ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

Neural Networks and Neurocontrollers
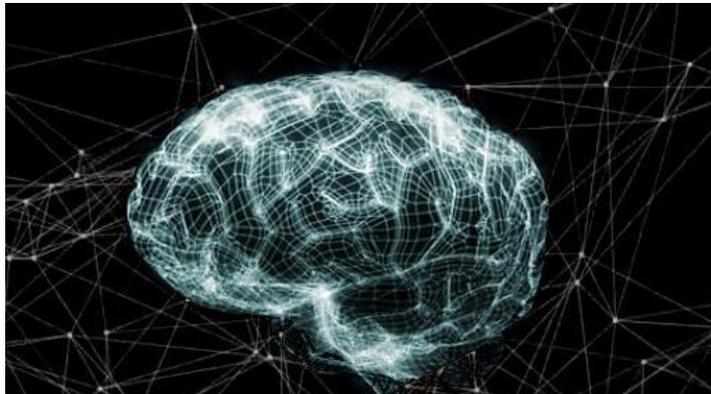
10/22/2019

Outline

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

History

- **Artificial Neural Networks** (ANNs) are an abstract simulation of the nervous system, which contains a set of neurons exchanging information through connections (*axons*)

- The ANN model try to mimic axons and dendrites of the nervous system.

- The first neural model was proposed by McCulloch and Pitts (1943). The model was presented as a computational model of the nervous activity. After this, other models were proposed John von Neumann, Marvin Minsky, Frank Rosenblatt, etc.

Two types of neuron models…

- Biological model. It has the objective of replicating biological neural systems, i.e. visual and auditive functionalities. These models are used to validate and verify hypothesis about biological systems.

- The second type is focused on the applications. The models are strongly influenced by application needs. They are called connectionist architectures.
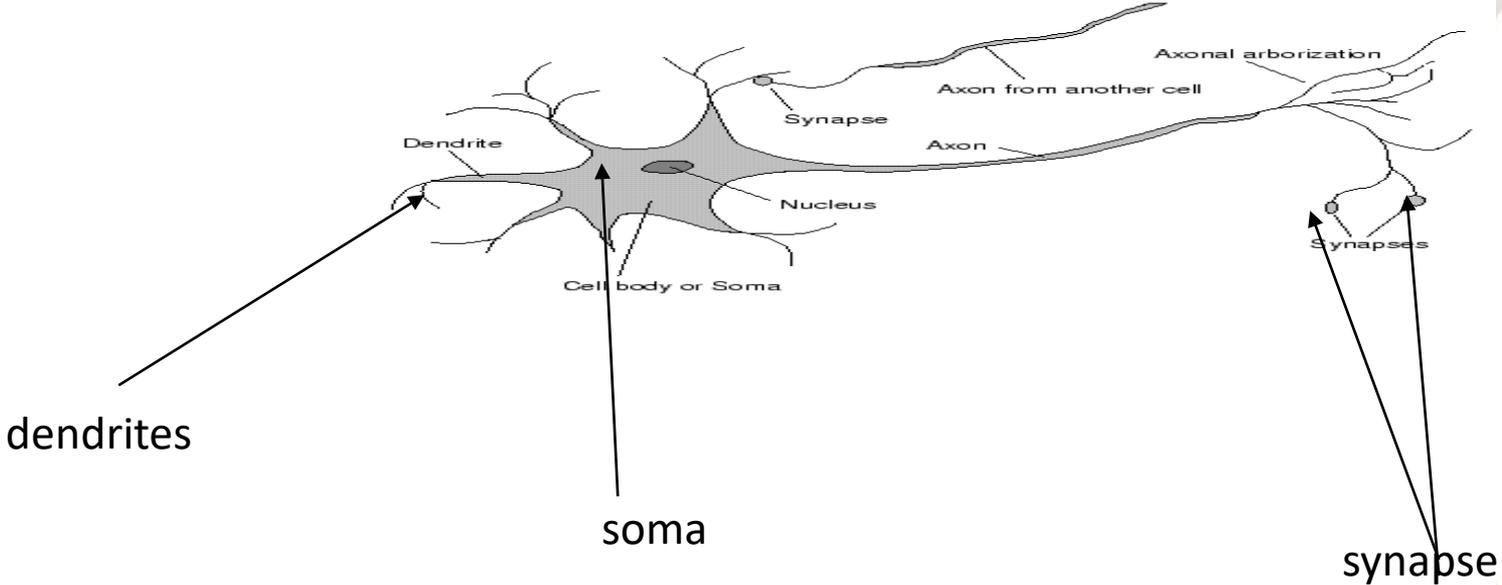
<u>We will focus on the second one!</u>

Biological neuron

dendrites

soma

synapse
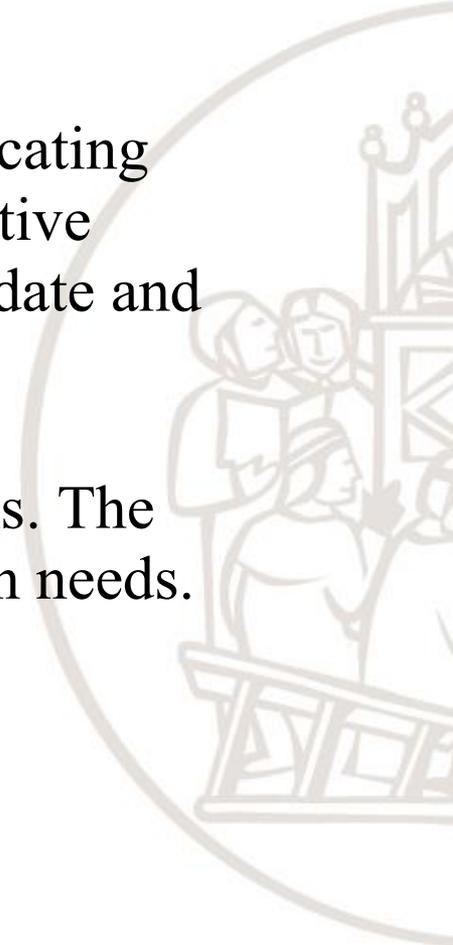
Biological neuron

- Human brain contain 100 million neurons. Neuroscientific evidences show each neuron can have 10000 sinapses in input or output

- Switching time of a neuron is few milliseconds. It is slower than a logic gate, but it has a greater connectivity

- A neuron receives from synapses information which are summed

- If the excitatory signal is leading, the neuroni s activated and it generates information through the synapse

Neural Network structure

A neural network is composed by:

- A set of nodes (neurons), which is the basic unit
- A set of weights linked to connections.
- A set of thresholds or activation levels

The network **design** requires:

1. Number of basic unit.
2. Morphological structure.
3. Learning example encoding (input and output of the net)
4. Initialization and training of the weights linked to the connections, through a learning example set.
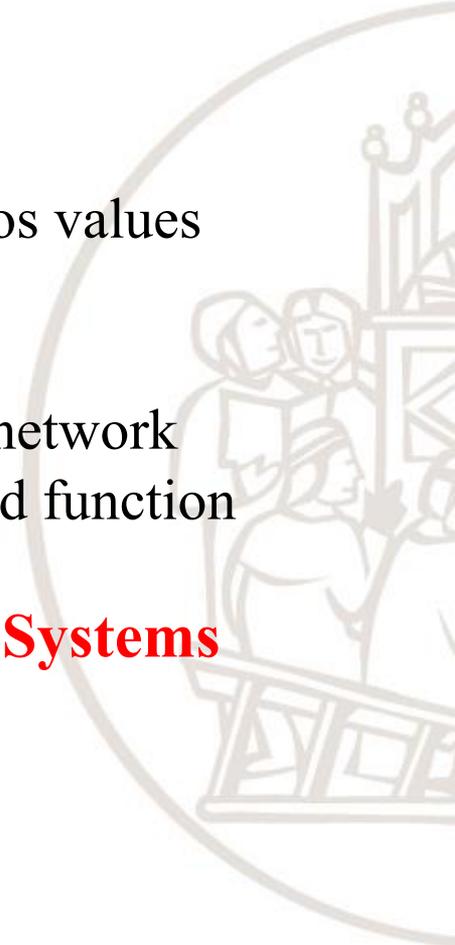
Neural network applications

Main features:

- The objective function can have discrete/continuos values
- Learning data can be noisy
- Learning time is NOT real-time
- Fast evaluation of the learning rate of the neural network
- It is not crucial to get the **semantics** of the learned function

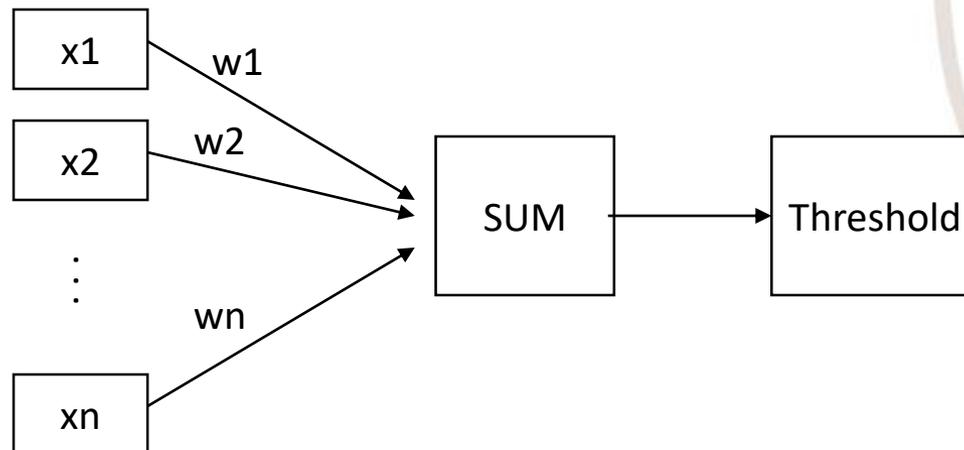**Robotics, Image Understanding, Biological Systems**

Learning strategies

- **Supervised Learning**

  -MLP and recurrent NN

- Unsupervised Learning
  - Clustering

- Competitive Learning
  - Kohonen networks

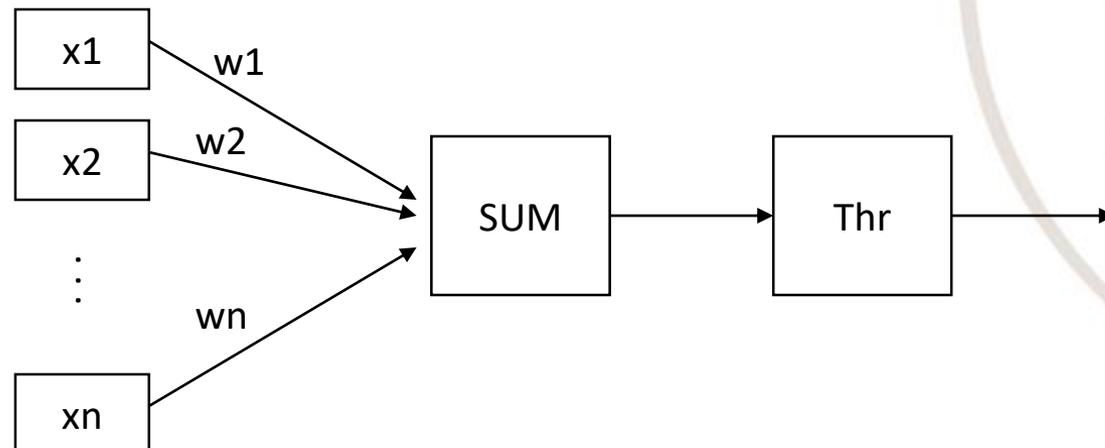- Reinforcement Learning

The perceptron

- The perceptron is the neural network basic unit
- It was defined by Rosenblatt (1962)
- Try to replicate the single neuron function

The perceptron

- Output values are boolean: 0 – 1

- Inputs x_i and weights w_i are real (positive or negative)

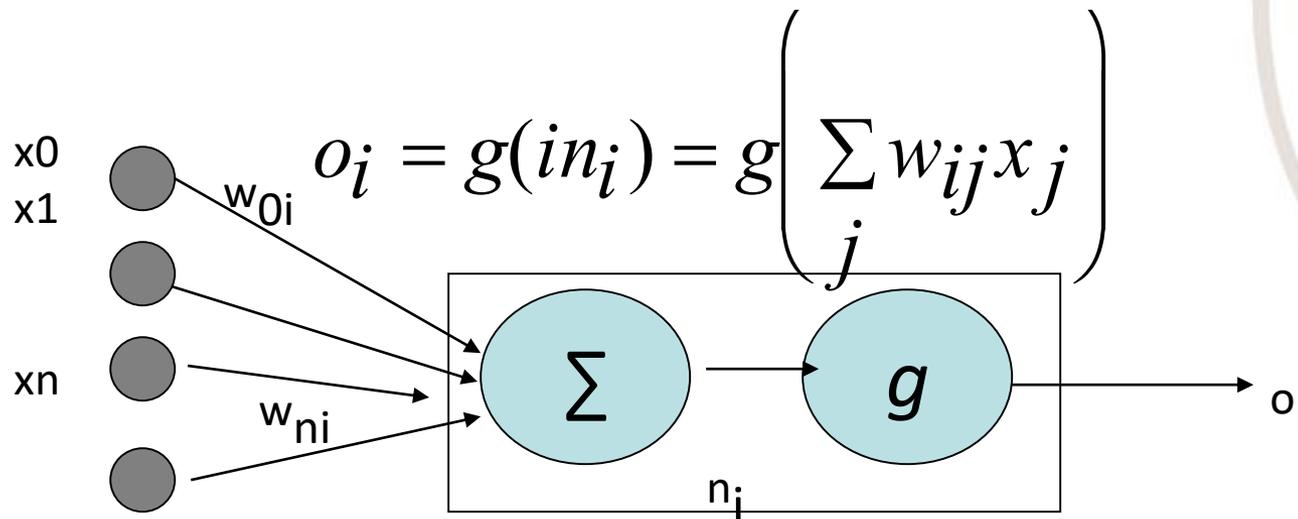- Learning consists in selecting value for weights and threshold

Sum and activation functions

a) **Input funtion**, *linear (SUM)*

$$in_i = \sum_j w_{ij} x_j = w_i x_i$$

b) **Activation function**, *non linear (THRESHOLD)*

$$o_i = g(in_i) = g\left(\sum_j w_{ij} x_j\right)$$

x0
x1

xn

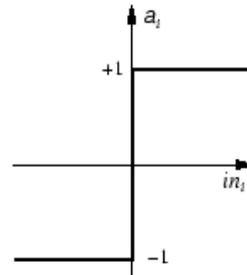w_{0i}

w_{ni}

Σ g

o_i

n_i

Activation functions
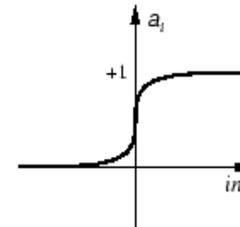
$$step_t(x) = \begin{cases} 1, \text{if } x > t \\ 0, \text{else} \end{cases}$$
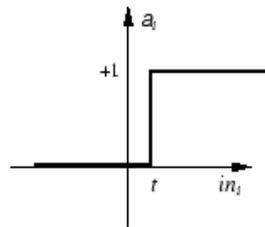
(a) Step function   (b) Sign function   (c) Sigmoid function

$$sign(x) = \begin{cases} +1, \text{if } x \geq 0 \\ -1, \text{else} \end{cases}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Objective function

- If the threshold function is *sign()* and x1..xn are the input values:

$$o(x) = 1 \ \textit{if} \ w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$$

$$o(x) = -1 \ \textit{else}$$

- Vector notation:

$$o(\vec{x}) = sign(\vec{w} \cdot \vec{x})$$

The Perceptron (classification generalization)

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

- Learning problem:
  - Set of points in a *n-dimensional* space
  1 classify into two groups (positives and negatives)
  2 Then, given a new point P, associate P with one group

  1 Classification problem
  2 generalizzation problem (learning concepts)

Perceptron – Training algorithm

- Initialize weights randomly
- Gives an example from the dataset<x,c(x)>
- Compute o(x)
- IF o(x)≠c(x) then update:
- η is the *learning rate*
- x_i is the ith feature value of x
- The perceptron error (E) is equal to (c-o)

$$w_i \leftarrow w_i + \Delta w_i$$
$$\Delta w_i = \eta(c(x) - o(x))x_i$$

Example test

- Suppose that o(x)=-1 (if the threshold function is sign(x)) and c(x)=1
- It is needed to modify weights
- Example:

$$x_i = 0, 8, \quad \eta = 0, 1, \quad c = 1, \quad o = -1$$

$$\Delta w_i = \eta(c - o)x_i = 0, 1(1 - (-1))0, 8 = 0, 16$$

- The w_i. value increases in order to reduce the error
- IF c(x)=-1 e o(x)=1

$$\Delta w_i = \eta(c - o)x_i = 0, 1(-1 - (+1))0, 8 = -0, 16$$

The perceptron

- Convergence theorem of *percepton* (Rosemblatt, 1962)
  - The perceptron is a <u>linear classifier</u>, therefore it will never get to the state with all the input vectors classified correctly if the training set is not <u>linearly separable</u>
  - In other words.. No local minima!
- The way to solve nonlinear problems is using multiple layers
- Solution: **Feed forward neural network and recurrent neural network**

Supervised learning
MLP networks

• All the neurons of a layer are connected to all the neurons of the next layer
• There are no connections between neurons in the same layer and between non adjacent layers
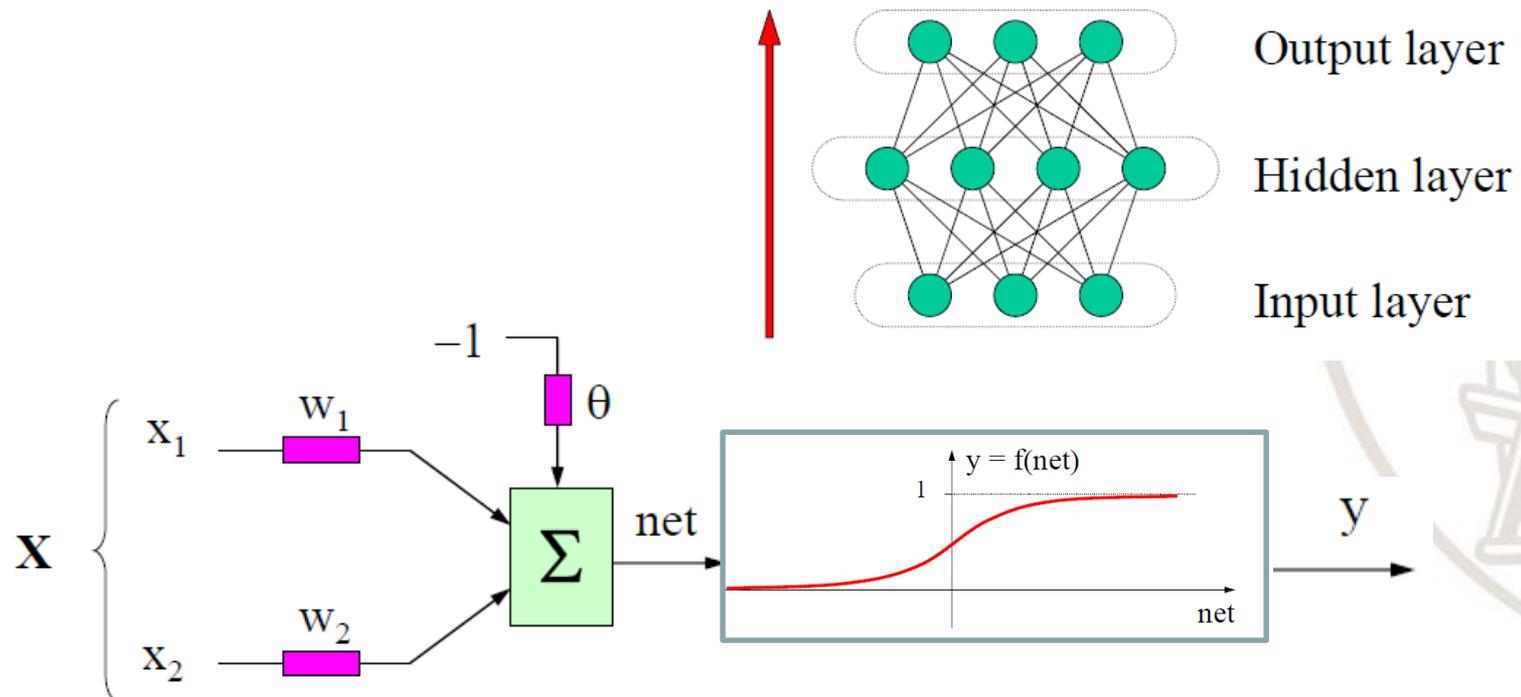
Feed forward neural network: back propagation algorithm

- Objectives:
  - Perceptron weights initialized randomly
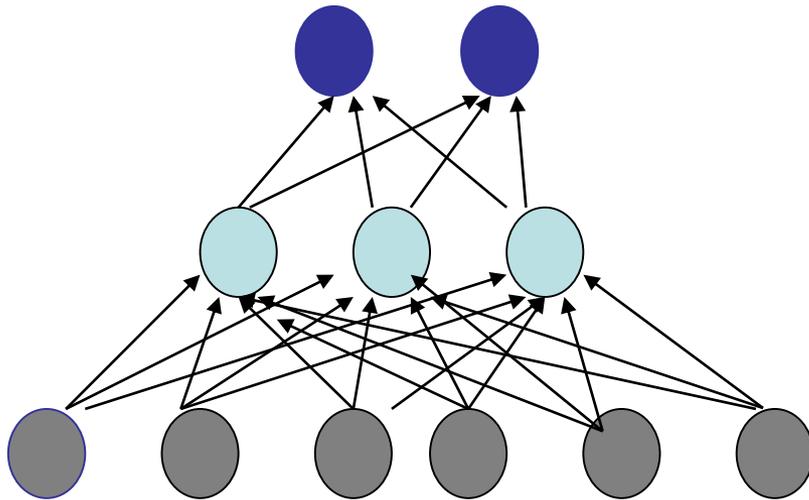 - Fast learning
  - Generalization capability

Backpropagation (2)

Threshold function: sigmoid

$$o(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) = \frac{1}{1 + e^{\vec{w} \cdot \vec{x}}}$$

Error function is as follow:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{x \in D} (t(x) - o(x))^2 =$$

$$\frac{1}{2} \sum_{x \in D} \sum_{k \in N_{out}} (t_k(x) - o_k(x))^2$$

- ⬤ I$_i$ Input units
- ⬤ H$_j$ Hidden units
- ⬤ O$_k$ Output units

Backpropagation (3)

Gola: **minimize error** between expected and real output

Update weights rule:

$$wji \leftarrow wji + \Delta wji$$

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

$$\delta_j = o_j(1-o_j)(t_j - o_j)$$

Weights w_{ij} (from n_j to n_i)
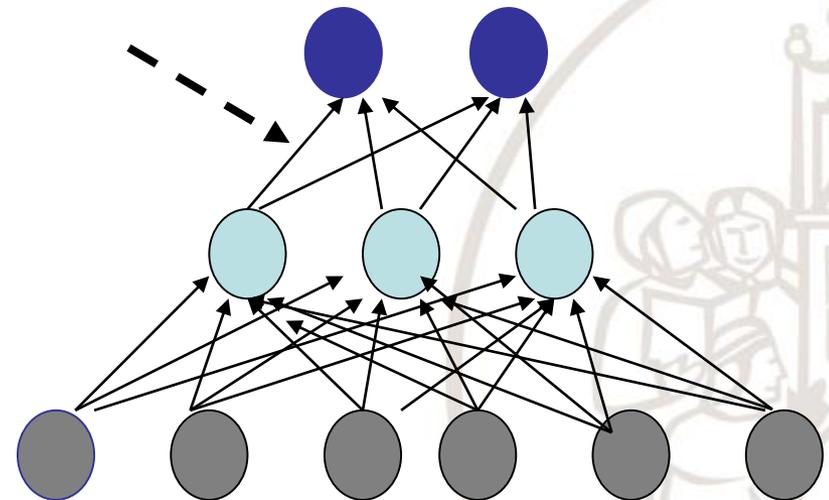
NNs produce **m** output values

Backpropagation (4)

- **<u>While</u>** unreached termination condition, **<u>execute</u>**:
- For each value $v \in D$: $(x, t(x))$ $(x=(x_1,x_2..x_n), t(x)=(t_1,t_2,..t_m)$:
  - I set of the input nodes $(1,2,,n)$, O set of output nodes $(1,2..m)$, N set of the net nodes
  - Compute the output of the net generated by the input v and the output of each node of the net $n_u \in N$ (x_i input of the input node $i_i \in I$, o_j output yielded by the node $n_j \in N$)
  - Compute error of the output node $o_k \in O$ as follows:
  $$\delta_k = o_k (1 - o_k)(t_k - o_k)$$

  - Compute the error for the hidden units $h_h \in H = (N-O \cup I)$ connected the the output nodes, as follows: $\delta_h = o_h (1 - o_h) \sum_{n_k \in O} w_{kh} \delta_k$

  - Compute the error for the other nodes
  - Updates the net weights as follow:

  $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

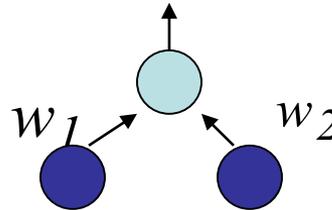 $$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Gradient computation

$$\Delta w_1 = -\eta \frac{\partial E(w_1 x_1 + w_2 x_2)}{\partial w_1} = -\eta \frac{\partial E}{\partial net_1} \frac{\partial net_1}{\partial w_1} = -\eta \frac{\partial E}{\partial net_1} x_1$$

$$net_1 = w_1 x_1 + w_2 x_2$$

$$E = \frac{1}{2}(t-o)^2$$

$$\frac{\partial E}{\partial net_1} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial net_1} \qquad \frac{\partial E}{\partial o} = \frac{1}{2} 2(t-o) \frac{\partial (t-o)}{\partial o} = -(t-o)$$

$$\frac{\partial o}{\partial net_1} = \frac{\partial \sigma(net_1)}{\partial net_1} = o(1-o)$$

What we used in the BP algorithm!!

$$\partial(\sigma(x)) = \sigma(x)(1-\sigma(x))$$

$$\Delta w_1 = \eta o(1-o)(t-o)x_1$$

Termination condition

- The process continues until all the examples (<x,t(x)>) have been processed
- When does the process stop? Minimize errors on the train set is not the best solution (*overfitting*)
- Minimize errors on a test set (T), this means to split D in D'∪T, training using D' and using T to verify the termination condition

Error in the training set
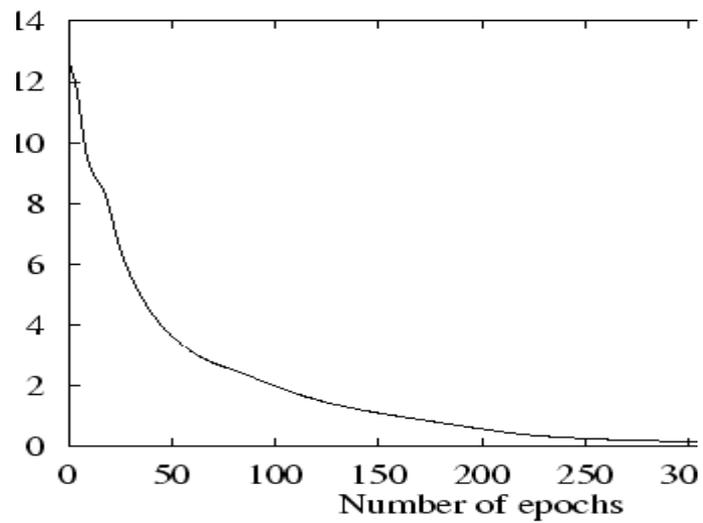
Does the algorithm converge?

ISTITUTO DI BIOROBOTICA

Scuola Superiore Sant'Anna

- Gradient algorithm issues:
  - Can stop on local minima
  - A local minimum can give solutions which are far from the global minimum
  - Sometimes there are a lot of local minima…
- A possible solution: training with changing initial weight values

Weights Updating Rule

- Considering the n-th valueof di D, updating rule becomes:

$$\Delta wij(n) \leftarrow \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1) \qquad \text{Momentum}$$

- Pros:
  - Overcoming local minima
  - Keeping stable value for the weights in the «flat zones»
  - Increase velocity when gradient does not change
- Cons:
  - If momentum value is too high can stop on local MAXIMA
  - One more tuning value

A few considerations…

- Initializing weight values is basic to reach convergence

- BP depends on the learning rate η. This can make the net diverging.

- It can be useful to use different values of η for the network layers

Recurrent neural networks

They are networks that learn to associate an input pattern with a sequence of output patterns

$$Xk \Rightarrow Yk1, Yk2, ..., YkL$$
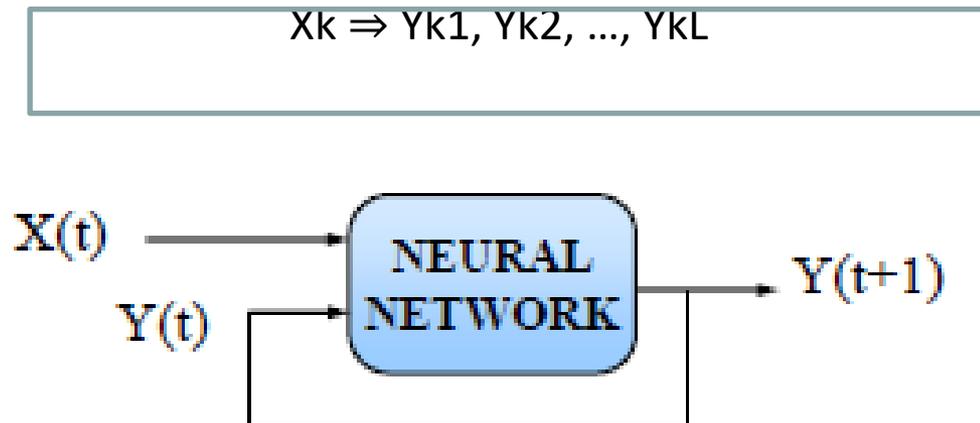
A recurrent neural network (RNN) is a class of neural networks where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior.

Learning strategies

- Supervised Learning
  - MLP and Recurrent NN

  Unsupervised Learning

  - Clustering
- Competitive Learning
  - Kohonen networks
- Reinforcement Learning

Unsupervised learning

- Split non labeled input values in subsets(**cluster**)
  - Similar input values are in the same subset
  - Different input values are in different subsets
- Find new in an subsets in an unsupervised way (no labels provided)

Learning strategies

- Supervised Learning
  - MLP e reti neurali ricorrenti
  - RBF
- Unsupervised Learning
  - Clustering
- Competitive Learning
  - Reti di Kohonen
- Reinforcement Learning

Competitive Learning

- In some cases, the network output can be ambiguous
- Thanks to the **lateral inhibition**, neurons start competing to respond to a stimulus.
- The neuron having the greatest output wins the competition and specializes itself to recognize that stimulus.
- Thanks to the **excitatory connections**, neurons near the winner are also sensitive to similar inputs

An isomorphism is created between input and output space

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

- The winning neuron is selected using a global strategy just by comparing the outputs of the other neurons.
- Two techniques can be used:
  1. Select the neuron with the maximum output;
  2. Select the neuron whose weight vector is more similar to the

$$y_j = \sum_{i=1}^{n} w_{ji}x_i = W_j \bullet X = |W_j||X|\cos\theta$$

$$j_0 : \text{DIS}(W_{j0}, X) < \text{DIS}(W_j, X) \quad \forall j \neq j_0$$
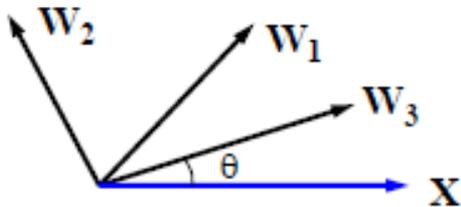
METHOD 1 - The winner on an input X is the one with the greatest output
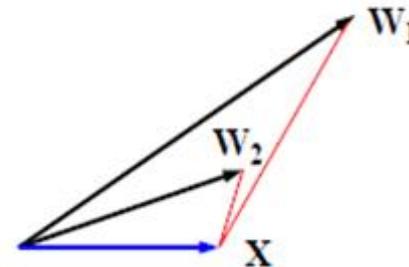
METHOD 2 -The winner neuron on input X is that having its weight vector more similar to X

Kohonen networks

The Kohonen network (or "self-organizing map", or SOM, for short) has been developed by Teuvo Kohonen.
The basic idea behind the Kohonen network is to setup a structure of interconnected processing units (neurons) which *compete* for the signal.
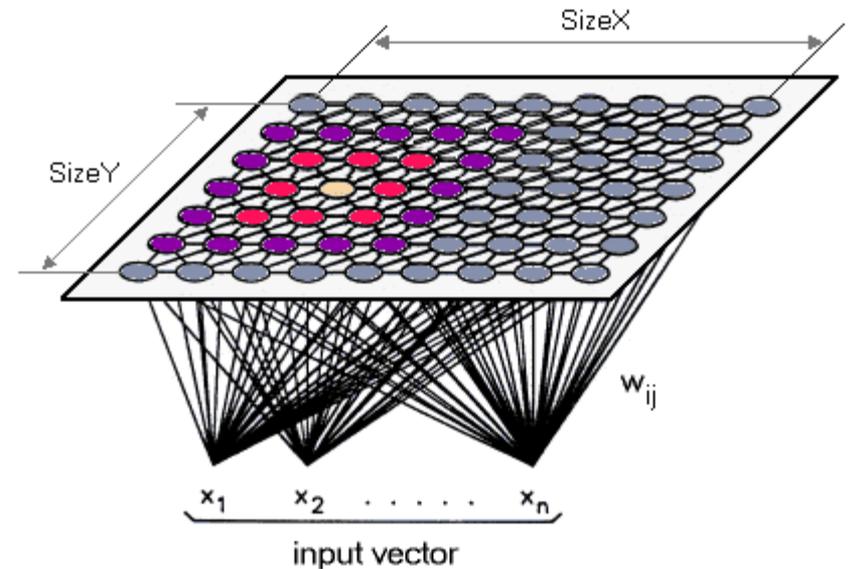
The SOM defines a **mapping** from the input data space spanned by $x_1..x_n$ onto a one- or two-dimensional array of nodes. The mapping is performed in a way that the topological relationships in the n-dimensional input space are maintained when mapped to the SOM. In addition, the local density of data is also reflected by the map: areas of the input data space which are represented by more data are mapped to a larger area of the SOM.

Learning strategies

- ## Supervised Learning
  -MLP e reti neurali ricorrenti
  - RBF
- ## Unsupervised Learning
  - Clustering
- ## Competitive Learning
  - Reti di Kohonen
- ## Reinforcement Learning

Reinforcement learning

Several actions are executed.
Successful actions are stored
(by weight variations).
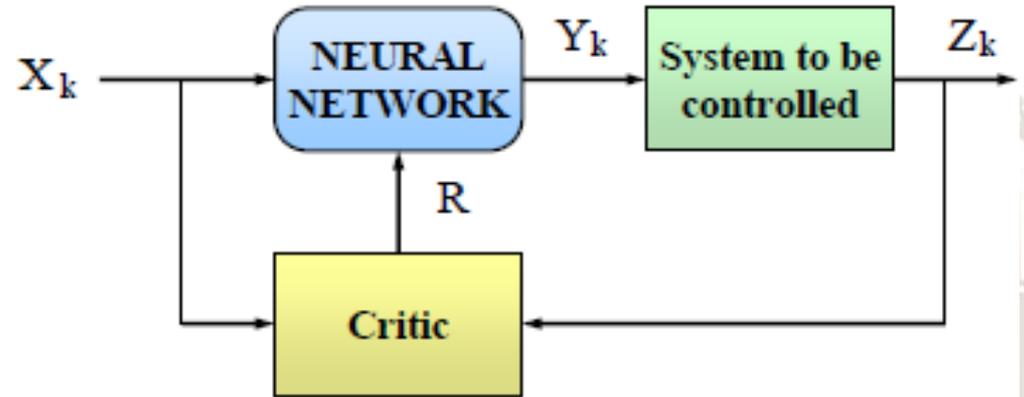
Punishments and rewards

An agent operates in the
environment and modify
actions based on the produced
consequences.
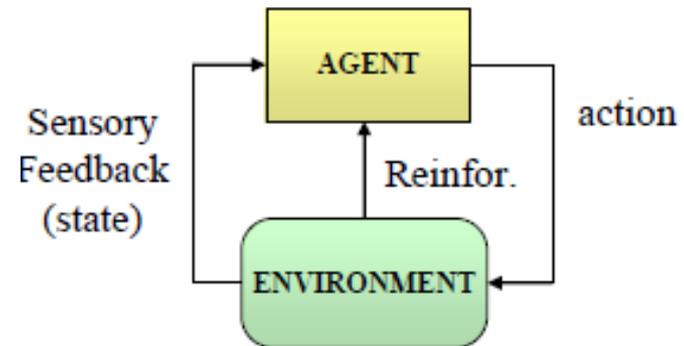
Robot control and neurocontrollers

Robot control
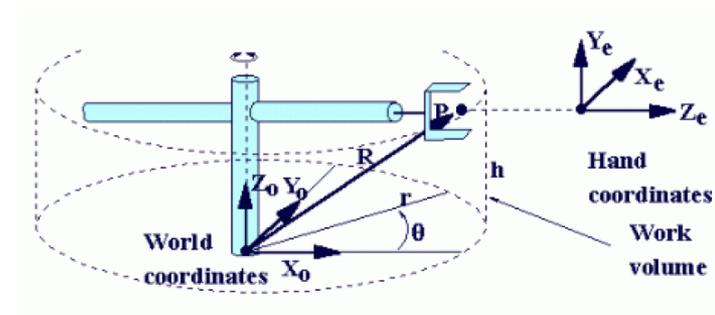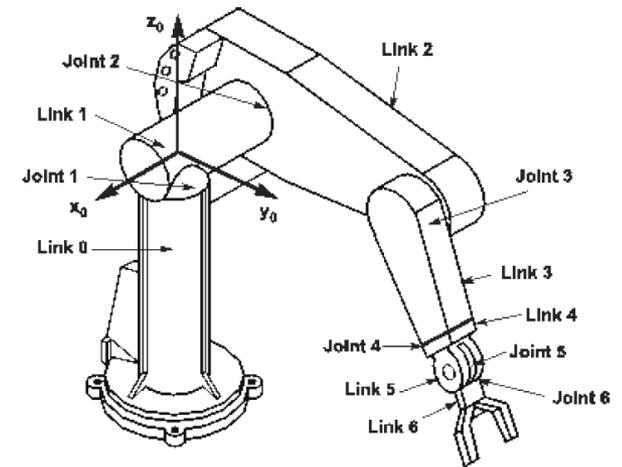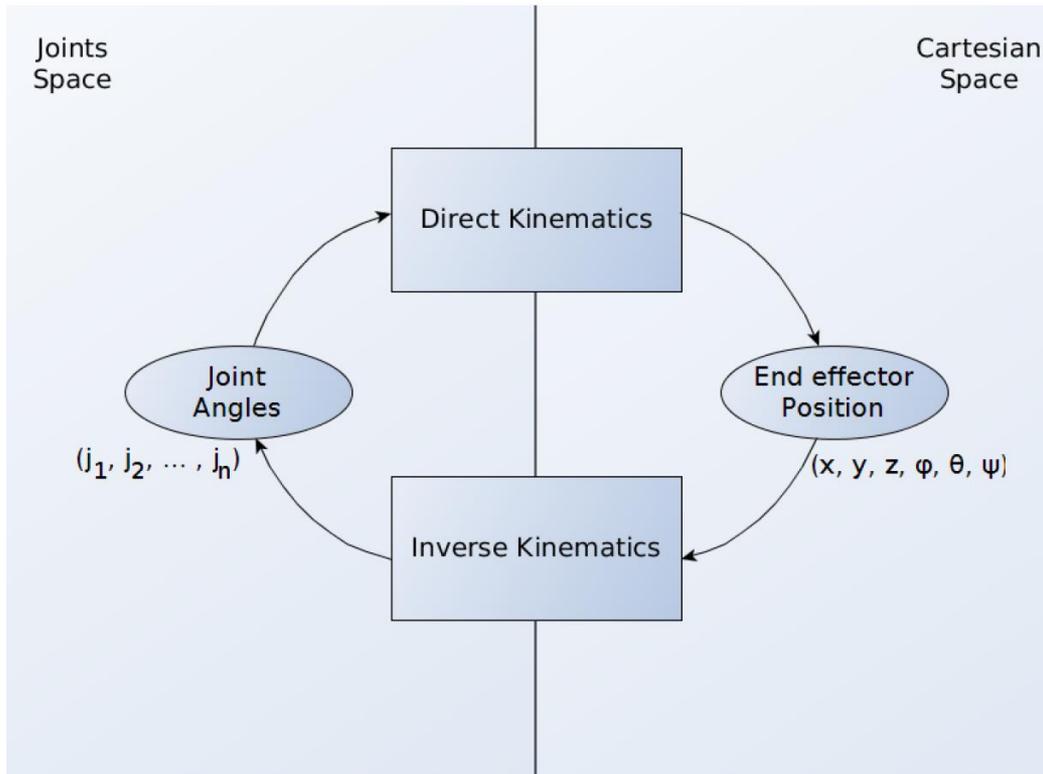
Computing IK using a NN

P_x P_y P_z

NN

θ_1 θ_2 θ_3
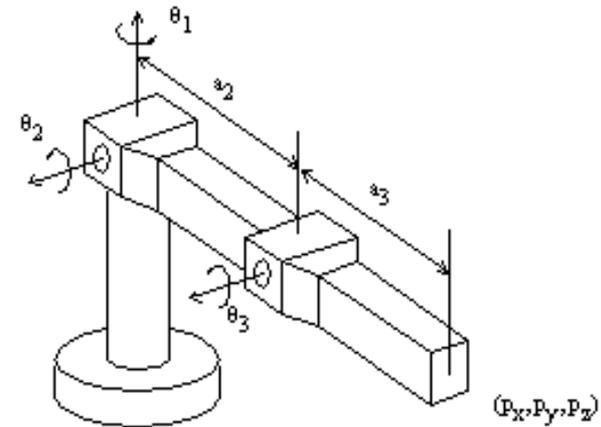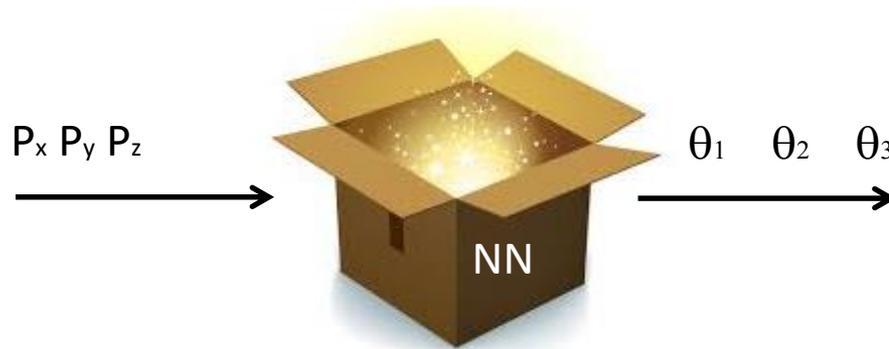
How do you make the network learning?

P_x
P_y
P_z

$+1$

θ_1
θ_2
θ_3
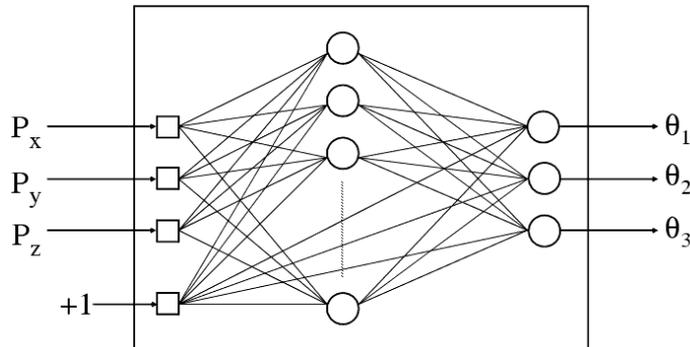
✓ Creating a dataset of <joint_posistions, end effector_positions> using the direct kinematics

Learning the Inverse Static Solution

- It is not always possible to compute the inverse kinematic solution using the joint positions

- For soft continuum robots actuated by cables it is possible to exploit the relation between the cable tendion and the end effector position, in order to control the tip

ISTITUTO DI BIOROBOTICA

Scuola Superiore
Sant'Anna
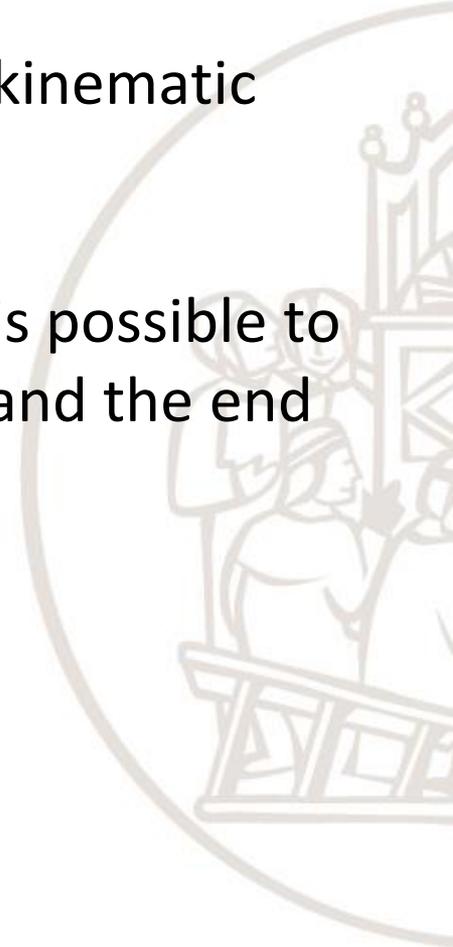
- **Control of the soft arm through the learning of the inverse model that allows to control the end effector position through the cable tension**

- **The inverse problem can be learned collecting points and exploiting the approximation capability of the NN as for the rigid robots**
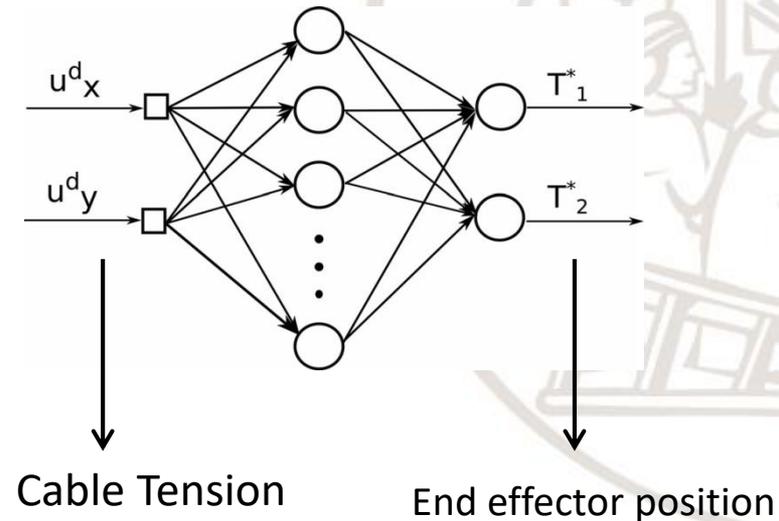
Cable Tension   End effector position

A Neural Network can be used to solve the inverse solution generating an adaptive approach :

The direct model relation is :

$$x = f(q) \qquad\qquad (1)$$

This particular representation is not invertible when m<n (redundant)

where, $x \in \Re^m$ **is the position and orientation vector of the end effector**; $q \in \Re^n$ is the joint vector f is a surjective function

Learning the Inverse Static Solution: an adaptive approach (II)

We can develop local representations by linearizing the function at a point (q^o) thereby obtaining :

$$\delta x = J(q^o)\delta q \quad (I)$$

Here $J(q^o)$ is the Jacobian matrix at the point q^o; δx and δq are infinitesimally small changes in x e q. The differential IK method involves generating of ($\delta x, \delta q, q$) and learning the mapping ($\delta x, q^o$) $\rightarrow \delta q$

The learning is feasible since the differential IK solutions form a convex set and therefore averaging multiple solutions still results in a valid solution

The method we have proposed involves expanding Eq. I and expressing it in terms of absolute positions, as shown below:

$$J(\boldsymbol{q}_i)\boldsymbol{q}_{i+1} = \boldsymbol{x}_{i+1} - f(\boldsymbol{q}_i) + J(\boldsymbol{q}_i)\boldsymbol{q}_i \text{ (II)}$$

$\boldsymbol{q}_{i+1}$ is the next actuator configuration for reaching a point $\boldsymbol{x}_{i+1}$ from the present configuration $\boldsymbol{q}_i$. Note that Eq. II is only valid when the configurations are infinitesimally close. However, for practical purposes this can be a good approximation for larger regions.

The analytical solution for the equation II can be written as:

$$\boldsymbol{q}_{i+1} = G(\boldsymbol{x}_{i+1} - f(\boldsymbol{q_i}) + J\boldsymbol{q_i}) + (I_n - GJ)\boldsymbol{z}$$

where G is a generalized inverse of $J(\boldsymbol{q_i})$ and I_n is the identity matrix, and $\boldsymbol{z}$ is an arbitrary n-dimensional vector. The first component represents the particular solution to the non-homogenous problem prescribed in Eq. II and the second component represents the infinite homogenous solutions. It can be proved that the solution space still forms a convex set. Therefore, a **universal function approximator (i.e. NN)** can be used for learning the mapping

$$(\boldsymbol{q_i}, \boldsymbol{x}_{i+1}) \rightarrow (\boldsymbol{q}_{i+1})$$

The samples (q_i, q_{i+1}, x_{i+1}) genereted are such that

$$| \, q_{i+1} - q_i \, | < \epsilon$$

An appropriate value of ϵ is between 10%-5% of the maximum actuator range

Learning the Inverse Static Solution: an adaptive approach (V)

- We use a feed-forward NN to learn the relation:
  - $$(q_i, x_i, x_{i+1}) \rightarrow (q_{i+1})$$

- **The values of** x_i, x_{i+1} are genereted using the direct model as showed for the learning of the IK of a rigid manipulator

TRAINING PHASE   TEST PHASE

(q_{i-1}, x_i, x_{i-1}) $\longrightarrow$ INPUT (q_i, x_i, x_{i+1}) $\longrightarrow$ INPUT

(q_i) $\longrightarrow$ Desired OUTPUT (q_{i+1}) $\longrightarrow$ Network OUTPUT

Learning the Inverse Static Solution: an adaptive approach (V): real robot implementation

ISTITUTO DI BIOROBOTICA

Scuola Superiore Sant'Anna

Six DoF Hybrid System (Pneumatic and Tendon)

I-Support Prototype

LEARNING

- 2000 sample points divided in the ratio 70:30 for training and testing respectively
- 2 hours for data collection, training and setting-up

TESTS

Twenty Five random points selected from workspace

	Mean Error	Standard Deviation
Position (mm)	5.58	3.08
X- axis rotation (degrees)	2.76	5.42
Y- axis rotation (degrees)	1.84	1.83
Z- axis rotation (degrees)	3.85	7.02

ISTITUTO
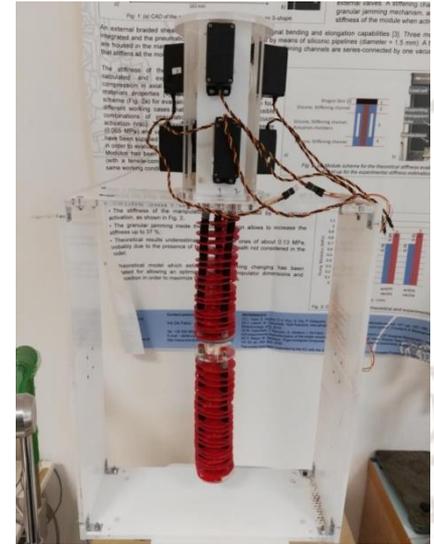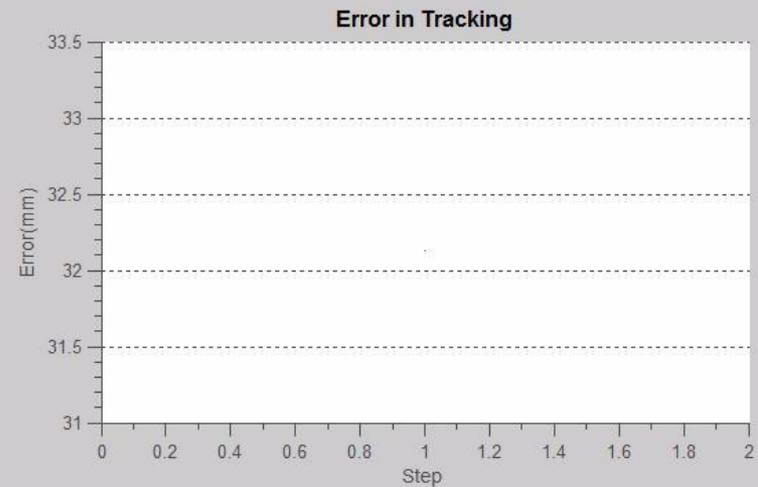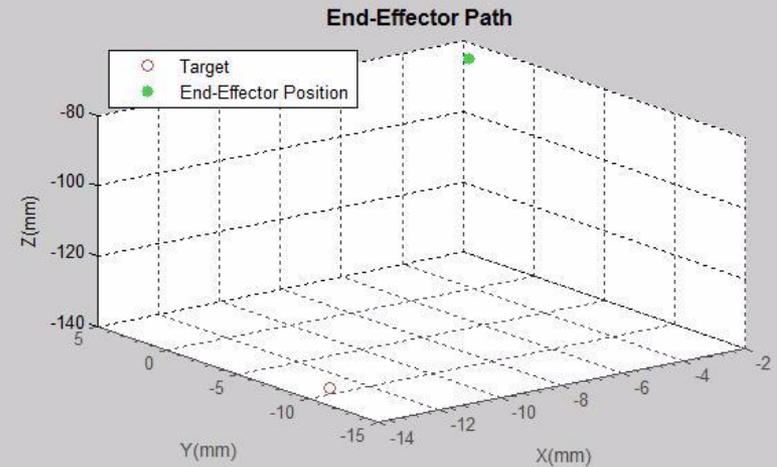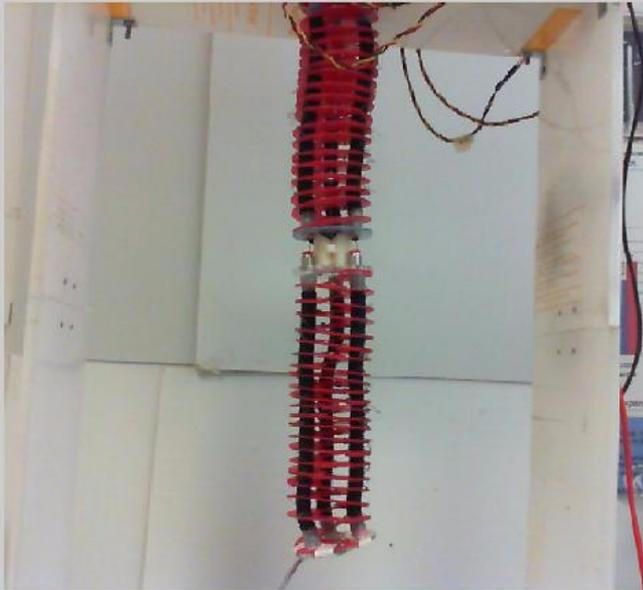
Unlike the case of rigid robots external disturbances modify the kinematics of the soft manipulator

This is the first experimental implementation of soft robots tracking under external disturbances

Offline signal prediction

- Using a NN to foresee the signal in the future
- Training the NN using past values
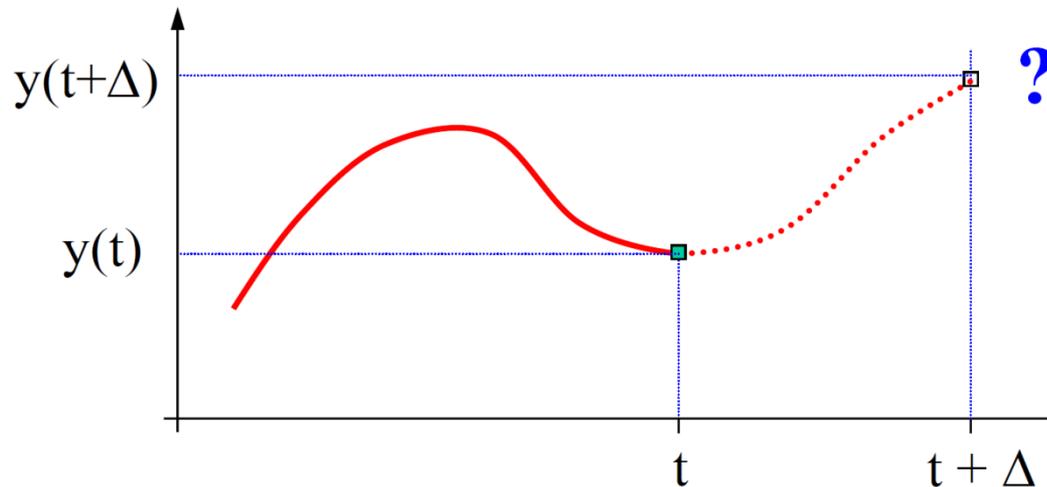
- Learning phase
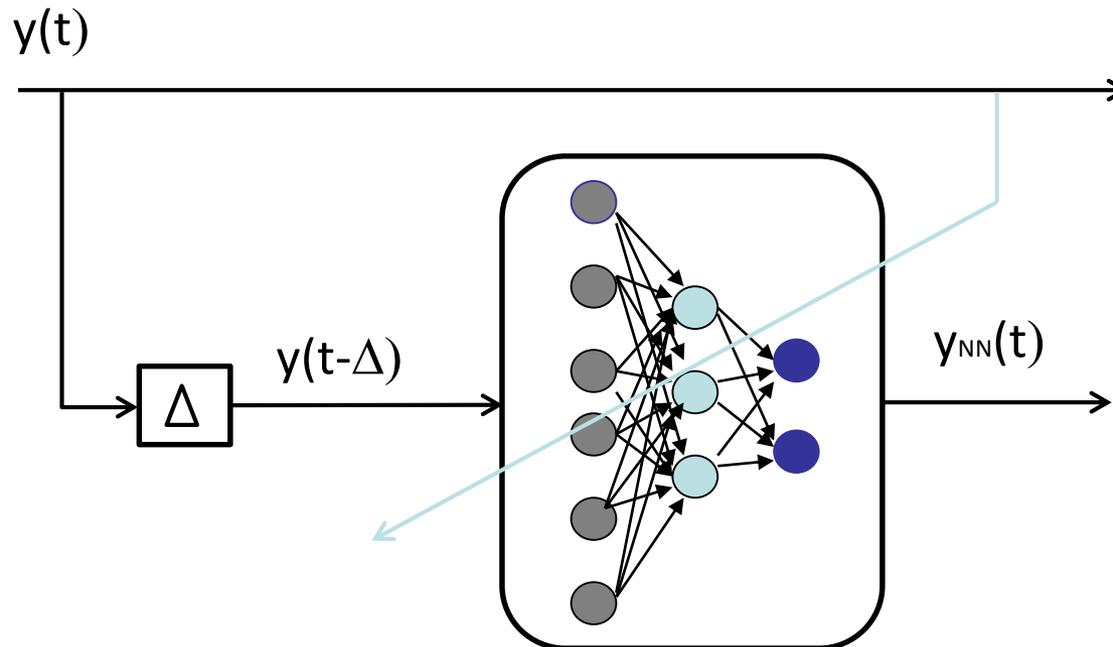
- The NN learns the relation between y(t) and y(t-Δ)

- Test phase

y(t)
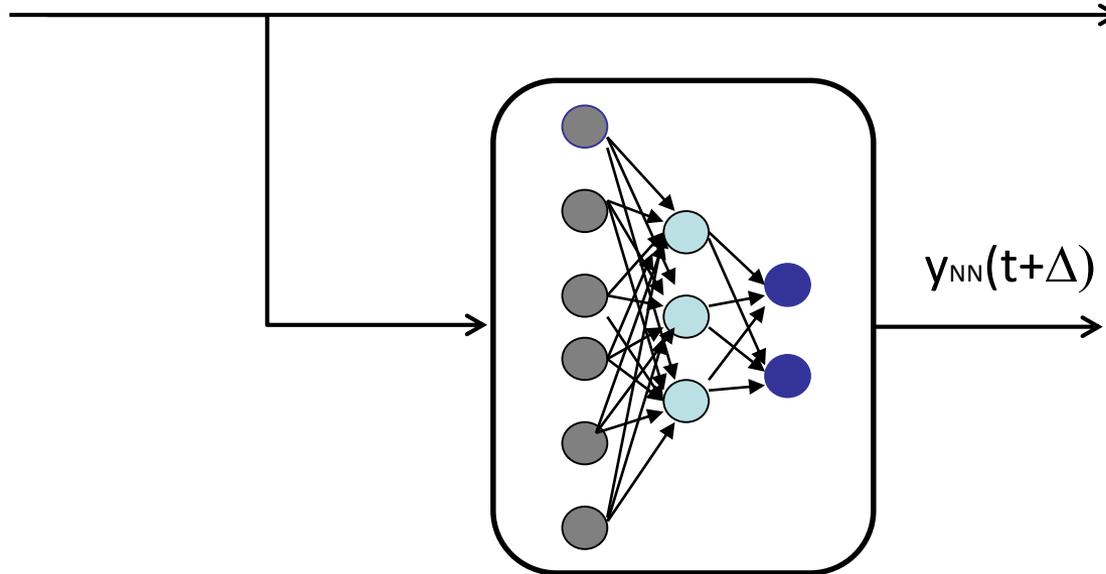
- The NN yields an estimation of the y(t+Δ)

Learning and prediction
Inconsistent training set
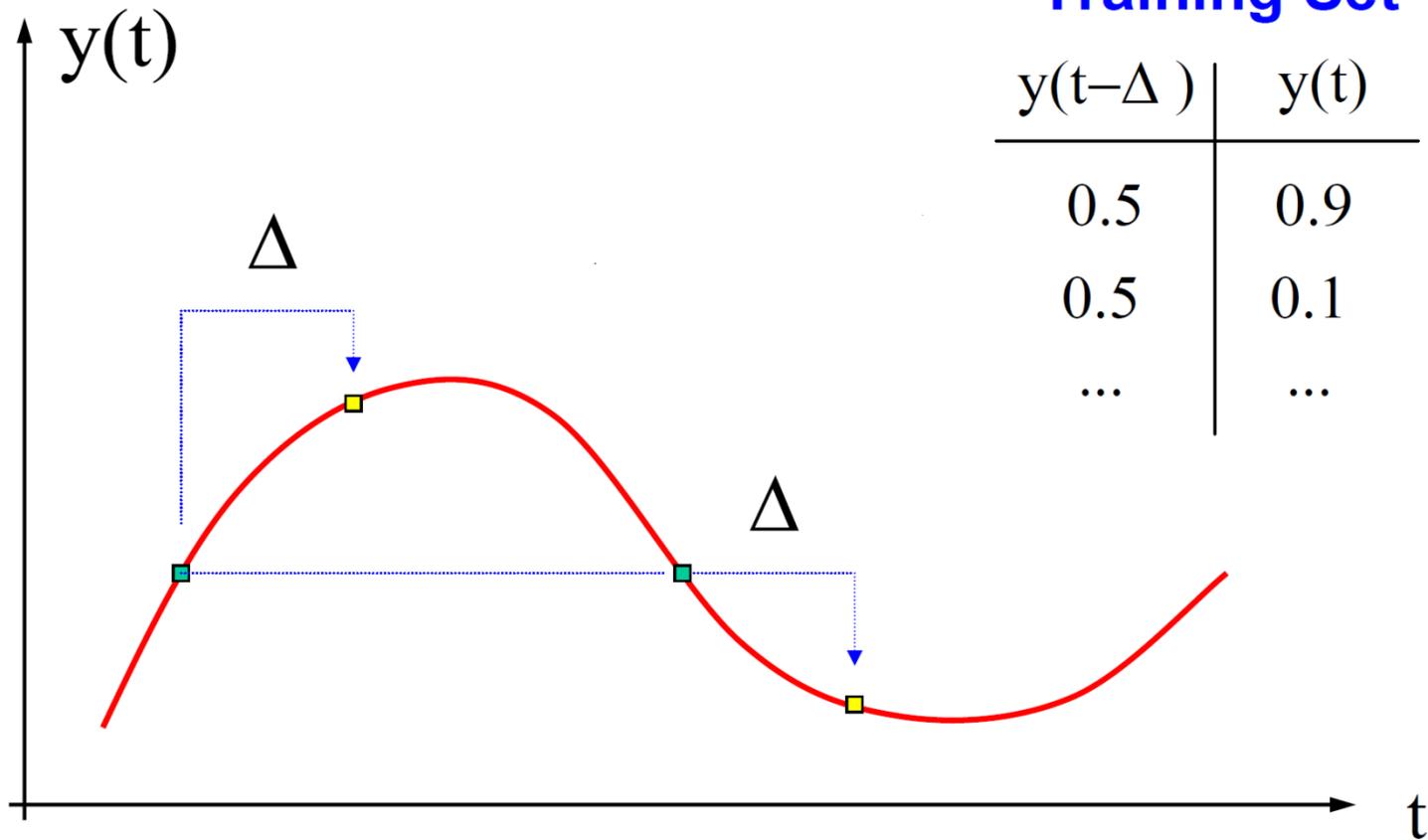
Prediction with delays

ISTITUTO DI BIOROBOTICA

Scuola Superiore Sant'Anna

y(t)

y(t-Δ)

y(t-2Δ)

y(t-nΔ)

$y_{NN}(t)$
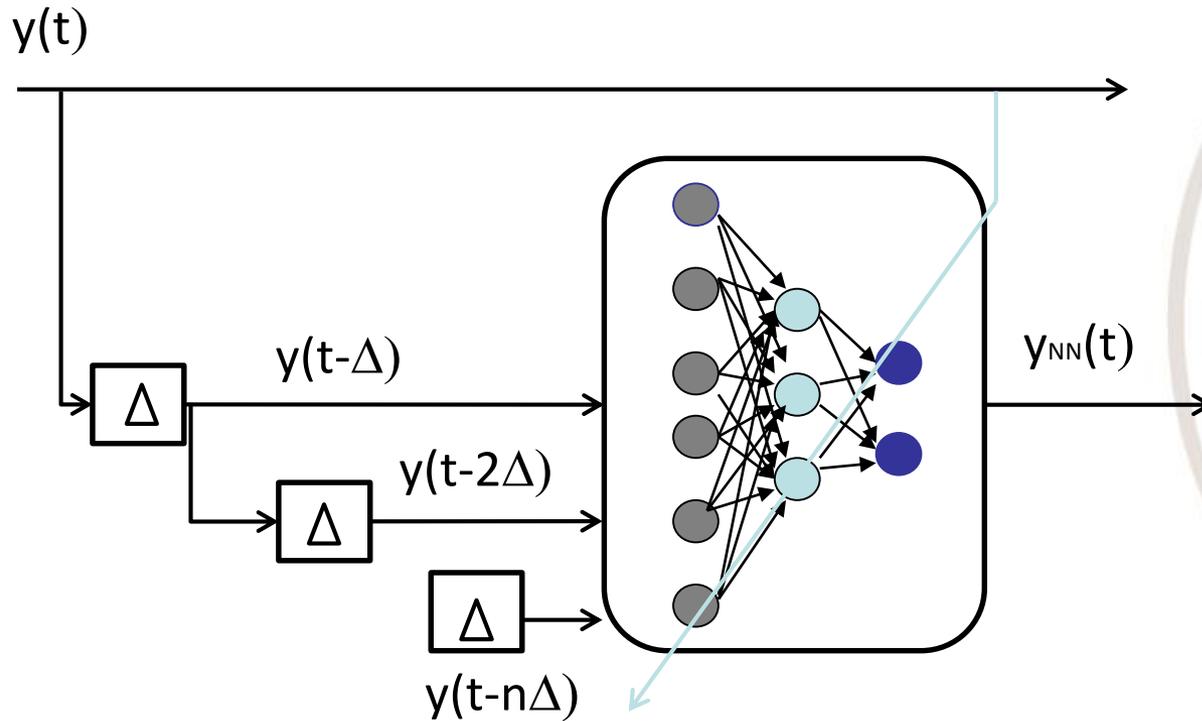
ONLINE prediction

- Generate a prediction with any a-priori knowledge of the signal
- Fast convergence with less input seen
- Simple model based on a single neuron (PERCEPTRON) receiving as input current and past values (x(t)-x (t-d))
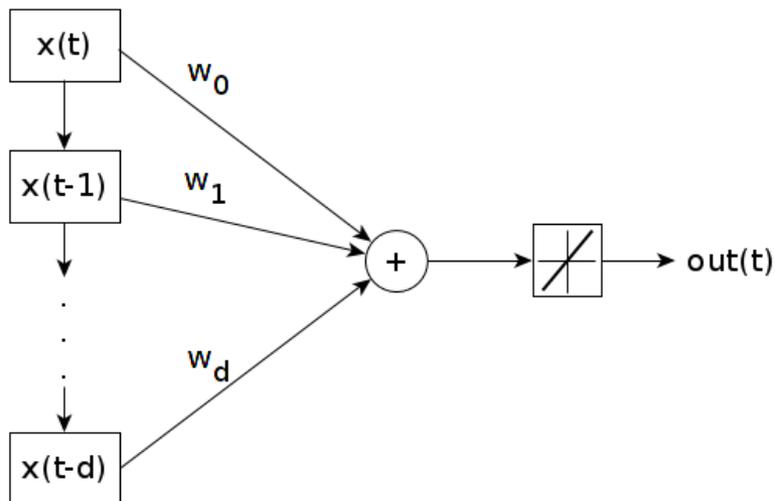
Es. Ten steps ahead prediction d=10

 training set (<input, desired output>) :
<X_1-X_{10}, X_{20}>,<X_2-X_{11}, X_{21}>, ...
<X_n-X_{n+10}, X_{n+20}>

Network output: y_{10},y_{11}... y_{n+10}

Head stabilization in biped locomotion
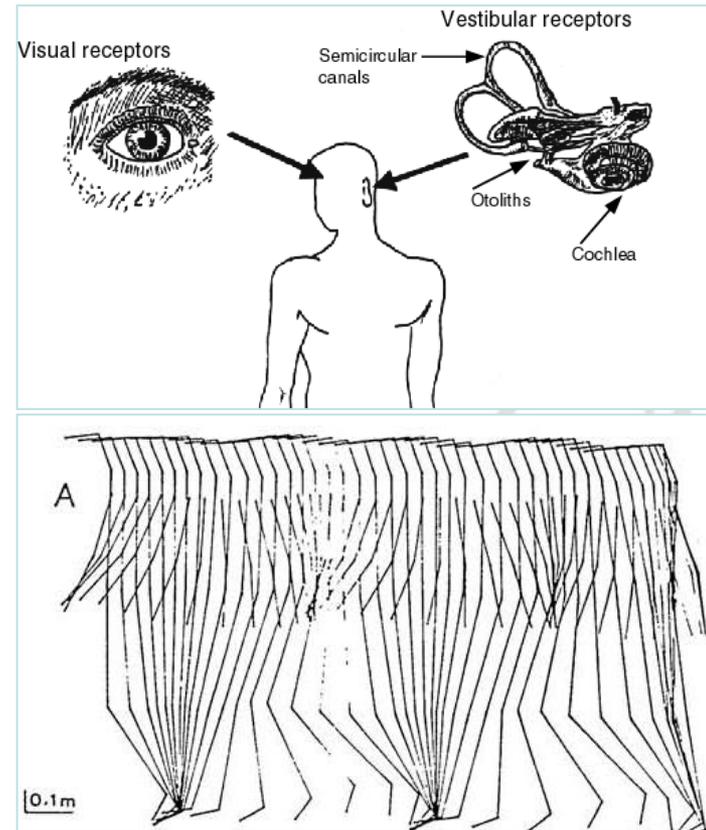
Pozzo T. et al. (1990)

The brain uses the information coming from vestibular system to generate a *unified inertial reference frame*, centred in the head, that allows whole-body coordinated movements and head-oriented locomotion.

Berthoz A., 2002, *The sense of movement*. Harvard University Press

Adaptive head stabilization model

Block diagram: $\nu^r, \phi^r, \psi^r \rightarrow (+/-) \rightarrow \nu^e, \phi^e, \psi^e \rightarrow$ **FEL model** $\rightarrow \hat{\nu}, \hat{\phi}, \hat{\psi} \rightarrow$ **Neural Network** $\rightarrow \vartheta_1, \vartheta_2, \vartheta_3 \rightarrow$ **System**. Feedback: ν, ϕ, ψ and $\hat{\nu}^t, \hat{\phi}^t, \hat{\psi}^t$
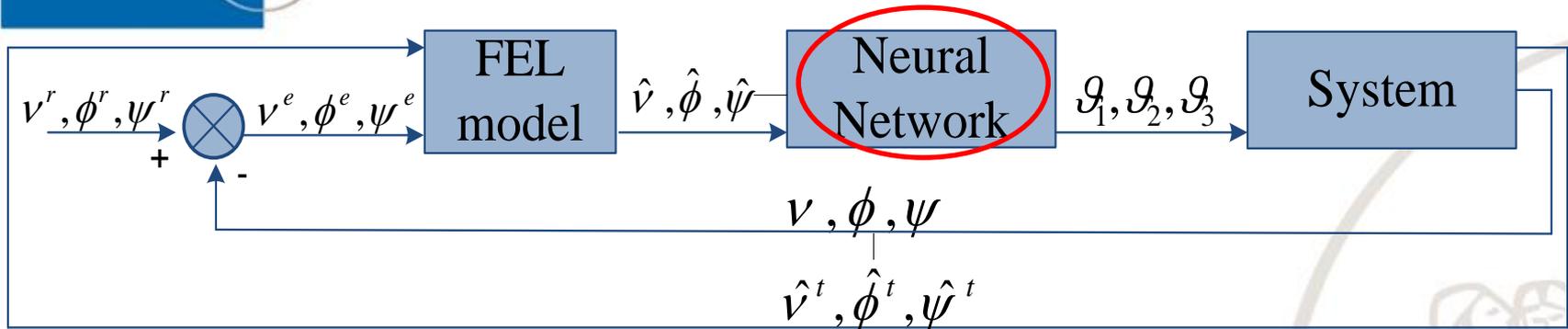
✓ The controller is based on a feed feedback error learning (**FEL**) model. This model estimates the orientation of the head , which allow following a reference orientation .

✓ The output of this model is sent as input to a **Neural Network** which computes the joint positions relative to the estimated orientation

✓ Artificial Neural Network capable of solving the inverse kinematics problem without using the closed form solution.
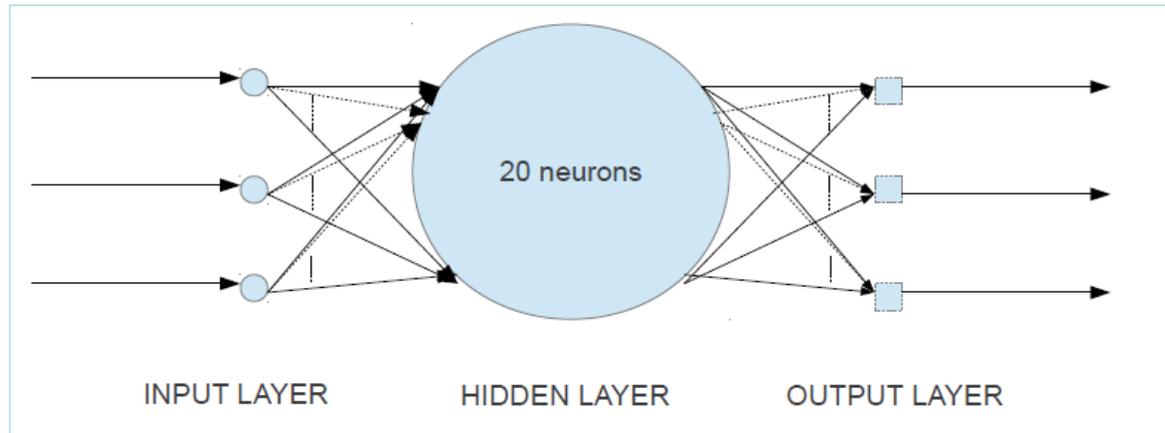
✓ The network has one hidden layer of 20 units. It takes as input the head orientation (ν, ϕ, ψ) and as output the neck joints angles $(\theta 1, \theta 2, \theta 3)$.

Adaptive head stabilization model – results on the Sabian robot

EXTERNAL PERTURBATION TEST
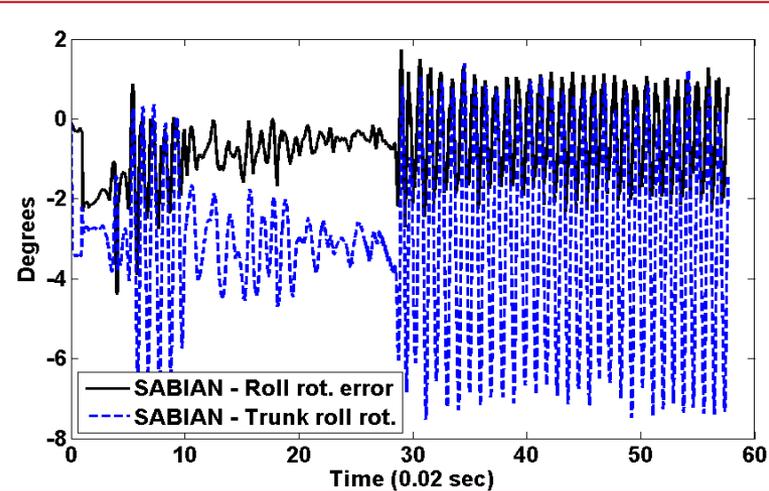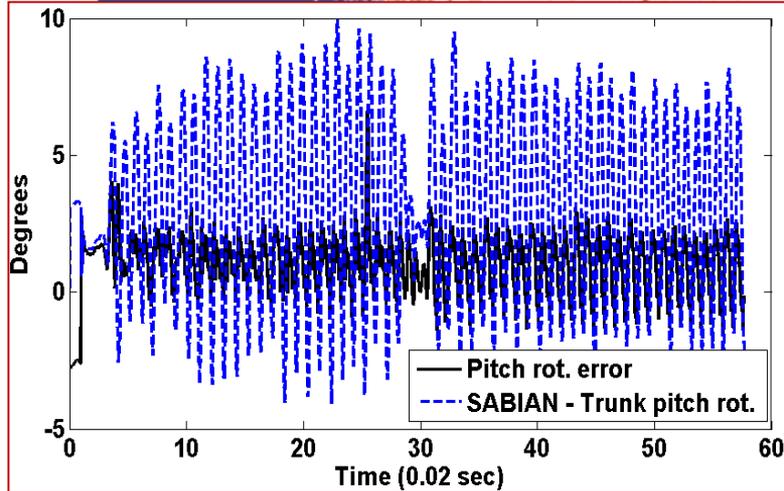
✓ The pitch and the roll rotation error are less than **2 degrees**, while the trunk peak-to-peak amplitude was almost 15 degrees for the pitch and 8 for the roll

E.Falotico, N. Cauli, K. Hashimoto, P. Kryczka, A. Takanishi, P. Dario, A.Berthoz, C.Laschi, (2012). "Head stabilization based on a feedback error learning in a humanoid robot". *Proceedings of the 21st IEEE International Symposium on Robot and Human Interactive Communication, Paris, France, September 9-13, 2012.*
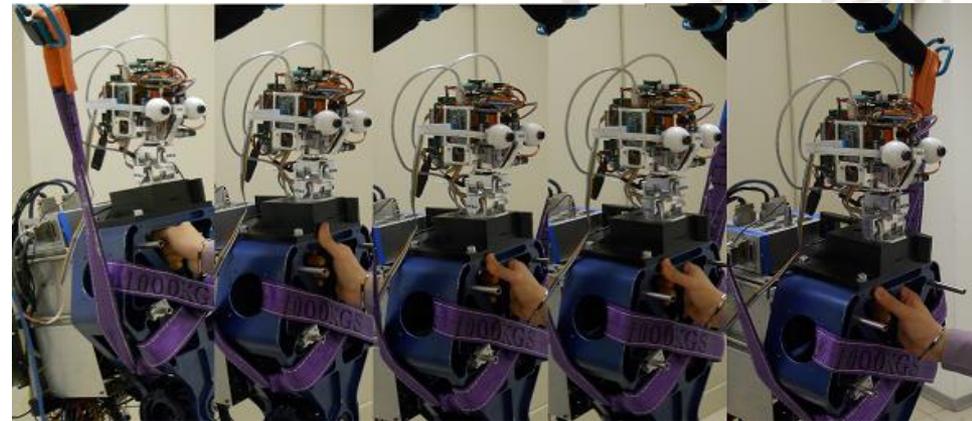
Adaptive head stabilization model – results on the Sabian robot

ISTITUTO DI BIOROBOTICA

Scuola Superiore Sant'Anna

WALKING TEST
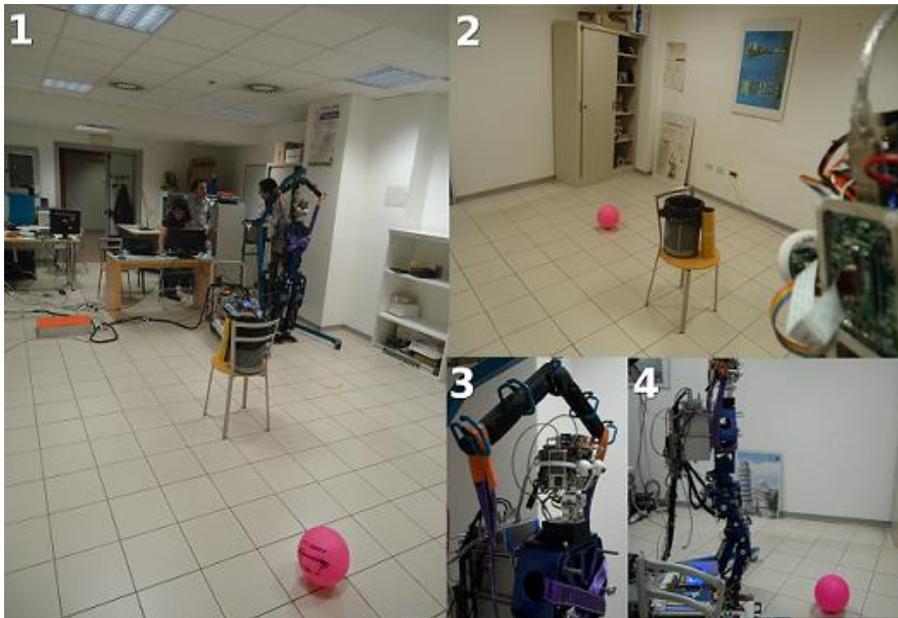
1. The scenario with the robot in the starting position and the red ball indicating the end of the path.
2. The scenario from the robot viewpoint.
3 & 4. The robot performing the walking pattern with head stabilization.
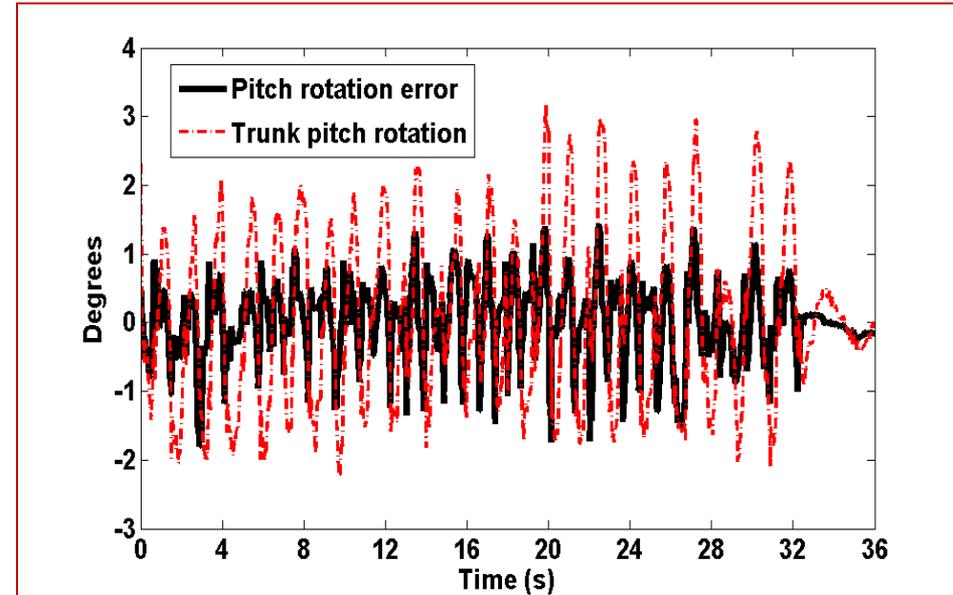
✓ This experiment started with the network weights and the regressor parameters of the FEL set to the values reached at the end of a training phase

✓The peak to peak amplitude of the head is less than **2 degrees** during the whole pattern execution.

Competitive learning model for the generation of a saliency map

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

Saccade target
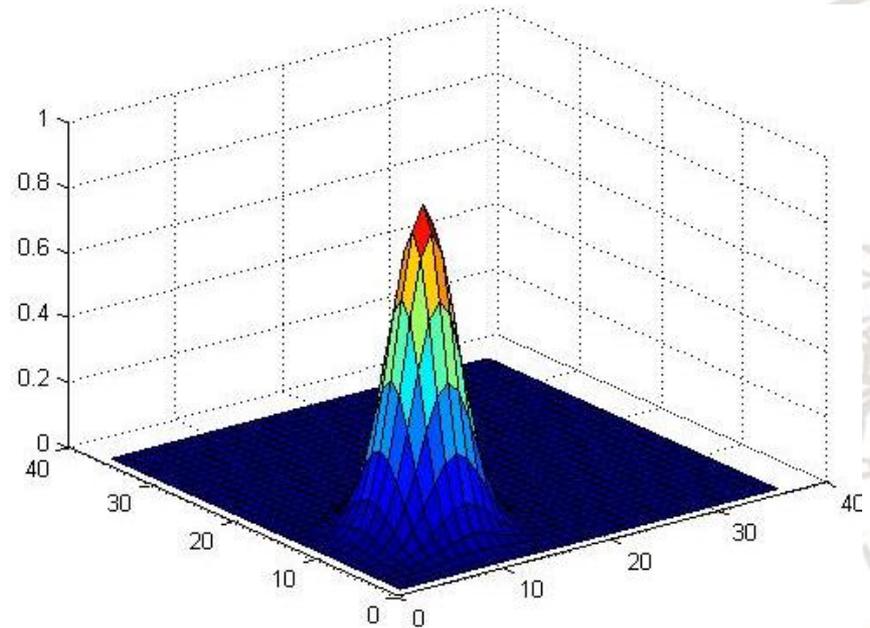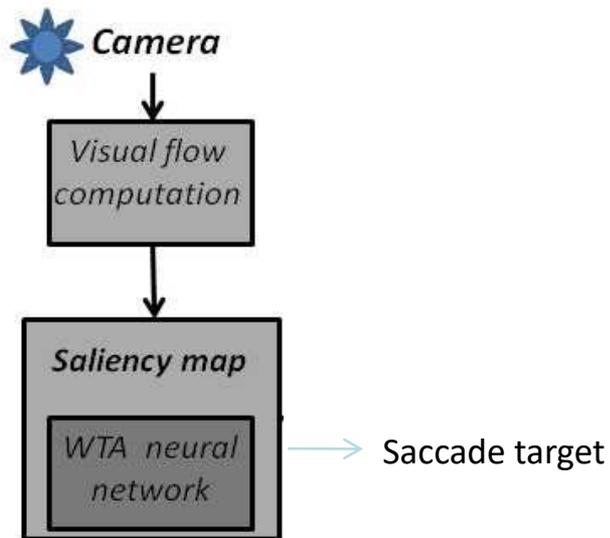
The neural network model the role of the visual cortex (Amari et al,1977)

The objective is to select among competitive spatial input in order to provide the point interest in the visual field

THE AVP SCHEME
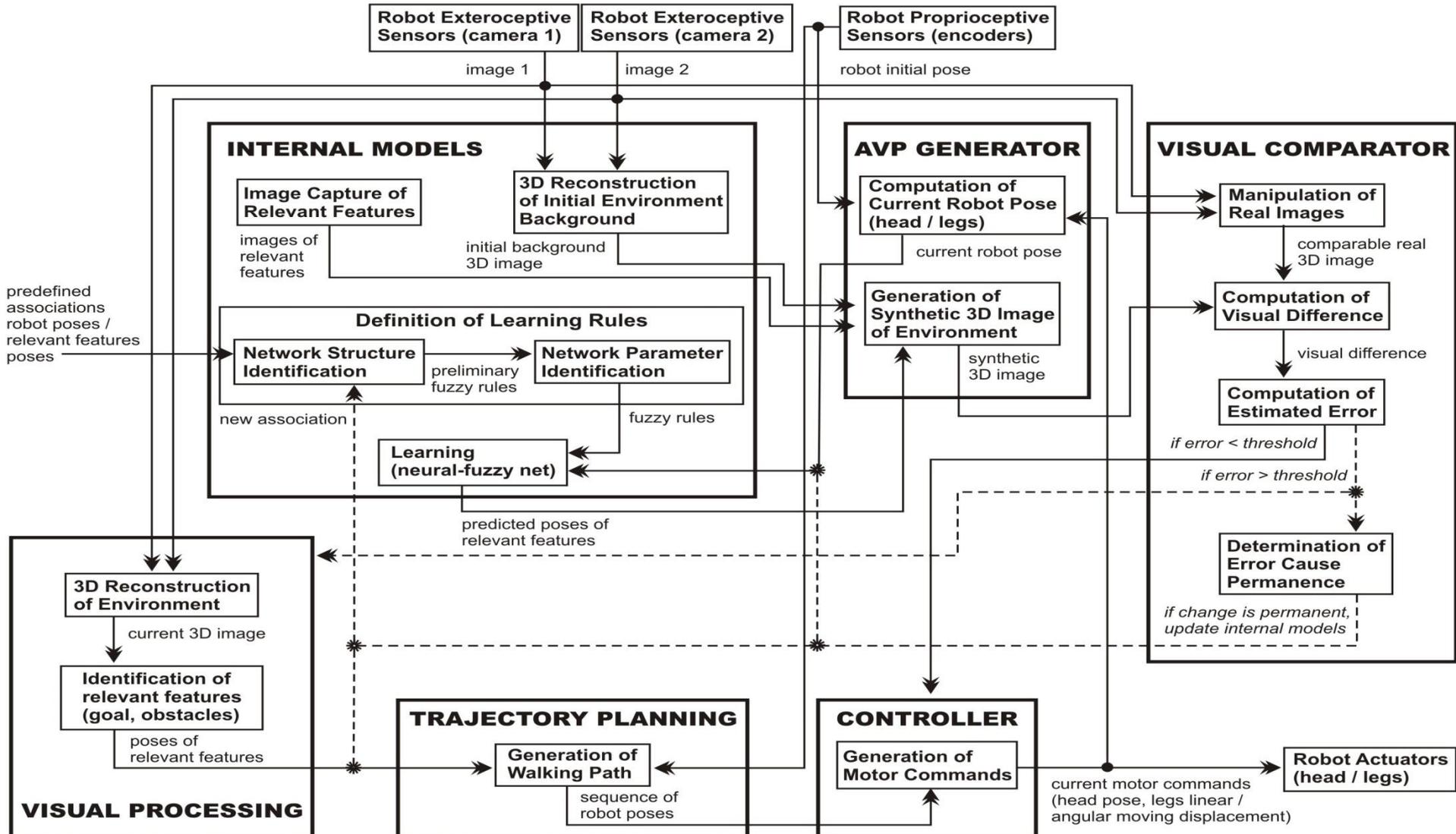
Traditional Perception-Action cycle for the AVP architecture

• **Visual Processing** module takes as input current images from both robot cameras to reconstruct the environment producing the **relevant feature position.**

• The poses of relevant features are sent to a **Trajectory Planning** module to generate the walking path

• The **Controller** module then takes the first robot pose from the sequence of poses planned by the Trajectory Planning module and produces the corresponding motor commands

• This cycle continues until the robot reaches the target.

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

• **Internal Models** of the environment and of the task to be performed are necessary to *predict future visual perceptions*.

•Images of different features relevant to the locomotion task are captured and memorized.

ISTITUTO DI BIOROBOTICA

Scuola Superiore Sant'Anna

- At every step, the **Visual Comparator** module compares the current image of the environment with a synthetic image predicted by the **AVP Generator** module.

- To produce the synthetic image, AVP Generator computes the current robot pose taking into account the initial pose of the robot and the motor commands executed at the immediately previous time step.

- The current robot pose is sent to the **Learning** sub-module of Internal Models, where the neural network predicts the corresponding poses of the relevant features.

- The AVP Generator then takes as input the memorized images of the relevant features, and **creates a synthetic image** by pasting them on the environmental background at poses predicted by the neural net.

Implementation of internal models for EP generation
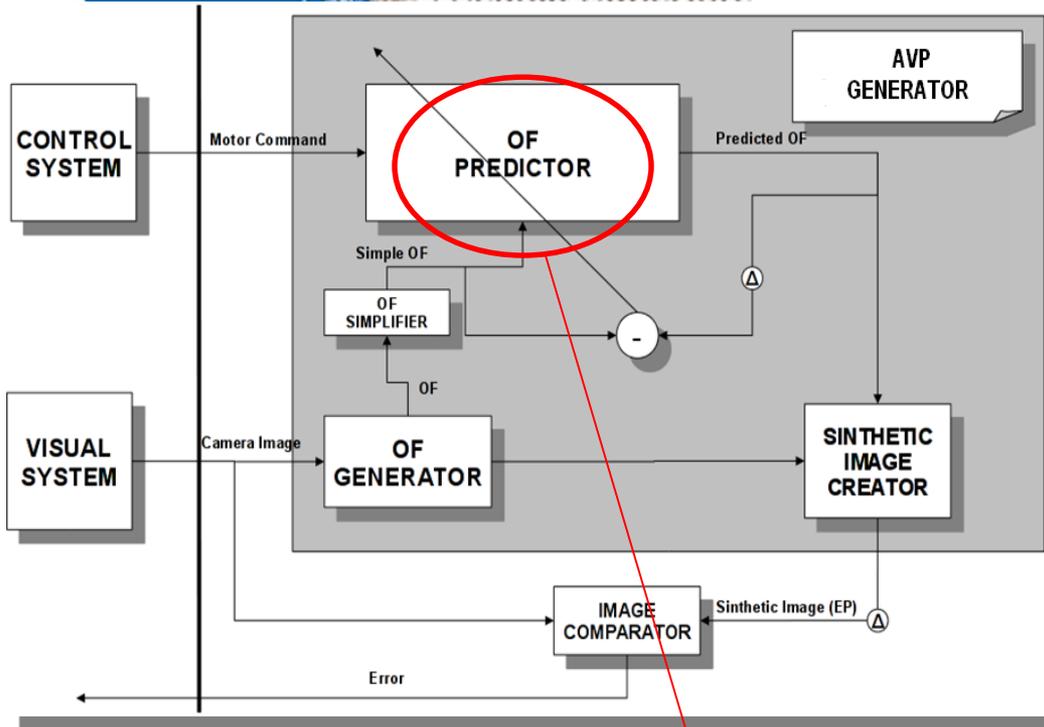
Internal model

- **OF GENERATOR**: generates the Optical Flow from the camera image with Lucas-Kanade algorithm.
- **OF SIMPLIFIER**: generates a Simple OF dividing in zones the Optical Flow and calculating the mean flow vector for each zone.
- **OF PREDICTOR**: predicts the next step OF using an ESN. Learning is performed off-line.
- **SYNTHETIC IMAGE CREATOR**: generates a sinthetic image rapresenting the next step camera image.
- **IMAGE COMPARATOR**: generates the error between the synthetic image and the corresponding camera image.

ESN GENERAL SETTINGS
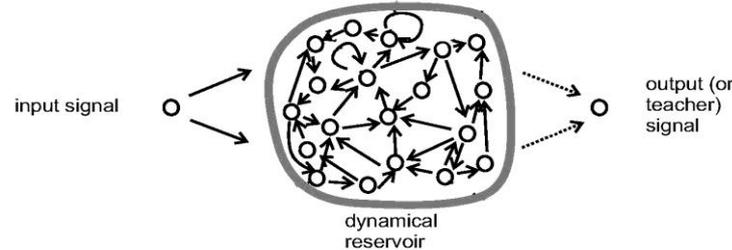
- Off-line learning that minimizes the MSE.
- No feedback connections.
- Input signal composed by motor commands and simple Optic Flow
- Output signal composed by next step simple Optic Flow.

ISTITUTO DI BIOROBOTICA

Scuola Superiore
Sant'Anna
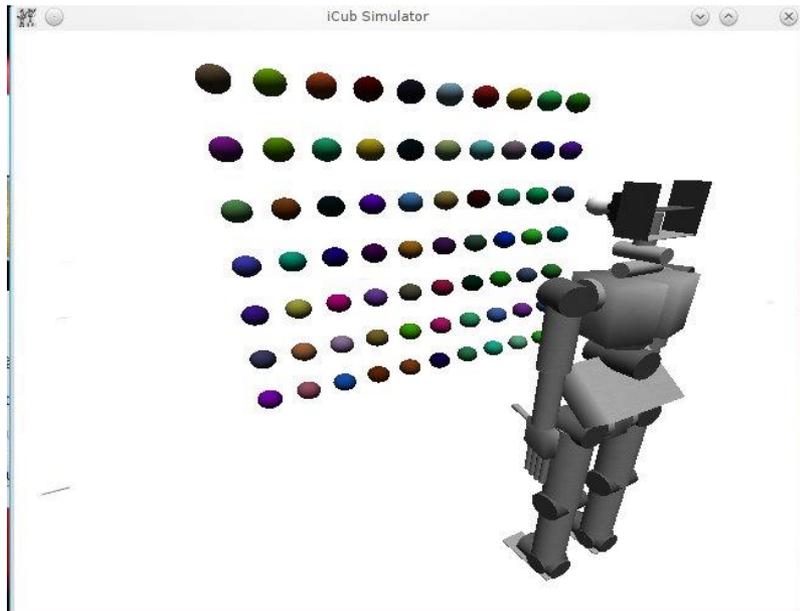
EXPERIMENTS

- Background and floor were replaced by a white screen.
- A matrix of coloured spheres was placed in front of the robot.

ISTITUTO DI BIOROBOTICA

Scuola Superiore Sant'Anna
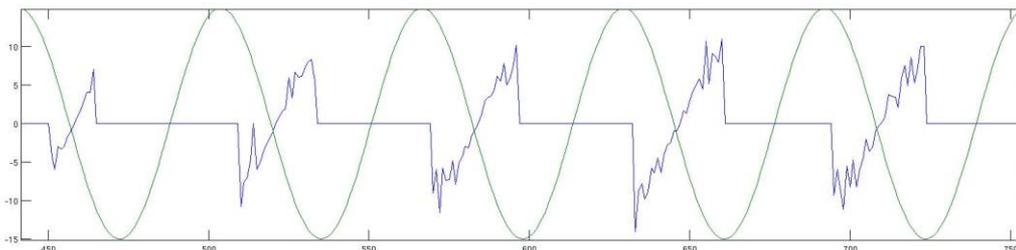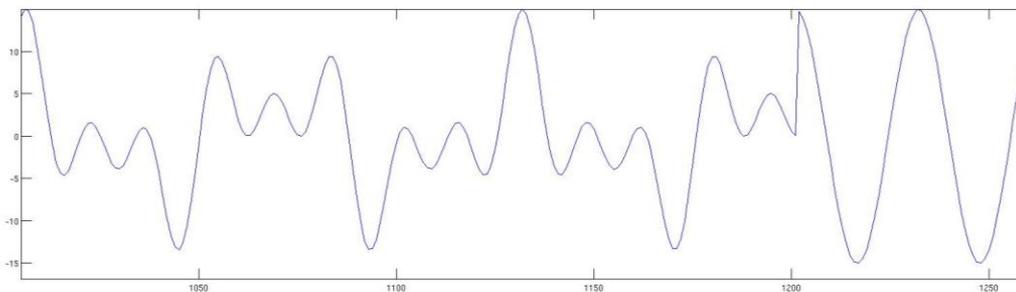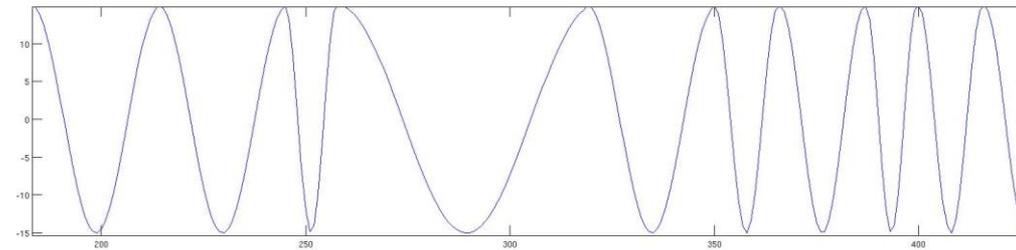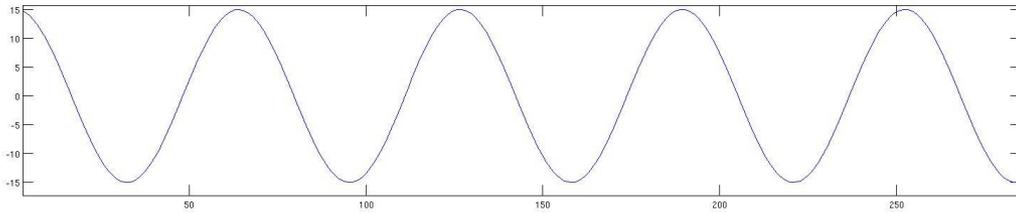
A) Motor command: sinusoid of amplitude 15 and frequency 0.1. Dataset of 1500 elements (1200 training, 300 test).

B) Motor command: sequence of sinusoids of amplitude 15 and frequencies between 0.1 and 0.5. Dataset of 1500 elements (1200 training, 300 test).

C) Training set composed by the sum of 3 sinusoids with amplitude 5 and frequency 0.1 0.25 and 0.4 (1200 elements). Test set composed by a sinusoid with amplitude 15 and frequency 0.2 (300 elements).

D) Motor command: same sinusoid as A. Environment: the matrix of spheres has columns more distants than before. Dataset of 1500 elements (1200 training, 300 test).

Implementation of internal models for EP generation
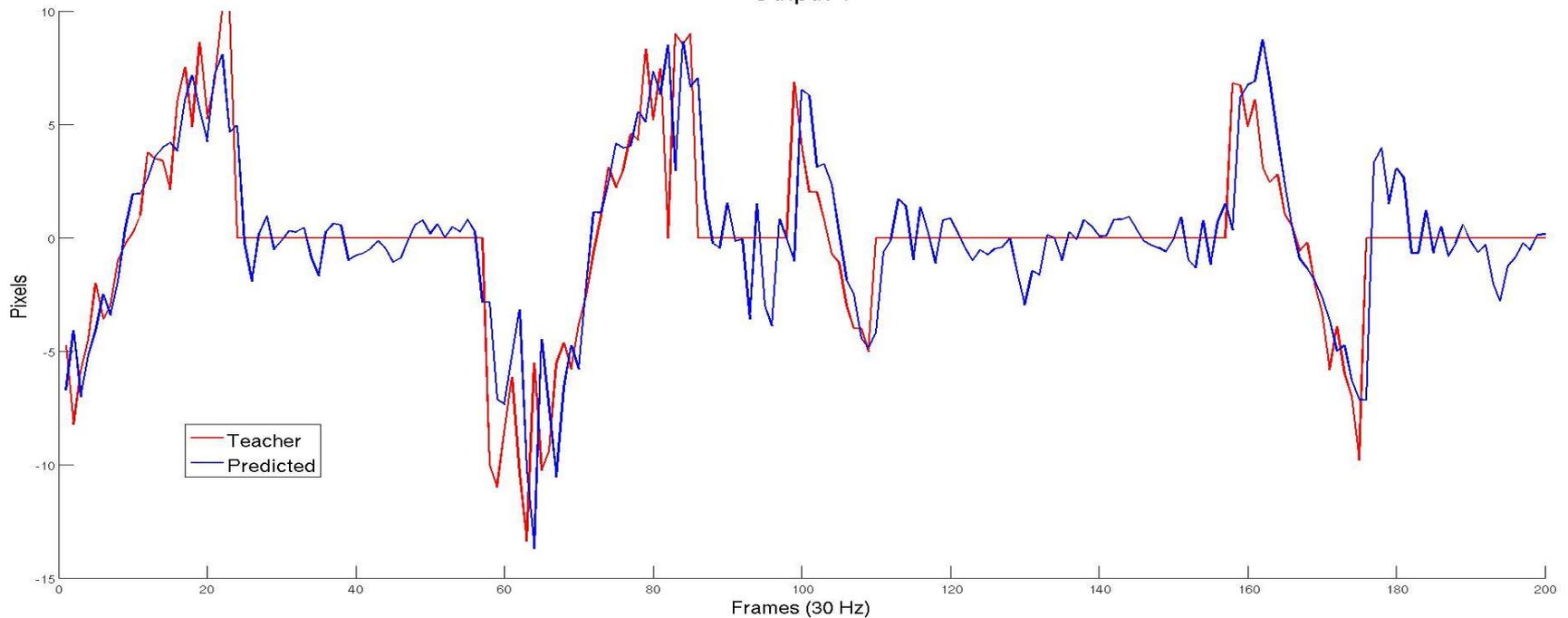
## Trial 1 - SINUSOID

Output 2
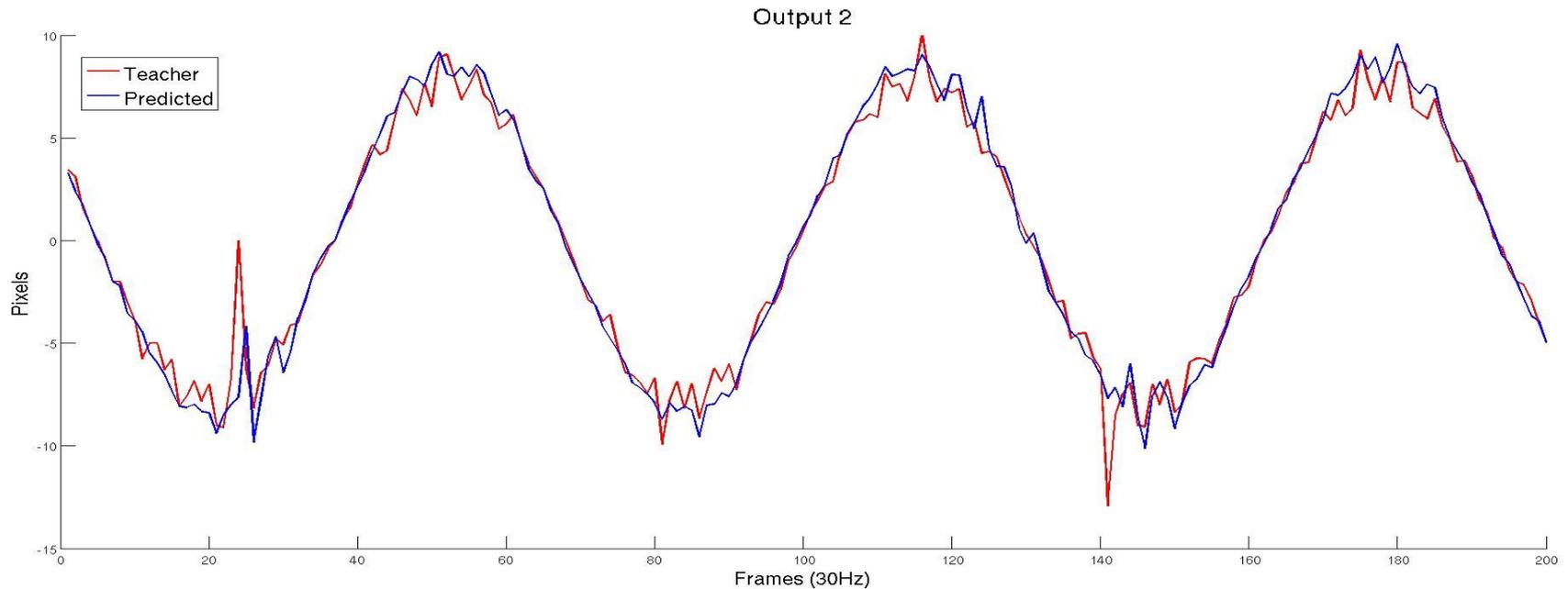
- Input dim: 5, Spectral radius: 0.7, Reservoir dim: 75
- Train Error (pixels MSE):   0.16558   0.15946   0.15664   0.16499
- Test Error (pixels MSE):   0.17732   0.18431   0.15675   0.16521

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

Trial 2 – SINUSOID SEQUENCE

Output 2
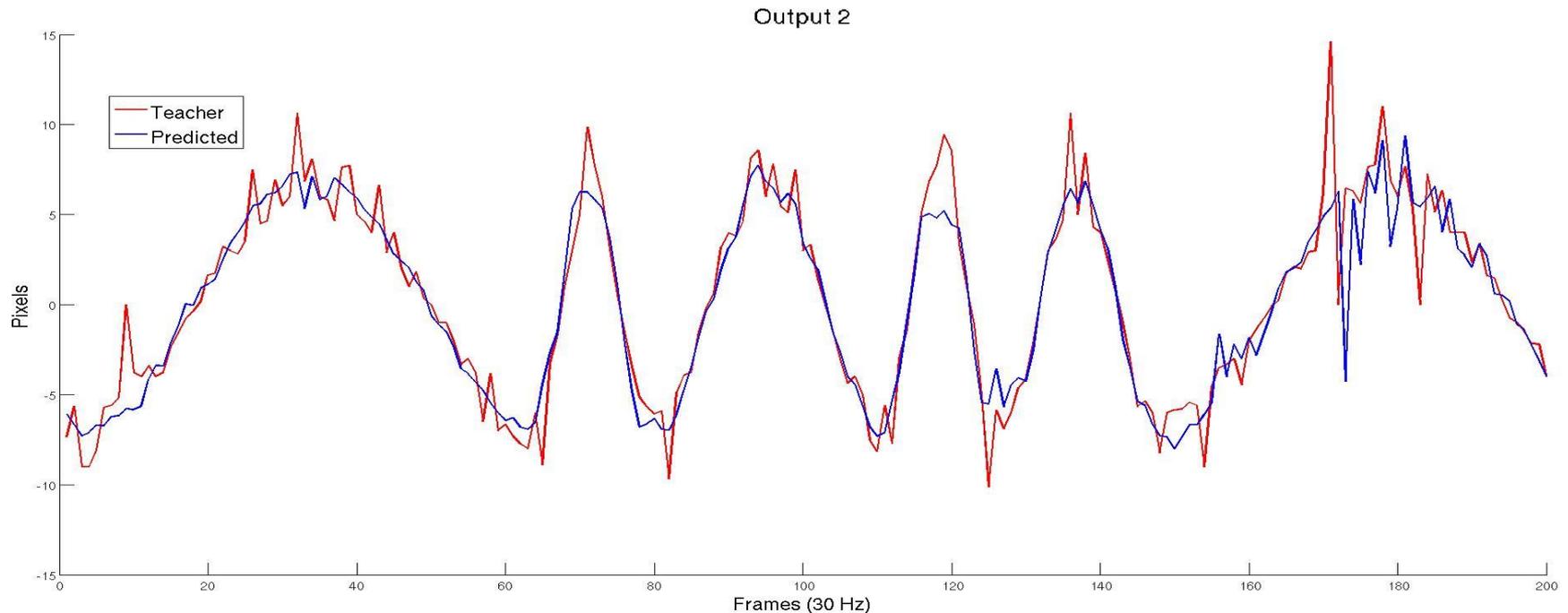
- Input dim: 5, Spectral radius: 0.7, Reservoir dim: 75
- Train Error (pixels MSE):   0.23676   0.23667   0.23405   0.25608
- Test Error (pixels MSE):   0.28956   0.31899   0.31899   0.32285

ISTITUTO
DI BIOROBOTICA

Scuola Superiore
Sant'Anna

Trial 3 – SINUSOID SUM
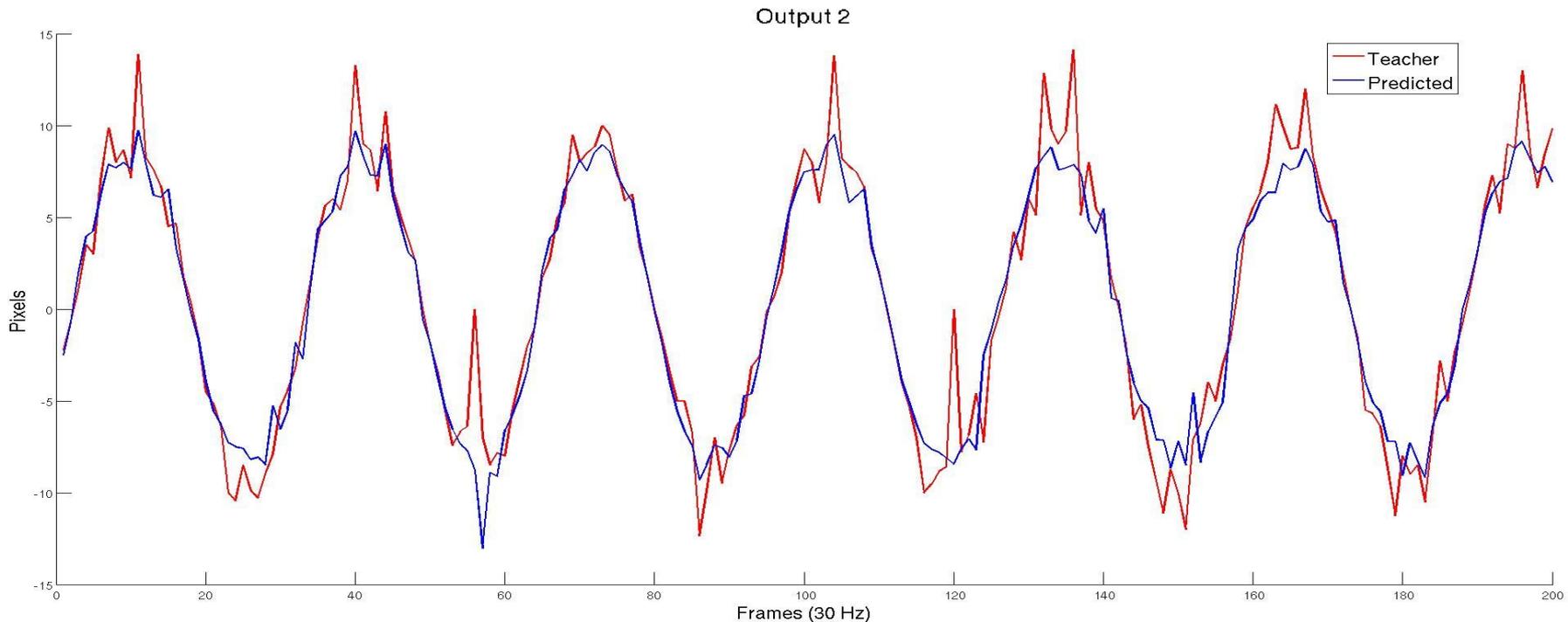
Output 2

- Input dim: 5, Spectral radius: 0.7, Reservoir dim: 75
- Train Error (pixels MSE):   0.21956   0.2316   0.2188   0.23137
- Test Error (pixels MSE):   0.31305   0.28552   0.28264   0.30684

Trial 4 – DISTANT SPHERES

- Input dim: 5, Spectral radius: 0.7, Reservoir dim: 75
- Train Error (pixels MSE):   0.50885   0.33475   0.3148   0.44656
- Test Error (pixels MSE):   0.63688   0.6102   0.61788   0.62655

A controller for the gaze : a GNG based neurocontroller

✓ Biologically-inspired model
  ✓ Sensory-motor map
  ✓ Excitatory/ihibitory stimuli
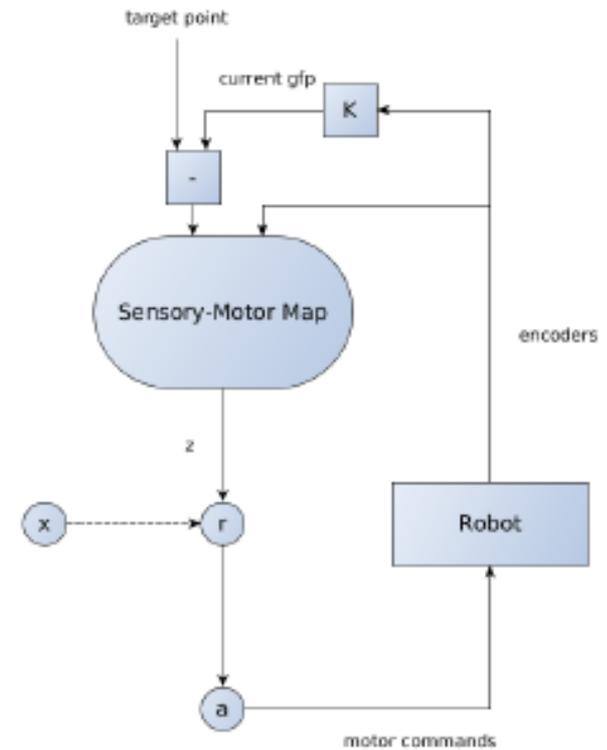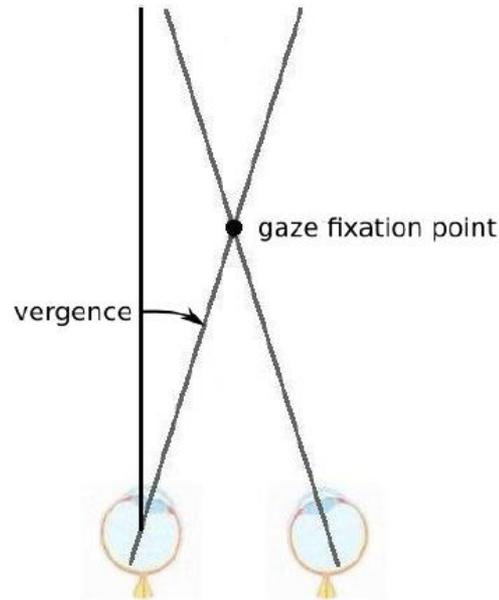
Asuni, G., Teti, G., Laschi, C., Guglielmelli, E., and Dario, P. (2005, April). A robotic head neuro-controller based on biologically-inspired neural models. In Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on (pp. 2362-2367). IEEE.

Growing Neural Gas model
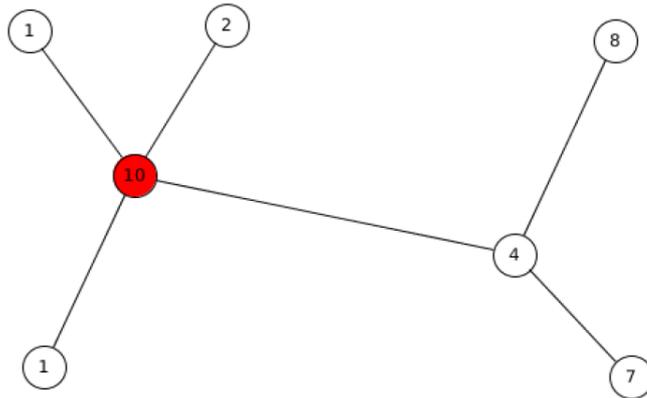
The node with the highest accumulated error is selected
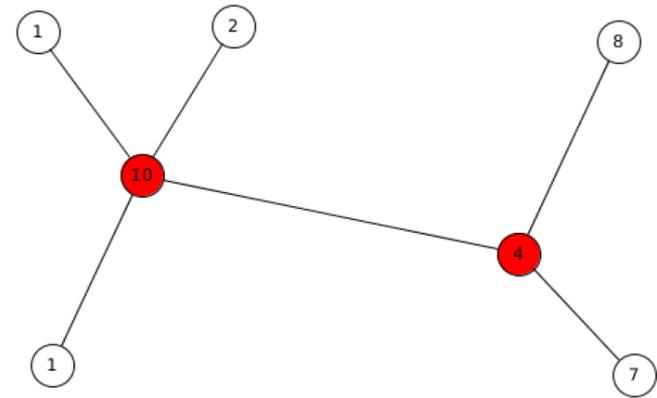
The neighbour with more accumulated error is selected
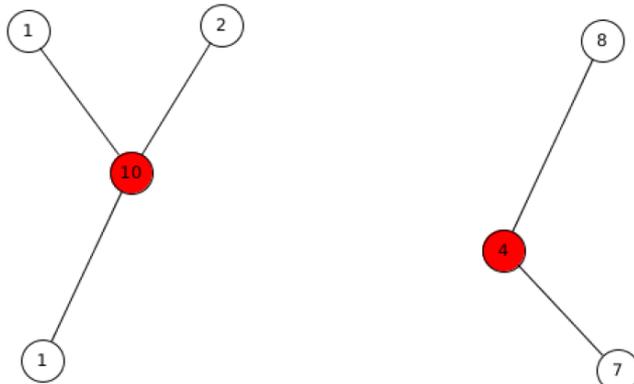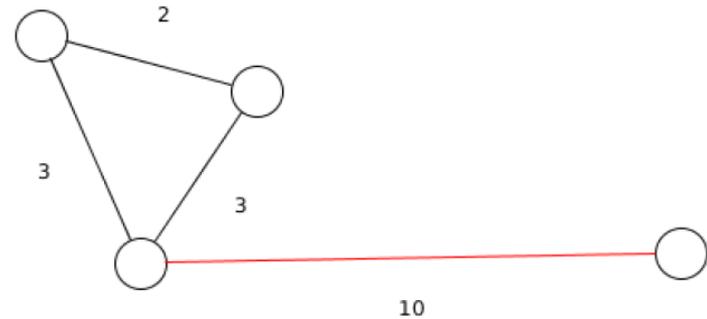
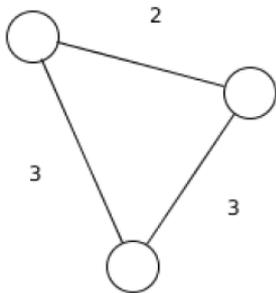The edge connecting the two units is removed
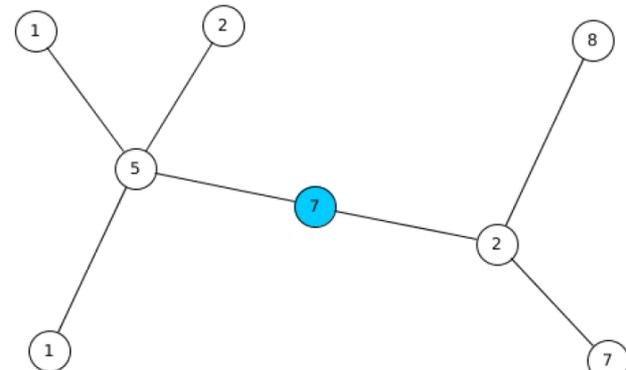
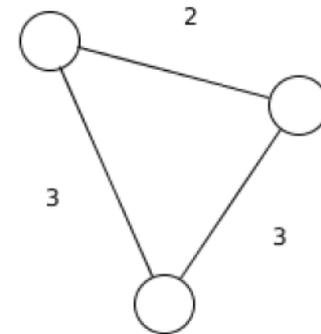A new unit is added

Growing Neural Gas model
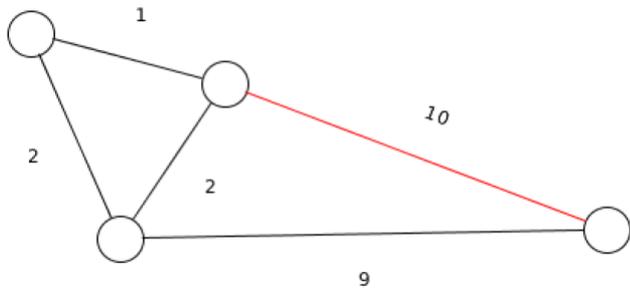
Any edge aged more than 10 is marked for deletion

After ageing another edge must be removed

The unit has no emanating edges
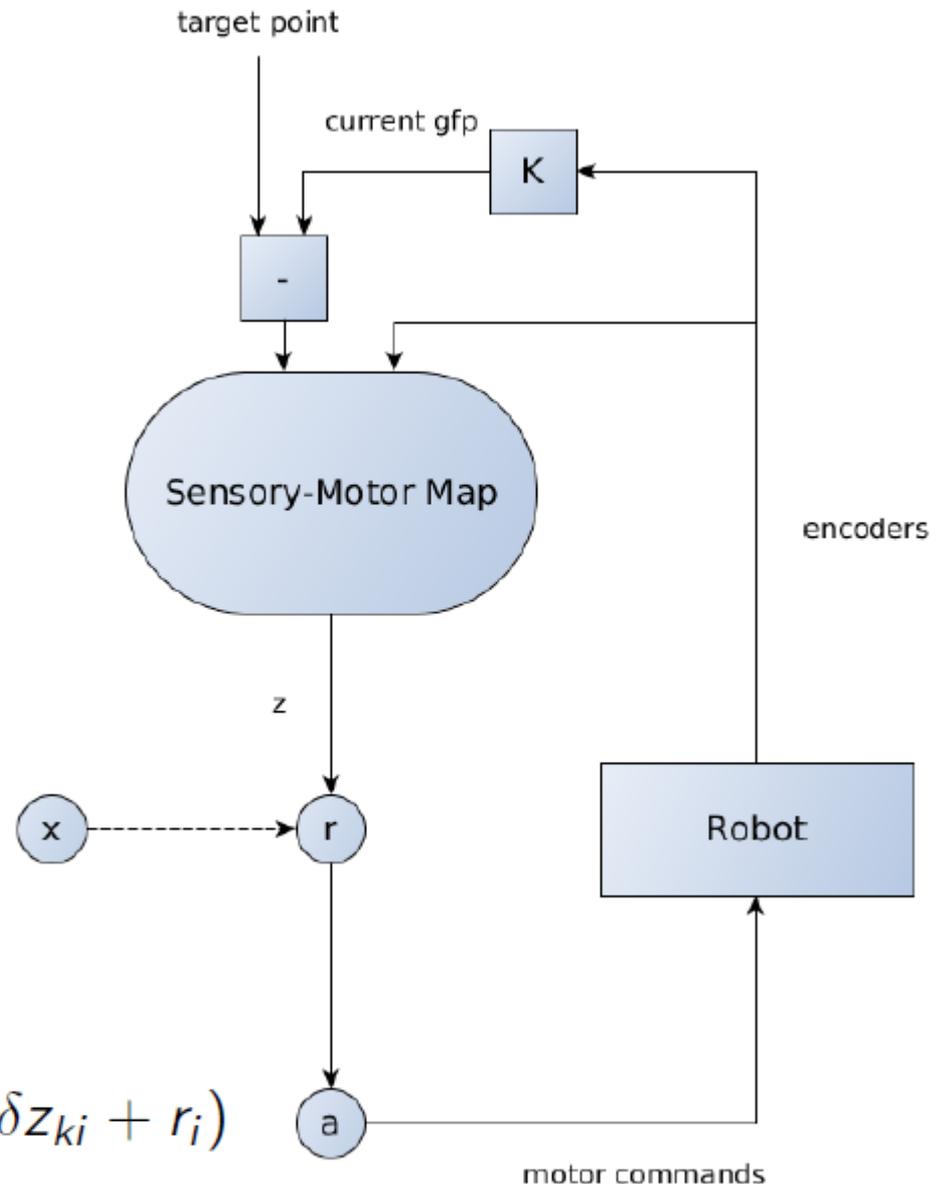
The unit is removed from the network

Learning

Learning is done in two phases

Phase 1: generation of the sensory-motor map, that is training of the GNG independently from the rest of the model

Phase 2: generation of endogenous movements
(motor babbling, x) and
update of the weights z of
the connections starting
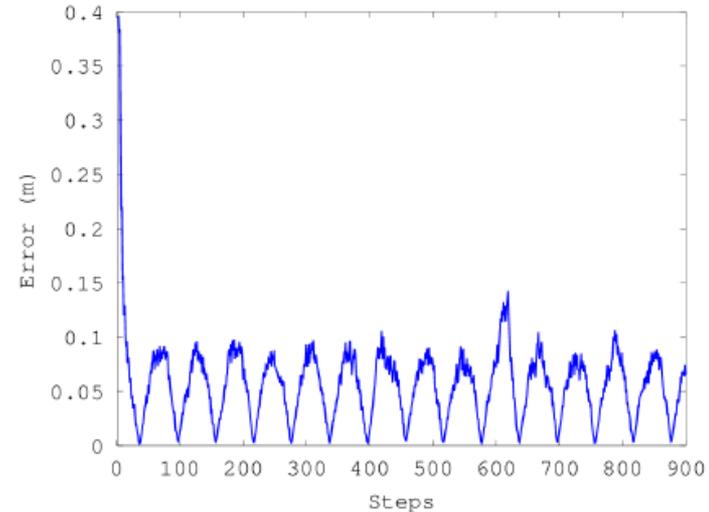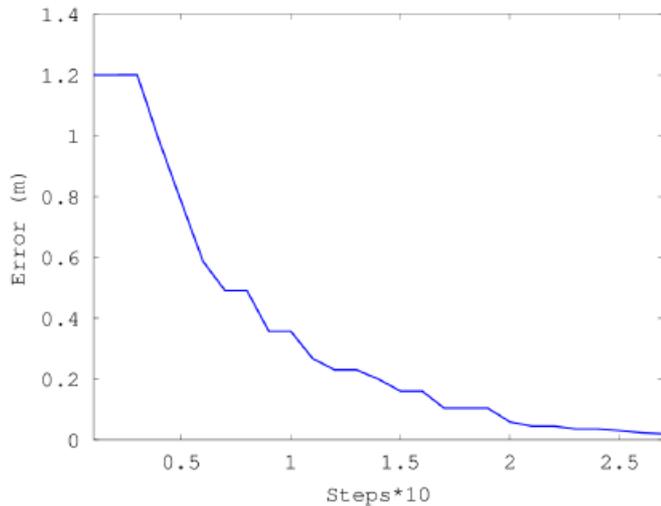from the winning units and
from its neighbors:

$$\frac{\mathrm{d}z_{ki}}{\mathrm{d}t} = \gamma \cdot c_k \cdot (-\delta z_{ki} + r_i)$$

target point

current gfp

K

-

Sensory-Motor Map

encoders

z

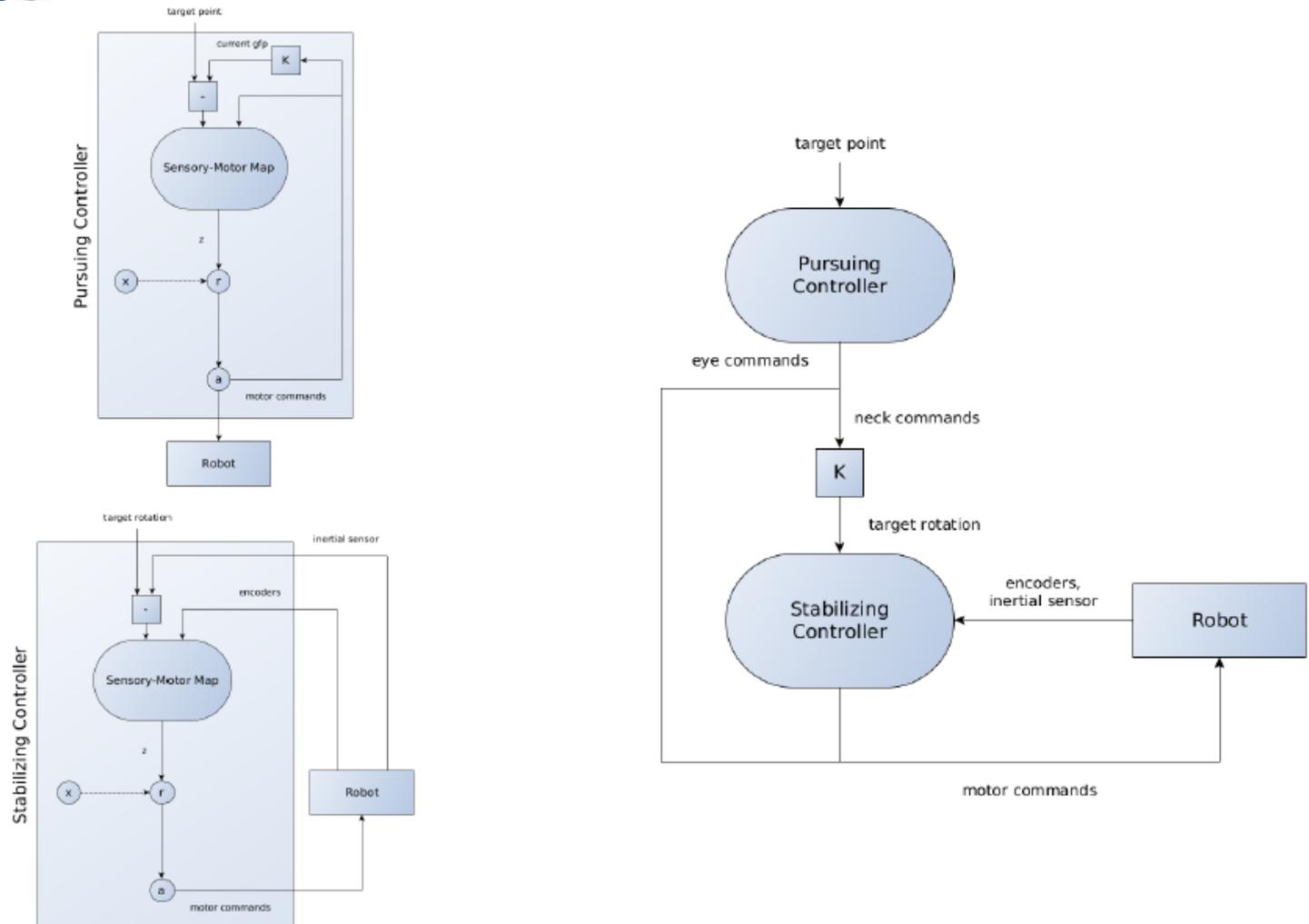x ------> r

a

Robot

motor commands

Results: reaching

✓ The controller is fast enough to follow a moving target. This has been tested by making the target do a sinusoidal movement with an amplitude of 40cm at different frequencies

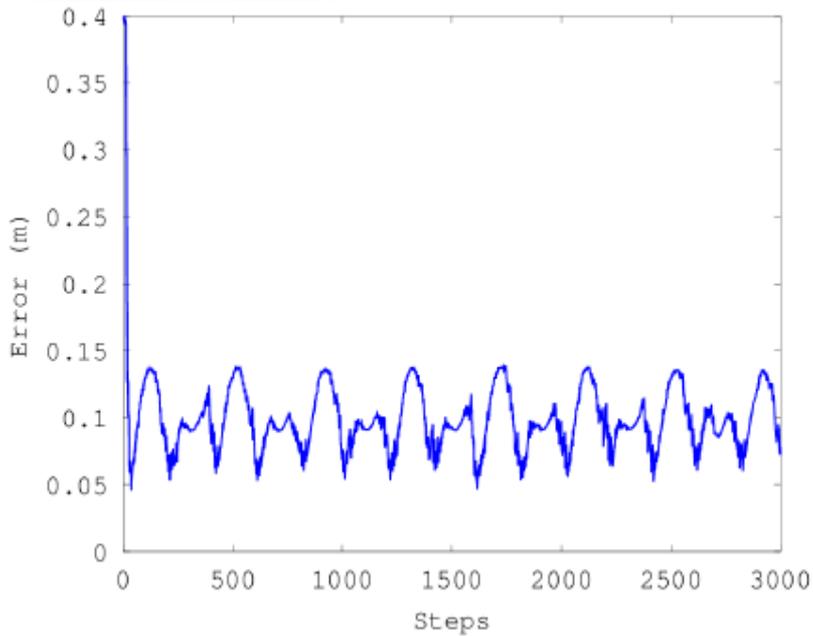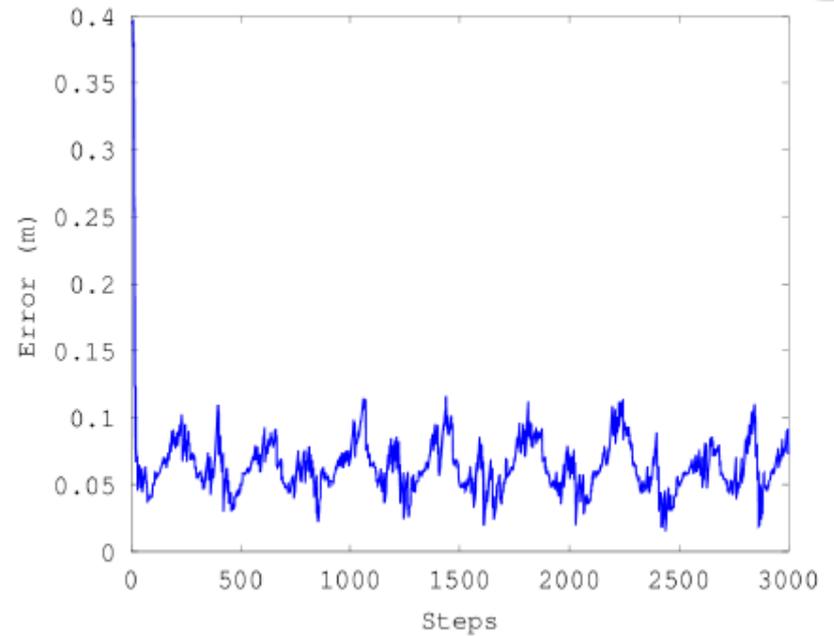| f (Hz)	$\overline{E}$ (m)	E_{max} (m)	E_{min} (m)
0.1	0.022072	0.046741	0.000456
0.25	0.054736	0.142454	0.001733
0.5	0.092294	0.210824	0.002447
1	0.140470	0.274547	0.006962
2	0.160077	0.284362	0.008896

Results: stabilized pursuit

No stabilization:
E=0.0983 m

With stabilization:
E=0.063 m