Regression for Robotics
and Motor Adaptation

Olivier Sigaud

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

September 18, 2018

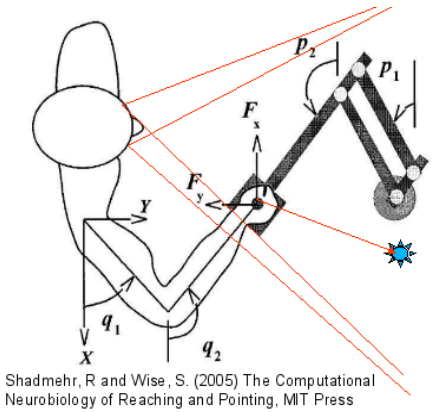
Learning one's body

- Babies don't know well their body

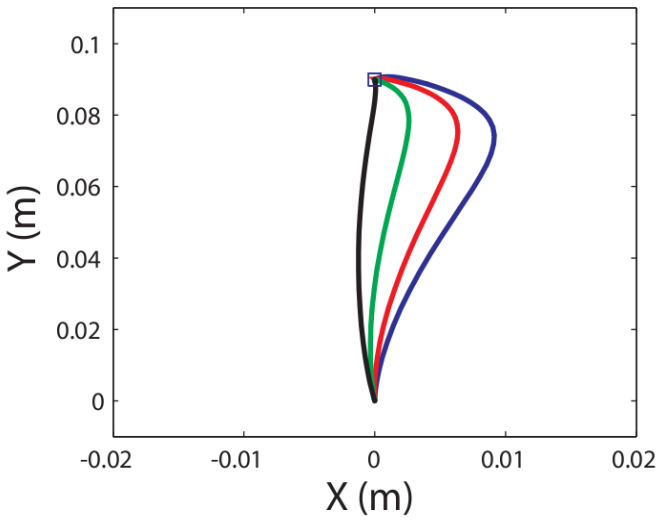
Motor adaptation

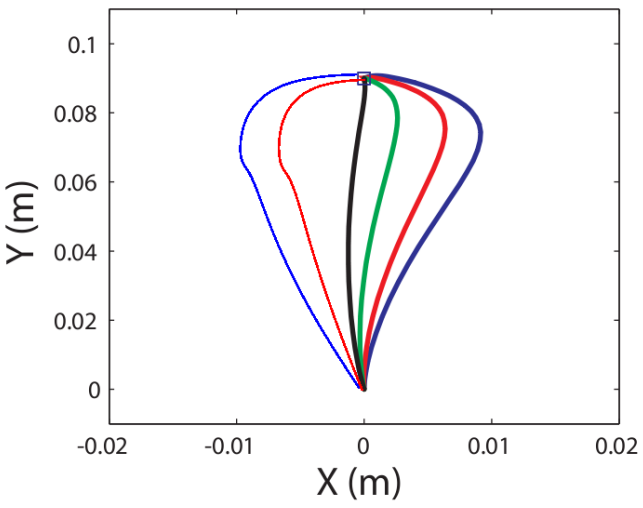
- Adapting one's body model (kinematics, dynamics, ...) under changing circumstances

Motor adaptation: standard experiment



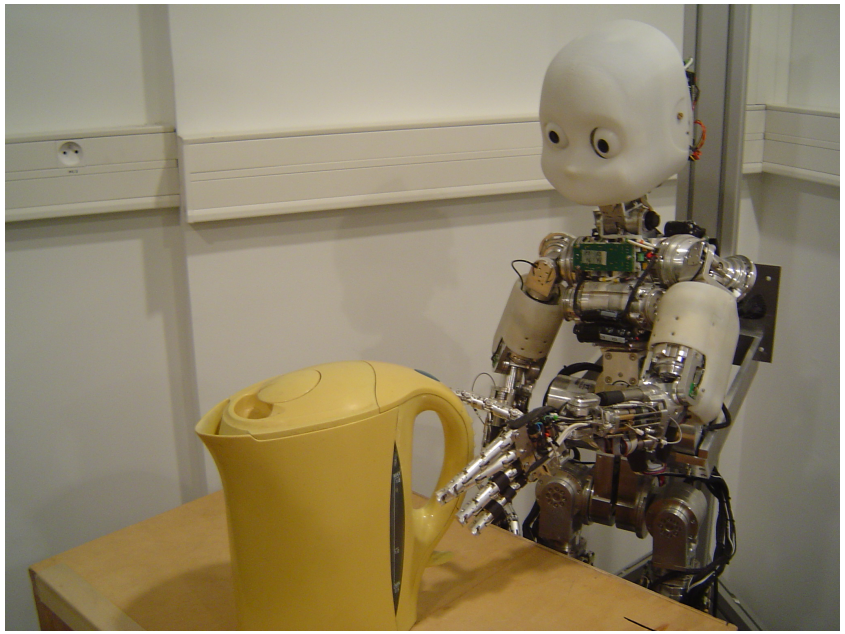
Shadmehr, R and Wise, S. (2005) The Computational Neurobiology of Reaching and Pointing, MIT Press





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

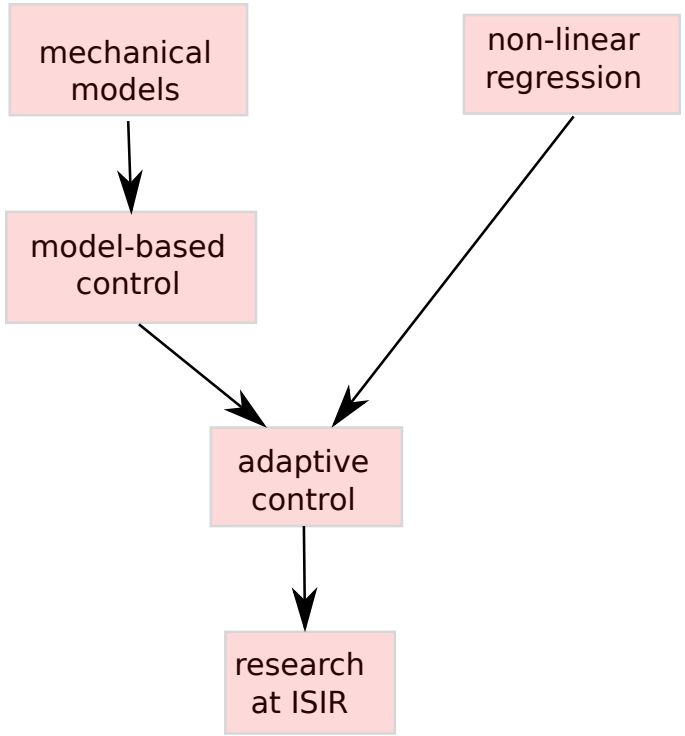
Interest for robotics



Learning interaction models

- Impossible to model unknown objects

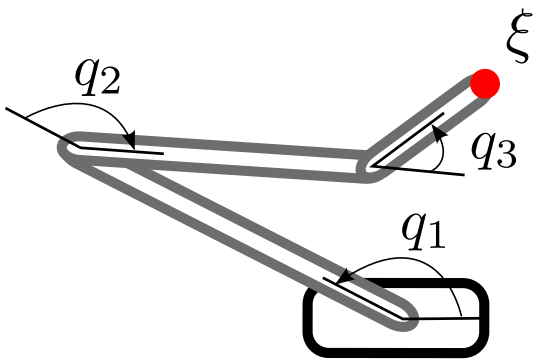
Outline

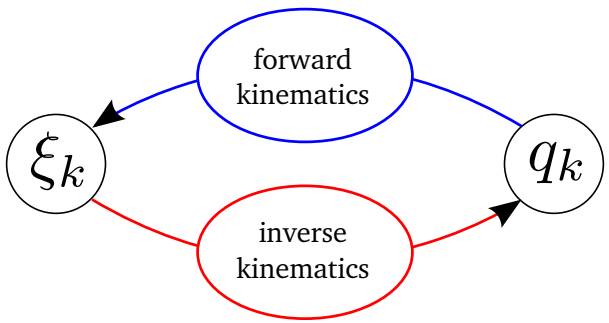


- Tools (regression + control framework) to give a basic account of motor adaptation
- Quick recap on robotics model and control
- Tour of regression algorithms
- Applications

Kinematics

$\boldsymbol{\xi}$: operational position
$\boldsymbol{q}$: articular position

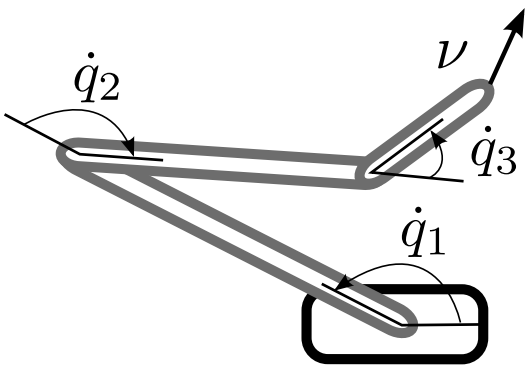


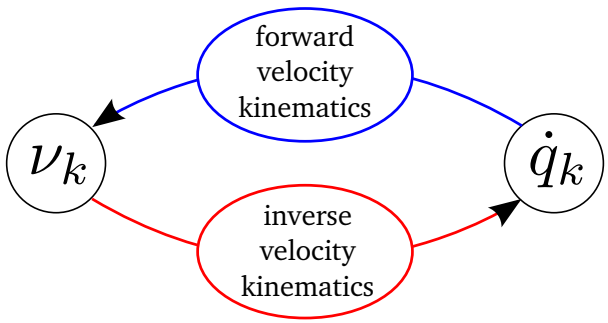


$$\xi_x = l_1 cos(q_1) + l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3)$$
$$\xi_y = l_1 sin(q_1) + l_2 sin(q_1 + q_2) + l_3 sin(q_1 + q_2 + q_3)$$

Velocity kinematics - Jacobian

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



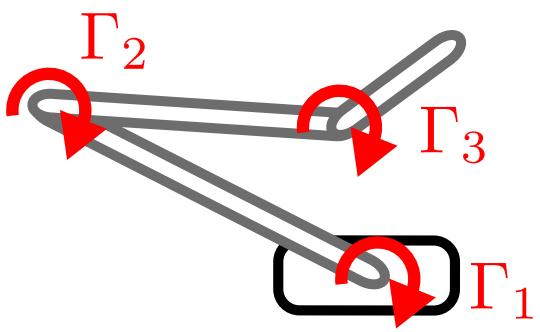


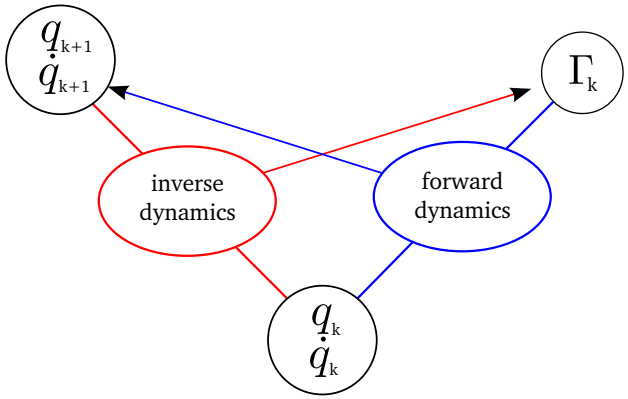
$\nu_x = -(l_1 sin(q_1) + l_2 sin(q_1 + q_2) + l_3 sin(q_1 + q_2 + q_3))\dot{q}_1 - (l_2 sin(q_1 + q_2) + l_3 sin(q_1 + q_2 + q_3))\dot{q}_2 - l_3 sin(q_1 + q_2 + q_3)\dot{q}_3$

$\nu_y = (l_1 cos(q_1) + l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3))\dot{q}_1 + (l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3))\dot{q}_2 + l_3 cos(q_1 + q_2 + q_3)\dot{q}_3$

$\boldsymbol{\nu} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}}$

Dynamics: where forces come into play





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

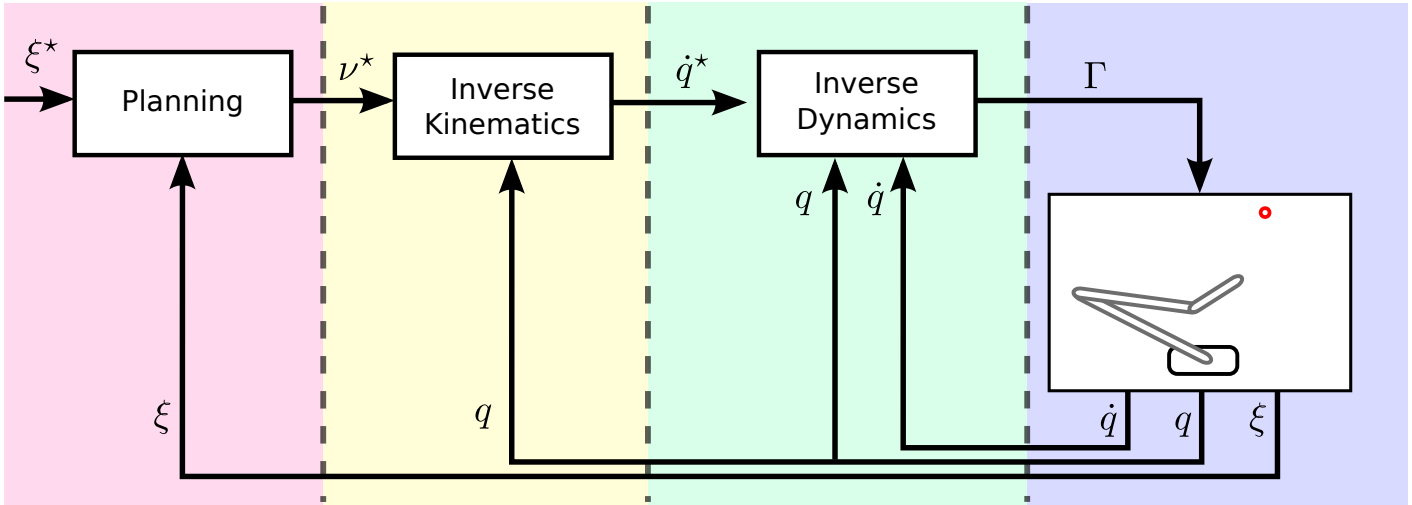
$$\ddot{\boldsymbol{q}} = A(\boldsymbol{q})^{-1}\left(\boldsymbol{\tau} - \boldsymbol{n}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{g}(\boldsymbol{q}) - \boldsymbol{\epsilon}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{\tau}^{ext}\right)$$
$$\boldsymbol{\tau} = A(\boldsymbol{q})\,\ddot{\boldsymbol{q}}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

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

- $\boldsymbol{q}$: articular position
- $\dot{\boldsymbol{q}}$: articular velocity
- $\dot{\boldsymbol{q}}$: articular acceleration
- $\boldsymbol{\tau}$: torques

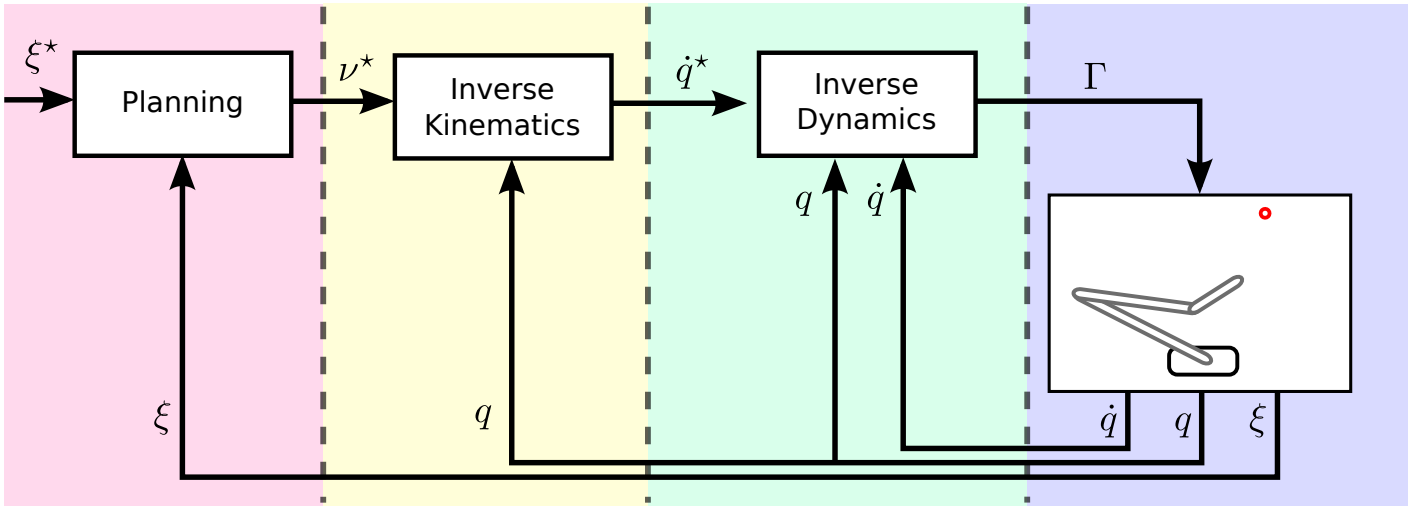
ISIR INSTITUT DES SYSTÈMES INTELLIGENTS ET DE ROBOTIQUE

Resolved 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

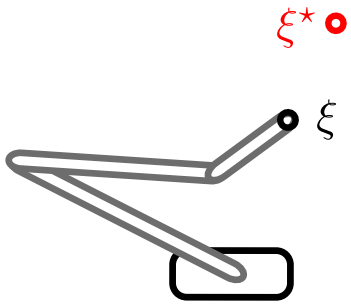
Resolve Motion Rate Control - Trajectory generation

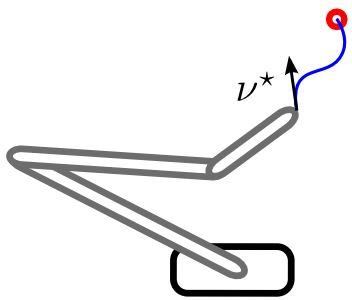


ξ: operational position

$\xi^\dagger$: desired operational position

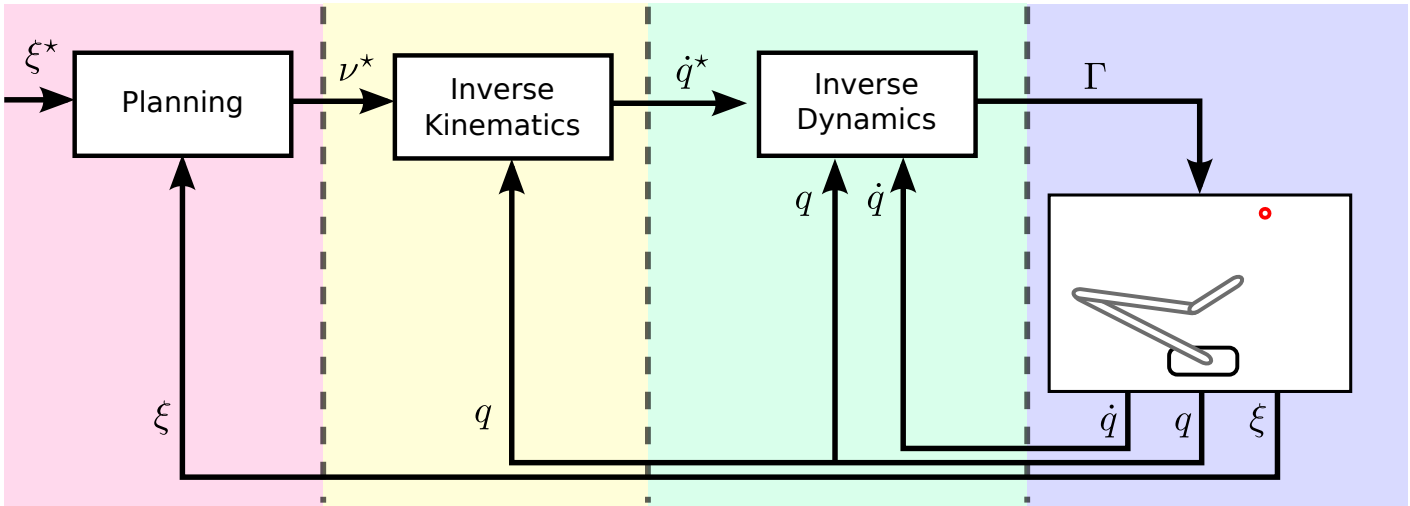
$\nu^\star$: desired operational velocity





First step, create a goal attractor. $\nu^\star = K_p \left(\xi^\dagger - \xi\right)$

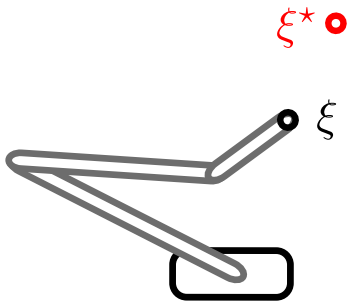
Resolve Motion Rate Control - Trajectory generation

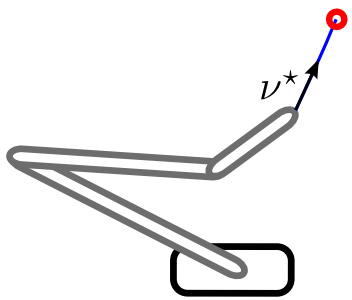


ξ: operational position

$\xi^\dagger$: desired operational position

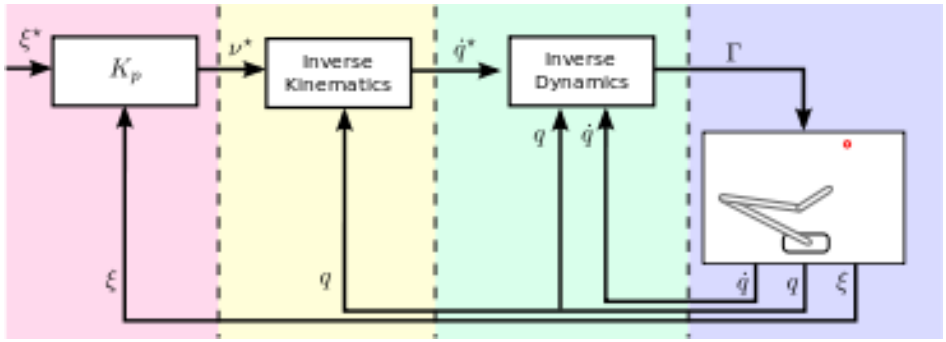
$\nu^\star$: desired operational velocity





First step, create a goal attractor. $\nu^\star = K_p \left(\xi^\dagger - \xi\right)$

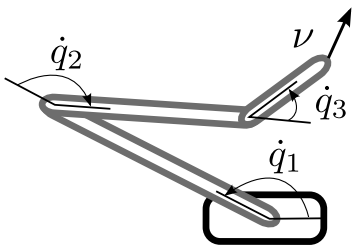
Resolve Motion Rate Control - Inverse kinematics



q: articular position

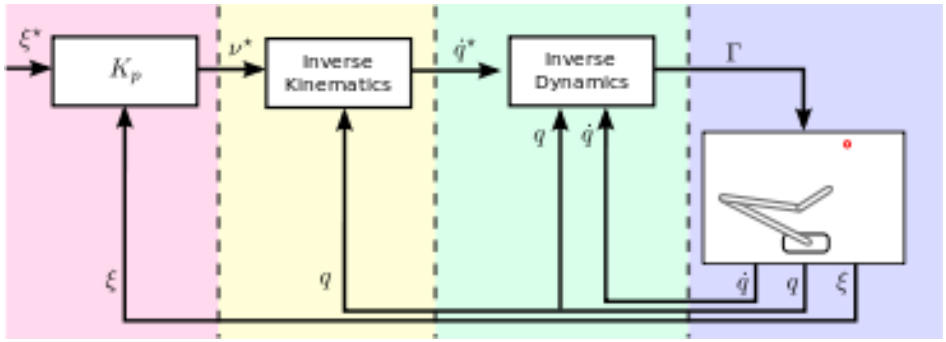
ν: operational velocity

$\dot{q}$: articular velocity



Second step, inverse the kinematics. $\boldsymbol{\nu} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}} \rightarrow \dot{\boldsymbol{q}}^\star = J(\boldsymbol{q})^+\,\boldsymbol{\nu}^\star$

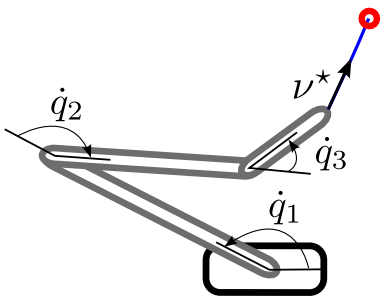
Resolve Motion Rate Control - Inverse kinematics



q: articular position

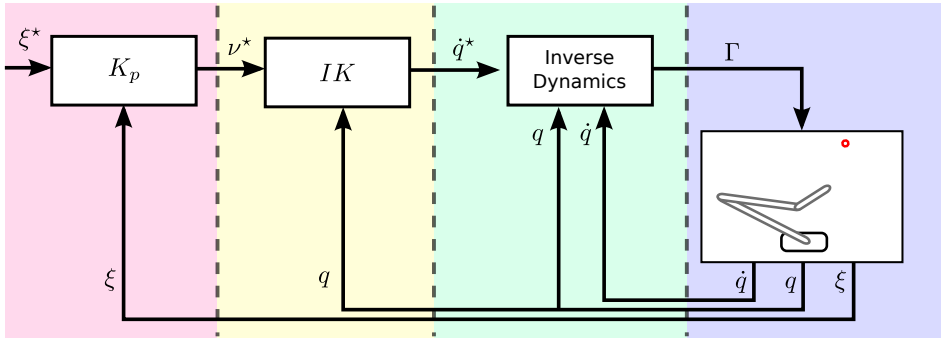
ν: operational velocity

$\dot{q}$: articular velocity



Second step, inverse the kinematics. $\boldsymbol{\nu} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}} \rightarrow \dot{\boldsymbol{q}}^\star = J(\boldsymbol{q})^+\,\boldsymbol{\nu}^\star$

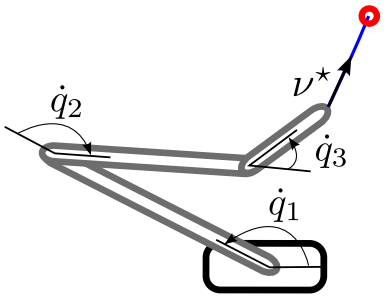
Resolve Motion Rate Control - Inverse kinematics



q: articular position

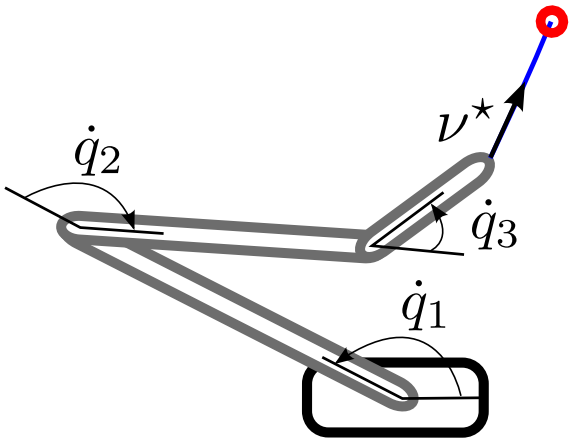
ν: operational velocity

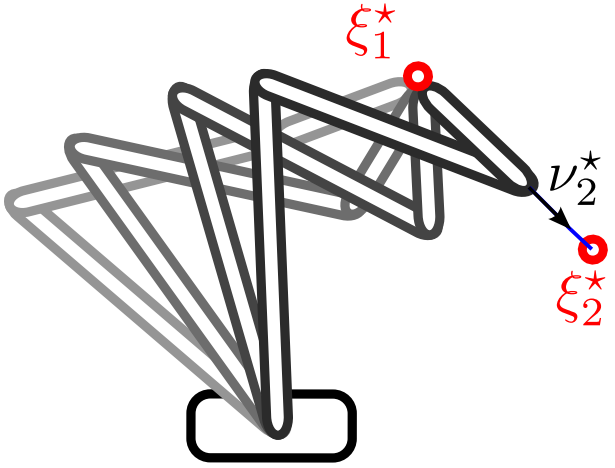
$\dot{q}$: articular velocity



Second step, inverse the kinematics. $\boldsymbol{\nu} = J(\boldsymbol{q})\,\dot{\boldsymbol{q}} \rightarrow \dot{\boldsymbol{q}}^\star = J(\boldsymbol{q})^+\,\boldsymbol{\nu}^\star$

Control redundancy

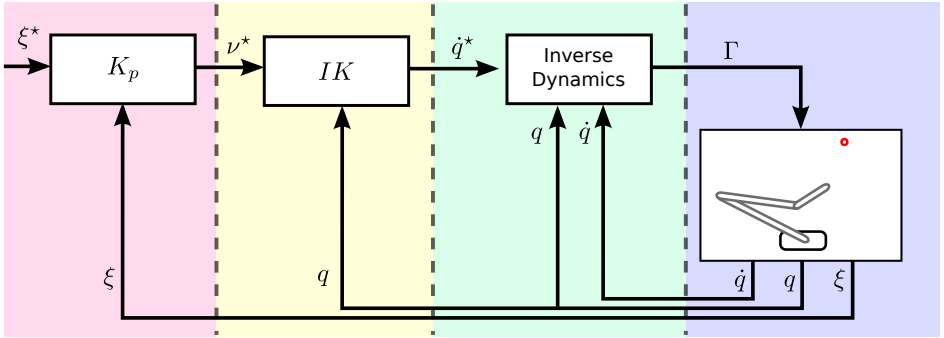




$$\dot{\boldsymbol{q}}^\star = J\left(\boldsymbol{q}\right)^+ \boldsymbol{\nu}^\star \qquad \dot{\boldsymbol{q}}^\star = J_1\left(\boldsymbol{q}\right)^+ \boldsymbol{\nu}_1^\star + \left(J_2\left(\boldsymbol{q}\right) P_{J_1}\right)^+ \boldsymbol{\nu}_2^\star$$

- 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



Γ: torques

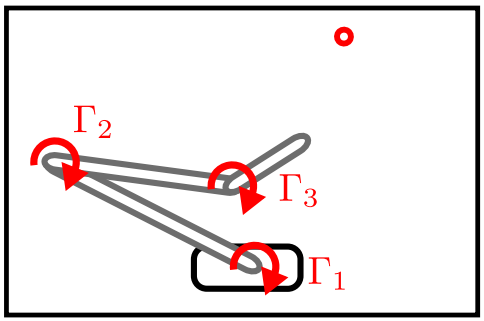
M: inertia matrix

b: Coriolis and centrifugal effects

g: gravity

ϵ: unmodeled effects

Γ^{ext}: external forces

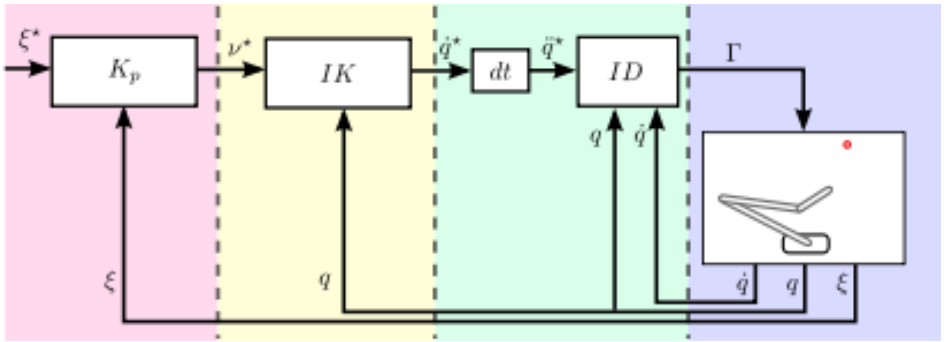


Third step, compute the inverse dynamics

$$\boldsymbol{\tau^{con}} = M(\boldsymbol{q})\,\ddot{\boldsymbol{q}}^\star + \boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\epsilon}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{\tau}^{ext}$$

$$\boldsymbol{\tau^{con}} = \mathcal{ID}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}^\star)$$

Resolve Motion Rate Control - Inverse Dynamics



Γ: torques

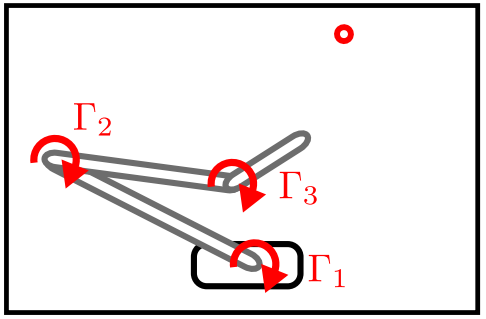
M: inertia matrix

b: Coriolis and centrifugal effects

g: gravity

ϵ: unmodeled effects

Γ^{ext}: external forces



Third step, compute the inverse dynamics

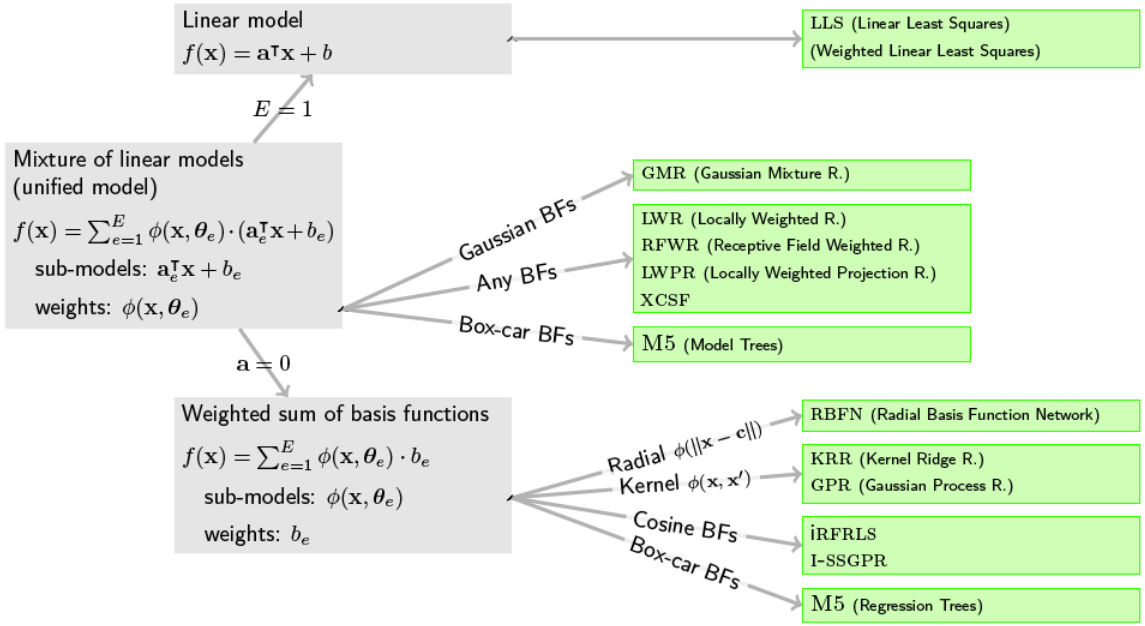
$$\boldsymbol{\tau^{con}} = M(\boldsymbol{q})\,\ddot{\boldsymbol{q}}^\star + \boldsymbol{b}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\epsilon}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{\tau}^{ext}$$

$$\boldsymbol{\tau^{con}} = \mathcal{ID}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}^\star)$$

Questions

- For a redundant robot, there is an infinity of inverse dynamical models Right or Wrong ?
- For a redundant robot, there is an infinity of forward kinematics models Right or Wrong ?

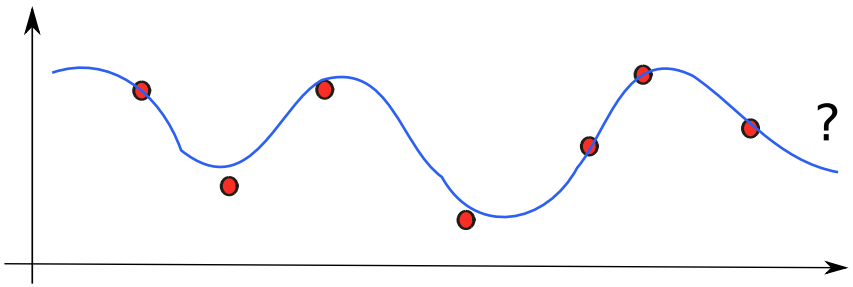
Outline of methods



- Two different approaches:
  - Projecting the input space into a feature space using non-linear basis functions (shown with RBFNs)
  - Multiple local and weighted least square regressions (shown with LWR)
- We highlight the similarity between both approaches
- Then we list algorithms from each family

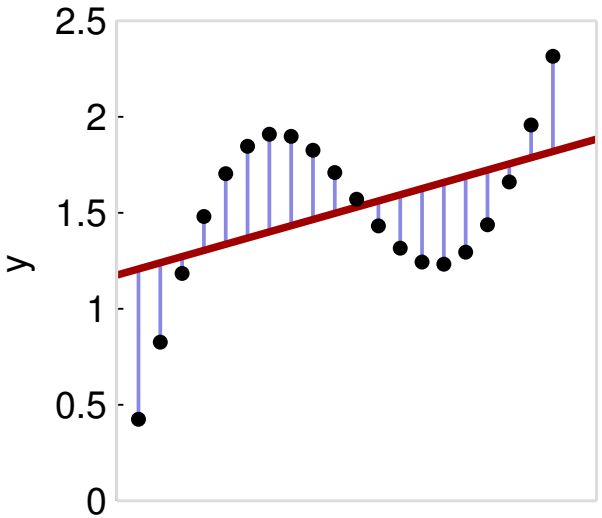
Stulp, F. and Sigaud, O. (2015). Many regression algorithms, one unified model: A review. *Neural Networks*, 69:60–79.

Regression: basic process and notations



- Input: N samples $\boldsymbol{x}_n \in \mathbb{R}^D$, $y_n \in \mathbb{R}$,
- Stored in $\boldsymbol{y} = [y_1, \cdots, y_N]$, $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$ (*design matrix*)
- Output: the latent function f such that $\boldsymbol{y} = f(\boldsymbol{X})$

Least Squares



Least squares. Black dots represent 20 training examples, and the thick (red) line is the learned latent function $f(\mathbf{x})$. Vertical lines represent residuals.

- In the linear case, we get $\boldsymbol{y} = f(\boldsymbol{X}) = \boldsymbol{w}^T \boldsymbol{X}$, where $\boldsymbol{w}$ is a vector of weights (to deal with the offset, increase $\boldsymbol{X}$ with a row of ones).
- We minimize residuals, thus

$$\boldsymbol{w}^* = \min_{\boldsymbol{w}} \underbrace{\|\boldsymbol{y} - \boldsymbol{w}\boldsymbol{X}\|^2}_{J(w)}$$

- Min reached where derivative is null, thus $\boldsymbol{w}^* = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$

Regularized Least Squares

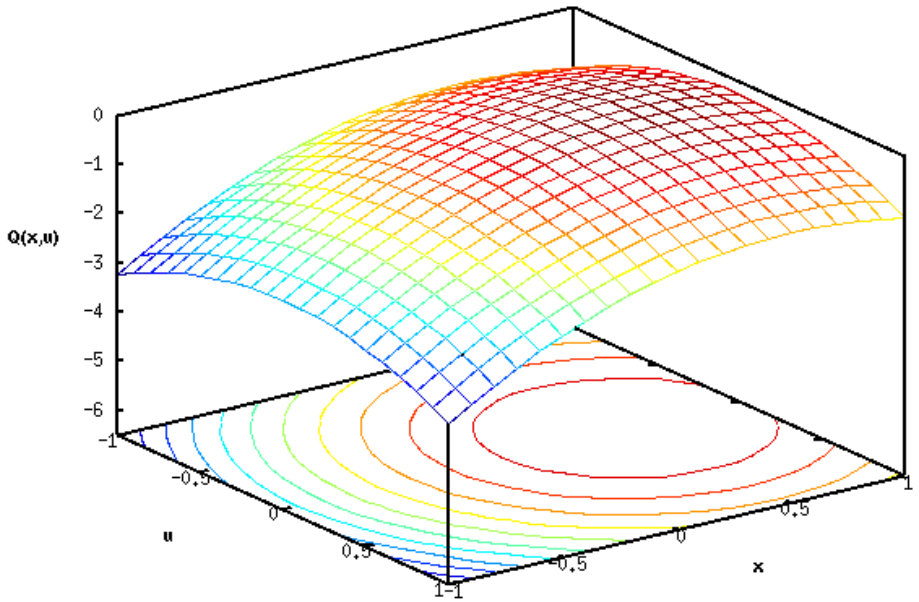
- Potential singularities in $\boldsymbol{X}^T\boldsymbol{X}$ can generate very large $\boldsymbol{w}^*$ weights
- Regularized Least Squares (Ridge Regression, RR): penalize large weights
- Optimize with lower weights (sacrifice optimality):
- $$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{w}\|^2, \tag{1}$$
- Analytical solution:

$$\boldsymbol{w}^* = (\lambda\boldsymbol{I} + \boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2}$$

Basis Function Networks: general idea

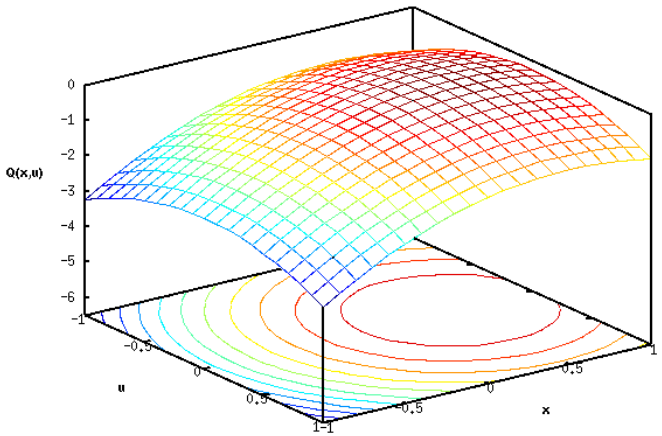
- Project the non-linear function to a different space...
- ... where the latent function is linear
- General form: $f(\mathbf{x}) = \sum_{e=1}^{E} w_e \cdot \phi(\mathbf{x}, \boldsymbol{\theta}_e)$

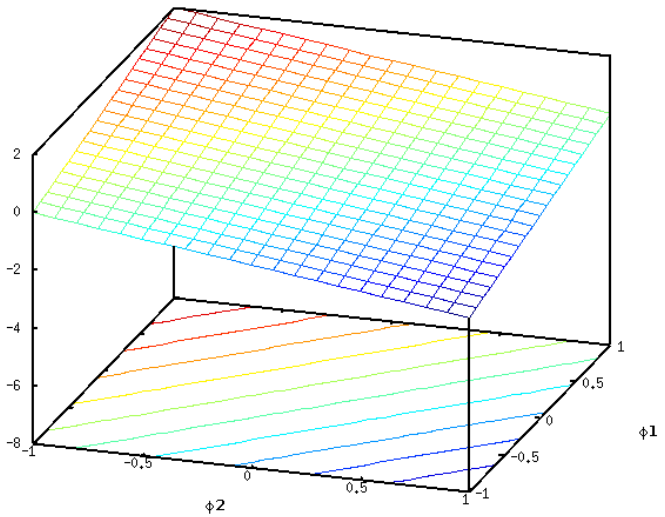
Learning with features: example



- The function to be approximated is $f(x_1, x_2) = |x_1 - x_1^*|^2 + |x_2|^2$
- We define features $\phi_i(x_1, x_2)$ over (x_1, x_2)
- We look for $\boldsymbol{w}$ such that $\hat{f}(x_1, x_2) = \Sigma_i w_i \phi_i(x_1, x_2)$

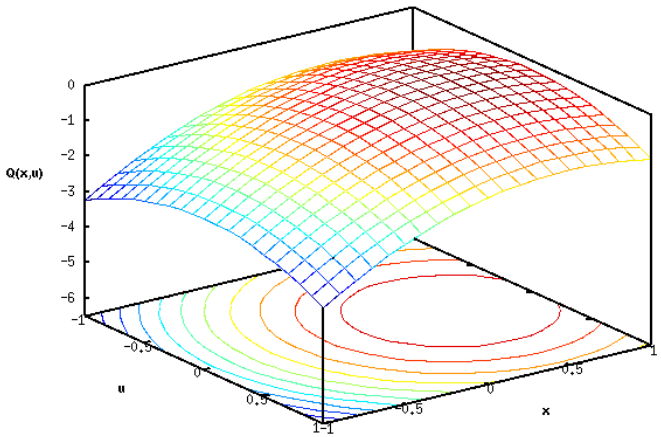
With poor features

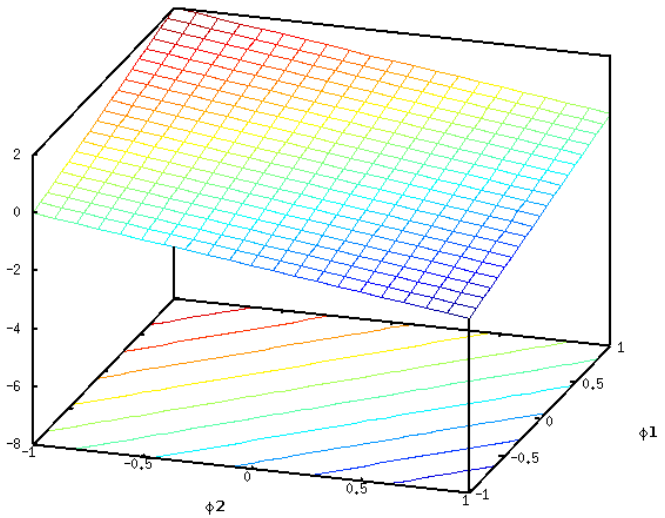




- If we take $\phi_1(x_1, x_2) = x_1$ and $\phi_2(x_1, x_2) = x_2$
- We cannot do better than $\hat{f}(x_1, x_2) = w_1 x_1 + w_2 x_2$
- Very poor linear approximation

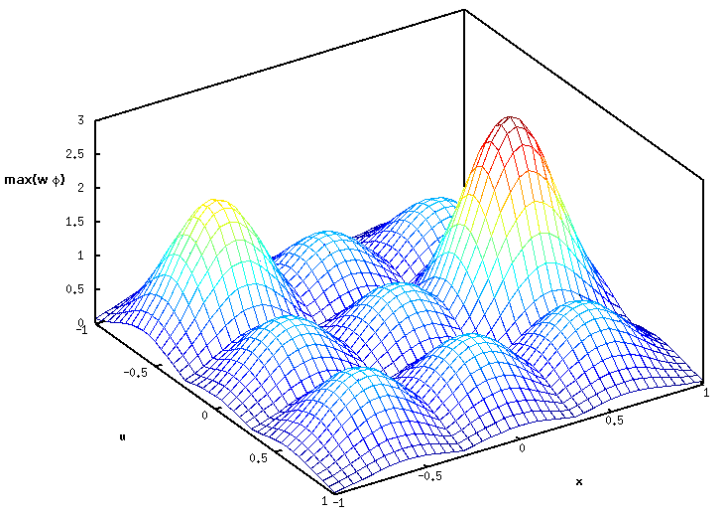
With good features

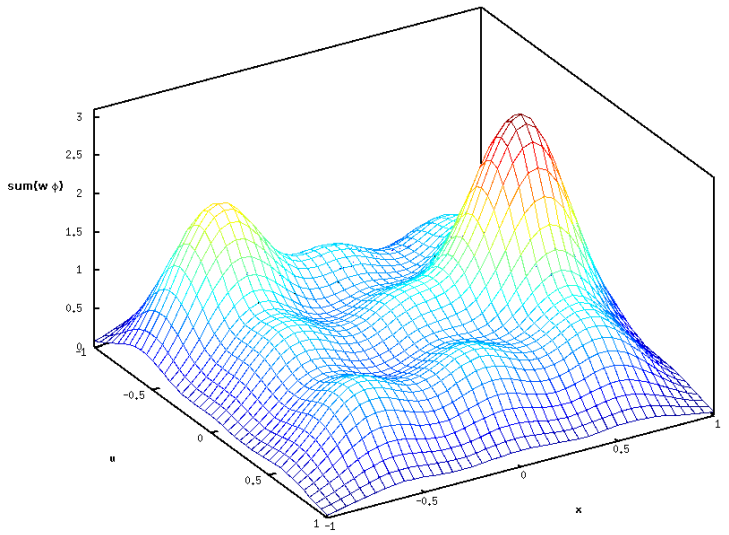




- If we take $\phi_1(x_1, x_2) = |x_1 - x_1^*|^2$ and $\phi_2(x_1, x_2) = |x_2|^2$
- Then $w_1\phi_1(x_1, x_2) + w_2\phi_2(x_1, x_2) = |x_1 - x_1^*|^2 + |x_2|^2 \rightarrow w_1 = 1$ and $w_2 = 1$
- Perfect approximation
- Finding good features is critical

Standard features: Gaussian basis functions





- The more features, the better the approximation
- ... but the more expensive the computation

Kernel Ridge Regression (KRR) = Kernel Regularised Least Squares (KRGLS)

- Define features with a kernel function $k(\mathbf{x}, \mathbf{x}_i)$ per point $\mathbf{x}_i$
- Define the Gram matrix as a kernel matrix:

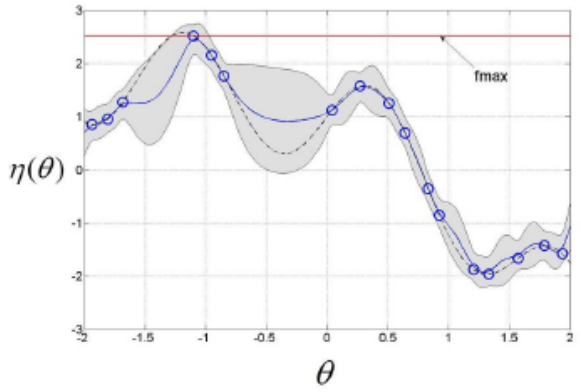
$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \tag{3}$$

- Computing the weights is done with RR using

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

- Note that $\mathbf{K}$ is symmetric
- The kernel matrix $\mathbf{K}$ 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 $\mathbf{x}$ is done by assuming that the novel output are y also sampled from a multi-variate Gaussian with

$$\mathbf{k}(\mathbf{x}, \mathbf{X}) = [k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_n)], \text{ and}$$

$$\begin{bmatrix} \mathbf{y} \\ y \end{bmatrix} \sim \mathbb{N}\mathbf{0}\begin{bmatrix} \mathbf{K} & \mathbf{k}(\mathbf{x},\mathbf{X})^\intercal \\ \mathbf{k}(\mathbf{x},\mathbf{X}) & k(\mathbf{x},\mathbf{x}) \end{bmatrix} \tag{5}$$

- The best estimate for y is the mean, and the variance in y is

$$\overline{y} = \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{y} \tag{6}$$

$$var(y) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{k}(\mathbf{x}, \mathbf{X})^\intercal.$$

Ebden, M. (2008). Gaussian processes for regression: A quick introduction. Technical report, Department on Engineering Science, University of Oxford

GPR ∼ KRR

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

$$\mathbf{w}^* = \mathbf{K}^{-1}\mathbf{y}, \tag{7}$$

- We can rewrite (6) as follows:

$$\begin{aligned} \overline{y} &= \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{y} && (8)\\ &= [k(\mathbf{x}, \mathbf{x}_1), k(\mathbf{x}, \mathbf{x}_N)] \cdot \mathbf{w}^* && (9)\\ &= \sum_{n=1}^{N} w_n^* \cdot k(\mathbf{x}, \mathbf{x}_n). && (10) \end{aligned}$$

  The mean of GPR is the same weighted sum of basis functions as in KRR, and (10) has the same form as the unified representation in (21).

- KRR computes a regularized version of the weights computed by GPR, with an additional regularization parameter λ.

Radial Basis Function Networks: definition and solution

- Radial Basis Functions versus Kernels (Gaussians $\phi(\mathbf{x}, \boldsymbol{\theta}_e) = e^{-\frac{1}{2}(\mathbf{x}-c_e)^T \boldsymbol{\Sigma}_e^{-1}(\mathbf{x}-c_e)}$ are both)
- We define a set of E basis functions (often Gaussian)

$$f(\mathbf{x}) = \sum_{e=1}^{E} w_e \cdot \phi(\mathbf{x}, \boldsymbol{\theta}_e) \tag{11}$$

$$= \mathbf{w}^\intercal \cdot \boldsymbol{\phi}(\mathbf{x}). \tag{12}$$

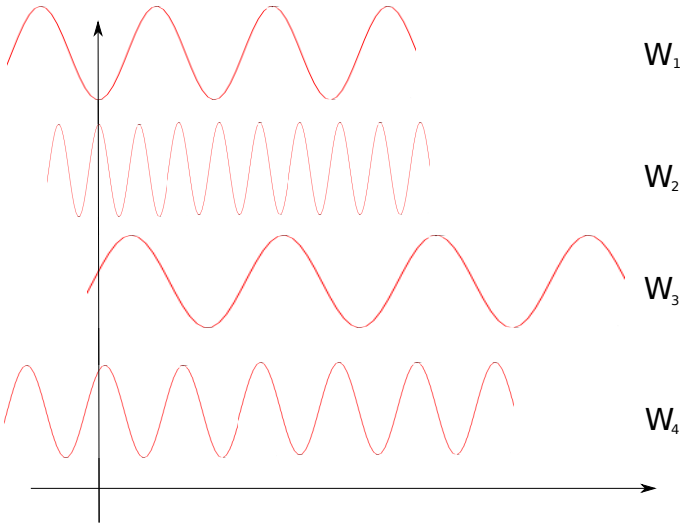
- We also define the *Gram matrix*

$$\mathbf{G} = \begin{bmatrix} \phi(\mathbf{x}_1, \boldsymbol{\theta}_1) & \phi(\mathbf{x}_1, \boldsymbol{\theta}_2) & \cdots & \phi(\mathbf{x}_1, \boldsymbol{\theta}_E) \\ \phi(\mathbf{x}_2, \boldsymbol{\theta}_1) & \phi(\mathbf{x}_2, \boldsymbol{\theta}_2) & \cdots & \phi(\mathbf{x}_2, \boldsymbol{\theta}_E) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N, \boldsymbol{\theta}_1) & \phi(\mathbf{x}_N, \boldsymbol{\theta}_2) & \cdots & \phi(\mathbf{x}_N, \boldsymbol{\theta}_E) \end{bmatrix} \tag{13}$$

- and we get the least squares solution

$$\mathbf{w}^* = (\mathbf{G}^\intercal \mathbf{G})^{-1} \mathbf{G}^\intercal \mathbf{y}. \tag{14}$$

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

Gijsberts, A. & Metta, G. (2011) "Incremental learning of robot dynamics using random features." In *IEEE International Conference on Robotics and Automation* (pp. 951–956).

Least Square computation: summary

- Linear case

$$\mathbf{w}^* = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y} \qquad (LS) \qquad (15)$$

$$\mathbf{w}^* = (\lambda \mathbf{I} + \mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y}. \qquad (RLS) \qquad (16)$$

- Gram matrix case

$$\mathbf{w}^* = (\mathbf{G}^\intercal \mathbf{G})^{-1} \mathbf{G}^\intercal \mathbf{y} \qquad (RBFN) \qquad (17)$$

- Kernel matrix case

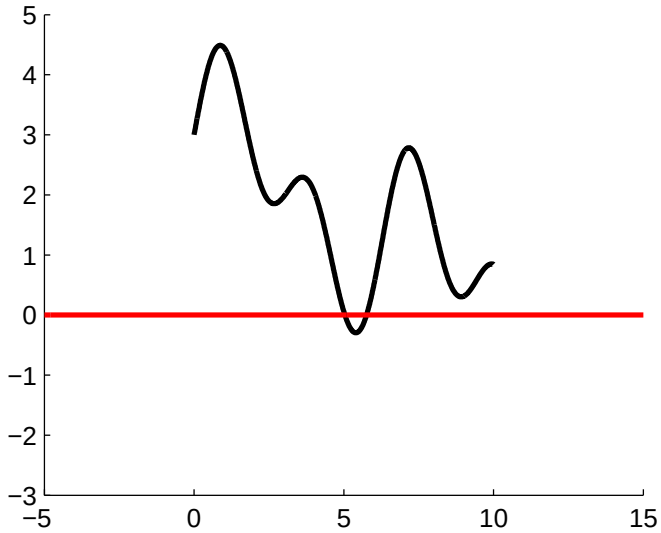
$$\mathbf{w}^* = \mathbf{K}^{-1} \mathbf{y}, \qquad (GPR) \qquad (18)$$

$$\mathbf{w}^* = (\lambda \mathbf{I} + \mathbf{K})^{-1} \mathbf{y}. \qquad (KRR) \qquad (19)$$

Basis Function Networks: computation

- Solving $\mathbf{w}^* = (\mathbf{G}^\intercal \mathbf{G})^{-1} \mathbf{G}^\intercal \mathbf{y}$ requires inverting $(\mathbf{G}^\intercal \mathbf{G})$
- 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...
- True of all other BFN algorithms

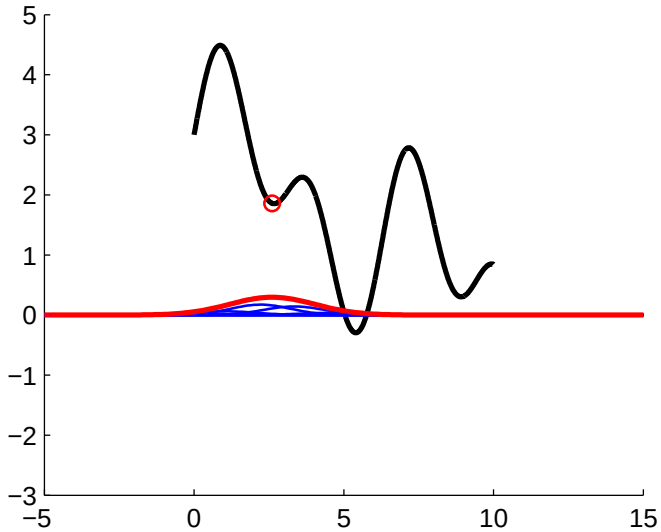
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

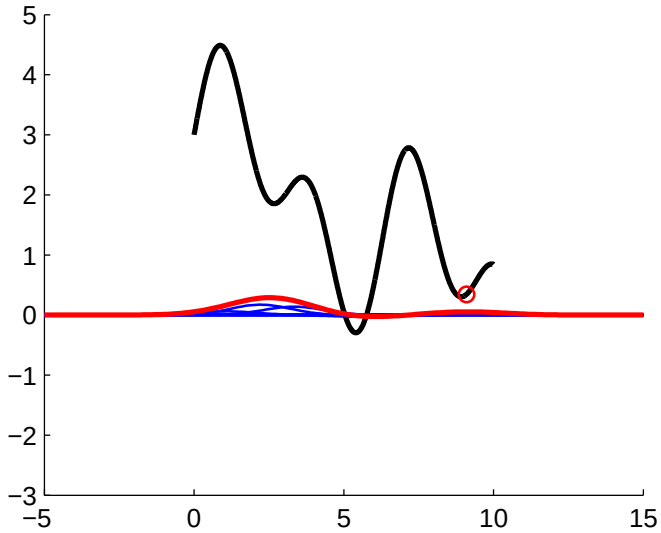
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

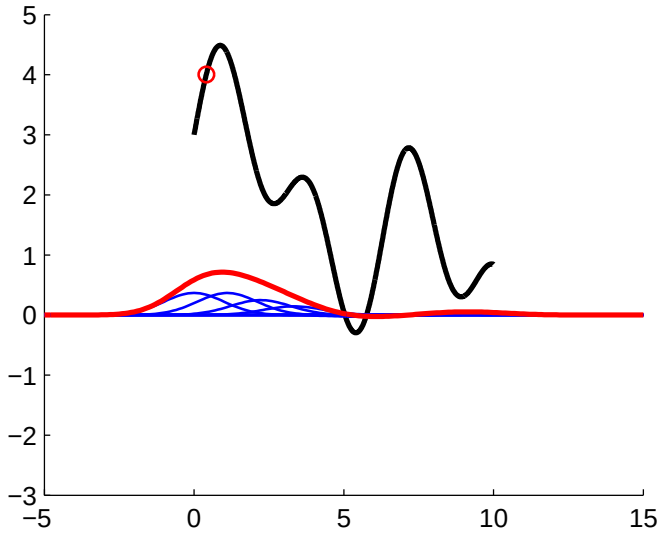
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

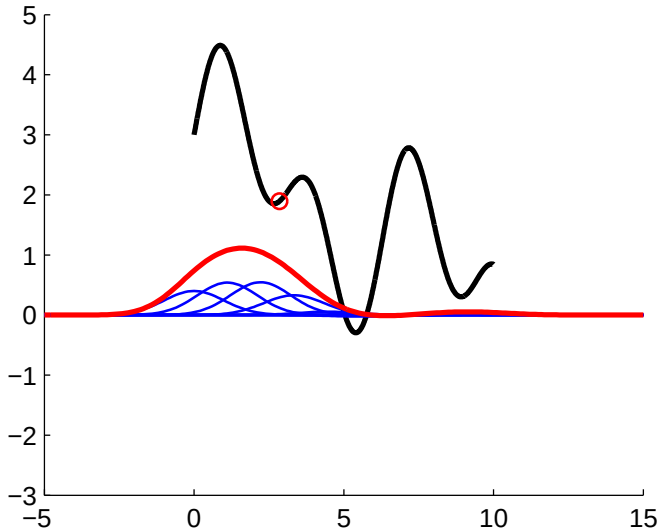
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

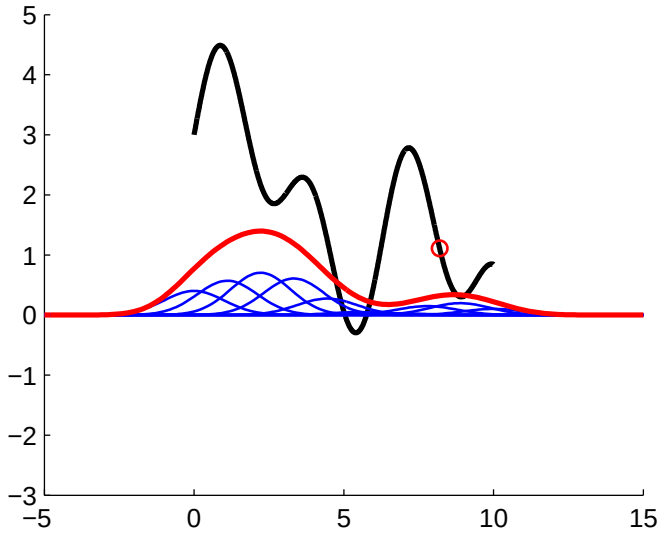
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

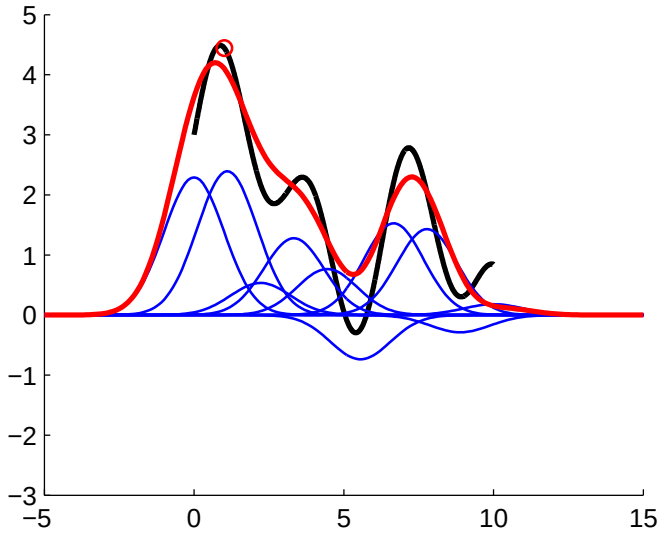
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

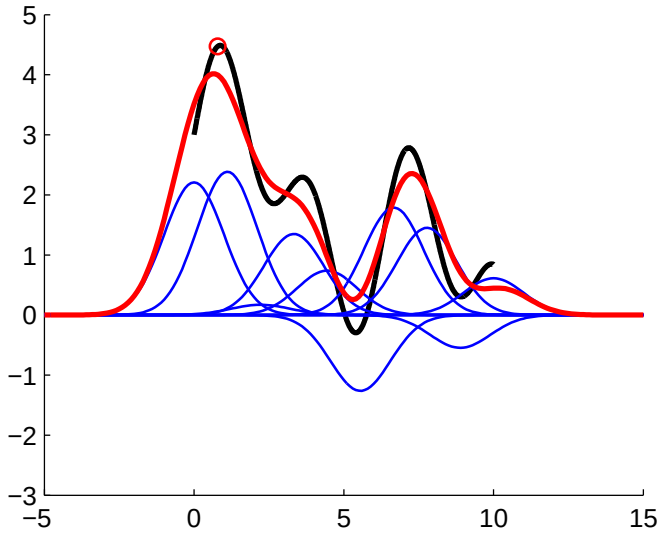
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

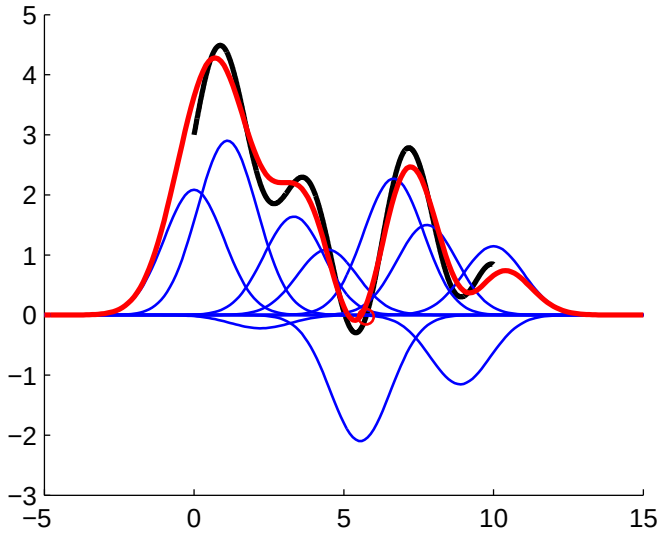
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

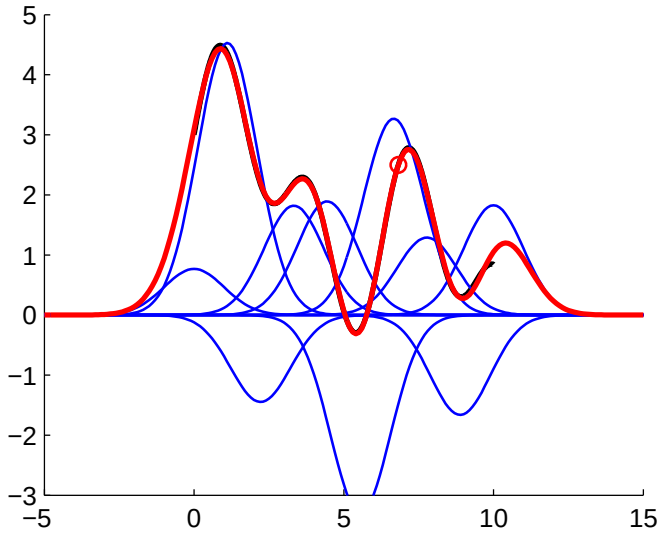
Radial Basis Function Networks (Illustration)



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

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

Radial Basis Function Networks (Illustration)



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

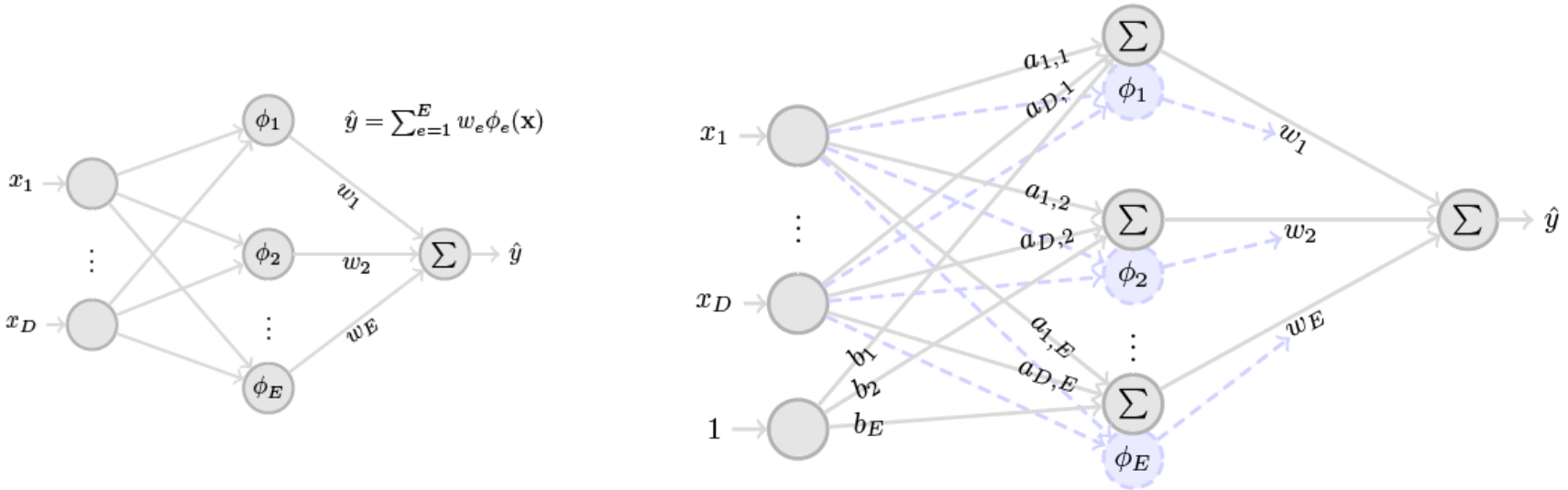
Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

Basis Function Networks: summary

Algorithm	Regularized?	Number of BFs?	Features?
RBFN	Yes	E	RBFs
KRR	Yes	N	kernels
GPR	No	N	kernels
iRFRLS	Yes	E	cosine
I-SSGPR	Yes	E	cosine

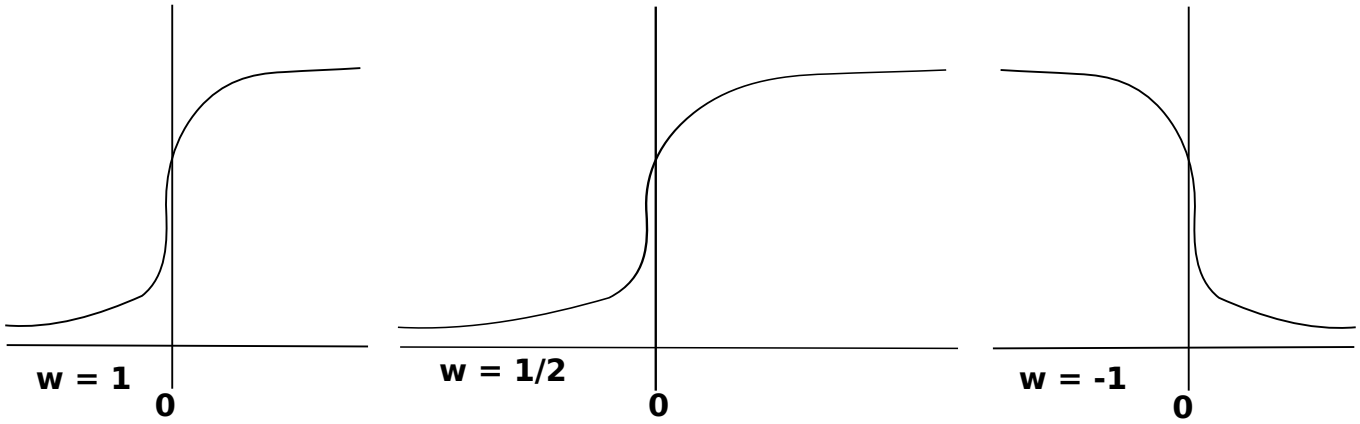
Table: Design of all weigthed basis function algorithms.

The case of (feedforward) neural networks



- Shares the same structure as all basis function networks
- Sigmoids instead of Gaussians: better split of space in high dimensions

Tuning neural networks for regression



- Weight of output layer: regression
- Weight of input layer(s): tuning basis functions
- The backprop algo tunes both output and hidden weights
- Stochastic optimization of input weights, linear regression on output weights? (see e.g. Reservoir computing)
- Deep neural nets: get more tunable features with less parameters
- Discovers the adequate features by itself

Locally Weighted Regression

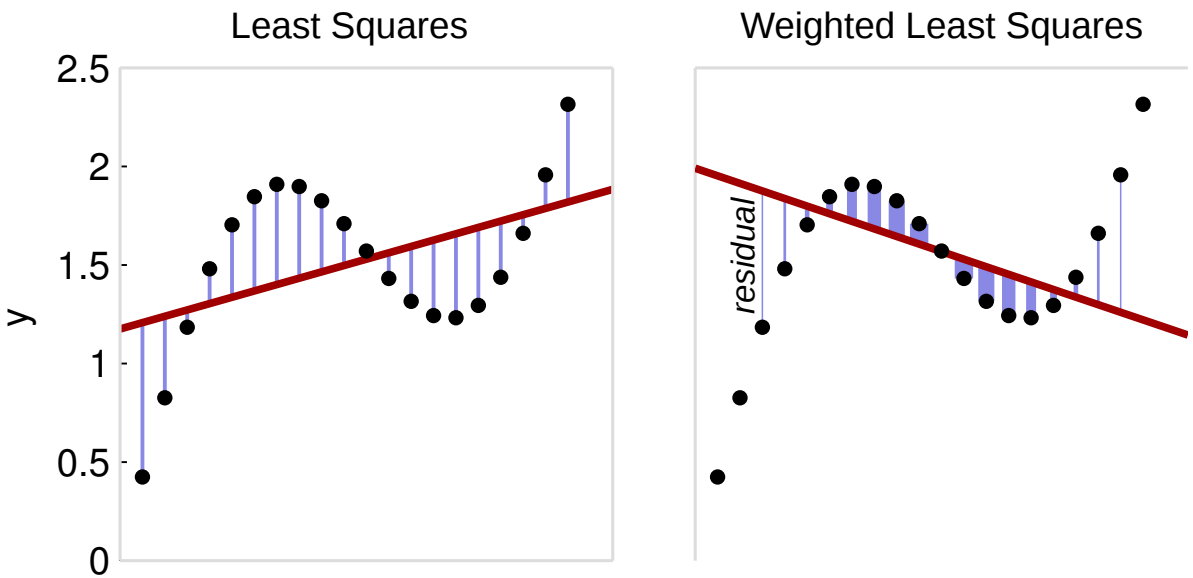
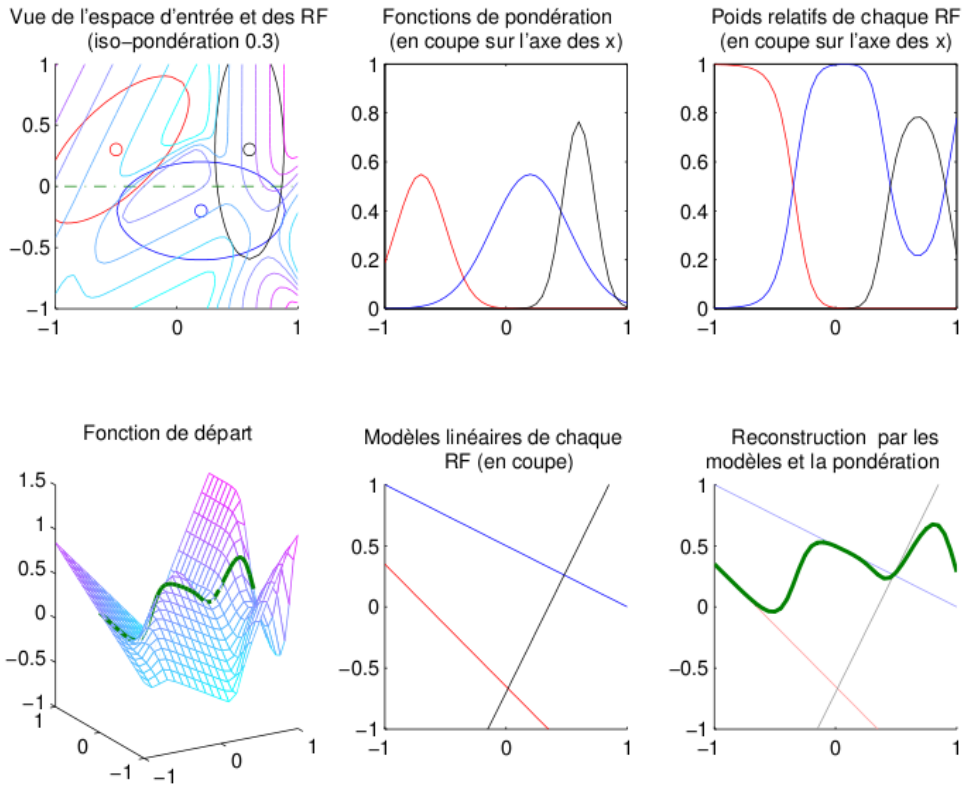


Figure: The thickness of the lines indicates the weights.

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

Atkeson, C. (1991). Using locally weighted regression for robot learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 958–963.

LWR approximation: graphical intuition



- Each RF tunes a local linear model

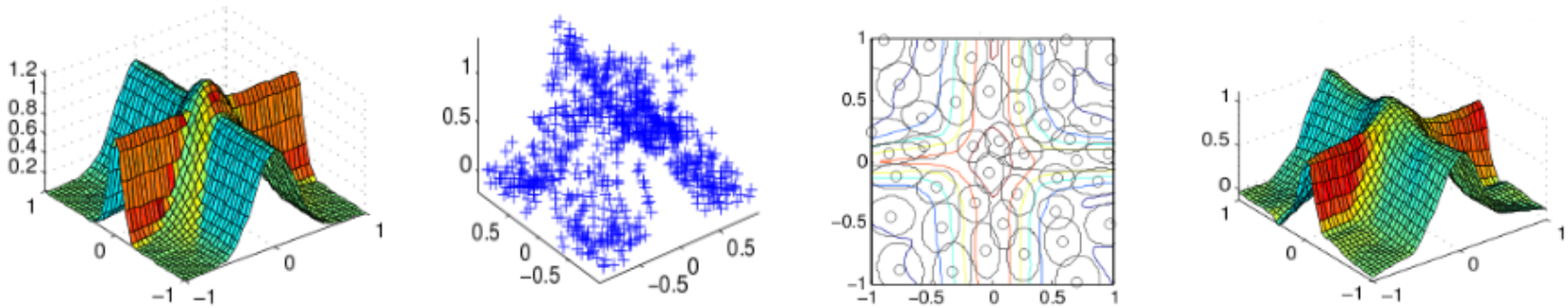
$$\Psi_e(\mathbf{x}) = \mathbf{a}_e^\intercal \mathbf{x} + b_e$$

- Gaussians tell you how much each RF contributes to the output

$$y = \frac{\sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e)\Psi_e(\mathbf{x})}{\sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e)}$$

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

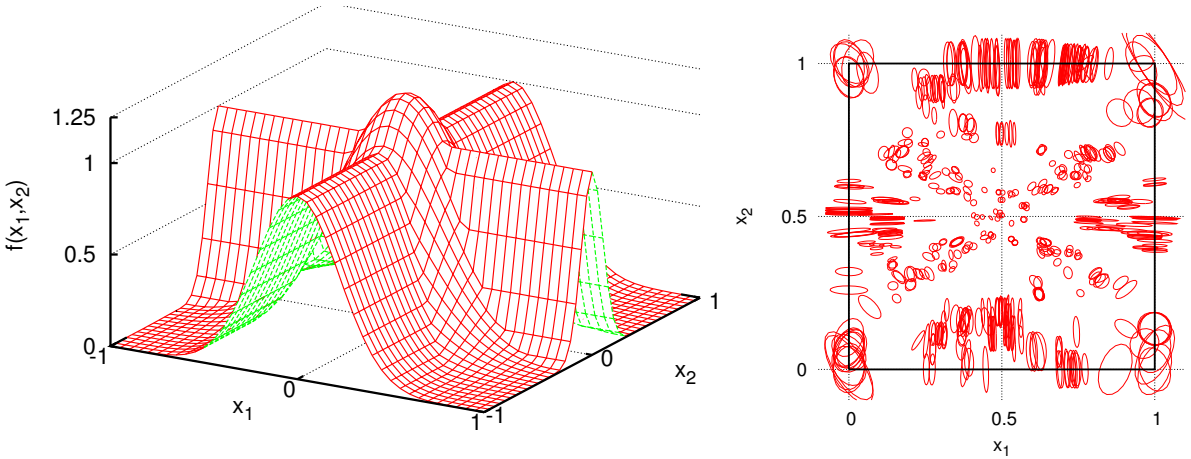
LWPR: general goal



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

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60.

XCSF: overview



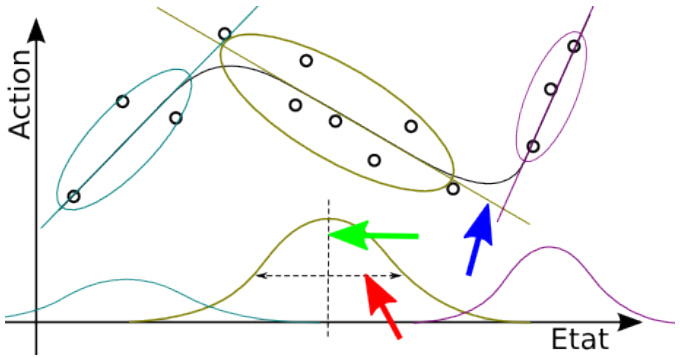
- XCSF is a Learning Classifier System [Holland, 1975]
- Linear models weighted by Gaussian functions (similar to LWPR)
- Linear models are updated using RLS
- Gaussian functions adaptation: Σ_e^{-1} and c_e are updated using a GA
- Key feature: distinguish weights space and models space (example: $\mathbf{x}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

LWPR: $f(\mathbf{x}) = \sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e) \cdot (b_e + \mathbf{a}_e^\intercal \mathbf{x})$ XCSF: $f(\mathbf{x}) = \sum_{e=1}^{E} \phi(q, \boldsymbol{\theta}_e) \cdot (b_e + \mathbf{a}_e^\intercal \dot{q})$

- Condensation: reduce population to generalize better

Wilson, S. W. (2001). 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.

GMR



$$y = \sum_{k=1}^{K} h_k(\mathbf{x})(\mu_{k,Y} + \Sigma_{k,YX}\Sigma_{k,Y}^{-1}(\mathbf{x} - \mu_{k,X}))$$

With

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

- From input-output manifold to input-output function
- Same representation as the others, using $\mathbf{a}_e^\intercal = \Sigma_{e,YX}\Sigma_{e,Y}^{-1}$ and $b_e = \mu_{e,Y} - \Sigma_{e,YX}\Sigma_{e,X}^{-1}\mu_{e,X}$
- We get

$$\widetilde{y} = \sum_{e=1}^{E} \frac{\pi_e \phi(\mathbf{x}, \boldsymbol{\theta}_e)}{\sum_{l=1}^{E} \pi_l \phi(\mathbf{x}, \boldsymbol{\theta}_l)} (\mathbf{a}_e^\intercal \mathbf{x} + b_e),$$

- Same as usual + scaling with the priors $\pi_e \rightarrow \pi_e = 1$ in standard model.
- Incorporates Bayesian variance estimation $\rightarrow$ The richest representation

Hersch, M., Guenter, F., Calinon, S., & Billard, A. (2008) "Dynamical system modulation for robot learning via kinesthetic demonstrations." *IEEE Transactions on Robotics*, 24(6), 1463–1467.

LWR methods: main features

Algo	LWR	LWPR	GMR	XCSF
Number of RFs	fixed	growing	fixed	adaptive
Position of RFs	fixed	fixed	adaptive	adaptive
Size of RFs	fixed	adaptive	adaptive	adaptive

- The main differences are in meta-parameter tuning

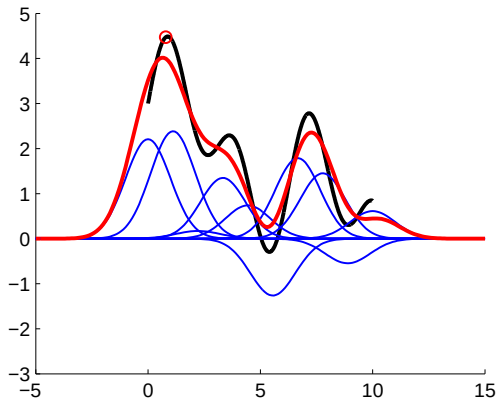
LWR versus RBFNs

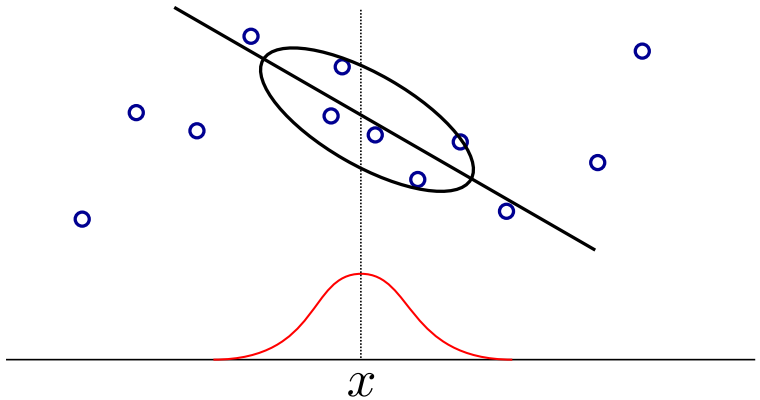
$$f(\mathbf{x}) = \sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e) \cdot (b_e + \mathbf{a}_e^\intercal \mathbf{x}) \tag{20}$$

$$f(\mathbf{x}) = \sum_{e=1}^{E} \phi(\mathbf{x}, \boldsymbol{\theta}_e) \cdot w_e, \tag{21}$$

- Eq. (21) is a special case of (20) with $\mathbf{a}_e = \mathbf{0}$ and $b_e = w_e$.
- RBFNs: performs one LS computation in a projected space
- LWR: performs many LS computation in local domains

Take home messages





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

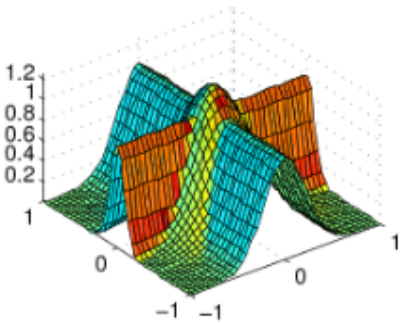
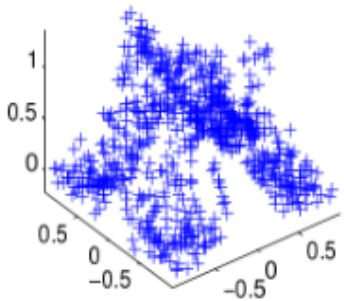
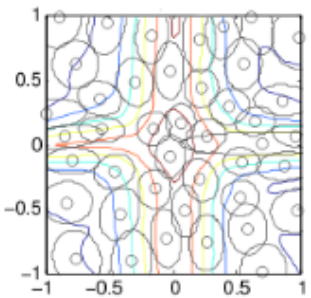
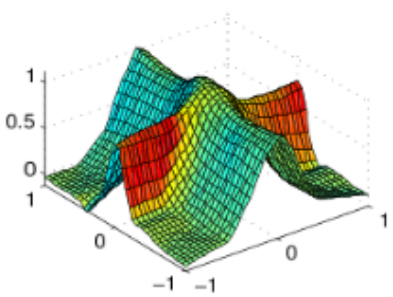
Sigaud. O. , Salaün, C. and Padois, V. (2011) "On-line regression algorithms for learning mechanical models of robots: a survey," *Robotics and Autonomous Systems*, 59:1115-1129.

Questions

- In Locally Weighted Regression approaches, a single regression in a projected space is performed
  Right or Wrong ?
- GMR is a Locally Weighted Regression approach
  Right or Wrong ?
- The model behind Locally Weighted Regression has more parameters than the one behind Basis Function Networks
  Right or Wrong ?

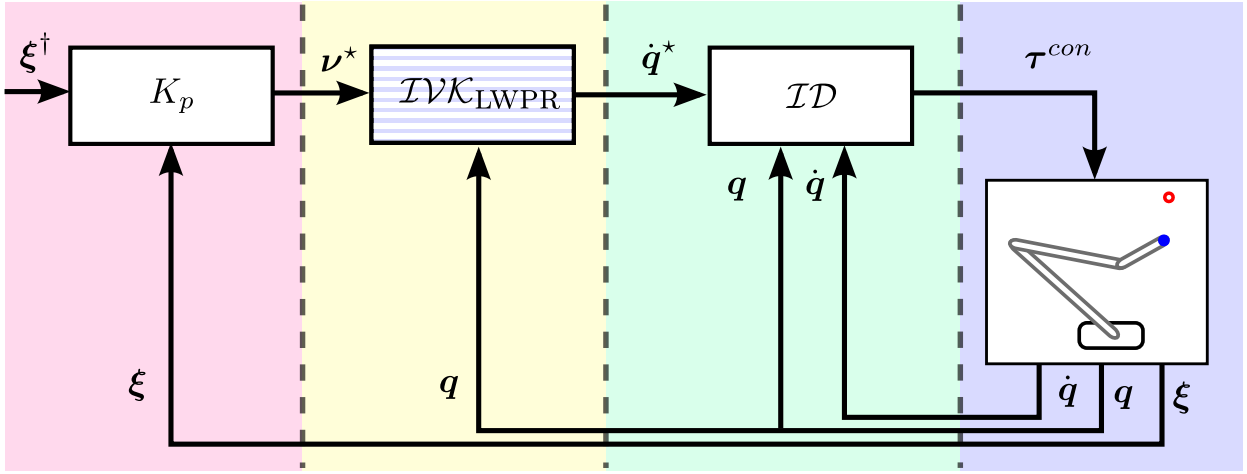
Learning mechanical models

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

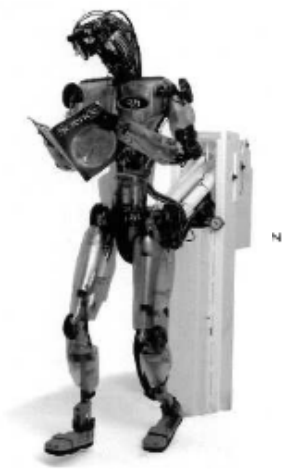





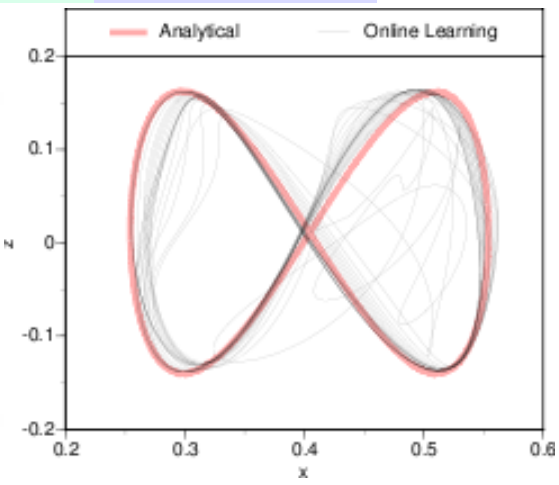
- Regression methods can approximate such functions
- The mapping can be learned incrementally from samples
- Can be used for interaction with unknown objects or users

Learning inverse kinematics with LWPR



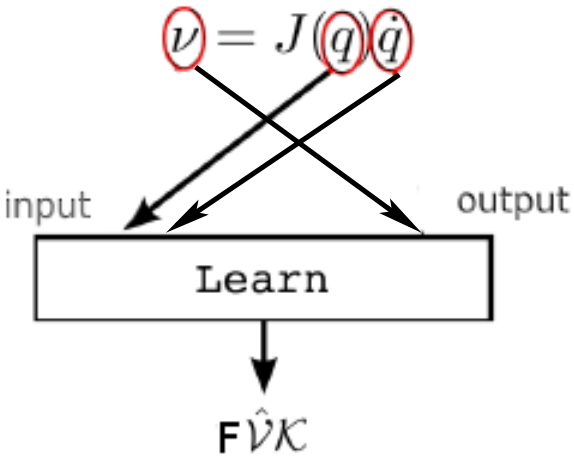
- The model is learned with random movements along an operational trajectory
- Input dimension: $dim(\xi + q) = 29$
- Output dimension: $dim(\dot{q}) = 26$



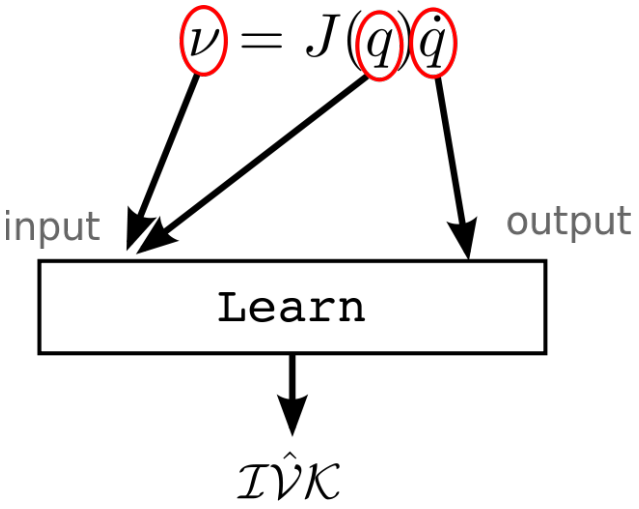


D'Souza, A., Vijayakumar, S., and Schaal, S. (2001b). Learning inverse kinematics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 298–303.

Learning forward/inverse velocity kinematics with LWPR



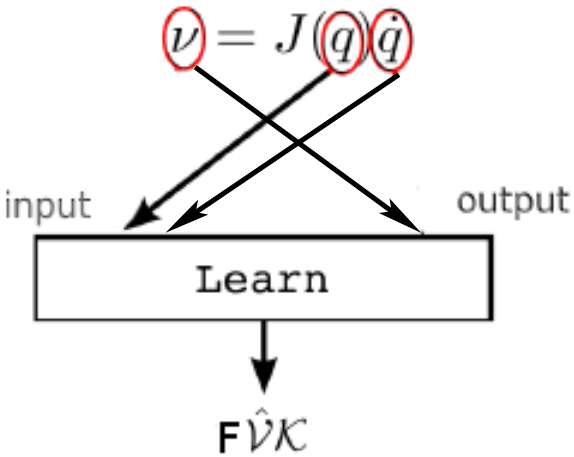
Forward kinematics



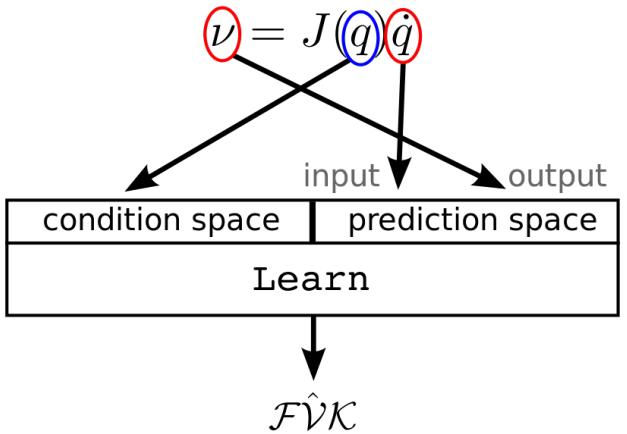
Inverse kinematics

- 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

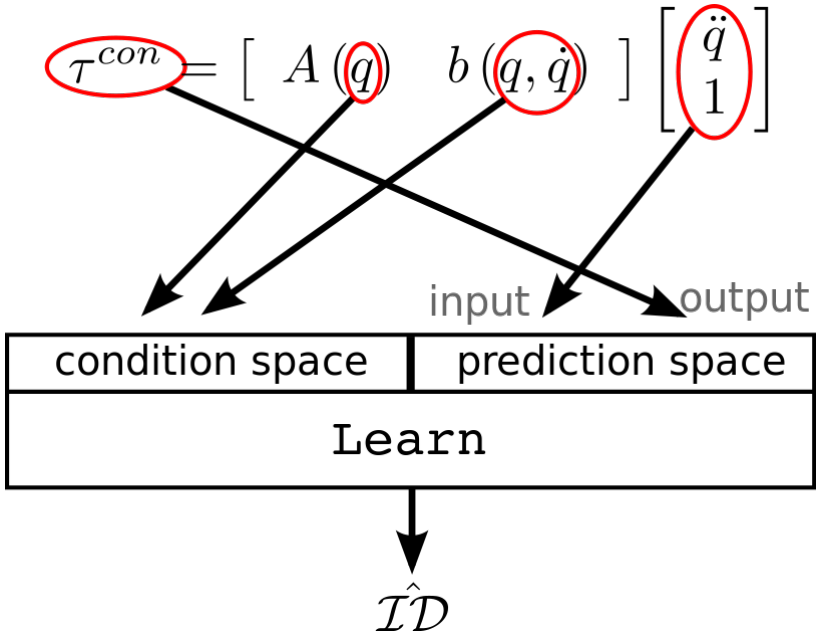


Forward kinematics with XCSF

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

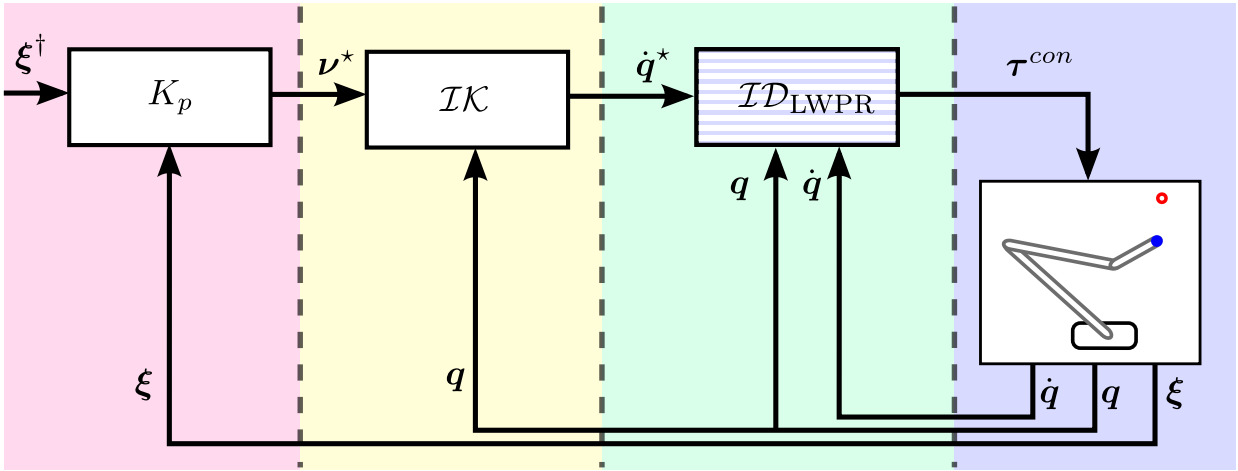
Butz, M., Pedersen, G., and Stalph, P. (2009). Learning sensorimotor control structures with XCSF: redundancy exploitation and dynamic control. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1171–1178. ACM.

Learning dynamics with XCSF

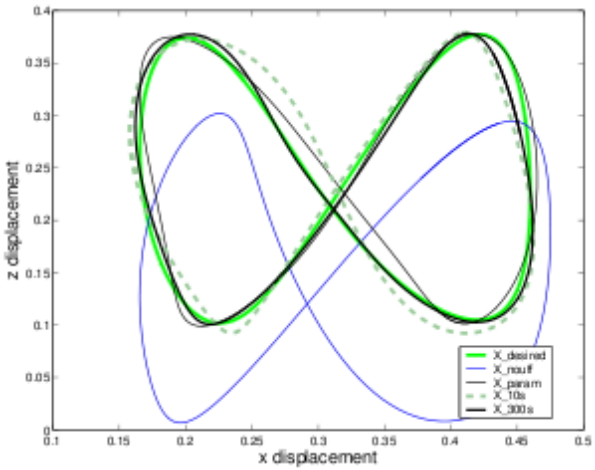


- Learning dynamics is more difficult
- In dynamics, there is no redundancy
- The dynamics model is 2/3 smaller with XCSF than with LWPR

Learning inverse dynamics with LWPR



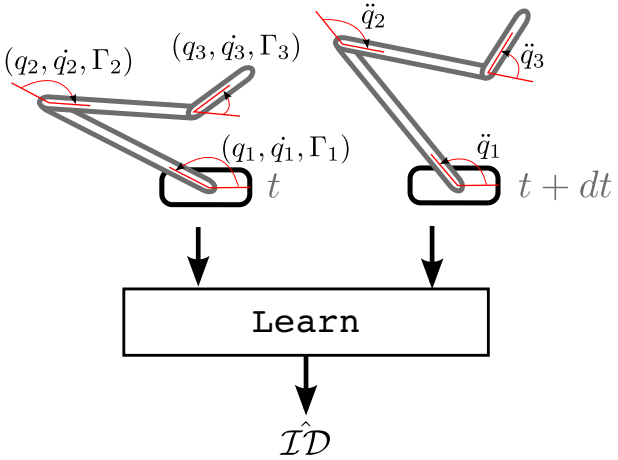
- The model is learned along an operational trajectory
- Input dimension: $dim(q + \dot{q} + \ddot{q}) = 90$
- Output dimension: $dim(\Gamma) = 30$
- $7,5.10^6$ training data points and 2200 receptive fields



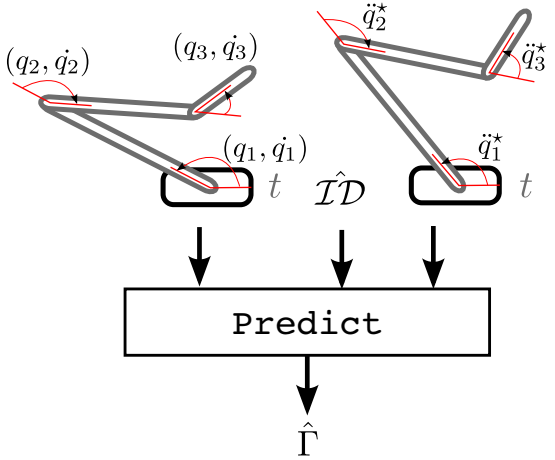
Vijayakumar, S., D'Souza, A., and Schaal, S. (2005). LWPR: A scalable method for incremental online learning in high dimensions. Technical report, Edinburgh: Press of University of Edinburgh.

Learning inverse dynamics

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

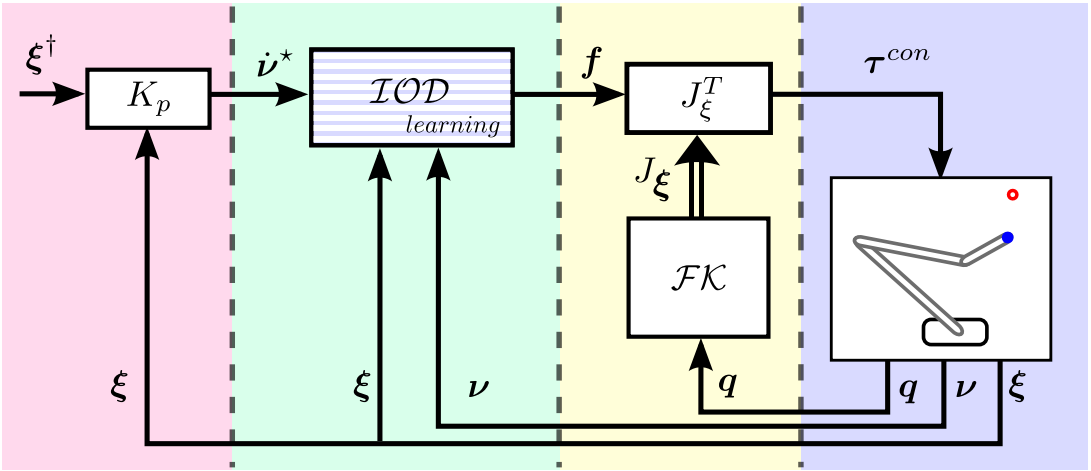


Learning inverse dynamics with random movements along a trajectory

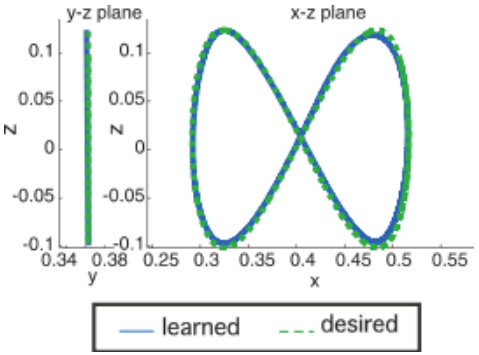


Predict inverse dynamics

Learning inverse operational dynamics

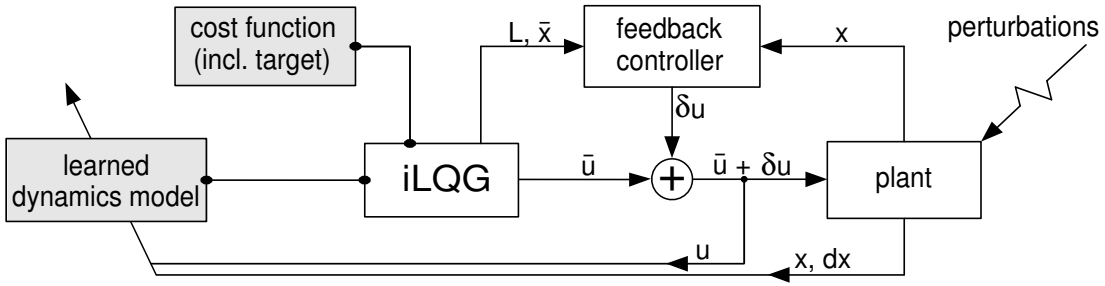


- Peters and Schaal (2008) learn inverse dynamics in the operational space.
- The model is learned along an operational trajectory.
- Input dimension : $dim(q + \dot{q} + \nu) = 17$
- Output dimension: $dim(\Gamma) = 7$

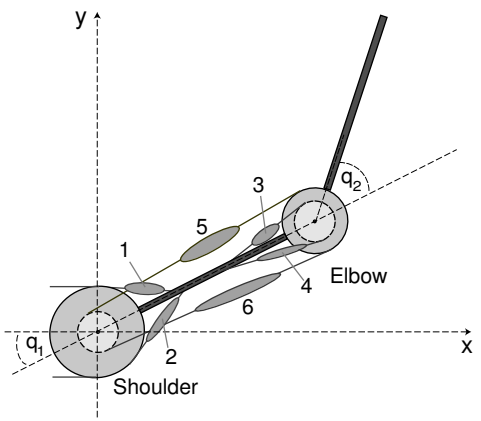


Peters, J. and Schaal, S. (2008). Learning to control in operational space. *International Journal in Robotics Research*, 27(2):197–212.

Optimal control with dynamics learned with LWPR



- The inverse dynamics model is learned in the whole space.
- Input dimension :$dim(q + \dot{q} + u) = 10$. Output dimension : $dim(\ddot{q}) = 2$.
- $1,2.10^6$ training data points and 852 receptive fields
- Learning a model of redundant actuation



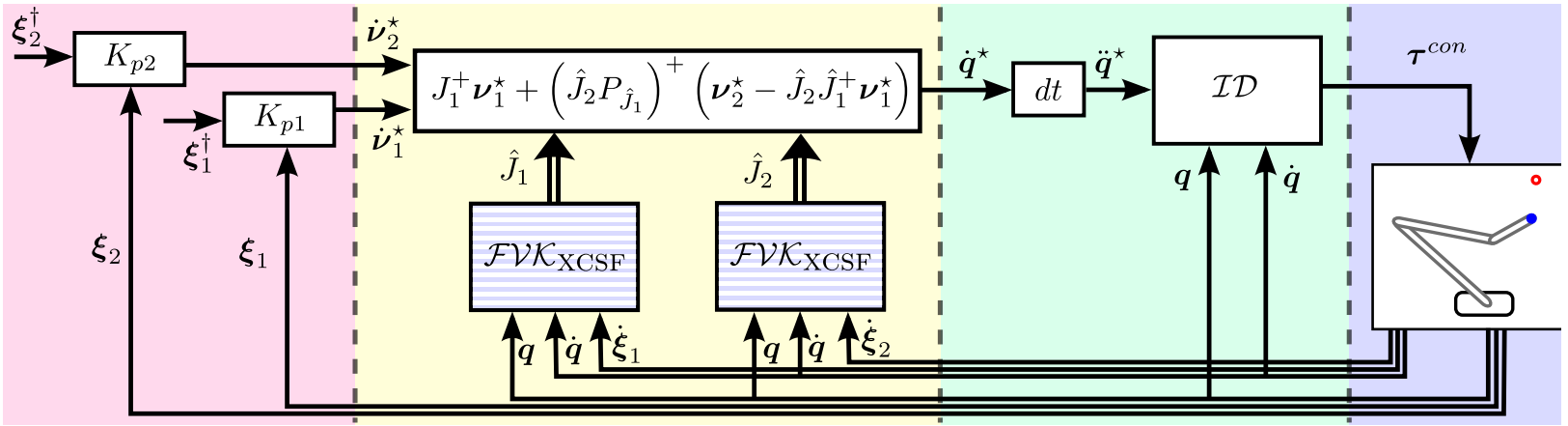
Mitrovic, D., Klanke, S., and Vijayakumar, S. (2008). Adaptive optimal control for redundantly actuated arms. In *Proceedings of the Tenth International Conference on Simulation of Adaptive Behavior*.

Properties of models

	learn forward velocity kinematics	learn inverse velocity kinematics	learn inverse dynamics	learn in whole space	control redundant actuators	control kinematic redundancy
D'Souza et al. (2001)		●				
Vijayakumar et al. (2005)			●			
Peters et al.(2008)		●	●			
Sun et al. (2004)	●			●		
Butz et al. (2009)	●			●		○
Mitrovic et al. (2008)			●		●	
Salaün et al. (2010)	●		●	●		●

- [D'Souza et al., 2001b], [Vijayakumar et al., 2005] and [Peters & 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.

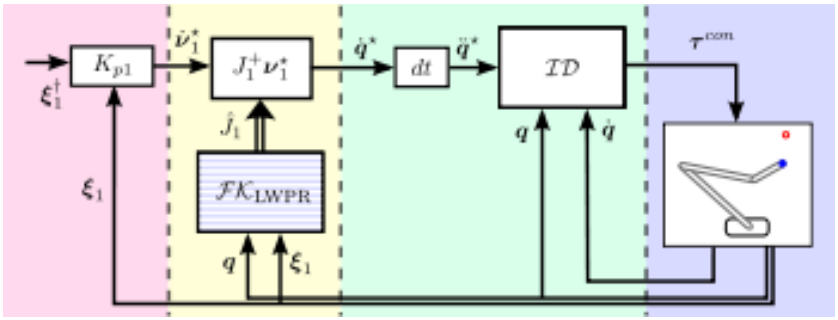
Camille Salaün's work: combining tasks

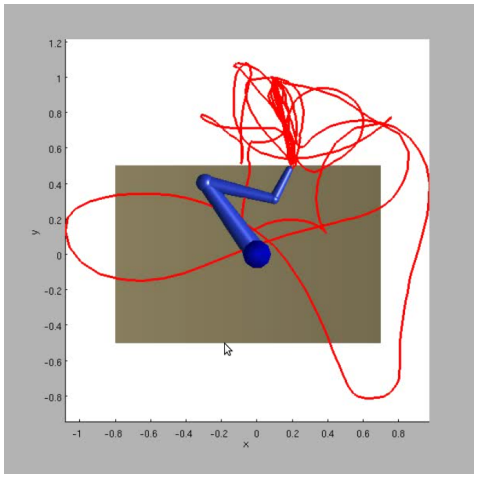


To perform several tasks with learnt models, we have chosen to

- 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

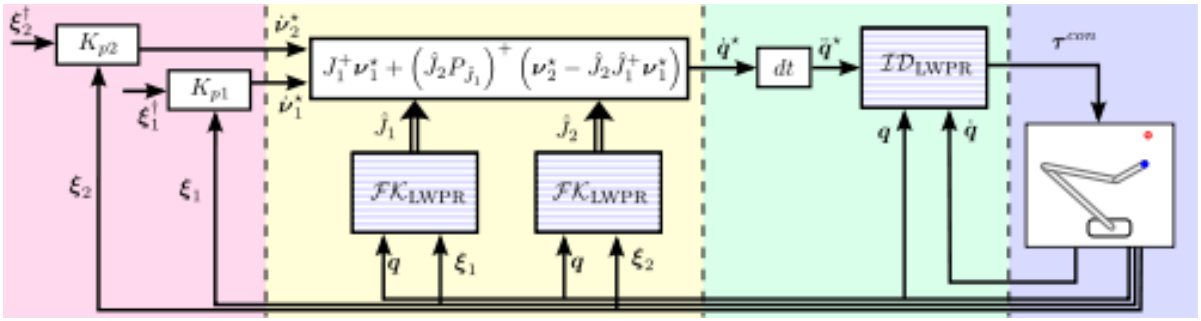


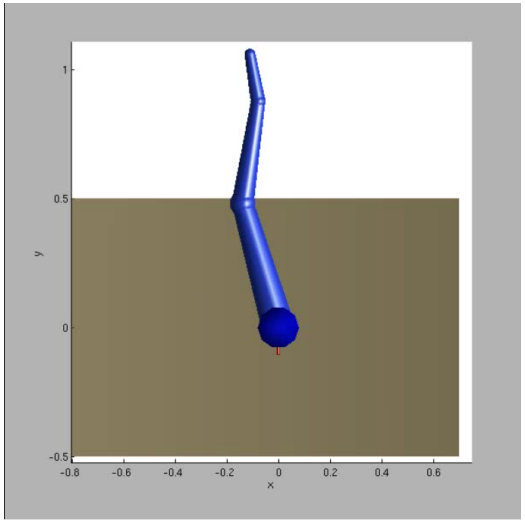


Point to point task

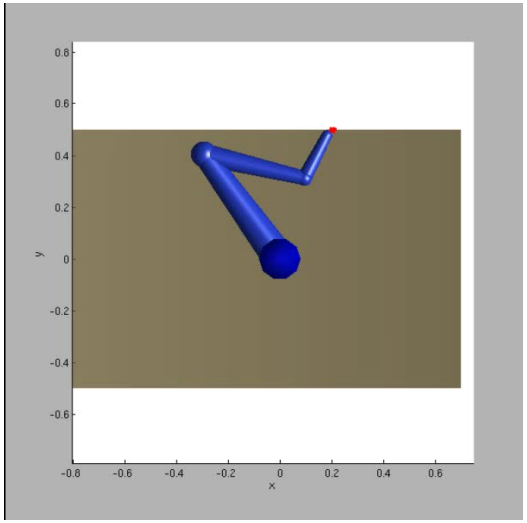
500 steps babbling with the kinematics model we want to learn.

Controlling redundancy with LWPR



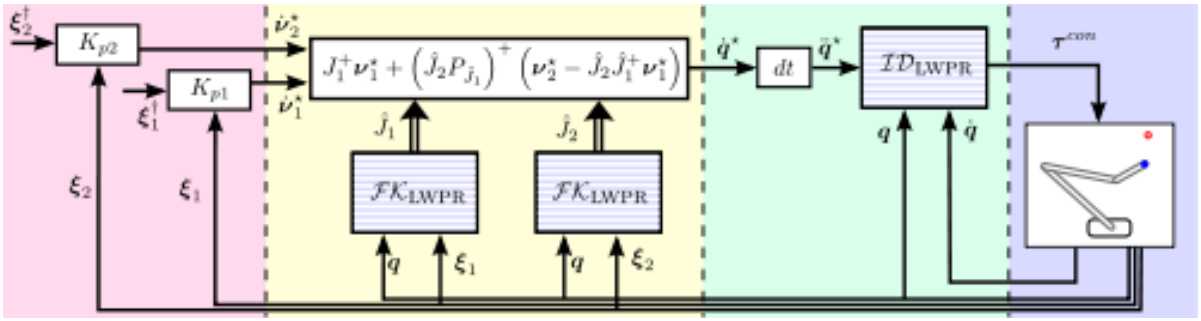


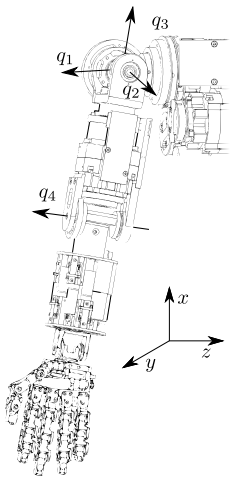
compatible task

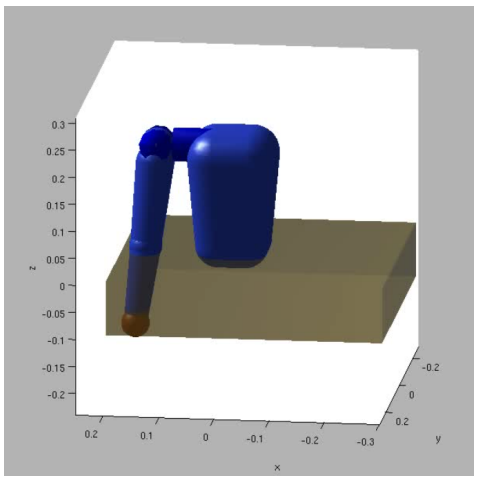


incompatible task

Learning kinematics of iCub in simulation

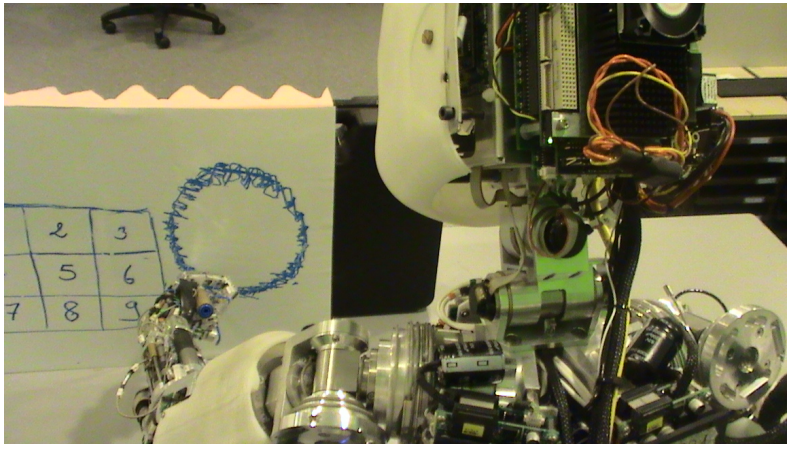


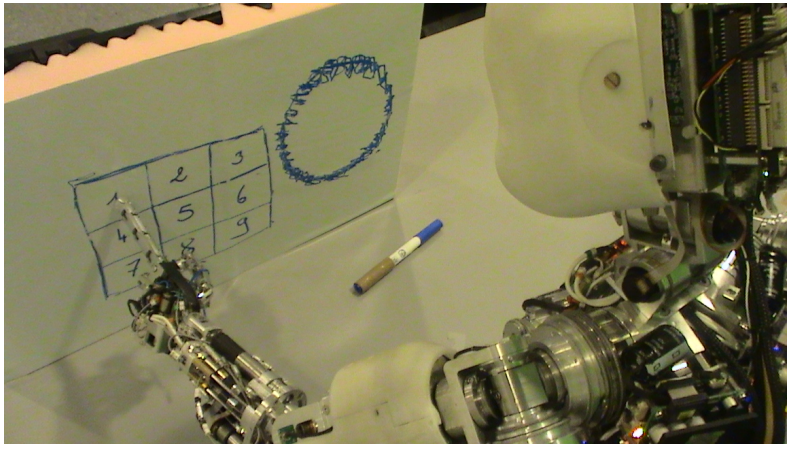


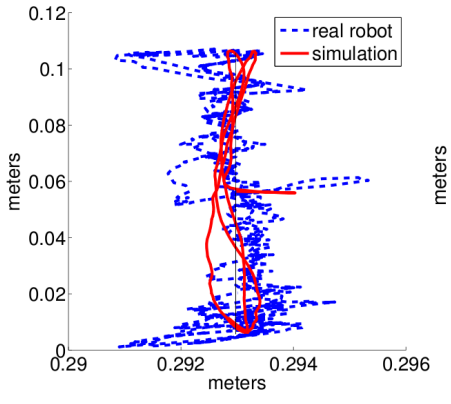


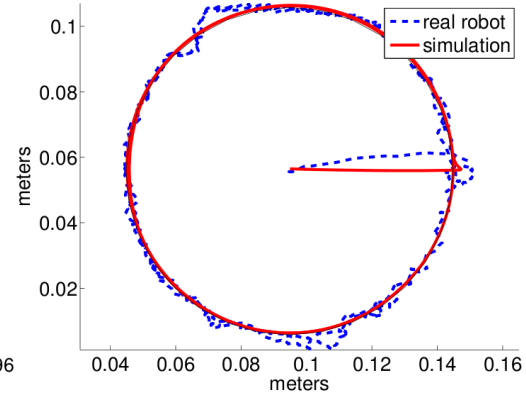
- Simulation of a three degrees of freedom shoulder plus one degrees of freedom elbow

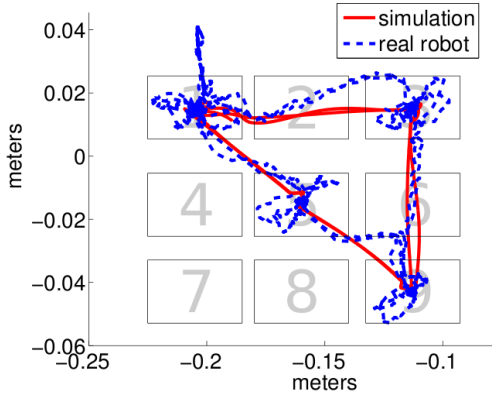
Learning kinematics on the real robot







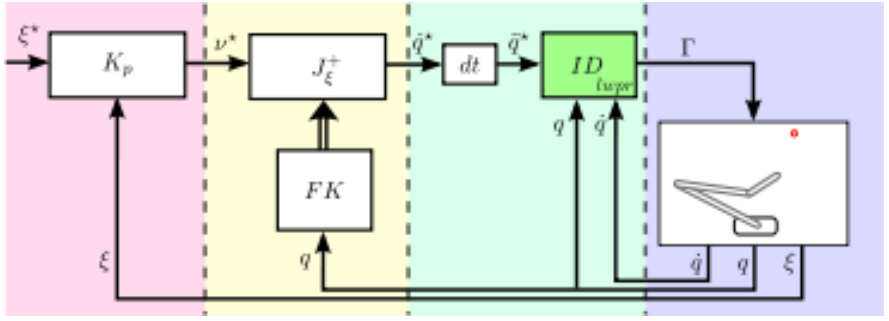


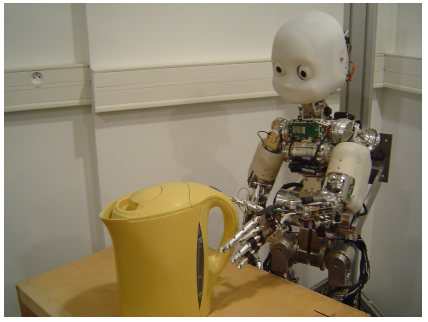


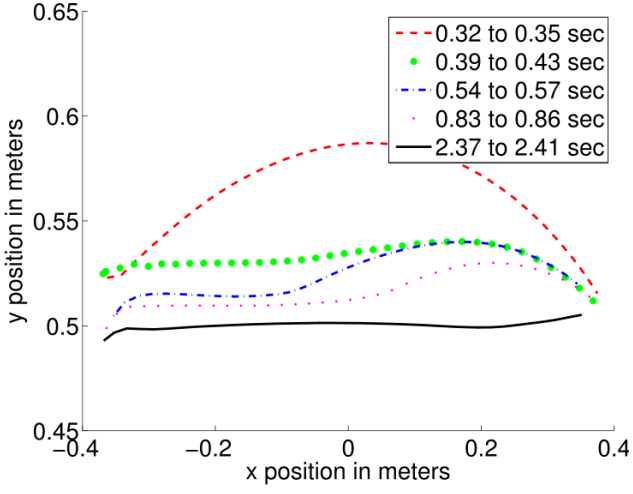
iCub realising two tasks: following a circle and clicking a numpad

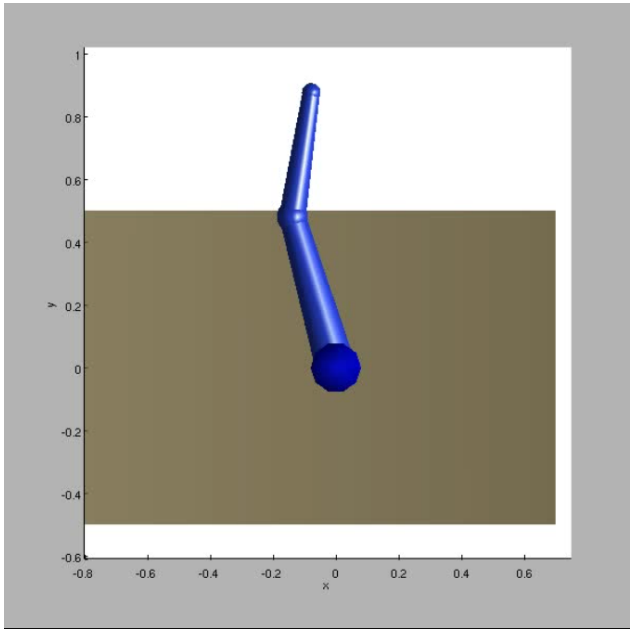
ISIR
INSTITUT
DES SYSTÈMES
INTELLIGENTS
ET DE ROBOTIQUE

Inverse dynamics and motor adaptation



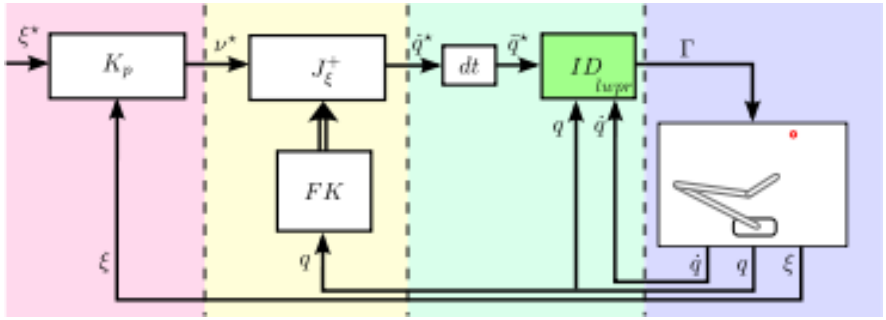


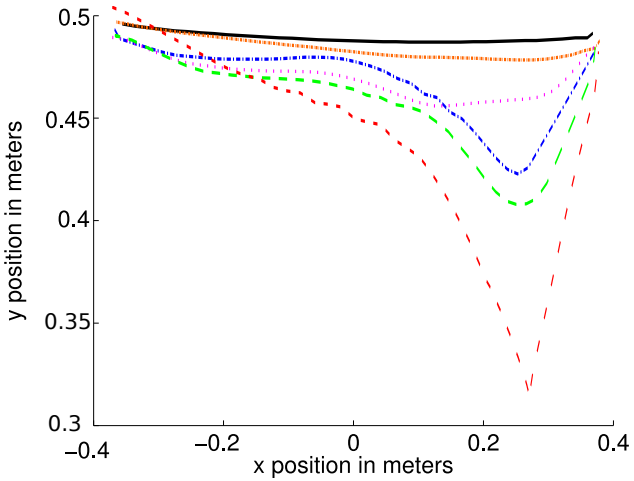


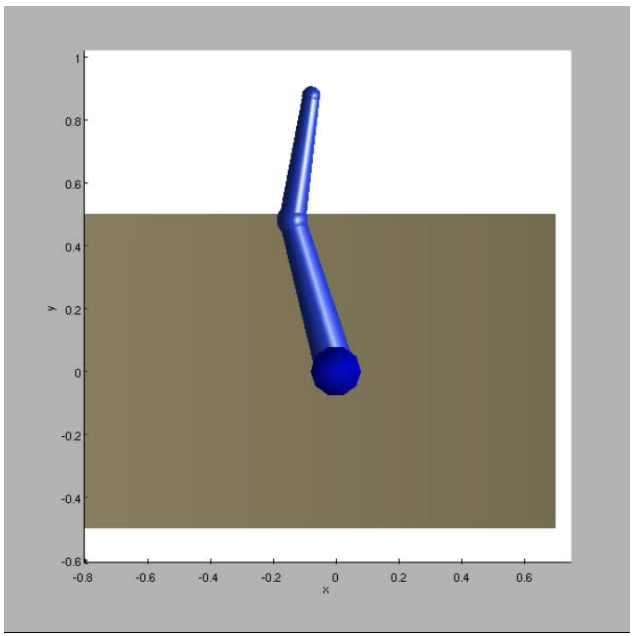


Applying a vertical force after 2 seconds during a point to point task.

Inverse dynamics and after effects



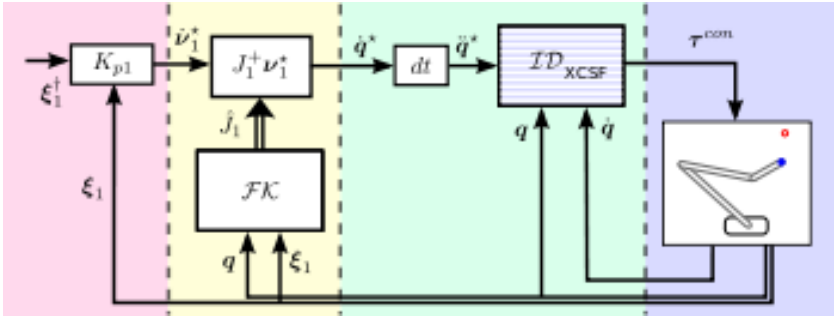




Releasing the force after 2 seconds during a point to point task.

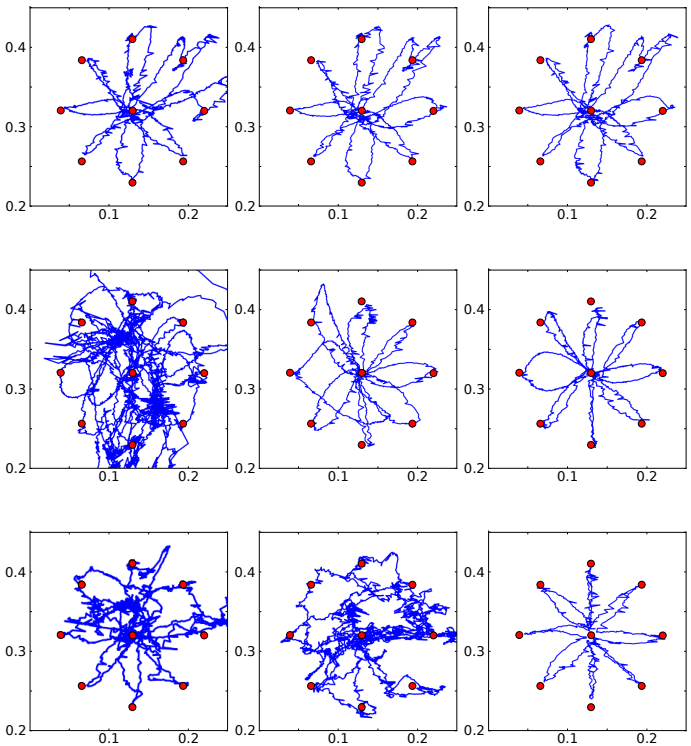
- We reproduce Shadmehr's experiments

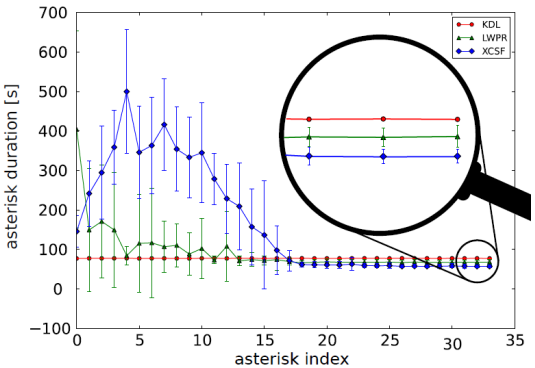
Learning dynamics

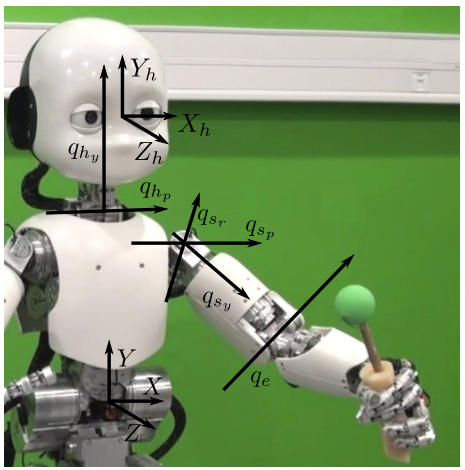


- Simulation of a three degrees of freedom planar arm

Learning forward models



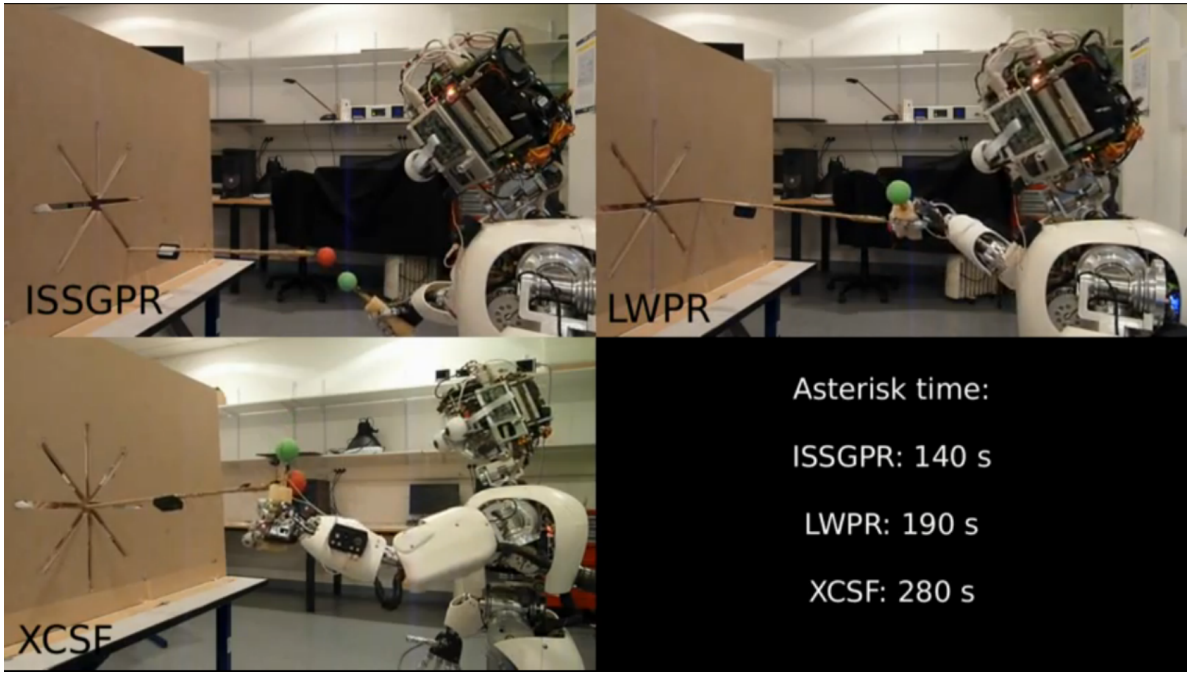




- For complex robots, the CAD model is not so accurate (calibration issue)

Sicard, G., Salaün, C., Ivaldi, S., Padois, V., and Sigaud, O. (2011) Learning the velocity kinematics of icub for model-based control: XCSF versus LWPR. In *Proceedings Humanoids 2011*, pp. 570-575.

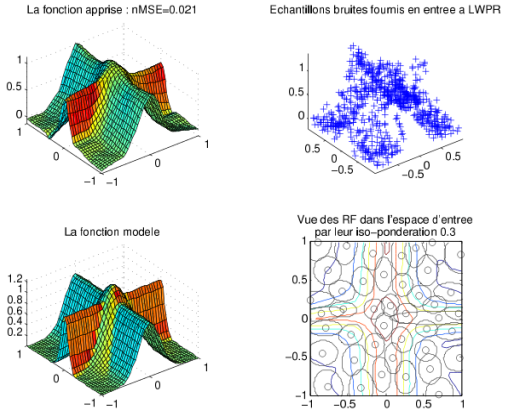
Comparing algorithms

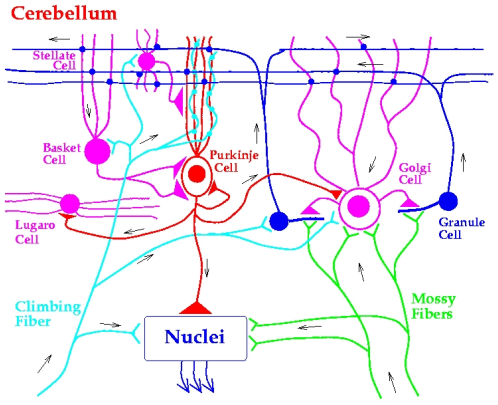


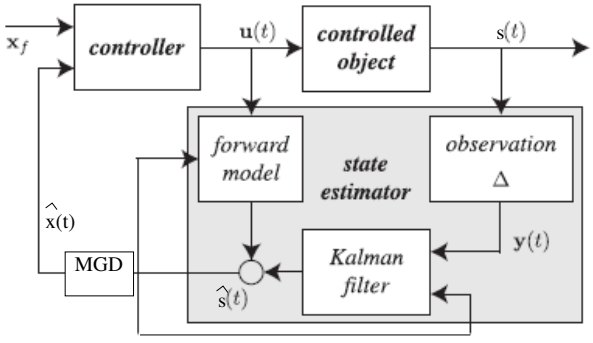
- Main difficulty: tuning parameters for fair comparison
- Many specific difficulties for robotics reproducibility

Droniou, A., Ivaldi, S., Padois, V., and Sigaud, O. (2012) Autonomous Online Learning of Velocity Kinematics on the iCub: a Comparative Study. In *IROS 2012, to appear*

Motor adaptation and the cerebellum

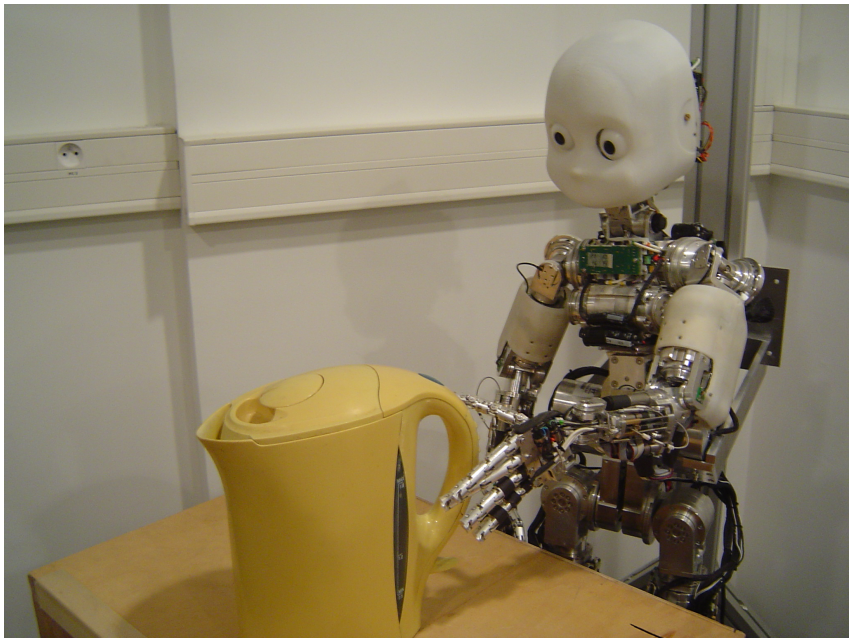






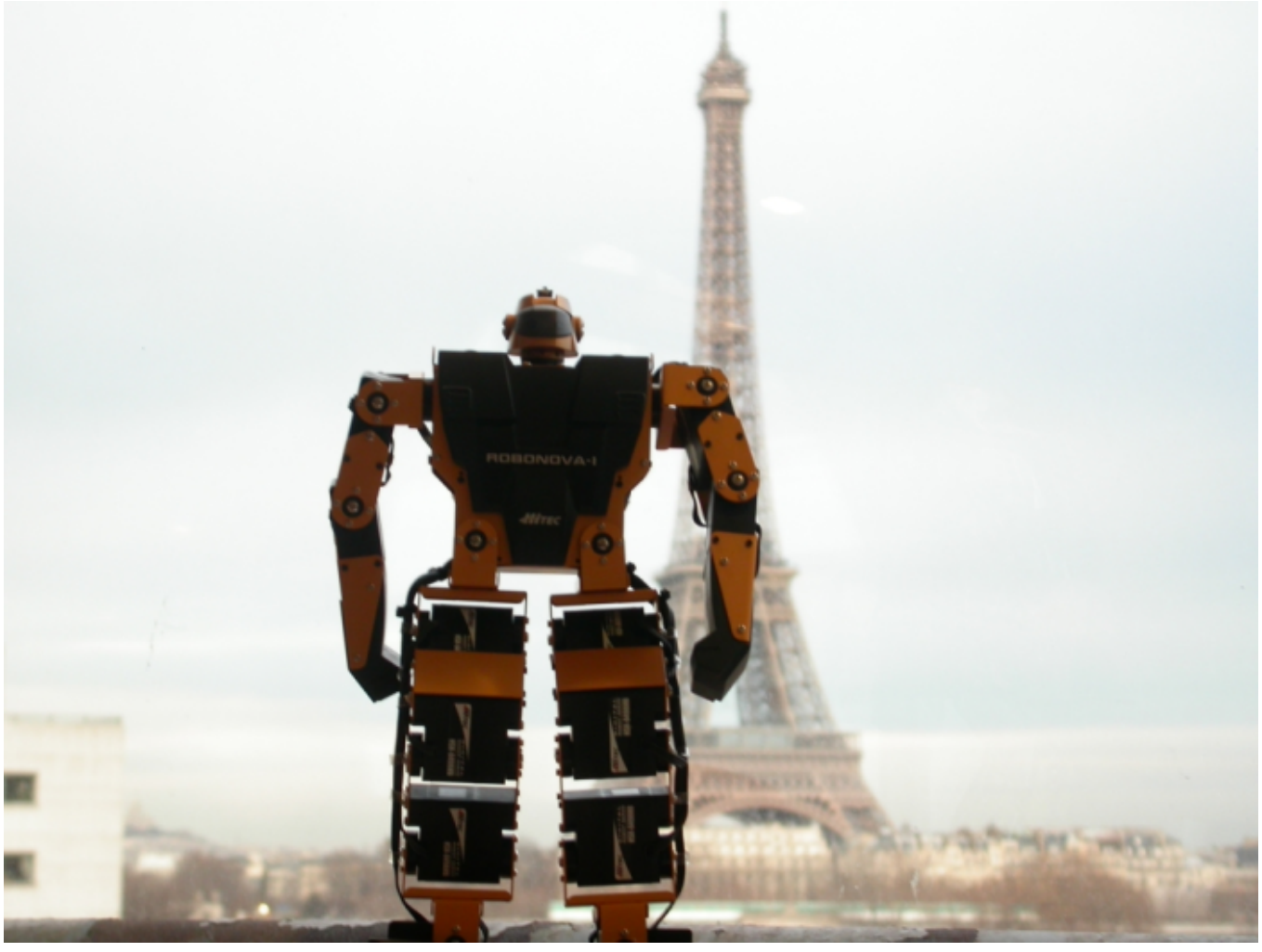
- Structural similarity between LWPR-like algos and cerebellum: Purkinje Cells = receptive fields
- + the problem of state estimation over time given delays

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?



Atkeson, C. (1991).

Using locally weighted regression for robot learning.
Edité dans *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 958–963.

Butz, M., Pedersen, G., & Stalph, P. (2009).

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

Cybenko, G. (1989).

Approximation by superpositions of a sigmoidal function.
Mathematics of Control, Signals, and Systems (MCSS), 2(4):303–314.

Droniou, A., Ivaldi, S., Padois, V., & Sigaud, O. (2012).

Autonomous online learning of velocity kinematics on the icub: a comparative study.
Edité dans *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3577–3582, Portugal.

D'Souza, A., Vijayakumar, S., & Schaal, S. (2001a).

Learning inverse kinematics.
Edité dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 298–303.

D'Souza, A., Vijayakumar, S., & Schaal, S. (2001b).

Learning inverse kinematics.
Edité dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 298–303.

Ebden, M. (2008).

Gaussian processes for regression: A quick introduction.
Rapport technique, Department on Engineering Science, University of Oxford.

Gijsberts, A. & Metta, G. (2011).

Incremental learning of robot dynamics using random features.
Edité dans *IEEE International Conference on Robotics and Automation*, pages 951–956.

Hersch, M., Guenter, F., Calinon, S., & Billard, A. (2008).

Dynamical system modulation for robot learning via kinesthetic demonstrations.
IEEE Transactions on Robotics, 24(6):1463–1467.

Holland, J. H. (1975).

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

Mitrovic, D., Klanke, S., & Vijayakumar, S. (2008).

Adaptive optimal control for redundantly actuated arms.
Edité dans *Proceedings of the Tenth International Conference on Simulation of Adaptive Behavior*, pages 93–102.

Peters, J. & Schaal, S. (2008).

Learning to control in operational space.
International Journal in Robotics Research, 27(2):197–212.

Schaal, S., Atkeson, C. G., & Vijayakumar, S. (2002).

Scalable techniques from nonparametric statistics for real time robot learning.
Applied Intelligence, 17(1):49–60.

Sicard, G., Salaün, C., Ivaldi, S., Padois, V., & Sigaud, O. (2011).

Learning the velocity kinematics of icub for model-based control: XCSF versus LWPR.
Edité dans *Proceedings of the 11th IEEE-RAS International Conference on Humanoid Robots*, pages 570 – 575, Bled, Slovenia.

Sigaud, O., Salaün, C., & Padois, V. (2011).

On-line regression algorithms for learning mechanical models of robots: a survey.
Robotics and Autonomous Systems, 59(12):1115–1129.

Stulp, F. & Sigaud, O. (2015).

Many regression algorithms, one unified model: A review.
Neural Networks, 69:60–79.

Vijayakumar, S., D'Souza, A., & Schaal, S. (2005).

LWPR: A scalable method for incremental online learning in high dimensions.
Rapport technique, Edinburgh: Press of University of Edinburgh.

Wilson, S. W. (2001).

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