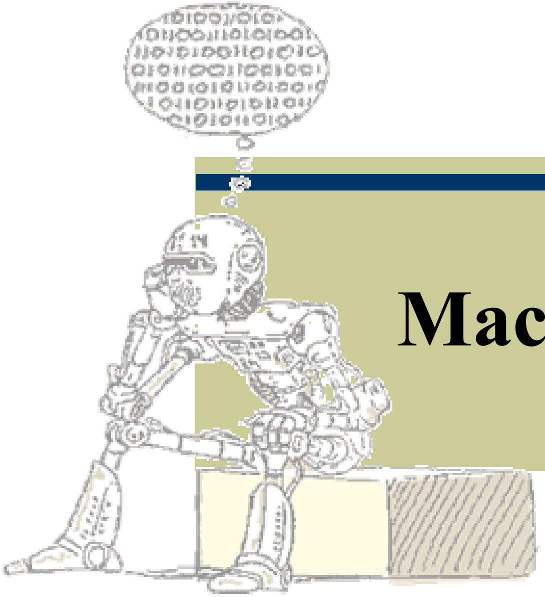


Machine Learning & Sensorimotor Control

Lecturer: Dr. Sethu Vijayakumar

[Course URL: http://homepages.inf.ed.ac.uk/svijayak/teaching/MLSC]

MLSC – Course Logistics

Scoring and Grades

30% of marks are allocated to the (two) homework assignments, 20% to the class presentation and 50% to the final exam.

Homework Assignments

They are individual assignments and have to be done individually !!

Class Presentation

You will be asked to review a topic and make a short presentation in class in addition to writing a short report. The topic will be allocated from a predefined list on motor control and the 'seed' paper will be provided to you. You are free to use your discretion in looking for additional materials to facilitate your understanding and presentation.

Class Participation

… is equally important as homeworks and exams in determining your final grade.

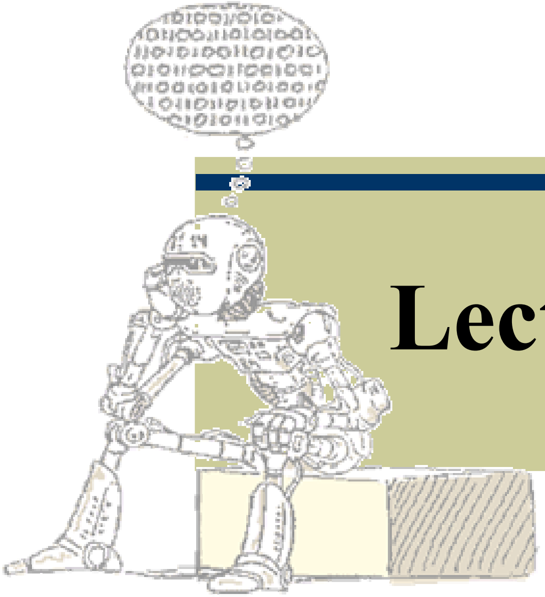
MLSC Course Logistics -II

Humanoid Robot : Judo Competition

One of the homeworks will be a programming task involving programming a controller for a humanoid robot (to play JUDO) in Java. Don't worry …not so difficult as it seems!!

The basic framework is available to you at: www.cyberbotics.com

1. Register a robot name to obtain an Internet directory where they will be able to upload their controller files
2. Download and install the free version of Webots containing all the contest material (robot models and tatami). This software runs on Windows, Linux and Mac OS X
3. Develop a Java control program driving the provided model of humanoid robot to perform a Judo match. Such a program should process sensor information (camera, distance sensors, touch sensors, etc.) and control the servo motors of the robot.



Lecture I – Introduction & Basics

Contents:

- The *What* & *Why* of Machine Learning
- Why do we need ML for control?

What is Machine Learning ?

- **Definition :**
  - Process by which a computer algorithm finds a solution to a problem

    Here …
    - ‘machine’ = computer
    - ‘learning’ = finding a good solution
- **Example:**
  - Learn to drive a car
  - Play chess / backgammon
  - Classify handwritten characters

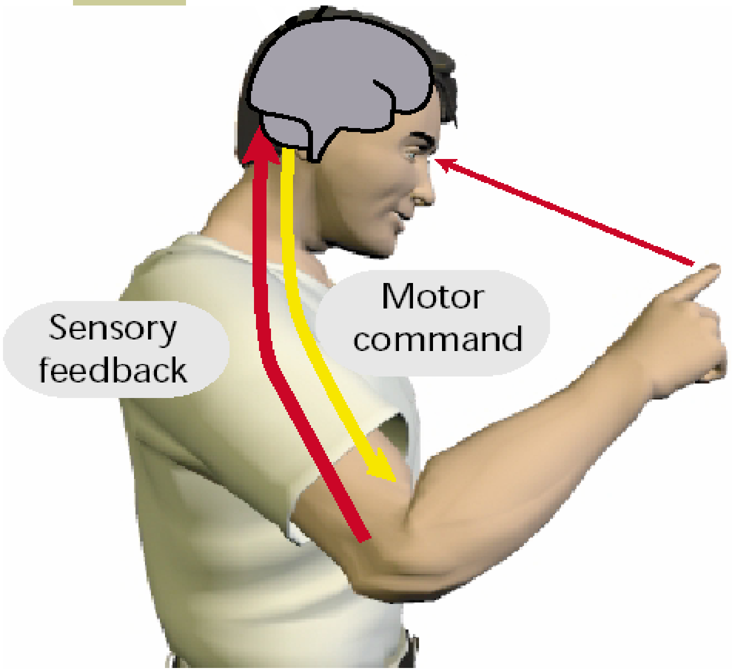
Uses of ‘learning’

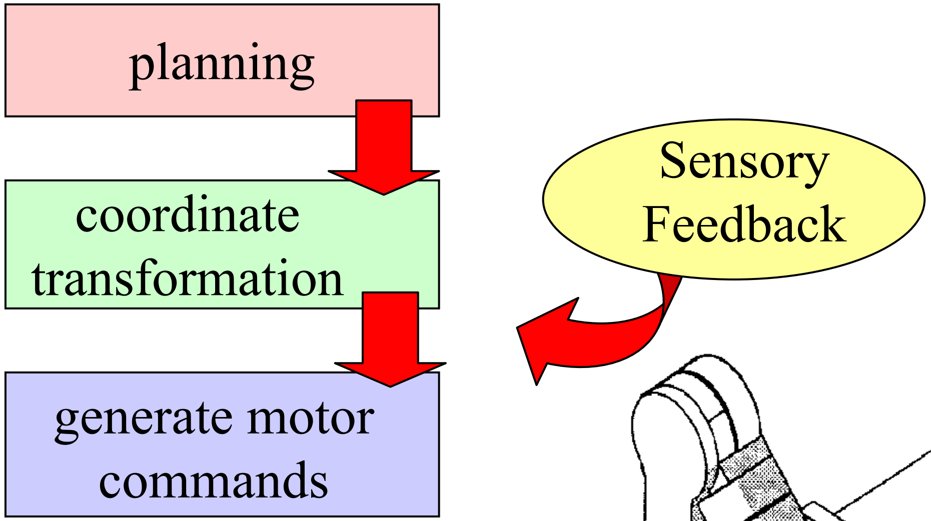
- **Knowledge extraction.**
  - Find a simpler model that explains data and hence, have an *explanation* about the data generation process.
- **Compression.**
  - Represent the generated data by the underlying principle and hence, use *less memory* to store and process.
- **Prediction.**
  - Predict future data based on past fits.

Classification of Machine Learning

- **Based on** *level of feedback*
  - Supervised – True output provided
  - Reinforcement – Only indirect output provided (reward/punishment)
  - Unsupervised – No feedback & no output
- **Based on** *type of output*
  - Concept Learning – Binary output based on +ve/-ve examples
  - Classification – Classifying into one among many classes
  - Regression – Numeric, ordered output
- **Based on** *sampling*
  - Independent (iid) samples
  - Dependent samples

Computational motor control

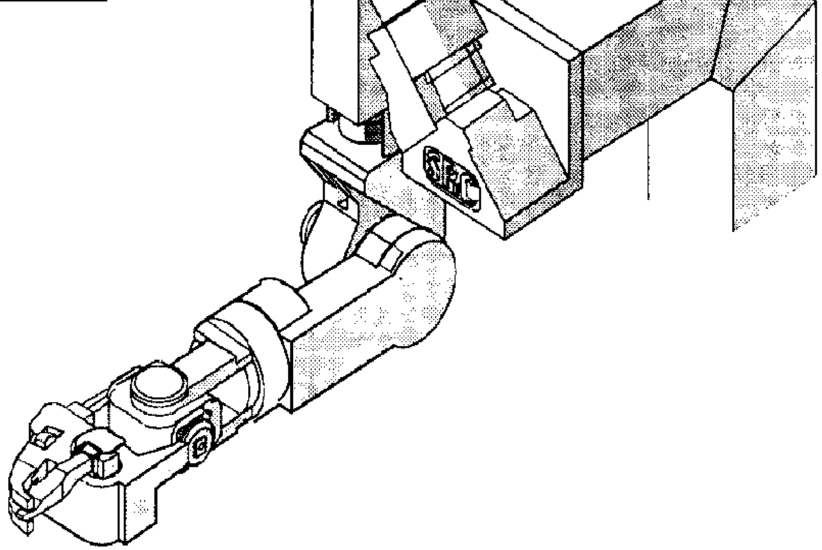



planning

coordinate transformation

generate motor commands

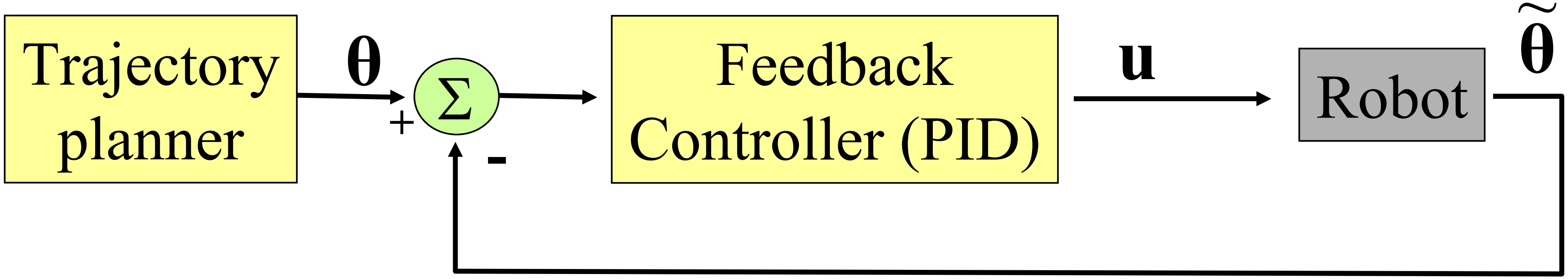
Sensory Feedback



Computational motor control deals with the study of relationship between sensory signals and motor commands

Biological motor control and Biomimetic control systems have very similar goals, constraints and concerns !!

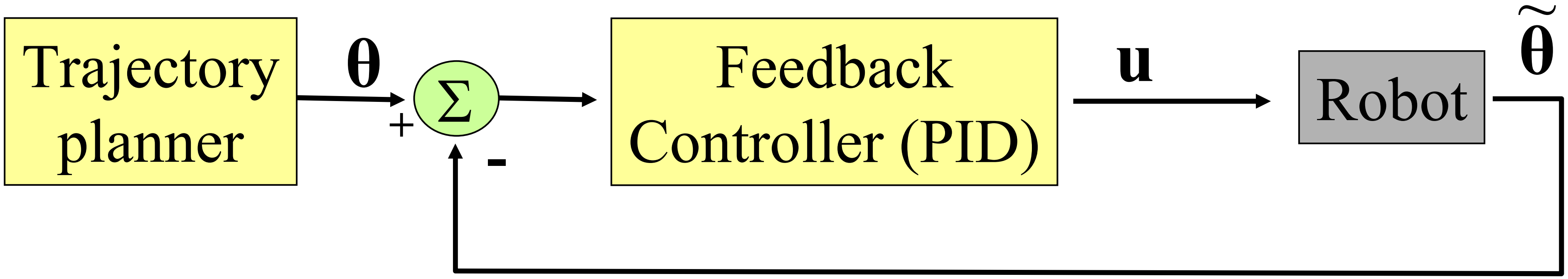
Control: basics



θ	Desired robot posture
$\tilde{\theta}$	Actual robot posture
u	Command (torque)

PID: Proportional Integral Control

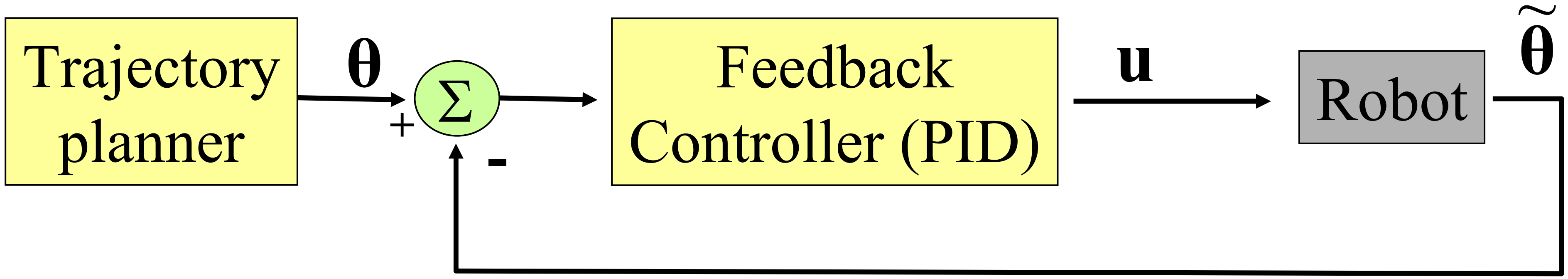
Control: Basics (PID Control)



Problem: we know that increasing *u* (the voltage to the motor) increases θ (the position), but it is difficult to predict by how much.

➔ Need for a PID control loop

Control: Basics (PID Control)



$$e = \tilde{\theta} - \theta \qquad u = K_p \cdot e + K_i \cdot \int e \cdot \partial t + K_d \cdot \frac{\partial e}{\partial t}$$

θ Desired robot posture

$\tilde{\theta}$ Actual robot posture

u Command (torque)

e Error

K_p = Proportional gain

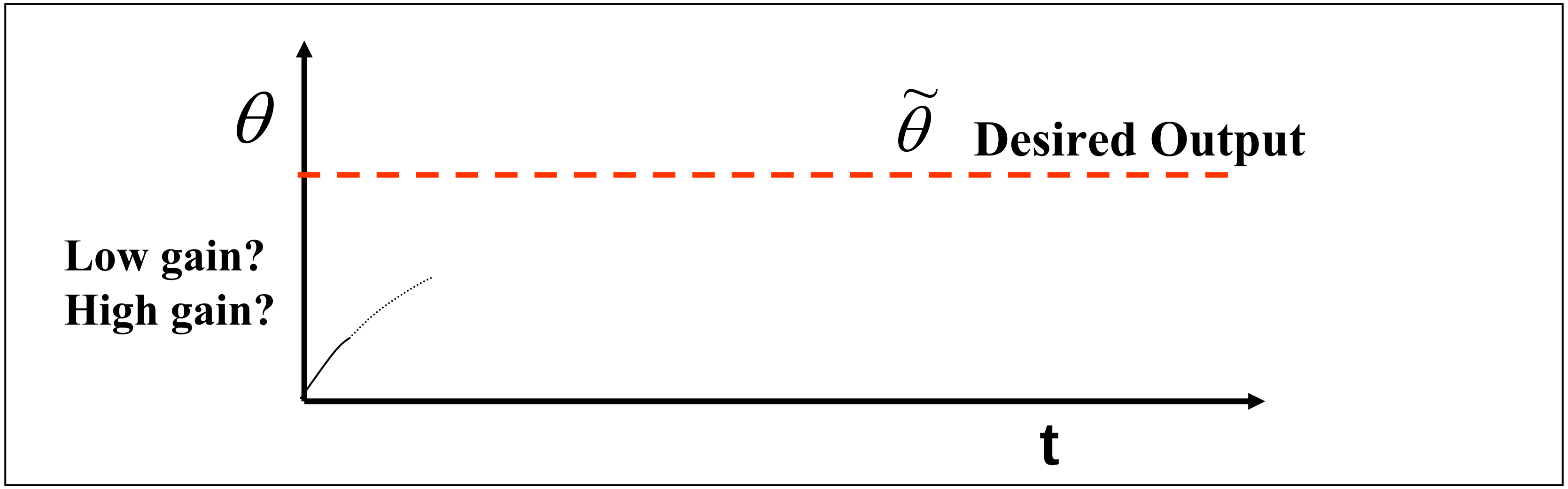
K_i = Integral gain

K_d = Derivative gain

Proportional Control

- **Set the torque (u) proportional to the position error**

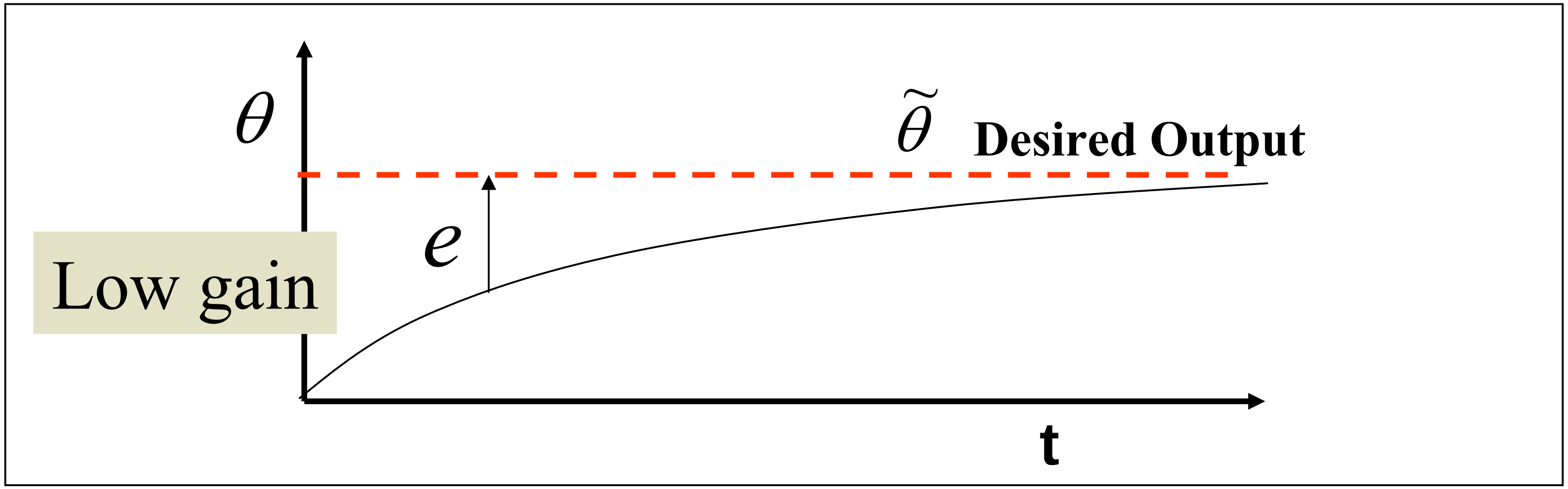
$$\boxed{u = K_p \cdot e}$$



Proportional Control: low gain

- **Set the torque (u) proportional to the position error**

$$\boxed{u = K_p \cdot e}$$

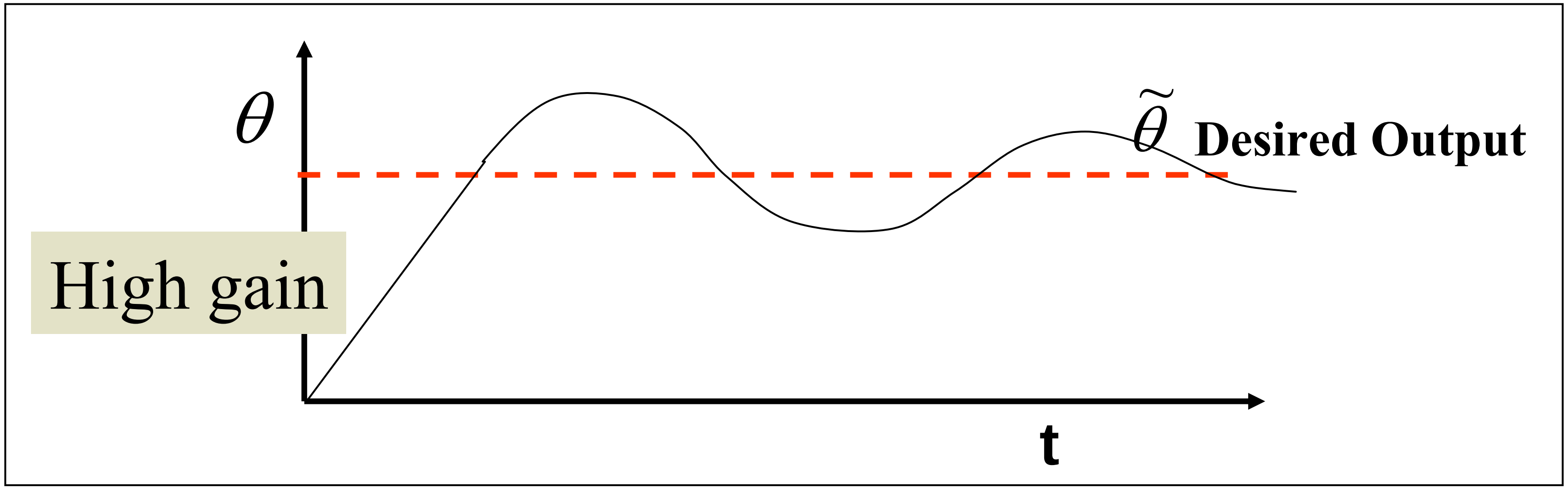


Problem: Long time to desired state

Proportional Control: high gain

- **Set the torque (u) proportional to the position error**

$$u = K_p \cdot e$$

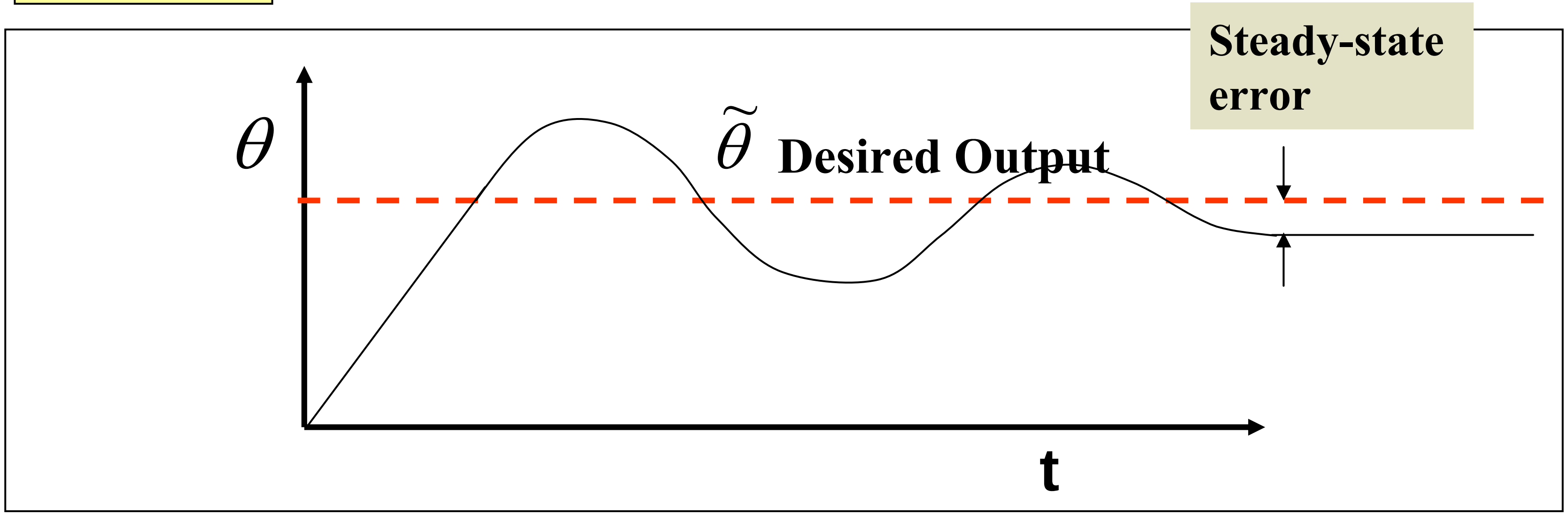


Problem: Overshoot

Steady state error

- **Set the torque (u) proportional to the position error**

$$\boxed{u = K_p \cdot e}$$

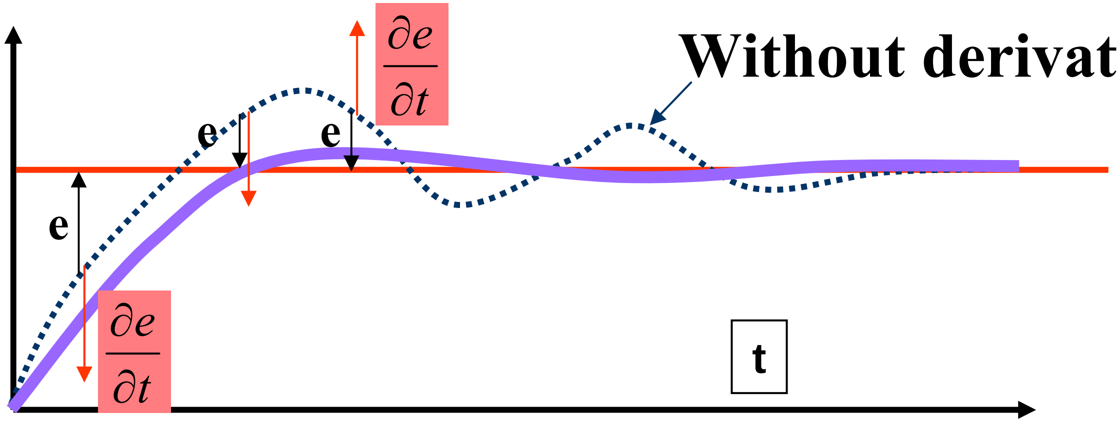


Problem: State does not converge to desired output

Solution: Integral or derivative control

PID control

$$u = K_p \cdot e + K_i \cdot \int e \cdot \partial t + K_d \cdot \frac{\partial e}{\partial t}$$



- With derivative action, the controller output is proportional to the rate of change of the measurement or error: braking effect.
- Adds **damping** → brakes the dynamics: reduces overshoot and tends to reduce the settling time
- Problem: if too large, will slow down the dynamics, and increase the rising and settling time

PID control summary

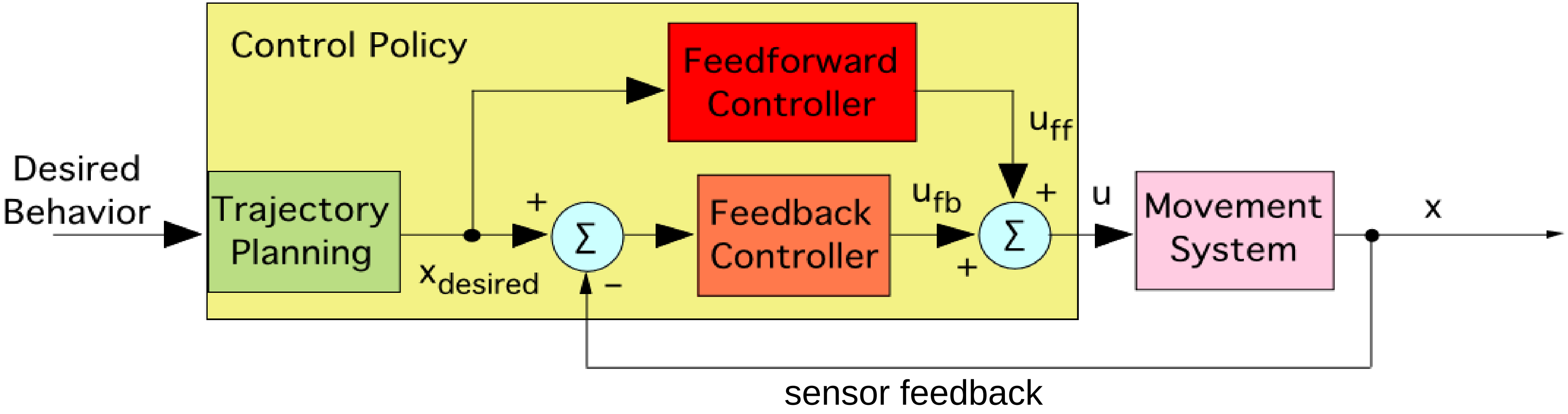
- A proportional controller (Kp) will have the effect of reducing the rise time and will reduce, but never eliminate, the steady-state error.

- An integral control (Ki) will have the effect of eliminating the steady-state error, but it may make the transient response worse.

- A derivative control (Kd) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response.

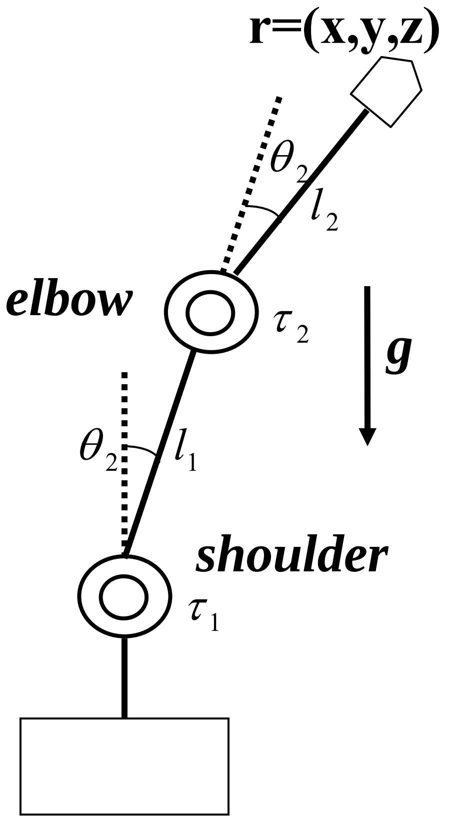
Gain	*Rise Time*	*Overshoot*	*Settling Time*	*S-S Error*
Kp	Decrease	**Increase**	**Small Change**	**Decrease**
Ki	**Decrease**	**Increase**	**Increase**	Eliminate
Kd	**Small increase**	Decrease	Decrease **(might increase)**	**Small Change**

Feedforward Controller



A feedforward model (also called 'internal models', 'inverse dynamics') is essential to make fast movements with high compliance—low stiffness (why?)

Inverse Dynamics: 2DOF arm



$$
\begin{aligned}
\tau_1 = {} & (I_1 + m_1 L_{G1}^2 + I_2 + m_2 L_{G2}^2 + m_2 L_1^2)\ddot{\theta}_1 \\
& + (I_2 + m_2 L_{G2}^2)\ddot{\theta}_2 \\
& + (2m_2 L_1 L_{G2}^2 \ddot{\theta}_2 + m_2 L_1 L_{G2}^2 \ddot{\theta}_1)\cos\theta_2 \\
& - (m_1 L_1 L_{G2} \ddot{\theta}_2^{\,2} + m_1 L_1 L_{G2} \dot{\theta}_1 \dot{\theta}_2)\sin\theta_2 \\
& - (m_1 L_{G1} + m_2 L_1) g \sin\theta_1 \\
& - m_2 L_{G1} g \sin\theta_1 \cos\theta_2 \\
& - m_2 L_{G2} g \sin\theta_2 \cos\theta_1
\end{aligned}
$$

$$
\begin{aligned}
\tau_2 = {} & (I_2 + m_2 L_{G2}^2)\ddot{\theta}_1 + (I_2 + m_2 L_{G2}^2)\ddot{\theta}_2 \\
& + m_2 L_1 L_{G2} \ddot{\theta}_1 \cos\theta_2 \\
& + m_2 L_1 L_{G2} \dot{\theta}_2 \sin\theta_2 \\
& - m_2 L_{G2} g \sin\theta_1 \cos\theta_2 \\
& - m_2 L_{G2} g \sin\theta_2 \cos\theta_1
\end{aligned}
$$

Inverse Dynamics derived based on ***equations of motion*** *and* ***rigid body dynamics*** *assumptions*

Can be used as a forward model and minor deviations corrected through feedback control

Feedforward model for a 7DOF robot

- **Inverse dynamics of a 7DOF anthropomorphic robot arm**

$$\tau = f(\theta, \dot{\theta}, \ddot{\theta})$$

$$f : \Re^{21} \rightarrow \Re^{7}$$

DEXTROUS ARM™
SRC
SARCOS
ENTITIES • ABOUT US • IN THE NEWS • COOL VIDEOS • EMPLOYMENT • CONTACT US • PROJECT INDEX

SARCOS dexterous arm

Analytical 7DOF Inverse Dynamics

```
v11 = 0;
v12 = 0;
v13 = th1d;
v14 = 0;
v15 = 0;
v16 = 0;

v21 = -(v13*Cos(th2));
v22 = v13*Sin(th2);
v23 = th2d;
v24 = 0;
v25 = 0;
v26 = 0;

v31 = -(v23*Cos(th3)) + v22*Sin(th3);
v32 = v22*Cos(th3) + v23*Sin(th3);
v33 = th3d + v21;
v34 = 0;
v35 = 0;
v36 = 0;

v41 = v33*Cos(th4) + v32*Sin(th4);
v42 = v32*Cos(th4) - v33*Sin(th4);
v43 = th4d - v31;
v44 = -(l1*v31*Sin(th4));
v45 = -(l1*v31*Cos(th4));
v46 = -(l1*v32);

v51 = -(v43*Cos(th5)) + v42*Sin(th5);
v52 = v42*Cos(th5) + v43*Sin(th5);
v53 = th5d + v41;
v54 = -(v46*Cos(th5)) + v45*Sin(th5);
v55 = v45*Cos(th5) + v46*Sin(th5);
v56 = v44;

v61 = v53*Cos(th6) + v52*Sin(th6);
v62 = v52*Cos(th6) - v53*Sin(th6);
v63 = th6d - v51;
v64 = v56*Cos(th6) - l2*v51*Sin(th6) +
v55*Sin(th6);
v65 = -(l2*v51*Cos(th6)) + v55*Cos(th6) -
v56*Sin(th6);
v66 = -(l2*v52) - v54;

v71 = -(v63*Cos(th7)) + v62*Sin(th7);
v72 = v62*Cos(th7) + v63*Sin(th7);
v73 = th7d + v61;
v74 = -(v66*Cos(th7)) + v65*Sin(th7);
v75 = v65*Cos(th7) + v66*Sin(th7);
v76 = v64;

a11 = 0;
a12 = 0;
a13 = th1dd;
a14 = G*Cos(th1)*Sin(a);
a15 = -(G*Sin(a)*Sin(th1));
a16 = G*Cos(a);

a21 = th2d*v22 - a13*Cos(th2);
a22 = -(th2d*v21) + a13*Sin(th2);
a23 = th2dd;
a24 = -(a16*Cos(th2)) + a15*Sin(th2);
a25 = a15*Cos(th2) + a16*Sin(th2);
a26 = a14;

a31 = th3d*v32 - a23*Cos(th3) + a22*Sin(th3);
a32 = -(th3d*v31) + a22*Cos(th3) + a23*Sin(th3);
a33 = a21 + th3dd;
a34 = -(a26*Cos(th3)) + a25*Sin(th3);
a35 = a25*Cos(th3) + a26*Sin(th3);
a36 = a24;
```

```
fnet61 = a64*m6 - m6*v63*v65 + m6*v62*v66 - m6*Power(v62,2)*xcm6 - m6*Power(v63,2)*xcm6 - a63*m6*ycm6 + a62*m6*zcm6 + v61*(m6*v62*ycm6 + m6*v63*zcm6);
fnet62 = a65*m6 + m6*v63*v64 - m6*v61*v66 + a63*m6*xcm6 - m6*Power(v61,2)*ycm6 - m6*Power(v63,2)*ycm6 - a61*m6*zcm6 + v62*(m6*v61*xcm6 + m6*v63*zcm6);
fnet63 = a66*m6 - m6*v62*v64 + m6*v61*v65 - a62*m6*xcm6 + a61*m6*ycm6 + v63*(m6*v61*xcm6 + m6*v62*ycm6) - m6*Power(v61,2)*zcm6 - m6*Power(v62,2)*zcm6;
fnet64 = a61*j611 + a62*j612 + a63*j613 + v65*(-(m6*v66) + m6*v62*xcm6) + v66*(m6*v65 + m6*v63*xcm6) + v62*(j623*v62 - j622*v63 - m6*v65*xcm6) + v63*(j633*v62 - j623*v63 - m6*v66*xcm6) + a66*m6*ycm6 - a65*m6*zcm6 + v64*(-
(m6*v62*ycm6) - m6*v63*zcm6) + v61*(j613*v62 - j612*v63 + m6*v65*ycm6 + m6*v66*zcm6);
fnet65 = a61*j612 + a62*j622 + a63*j623 - a66*m6*xcm6 + v64*(m6*v66 + m6*v61*ycm6) + v66*(-(m6*v64) + m6*v63*ycm6) + v61*(-(j613*v61) + j611*v63 - m6*v64*ycm6) + v63*(-(j633*v61) + j613*v63 - m6*v66*ycm6) + a64*m6*zcm6 + v65*(-
(m6*v61*xcm6) - m6*v63*zcm6) + v62*(-(j623*v61) + j612*v63 + m6*v64*xcm6 + m6*v66*zcm6);
fnet66 = a61*j613 + a62*j623 + a63*j633 + a65*m6*xcm6 - a64*m6*ycm6 + v66*(-(m6*v61*xcm6) - m6*v62*ycm6) + v63*(j623*v61 - j613*v62 + m6*v64*xcm6 + m6*v65*ycm6) + v64*(-(m6*v65) + m6*v61*zcm6) + v65*(m6*v64 + m6*v62*zcm6) +
v61*(j612*v61 - j611*v62 - m6*v64*zcm6) + v62*(j622*v61 - j612*v62 - m6*v65*zcm6);

fnet71 = a74*m7 - m7*v73*v75 + m7*v72*v76 - m7*Power(v72,2)*xcm7 - m7*Power(v73,2)*xcm7 - a73*m7*ycm7 + a72*m7*zcm7 + v71*(m7*v72*ycm7 + m7*v73*zcm7);
fnet72 = a75*m7 + m7*v73*v74 - m7*v71*v76 + a73*m7*xcm7 - m7*Power(v71,2)*ycm7 - m7*Power(v73,2)*ycm7 - a71*m7*zcm7 + v72*(m7*v71*xcm7 + m7*v73*zcm7);
fnet73 = a76*m7 - m7*v72*v74 + m7*v71*v75 - a72*m7*xcm7 + a71*m7*ycm7 + v73*(m7*v71*xcm7 + m7*v72*ycm7) - m7*Power(v71,2)*zcm7 - m7*Power(v72,2)*zcm7;
fnet74 = a71*j711 + a72*j712 + a73*j713 + v75*(-(m7*v76) + m7*v72*xcm7) + v76*(m7*v75 + m7*v73*xcm7) + v72*(j723*v72 - j722*v73 - m7*v75*xcm7) + v73*(j733*v72 - j723*v73 - m7*v76*xcm7) + a76*m7*ycm7 - a75*m7*zcm7 + v74*(-
(m7*v72*ycm7) - m7*v73*zcm7) + v71*(j713*v72 - j712*v73 + m7*v75*ycm7 + m7*v76*zcm7);
fnet75 = a71*j712 + a72*j722 + a73*j723 - a76*m7*xcm7 + v74*(m7*v76 + m7*v71*ycm7) + v76*(-(m7*v74) + m7*v73*ycm7) + v71*(-(j713*v71) + j711*v73 - m7*v74*ycm7) + v73*(-(j733*v71) + j713*v73 - m7*v76*ycm7) + a74*m7*zcm7 + v75*(-
(m7*v71*xcm7) - m7*v73*zcm7) + v72*(-(j723*v71) + j712*v73 + m7*v74*xcm7 + m7*v76*zcm7);
fnet76 = a71*j713 + a72*j723 + a73*j733 + a75*m7*xcm7 - a74*m7*ycm7 + v76*(-(m7*v71*xcm7) - m7*v72*ycm7) + v73*(j723*v71 - j713*v72 + m7*v74*xcm7 + m7*v75*ycm7) + v74*(-(m7*v75) + m7*v71*zcm7) + v75*(m7*v74 + m7*v72*zcm7) +
v71*(j712*v71 - j711*v72 - m7*v74*zcm7) + v72*(j722*v71 - j712*v72 - m7*v75*zcm7);

f71 = fnet71;
f72 = fnet72;
f73 = fnet73;
f74 = fnet74;
f75 = fnet75;
f76 = fnet76;

f61 = fnet61 + f73;
f62 = fnet62 + f72*Cos(th7) + f71*Sin(th7);
f63 = fnet63 - f71*Cos(th7) + f72*Sin(th7);
f64 = fnet64 + f76;
f65 = fnet65 + f75*Cos(th7) + f74*Sin(th7);
f66 = fnet66 - f74*Cos(th7) + f75*Sin(th7);

f51 = fnet51 - f63;
f52 = fnet52 + f62*Cos(th6) + f61*Sin(th6);
f53 = fnet53 + f61*Cos(th6) - f62*Sin(th6);
f54 = fnet54 - f66 - f62*l2*Cos(th6) - f61*l2*Sin(th6);
f55 = fnet55 - f63*l2 + f65*Cos(th6) + f64*Sin(th6);
f56 = fnet56 + f64*Cos(th6) - f65*Sin(th6);

f41 = fnet41 + f53;
f42 = fnet42 + f52*Cos(th5) + f51*Sin(th5);
f43 = fnet43 - f51*Cos(th5) + f52*Sin(th5);
f44 = fnet44 + f56;
f45 = fnet45 + f55*Cos(th5) + f54*Sin(th5);
f46 = fnet46 - f54*Cos(th5) + f55*Sin(th5);

f31 = fnet31 - f43;
f32 = fnet32 + f42*Cos(th4) + f41*Sin(th4);
f33 = fnet33 + f41*Cos(th4) - f42*Sin(th4);
f34 = fnet34 - f46 - f42*l1*Cos(th4) - f41*l1*Sin(th4);
f35 = fnet35 - f43*l1 + f45*Cos(th4) + f44*Sin(th4);
f36 = fnet36 + f44*Cos(th4) - f45*Sin(th4);
f21 = fnet21 + f33;
f22 = fnet22 + f32*Cos(th3) + f31*Sin(th3);
f23 = fnet23 - f31*Cos(th3) + f32*Sin(th3);
f24 = fnet24 + f36;
f25 = fnet25 + f35*Cos(th3) + f34*Sin(th3);
f26 = fnet26 - f34*Cos(th3) + f35*Sin(th3);
f11 = fnet11 + f23;
f12 = fnet12 + f22*Cos(th2) + f21*Sin(th2);
f13 = fnet13 - f21*Cos(th2) + f22*Sin(th2);
f14 = fnet14 + f26;
f15 = fnet15 + f25*Cos(th2) + f24*Sin(th2);
f16 = fnet16 - f24*Cos(th2) + f25*Sin(th2);

torque1 = f16;
torque2 = f26;
torque3 = f36;
torque4 = f46;
torque5 = f56;
torque6 = f66;
torque7 = f76;
```

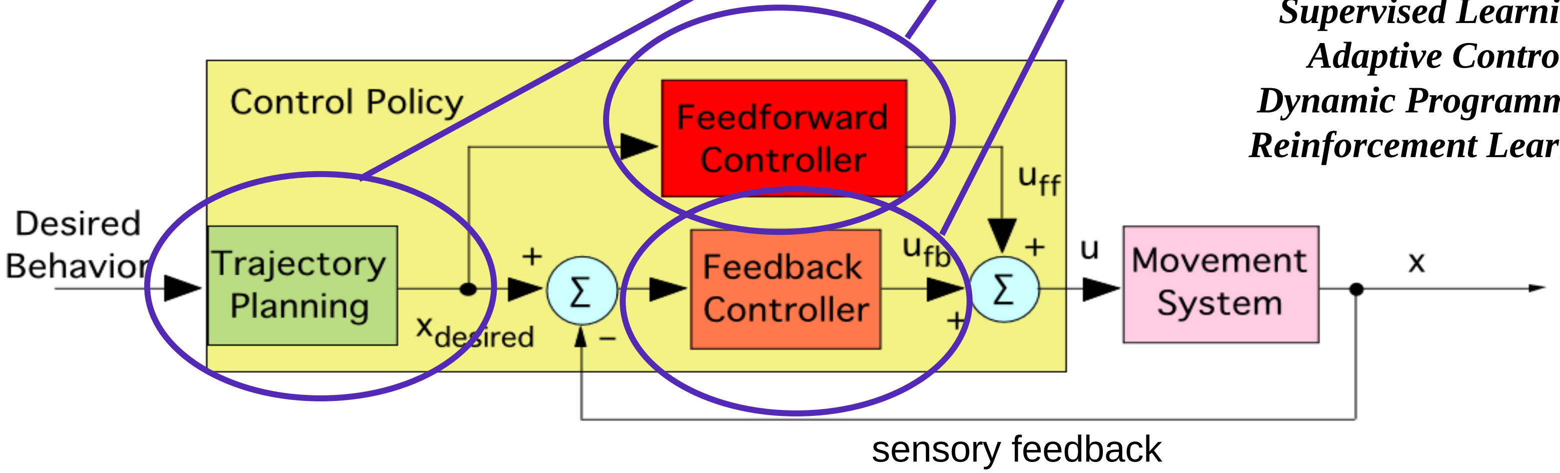
```
fnet21 = a24*m2 - m2*Power(v22,2)*xcm2 - m2*Power(v23,2)*xcm2 - a23*m2*ycm2 + a22*m2*zcm2 + v21*(m2*v22*ycm2 + m2*v23*zcm2);
fnet22 = a25*m2 + a23*m2*xcm2 - m2*Power(v21,2)*ycm2 - m2*Power(v23,2)*ycm2 - a21*m2*zcm2 + v22*(m2*v21*xcm2 + m2*v23*zcm2);
fnet23 = a26*m2 - a22*m2*xcm2 + a21*m2*ycm2 + v23*(m2*v21*xcm2 + m2*v22*ycm2) - m2*Power(v21,2)*zcm2 - m2*Power(v22,2)*zcm2;
fnet24 = a21*j211 + a22*j212 + a23*j213 + v21*(j213*v22 - j212*v23) + v22*(j223*v22 - j222*v23) + v23*(j233*v22 - j223*v23) + a26*m2*ycm2 - a25*m2*zcm2;
fnet25 = a21*j212 + a22*j222 + a23*j223 + v21*(-(j213*v21) + j211*v23) + v22*(-(j223*v21) + j212*v23) + v23*(-(j233*v21) + j213*v23) - a26*m2*xcm2 +
a24*m2*zcm2;
fnet26 = a21*j213 + a22*j223 + a23*j233 + v21*(j212*v21 - j211*v22) + v22*(j222*v21 - j212*v22) + (j223*v21 - j213*v22)*v23 + a25*m2*xcm2 - a24*m2*ycm2;

fnet31 = a34*m3 - m3*Power(v32,2)*xcm3 - m3*Power(v33,2)*xcm3 - a33*m3*ycm3 + a32*m3*zcm3 + v31*(m3*v32*ycm3 + m3*v33*zcm3);
fnet32 = a35*m3 + a33*m3*xcm3 - m3*Power(v31,2)*ycm3 - m3*Power(v33,2)*ycm3 - a31*m3*zcm3 + v32*(m3*v31*xcm3 + m3*v33*zcm3);
fnet33 = a36*m3 - a32*m3*xcm3 + a31*m3*ycm3 + v33*(m3*v31*xcm3 + m3*v32*ycm3) - m3*Power(v31,2)*zcm3 - m3*Power(v32,2)*zcm3;
fnet34 = a31*j311 + a32*j312 + a33*j313 + v31*(j313*v32 - j312*v33) + v32*(j323*v32 - j322*v33) + v33*(j333*v32 - j323*v33) + a36*m3*ycm3 - a35*m3*zcm3;
fnet35 = a31*j312 + a32*j322 + a33*j323 + v31*(-(j313*v31) + j311*v33) + v32*(-(j323*v31) + j312*v33) + v33*(-(j333*v31) + j313*v33) - a36*m3*xcm3 +
a34*m3*zcm3;
fnet36 = a31*j313 + a32*j323 + a33*j333 + v31*(j312*v31 - j311*v32) + v32*(j322*v31 - j312*v32) + (j323*v31 - j313*v32)*v33 + a35*m3*xcm3 - a34*m3*ycm3;

fnet41 = a44*m4 - m4*v43*v45 + m4*v42*v46 - m4*Power(v42,2)*xcm4 - m4*Power(v43,2)*xcm4 - a43*m4*ycm4 + a42*m4*zcm4 + v41*(m4*v42*ycm4 +
m4*v43*zcm4);
fnet42 = a45*m4 + m4*v43*v44 - m4*v41*v46 + a43*m4*xcm4 - m4*Power(v41,2)*ycm4 - m4*Power(v43,2)*ycm4 - a41*m4*zcm4 + v42*(m4*v41*xcm4 +
m4*v43*zcm4);
fnet43 = a46*m4 - m4*v42*v44 + m4*v41*v45 - a42*m4*xcm4 + a41*m4*ycm4 + v43*(m4*v41*xcm4 + m4*v42*ycm4) - m4*Power(v41,2)*zcm4 -
m4*Power(v42,2)*zcm4;
fnet44 = a41*j411 + a42*j412 + a43*j413 + v45*(-(m4*v46) + m4*v42*xcm4) + v46*(m4*v45 + m4*v43*xcm4) + v42*(j423*v42 - j422*v43 - m4*v45*xcm4) +
v43*(j433*v42 - j423*v43 - m4*v46*xcm4) + a46*m4*ycm4 - a45*m4*zcm4 + v44*(-(m4*v42*ycm4) - m4*v43*zcm4) + v41*(j413*v42 - j412*v43 + m4*v45*ycm4 +
m4*v46*zcm4);
fnet45 = a41*j412 + a42*j422 + a43*j423 - a46*m4*xcm4 + v44*(m4*v46 + m4*v41*ycm4) + v46*(-(m4*v44) + m4*v43*ycm4) + v41*(-(j413*v41) + j411*v43 -
m4*v44*ycm4) + v43*(-(j433*v41) + j413*v43 - m4*v46*ycm4) + a44*m4*zcm4 + v45*(-(m4*v41*xcm4) - m4*v43*zcm4) + v42*(-(j423*v41) + j412*v43 +
m4*v44*xcm4 + m4*v46*zcm4);
fnet46 = a41*j413 + a42*j423 + a43*j433 + a45*m4*xcm4 - a44*m4*ycm4 + v46*(-(m4*v41*xcm4) - m4*v42*ycm4) + v43*(j423*v41 - j413*v42 + m4*v44*xcm4 +
m4*v45*ycm4) + v44*(-(m4*v45) + m4*v41*zcm4) + v45*(m4*v44 + m4*v42*zcm4) + v41*(j412*v41 - j411*v42 - m4*v44*zcm4) + v42*(j422*v41 - j412*v42 -
m4*v45*zcm4);

fnet51 = a54*m5 - m5*v53*v55 + m5*v52*v56 - m5*Power(v52,2)*xcm5 - m5*Power(v53,2)*xcm5 - a53*m5*ycm5 + a52*m5*zcm5 + v51*(m5*v52*ycm5 +
m5*v53*zcm5);
fnet52 = a55*m5 + m5*v53*v54 - m5*v51*v56 + a53*m5*xcm5 - m5*Power(v51,2)*ycm5 - m5*Power(v53,2)*ycm5 - a51*m5*zcm5 + v52*(m5*v51*xcm5 +
m5*v53*zcm5);
fnet53 = a56*m5 - m5*v52*v54 + m5*v51*v55 - a52*m5*xcm5 + a51*m5*ycm5 + v53*(m5*v51*xcm5 + m5*v52*ycm5) - m5*Power(v51,2)*zcm5 -
m5*Power(v52,2)*zcm5;
fnet54 = a51*j511 + a52*j512 + a53*j513 + v55*(-(m5*v56) + m5*v52*xcm5) + v56*(m5*v55 + m5*v53*xcm5) + v52*(j523*v52 - j522*v53 - m5*v55*xcm5) +
v53*(j533*v52 - j523*v53 - m5*v56*xcm5) + a56*m5*ycm5 - a55*m5*zcm5 + v54*(-(m5*v52*ycm5) - m5*v53*zcm5) + v51*(j513*v52 - j512*v53 + m5*v55*ycm5 +
m5*v56*zcm5);
fnet55 = a51*j512 + a52*j522 + a53*j523 - a56*m5*xcm5 + v54*(m5*v56 + m5*v51*ycm5) + v56*(-(m5*v54) + m5*v53*ycm5) + v51*(-(j513*v51) + j511*v53 -
m5*v54*ycm5) + v53*(-(j533*v51) + j513*v53 - m5*v56*ycm5) + a54*m5*zcm5 + v55*(-(m5*v51*xcm5) - m5*v53*zcm5) + v52*(-(j523*v51) + j512*v53 +
m5*v54*xcm5 + m5*v56*zcm5);
fnet56 = a51*j513 + a52*j523 + a53*j533 + a55*m5*xcm5 - a54*m5*ycm5 + v56*(-(m5*v51*xcm5) - m5*v52*ycm5) + v53*(j523*v51 - j513*v52 + m5*v54*xcm5 +
m5*v55*ycm5) + v54*(-(m5*v55) + m5*v51*zcm5) + v55*(m5*v54 + m5*v52*zcm5) + v51*(j512*v51 - j511*v52 - m5*v54*zcm5) + v52*(j522*v51 - j512*v52 -
m5*v55*zcm5);
```

```
a41 = th4d*v42 + a33*Cos(th4) + a32*Sin(th4);
a42 = -(th4d*v41) + a32*Cos(th4) - a33*Sin(th4);
a43 = -a31 + th4dd;
a44 = th4d*v45 + a36*Cos(th4) + a35*Sin(th4) - a31*l1*Sin(th4);
a45 = -(th4d*v44) + a35*Cos(th4) - a31*l1*Cos(th4) - a36*Sin(th4);
a46 = -a34 - a32*l1;

a51 = th5d*v52 - a43*Cos(th5) + a42*Sin(th5);
a52 = -(th5d*v51) + a42*Cos(th5) + a43*Sin(th5);
a53 = a41 + th5dd;
a54 = th5d*v55 - a46*Cos(th5) + a45*Sin(th5);
a55 = -(th5d*v54) + a45*Cos(th5) + a46*Sin(th5);
a56 = a44;

a61 = th6d*v62 + a53*Cos(th6) + a52*Sin(th6);
a62 = -(th6d*v61) + a52*Cos(th6) - a53*Sin(th6);
a63 = -a51 + th6dd;
a64 = th6d*v65 + a56*Cos(th6) + a55*Sin(th6) - a51*l2*Sin(th6);
a65 = -(th6d*v64) + a55*Cos(th6) - a51*l2*Cos(th6) - a56*Sin(th6);
a66 = -a54 - a52*l2;

a71 = th7d*v72 - a63*Cos(th7) + a62*Sin(th7);
a72 = -(th7d*v71) + a62*Cos(th7) + a63*Sin(th7);
a73 = a61 + th7dd;
a74 = th7d*v75 - a66*Cos(th7) + a65*Sin(th7);
a75 = -(th7d*v74) + a65*Cos(th7) + a66*Sin(th7);
a76 = a64;

fnet11 = a14*m1 - m1*Power(v13,2)*xcm1 - a13*m1*ycm1;
fnet12 = a15*m1 + a13*m1*xcm1 - m1*Power(v13,2)*ycm1;
fnet13 = a16*m1;
fnet14 = a13*j113 - j123*Power(v13,2) + a16*m1*ycm1 - a15*m1*zcm1;
fnet15 = a13*j123 + j113*Power(v13,2) - a16*m1*xcm1 + a14*m1*zcm1;
fnet16 = a13*j133 + a15*m1*xcm1 - a14*m1*ycm1;
```

Function Approximation

$$\tau = f(\theta, \dot{\theta}, \ddot{\theta})$$

Learning Internal Models or Control Policies is essentially performing function approximation



Supervised Learning
Adaptive Control
Dynamic Programming
Reinforcement Learning

Biological vs. Biomimetic motor control

- **Biological motor control:**
  - Has to deal with huge sensorimotor delays ~*200ms*
  - And yet be able to react *fast* with relatively *low gains*
  - *Adapt* to changed dynamics and variable loads

…case for *adaptive* ***internal models***

- **Biomimetic systems:**
  - Have to operate with relatively *low feedback gains in order to be compliant* [not stiff, but give-in]
  - Are highly non-linear systems that are *hard to model analytically*
  - Yet has to react *fast* !!
  - *Adapt* to wear and tear, friction/viscous forces and changing loads

…case for *learning* ***feedforward control***

Next Lecture: Linear Algebra revisited

- **Basics of Linear Algebra**
  - Read material prior to class since we will cover this material very fast