A Machine Learning approach

to Motor Adaptation

Olivier Sigaud

Université Pierre et Marie Curie, PARIS 6
http://people.isir.upmc.fr/sigaud

September 19, 2014
Learning one’s body

- Babies don’t know well their body
Motor adaptation

- Adapting one’s body model (kinematics, dynamics, ...) under changing circumstances
Introduction

Motor adaptation: standard experiment

- Standard view: Motor adaptation results from learning a model of the dynamics
Interest for robotics

Learning interaction models

- Impossible to model unknown objects
- Studying potential of supervised learning algorithms to control robots in unpredictable situations
- Presenting the tools (statistical learning + control framework) to give a basic account of motor adaptation
A Machine Learning approach

Introduction

Outline

- mechanical models
- model-based control
- adaptive control
- non-linear regression
- research at ISIR
Table of contents

Mechanical models

Control

Learning methods

Learning Robotics models

Results

Perspectives

References
Kinematics: some Mechanics

\[ \xi_x = l_1 \cos(q_1) + l_2 \cos(q_2) + l_3 \cos(q_3) \]
\[ \xi_y = l_1 \sin(q_1) + l_2 \sin(q_2) + l_3 \sin(q_3) \]

\( \xi \): operational position
\( q \): articular position
Velocity kinematics - Jacobian

\[ \nu_x = -l_1 \sin(q_1) \dot{q}_1 - l_2 \sin(q_2) \dot{q}_2 - l_3 \sin(q_3) \dot{q}_3 \]
\[ \nu_y = l_1 \cos(q_1) \dot{q}_1 + l_2 \cos(q_2) \dot{q}_2 + l_3 \cos(q_3) \dot{q}_3 \]
\[ \nu = J(q) \dot{q} \]

\( \dot{q} \): articular velocity
\( \nu \): operational velocity
Dynamics: where forces come into play

Forward and inverse dynamics (Lagrange or Newton-Euler equations)

\[
\ddot{q} = A(q)^{-1} \left( \tau - n(q, \dot{q}) - g(q) - \epsilon(q, \dot{q}) + \tau^{ext} \right)
\]

\[
\tau = A(q) \ddot{q} + n(q, \dot{q}) + g(q) + \epsilon(q, \dot{q}) - \tau^{ext}
\]

- \(A\): inertia matrix
- \(n\): Coriolis and centrifugal effects
- \(g\): gravity
- \(\epsilon\): unmodeled effects
- \(\tau^{ext}\): external forces

- \(q\): articular position
- \(\dot{q}\): articular velocity
- \(\ddot{q}\): articular acceleration
- \(\tau\): torques
Resolve Motion Rate Control (Whitney 1969)

- Also called CLIK (Closed Loop Inverse Kinematics)
- From task to torques
- Three steps architecture
  - Trajectory generation
  - Inverse Kinematics and redundancy
  - Inverse Dynamics
First step, create a goal attractor.

\[ \nu^* = K_p (\xi^\dagger - \xi) \]
A Machine Learning approach

Control

Resolve Motion Rate Control - Trajectory generation

First step, create a goal attractor.

\[ \nu^* = K_p (\xi^* - \xi) \]
A Machine Learning approach

Control

Resolve Motion Rate Control - Inverse kinematics

$q$: articular position

$\nu$: operational velocity

$\dot{q}$: articular velocity

Second step, inverse the kinematics.

$$\nu = J(q) \dot{q} \rightarrow \dot{q}^* = J(q)^+ \nu^*$$
A Machine Learning approach

Control

Resolve Motion Rate Control - Inverse kinematics

$q$: articular position
$\nu$: operational velocity
$\dot{q}$: articular velocity

Second step, inverse the kinematics.

$\nu = J(q) \dot{q} \rightarrow \dot{q}^* = J(q)^+ \nu^*$
Resolve Motion Rate Control - Inverse kinematics

$q$: articular position
$\nu$: operational velocity
$\dot{q}$: articular velocity

Second step, inverse the kinematics.

$\nu = J(q) \dot{q} \rightarrow \dot{q}^* = J(q)^+ \nu^*$
Control redundancy

\[
\dot{q}^* = J(q)^+ \nu^*
\]

\[
\dot{q}^* = J_1(q)^+ \nu_1^* + (J_2(q) P_{J_1})^+ \nu_2^*
\]

- redundancy: more actuated degrees of freedom than those necessary to realise a task
- \( P_J \) is a projector used to control redundancy
- necessary to have access to \( J \) to compute \( P_J \)
Resolve Motion Rate Control - Inverse Dynamics

\[ \Gamma : \text{torques} \]
\[ M : \text{inertia matrix} \]
\[ b : \text{Coriolis and centrifugal effects} \]
\[ g : \text{gravity} \]
\[ \epsilon : \text{unmodeled effects} \]
\[ \Gamma^{ext} : \text{external forces} \]

Third step, compute the inverse dynamics

\[ \tau^{con} = M(q)\ddot{q}^* + b(q, \dot{q}) + g(q) + \epsilon(q, \dot{q}) - \tau^{ext} \]

\[ \tau^{con} = \mathcal{I}D(q, \dot{q}, \ddot{q}^*) \]
A Machine Learning approach
Control

Resolve Motion Rate Control - Inverse Dynamics

\[ \tau^{\text{con}} = M(q) \ddot{q}^* + b(q, \dot{q}) + g(q) + \epsilon(q, \dot{q}) - \tau^{\text{ext}} \]

\[ \tau^{\text{con}} = TD(q, \dot{q}, \ddot{q}^*) \]

\( \Gamma \): torques

\( M \): inertia matrix

\( b \): Coriolis and centrifugal effects

\( g \): gravity

\( \epsilon \): unmodeled effects

\( \Gamma^{\text{ext}} \): external forces
Learning mechanical models by function approximation = regression

- Input: $N$ samples $x_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$,
- Stored in $y = [y_1, \cdots, y_N]$, $X = [x_1, \cdots, x_N]$ (design matrix)
- Output: the latent function $f$ such that $y = f(X)$
Outline of methods

- Linear model
  \[ f(x) = a^T x + b \]
  \[ E = 1 \]
  \[ a = 0 \]

- Mixture of linear models (unified model)
  \[ f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot (a_e^T x + b_e) \]
  \[ \text{sub-models: } a_e^T x + b_e \]
  \[ \text{weights: } \phi(x, \theta_e) \]

- Weighted sum of basis functions
  \[ f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot b_e \]
  \[ \text{sub-models: } \phi(x, \theta_e) \]
  \[ \text{weights: } b_e \]

- LLS (Linear Least Squares)
  (Weighted Linear Least Squares)

- GMR (Gaussian Mixture R.)
- LWR (Locally Weighted R.)
- RFWR (Receptive Field Weighted R.)
- LWPR (Locally Weighted Projection R.)
- XCSF
- M5 (Model Trees)

- RBFN (Radial Basis Function Network)
- KRR (Kernel Ridge R.)
- GPR (Gaussian Process R.)
- iRFRLS
- i-SSGPR
- M5 (Regression Trees)

Figure: Classification. Based only on output representation, not on algorithmic properties.
Least Squares

In the linear case, we get \( f(X) = w^T X \), where \( w \) is a vector of weights.

We obtain

\[
\min_w \| y - wX \|^2 \quad J(w)
\]

Thus \( w^* = (X^T X)^{-1} X^T y \)
A model can be learned incrementally and iteratively from samples

- **Incremental**: the model is improved each time a new data point is received
- **Iterative**: the model $M_{i+1}$ is improved from $M_i$ (e.g. gradient descent)
- **Standard approach**: linear approx. with (Recursive) Least Squares (RLS)
Regularized Least Squares

- Potential singularities in $X^T X$ can generate very large $w^*$ weights
- Regularized Least Squares (Ridge Regression): penalize large weights
- Optimize with lower weights (sacrifice optimality):
  \[
  w^* = \arg \min_w \frac{\lambda}{2} \|w\|^2 + \frac{1}{2} \|y - X^T w\|^2, \tag{1}
  \]
- Analytical solution:
  \[
  w^* = \left(\lambda I + X^T X\right)^{-1} X^T y. \tag{2}
  \]
- Iterative and incremental computation of $(\lambda I + X^T X)^{-1}$ through Cholesky decomposition
From Linear to Non-linear Least Squares: Outline

- Two different approaches:
  - Performing multiple local and weighted least square regressions (shown with LWR)
  - Projecting the input space into a feature space using non-linear basis functions (shown with RBFNs)

- We highlight the similarity between both approaches
- Then we list algorithms from each family
The function to be approximated is \( f(x, u) = |x - x_{\text{target}}|^2 + |u|^2 \)

- We define features \( \phi_i(x, u) \) over \( (x, u) \)
- We look for \( w \) such that \( \hat{f}(x, u) = \sum_i w_i \phi_i(x, u) \)
With poor features

- If we take $\phi_1(x, u) = x$ and $\phi_2(x, u) = u$
- We cannot do better than $\hat{f}(x, u) = w_1 x + w_2 u$
- Very poor linear approximation
A Machine Learning approach
Learning methods
Basis Function Network Methods

With good features

If we take $\phi_1(x, u) = |x - x_{target}|^2$ and $\phi_2(x, u) = |u|^2$

Then $\hat{f}(x, u) = w_1 \phi_1(x, u) + w_2 \phi_2(x, u) \rightarrow w_1 = 1$ and $w_2 = 1$

Perfect approximation

Finding good features is critical
Standard features: Gaussian basis functions

- The more features, the better the approximation
Kernel Ridge Regression (KRR)

- Define features with a kernel function $k(x, x_i)$ per point $x_i$
- Define the Gram matrix as a kernel matrix:

$$K(X, X) =
\begin{bmatrix}
  k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_N) \\
  k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_N) \\
  \vdots & \vdots & \ddots & \vdots \\
  k(x_N, x_1) & k(x_N, x_2) & \cdots & k(x_N, x_N)
\end{bmatrix}. \quad (3)$$

- Computing the weights is done with ridge regression using

$$w^* = (\lambda I + K(X, X))^{-1} y, \quad (4)$$

- The kernel matrix $K$ grows with the number of points (kernel expansion)
- The matrix inversion may become too expensive
- Solution: finite set of features (RBFNs), incremental methods
Gaussian Process Regression (GPR)

- Predicting $y^*$ for a novel input $x^*$ is done by assuming that the novel output are $y^*$ also sampled from a multi-variate Gaussian with
  \[
  \begin{bmatrix} y \\ y^* \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} k(x_*, X) \\ k(x_*, X) k(x_*, x_*)^T \end{bmatrix}, \begin{bmatrix} k(x_*, x_1) & \ldots & k(x_*, x_n) \end{bmatrix} \right),
  \]

- The best estimate for $y_*$ is the mean, and the uncertainty in $y_*$ is captured by the variance as
  \[
  \bar{y}_* = k(x_*, X)K(X, X)^{-1}y
  \]
  \[
  \text{var}(y_*) = k(x_*, x_*) - k(x_*, X)K(X, X)^{-1}K(X, X)^T.
  \]
GPR ~ KRR

- When computing the mean $\bar{y}_*$, $K(X, X)$ and $y$ depend only on the training data, not the novel input $x^*$. Therefore, $K(X, X)^{-1}y$ can be compacted in one weight vector, which does not depend on the query $x^*$. We call this vector $w^*$ and we get

$$w^* = K(X, X)^{-1}y,$$  \hspace{1cm} (7)

- We can rewrite (5) as follows:

$$\bar{y}_* = k(x^*, X)K(X, X)^{-1}y$$
$$= [k(x^*, x_1), k(x^*, x_N)] \cdot w^*$$
$$= \sum_{n=1}^{N} w_n \cdot k(x^*, x_n).$$ \hspace{1cm} (9)

The mean of the gpr is the same weighted sum of basis functions as in krr, and (10) has the same form as the unified representation in (20).

- krr computes a regularized version of the weights computed by gpr, with an additional regularization parameter $\lambda$. 
Radial Basis Function Networks: definition and solution

- We define a set of $E$ basis functions (often Gaussian)

$$f(x) = \sum_{e=1}^{E} w_e \cdot \phi(x, \theta_e)$$  \hspace{1cm} (11)

$$= w^\top \cdot \phi(x).$$  \hspace{1cm} (12)

- We also define the Gram matrix

$$\Theta = \begin{bmatrix}
\phi(x_1, \theta_1) & \phi(x_1, \theta_2) & \cdots & \phi(x_1, \theta_E) \\
\phi(x_2, \theta_1) & \phi(x_2, \theta_2) & \cdots & \phi(x_2, \theta_E) \\
\vdots & \vdots & \ddots & \vdots \\
\phi(x_N, \theta_1) & \phi(x_N, \theta_2) & \cdots & \phi(x_N, \theta_E)
\end{bmatrix}$$  \hspace{1cm} (13)

- and we get the least squares solution

$$w^* = (\Theta^\top \Theta)^{-1} \Theta^\top y.$$  \hspace{1cm} (14)
Radial Basis Function Networks: computation

- Solving $w^* = (\Theta^T\Theta)^{-1}\Theta^Ty$ requires inverting $(\Theta^T\Theta)$
- That is cubic in the number of points
- Complexity can be reduced to $O(N^2)$ by using the Sherman-Morrisson formula, giving rise to an incremental update of the inverse, but this method is sensitive to rounding errors. A numerically more stable option consists in updating the Cholesky factor of the matrix using the qr algorithm.
- Other approaches: gradient descent on weights, Recursive Least Squares...
A Machine Learning approach
- Learning methods
  - Basis Function Network Methods

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

A Machine Learning approach

Learning methods

Basis Function Network Methods

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

A Machine Learning approach

Learning methods

Basis Function Network Methods

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

A Machine Learning approach
- Learning methods
  - Basis Function Network Methods

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

A Machine Learning approach

- Learning methods
- Basis Function Network Methods

Radial Basis Function Networks (Illustration)

- Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

Instead of matrix inversion, use some incremental/iterative approach (RLS, gradient descent...)

A Machine Learning approach
Learning methods
Basis Function Network Methods

Incremental Receptive Fields Regularized Least Squares

Approximate the function through its (approximate) Fourier transform using random features
\[ z_k(X_i) = \frac{\sqrt{2}}{\sqrt{D}} \cos(\omega_k^T X_i + b_k), \]
with \( \omega_k \sim \mathcal{N}(0, 2\gamma I) \) and \( b_k \sim \mathcal{U}(0, 2\pi) \).

As RBFNs, but with \( K \) cosinus features \( \rightarrow \) global versus local

Provides a strong grip against over-fitting (ignoring the high frequencies)

In practice, efficient for large enough \( K \), and easy to tune

I-SSGPR: same tricks based on GPR

Least Square computation: summary

- **Linear case**

  \[ w^* = (X^T X)^{-1} X^T y \]  
  \[ w^* = (\lambda I + X^T X)^{-1} X^T y. \]  
  (regularized)  

- **Kernel matrix case**

  \[ w^* = K^{-1} y, \]  
  \[ w^* = (\lambda I + K)^{-1} y. \]  
  (regularized)
## Basis Function Networks: summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Regularized?</th>
<th>Number of BFs?</th>
<th>Features?</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbfm</td>
<td>Yes</td>
<td>$E$</td>
<td>RBFs</td>
</tr>
<tr>
<td>krr</td>
<td>Yes</td>
<td>$N$</td>
<td>kernels</td>
</tr>
<tr>
<td>gpr</td>
<td>No</td>
<td>$N$</td>
<td>kernels</td>
</tr>
<tr>
<td>irfrls</td>
<td>Yes</td>
<td>$E$</td>
<td>cosine</td>
</tr>
<tr>
<td>i-ssgpr</td>
<td>Yes</td>
<td>$E$</td>
<td>cosine</td>
</tr>
</tbody>
</table>

**Table:** Design of all weighted basis function algorithms.
Locally Weighted Regression

Figure: The thickness of the lines indicates the weights.

- Linear models are tuned with Least Squares
- Their importance is represented by a Gaussian function

**A Machine Learning approach**

**Learning methods**

**Locally Weighted Regression Methods**

**LWR approximation: graphical intuition**

\[ \phi(x, \theta_e) = e^{-\frac{1}{2}(x-c_e)^T \Sigma_e^{-1}(x-c_e)} \]

- Each RF tunes a local linear model
  \[ \Psi_e(x) = a_e^T x + b_e \]

- Gaussians tell you how much each RF contributes to the output
  \[ \tilde{y}(x) = \frac{\sum_{e=1}^E \phi(x, \theta_e) \Psi_e(x)}{\sum_{e=1}^E \phi(x, \theta_e)} \]

- The global output (green line) is a weighted combination of linear models (straight lines)
LWPR: general goal

- Non-linear function approximation in very large spaces
- Using PLS to project linear models in a smaller space
- Good along local trajectories

XCSF: overview

- XCSF is a Learning Classifier System [Holland, 1975]
- Linear models weighted by Gaussian functions (similar to LWPR)
- Linear models are updated using RLS
- Gaussian functions adaptation: $\Sigma_e^{-1}$ and $c_e$ are updated using a GA
- Key feature: distinguish Gaussian weights space and linear models space (example: $x = \langle q, \dot{q} \rangle$)

LWPR:
$$f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot (b_e + a^\top_e x)$$

XCSF:
$$f(x) = \sum_{e=1}^{E} \phi(q, \theta_e) \cdot (b_e + a^\top_e \dot{q})$$

- Condensation: reduce population to generalize better
A Machine Learning approach

- Learning methods
- Locally Weighted Regression Methods

GMR

\[
y_{new} = \sum_{k=1}^{K} h_k(X_{new})(\mu_k,Y + \Sigma_{k,YX} \Sigma_{k,Y}^{-1}(X_{new} - \mu_k,X))
\]

With

\[
\mu_k = [\mu_k^T, \mu_k^T]^T \quad \text{and} \quad \Sigma_k = \begin{pmatrix} \Sigma_{k,X} & \Sigma_{k,XY} \\ \Sigma_{k,YX} & \Sigma_{k,YY} \end{pmatrix}
\]

LWR methods: main features

<table>
<thead>
<tr>
<th>Algo</th>
<th>LWR</th>
<th>LWPR</th>
<th>GMR</th>
<th>XCSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of RFs</td>
<td>fixed</td>
<td>growing</td>
<td>fixed</td>
<td>adaptive</td>
</tr>
<tr>
<td>Position of RFs</td>
<td>fixed</td>
<td>fixed</td>
<td>adaptive</td>
<td>adaptive</td>
</tr>
<tr>
<td>Size of RFs</td>
<td>fixed</td>
<td>adaptive</td>
<td>adaptive</td>
<td>adaptive</td>
</tr>
</tbody>
</table>
LWR versus RBFNs

\[ f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot (b_e + a_e \mathbf{x}) \]  \hspace{1cm} (19)

\[ f(x) = \sum_{e=1}^{E} \phi(x, \theta_e) \cdot w_e, \]  \hspace{1cm} (20)

- Eq. (20) is a special case of (19) with \( a_e = 0 \) and \( b_e = w_e \).
- RBFNs: performs one LS computation in a projected space
- LWR: performs many LS computation in local domains
Take home message

- Basis Function Networks vs Mixture of linear models
- **LWPR**: PLS, fast implementation, the reference method
- **XCSF**: distinguish gaussian weights space and linear models space
- **GMR**: few features
- **ISSGPR**: easy tuning, no over-fitting
- See tutorial paper

A Machine Learning approach
Learning Robotics models

Learning mechanical models

- Forward kinematics: \( \dot{\xi} = F_\theta(q, \dot{q}) \)  \( \dot{\xi} = J(q) \dot{q} \)
- Forward dynamics: \( \ddot{q} = G_\theta(q, \dot{q}, \Gamma) \)  \( \ddot{q} = A(q)^{-1}(\Gamma - n(q, \dot{q})) \)

- Regression methods can approximate such functions
- The mapping can be learned incrementally from samples
- Can be used for interaction with unknown objects or users
A Machine Learning approach
Learning Robotics models

Learning inverse kinematics with LWPR

- The model is learned with random movements along an operational trajectory
- Input dimension: $\text{dim}(\xi + q) = 29$
- Output dimension: $\text{dim}(\dot{q}) = 26$

Learning forward/inverse velocity kinematics with LWPR

- Learning inverse kinematics is conceptually simpler
- But one loses the opportunity to make profit of redundancy
- Rather learn forward kinematics and inverse it
Learning forward velocity kinematics with LWPR/XCSF

Forward kinematics with LWPR
- Learning the forward velocity kinematics of a Kuka kr16 in simulation.
- They add a constraint to inverse the kinematics and determine the joint velocities.

Learning dynamics with XCSF

- Learning dynamics is more difficult
- In dynamics, there is no redundancy
- The dynamics model is 2/3 smaller with XCSF than with LWPR
Learning inverse dynamics with LWPR

- The model is learned along an operational trajectory
- Input dimension: $\text{dim}(q + \dot{q} + \ddot{q}) = 90$
- Output dimension: $\text{dim}(\Gamma) = 30$
- 7.5$\times$10$^6$ training data points and 2200 receptive fields

Learning inverse dynamics

\[ \ddot{q} = A(q)^{-1} \left( \tau - \eta(q, \dot{q}) - g(q) - \epsilon(q, \dot{q}) + \tau^{\text{ext}} \right) \]

Learning inverse dynamics with random movements along a trajectory

Predict inverse dynamics
Peters and Schaal (2008) learn inverse dynamics in the operational space.

The model is learned along an operational trajectory.

Input dimension: \( \dim(q + \dot{q} + \nu) = 17 \)

Output dimension: \( \dim(\Gamma) = 7 \)

Optimal control with dynamics learned with LWPR

- The inverse dynamics model is learned in the whole space.
- Input dimension: $\text{dim}(q + \dot{q} + u) = 10$. Output dimension: $\text{dim}(\ddot{q}) = 2$. 
- $1,2 \times 10^6$ training data points and 852 receptive fields
- Learning a model of redundant actuation

Properties of models

- [D’Souza et al., 2001b], [Vijayakumar et al., 2005] and [Peters and Schaal, 2008] learn kinematics and dynamics along a trajectory.

- [Butz et al., 2009] learn kinematics in the whole space but do not make profit of redundancy to combine several tasks.

- [Mitrovic et al., 2008] learn dynamics in the whole space to control redundant actuators.
Camille Salaün’s work: combining tasks

To perform several tasks with learnt models, we have chosen to

- learn separately forward kinematics and inverse dynamics
- use classical mathematical inversion to resolve redundancy
- learn models on whole space
- use LWPR and XCSF as learning algorithms
Learning kinematics with LWPR

500 steps babbling with the kinematics model we want to learn.
Controlling redundancy with LWPR

compatible task

incompatible task
Learning kinematics of iCub in simulation

- Simulation of a three degrees of freedom shoulder plus one degrees of freedom elbow
Learning kinematics on the real robot

iCub realising two tasks: following a circle and clicking a numpad
Inverse dynamics and motor adaptation

Applying a vertical force after 2 seconds during a point to point task.
Inverse dynamics and after effects

Releasing the force after 2 seconds during a point to point task.

- We reproduce Shadmehr’s experiments
Learning dynamics

- Simulation of a three degrees of freedom planar arm
Learning forward models

For complex robots, the CAD model is not so accurate (calibration issue)

Comparing algorithms

- Main difficulty: tuning parameters for fair comparison
- Many specific difficulties for robotics reproducibility

Motor adaptation and the cerebellum

- Structural similarity between LWPR-like algos and cerebellum: Purkinje Cells = receptive fields
- + the problem of state estimation over time given delays
Learning dynamical interactions with objects

- Using a force/torque sensor to detect exerted force on shoulder
- Using artificial skin to detect contact points
- Compliant control of motion (CODYCO EU project)
- Learning high-dimensional models
Any question?
Using locally weighted regression for robot learning.

Learning sensorimotor control structures with XCSF: redundancy exploitation and dynamic control.
In Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pages 1171–1178. ACM.

Approximation by superpositions of a sigmoidal function.

Autonomous online learning of velocity kinematics on the icub: a comparative study.

Learning inverse kinematics.

Learning inverse kinematics.

Gaussian processes for regression: A quick introduction.

Incremental learning of robot dynamics using random features.  

Dynamical system modulation for robot learning via kinesthetic demonstrations.  

*Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*.  
University of Michigan Press, Ann Arbor, MI.

Adaptive optimal control for redundantly actuated arms.  

Learning to control in operational space.  

Scalable techniques from nonparametric statistics for real time robot learning.  

Learning the velocity kinematics of icub for model-based control: XCSF versus LWPR.  

On-line regression algorithms for learning mechanical models of robots: a survey.  
References

LWPR: A scalable method for incremental online learning in high dimensions.

Function approximation with a classifier system.
In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pages 974–981,
San Francisco, California, USA. Morgan Kaufmann.