F.L. Lewis, Dan Popa
Automation & Robotics Research Institute (ARRI)
The University of Texas at Arlington, USA
Guido Herrmann
Bristol Robotics Lab, University of Bristol, UK

Supported by :
NSF - Paul Werbos
ARO- Sam Stanton
AFOSR- Fariba Fahroo

Reinforcement Methods for Autonomous Online Learning of Optimal Robot Behaviors

The University of Texas ARLINGTON

Talk available online at
http://ARRI.uta.edu/acs

ARRI

Invited by Rolf Johansson

- Optimal Control
- Reinforcement learning
- Policy Iteration
- Q Learning
- Humanoid Robot Learning Control Using RL
- Telerobotic Interface Learning Using RL

It is man's obligation to explore the most difficult questions in the clearest possible way and use reason and intellect to arrive at the best answer.

Man's task is to understand patterns in nature and society.

The first task is to understand the individual problem, then to analyze symptoms and causes, and only then to design treatment and controls.

Ibn Sina 1002-1042
(Avicenna)

Importance of Feedback Control

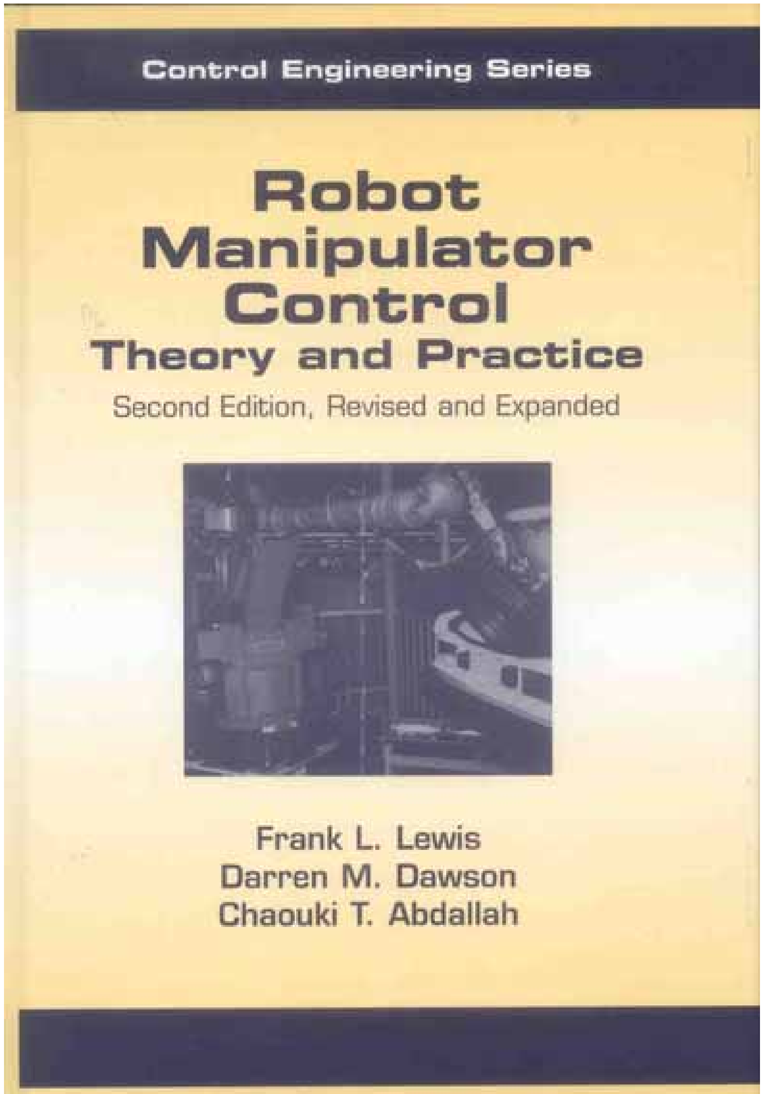
Darwin- FB and natural selection
Volterra- FB and fish population balance
Adam Smith- FB and international economy
James Watt- FB and the steam engine
FB and cell homeostasis

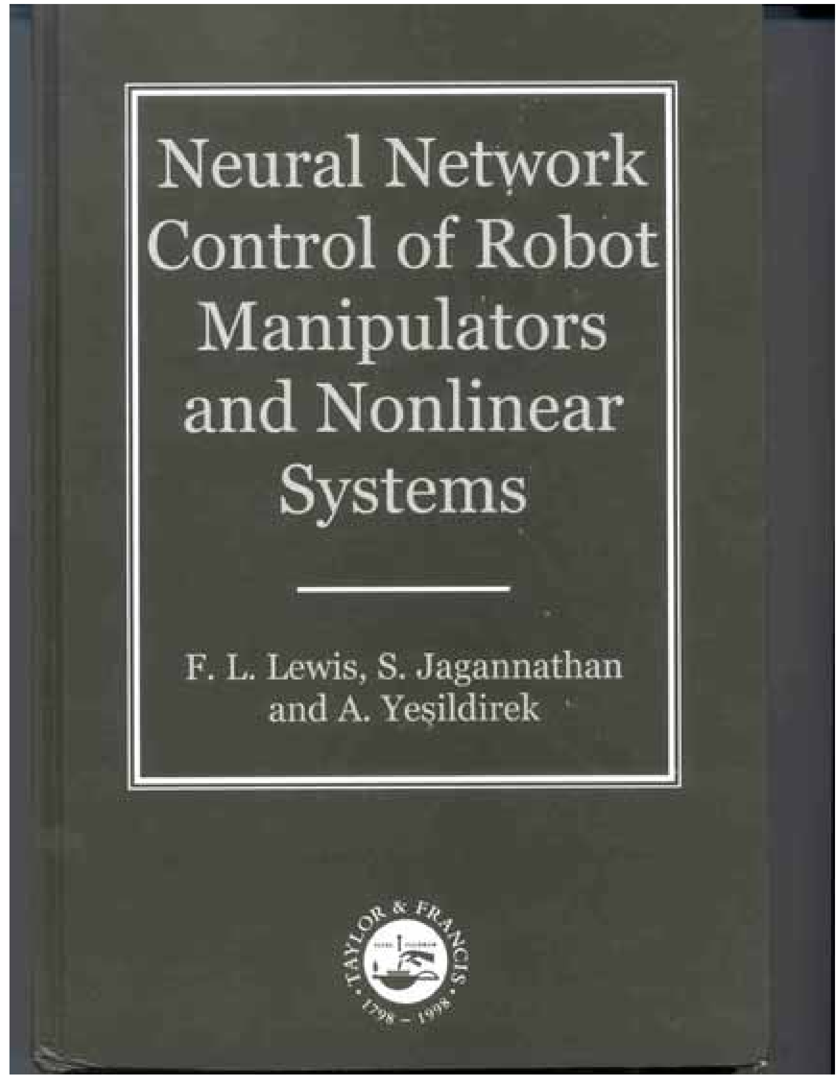
The resources available to most species for their survival are meager and limited

Nature uses Optimal control

F.L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," IEEE Circuits & Systems Magazine, Invited Feature Article, pp. 32-50, Third Quarter 2009.

IEEE Control systems magazine, to appear.





Discrete-Time Optimal Control

system $x_{k+1} = f(x_k) + g(x_k)u_k$

cost $V_h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i)$ Example $r(x_k, u_k) = x_k^T Q x_k + u_k^T R u_k$

Difference eq. equivalent $V_h(x_k) = r(x_k, u_k) + \gamma \sum_{i=k+1}^{\infty} \gamma^{i-(k+1)} r(x_i, u_i)$

Control policy $u_k = h(x_k)$ = the prescribed control input function

Example $u_k = -Kx_k$ Linear state variable feedback

Bellman equation
$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) \quad , \quad V_h(0) = 0$$
$$V_h(x_k) = x_k^T Q x_k + u_k^T R u_k + \gamma V_h(x_{k+1})$$

Bellman's Principle gives Bellman opt. eq= DT HJB
$$V^*(x_k) = \min_{u_k}(r(x_k, u_k) + \gamma V^*(x_{k+1}))$$

Optimal Control
$$h^*(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V^*(x_{k+1}))$$
$$u^*(x_k) = -\tfrac{1}{2} R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}}$$

Off-line solution
Dynamics must be known

DT Optimal Control – Linear Systems Quadratic cost (LQR)

system

$$x_{k+1} = Ax_k + Bu_k$$

cost

$$V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u_i^T R u_i$$

Fact. The cost is quadratic $V(x_k) = x_k^T P x_k$ for some symmetric matrix P

HJB = DT Riccati equation

$$0 = A^T PA - P + Q - A^T PB(R + B^T PB)^{-1} B^T PA$$

Optimal Control $u_k = -Lx_k$

$$L = (R + B^T PB)^{-1} B^T PA$$

Optimal Cost

$$V^*(x_k) = x_k^T P x_k$$

Off-line solution
Dynamics must be known

We want robot controllers that learn optimal control solutions online in real-time

Synthesis of

- Computational intelligence
- Control systems
- Neurobiology

Different methods of learning

Machine learning- the formal study of learning systems

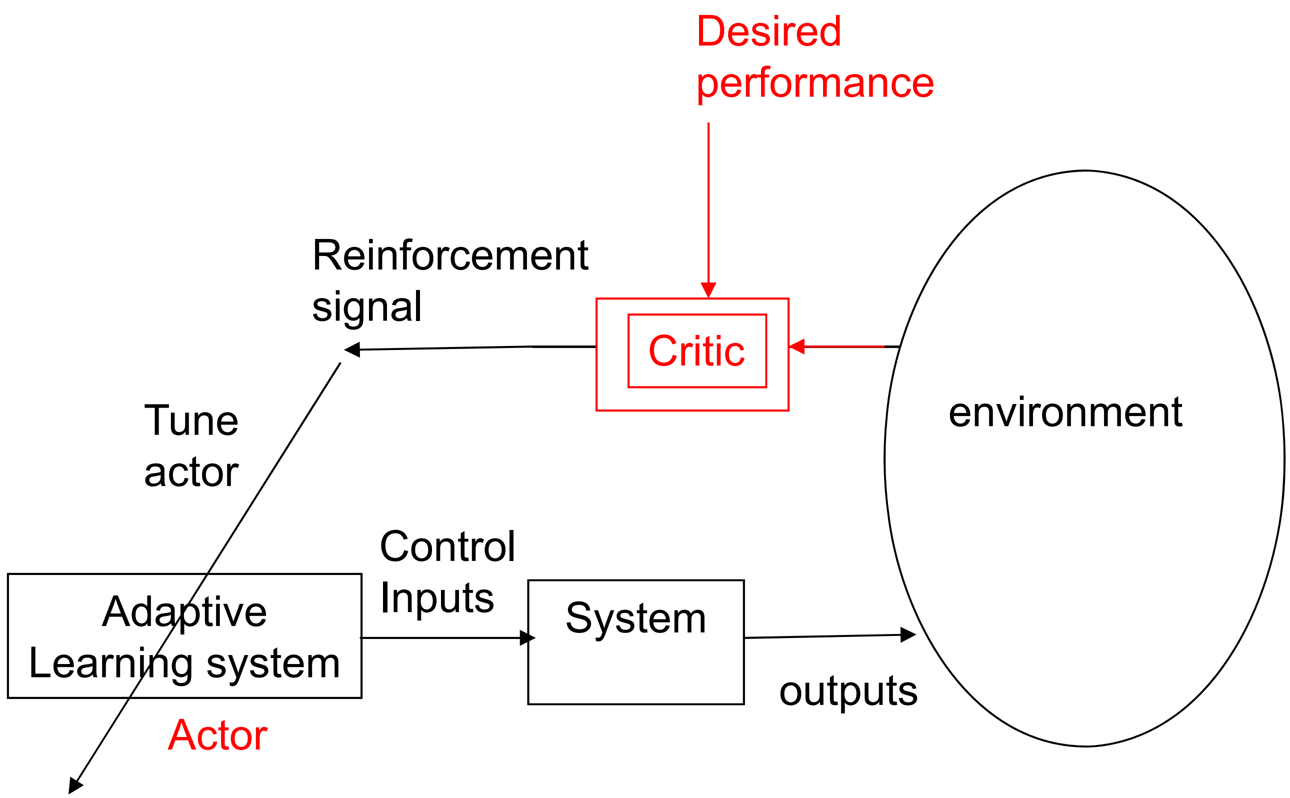
Supervised learning
Unsupervised learning
Reinforcement learning

Different methods of learning

Reinforcement learning
Ivan Pavlov 1890s

We want OPTIMAL performance
- ADP- Approximate Dynamic Programming

Actor-Critic Learning



RL Policy Iterations to Solve Optimal Control Problem

system $$x_{k+1} = f(x_k) + g(x_k)u_k$$

cost $$V_h(x_k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(x_i, u_i)$$

Difference eq. equivalent $$V_h(x_k) = r(x_k, u_k) + \gamma \sum_{i=k+1}^{\infty} \gamma^{i-(k+1)} r(x_i, u_i)$$

Bellman equation $$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1}) \quad , \quad V_h(0) = 0$$

$$\boxed{V_h(x_k) = x_k^T Q x_k + u_k^T R u_k + \gamma V_h(x_{k+1})}$$

Bellman's Principle gives Bellman opt. eq= DT HJB

$$V^*(x_k) = \min_{u_k}(r(x_k, u_k) + \gamma V^*(x_{k+1}))$$

Focus on these two eqs.

Optimal Control $$h^*(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V^*(x_{k+1}))$$

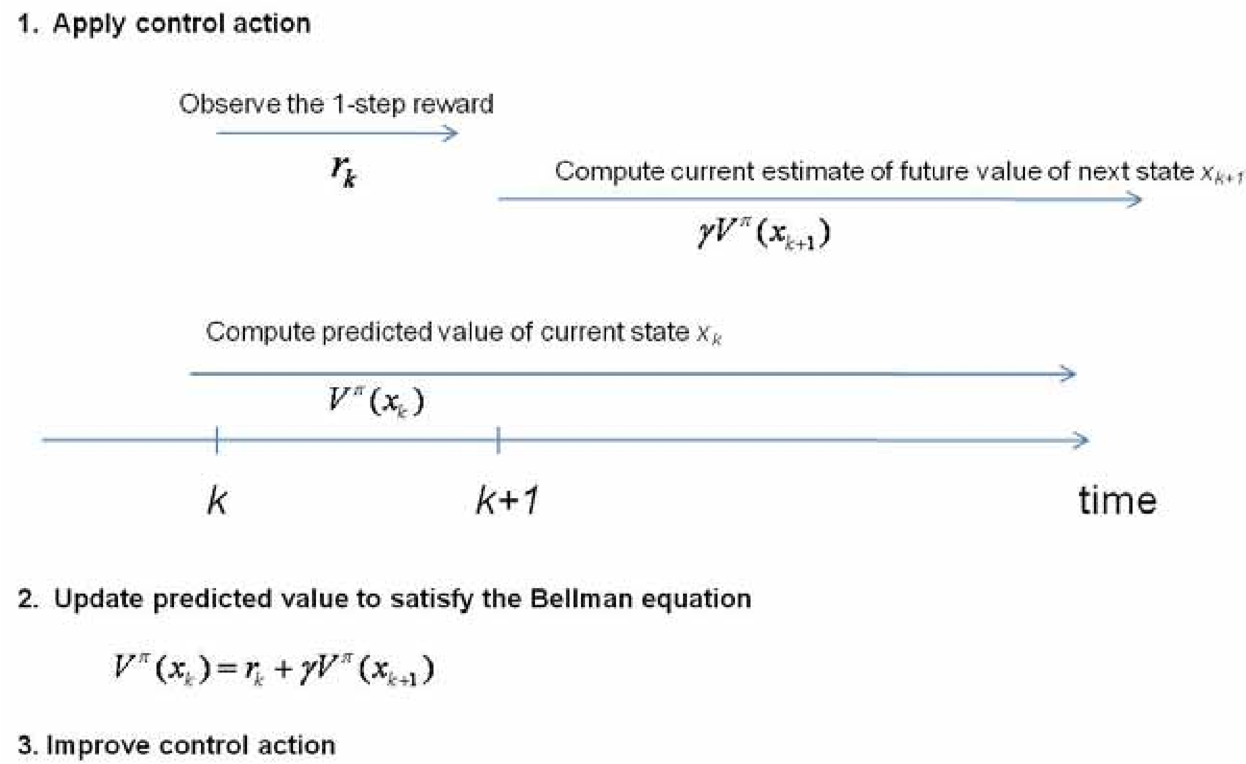
$$\boxed{u^*(x_k) = -\tfrac{1}{2} R^{-1} g(x_k)^T \frac{\partial V^*(x_{k+1})}{\partial x_{k+1}}}$$

Bellman Equation

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$

Can be interpreted as a consistency equation that must be satisfied by the value function at each time stage.

Expresses a relation between the current value of being in state *x* and the value(s) of being in next state *x'* given that policy



Captures the action, observation, evaluation, and improvement mechanisms of reinforcement learning.

Temporal Difference Idea

$$e_k = -V_h(x_k) + r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$

Policy Evaluation and Policy Improvement

consider algorithms that repeatedly interleave the two procedures:

Policy Evaluation by Bellman Equation:

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$

Policy Improvement:

$$h'(x_k) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{\partial V(x_{k+1})}{\partial x_{k+1}}$$

Policy Improvement makes $V_{h'}(x) \le V_h(x)$

(Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998).

the policy $h'(x_k)$ is said to be *greedy* with respect to value function $V_h(x)$

At each step, one obtains a policy that is no worse than the previous policy.
Can prove convergence under fairly mild conditions to the optimal value and optimal policy.
Most such proofs are based on the Banach Fixed Point Theorem.
One step is a contraction map.

There is a large family of algorithms that implement the
policy evaluation and policy improvement procedures in various ways

DT Policy Iteration to solve HJB

Cost for any given control policy $h(x_k)$ satisfies the recursion

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$

Bellman eq.

Recursive form
Consistency equation

Recursive solution

Pick stabilizing initial control

Policy Evaluation – solve Bellman Equation

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

f(.) and g(.) do not appear

Policy Improvement

$$h_{j+1}(x_{k+1}) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$

Howard (1960) proved convergence for MDP

(Bertsekas and Tsitsiklis 1996, Sutton and Barto 1998).

the policy $h_{j+1}(x_k)$ is said to be *greedy* with respect to value function $V_{j+1}(x)$

At each step, one obtains a policy that is no worse than the previous policy.
Can prove convergence under fairly mild conditions to the optimal value and optimal policy.
Most such proofs are based on the Banach Fixed Point Theorem.
One step is a contraction map.

Methods to implement Policy Iteration

- Exact Computation- needs full system dynamics
- Temporal Difference- for robot trajectory following
- Montecarlo Learning- for learning episodic robot tasks

DT Policy Iteration – Linear Systems Quadratic Cost- LQR

$$x_{k+1} = Ax_k + Bu_k = (A - BL)x_k, \qquad u_k = -Lx_k$$

For any stabilizing policy, the cost is

$$V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u^T(x_i) R u(x_i)$$

LQR value is quadratic $V(x) = x^T P x$

Solves Lyapunov eq. without knowing A and B

DT Policy iterations

$$V_{j+1}(x_k) = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k) + V_{j+1}(x_{k+1})$$

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}}$$

Equivalent to an Underlying Problem- DT LQR:

$$(A - BL_j)^T P_{j+1} (A - BL_j) - P_{j+1} = -Q - L_j^T R L_j$$

DT Lyapunov eq.

$$L_{j+1} = (R + B^T P_{j+1} B)^{-1} B^T P_{j+1} A$$

Hewer proved convergence in 1971

Policy Iteration Solves Lyapunov equation WITHOUT knowing System Dynamics

DT Policy Iteration – How to implement online?
Linear Systems Quadratic Cost- LQR

$$x_{k+1} = Ax_k + Bu_k \qquad V(x_k) = \sum_{i=k}^{\infty} x_i^T Q x_i + u(x_i) R u(x_i)$$

LQR cost is quadratic $V(x) = x^T P x$ for some matrix P

DT Policy iterations

Solves Lyapunov eq. without knowing A and B

$$V_{j+1}(x_k) = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k) + V_{j+1}(x_{k+1})$$

$$x_k^T P_{j+1} x_k - x_{k+1}^T P_{j+1} x_{k+1} = x_k^T Q x_k + u_j^T R u_j$$

$$\begin{bmatrix} x_k^1 & x_k^2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \end{bmatrix} - \begin{bmatrix} x_{k+1}^1 & x_{k+1}^2 \end{bmatrix} \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \end{bmatrix}$$

$$= \begin{bmatrix} p_{11} & p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} (x_k^1)^2 \\ 2x_k^1 x_k^2 \\ (x_k^2)^2 \end{bmatrix} - \begin{bmatrix} p_{11} & p_{12} & p_{22} \end{bmatrix} \begin{bmatrix} (x_{k+1}^1)^2 \\ 2x_{k+1}^1 x_{k+1}^2 \\ (x_{k+1}^2)^2 \end{bmatrix}$$

Quadratic basis set

$$W_{j+1}^T \left[\varphi(x_k) - \varphi(x_{k+1}) \right] = x_k^T Q x_k + u_j^T(x_k) R u_j(x_k)$$

Then update control using

$$h_j(x_k) = L_j x_k = (R + B^T P_j B)^{-1} B^T P_j A x_k$$

Need to know A AND B
for control update

Implementation- DT Policy Iteration
Nonlinear Case

Value Function Approximation (VFA)

$$V(x) = W^T \varphi(x)$$

weights

basis functions

LQR case- V(x) is quadratic

$$V(x) = x^T P x = W^T \varphi(x)$$

$$\varphi(x) = \left[x_1^2, \ldots, x_1 x_n, x_2^2, \ldots, x_2 x_n, \ldots, x_n^2\right]'.$$

Quadratic basis functions

$$W^T = [p_{11} \quad p_{12} \quad \cdots]$$

Nonlinear system case- use Neural Network

Implementation- DT Policy Iteration

Value function update for given control – Bellman Equation

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

Assume measurements of x_k and x_{k+1} are available to compute u_{k+1}

VFA $\quad V_j(x_k) = W_j^T \varphi(x_k)$

Then

regression matrix

$$W_{j+1}^T \left[\varphi(x_k) - \gamma \varphi(x_{k+1}) \right] = r(x_k, h_j(x_k))$$

Since x_{k+1} is measured,
do not need knowledge of f(x)
or g(x) for value fn. update

Solve for weights in real-time using RLS
or, batch LS- many trajectories with different initial conditions over a compact set

Then update control using

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}} = -\frac{1}{2} R^{-1} g(x_k)^T \nabla \varphi^T(x_{k+1}) W_{j+1}^T$$

Need to know $g(x_k)$ for control update

1. Select control policy

Solves Lyapunov eq. without knowing dynamics

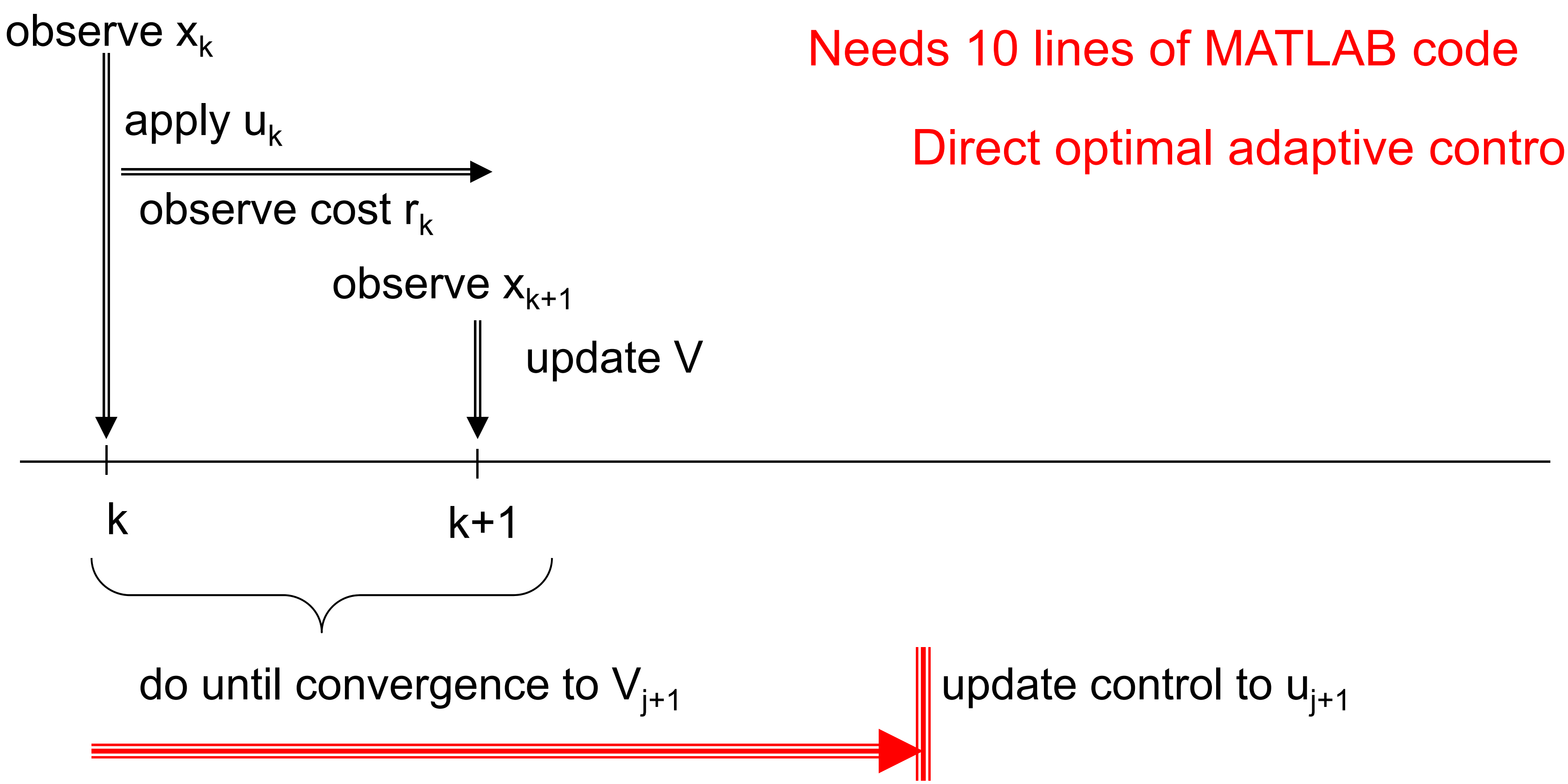
2. Find associated cost $V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$

$$W_{j+1}^T \left[\varphi(x_k) - \gamma \varphi(x_{k+1}) \right] = r(x_k, h_j(x_k))$$

3. Improve control $u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_j(x_{k+1})}{dx_{k+1}}$

Needs 10 lines of MATLAB code

Direct optimal adaptive control



Persistence of Excitation

$$W_{j+1}^T\left[\varphi(x_k)-\gamma\varphi(x_{k+1})\right]=r(x_k,h_j(x_k))$$

Regression vector must be PE

Adaptive Critics

The Adaptive Critic Architecture

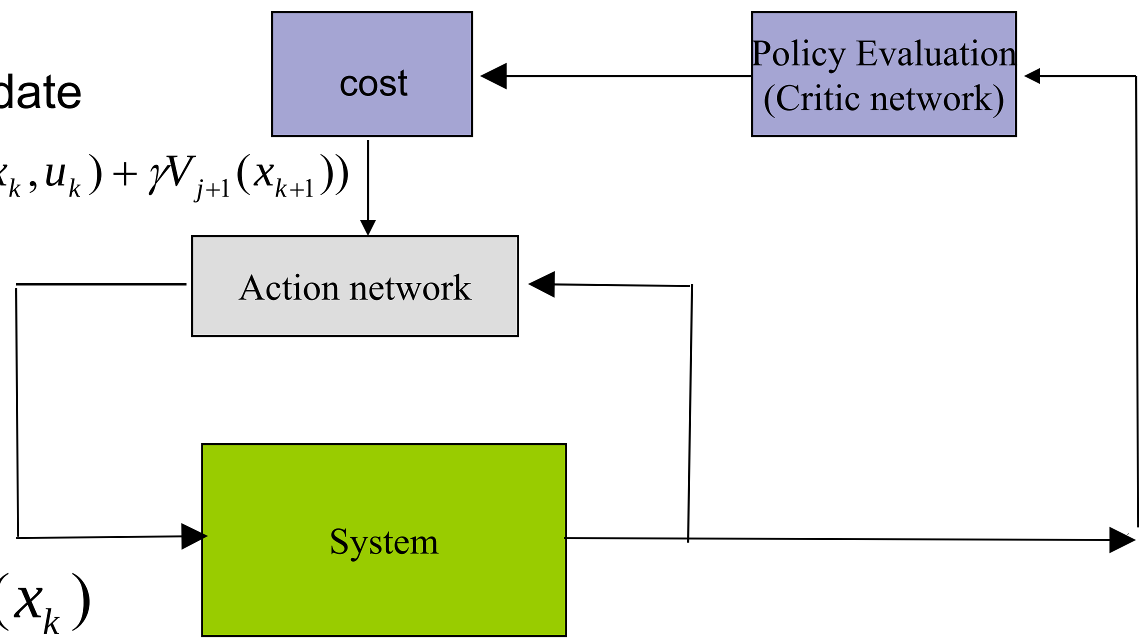
Use RLS until convergence

Value update

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

Control policy update

$$h_{j+1}(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$



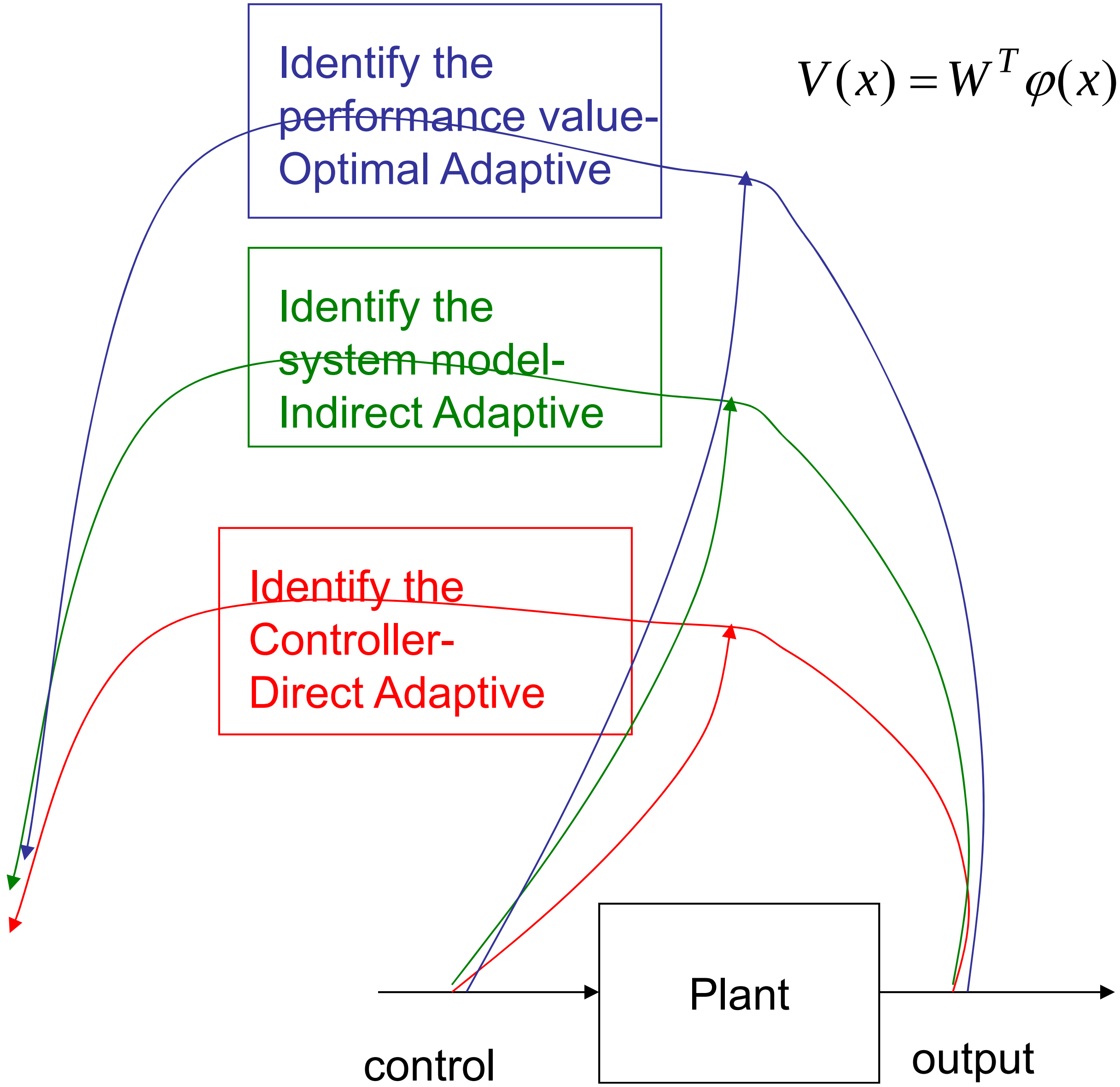
$h_j(x_k)$

Control policy

Leads to ONLINE FORWARD-IN-TIME implementation of optimal control

Optimal Adaptive Control

Adaptive Control



$$V(x) = W^T \varphi(x)$$

Greedy Value Fn. Update- Approximate Dynamic Programming
Value Iteration= Heuristic Dynamic Programming (HDP)

Paul Werbos

Policy Iteration

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

$$h_{j+1}(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$

Lyapunov eq.

For LQR
Underlying RE

$$(A - BL_j)^T P_{j+1}(A - BL_j) - P_{j+1} = -Q - L_j^T R L_j$$

Hewer 1971

$$L_j = -(R + B^T P_j B)^{-1} B^T P_j A$$

Initial stabilizing control is needed

Value Iteration

Two occurrences of cost allows def. of greedy update

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_j(x_{k+1})$$

$$h_{j+1}(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$

Simple recursion

For LQR
Underlying RE

$$P_{j+1} = (A - BL_j)^T P_j (A - BL_j) + Q + L_j^T R L_j$$

$$L_j = -(R + B^T P_j B)^{-1} B^T P_j A$$

Lancaster & Rodman
proved convergence

Initial stabilizing control is NOT needed

Estimate for the future stage cost-to-go

Compare Value Iteration

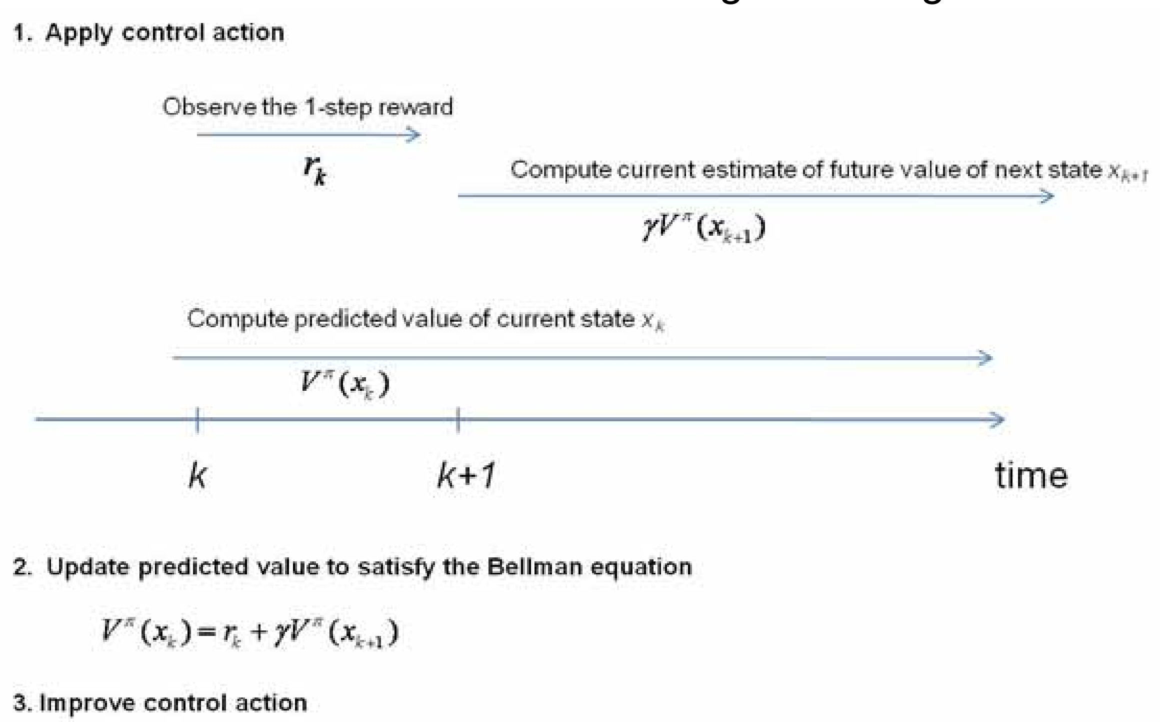
$$V_{j+1}(x) = \sum_u \pi_j(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_j(x') \right]$$

To Dynamic Programming

$$V_k^\pi(x) = \sum_u \pi(x,u) \sum_{x'} P_{xx'}^u \left[R_{xx'}^u + \gamma V_{k+1}^\pi(x') \right].$$

one can interpret $V_j(x')$

as an approximation or estimate for the future stage cost-to-go from the future state x'



A problem with DT Policy Iteration and VI

Policy Evaluation

Assume measurements of x_k and x_{k+1} are available to compute u_{k+1}

$$V_j(x_k) = W_j^T \varphi(x_k)$$

Then

$$W_{j+1}^T \left[\varphi(x_k) - \gamma\varphi(x_{k+1})\right] = r(x_k, h_j(x_k))$$

Since x_{k+1} is measured,
do not need knowledge of f(x)
or g(x) for value fn. update

Policy Improvement

$$u_{j+1}(x_{k+1}) = -\frac{1}{2} R^{-1} g(x_k)^T \frac{dV_{j+1}(x_{k+1})}{dx_{k+1}}$$

LQR case

$$h_j(x_k) = L_j x_k = (R + B^T P_j B)^{-1} B^T P_j A x_k$$

Need to know $f(x_k)$ AND $g(x_k)$
for control update

Easy to fix – use 2 NN

Standard Neural Network VFA for On-Line Implementation

Asma Al-Tamimi & F. Lewis

NN for Value - Critic

$$\hat{V}_i(x_k, W_{Vi}) = W_{Vi}^T \phi(x_k)$$

NN for control action

$$\hat{u}_i(x_k, W_{ui}) = W_{ui}^T \sigma(x_k)$$

(can use 2-layer NN)

HDP

$$V_{i+1}(x_k) = x_k^T Q x_k + u^T R u + V_i(x_{k+1})$$

$$x_{k+1} = f(x_k) + g(x_k) u(x_k)$$

$$u_i(x_k) = \arg\min_u (x_k^T Q x_k + u^T R u + V_i(x_{k+1}))$$

Define target cost function

$$\begin{aligned} d(\phi(x_k), W_{Vi}^T) &= x_k^T Q x_k + \hat{u}_i^T(x_k) R \hat{u}_i(x_k) + \hat{V}_i(x_{k+1}) \\ &= x_k^T Q x_k + \hat{u}_i^T(x_k) R \hat{u}_i(x_k) + W_{Vi}^T \phi(x_{k+1}) \end{aligned}$$

Explicit equation for cost – use LS for Critic NN update or RLS

$$W_{Vi+1} = \arg\min_{W_{Vi+1}} \{ \int_\Omega | W_{Vi+1}^T \phi(x_k) - d(\phi(x_k), W_{Vi}^T) |^2 \, dx_k \} \Longrightarrow W_{Vi+1} = \left(\int_\Omega \phi(x_k) \phi(x_k)^T dx \right)^{-1} \int_\Omega \phi(x_k) d^T(\phi(x_k), W_{Vi}^T, W_{ui}^T) dx$$

or $$W_{Vi+1}\big|_{m+1} = W_{Vi+1}\big|_m + \beta \phi^T(x_k) \left(-W_{Vi+1}^T\big|_m \phi(x_k) + r(x_k, u_k) + W_{Vi}^T \phi(x_{k+1}) \right)$$

Implicit equation for DT control- use gradient descent for action update

$$W_{ui} = \arg\min_W \left(\begin{aligned} & x_k^T Q x_k + \hat{u}^T(x_k, W) R \hat{u}(x_k, W) + \\ & \hat{V}_i(f(x_k) + g(x_k) \hat{u}(x_k, W)) \end{aligned} \right) \Bigg|_\Omega \Longrightarrow W_{ui(j+1)} = W_{ui(j)} - \alpha \frac{\partial (x_k^T Q x_k + \hat{u}_{i(j)}^T R \hat{u}_{i(j)} + \hat{V}_i(x_{k+1}))}{\partial W_{ui(j)}}$$

$$W_{ui}^{j+1} = W_{ui}^j - \alpha \sigma(x_k)(2R\hat{u}_{i(j)} + g(x_k)^T \frac{\partial \phi^T(x_{k+1})}{\partial x_{k+1}} W_{Vi})^T$$

Backpropagation- P. Werbos

Interesting Fact for HDP for Nonlinear systems

Linear Case $h_j(x_k) = L_j x_k = -(I + B^T P_j B)^{-1} B^T P_j A x_k$

must know system A and B matrices

NN for control action

Information about A is stored in NN

$$\hat{u}_i(x_k, W_{ui}) = W_{ui}^T \sigma(x_k)$$

Implicit equation for DT control- use gradient descent for action update

$$W_{ui} = \arg\min_{\alpha} \left(\begin{array}{l} x_k^T Q x_k + \hat{u}^T(x_k,\alpha) R \hat{u}(x_k,\alpha) + \\ \hat{V}_i(f(x_k) + g(x_k)\hat{u}(x_k,\alpha)) \end{array} \right) \Bigg|_{\Omega} \Longrightarrow$$

$$W_{ui(j+1)} = W_{ui(j)} - \alpha \frac{\partial (x_k^T Q x_k + \hat{u}_{i(j)}^T R \hat{u}_{i(j)} + \hat{V}_i(x_{k+1}))}{\partial W_{ui(j)}}$$

$$W_{ui}^{j+1} = W_{ui}^{j} - \alpha \sigma(x_k)(2R\hat{u}_{i(j)} + g(x_k)^T \frac{\partial \phi(x_{k+1})}{\partial x_{k+1}} W_{Vi})^T$$

g(.) is needed

Note that state drift dynamics $f(x_k)$ is NOT needed since:

1. NN Approximation for action is used
2. x_{k+1} is measured in training phase

Discrete-time nonlinear HJB solution using Approximate dynamic programming

- Simulation Example 1
- Linear system – Aircraft longitudinal dynamics

$$A=\begin{bmatrix} 1.0722 & 0.0954 & 0 & -0.0541 & -0.0153 \\ 4.1534 & 1.1175 & 0 & -0.8000 & -0.1010 \\ 0.1359 & 0.0071 & 1.0 & 0.0039 & 0.0097 \\ 0 & 0 & 0 & 0.1353 & 0 \\ 0 & 0 & 0 & 0 & 0.1353 \end{bmatrix} \quad B=\begin{bmatrix} -0.0453 & -0.0175 \\ -1.0042 & -0.1131 \\ 0.0075 & 0.0134 \\ 0.8647 & 0 \\ 0 & 0.8647 \end{bmatrix}$$

Unstable, Two-input system

$$0 = A^T PA - P + Q - A^T PB(R + B^T PB)^{-1} B^T PA$$

- The HJB, i.e. ARE, Solution

$$P=\begin{bmatrix} 55.8348 & 7.6670 & 16.0470 & -4.6754 & -0.7265 \\ 7.6670 & 2.3168 & 1.4987 & -0.8309 & -0.1215 \\ 16.0470 & 1.4987 & 25.3586 & -0.6709 & 0.0464 \\ -4.6754 & -0.8309 & -0.6709 & 1.5394 & 0.0782 \\ -0.7265 & -0.1215 & 0.0464 & 0.0782 & 1.0240 \end{bmatrix} \quad L=\begin{bmatrix} -4.1136 & -0.7170 & -0.3847 & 0.5277 & 0.0707 \\ -0.6315 & -0.1003 & 0.1236 & 0.0653 & 0.0798 \end{bmatrix}$$

Discrete-time nonlinear HJB solution using Approximate dynamic programming

- Simulation
- The Cost function approximation – quadratic basis set

$$\hat{V}_{i+1}(x_k, W_{Vi+1}) = W_{Vi+1}^T \phi(x_k)$$

$$\phi^T(x) = \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 & x_1x_4 & x_1x_5 & x_2^2 & x_2x_3 & x_4x_2 & x_2x_5 & x_3^2 & x_3x_4 & x_3x_5 & x_4^2 & x_4x_5 & x_5^2 \end{bmatrix}$$

$$W_V^T = \begin{bmatrix} w_{V1} & w_{V2} & w_{V3} & w_{V4} & w_{V5} & w_{V6} & w_{V7} & w_{V8} & w_{V9} & w_{V10} & w_{V11} & w_{V12} & w_{V13} & w_{V14} & w_{V15} \end{bmatrix}$$

- The Policy approximation – linear basis set

$$\hat{u}_i = W_{ui}^T \sigma(x_k)$$

$$\sigma^T(x) = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}$$

$$W_u^T = \begin{bmatrix} w_{u11} & w_{u12} & w_{u13} & w_{u14} & w_{u15} \\ w_{u21} & w_{u22} & w_{u23} & w_{u24} & w_{u25} \end{bmatrix}$$

Discrete-time nonlinear HJB solution using Approximate dynamic programming

- Simulation

The convergence of the cost

$$W_V^T = [55.5411 \quad 15.2789 \quad 31.3032 \quad -9.3255 \quad -1.4536 \quad 2.3142 \quad 2.9234 \quad -1.6594 \quad -0.2430$$

$$24.8262 \quad -1.3076 \quad 0.0920 \quad 1.5388 \quad 0.1564 \quad 1.0240]$$

$$\begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} & P_{15} \\ P_{21} & P_{22} & P_{23} & P_{24} & P_{25} \\ P_{31} & P_{32} & P_{33} & P_{34} & P_{35} \\ P_{41} & P_{42} & P_{43} & P_{44} & P_{45} \\ P_{51} & P_{52} & P_{53} & P_{54} & P_{55} \end{bmatrix} = \begin{bmatrix} w_{V1} & 0.5w_{V2} & 0.5w_{V3} & 0.5w_{V4} & 0.5w_{V5} \\ 0.5w_{V2} & w_{V6} & 0.5w_{V7} & 0.5w_{V8} & 0.5w_{V9} \\ 0.5w_{V3} & 0.5w_{V7} & w_{V10} & 0.5w_{V11} & 0.5w_{V12} \\ 0.5w_{V4} & 0.5w_{V8} & 0.5w_{V11} & w_{V13} & 0.5w_{V14} \\ 0.5w_{V5} & 0.5w_{V9} & 0.5w_{V12} & 0.5w_{V14} & w_{V15} \end{bmatrix}$$

Actual ARE soln:

$$P = \begin{bmatrix} 55.8348 & 7.6670 & 16.0470 & -4.6754 & -0.7265 \\ 7.6670 & 2.3168 & 1.4987 & -0.8309 & -0.1215 \\ 16.0470 & 1.4987 & 25.3586 & -0.6709 & 0.0464 \\ -4.6754 & -0.8309 & -0.6709 & 1.5394 & 0.0782 \\ -0.7265 & -0.1215 & 0.0464 & 0.0782 & 1.0240 \end{bmatrix}$$

Discrete-time nonlinear HJB solution using Approximate dynamic programming

- Simulation

  The convergence of the control policy

$$W_u = \begin{bmatrix} 4.1068 & 0.7164 & 0.3756 & -0.5274 & -0.0707 \\ 0.6330 & 0.1005 & -0.1216 & -0.0653 & -0.0798 \end{bmatrix}$$

$$\begin{bmatrix} L_{11} & L_{12} & L_{13} & L_{14} & L_{15} \\ L_{21} & L_{22} & L_{23} & L_{24} & L_{25} \end{bmatrix} = -\begin{bmatrix} w_{u11} & w_{u12} & w_{u13} & w_{u14} & w_{u15} \\ w_{u21} & w_{u22} & w_{u23} & w_{u24} & w_{u25} \end{bmatrix}$$

Actual optimal ctrl. $L = \begin{bmatrix} -4.1136 & -0.7170 & -0.3847 & 0.5277 & 0.0707 \\ -0.6315 & -0.1003 & 0.1236 & 0.0653 & 0.0798 \end{bmatrix}$

$$0 = A^T PA - P + Q - A^T PB(R + B^T PB)^{-1} B^T PA$$

Note- In this example, drift dynamics matrix A is NOT Needed.
Riccati equation solved online without knowing A matrix

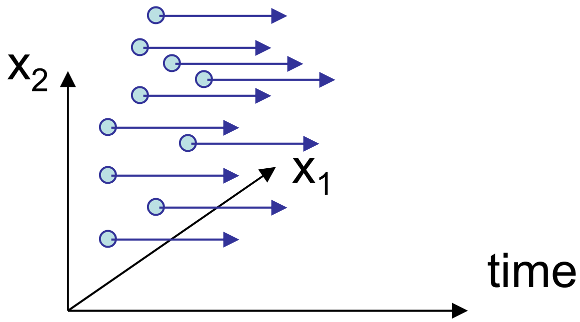
Issues with Nonlinear ADP

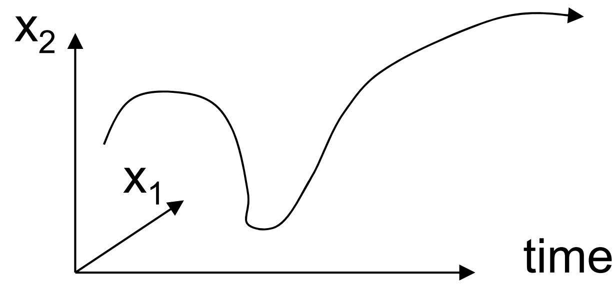
LS solution for Critic NN update

Selection of NN Training Set

$$W_{Vi+1}=\left(\int_{\Omega}\phi(x_k)\phi(x_k)^T\,dx\right)^{-1}\int_{\Omega}\phi(x_k)d^T(\phi(x_k),W_{Vi}^T,W_{ui}^T)\,dx$$

$$W_{Vi+1}\big|_{m+1}=W_{Vi+1}\big|_{m}+\beta\phi^T(x_k)\left(-W_{Vi+1}^T\big|_{m}\boldsymbol{\phi}(x_k)+r(x_k,u_k)+W_{Vi}^T\boldsymbol{\phi}(x_{k+1})\right)$$





Integral over a region of state-space
Approximate using a set of points

Batch LS

Take sample points along a single trajectory

Recursive Least-Squares RLS

Set of points over a region vs. points along a trajectory

For Linear systems- these are the same under PE condition

Exploitation (optimal regulation) vs Exploration

PE allows local smooth solution of Bellman eq.

Adaptive Critics

Use RLS until convergence

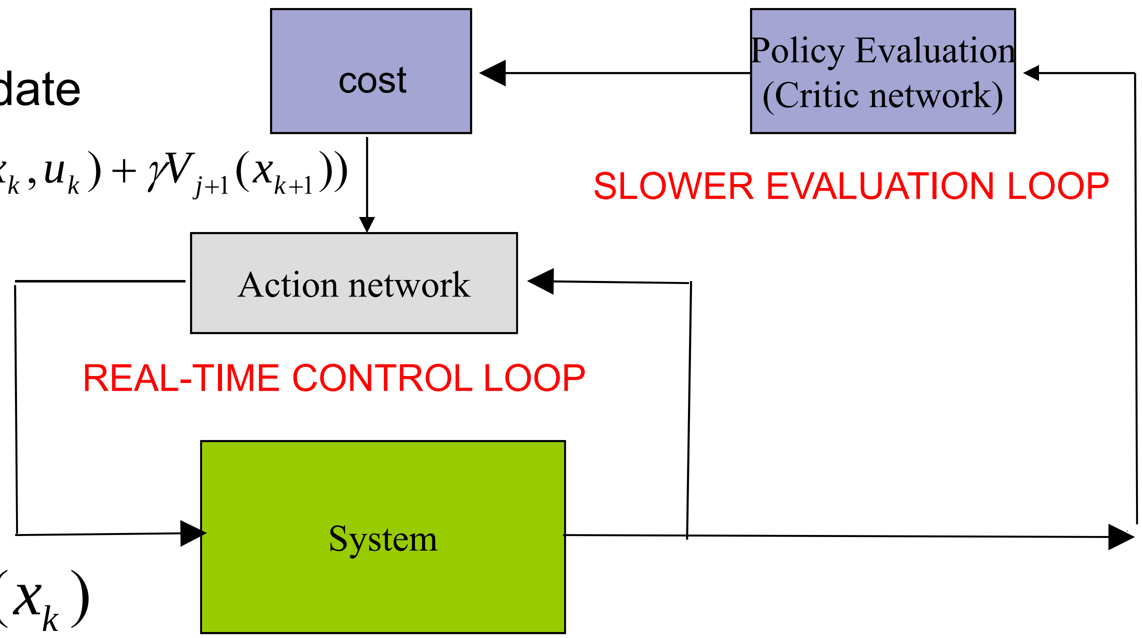
The Adaptive Critic Architecture

Value update

$$V_{j+1}(x_k) = r(x_k, h_j(x_k)) + \gamma V_{j+1}(x_{k+1})$$

Control policy update

$$h_{j+1}(x_k) = \arg\min_{u_k}(r(x_k, u_k) + \gamma V_{j+1}(x_{k+1}))$$



$h_j(x_k)$

Control policy

Leads to ONLINE FORWARD-IN-TIME implementation of optimal control

Optimal Adaptive Control

Oscillation is a fundamental property of neural tissue

Brain has multiple adaptive clocks with different timescales

gamma rhythms 30-100 Hz, hippocampus and neocortex
high cognitive activity.

- consolidation of memory
- spatial mapping of the environment – place cells

The high frequency processing is due to the large amounts of sensorial data to be processed

theta rhythm, Hippocampus, Thalamus, 4-10 Hz
sensory processing, memory and voluntary control of movement.

Spinal cord

Motor control 200 Hz

D. Vrabie, F. Lewis, and Dr. Dan Levine- RL for Continuous-Time Systems

Il Hong Suh-

Gregory

Popper

Skinner

Darwin

Limbic system

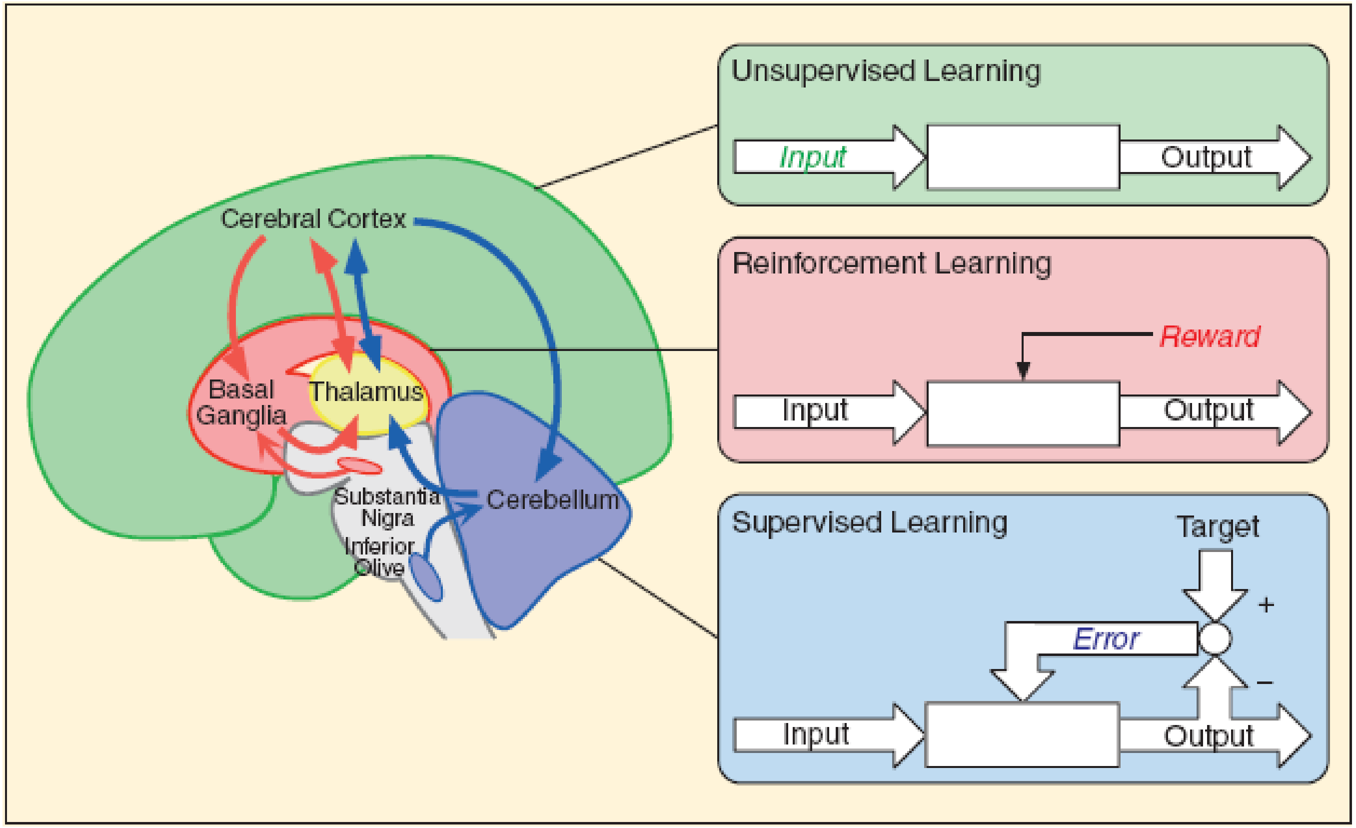
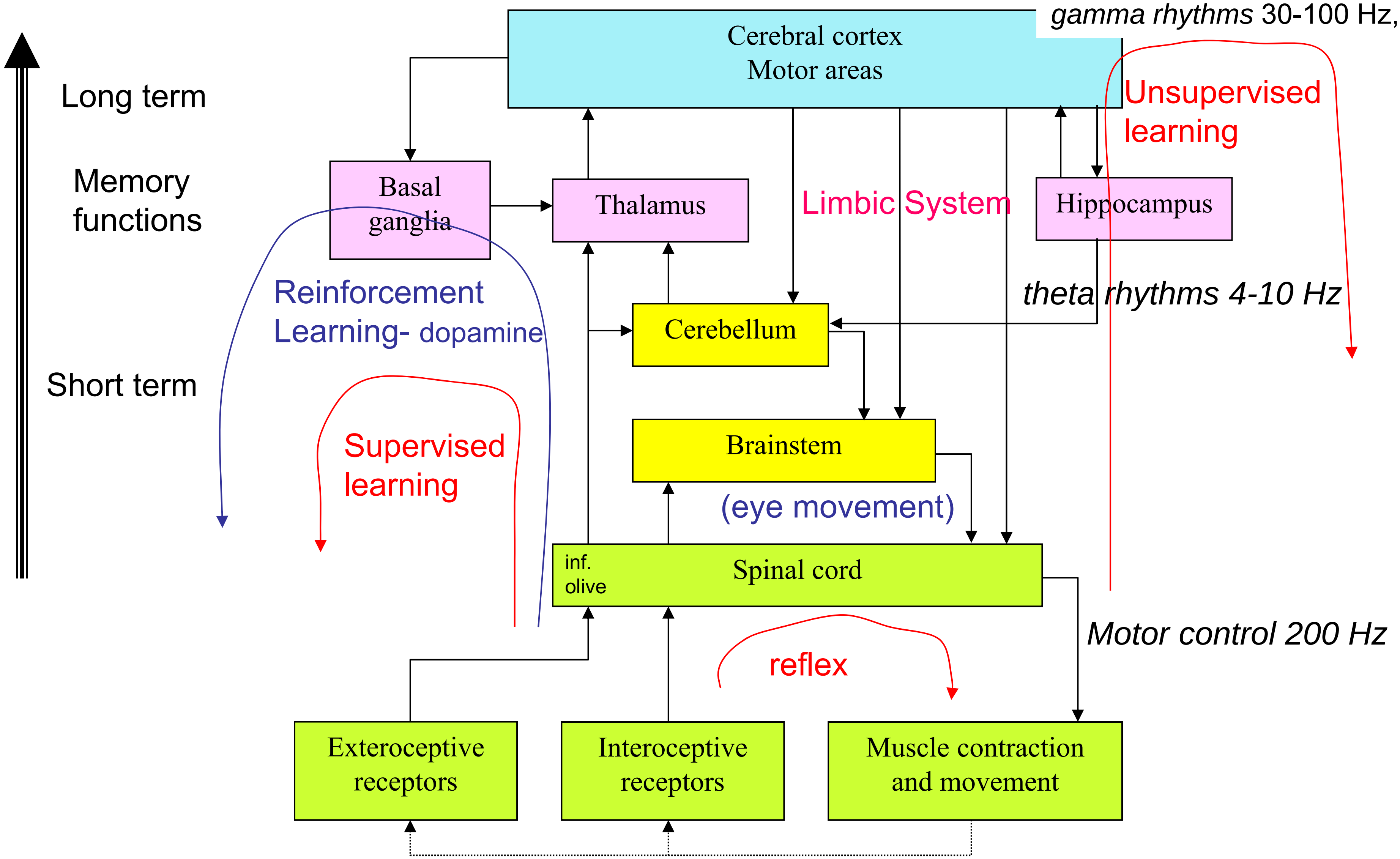


Figure 1. *Learning-oriented specialization of the cerebellum, the basal ganglia, and the cerebral cortex [1], [2]. The cerebellum is specialized for supervised learning based on the error signal encoded in the climbing fibers from the inferior olive. The basal ganglia are specialized for reinforcement learning based on the reward signal encoded in the dopaminergic fibers from the substantia nigra. The cerebral cortex is specialized for unsupervised learning based on the statistical properties of the input signal.*

Doya, Kimura, Kawato 2001

Summary of Motor Control in the Human Nervous System

picture by E. Stingu
D. Vrabie

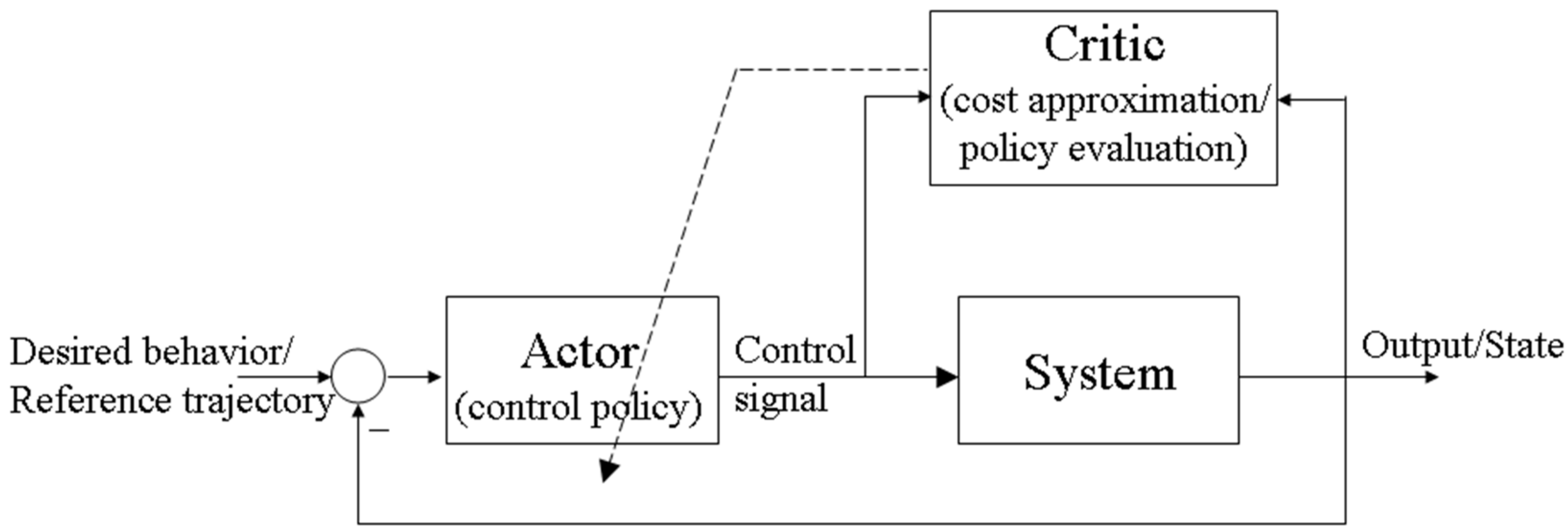


Hierarchy of multiple parallel loops

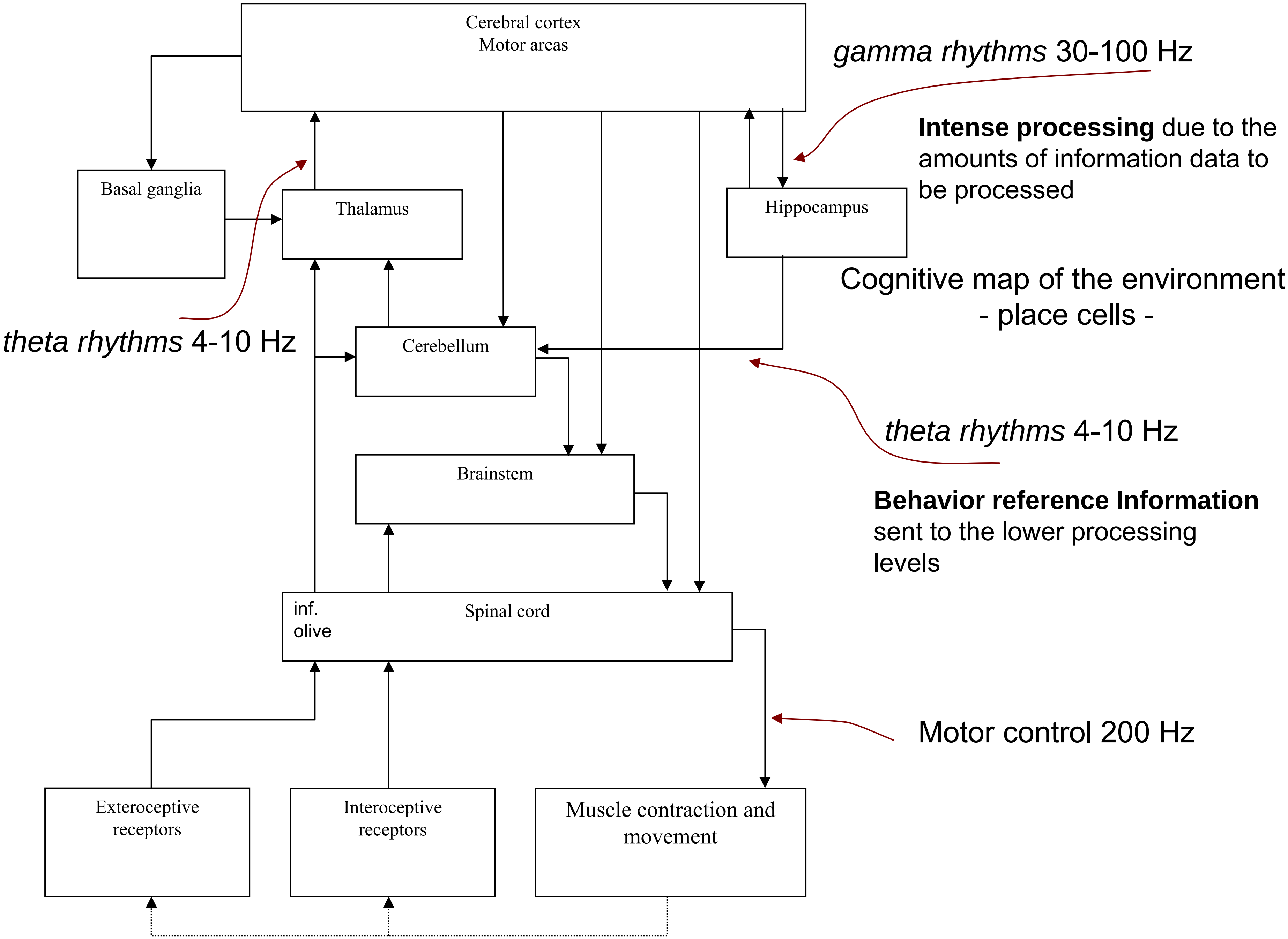
Adaptive Critic structure

Reinforcement learning

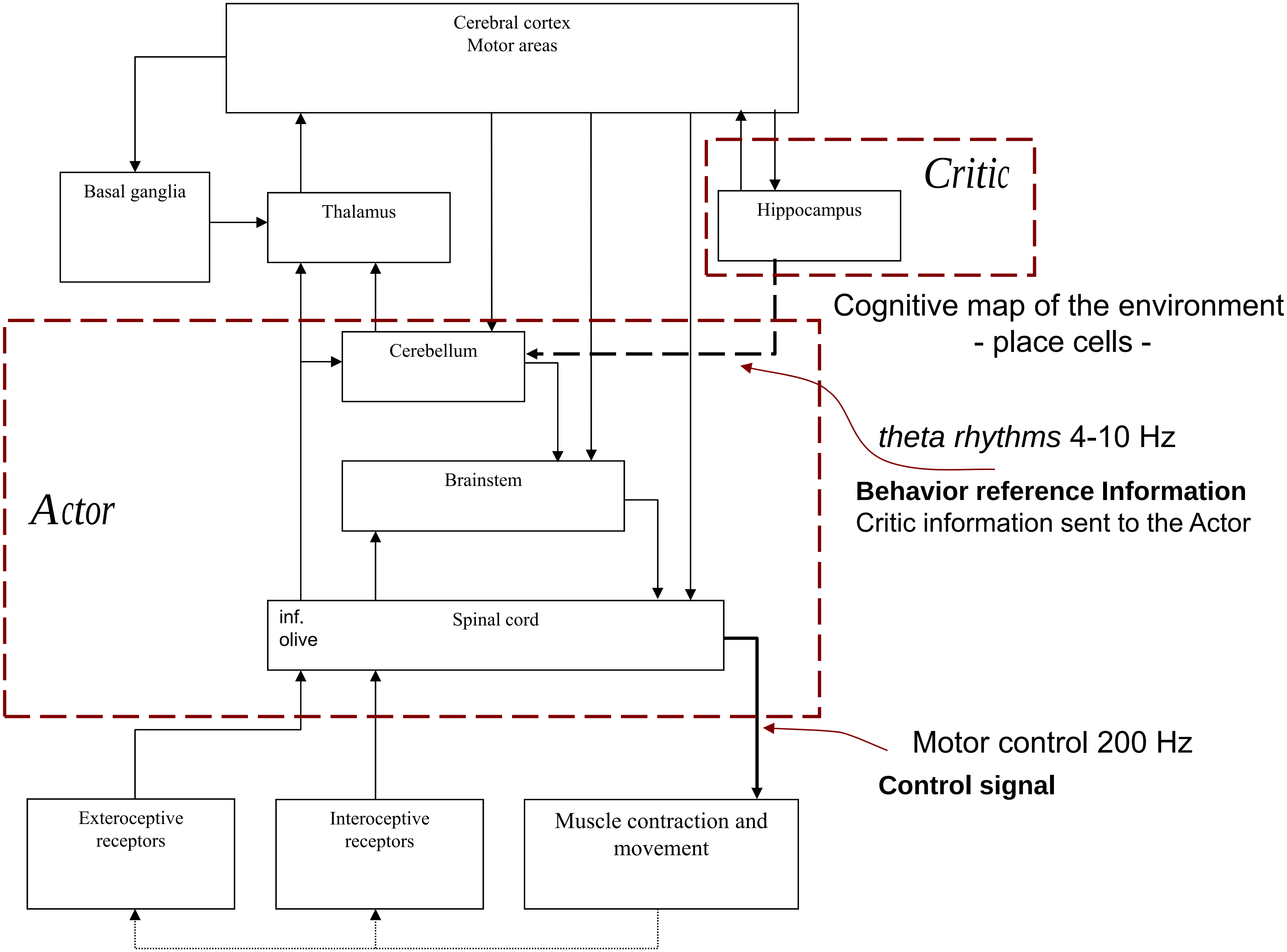
Theta waves 4-8 Hz



Motor control 200 Hz



Cerebral cortex
Motor areas
Basal ganglia
Thalamus
Hippocampus
Cerebellum
Brainstem
inf.
olive
Spinal cord
Exteroceptive receptors
Interoceptive receptors
Muscle contraction and movement
gamma rhythms 30-100 Hz
Intense processing due to the amounts of information data to be processed
Cognitive map of the environment
- place cells -
theta rhythms 4-10 Hz
theta rhythms 4-10 Hz
Behavior reference Information sent to the lower processing levels
Motor control 200 Hz


Cerebral cortex
Motor areas
Basal ganglia
Thalamus
Hippocampus
Critic
Cognitive map of the environment
- place cells -
Cerebellum
Actor
theta rhythms 4-10 Hz
Behavior reference Information
Critic information sent to the Actor
Brainstem
inf.
olive
Spinal cord
Motor control 200 Hz
Control signal
Exteroceptive
receptors
Interoceptive
receptors
Muscle contraction and
movement

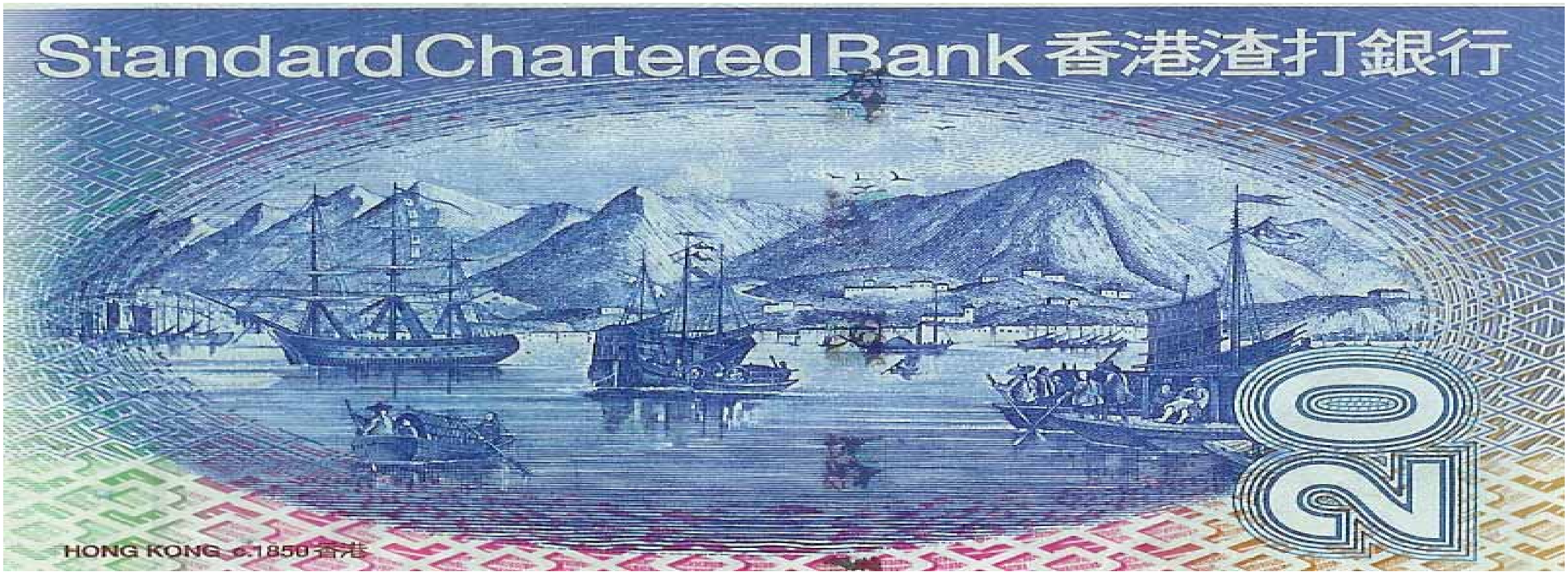
Synthesis of

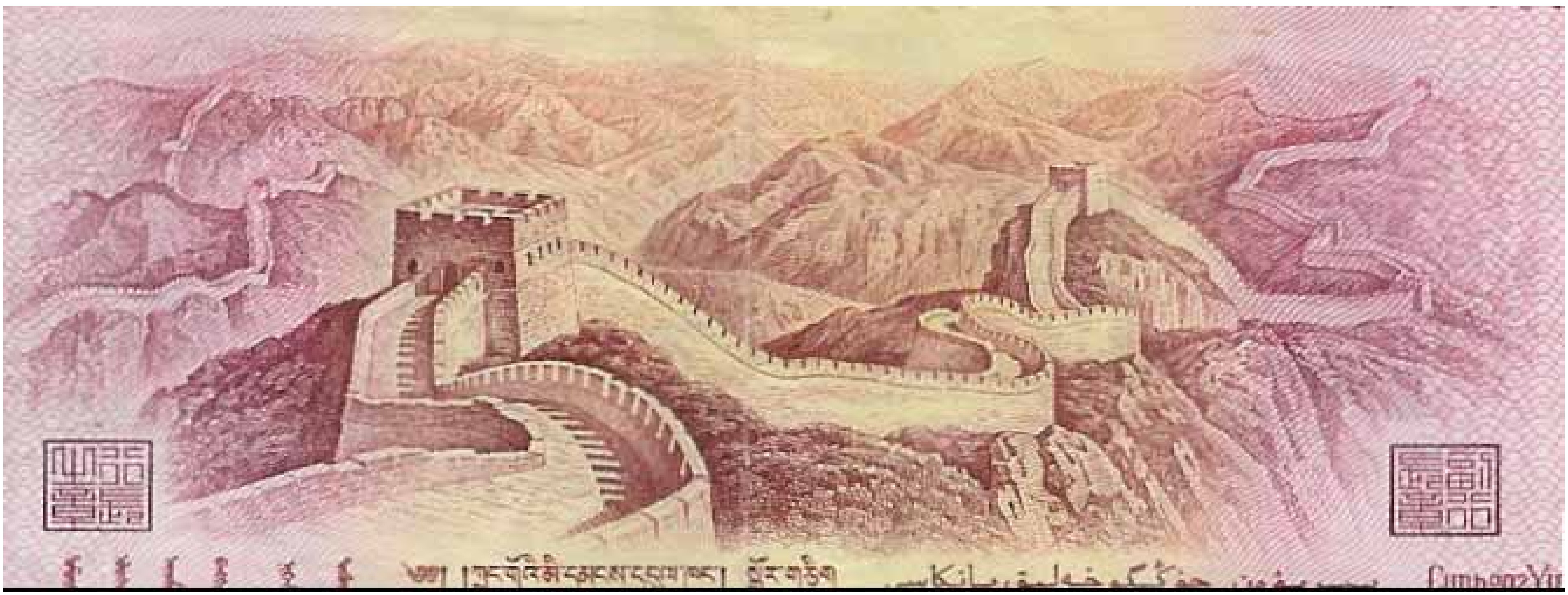
- Computational intelligence
- Control systems
- Neurobiology

Different methods of learning

Machine learning- the formal study of learning systems

Supervised learning
Unsupervised learning
Reinforcement learning


Standard Chartered Bank 香港渣打銀行
20
HONG KONG c.1850 香港



Adaptive (Approximate) Dynamic Programming

Four ADP Methods proposed by Paul Werbos

Critic NN to approximate:

Heuristic dynamic programming

Value Iteration

Value $V(x_k)$

AD Heuristic dynamic programming
(Watkins Q Learning)

Q function $Q(x_k, u_k)$

Dual heuristic programming

Gradient - costate $\dfrac{\partial V}{\partial x}$

AD Dual heuristic programming

Gradients $\dfrac{\partial Q}{\partial x}, \dfrac{\partial Q}{\partial u}$

Action NN to approximate the Control

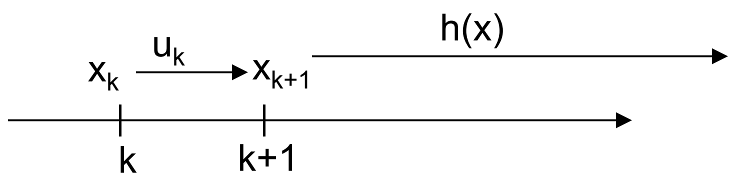
Bertsekas- Neurodynamic Programming

Barto & Bradtke- Q-learning proof (Imposed a settling time)

Q Learning- Watkins   Action Dependent ADP – Paul Werbos

Value function recursion for given policy $h(x_k)$

$$V_h(x_k) = r(x_k, h(x_k)) + \gamma V_h(x_{k+1})$$



Define Q function

$$Q_h(x_k, \underline{u_k}) = r(x_k, \underline{u_k}) + \gamma V_h(x_{k+1})$$

u_k arbitrary

policy *h(.)* used after time k

Note $\quad Q_h(x_k, h(x_k)) = V_h(x_k)$

Bellman eq for Q $\quad Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1}))$

Simple expression of Bellman's principle

$$V^*(x_k) = \min_{u_k}(Q^*(x_k, u_k)) \qquad h^*(x_k) = \arg\min_{u_k}(Q^*(x_k, u_k))$$

Optimal Adaptive Control for completely unknown DT systems

Q Function Definition

Specify a control policy $u_j = h(x_j); \quad j = k, k+1, \ldots$

Define Q function

$$Q_h(x_k, \underline{u_k}) = r(x_k, \underline{u_k}) + \gamma V_h(x_{k+1})$$

u_k arbitrary

policy *h(.)* used after time k

Note $Q_h(x_k, h(x_k)) = V_h(x_k)$

Bellman equation for Q $Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1}))$

Optimal Q function $Q^*(x_k, u_k) = r(x_k, u_k) + \gamma V^*(x_{k+1}))$

$$Q^*(x_k, u_k) = r(x_k, u_k) + \gamma Q^*(x_{k+1}, h^*(x_{k+1}))$$

Optimal control solution

$$V^*(x_k) = Q^*(x_k, h^*(x_k)) = \min_h (Q_h(x_k, h(x_k))) \qquad h^*(x_k) = \arg\min_h (Q_h(x_k, h(x_k)))$$

Simple expression of Bellman's principle

$$V^*(x_k) = \min_{u_k} (Q^*(x_k, u_k)) \qquad h^*(x_k) = \arg\min_{u_k} (Q^*(x_k, u_k))$$

Q Learning does not need to know $f(x_k)$ or $g(x_k)$

For LQR $\quad V(x) = W^T \varphi(x) = x^T P x \quad$ V is quadratic in x

---

$$x_{k+1} = Ax_k + Bu_k$$

$$Q_h(x_k, u_k) = r(x_k, u_k) + V_h(x_{k+1})$$

$$= x_k^T Q x_k + u_k^T R u_k + (Ax_k + Bu_k)^T P (Ax_k + Bu_k)$$

$$= \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q + A^T P A & A^T P B \\ B^T P A & R + B^T P B \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \equiv \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T H \begin{bmatrix} x_k \\ u_k \end{bmatrix} = \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix}$$

Q is quadratic in x and u

Control update is found by $\quad 0 = \dfrac{\partial Q}{\partial u_k} = 2[B^T P A x_k + (R + B^T P B) u_k] = 2[H_{ux} x_k + H_{uu} u_k]$

so $\quad u_k = -(R + B^T P B)^{-1} B^T P A x_k = -H_{uu}^{-1} H_{ux} x_k = L_{j+1} x_k$

Control found only from Q function
A and B not needed

Q Learning– Action Dependent HDP – Paul Werbos

Q function for any given control policy *h(x_k)* satisfies the Bellman equation

$$Q_h(x_k, u_k) = r(x_k, u_k) + \gamma Q_h(x_{k+1}, h(x_{k+1}))$$

Policy Iteration Using Q Function- Recursive solution to HJB

Pick stabilizing initial control policy

Find Q function

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_j(x_{k+1}, h_j(x_{k+1}))$$

Update control

$$h_{j+1}(x_k) = \arg\min_{u_k}(Q_{j+1}(x_k, u_k))$$

Now $f(x_k,u_k)$ not needed

Bradtke & Barto (1994) proved convergence for LQR

Implementation- DT Q Function Policy Iteration

Bradtke and Barto

For LQR

Q function update for control $u_k = L_j x_k$ is given by

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_{j+1}(x_{k+1}, L_j x_{k+1})$$

Assume measurements of u_k, x_k and x_{k+1} are available to compute u_{k+1}

QFA – Q Fn. Approximation

$$Q(x,u) = W^T \varphi(x,u)$$

Now u is an input to the NN- Werbos- Action dependent NN

Then

regression matrix

$$W_{j+1}^T \left[\varphi(x_k, u_k) - \gamma \varphi(x_{k+1}, L_j x_{k+1})\right] = r(x_k, L_j x_k)$$

Solve for weights using RLS or backprop.

Since x_{k+1} is measured in training phase,
do not need knowledge of f(x) or g(x) for value fn. update

For LQR case

$$\varphi(x) = \left[x_1^2, \ldots, x_1 x_n, x_2^2, \ldots, x_2 x_n, \ldots, x_n^2\right]'.$$

Model-free policy iteration

Q Policy Iteration

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_{j+1}(x_{k+1}, L_j x_{k+1})$$

Bradtke, Ydstie, Barto

$$W_{j+1}^T \left[\varphi(x_k, u_k) - \gamma \varphi(x_{k+1}, L_j x_{k+1}) \right] = r(x_k, L_j x_k)$$

Control policy update

Stable initial control needed

$$h_{j+1}(x_k) = \arg\min_{u_k}(Q_{j+1}(x_k, u_k)) \qquad u_k = -H_{uu}^{-1} H_{ux} x_k = L_{j+1} x_k$$

Greedy Q Fn. Update - Approximate Dynamic Programming
ADP Method 3. Q Learning
Action-Dependent Heuristic Dynamic Programming (ADHDP)

Paul Werbos

Greedy Q Update

Model-free HDP

Stable initial control NOT needed

$$Q_{j+1}(x_k, u_k) = r(x_k, u_k) + \gamma Q_j(x_{k+1}, h_j(x_{k+1}))$$

$$W_{j+1}^T \varphi(x_k, u_k) = r(x_k, L_j x_k) + W_j^T \gamma \varphi(x_{k+1}, L_j x_{k+1}) \equiv \text{target}_{j+1}$$

Update weights by RLS or backprop.

Q learning actually solves the Riccati Equation
WITHOUT knowing the plant dynamics

Model-free ADP

Direct OPTIMAL ADAPTIVE CONTROL

Works for Nonlinear Systems

Proofs?
Robustness?
Comparison with adaptive control methods?

A Q-Learning Based Adaptive Optimal Controller Implementation for a Humanoid Robot Arm

Said Ghani Khan[1], Guido Herrmann[1], Frank L. Lewis[2], Tony Pipe[1], Chris Melhuish[1]

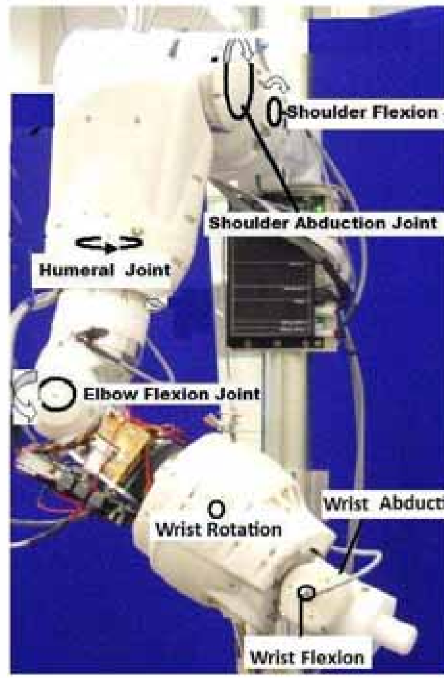
1: Bristol Robotics Laboratory, University of Bristol and University of the West of England, Bristol, UK

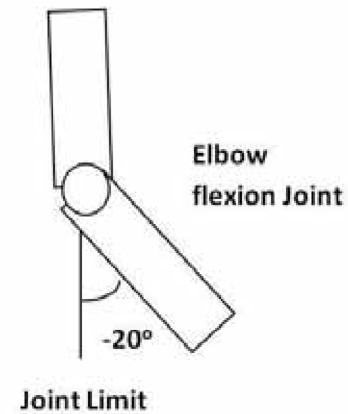
2: ARRI, Texas University at Arlington, USA

Conference on Decision and Control (CDC) 2011, Orlando

11 December 2011

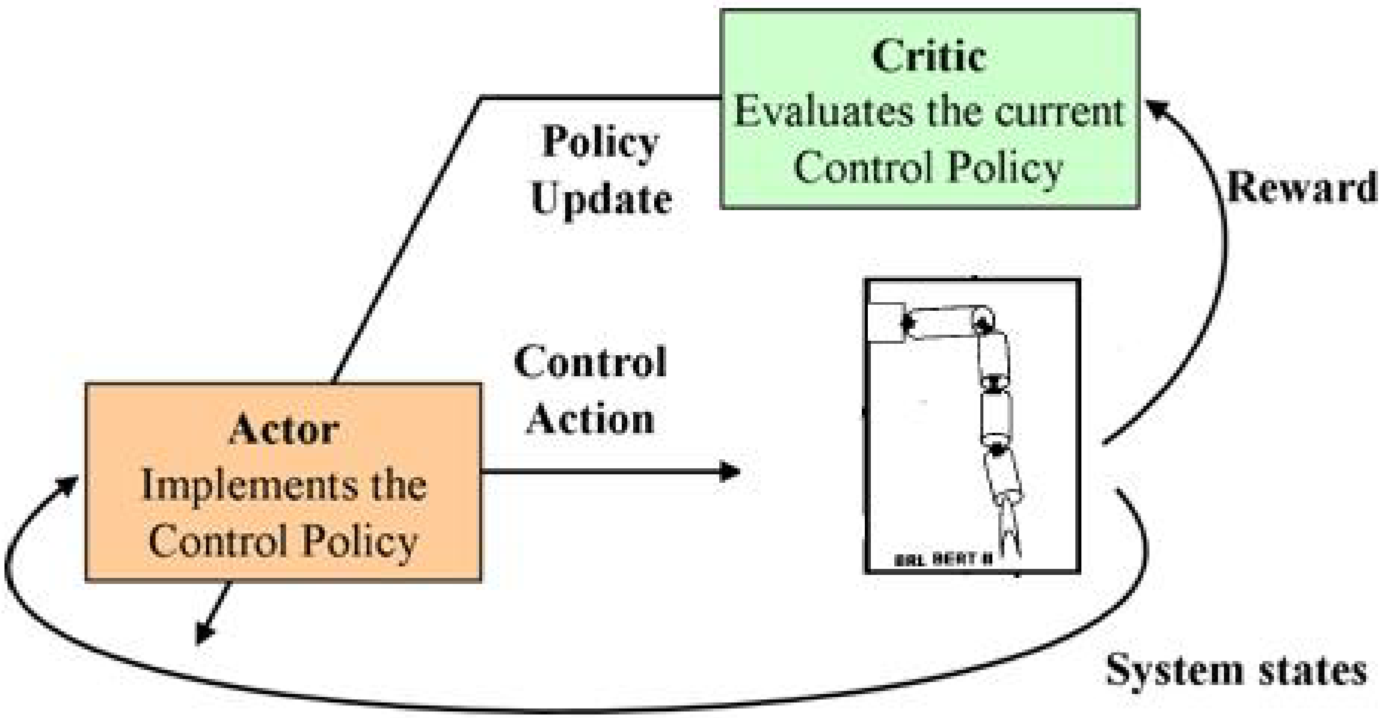
BRL BERT II ARM





The mechanical design and manufacturing for the BERT II torso including hand and arm has been conducted by Elumotion (www.elumotion.com), a Bristol Based company

ADP Actor-Critic Scheme

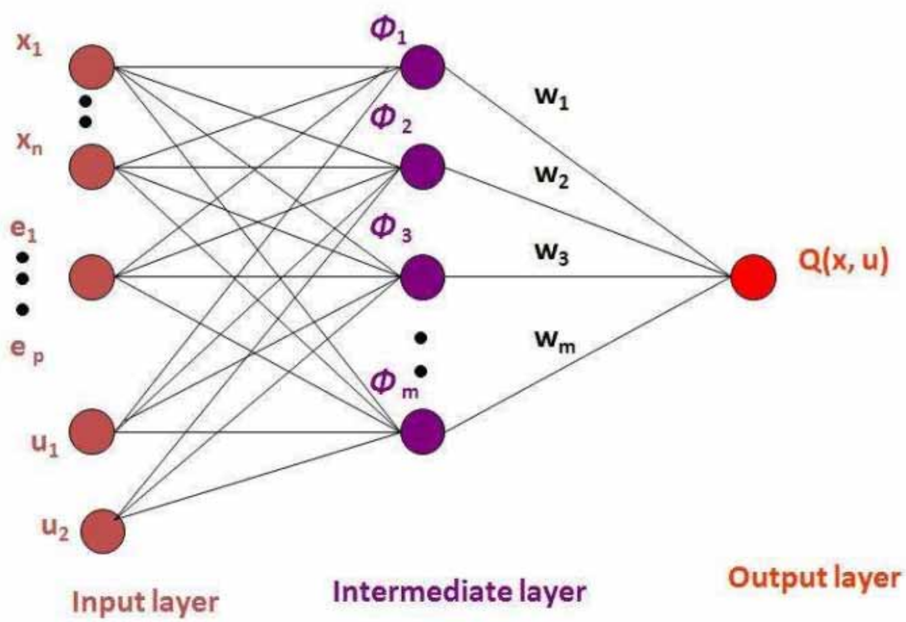


Stingue et al. 2010

Algorithm

The cost of control is modeled via an NN

$$\hat{Q}(x_k, u_k, d_k, w_i) = w_i^T \varphi(z_k(x_k, u_k, d_k))$$



Algorithm

The function $z_k(x_k, u_k, d_k)$ is a vector, linear in the control and control error and system states, e.g.

$$z_k(x_k, u_k, d_k) = \begin{bmatrix} u_{k_1} \\ \vdots \\ u_{k_m} \\ x_{k_1} - d_{k_1} \\ \vdots \\ x_{k_n} - d_{k_n} \\ x_{k_1} \\ \vdots \\ x_{k_n} \end{bmatrix}$$

Selected elements of the Kronecker product of z_k will be used as functions of a *polynomial* neural network $\varphi(\bullet)$

(greater detail later)

Note that the control signals u_k are at most quadratic in $\varphi(\bullet)$. *This again is a practical assumption.*

Introducing Constraints

The cost function is modified to include constraints

$$C(q) = \begin{cases} tan^2(\frac{q}{q_L} \times \frac{\pi}{2}), & if ||q|| < q_L \times \lambda \\ \\ tan^2(\frac{q}{q_L} \times \frac{\pi}{2}), & if ||q|| \geq q_L \times \lambda \end{cases}$$

$0<\lambda<1$

$q_L>0$ is the joint limit.

The new cost function....

$$r(x_k, u_k, d_k) \;=\; e_k^T Q_c e_k + (u_{k+1} - u_k)^T S(u_{k+1} - u_k) + (u_k)^T R(u_k) + \Lambda C(q)$$

where, Λ is a positive constant.

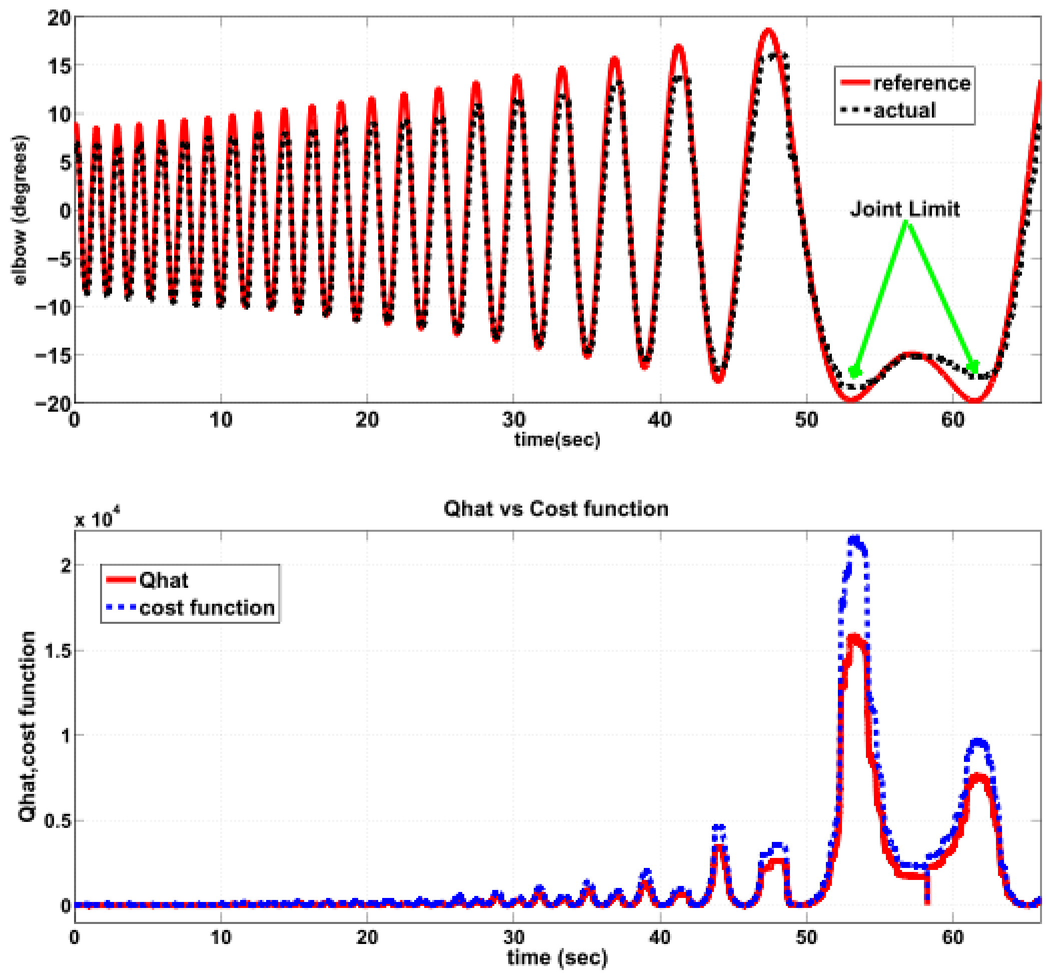
Introducing Constraints - Modelling Q

The NN nodes are obtained by the Kronecker product of :

$$z_k(x_k, e_k, u_k) \;=\; [u_i, e_1, e_2, e_1^2, e_2^2, x_1^2, x_2^2, x_1^3, x_2^3 \cdots + x_1^4, x_2^4, tan(\frac{x_1}{q_L} \times \frac{\pi}{2})]^T$$

Additional neurons are added to deal with the extra nonlinearity due to constraints

Constrained Case-Experiment

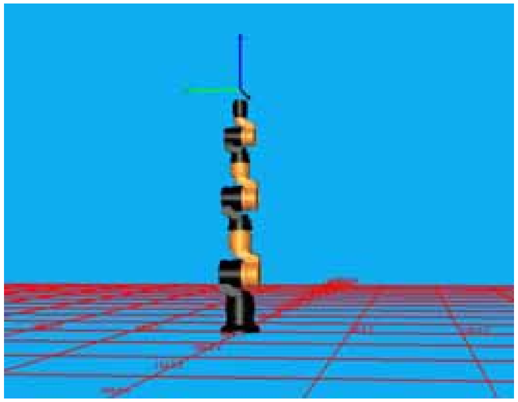


Reinforcement Learning Approach for Tele-Robotic Interaction Interface

Jartuwat Rajruangrabin and Dan Popa

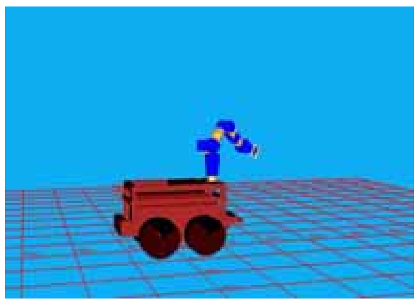
Automation & Robotics Research Institute
The University of Texas at Arlington

Experiment Platform



Simulated 7-degrees of freedom robotic manipulator used as a system to be controlled

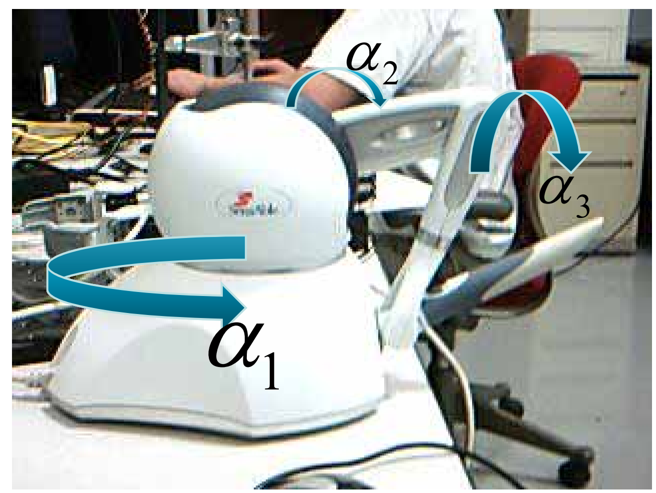
$$q^e(t) = [x(t) \quad y(t) \quad z(t) \quad \theta(t) \quad \varphi(t) \quad \psi(t)]^T$$



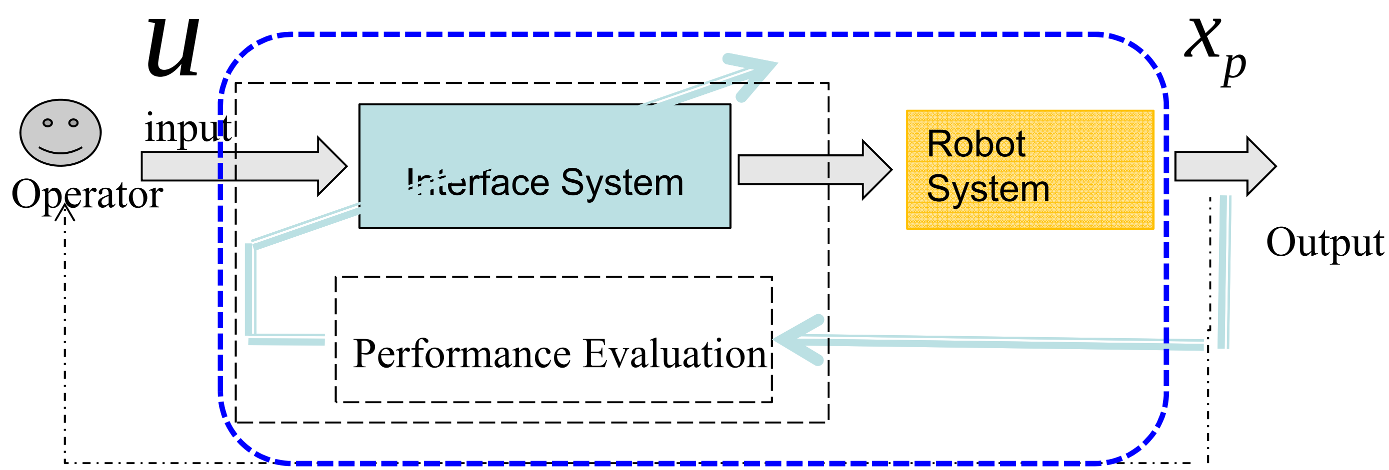
6-degrees of freedom robotic manipulator mounted on a differential drive mobile robot platform

Haptic Device used as Input Interface

$$u(t) = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$



Past Work / Challenges



Robot-Interface System

$$\dot{x}_p(t) = A_p x_p(t) + B_p u(t)$$

Challenges

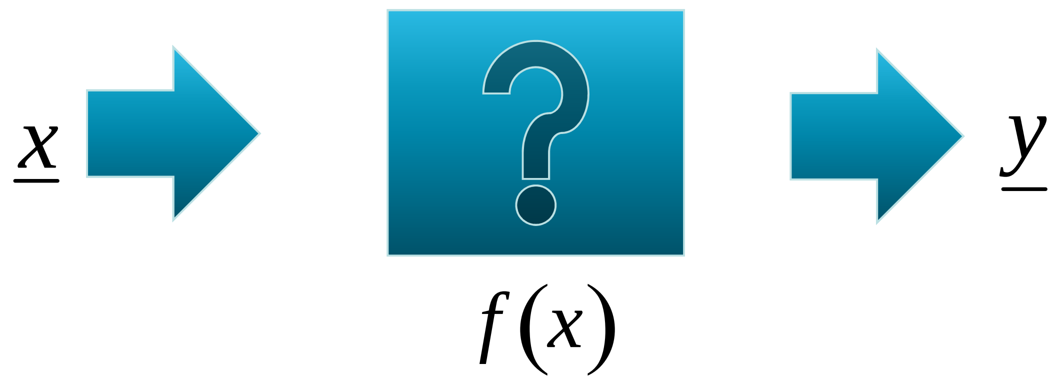
Set of input / output pairs have to be specified.

- Desired trajectory is known

What if we cannot specify the desired output trajectory directly?

- Use Reinforcement Learning

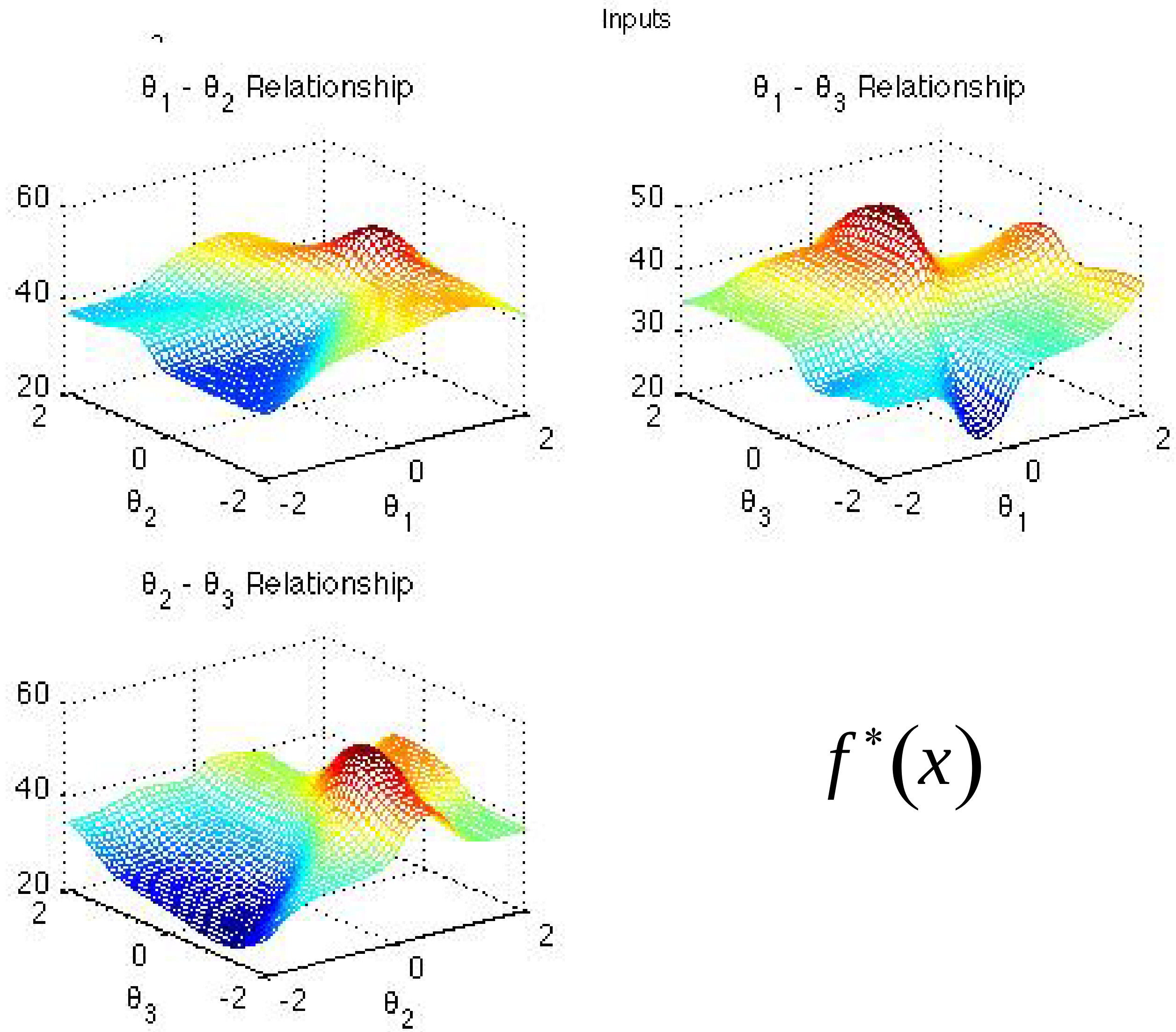
Interface Mapping



What can we do to get f(x)?

The simplest way is to obtain a set of inputs and a set of outputs and calculate the relationship (Curve Fitting)

Static Mapping Approach



$$f^*(x)$$

Reinforcement Learning

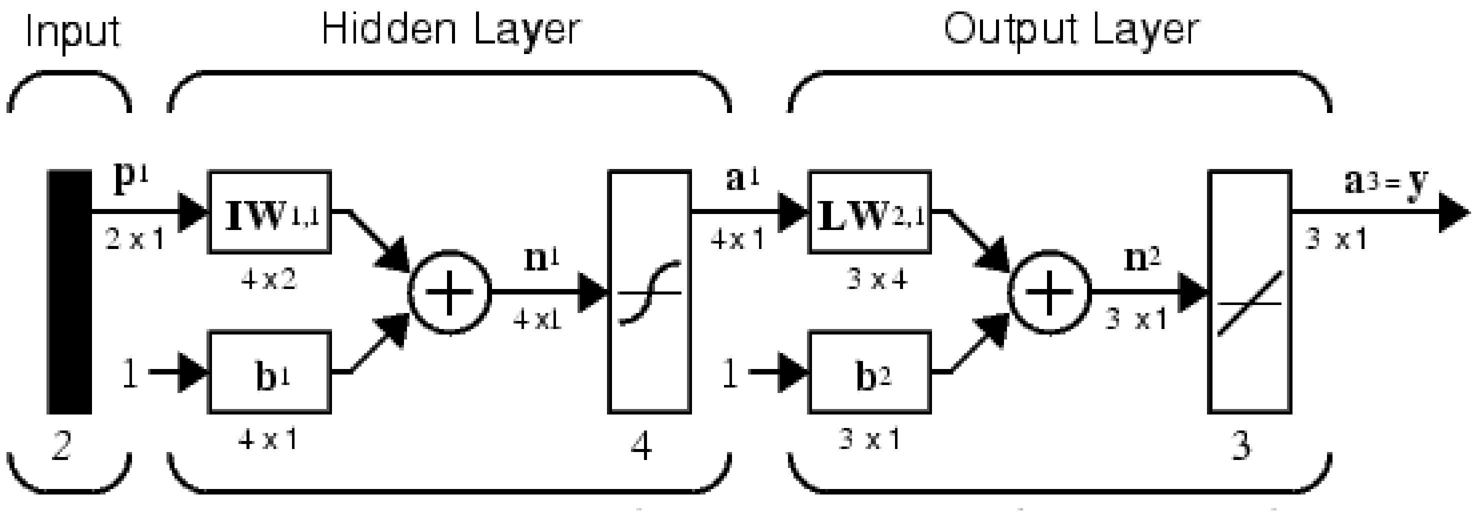
What if we cannot specify the desired output trajectory directly ?

Reinforcement Learning

With RL we do not have to specify a desire trajectory.

Instead, a reward function is used

NN Training



$$-\frac{\partial E}{\partial \vec{w}}$$

Log sigmoid function is used as a neuron activation function

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

RL Implementation Haptic/Robot Arm on a Mobile Robot

Objective:

-Exp1 : Implement RL with reward function that allow the user to control movement of the mobile platform

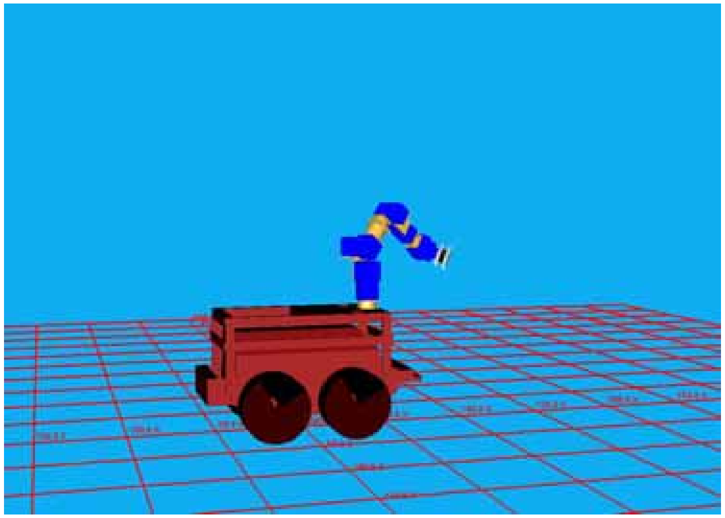
-Exp2 : Reverse the direction mapping of mobile platform based on Exp1

$$\underline{y}_{k+1} = f_{\vec{w}}\left(\underline{y}_k, \underline{x}\right)$$

Step 1(Initialization): Train the NN so that the weights are optimal according to the desired trajectory

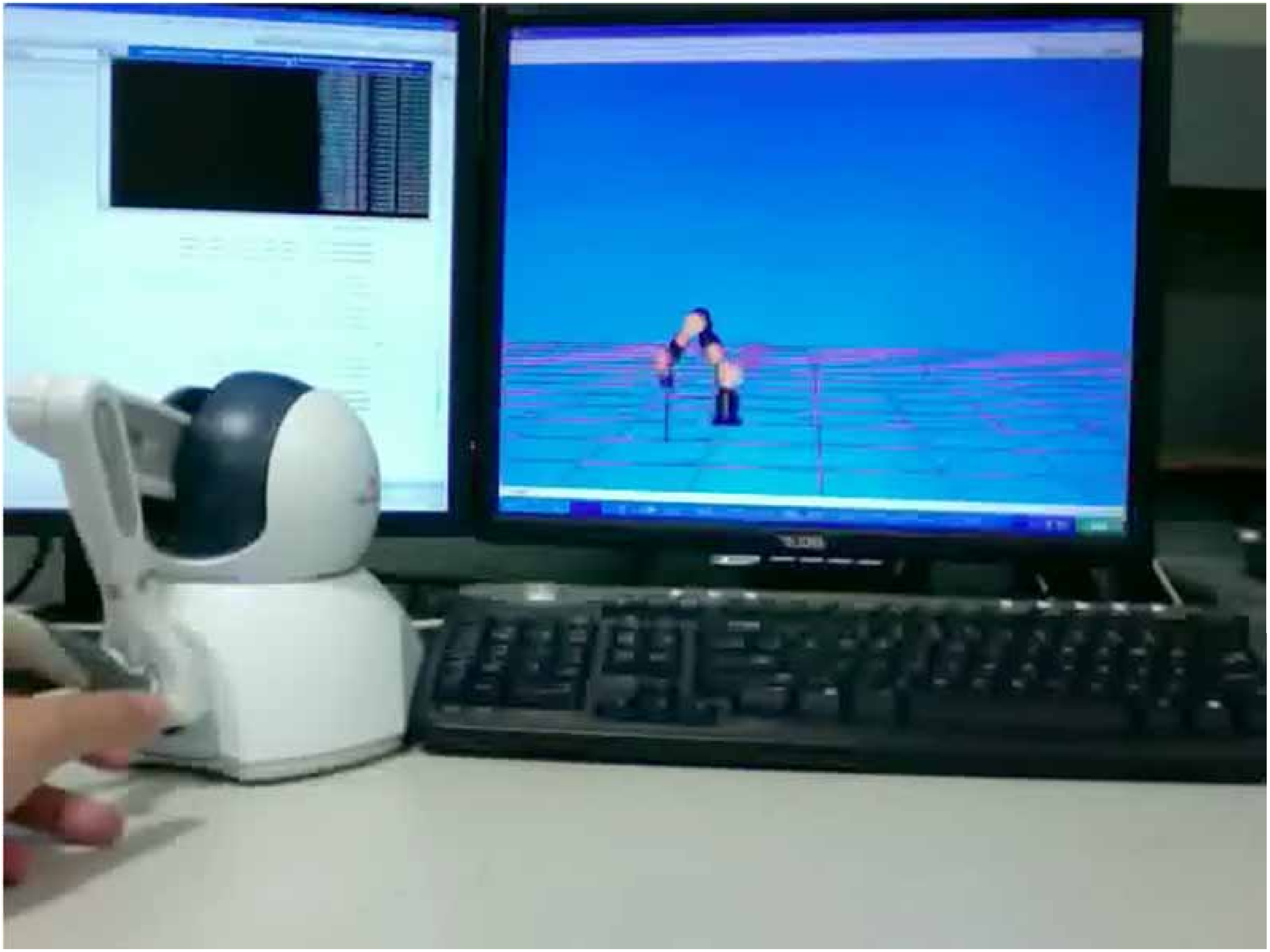
Step 2: (Online Learning) Implement the TD(λ) learning algorithm

$$r_{V_L} = \begin{cases} \mu\left(y_x - y_{x_{\max}} + \Delta x\right) & y_x > \left(y_{x_{\max}} - \Delta x\right) \\ -\mu\left(y_x - y_{x_{\min}} - \Delta x\right) & y_x < \left(y_{x_{\min}} + \Delta x\right) \\ 0, & otherwise \end{cases}$$

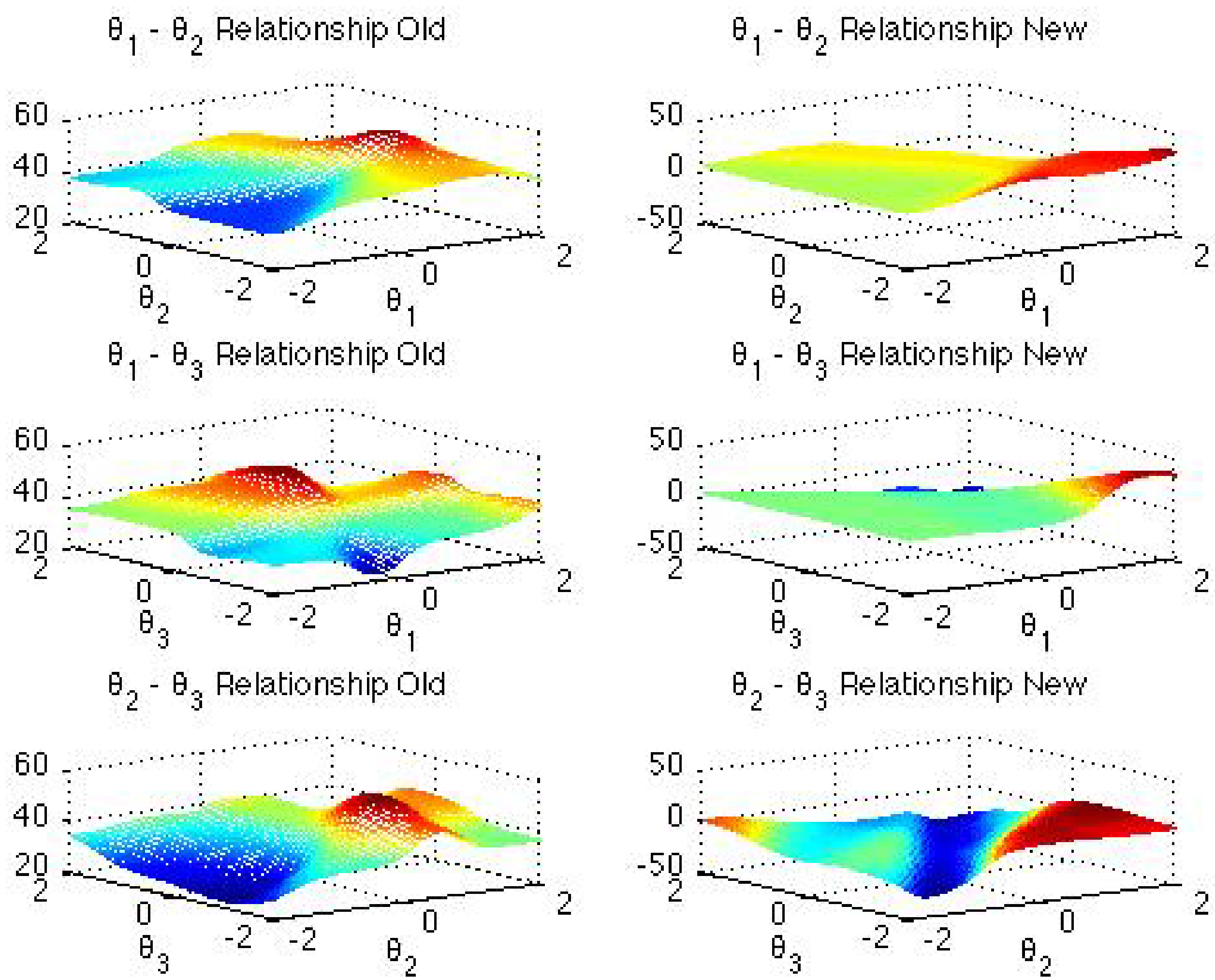


$$r_{V_R} = \begin{cases} \mu\left(y_x - y_{x_{\max}} + \Delta x\right) & y_x > \left(y_{x_{\max}} - \Delta x\right) \\ -\mu\left(y_x - y_{x_{\min}} - \Delta x\right) & y_x < \left(y_{x_{\min}} + \Delta x\right) \\ 0, & otherwise \end{cases}$$

Experiment Result – Online TD(λ) Learning



Experiment Result – Contour Shaping



Reverse and Scale Y - Mapping Update Through TD(λ) Learning Algorithm

NEW ROBOTIC TREATMENT SYSTEMS FOR CHILDHOOD CEREBRAL PALSY and AUTISM

D. O. Popa[A], I. Ranatunga[A], D. Hanson[B], F. Makedon[C]

[A]Department of Electrical Engineering, University of Texas at Arlington,USA
[B]Hanson Robotics Inc., Plano, TX, USA
[C]Department of Computer Science & Engineering Department,
University of Texas at Arlington, USA

This work was supported by:
US National Science Foundation Grants #*CPS 1035913*, and #*CNS 0923494.*
TxMed consortium grant: "Human-Robot Interaction System for Early Diagnosis and Treatment of Childhood Autism Spectrum Disorders (RoDiCA)"

Introduction

- Two assistive robotic systems aimed at the treatment of children with certain motor and cognitive impairments.

- In the Neptune project [1]
  - Mobile manipulator for children suffering from Cerebral-Palsy.
  - Mobile robot base and a 6DOF robotic arm, interfaced via:
    - Wii Remote, iPad, Neuroheadset, the Kinect, and Force sensing robotic skin
  - Therapeutic outcomes
    - Hand and head gesture recognition and reward.
    - Hand motion excercises using IPAD Games (CPlay, CPMaze, ProlloquoToGo) held by the robot.

- The RoDiCA project [2]
  - focuses on treating cognitive impairments in children suffering from ASD
  - Zeno is a robotic platform developed by Hanson Robotics, based on a patented realistic skin.
  - Therapeutic outcomes
    - Real time subject tracking/joint attention
    - Advanced head-eye and hand coordination
    - Facial gesture recognition and synthesis
    - Data logging and analysis.

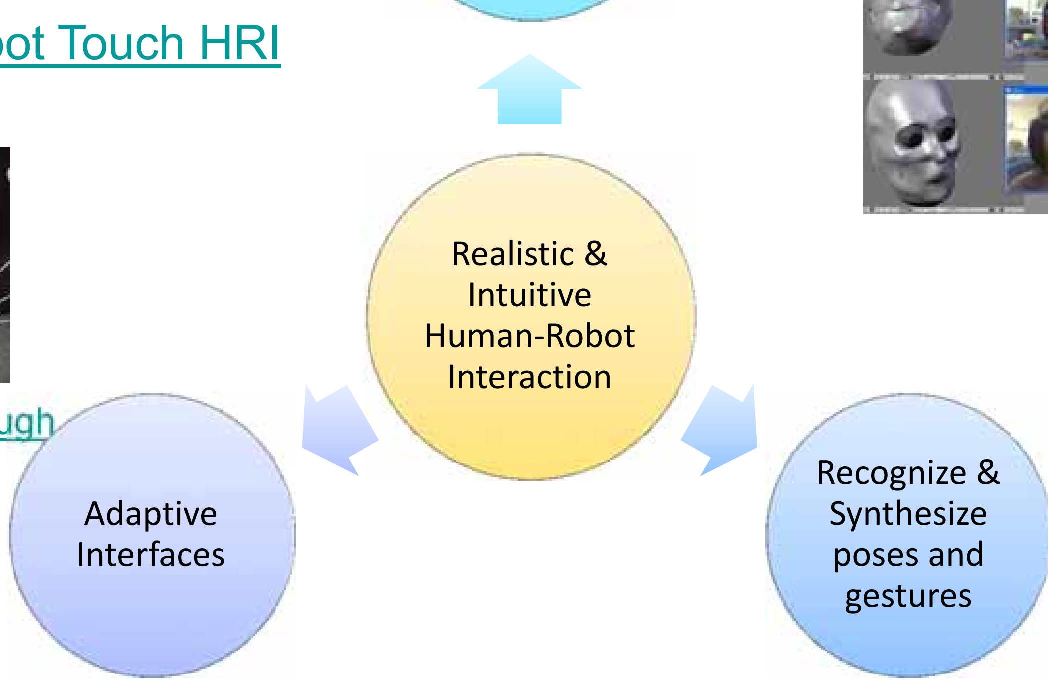
Neptune Mobile manipulator with iPad attached.

Zeno (by Hanson RoboKind Inc.) generating facial expressions and maintaining eye contact.

Advanced Control for Human Robot Interaction

Robot Touch HRI

Neptune Control through Neural Headband

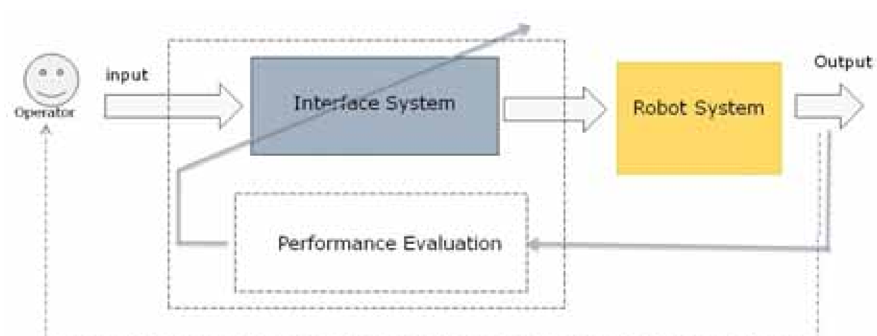


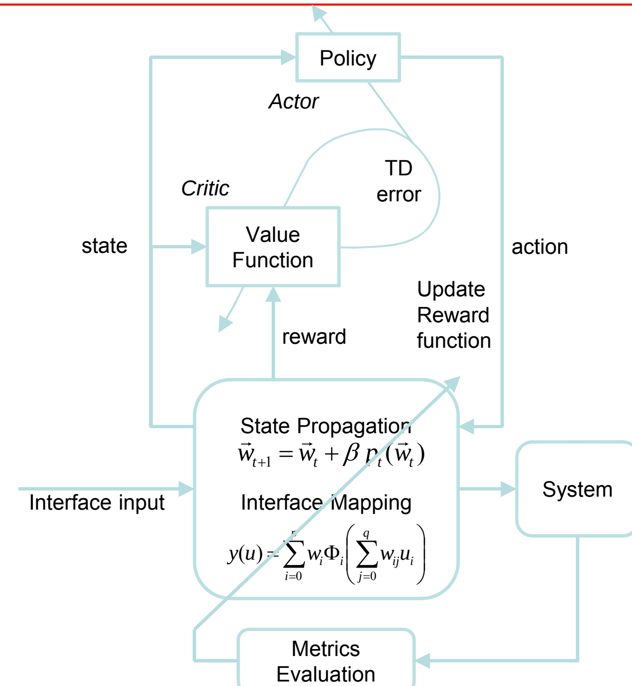
Visual HRI

Zeno Video

Adaptive Interfaces

- The supervisory control of multi-DOF robots is a demanding application.
- If a single operator is tasked with direct control, performing coordinated tasks becomes non-intuitive.
- We use Reinforcement Learning TD(lambda) scheme in order to adaptively change the mapping of DOF's from the operator user interface to the robot.





Our revels now are ended. These our actors,
As I foretold you, were all spirits, and
Are melted into air, into thin air.

The cloud-capped towers, the gorgeous palaces,
The solemn temples, the great globe itself,
Yea, all which it inherit, shall dissolve,
And, like this insubstantial pageant faded,
Leave not a rack behind.

We are such stuff as dreams are made on,
and our little life is rounded with a sleep.

Prospero, in The Tempest, act 4, sc. 1, l. 152-6, Shakespeare

Automation & Robotics Research Institute
University of Texas at Arlington

An Approximate Dynamic Programming Based Controller for an Underactuated 6DoF Quadrotor

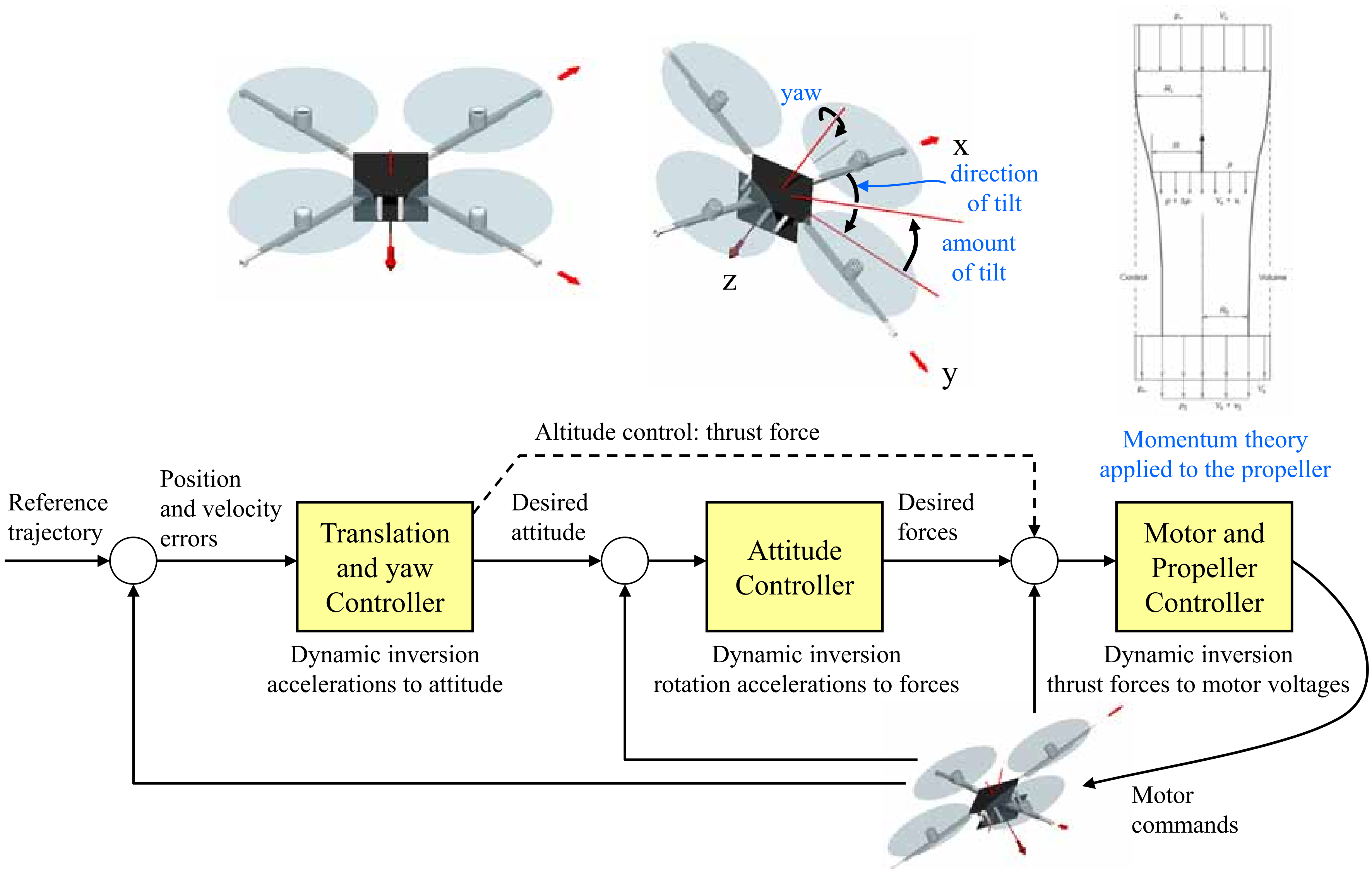
Emanuel Stingu

Frank Lewis

Supported by
ARO grant W91NF-05-1-0314
NSF grant ECCS-0801330

3 control loops

The quadrotor has 17 states and only 4 control inputs, thus it is very under-actuated.
Three control loops with dynamic inversion are used to generate the 4 control signals.



The actor is $u_k = h(x_{k-1}, z_{k-1})$ where $u = \begin{bmatrix} \mathbf{q}_{ad} & a_{z\ ad} & \mathbf{\Omega}_{ad} & \mathbf{V}_{mot\ ad} \end{bmatrix}^T$

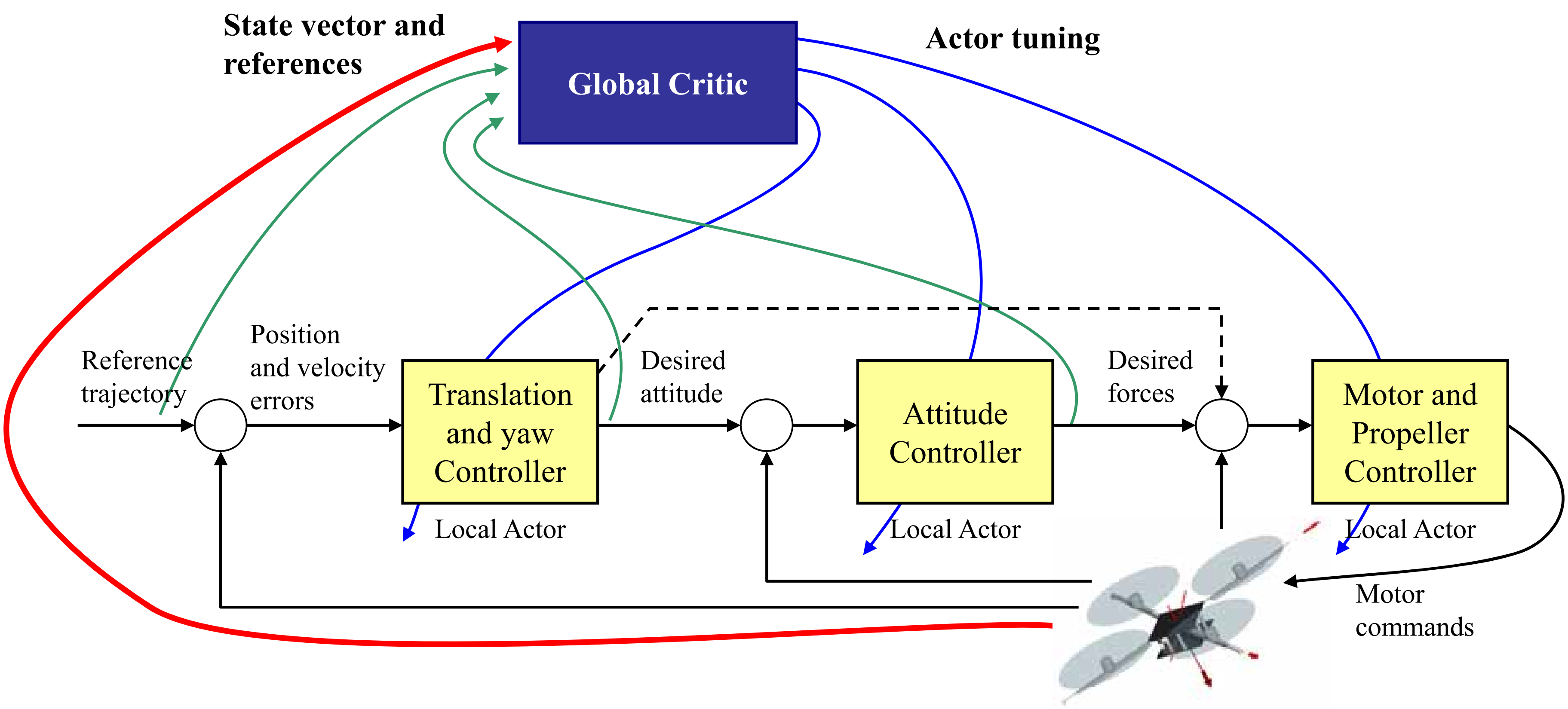
and

$$h(x_k, z_k) = \left[h^{1T}(x_k^1, z_k^1) \quad h^{2T}(x_k^2, z_k^2) \quad h^{3T}(x_k^3, z_k^3) \right]^T$$

Subsets of the state and tracking error vectors

The critic is

$$V_h(x_k, z_k) = \sum_{j=k}^{\infty} \gamma^{j-k} r\left(x_j, z_j, u_j\right)$$



Once the value of the Q function at $(x_{k-1}, z_{k-1}, u_{k-1})$ is known, a backup of it is made into the RBF neural network by adjusting the weights W and/or by adding more neurons and by reconfiguring their other parameters. This is a separate process that just needs to know the (x, z, u) coordinates and the new value to store.

$$Q(x_{k-1}, z_{k-1}, u_{k-1}) = W^T \phi(x_{k-1}, z_{k-1}, u_{k-1}) + \alpha \begin{bmatrix} r(x_{k-1}, z_{k-1}, h(x_{k-2}, z_{k-2})) + \\ +\gamma W^T \phi(x_k, z_k, u_k) - W^T \phi(x_{k-1}, z_{k-1}, u_{k-1}) \end{bmatrix}$$

The update of the Q value is not made completely towards the new value. This slows down the learning, but adds robustness.

$$Q_{stored} = Q_{old} + \alpha (Q_{new} - Q_{old}), \quad 0 < \alpha < 1$$

The policy update step is done by simply solving

after the new Q value was stored. The value for h is stored into the actor RBF neural network using the same mechanism as before:

$$\frac{\partial}{\partial u} Q(x_k, z_k, u) = 0$$

$$h(x_k, z_k) = U^T \sigma(x_k, z_k) + \beta \left[u - U^T \sigma(x_k, z_k) \right]$$

The Curse of Dimensionality

The actor acts as a nonlinear function approximator. Normally we have

$$u_{k+1} = h(x_k)$$

In the quadrotor case, because the reference is not zero and the system is nonlinear, we need

$$u_{k+1} = h(x_k, z_k)$$

For each of the position, attitude and motor/propeller loops the state vector includes the local states and the external states that have a big coupling effect on the loop performance.
It is easy to see that this way the input space can easily have n=14 or more dimensions.
A RBF neural network with the neurons placed on a grid with N elements in each dimension would require neurons. For N=5 and n=14, N^n are required. $6 \cdot 10^9$

Placing neurons on a grid is no better than a look-up table. The solutions to reducing the number of neurons are the following:

- preprocess the states to provide signals with physical significance as inputs
- combine multiple states into a lower dimension signal
- map multiple equivalent regions from the state-space into only one.