fixed point attractor at the target by imposing fixed points with a small value of $d (\approx 1 \text{ mm})$.

By applying the NAR-RM procedure to the 41 models obtained used unconstrained LS that already had the minimum requirements to be considered RM models, it was possible to improve SEA for 30 of them, $\alpha$-distance for 39 and both criteria simultaneously for 28 models, as shown in Table 3.

To evaluate the effect of the degree of nonlinearity in the NAR-RM models we refer to Table 4. Notice that the degree of nonlinearity $\ell = 5$ obtained best SEA in 22 models, against 7 of $\ell = 4$ and 1 of $\ell = 3$. By comparing the value of $\alpha$-distance, an opposite behavior is observed in relation to $\ell$. As can be seen, $\ell = 3$ presented the best $\alpha$-distance in 18 patterns, against $\ell = 4$ with 8 and $\ell = 5$ with 4. Then, using the average values of SEA and $\alpha$-distance, we concluded that $\ell = 5$ is the best option for the LASA Handwriting Dataset library.

Fig. 5 presents the phase portraits of 30 models with the best SEA values with at least 10 mm of $\alpha$-distance obtained using NAR-RM. On the left hand side of Fig. 5 the basins of attraction are presented and on the right hand side are the trajectories obtained when the initial points are set to be the initial points of the demonstrations. Those results are comparable (and better in several cases) to the best results published in the literature using the same data base [4,6–9]. A numerical comparison with some of these methods is performed in the next section.

The generalization of the obtained models is exemplified in Fig. 6, were it is shown the reproduction of four of the models in Fig. 5 when the start point is different from the initial points of the demonstrations.

6.2. Comparison with previous work

We compared the performance of NAR-RM with state-of-the-art methods using the benchmark for learned reaching motion generation in robotics [31].

The benchmark considers four scenarios, which include typical perturbations that occur in robot motion: (i) generalization, which is based on the initialization of the model in different starting points; (ii) discrete push, which is based on a sudden displacement of the robot during motion; (iii) continuous push, where the motion is continuously perturbed during a time window; and (iv) target displacement, when the target position is modified during motion.

In each scenario, some metrics are used to evaluate the evolution and performance of the methods. The metrics are divided

2 The benchmark is available at: https://www.amarsi-project.eu/benchmark-framework.
Fig. 5. The 30 models with the best SEA values with at least 10 mm of $\alpha$-distance obtained using NAR-RM. On the left side the target basin of attraction, the fixed points and demonstrations are showed. On the right hand side are the reproductions given by the models with the same initial points of the demonstrations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 6. Execution of four models of Fig. 5 starting from points that are different from the initial points of the demonstrations. Demonstrated reaching motions are shown in red and executed motions are in black. Axes are shown in millimeters.

Table 5
Different metrics used by the benchmark [31].

<table>
<thead>
<tr>
<th>Category</th>
<th>Property</th>
<th>Measure</th>
<th>Label used in Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric</td>
<td>path</td>
<td>RMSE</td>
<td>trajectory-position-error</td>
</tr>
<tr>
<td></td>
<td>target position error</td>
<td>$L^2$-norm</td>
<td>target-position-error</td>
</tr>
<tr>
<td>Kinematic</td>
<td>velocity profile</td>
<td>RMSE</td>
<td>trajectory-velocity-error</td>
</tr>
<tr>
<td></td>
<td>speed profile</td>
<td>$R^2$</td>
<td>R2-speed</td>
</tr>
<tr>
<td></td>
<td>target velocity error</td>
<td>$L^2$-norm</td>
<td>target-velocity-error</td>
</tr>
<tr>
<td></td>
<td>movement duration</td>
<td>see Equation(6) in [31]</td>
<td>normalizedFinalTime</td>
</tr>
<tr>
<td>Kinematic &amp; Geometric</td>
<td>trajectory duration</td>
<td>$R^2$</td>
<td>R2</td>
</tr>
<tr>
<td></td>
<td>power law</td>
<td>$\Delta R^2$</td>
<td>PL-R2</td>
</tr>
<tr>
<td></td>
<td>minimum jerk</td>
<td>RMSD</td>
<td>mean-jerk</td>
</tr>
<tr>
<td>Software model</td>
<td>processing time</td>
<td>ms</td>
<td>MeanComputationTime</td>
</tr>
</tbody>
</table>
Four kinematic metrics were evaluated. The first one is trajectory-velocity-error, which measures the mean square error (RMSE) between the reproduced velocity and the demonstration velocity. The results for this metric are shown in Fig. 8(a). It is observed that CLF-DM presented the best median value, followed by NIFV-LL. NAR-RM obtained the third best value, which is very close to the NIVF-LL. The second metric, named R2-speed, compares the velocity profiles of the trajectories demonstrated and reproduced with standardized duration using the coefficient of determination, denoted by $R^2$ measure. Fig. 8(b) shows the results of the R2-speed metric, where CLF-DM obtained the best medians values, followed by NIFV-LL and NAR-RM. Again the median value of NAR-RM and NIFV-LL are very close. NormalizedFinalTime is the third metric, which compares the duration of the reproduction movement with the corresponding demonstration. In this metric, the best value is the closest to 1. In Fig. 8(c) it is possible to observe that NIVF-LL presented the best median value and NAR-RM the worst. Even with the worst result, the value obtained by NAR-RM was only 14% worse than the best result (see Table 6). The last metric of the kinematic level, target-velocity-error metric, measure the final velocity when reaching the target using L2-norm. The results obtained for all the methods in the benchmark had the median value of zero.

In relation to the control laws, i.e. geometric and kinematic metrics together, metrics R2, PL-R2 and mean-jerk were used. Metric R2 was used to evaluate the trajectory accuracy, where the shape information an velocity profile are evaluated simultaneously. Fig. 9(a) shows the results of R2, where CLF-DM obtained the best result, followed by NIFV-LL and NAR-RM, with medians values obtained by the three methods being very close (see Table 6). PL-R2 is the 2/3 power law that uses the R2 measure to predict the speed profile of the trajectories. The result of the metric can be seen in Fig. 9(b) where NIVF-QLG obtained the best performance followed closely by NAR-RM. The last metric, mean-jerk, uses the root of the mean squared derivative (RMSD) of a trajectory and the results are shown in Fig. 9(c). For this measure the NIVF-QLG presented the best result followed by NAR-RM.

Table 6 shows the median values and the percentage in which each value is worse in relation to the best value found, represented by the notation $w\%$, for the geometric, kinematic and kinematic & geometric levels. Analyzing the Table 6, we have seen that CLF-DM proves to be the best method when considering geometric and kinematic metrics, but presented the worst performance in two of the three metrics of the kinematic & geometric set. NIFV-LL and NIVF-QLG showed poor performance in the geometric metric and an average performance in the kinematic metrics. Regarding the geometric & geometric set, NIVF-QLG obtained an average result, while NIVF-QLG proved to be the best method. Evaluating the NAR-RM, we have seen that it obtained an excellent result in the geometric metric, an average performance in the kinematic metric and a good result in the kinematic & geometric metrics. By analyzing all this context, we conclude that NAR-RM, even not been the best method in each individual metric evaluated, may be considered the best alternative when one wants a good precision in the trajectories learned (geometric level), and, at the same time, needs some guarantee that the trajectories can be followed by a low level controller (kinematic & geometric level). This characteristics cannot be achieved simultaneously by CLF-DM, which is very good on geometric and kinematic metrics, but requires a lot from the controller, and NIVF-QLG, which demand low controller effort but presents large geometric and kinematic errors.

The MeanComputationalTime metric is used to estimate the computational complexity of the motion generator algorithm to provide the next desired state, based on the current state of the motion. The NIVF-LL and NIVF-QLG methods obtained the shortest median time with a value of 0.07 ms, CLF-DM with 0.23 ms and NAR-RM with 0.54 ms. Our method showed the poorest performance among the methods evaluated, this probably happened because, since our method is not global, it is necessary some extra checks to see if the trajectories are inside the target’s basin of attraction. However, the time it takes to compute is still very small indicating it can be used in real time, as will be shown in the next two subsections.

Fig. 7. The performance of NAR-RM with the other methods for geometric level metric trajectory-position-error. The results of the metric for the four benchmarks were compiled and are shown together.
Fig. 8. The performance of NAR-RM with the other methods for kinetic level metrics: trajectory-speed-errors, R2-speed, normalizedFinalTime. The results of each metric for the four benchmarks were compiled and are shown together.

Fig. 9. The performance of NAR-RM with the other methods for Kinematic & geometric level metrics: R2, PL-R2 and mean-jerk. The results of each metric for the four benchmarks were compiled and are shown together.

Table 6

<table>
<thead>
<tr>
<th>Method</th>
<th>Geometric trajectory-position-error median</th>
<th>w%</th>
<th>Geometric trajectory-velocity-error median</th>
<th>w%</th>
<th>R2-speed median</th>
<th>w%</th>
<th>normalized-FinalTime median</th>
<th>w%</th>
<th>Kinematic &amp; Geometric R2 median</th>
<th>w%</th>
<th>PL-R2 median</th>
<th>w%</th>
<th>mean-jerk median</th>
<th>w%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLF-DM</td>
<td>5.08</td>
<td>0</td>
<td>0.051</td>
<td>0</td>
<td>-0.23</td>
<td>0</td>
<td>1.03</td>
<td>3</td>
<td>0.94</td>
<td>0</td>
<td>4.46</td>
<td>79</td>
<td>131.06</td>
<td>119</td>
</tr>
<tr>
<td>NAR-RM</td>
<td>5.34</td>
<td>5</td>
<td>0.067</td>
<td>31</td>
<td>-1.07</td>
<td>360</td>
<td>1.15</td>
<td>14</td>
<td>0.86</td>
<td>9</td>
<td>20.39</td>
<td>2</td>
<td>83.93</td>
<td>40</td>
</tr>
<tr>
<td>NIVF-LL</td>
<td>8.09</td>
<td>50</td>
<td>0.064</td>
<td>25</td>
<td>1.90</td>
<td>0</td>
<td>1.00</td>
<td>0</td>
<td>0.88</td>
<td>6</td>
<td>16.48</td>
<td>21</td>
<td>111.02</td>
<td>85</td>
</tr>
<tr>
<td>NIVF-QLG</td>
<td>7.61</td>
<td>50</td>
<td>0.075</td>
<td>47</td>
<td>-1.65</td>
<td>606</td>
<td>1.12</td>
<td>12</td>
<td>0.80</td>
<td>15</td>
<td>20.81</td>
<td>0</td>
<td>59.92</td>
<td>0</td>
</tr>
</tbody>
</table>

6.3. Application to a mobile robot

NAR-RM was used to teach a mobile robot how to reach a target from several initial positions. Applications for this include object manipulation by pushing, where a robot needs to reach a specific point of the object by following a restricted set of trajectories that would move the object in the correct direction. In our experiments we used an iRobot’s Create robot equipped with a small laptop and localized by set of external cameras and the ARToolkit software. Fig. 10(a) shows a snapshot of the robot during the experiment. A video of the experiment can be found at https://goo.gl/AqMLAH.

In the first part of the experiment the robot was controlled by a human using a joystick while position data was saved. After the model is learned, a feedback linearization controller was used to track the resulting vector field [32]. Basically, using the identified NAR model $F(x, y)$, we compute a vector $u = F(x, y) - [x, y]^T$, for each robot position $[x, y]$, set its norm to the desired velocity of the robot and compute its linear and angular velocity as:

$$
\begin{bmatrix}
\nu \\
\omega
\end{bmatrix} = \begin{bmatrix}
\cos(\theta) & \sin(\theta) \\
-\sin(\theta) & \cos(\theta)
\end{bmatrix} \frac{d}{d} [x, y]^T,
$$

where $d$ is the distance of a control point in front of the robot and $\theta$ is the robot orientation.

Fig. 11 shows robot trajectories executed based on the identified model overlaid on the model phase portrait. Results of other experiments can be seen in the accompanying video.

6.4. Application to a manipulator

NAR-RM was also applied to teach reach motions to a robotic manipulator. Unlike previous sections where we considered planar trajectories, in this experiment we used 3D trajectories represented by the position of the actuator in the Cartesian space, i.e. $(x, y, z)$.

In our experiments we used a Comau Smart Six manipulator equipped with an in-house-built sensor that allows a user to drive the robot end-effector by pushing it around the workspace.

In the first part of the experiment the robot was controlled by a human using the sensor to reproduce 3 trajectories similar to the green one in Fig. 10(b). Each trajectory was represented by 51 discrete points and the positions were obtained using the
direct kinematics implemented by the robot system. Instead of SEA, which was only defined for 2D, the model in 3D was learned using the RMSE metric to measure the accuracy of the reproduction of a set of demonstrations \( Y(\mathbf{m}) \). RMSE is computed as:

\[
\text{RMSE} = \sqrt{\frac{\sum_{m=1}^{M} \sum_{t=1}^{T(\mathbf{m})} (Y(\mathbf{m},t) - \hat{Y}(\mathbf{m},t))^2}{\sum_{m=1}^{M} \sum_{t=1}^{T(\mathbf{m})} (Y(\mathbf{m},t) - \bar{Y}(\mathbf{m}))^2}},
\]

where \( T(\mathbf{m}) \) is the number of points in the \( \mathbf{m} \)th demonstration, \( \bar{Y}(\mathbf{m}) \) is the average value of the trajectory \( \mathbf{m} \), \( Y(\mathbf{m},t) \) is the point \( t \) of trajectory \( \hat{Y}(\mathbf{m}) \) that starts in the same starting point of the demonstration \( Y(\mathbf{m}) \) and is generated by the model.

To evaluate the model, some executions were performed by initializing the robot in distinct parts of the workspace as shown in Fig. 12. A video of this and other experiments can be seen at https://goo.gl/AqMLAH.

7. Discussion

The results in the previous section show that NAR-RM can be used to teach robots by demonstrations. We compared the approach with state-of-the-art methods using a benchmark and demonstrated that our approach is compatible with previous ones and may be considered the best alternative when it is needed a good precision in the trajectories learned, and, at the same time, a low effort for the low level controller. Another possible advantage of the proposed approach in relation to previous methods is the simplicity and compactness of the obtained model. Also, differently from other models (e.g. neural networks), the one proposed here can benefit from the vast literature on system identification that is able to use the model structure and parameters to extract some characteristics of the system [28,33,34].

Regarding the guarantee of stability, we clarify that at the moment, there is no general theoretical result for ensuring global stability of nonlinear systems. Even Lyapunov’s theory usually depends on the search for the greatest possible domain of attraction, within which the system is stable, relying on a pre-specified choice, often a suboptimal one, of a positive definite candidate Lyapunov function, as SEDS [4] and \( r \)-SEDS [7]. Another possibility is the use of an external signal to ensure stability as in CLF-DM [6]. Other approaches such as BM [5], NIVF [8] and now, NAR-RM, only present guarantees of local asymptotic stability to the target. It is important to mention that, in the authors’ opinion, this is not a significant drawback, since a desirable characteristic of the learned system is that the format of the executed trajectory should resemble the demonstrations. This is not guaranteed for all global stable methods, which may generate totally different motions for points that are far from the target. Therefore, the major advantage of global methods, which is a good robustness to perturbations, caused by the existence of an infinite \( \alpha \)-distance value, may be overcome by fairly high \( \alpha \)-distances. This is corroborated by the numerical results presented before, that indicate that NAR-RM can deal with typical perturbations that occur in robot motion and still have good trajectory reproducibility.
The proposed method was the slowest one among the ones tested using the benchmark. As discussed before, this may happened because we have extra tests to guarantee that the motion is inside the target basin of attraction. However, the method is still quite fast and applicable to controlling real systems, as illustrated in the previous subsections using the mobile robot and the manipulator. In fact, a C++ implementation of the execution phase of the method in a Core i5 computer only takes 38 μs, on average, to run.

A drawback of the proposed method is the significant increase in computation time with the number of dimensions to estimate the basin of attraction.

8. Conclusion

This paper presented NAR-RM, a new method for programming a robot by demonstrations. NAR-RM encodes reaching motions using polynomial, non-linear autoregressive (NAR) models. The identified models are autonomous and guarantee local asymptotic stability to the target. Thus, besides a good reproduction of the demonstrations, the method extrapolates reasonable movements when the robot starts at positions that are different from the ones learned.

Instead of directly using standard system identification techniques, this paper proposes the novel idea of modifying the location of some fixed points of a model with the objective of obtaining or improving reaching motion models. We proposed computational algorithms that efficiently identify the model using a set of demonstrations and provide a MATLAB implementation of such algorithms [16].

To evaluate the accuracy of the learned movements we use the swept error area (SEA) and introduced a new metric, called α-distance, to estimate the quality of the target basin of attraction. Our experimental results demonstrate that the method is capable of identifying (learning) dynamic systems that model reaching motions with good accuracy and robustness guarantees. By comparing the proposed method with the state-of-the-art we conclude that the method is an excellent alternative to current methods when it is needed a good precision in the trajectories learned, and, at the same time, a low demand of the low level controller. Actual robot experiments illustrate the applicability of the method to control real systems.

Future work includes the proposition of new strategies of choosing fixed points and estimating basin of attraction in the learning phase of the method. By replacing the greedy solution used in the current implementation by stochastic methods, for example, we could highly speed up model optimization. This would be important to apply the methodology to other systems, such as aerial vehicles, which work on higher dimensional spaces.

Acknowledgments

The authors acknowledge the financial support of CNPq, grant 465755/2014-3 (INCT), and Fapemig, grant TEC-APQ-00850-15. The first author also acknowledges the support from Federal University of Itajubá. G. Pereira and L. Aguirre hold fellowships from CNPq.

Appendix. Basin of attraction estimation

Assuming a discrete representation M of the workspace, Algorithm 3 can be used to estimate the basin of attraction of a model with an attractor at the target. The algorithm uses two sets, one to store the cells belonging to the basin of attraction called Basin and another to store the cells of M already tested, called Visited. Initially, the cell with the target is put in sets Basin and Visited (lines 2–3). While Visited does not have all cells of M the model is iterated for each cell C around the Visited cells, starting at the center point of C until it reaches a neighbor cell Basin (lines 5–7). If next does not belong to the basin, then C is linked to next, indicating that a iteration of the model starting from C reaches Basin (lines 8–9). If next belongs to the Basin, then C and all cells linked to it are put in Basin (line 11–12). All neighbors of the Visited cells are put in Visited (line 15), expanding the set of already checked cells and returning to line 4. After checking the entire map M, the algorithm returns the basin of attraction in set Basin.

An important observation is the possibility of having a fixed point attractor in cell C. In this case, the iteration of the model in line 7 would probably not reach another cell. In this case, the function in line 7 will never end if it is not able to detect such a situation. In our implementation, we only allow a maximum number of iterations in each cell. If the maximum number of iterations is reached, we assume that cell C has an attractor and return that next = C.

Algorithm 3: Estimate target basin of attraction

<table>
<thead>
<tr>
<th>Input:</th>
<th>Model, map M</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>Put target cell in Basin;</td>
<td></td>
</tr>
<tr>
<td>Put target cell in Visited;</td>
<td></td>
</tr>
<tr>
<td>while Visited does not have all cells of M do</td>
<td></td>
</tr>
<tr>
<td>foreach C cell in the neighborhood of Visited do</td>
<td></td>
</tr>
<tr>
<td>Put C in Aux;</td>
<td></td>
</tr>
<tr>
<td>Iterate the Model starting from C until reaches the next cell next;</td>
<td></td>
</tr>
<tr>
<td>if next ∈ Basin then</td>
<td></td>
</tr>
<tr>
<td>link the cell C to the Cnext;</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>Put C in Basin;</td>
<td></td>
</tr>
<tr>
<td>Put cells linked to C in Basin;</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>Put Aux in Visited</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>return Basin;</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

References
