Predicting the internal model of a robotic system from its morphology

Angel J. Duran, Angel P. del Pobil

Robotic Intelligence Laboratory, Jaume I University, Castellon, Spain

Abstract

The estimation of the internal model of a robotic system results from the interaction of its morphology, sensors and actuators, with a particular environment. Model learning techniques, based on supervised machine learning, are widespread for determining the internal model. An important limitation of such approaches is that once a model has been learnt, it does not behave properly when the robot morphology is changed. From this it follows that there must exist a relationship between them. We propose a model for this correlation between the morphology and the internal model parameters, so that a new internal model can be predicted when the morphological parameters are modified. Different neural network architectures are proposed to address this high dimensional regression problem. A case study is analyzed in detail to illustrate and evaluate the performance of the approach, namely, a pan-tilt robot head executing saccadic movements. The best results are obtained for an architecture with parallel neural networks. Our results can be instrumental in state-of-the-art trends such as self-reconfigurable robots, reproducible research, cyber-physical robotic systems or cloud robotics, in which internal models would available as shared knowledge, so that robots with different morphologies can readily exhibit a particular behavior in a given environment.

Keywords: model learning, internal model, morphology, neural networks, visual learning

1. Introduction

A robot system interacting with a particular environment is characterized by its morphology and internal model. The morphology could be considered as a representation of the physical properties of the robotic system. Most of these properties can be measured. In turn, the internal model represents the interaction between the robot system and the environment. Different research areas within Robotics have been established that differ in the way in which the relations among these three elements are handled. Thus, in some cases, the morphology of a robot is determined by its interaction with the environment [1], whereas in other cases, the simulation of the environment and incomplete self-knowledge models the robot behavior [2]. A third perspective estimates the internal model from the interaction of a particular kind of robot with the environment. These model learning approaches typically consider exclusively the relationships between states and actions, and the information about the states and actions of the past, present and even the expected future is needed to model the robot behavior. The process of learning is a regression problem where the training samples are obtained from the state and controls of the plant along time [3]. Internal-model-based control theory is well established, but internal models are typically expressed as mathematical models of the plant, normally by means of a set of differential equations [4].

While classical robotics relies on those manually generated models, an autonomous cognitive robot needs to automatically generate internal models based on information extracted from data streams accessible to the robot from the environment [5]. From an operational point of view, there are two approaches to deal with the adaptation of the robot internal model: adaptive control and model learning. Whereas the former uses on-line parameter identification [6], the latter uses supervised learning. In our research, we adopt the model learning approach because it makes no assumption about the structure of the model and includes all phenomena in a general function built out of experimental data.

A relationship always exists between the internal model and the morphology, and they are inseparable from each other because both affect how information is processed in the robotic system [7]. The first aim

Email addresses: abosch@uji.es (Angel J. Duran), pobil@uji.es (Angel P. del Pobil)

of this article is precisely to propose a model to learn the relationship between the internal and morphological parameters of a robotic system (section 2). To achieve this goal, we change the point of view of the problem, focusing on the environment, because robots need to approximate the model of the environment to develop their behaviors. Thus, instead of the problem of one robot updating their internal model according to the changes of the environment, we look at the problem as one environment interacting with many robots with their known morphology. After learning the internal model, its relationship with the morphology can be estimated. Since this relationship is unknown, we propose an approach based on neural networks to learn it. This kind of methods are recognized to solve problems when a complete formulation is not known or a mathematical representation is not explicitly available [8].

The ability to predict the internal model of a robot from its morphology has a great interest in the current state of the art in robotics research and applications. So-called *morphofunctional* machines can change their functionality by modifying their morphology and some modular self-reconfigurable robots can *morph* [7] . The rationale is that much of the functionality of a robot is due to its particular morphology, and by altering it depending on the task, its performance, adaptivity and versatility, will substantially improve.

In the last decade there has been a growing concern in the community regarding how progress in robotics research has been hampered due to differences in hardware that make difficult to compare alternative approaches, apply benchmarks, or replicate results [9]. The problem is that robots with a similar design still have differences in their morphological parameters. A possible solution is to make knowledge transferable to different *embodiments* or morfologies. For instance, Felip et al. [10] proposed a methodology based on abstract state machines that are automatically translated to embodiment-specific models. Getting the internal model directly from the morphology will contribute to progress in this direction.

Arguably, our research can have a high potential for impact in the 4th industrial revolution, the so-called Industry 4.0. In this context, a robot is no longer regarded as a standalone machine, but rather as a networked *Cyber Physical System* [11] [12] endowed with interoperability, i.e., the ability to connect and communicate with other devices via the Internet [13]. In this sense, the rationale of so-called *Cloud Robotics* [14][15] is that instead of trying to increase the performance and functionality of isolated robot systems, knowledge is reused through the shared memory of multiple robots. Endeavours such as *RoboEarth* [16] collect, store, and share data independent of specific robot hardware. However, for this to be fully operational, internal models should be available on-line so that different robots can exhibit the same behavior in a given environment.

Our second goal is to verify how the modeled relationship can be learned in a real problem. For this, a case study is presented in section 3; namely, a pan-tilt robot head executing saccadic movements. As a result, we remark that learning the relationship between the internal model and the morphology is a high dimensional regression problem for the proposed case study. For managing this machine learning problem, three types of neural networks are proposed in section 4. The results of learning the internal model from the morphological parameters by using the proposed networks are described in section 5, along with a comparative study, and followed by a final discussion and conclusions.

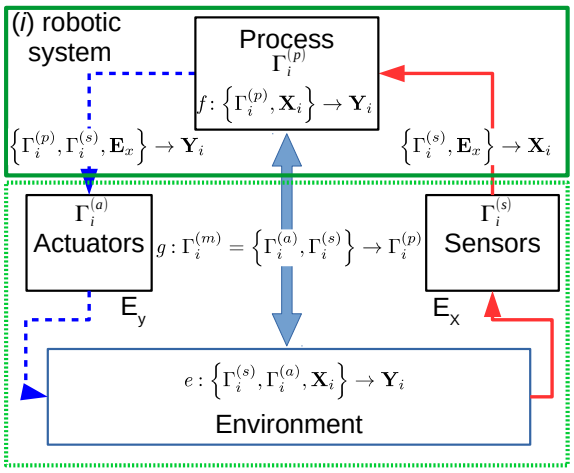

Figure 1: Schema showing the information fluxes and transformations among the different system components and the environment.

2. Model description

2.1. The relationship between morphological and internal model parameters

The proposed methodology in this article, considers a robotic system (i) immersed in a particular and stable environment (Figure 1). From the robot point of view, its sensors transform the data flux ($\mathbf{E}_x$) from the environment and generate an input ($\mathbf{X}_i$) for the robot internal model. This one processes the inputs and generates the outputs ($\mathbf{Y}_i$) to the robot actuators which modify the state of the environment by using an output data flux

($\mathbf{E}_y$) in some way. The flux $\mathbf{E}_x$ only depends on the environment state, however the $\mathbf{X}_i$ input depends on the robot morphology (generally on the sensor parameters ($\Gamma_i^{(s)}$) and $\mathbf{E}_x$. This input flux is processed by the internal model (f) which takes into account the internal model parameters $\Gamma_i^{(p)} = \left\{\gamma_{i,1}^{(p)}, \gamma_{i,2}^{(p)}, ..., \gamma_{i,k}^{(p)}\right\}$, where k is the number of internal parameters for *(i)* system. The function f embodies everything there is to know about the relationship $\mathbf{X}_i$ and $\mathbf{Y}_i$ through the internal model parameters. Finally, the output flux that modifies the environment will be a function of the parameters of the robot actuators ($\Gamma_i^{(a)}$). The set $\Gamma_i = \{\Gamma_i^{(s)} \cup \Gamma_i^{(p)} \cup \Gamma_i^{(a)}\}$ represents the parameters of the robotic system, both morphological and those of the internal model. The set $\Gamma_i^{(m)} = \{\Gamma_i^{(s)} \cup \Gamma_i^{(a)}\} = \{\gamma_{i,1}^{(m)}, \gamma_{i,2}^{(m)}, ..., \gamma_{i,h}^{(m)}\}$ contains the morphological parameters of the system, and h is the total number of sensor and actuator parameters.

Now, the point of view is changed to focus on the environment. The input flux to the environment will be the output flux of the internal model ($\mathbf{Y}_i$). From the environment point of view the relationship (e) between $\mathbf{Y}_i$ and $\mathbf{X}_i$ depends only on the morphological parameters of the system ($\Gamma_i^{(m)}$) and $\mathbf{X}_i$. Whereas from the system point of view, the relationship between $\mathbf{Y}_i$ and $\mathbf{X}_i$ is only a function of the internal model parameters ($f(\Gamma_i^{(p)}, \mathbf{X}_i)$). For a suitable interaction with the environment, the internal model (f) must be an approximation of the real model of the environment (e). The correlation between the morphological and internal model parameters is given by the following functional relationships (figure 1):

$$\Gamma_i^{(p)} = g(\Gamma_i^{(m)}) \tag{1}$$

This statement will be true if the internal model is properly learned and the morphological parameters satisfy:

$$\left.\frac{\partial \Gamma_i^{(s)}}{\partial \mathbf{E}_x}\right|_i = \left.\frac{\partial \Gamma_i^{(a)}}{\partial \mathbf{E}_y}\right|_i = 0 \tag{2}$$

This equation (2) defines the notion of morphological parameters in our formalism, in the sense that they must accomplish the condition that they are independent of the input and output data fluxes from and to the environment for system (i). For example, the focal length of the camera is considered as a morphological parameter if its value does not vary with the input information; i.e., it is not in autofocus mode.

In general, the relationship defined by function f, is estimated using the inputs ($\mathbf{X}_i$) and outputs ($\mathbf{Y}_i$) of the system, and the g transformation is embodied in the environment data flux transformations. The inputs $\mathbf{X}_i$ and outputs $\mathbf{Y}_i$ are represented by a sequential set of vectors $\mathbf{X}_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, ..., \mathbf{x}_{i,u}\}$ and $\mathbf{Y}_i = \{\mathbf{y}_{i,1}, \mathbf{y}_{i,2}, ..., \mathbf{y}_{i,u}\}$ where u is the number of samples ($\mathcal{D}_i = \{\mathbf{X}_i, \mathbf{Y}_i\}$). We assume that each pair of components of $\mathcal{D}_i$ are independent measures of the system. In model learning frameworks, the optimal model structure should be obtained from training data [5], and the model can be fitted into the following function:

$$\mathbf{Y}_i = f(\mathbf{X}_i, \Gamma_i^{(p)}) + \epsilon_i;\ f(\mathbf{X}_i, \Gamma_i^{(p)}) = \phi(\mathbf{X}_i)^T \Gamma_i^{(p)} \tag{3}$$

From a probabilistic point of view, $f(\mathbf{X}_i, \Gamma_i^{(p)})$ can be considered as a conditional expectation $\mathbb{E}(\mathbf{Y}_i \mid \mathbf{X}_i)$, therefore ϵ_i is an expectational error term that can be expressed as $\epsilon_i \equiv Y_i - \mathbb{E}(Y_i|X_i)$.

The model is defined by the value of the internal model parameters $\Gamma_i^{(p)}$. Due to the independence of the measures in $\mathcal{D}_i$, the likelihood function for $\Gamma_i^{(p)}$ is:

$$\mathcal{L}_u(\Gamma_i^{(p)} \mid \mathcal{D}_i) = \frac{1}{u}\sum_{s=1}^{u} log(\mathrm{p}(\mathcal{D}_{i,s} \mid \Gamma_i^{(p)}) \tag{4}$$

Maximizing the function $\mathcal{L}_u(\Gamma_i^{(p)} \mid \mathcal{D}_i)$, we obtain the maximum likelihood estimator (MLE) for $\Gamma_i^{(p)}$ as:

$$\hat{\Gamma}_i^{(p)} \in \underset{\Gamma_i^{(p)}}{\operatorname{argmax}}\, \mathcal{L}_u(\Gamma_i^{(p)} \mid \mathcal{D}_i) \tag{5}$$

There exist many optimization procedures to solve this equation (5). The above definition is not complete because there is no guarantee that such a maximum exists or, when it does exist, it is unique. That is, the proposed system is not able to properly learn the environment model. Depending on the similarity between the real model of the environment and the model proposed by equation (3), this statement will be true. We suppose all proposed systems are able to learn the environment model. As a result of equation (5), two properties are satisfied: first, MLE for $\Gamma_i^{(p)}$ is consistent, that is, when u is large enough, the estimator $\hat{\Gamma}_i^{(p)}$ converges in probability to $\Gamma_i^{(p)}$:

$$\lim_{u\to\infty} \mathrm{p}\left(|\ \hat{\Gamma}_i^{(p)} - \Gamma_i^{(p)}\ | > \delta\right) \to 0 \tag{6}$$

Consequently, the value of $\hat{\Gamma}_i^{(p)}$ can be considered constant when u is large enough. That is, there exists a unique set of optimum parameters $\hat{\Gamma}_i^{(p)}$, that leads to the best approximation of the true function by a certain model. Second, the maximum likelihood estimator is asymptotically normal. That is, as u becomes large, $\hat{\Gamma}_i^{(p)}$ converges in a multivariate normal random variable whose variance is a diagonal matrix. The asymptotic

normality of the maximum likelihood estimator is expressed as:

$$\sqrt{u}\left(\hat{\Gamma}_i^{(p)} - \Gamma_i^{(p)}\right) \xrightarrow{d} \mathcal{N}\left(0, \sigma_{ML}^2 \mathbf{I}\right) \quad (7)$$

where σ_{ML}^2 is called the asymptotic variance of the estimate $\hat{\Gamma}_i^{(p)}$. Asymptotic normality says that the estimator not only converges to the unknown parameter $\Gamma_i^{(p)}$, but it converges fast enough, at a rate $1/\sqrt{u}$.

2.2. Extracting knowledge from multiple robotic systems

If instead of only one system (i), we have v similar systems with different morphologies[1], and each one has an internal model whose properties are fitted into (3), the result is a set of pairs $\{\hat{\Gamma}^{(p)}, \Gamma^{(m)}\} = \{\{\hat{\Gamma}_i^{(p)}, \Gamma_i^{(m)}\} \forall i \in [1, v]\}$. From a statistical point of view, both variables are considered as random because we randomly define $\Gamma^{(m)}$ as the system morphology in a particular range of values and each $\hat{\Gamma}_i^{(p)}$ was obtained from independent trials. Therefore, a regression model such as (3) can be used to learn the relationship between them:

$$\hat{\Gamma}^{(p)} = g(\Gamma^{(m)}, \mathbf{W}) + \xi;\ g(\Gamma^{(m)}, \mathbf{W}) = \Phi(\Gamma^{(m)})^T \mathbf{W} \quad (8)$$

As in the previous case, a maximum likelihood estimator for $\hat{\mathbf{W}}$ can be used for fitting the values of morphological parameters to $\hat{\Gamma}^{(p)}$ obtained for each system. Once the value of $\hat{\mathbf{W}}$ is calculated, a prediction model can be used to obtain an estimation of $\hat{\Gamma}_i^{(p)}$ from the specific morphological parameters. Given a new system (j), defined by its morphological parameters, its internal model parameters can be estimated as:

$$\hat{\Gamma}_j^{(p)*} = \Phi(\Gamma_j^{(m)})^T \hat{\mathbf{W}} \quad (9)$$

2.3. Machine learning problem

Our aim is to find the function g, mapping the morphological parameters and the internal model parameters, as defined by (1). For this, we will need v robotic systems. Each one of these v systems will have previously interacted with the same environment in order to obtain its set of internal model parameters $\Gamma_i^{(p)}$. The size of this set depends on the number of inputs and outputs of the robotic system and the adaptation solution used for estimating the internal model. In turn, the size of $\Gamma_i^{(m)}$ is related to the complexity of the system morphology, and the number of morphological parameters is usually substantially smaller than the number of internal model parameters $\Gamma_i^{(p)}$. Finally, a machine learning tool is needed to approximate equation (8). This tool should be flexible and sometimes it will have to manage a great number of components for $\Gamma_i^{(p)}$ and $\Gamma_i^{(m)}$. An appproach based on neural networks is feasible due to the scalability of the resulting architectures. Since typically the number of components in $\Gamma_i^{(p)}$ is large, the high dimension of the neural network output is a challenging problem [17] and we need to test several approaches to solve it.

3. Case study

In this section, we describe a robotic system that we will use as a case study to test our framework for relating the internal model parameters with the morphological parameters, and using neural networks as machine learning tools. Namely, a pan-tilt robot head executing saccadic movements. There are a number of reasons that justify the selection of this particular example. First, it is a well-known and extensively tested case, with an abundant literature (see for instance the work of Rucci, Edelman and Wray [18] [19] in which they report data of relevant robotic studies using neural network models to adapt gaze shifts to perturbations of the robot morphology); second, the head is composed of sensors and actuators that can be modeled analytically; and, importantly, the proposed regression problem (8) can be separated in two: one can be addressed with a simple neural network, whereas the other is a high dimensional regression. It is to be noted that we need many variations of the robot head; in fact, we will use a huge amount of morphologies in order to be in a data-rich situation, as we explain further ahead. Thus, we can look as this case as having many different pan-tilt heads, with different morphologies, interacting with the same environment, and exhibiting a particular behavior. The environment will be the visual space in front of the robotic system, and the behaviour will be saccadic movements.

3.1. The morphology of the robot system

A simulated robot head is considered as the robotic system for testing the proposed model. This system presents morphological parameters that correspond both to actuators and sensors, since it is composed of two cameras and three controlled DOF. Eight additional prismatic joints are defined in the model so that changes in the head morphology can be easily introduced, resulting in a total of eleven DOF (Figure 2). Table (1) describes the Denavit-Hartenberg model for the left-hand

[1] Same number and kind of morphological parameters but different values

side of the head, the right side has the same description and it shares the first joint ρ_t. The prismatic joints $\{q_2, q_3, q_4, q_6\}_{left}$ and $\{q_2, q_3, q_4, q_6\}_{right}$ define the geometry of the head and do not modify their values while the head is executing a task. The values of these eight parameters ($\Gamma_i^{(a)} \in \mathbb{R}^8$) describe the robot morphology in terms of head actuators. Since the aim of this case study is to check whether or not the proposed approach is feasible as a tool for estimating the system internal mode, we assume, in order to avoid interferences, that the joint movements are precise enough so that noise does not need to be added to the model.

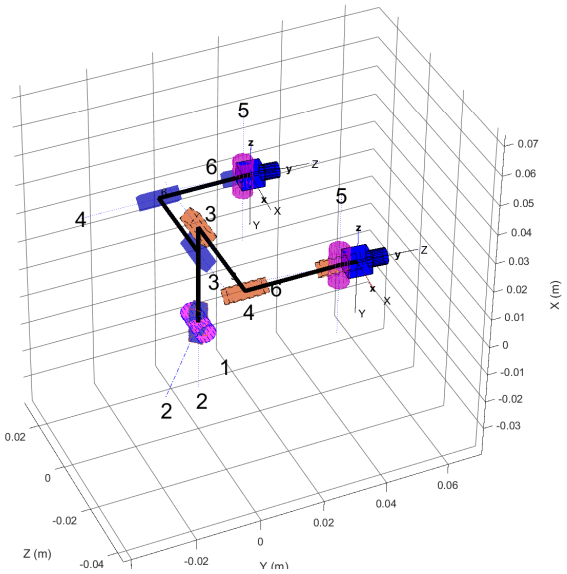

Figure 2: Example of the head model. In this schema, there are two overlapping kinematic chains. The three revolute joints are represented by cylinders. Joint number one is shared by both chains. Prismatic joints, that modify the head morphology, are represented by parallelepipeds.

The system is completed with two camera sensors, and the pinhole model is taken to simulate their behavior. Each camera can be described by using at least four parameters: focal length (f), pixel size (s) (supposed squared), width (w) and height (h) of the images. Therefore, the parameters $\{f, s, , w, h\}_{left}$ and $\{f, s, , w, h\}_{right}$ are regarded as the morphological sensor parameters ($\Gamma_i^{(s)} \in \mathbb{R}^8$). Thus, there are a total of 16 morphological parameters for each robotic head configuration ($\Gamma_i^{(m)} \in \mathbb{R}^{16}$).

3.2. The environment

The environment is a limited physical space in front of the pan-tilt head where a virtual object is randomly placed within the field of view of the two cameras. The size of this space is between 0.5m and 2.5m. This object is static while the head is moving. The error in the stimulus position is ignored, since it is out of the scope of this work.

Table 1: Denavit-Hartenberg model of the left side of the head. ρ_p, ρ_t are the revolute joints for the pan and common tilt motors. The model for the right-hand side is the same, and it shares the ρ_t joint

joint	ρ(rad)	r(m)	a(m)	α(rad)	Offset	Type
q_1	ρ_t	0	0	$\pi/2$	$\pi/2$	R
q_2	0	0	0	-$\pi/2$	0	P
q_3	$\pi/2$	0.055	0	$\pi/2$	0	P
q_4	$\pi/2$	0.055	0	$\pi/2$	0	P
q_5	ρ_p	0	0	$\pi/2$	π	R
q_6	0	0.01	0	$\pi/2$	0	P

3.3. The behavior of the robotic system

The system will exhibit a certain behavior as a result of its interaction with the environment. Varied different types of tasks or behaviors could be considered. One source of inspiration is biology. Since the robotic head is endowed with two cameras, and the environment defines a visual stimulus, there exists a natural similarity with the movements of eyes in some animals. A number of real robotic systems exist, that fit into the proposed morphology [18] [19] and some are inspired by the neuroscience of visuomotor coordination [20]. Thus, in this case study, the proposed robot head has to move its cameras in a way corresponding to the saccadic movements of the eyes. A saccade is a fast eye movement that shifts the gaze towards a target point of interest; it can also be used to scan the visual space [21]. The retinotopic position of the stimulus (i.e., its position on the image captured by the camera) has to be converted into a shift of the eye (camera) position. The saccadic gaze shifts are planned in 3D. Therefore, from the robot point of view, generating a saccade requires to solve an inverse control problem. In this saccadic behavior, the visual position of the stimulus in relation to a target position of the eyes is open loop with respect to vision. Learning this transformation requires to learn the inverse kinematic model of the robot head. For doing this, we use *feedback error learning* (FEL) [22]. This model consists of two inverse controllers: a fixed feedback controller (**B**) slowly drives the system toward the target and provides a learning cue to a second adaptive controller ($\mathbf{C}_f$). This last one yields an inverse model of the robot head which is used to rectify the output of **B** (Figure 3). The performance of the system is evaluated by the visual error after a saccadic movement. The visual stimulus should

be centered in the image after a saccade, but the approximation $\mathbf{P}^{-1}(\mathbf{t'}) \approx \mathbf{B}\mathbf{t}'$ introduces an error in the model (see Figure 3), even though the noise in the visual stimulus is not considered. Improving the execution of saccadic movements is not the goal of this work, but rather we use this specific behavior as a test case to illustrate and validate our model.

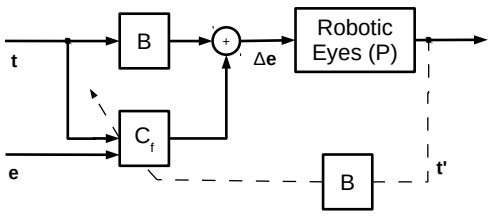

Figure 3: FEL. The visual position of the stimulus ($\mathbf{t}$) and the current eye position $\mathbf{e}$ are converted into a motor command $\Delta\mathbf{e}$ by summing up the contributions of a fixed element $\mathbf{B}$ and adaptive element $\mathbf{C}_f$. Changing the $\mathbf{C}_f$ state requires a motor error $\mathbf{P}^{-1}(\mathbf{t}') \approx \mathbf{B}\mathbf{t}'$.

The input of the controller is the visual target ($\mathbf{t}$) and the current eye position ($\mathbf{e}$), while the output is the saccade command (Δe). The robot head actuators receive this command and they move the head accordingly. After the movement, the inverse controller $\mathbf{C}_f$ is adapted using the new visual position of the stimulus $\mathbf{t}'$ converted into a motor error. Using this approach, the adaptive filter learns to compensate the poor response of the fixed feedback control ($\mathbf{B}$).

An *a priori* estimation of $\mathbf{B}$ is needed to convert the visual target into a movement of the eyes. The procedure to estimate the fixed controller is based on the fixation of the stimulus in the center of the camera images. Afterwards, motor babbling is generated [23]. Under these conditions, the variation of the visual target position ($\Delta\mathbf{t} = \mathbf{t}$) in the images can be correlated with the motor command (Δe). Therefore, if a set of b pairs $\{\mathbf{t}_i, \Delta\mathbf{e}_i\} = \{\{t_{i,j}, \Delta e_{i,j}\}\, \forall j \in [1, b]\}$ for a particular robotic system (i) is generated by motor babbling, $\mathbf{B}_i$ can be directly obtained from least squares regression:

$$\mathbf{B}_i = \Delta\mathbf{e}\mathbf{t}_i^T \left(\mathbf{t}_i\mathbf{t}_i^T\right)^{-1} \tag{10}$$

When $\mathbf{C}_f$ is learning, the mean visual error of a saccade movement decreases until the system reaches a point in which it learns so slowly that the adaptive controller barely changes. A general approach to learn the proposed $\mathbf{C}_f$ is to use a neural network with on-line adaptation. In the first attempts of the training process, the neural network used for representing $\mathbf{C}_f$ controller gives a poor performance because the weights of the neural network (θ_{0_i}) are untrained. After a number of interactions with the environment, the visual error should be reduced and the weights of the neural network (θ_i) barely change at this point. These weights store the information from the interaction of the system with the environment. Both the fixed controller parameters $\mathbf{B}_i$ and the adaptive controller parameters (the neural network weights θ_i) represent the internal model of the robotic system in this example $\hat{\Gamma}_i^{(p)} = \{\mathbf{B}_i, \theta_i\}$.

3.4. The dataset creation

For estimating the relationship between the morphological $\Gamma^{(m)}$ and the internal model $\Gamma^{(p)}$ parameters defined by (8), we have proposed a particular robot morphology which can properly be changed to generate many different robotic heads, and all of them exhibit the same behavior interacting with the same environment.

3.4.1. Generating multiple robot heads

A number of different setups of robot heads (v=44271) have been randomly simulated with the morphological parameters described in the previous section. For avoiding unfeasible configurations, a reasonable interval is defined for each parameter. The head morphology is generated by randomly selecting a parameter for the left side within the interval shown in the first column of Table (2), and then the value for the corresponding parameter for the right side is calculated by adding a random value within the range in the second column of the table. For creating the dataset some morphological parameters were scaled to avoid errors in the learning algorithm, e.g. the height and the width for the camera images were converted to decimeters using pixel size. In this way, all morphological values had the same magnitude.

3.4.2. Learning the robot behavior

Learning the fixed controller. Using the motor babbling procedure described in section 3.3, we generate a set of b pairs $\{\mathbf{t}_j, \Delta\mathbf{e}_j\}\forall j \in [1, b]$. In this case, $\mathbf{t}_j$ is the stimulus image position in both cameras after the head movement. Due to the fact that the stimulus is in the center of both images before the onset of the movement, $\mathbf{t}_j$ represents the visual variation in the stimulus position ($\mathbf{t}_j \in \mathbb{R}^4$). Furthermore, each head has three joints to control, then its joint space has three dimensions ($\Delta\mathbf{e}_j \in \mathbb{R}^3$). According to equation (10), matrix $\mathbf{B}_i$ has dimensions 3×4. $\mathbf{B}_i$ is computed for all v head setups. Parameter b has to be selected so that a good estimation of $\mathbf{B}_i$ is obtained, but trying to reduce the computation time. A number of head setups

Table 2: Morphological parameters to generate the robotic system dataset.

Prismatic joints (cm)	
Left side	Right side
$\Gamma^p_{i,1} = q_2 \in [-0.054, 0.054]$	$\Gamma^p_{i,2} = q_2 = \Gamma^p_{i,1} + [0, 0.01]$
$\Gamma^p_{i,3} = q_3 \in [0, 0.07]$	$\Gamma^p_{i,4} = q_3 = \Gamma^p_{i,3} + [0.035, 0.07]$
$\Gamma^p_{i,5} = q_4 \in [-0.02, 0.054]$	$\Gamma^p_{i,6} = q_4 = \Gamma^p_{i,5} + [0, 0.02]$
$\Gamma^p_{i,7} = q_6 \in [0, 0.01]$	$\Gamma^p_{i,8} = q_6 = \Gamma^p_{i,7} + [0, 0.01]$
Cameras parameters:	f(px); s(m/px);w(px);h(px)
Left camera	Right camera
$\Gamma^p_{i,9} = f_l \in [340, 1920]$	$\Gamma^p_{i,10} = f_r = \Gamma^p_{i,9} + [0, 200]$
$\Gamma^p_{i,11} = s_l \in [3.\ 10^{-6}, 7.\ 10^{-6}]$	$\Gamma^p_{i,12} = s_r \in [3.\ 10^{-6}, 7.\ 10^{-6}]$
$\Gamma^p_{i,13} = h_l \in [340, 1920]$	$\Gamma^p_{i,14} = h_r = \Gamma^p_{i,13} + [0, 200]$
$\Gamma^p_{i,15} = w_l \in [340, 1920]$	$\Gamma^p_{i,16} = w_r = \Gamma^p_{i,15} + [0, 200]$
if $\Gamma^p_{i,13} > \Gamma^p_{i,15}$, swap($\Gamma^p_{i,13}, \Gamma^p_{i,15}$)	if $\Gamma^p_{i,14} > \Gamma^p_{i,16}$, swap($\Gamma^p_{i,14}, \Gamma^p_{i,16}$)

(1000) were chosen randomly. The values in $\mathbf{B}_i$ were estimated by varying the number of iterations (**b**) for the selected robot heads. Afterwards, the calculated $\mathbf{B}_i$ is used to predict the visual stimulus position after a movement. The mean square error between the real stimulus position variation in the images and the predicted position shift can be considered as a quality measure for $\mathbf{B}_i$. Figure (4) illustrates the high stability of the $\mathbf{B}_i$ estimation after some 500 iterations. Therefore, we selected a value for b greater than 500 ($b = 600$).

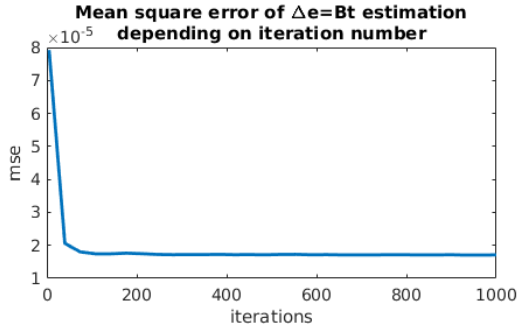

Figure 4: Mean square error for $\mathbf{B}_i$ estimation with 1000 robot head setups versus the number of iterations

Learning the adaptive controller. We trained on-line every head setup to learn their adaptive controllers. A neural network with a single hidden layer was used, keeping the same architecture for all the setups. Namely, an input layer with seven neurons: four neurons for stimulus position in the image (**t**) and three for the head motor commands (**e**); the output layer has three neurons according to the control command variations ($\Delta\mathbf{e}$). Gaussian activations using random sparse features [24] are used for the hidden layer. The randomness of these features allows us to use this set of features for all neural networks. In this way, these features are considered as a component of the neural network architecture. A unique set of random sparse features is generated (Ω_m) and they are used for the training and validation processes. Incremental sparse spectrum gaussian process regression (I-SSGPR) [25] is used for adapting the weights of the neural network. The parameters of the training algorithm have been previously tuned [25]: variance of the model ($\sigma_n^2 = 0.1$), signal variance ($\sigma_f^2 = 1.0$) and number of projections (D=300), therefore $\Omega_m \in \mathbb{R}^{600}$. Each neural network used for learning the adaptive controller of each head setup is initialized using random weights (θ_i^0). As indicated in [25], a covariance matrix (**A**) and a vector (**b**) are needed to train the neural network using a gaussian process (algorithm 1 in [25]). The weights of the networks are adapted using (**A**) and (**b**). Once the network is trained, these two elements are not used to estimate the network output since the mapping between the inputs and outputs is given by the trained weights θ_i only, regardless of how the weights were learned.

For training, a virtual object is placed in the environment, the cameras of the i robot head acquire the images and the visual position of the object is estimated. Using the current output of the neural network ($\mathbf{C}_{f_i}$) and the value provided by the fixed controller, previously calculated ($\mathbf{B}_i$), an incremental control action for the head motors is generated. The head moves according with these control commands and the cameras acquire the new visual stimulus position on the images. The neural network is adapted using the visual error in this position [21]. After a number of iterations, the neural network weights converge to stable values and the visual error is stabilized. To ensure convergence is reached, and considering the time consumption of the training process, each robot head is trained for 1000 iterations. At this point, the resulting weights of each neural network representing an adaptive controller are considered as part of the internal model parameters, since they partially determine the behavior of the system. Each trained weight matrix is represented by $\theta_i \in \mathbb{R}^{600\times3}$. This training process has been repeated three times for each head setup and the final θ_i values are taken as the mean of the three trials.

After the training process, the estimated $\mathbf{C}_f$ and $\mathbf{B}$ are used for executing saccadic movements. In particular, to estimate the error in the training process, each robot head setup was tested using 500 random saccades. Figure (5) shows the probability density function (*pdf*) for

the visual mean error based on these tests.

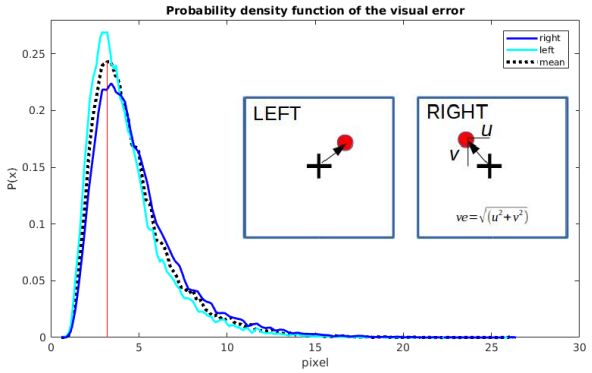

Figure 5: Visual error probability density functions (pdf) for the 44271 trained head setups using the estimated $\mathbf{B}$ and C_f for each one. The error for the two cameras and the average are shown. The considered measure of the final visual error is the mean visual distance in pixels between the stimulus position and the gaze point, for each camera. Thus, these three curves represent the visual error after training for the proposed set of morphologies. These pdfs fit fairly well into a log-normal distribution. The estimation of their parameters expressed in pixels are: for the mean ($\mu = 4.6$,$\sigma = 2.26$); left camera ($\mu = 4.3$, $\sigma = 2.08$); right camera ($\mu = 4.9$ and $\sigma = 2.4$).

Finally, a huge dataset composed of 44271 vectors and matrices has been created after learning the robot behavior. The structure of the dataset is: $\Gamma_i^{(m)} \in \mathbb{R}^{16}, \Gamma_i^{(p)} = \{\mathbf{B}_i \in \mathbb{R}^{3\times4}, \theta_i \in \mathbb{R}^{600\times3}\} \mid i \in [1, v = 44271]$. The challenge is to solve the regression problem formulated in (8). Fortunately, the problem can be decomposed into two parts for the proposed case study: on the one hand $\hat{\mathbf{B}} = g_1\left(\Gamma^m\right)$:

$$\mathbf{B} = g_1\left(\Gamma^{(m)}, \mathbf{W}_1\right) + \xi_1; g_1\left(\Gamma^{(m)}, \mathbf{W}_1\right) = \Phi_1\left(\Gamma^{(m)}\right)^T \mathbf{W}_1 \tag{11}$$

and on the other hand $\hat{\theta} = g_2\left(\Gamma^m\right)$:

$$\theta = g_2\left(\Gamma^{(m)}, \mathbf{W}_2\right) + \xi_2; g_2\left(\Gamma^{(m)}, \mathbf{W}_2\right) = \Phi_2\left(\Gamma^{(m)}\right)^T \mathbf{W}_2 \tag{12}$$

4. Proposed neural networks

4.1. Solving the regression problem for the fixed controller

The regression problem in equation (11) can be solved using a basic single layer neural network. The input layer contains 16 neurons corresponding to the 16 morphological parameters. In turn, there are 12 neurons in the output for 3x4 $\mathbf{B}$ parameters.

4.2. Solving the regression problem for the adaptive controller

Solving equation (12) can be hard due to the curse of dimensionality, with 16 input neurons and 1800 output neurons. When the dimensionality of the inputs is increased, the number of training samples needs to be increased exponentially for a nonparametric model regression [26]. The next sections present three neural network architectures for solving the problem.

4.2.1. The single layer feedforward neural network

Feedforward neural networks are the most widespread tool for solving regression problems. We propose a single layer network with 16 inputs units corresponding to the morphological parameters ($\Gamma^{(m)}$), 1800 units in the output layer related to the internal model parameters (θ) and a variable number of units in the hidden layer $\{500, 1000, 2000\}$. The neurons in the hidden layer use the hyperbolic tangent as nonlinear function whereas the output layer is linear (Figure 6). The algorithm to train the network is *scaled conjugate gradient backprogation* (SCG) [27].

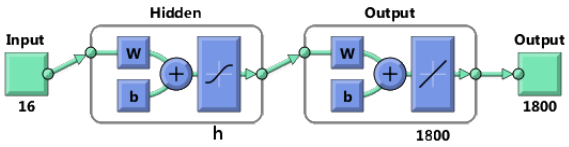

Figure 6: Schema of the single layer feedforward neural network proposed for learning the relation between the weights of the adaptive controller and the morphological parameters. The value h is the number of units in the hidden layer

4.2.2. The deep neural network

We propose a deep neural network architecture combining two stacks of autoencoders. An autoencoder (AE) is a simple learning circuit which aims to transform inputs into outputs with the least possible amount of distortion [28]. The autoencoder neural network is an unsupervised learning algorithm which tries to learn an approximation of the identity function subjected to several constraints, such as limiting the number of the hidden units. An autoencoder has two parts, a decoder and an encoder. The output of the encoder is a representation of the input, whereas the output of the decoder is the input reconstruction from the encoder representation [29]. The autoencoders have been tested and compared with other classical methods, such as principal component analysis (PCA), to reduce the data dimensionality [30].

Depending on the number of hidden units of the autoencoder in relation with the number of neurons in the input layer, two kind of the autoencoders can be considered. In the *contractive autoencoders* (CAE), the number of neurons in the hidden layer is smaller than the number of neurons in the input layer; therefore, the autoencoder is forced to learn a short representation of the input. In turn, when the input layer has less neurons than the hidden layer and a sparsity constraint is imposed, the autoencoder will still discover an interesting structure in the data [31][32]. This kind of autoencoder is called sparse autoencoder (SAE).

A stack of autoencoders is built by chaining the hidden layer activations of one autoencoder as inputs for the next one [33]. In this way, the autoencoders generate a hierarchical stack. One of these stacked of autoencoders is composed by CAEs and the other one is built using SAEs. Both are linked with each other using two single feedforward neural networks (Figure 7). The underlying idea to propose this architecture is to take advantage of the properties of both classes of autoencoders for compensating the huge difference between the inputs and outputs. Therefore, the morphological parameters ($\Gamma^{(m)}$) are the inputs of the sparse autoencoder stack. Since these autoencoders expand the information provided by the morphological parameters, the condition ($u'_2 > u'_0$) is satisfied. In turn, the adaptive controller parameters (θ) are the inputs of the stack of contractive autoencoders. In this case, the compression of the information is intended, therefore $u_2 < u_0$ (see Fig. 7).

We have tested many combinations of these network architectures. One of their advantages is the possibility to train each level of the network separately. However, our conclusion is that if the number of layers is large for the contractive autoencoders, the hidden layer activations are saturated whenever their size is smaller. In turn, the activation of the hidden layers of the stack of sparse autoencoders tend to zero when the number of layers is greater. To address this problem, the best configuration for the network layout is to use two contractive autoencoders and two sparse autoencoders. The activation function of the hidden layer for all autoencoders is the logistic function and the output function is linear.

The last activation layer of the stack of sparse autoencoders is the input of a single feedforward neural network with u'_L units in its hidden layer. The output of this neural network is the last layer of the stack of contractive autoencoders. The inverse single feedforward neural network can be trained too. These single networks have a hyperbolic tangent as activation function. Both the autoencoders and the single neural networks use the SCG algorithm for training.

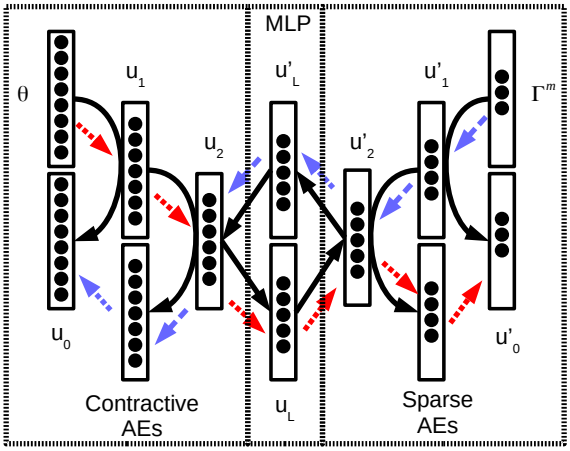

Figure 7: The proposed deep neural network architecture. It is composed of two stacks of autoencoders connected by two feedforward neural networks. The parameters $\{\{u_0, u_1, u_2\}, \{u'_0, u'_1, u'_2\}, \{u_L, u'_L\}\}$ are the number of neurons in each layer.

4.2.3. The parallel feedforward neural network

Thus far, the properties of $\hat{\Gamma}^{(p)}$ described in section 2.1 have not been considered. As stated in equation (7), $\hat{\Gamma}^{(p)}$ tends to a multivariate normal distribution with a diagonal matrix as variance. Therefore, each component of $\hat{\Gamma}^{(p)}$ can be viewed as an independent, normally distributed variable, and this applies also to the components of $\hat{\theta} = \{\hat{\gamma}_1, \hat{\gamma}_2, \cdots \hat{\gamma}_k\}$. At this point, the problem of discovering the relationship between $\Gamma^{(m)}$ and $\Gamma^{(p)}$ is decomposed into many small problems which have easier solutions. Formally, equation (12) is decomposed into many simpler equations:

$$\begin{aligned}
&\hat{\gamma}_1 = e_1(\Gamma^{(m)}, \omega_1) + \xi_1; e_1(\Gamma^{(m)}, \omega_1) = \Psi_1(\Gamma^{(m)})^T \omega_1 \\
&\hat{\gamma}_2 = e_2(\Gamma_i^{(m)}, \omega_2) + \xi_2; e_2(\Gamma^{(m)}, \omega_2) = \Psi_2(\Gamma^{(m)})^T \omega_2 \\
&\qquad \cdots\cdots\cdots\cdots\cdots\cdots \\
&\hat{\gamma}_k = e_k(\Gamma^{(m)}, \omega_k) + \xi_k; e_k(\Gamma^{(m)}, \omega_k) = \Psi_k(\Gamma^{(m)})^T \omega_k
\end{aligned} \tag{13}$$

In this particular case, for each trained robotic system i, a θ_i has been obtained, therefore $\hat{\gamma}_{i,t}$ is an element of the matrix θ_i where $i \in [1, 44271], t \in [1, 1800]$. Figure 8 shows the proposed decomposition in a graphical way. Each of these equations is dealt with by using a single layer feedforward neural network. Now, 1800 neural networks with 16 inputs and one output can be used, instead of one network with 16 inputs and 1800 outputs. These neural networks have to be trained in parallel since they share the same input but have different outputs. In this case, for all networks the activation

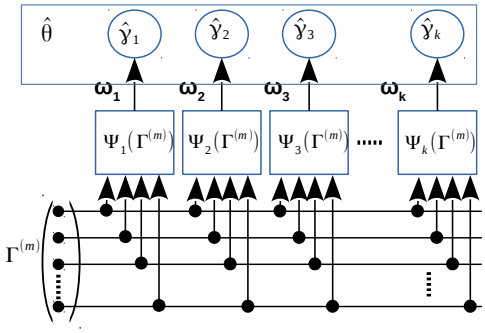

Figure 8: Proposed regression problem decomposition into many straightforward regressions.

function in the hidden layer is a hyperbolic tangent and the unique output corresponds to a weight of the adaptive controller. The input layer is shared by all networks and has 16 units. The architecture of each neural network corresponds to the one shown in figure (6) but with only one neuron in the output layer.

5. Experimental results

Once the case study was defined and the dataset was created, the three proposed approaches, based on different neural networks, were tested. The purpose is to evaluate their performance as methods for learning the relationship between the internal model and the morphological parameters. The selection of the best neural network model to solve this learning problem can be treated as a model selection problem, noting that the proposed architectures are really different. Two strategies have been applied to the three approaches. The first strategy is based on using a sequential test. As we already mentioned, we used a huge amount of morphologies for training, in order to be in a data-rich situation. Therefore, the best approach to model selection according to Hastie et al. [34] is to divide the dataset in three parts: a training set (26500 head setups), a validation set (4500 head setups) and a test set (13271 head setups). This last set is used for assessment of the generalization error of the final chosen model. This procedure has been repeated three times. To compare the different models, the average of the mean square error (MSE) of each trial for each neural network is estimated.

The second strategy is based on information criteria. Whereas the network complexity is not explicitly considered in the case of sequential tests, the information criteria addresses the model complexity by means of generalized degrees of freedom (GDF) [35]. This second strategy is usually applied to compare the different approaches, only when the final network architecture for each approach has been decided.

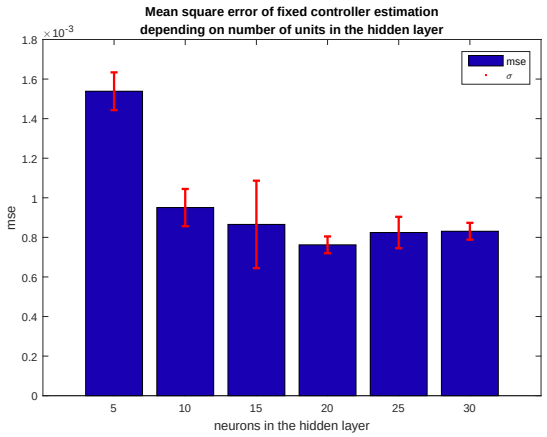

Figure 9: Mean square error obtained by different neural network hidden layer setups. The red line represents the standard deviation for three repetitions of the training

5.1. Learning the correlation between morphological and internal model parameters

5.1.1. Learning the fixed controller

In section 4.1 we defined the neural network for estimating the fixed controller. The main parameter for tunning is the number of neurons in the hidden layer. Several variations of the same neural network schema have been trained by changing the number of hidden layer units. The optimization algorithm for training is SCG and the hidden units have hyperbolic tangent as activation function. The training process and its posterior test have been repeated three times. Figure (9) shows the obtained results for the mean and standard deviation of the mean square error for **B** estimation after training. From these results, the best performance for this dataset is reached using 20 neurons in the hidden layer. Therefore, the best training performance for the estimation of **B** is: $MSE = (0.763 \pm 0.043)10^{-3}$).

This selected neural network solve the regression problem defined by equation (11).

5.1.2. Learning the adaptive controller using a single layer feedforward neural network

In section 4.2.1, the proposed neural network architecture has been described. For tunning its performance, different number of units in the hidden layer were tested with the generated dataset. In particular, we considered

500, 1000 and 2000 units in the hidden layer. These neural networks were trained three times with the validation and training datasets. Once this procedure is finished, they were tested using the test dataset for predicting $\hat{\theta}$ and solving the regression problem defined by (12). The obtained results are shown in Figure 10. The best performance is reached for a network configuration with 500 neurons in the hidden layer. In this case, the value of MSE is $(1.064 \pm 0.034)10^{-3}$.

5.1.3. *Learning the adaptive controller using a deep neural network*

We tested a large number of network setups and training parameters in order to come up with the proper architecture for the network described in section 4.2.2. Finally, we fixed the number of layers and the regularization parameters for defining the sparse and contractive autoencoders of each layer. In this way, three networks setups were defined ($Mirror_1$, $Mirror_2$ and $Mirror_3$). Their parameters are summarized in Tables 3 and 4. After training, the neural networks were evaluated using the test dataset. The best performance is achieved by $Mirror_2$ (Fig.10). The obtained MSE value for this neural network setup is $(0.484 \pm 0.034)10^{-3}$.

Table 3: Parameters used for training each autoencoder. These are fixed for the three network setups.

Type	CAE	CAE	SAE	SAE
L2. reg	10^{-6}	10^{-2}	10^{-5}	10^{-5}
Sparsity. reg	10^{-6}	10^{-4}	10^{-5}	10^{-5}
Sparsity.prop	10^{-2}	10^{-2}	10^{-1}	10^{-1}

Table 4: Distribution of the neurons in the different hidden layers of the autoencoder stacks and the feedforward neural networks (see Fig. 7). The layers u_0 and u'_0 are common to every architectures and they have 1800 and 16 neurons respectively.

	Neurons					
Layer	u_1	u_2	u_L	u_L	u'_2	u'_1
$Mirror_1$	600	300	100	100	300	50
$Mirror_2$	600	150	100	100	150	60
$Mirror_3$	900	300	100	100	300	100

5.1.4. *Learning the adaptive controller using parallel networks*

One single neural network is generated for each output and they are trained sequentially. Three different numbers of neurons in their hidden layers have been used: 5, 10 and 20. After training, these parallel networks are tested using the training dataset for estimating the mean square error. The achieved results are summarized in figure (10). The best result is obtained for 20 units in each hidden layer. The mean square error after three training and testing procedures is $(0.226 \pm 0.093)10^{-3}$.

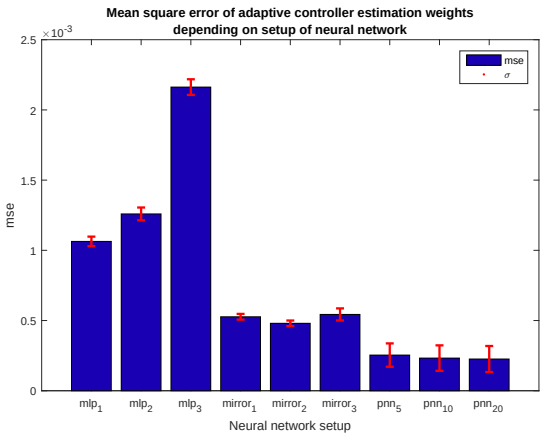

Figure 10: Mean square error obtained for different neural network setups. The red line represents the standard deviation over the three repetitions of the training.

	mlp_1	$mirror_2$	pnn_{20}
MSE·10^{-3}	1.064 ± 0.034	0.484 ± 0.020	0.226 ± 0.093
R^2	0.222	0.643	0.866
AIC	1.256	0.843	0.517
W_{AIC}	0.272	0.335	0.393
GDF	2,736	3,184	3,525
P	910,300	2,377,421	649,800

Table 5: Different performance indicators for model selection are shown with the purpose of comparing the network architectures. (AIC) Akaike's Information Criterion [36] and R^2 [37]. The values of GDF have been computed according to [35]. P is the number of weights in each architecture. The value of W_{AIC} represents the posterior probability of each model according to [38].

5.2. *Testing the result of predictions*

The various performace indicators obtained for each method served to compare the different methods. The aim now is to check whether the proposed methods are actually able to learn the relationship between the internal model and the morphological parameters. To achieve this goal, 500 head setups were selected from the test dataset. Each one of these robot heads is characterized by its morphology, defined by $\Gamma^{(m)}$. Using the proposed machine learning methods, equations (11) and

(12) are modeled to predict the internal model parameters for each head setup $\{\mathbf{B}_i, \theta_i\}$. For estimating the fixed controller of these 500 head setups, the network configuration with the best performance, as described in section 5.1.1 (20 neurons in the hidden layer), predicts $\hat{\mathbf{B}}_i$ from $\Gamma_i^{(m)}$ for each head. In turn, the initial adaptive controller parameters are estimated for different cases:*(i)* Taking their initial values randomly ($\hat{\theta}_o$ *randomly*). *(ii)* Using previously trained weights (with 1000 iterations) ($\hat{\theta}_o$ *trained*). *(iii)* Employing the three described neural networks in previous sections with the best performance for initializing the weights of the adaptive controller (mlp_1, $Mirror_2$ and pnn_{20}). In addition to the weights to run the updating process according to [25], the covariance matrix ($\mathbf{A}_t = \mathbf{R}_t^T\mathbf{R}_t$) and the transformation of the input vectors ($\mathbf{b}_t$) are needed. We consider that the initial matrix ($\mathbf{R}_0$) used for training all the adaptive controllers represents the maximum covariance of the system. From this, $\mathbf{b}_0$ is computed as: $\mathbf{b}_0 = \mathbf{R}_0^T\mathbf{R}_0\hat{\theta}$. Each of these 500 robot heads undergo a training process using the estimated $\{\hat{\mathbf{B}}_i, \hat{\theta}_{0_i}\}$ for the different five cases. The mean visual error with respect to the number of iterations is shown in Figure 11. The standard deviation of the visual error for the training process is depicted in Figure 12.

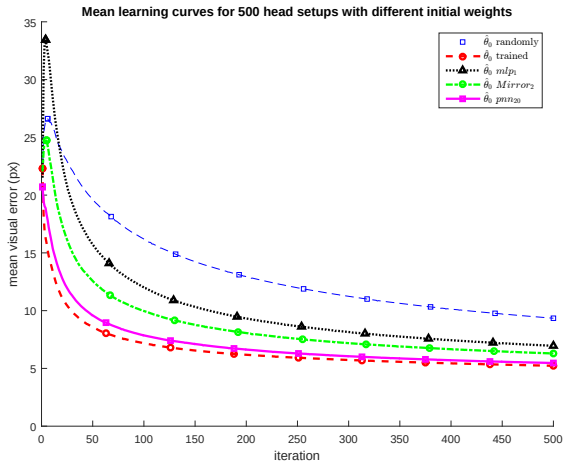

Figure 11: Mean of the visual error for 500 robot head setups during the training process using the estimations of the fixed controller and the initialization of the adaptive controller from the morphological parameters for different machine learning procedures

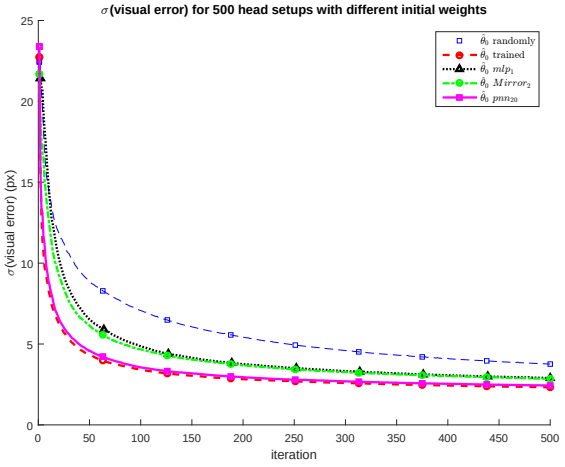

Figure 12: Standard deviation of the visual error for 500 robot head setups during the training process using the estimations of the fixed controller and the initialization of the adaptive controller from the morphological parameters for different machine learning procedures

6. Discussion

6.1. Neural network selection

Different performance indicators for model selection are shown in Table 5, with the purpose of comparing the network architectures. From that table and Figure 10, we can conclude that the best option for learning the relationship defined by equation (12) is the parallel neural networks (pnn_{20}) with 20 neurons in each hidden layer. Even though the deep network solution ($Mirror_2$) is close to these results, it does not reach the same precision. The worst behavior is that of the single layer feedforward neural network (mlp_1).

Comparing the three methods, and taking the best performance as base, the MSE for $Mirror_2$ is around 2.4 times the value of pnn_{20}, and the MSE for mlp_1 is about 4.7 times the value of the best performance. mlp_1 is clearly affected by the curse of dimensionality due to the high dimension of the output.

The additional performance indicators for model selection based on information criteria, that are shown in Table 5, support the above result to the effect that pnn_{20} is the preferred option. Indeed, AIC (Akaike's Information Criterion for model selection) [34], yields the smallest value for pnn_{20}. w_{AIC} (AIC-weights) is a normalized AIC, and represents the *a posteriori* likelihood of a model given the set of the three network models. The highest likelihood corresponds to (pnn_{20}). R^2 backs this up too (even though it is more reliable for linear regression). GDF (generalized degrees of freedom) for each model has been computed beforehand to estimate the AIC indexes. In nonlinear regression problems, GDF (equivalent to the number of parameters in linear regression) can be used as a measure of the complexity of a general modeling procedure [35]. Also shown in table 5, the number of weights for each model

(**P**) is appreciably higher than GDF, being the lowest value for (pnn_{20}) and the highest value for ($mirror_2$). However, the value of GDF is higher for pnn_{20} than it is for $mirror_2$; this suggests a better exploitation of the network weights in the case of pnn_{20}

The key to understanding this behavior is to analyze the traits of θ, i.e. the outputs of the proposed neural network models. θ are the final weights of an on-line neural network that estimates the adaptive controller for the robot head (C_f). When the robot starts to adapt, it is really learning the model that is defined by the environment. This knowledge is partially stored in the weights of the on-line neural network in such a way that their values barely change once the model has been learned. These weights correspond to the maximum likelihood estimator . When C_f is regarded as learned, equation (7) is accomplished, and therefore each weight component is probabilistically independent of the rest of them. This is a particularity that only applies to the parallel network model. For the other two network models the weights are statistically related.

6.2. Learning improvement for new robot morphologies

The experiment described in section 5.2 evaluates to what extent the different proposed neural networks improve the performance of the training process. Using equations (11) and (12), a fixed and an adaptive controller are predicted for a particular morphology, and then the adaptation process continues. In this way, if the prediction were perfect, the adaptation process should continue at that point. In Figure 11 the visual error is used as a measure of the real performance of the system.

The curve (θ_0 *randomly*) represents the learning curve when there is no previous information about the system. Additionally, the (θ_0 *ftrained*) curve represents that the networks starts the adaptation process initializing the adaptive controller with previously trained weights. In this case, the visual error is stabilized after just a few iterations. This short updating period of the visual error is due to the I-SSGPR algorithm that needs to update other parameters beside the weights (e.g. the system covariance matrix), which are initialized to start the training process with the same value for all setups. The curve (θ_0 *trained*) represents that previous information about the system is known and it is virtually as if it would continue the training at the point where it was. The more similar a training curve is to (θ_0 *trained*), the better its performance is.

The learning curves for the three proposed methods lie between (θ_0 *randomly*) and (θ_0 *trained*). Their relative performances correspond to their mean square error as described in section 5.1. As expected, (θ_0 pnn_{20}) is the most similar to (θ_0 *trained*), whereas the curve for mlp_1 yields the worst visual error after the untrained network case, and (θ_0 $Mirror_2$) is an intermediate case. This order is confirmed when the standard deviation is considered. In the updating process, the standard deviation decreases until a constant value. Figure 12 illustrates how these variations are grouped; indeed, for (θ_0 *trained*) it is very close to (θ_0 pnn_{20}), and similarly for (θ_0 $Mirror_2$) and (θ_0 mlp_1). That confirms that the behavior of the training curves using the values predicted by pnn_{20} is very similar to (θ_0 *trained*), that uses the trained weights.

7. Conclusions

A framework for predicting the internal model of a robotic system from its morphology has been presented. We show how, given the morphological parameters of the system, its internal model can be estimated, as suggested by equation (1), and following the hypothesis that the internal model is an approximation of the environment model. Moreover, if the internal model parameters are learned as a MLE, each one of these parameters is statistically independent (equation (7)).

Learning the internal model of a robotic system from its morphology can be a high-dimensional regression problem. This was indeed the case for the particular real example that we chose to verify how the modeled relationship can be learned in a real problem: a pan-tilt robot head executing saccadic movements. To address this problem, the internal model parameters were decomposed in two sets. The first one is a set of inter-dependent parameters for the proposed fixed controller. The estimation for this set was done using a single layer feedforward neural network. The second one is an independent set of parameters corresponding to the weights of the adaptive controller which was implemented using an on-line neural network with the I-SSGPR algorithm for its adaptation. In this second case, three different neural network architectures were proposed. Then, the best model was selected according with different performance indicators, such as MSE and indexes for model selection based on information criteria. Also, an experimental evaluation was conducted using the predictions of the proposed models to improve the online-training process for a new robotic system. Our conclusion is that in all cases the parallel neural network had better performance than the other two with a smaller number of weights. The independence condition of the learned weights (equation(7)) has proved to be essential for selecting the proper architecture for the learning tool.

7.1. Potencial impact

We have illustrated our ideas with a robot head executing saccades, but this is just a particular example and the possible applications of our model are countless. We mentioned in section 1 *morphofunctional* machines and self-reconfigurable robots that can modify their morphology to change their functionality [7]. For them, having the knowledge to rapidly generate their updated internal models after a change in morphology will greatly enhance their performance, adaptivity and versatility.

Similarly, in the current trend towards benchmarking and reproducible research [9], getting the internal model directly from the morphology can significantly contribute to effectively implement the same solutions in robot designs that, being similar, differ in their configuration parameters.

Finally, the greatest potential can be expected in the context of Industry 4.0. and *Cloud Robotics*. *Cyber-Physical* Robotic Systems will have access to big data in the form of libraries of global datasets; cloud computing for statistical analysis and learning; as well as collective robot learning [15]. Given the large variety of existing robot designs –and variations in the parameters of similar designs– for shared knowledge in the Cloud to be fully operational, internal models should be readily available there, as pre-computed datasets or as computing service on demand, so that different robots can actually take advantage of that knowledge and exhibit a rational behavior without extensive learning. We believe that the research presented in this paper can significantly contribute to progress along these lines.

8. Acknowledgements

This paper describes research done at the UJI Robotic Intelligence Laboratory. Support for this laboratory is provided in part by Ministerio de Economa y Competitividad (DPI2015-69041-R), by Fondo Europeo de Desarrollo Regional (FEDER), by Generalitat Valenciana (PROMETEOII/2014/028) and by Universitat Jaume I (P1-1B2014-52, PREDOC/2013/06).

References

[1] J. Bongard, V. Zykov, H. Lipson, Resilient machines through continuous self-modeling, Science 314 (5802) (2006) 1118–1121.

[2] R. Vaughan, M. Zuluaga, Use your illusion: Sensorimotor self-simulation allows complex agents to plan with incomplete self-knowledge, in: S. Nolfi, G. Baldassarre, R. Calabretta, J. C. T. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, D. Parisi (Eds.), From Animals to Animats 9, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 298–309.

[3] O. Sigaud, C. Salan, V. Padois, On-line regression algorithms for learning mechanical models of robots: A survey, Robotics and Autonomous Systems 59 (12) (2011) 1115–1129. doi:10.1016/j.robot.2011.07.006.

[4] A. Isidori, L. Marconi, A. Serrani, Fundamentals of internal-model-based control theory, in: Robust Autonomous Guidance, Springer, 2003, pp. 1–58.

[5] D. Nguyen-Tuong, J. Peters, Model learning for robot control: a survey, Cognitive Processing 12 (4) (2011) 319–340. doi:10.1007/s10339-011-0404-1.

[6] B. Siciliano, et al., Robotics: Modelling, Planning and Control, Advanced Textbooks in Control and Signal Processing, Springer London, 2008.

[7] R. Pfeifer, M. Lungarella, F. Iida, Self-Organization, Embodiment, and Biologically Inspired Robotics, Science 318 (November) (2007) 1088–1093. doi:10.1126/science.1145803.

[8] D. L. Hudson, M. E. Cohen, Neural networks and artificial intelligence for biomedical engineering, 2000.

[9] F. Bonsignorio, A. P. del Pobil, Toward replicable and measurable robotics research [from the guest editors], IEEE Robotics & Automation Magazine 22 (3) (2015) 32–35.

[10] J. Felip, J. Laaksonen, A. Morales, V. Kyrki, Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks, Robotics and Autonomous Systems 61 (3) (2013) 283–296.

[11] S. K. Khaitan, J. D. McCalley, Design techniques and applications of cyberphysical systems: A survey, IEEE Systems Journal 9 (2) (2015) 350–365.

[12] A. P. del Pobil, Robots as Cyberphysical Systems: The Challenges Ahead, keynote speech at the 12th ACM International Conference on Ubiquitous Information Management and Communication, Langkawi, Malaysia, 2018.

[13] M. Hermann, T. Pentek, B. Otto, Design principles for industrie 4.0 scenarios, in: System Sciences (HICSS), 2016 49th Hawaii International Conference on, IEEE, 2016, pp. 3928–3937.

[14] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, A. V. Vasilakos, Cloud robotics: Current status and open issues, IEEE Access 4 (2016) 2797–2807.

[15] B. Kehoe, S. Patil, P. Abbeel, K. Goldberg, A survey of research on cloud robotics and automation, IEEE Transactions on automation science and engineering 12 (2) (2015) 398–409.

[16] M. Waibel, M. Beetz, J. Civera, R. d'Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, et al., Roboearth, IEEE Robotics & Automation Magazine 18 (2) (2011) 69–82.

[17] A. L'Heureux, K. Grolinger, H. F. Elyamany, M. A. M. Capretz, Machine Learning With Big Data: Challenges and Approaches, IEEE Access 5 (2017) 7776–7797. doi:10.1109/ACCESS.2017.2696365.

[18] M. Rucci, G. M. Edelman, J. Wray, Adaptation of orienting behavior: from the barn owl to a robotic system, IEEE Transactions on Robotics and Automation 15 (1) (1999) 96–110. doi:10.1109/70.744606.

[19] M. Rucci, J. Wray, G. Edelman, Robust localization of auditory and visual targets in a robotic barn owl, Robotics and Autonomous Systems 30 (1) (2000) 181 – 193. doi:https://doi.org/10.1016/S0921-8890(99)00071-8.

[20] M. Rucci, D. Bullock, F. Santini, Integrating robotics and neuroscience: Brains for robots, bodies for brains, Advanced Robotics 21 (10) (2007) 1115–1129. doi:10.1163/156855307781389428.

[21] M. Antonelli, A. J. Duran, E. Chinellato, A. P. del Pobil, Learning the visual-oculomotor transformation: Effects on saccade control and space representation, Robotics and Autonomous Systems 71 (2015) 13–22. doi:10.1016/j.robot.2014.11.018.

[22] M. Kawato, Feedback-error-learning neural network for super-

vised motor learning, in: R. Eckmiller (Ed.), Advanced Neural Computers, North-Holland, Amsterdam, 1990, pp. 365–372. doi:10.1016/B978-0-444-88400-8.50047-9.

[23] R. Saegusa, G. Metta, G. Sandini, S. Sakka, Active motor babbling for sensorimotor learning, IEEE International Conference on Robotics and Biomimetics (2008) 794–799doi:10.1109/ROBIO.2009.4913101.

[24] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Advances in neural information processing systems, 2007, pp. 1177–1184.

[25] A. Gijsberts, G. Metta, Real-time model learning using Incremental Sparse Spectrum Gaussian Process Regression., Neural networks 41 (2013) 59–69. doi:10.1016/j.neunet.2012.08.011.

[26] M. Kohler, A. Krzyzak, H. Walk, Optimal global rates of convergence for nonparametric regression with unbounded data, Journal of Statistical Planning and Inference 139 (4) (2009) 1286–1296. doi:10.1016/j.jspi.2008.07.012.

[27] M. F. Møller, A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning Supervised Learning, Neural Networks 6 (1993) 525–533. doi:10.1016/S0893-6080(05)80056-5.

[28] P. Baldi, Autoencoders, Unsupervised Learning, and Deep Architectures, ICML Unsupervised and Transfer Learning (2012) 37–50.

[29] S. Rifai, X. Muller, Contractive Auto-Encoders : Explicit Invariance During Feature Extraction, Icml 85 (1) (2011) 833–840.

[30] A. J. Holden, et al., Reducing the Dimensionality of Data with Neural Networks, Science (New York, N.Y.) 313 (July) (2006) 504–507.

[31] A. Ng, Sparse autoencoder, CS294A Lecture notes 72 (2011) 1–19.

[32] L. Meng, S. Ding, Y. Xue, Research on denoising sparse autoencoder, International Journal of Machine Learning and Cybernetics 8 (5) (2017) 1719–1729. doi:10.1007/s13042-016-0550-y.

[33] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy Layer-Wise Training of Deep Networks, Advances in Neural Information Processing Systems 19 (1) (2007) 153. arXiv:0500581, doi:citeulike-article-id:4640046.

[34] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, Springer Series in Statistics, Springer New York, 2009.

[35] J. Ye, On measuring and correcting the effects of data mining and model selection, Journal of the American Statistical Association 93 (441) (1998) 120–131. doi:10.1080/01621459.1998.10474094.

[36] U. Anders, O. Korn, Model selection in neural networks, Neural Networks 12 (2) (1999) 309–323. doi:10.1016/S0893-6080(98)00117-8.

[37] J. B. Kadane, N. A. Lazar, Methods and criteria for model selection, Journal of the American Statistical Association 99 (465) (2004) 279–290.

[38] K. Burnham, D. Anderson, Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach (2nd ed), Vol. 172, 2002. doi:10.1016/j.ecolmodel.2003.11.004.