Cognitive Robotics

Motion (part 2)

Hans-Dieter Burkhard, Marija Brkic Bakaric
October 2014

Outline

Introduction

Kinematics of Poses

Kinematics of Drive Systems

Trajectories

Motion Planning

Motion Control

Motions of Legged Robots

Optimization/Learning of Motions

Biologically Inspired Motions

Motion Control

Commands for the actuators (motors) to reach next pose(s) determined e.g. by

- a given (predefined) trajectory
- maintaining special conditions (e.g. PID controller)
- reply to sensor input (e.g. sensor actor coupling)

regarding e.g.:

- Positions, Forces, Speed
- Real time requirements
- Compensation for
  - Environmental disturbance (short term)
  - Battery, temperature (middle term)
  - Wear (long term)

Motion control

Feedforward control/open loop control:

- Fixed predefined control
- Simple realization
- No adaptation

„blind“

Keyframe motions?

Feedback control/closed loop control:

- Sensor controlled motions
- Adaptation using sensor signals

Closed Loop Controller

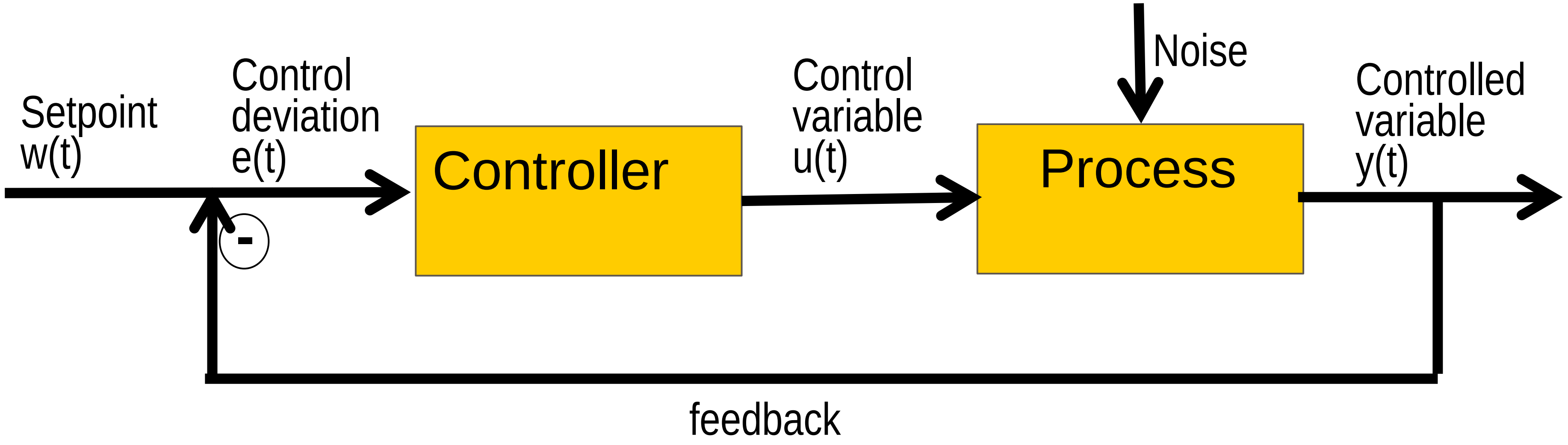
Controls a process such that specified objectives are achieved or maintained.

Setpoint:

The desired value of the process to be reached or maintained (e.g. bring the arm to a position or hold it on a position)

Control loop

The controlled variable y(t) should be equal to setpoint w(t).
The error e(t) := w(t) – y(t) is determined by feedback.
The controller determines the control variable u(t) related to e(t).



Control loop

Description without noise: $y(t+1) = f_{Process}(f_{Control}(w(t)-y(t)))$

Objectives: $e(T) = w(T) - y(T) = 0$
at a certain time T (or for all t >=T)

- Design of individual control from formal description.
- Usage of generic methods (fuzzy control, PID control).

Control loop

Can lead to overshooting and oscillations

Problems:

- Delayed control.
- Noise of process, sensors, and controls.
- Inertia of process.

Proportional Control (P-Control)

control $u(t) \sim$ deviation $e(t) := w(t)-y(t)$

$$u(t) = K \cdot e(t) \text{ with some constant } K$$

Small K: slow movement to setpoint w(t)

Large K: overshooting, oscillations

Integral Control (I-Control)

control u(t) ~ duration and amount of deviation e(t) := w(t)-y(t)

$$u(t) = K \cdot \sum_{i=1}^{l} e(t_i)\Delta t_i \quad \text{with some constant } K$$

Can compensate for low proportional control,

but continues changing for some time

Derivative Control (D-Control)

control $u(t)$ ~ change of deviation $e(t) := w(t)-y(t)$

$u(t) = K \cdot 1/Dt \cdot [\, e(t) - e(t-1) \,]$ with some constant K

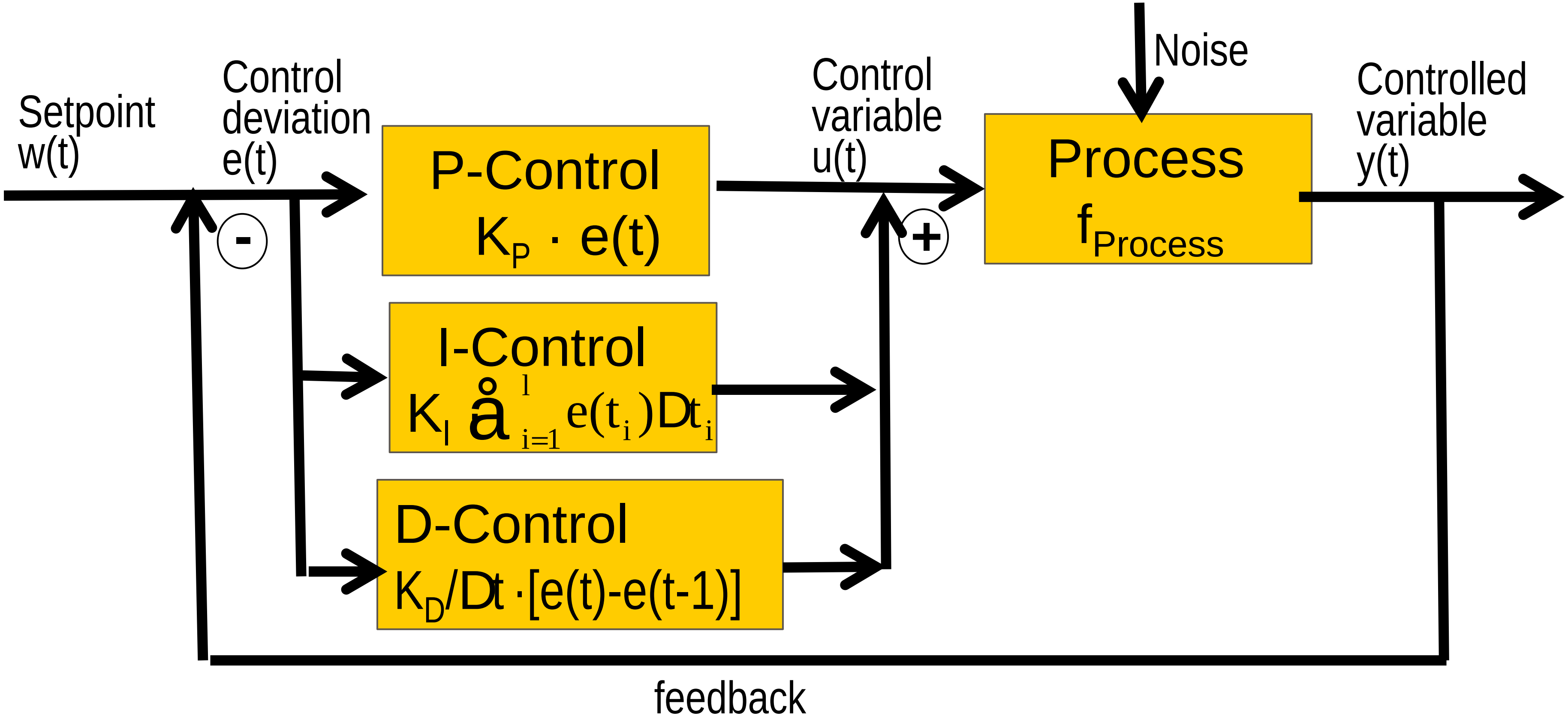
Fast respond to a "jump" of deviation.
No respond to permanently constant error.
Problem for noisy measurements.
Can only be used in combination with other controls .

Combination: PID-Controller

Similarly:
PI-Controller
PD-Controller

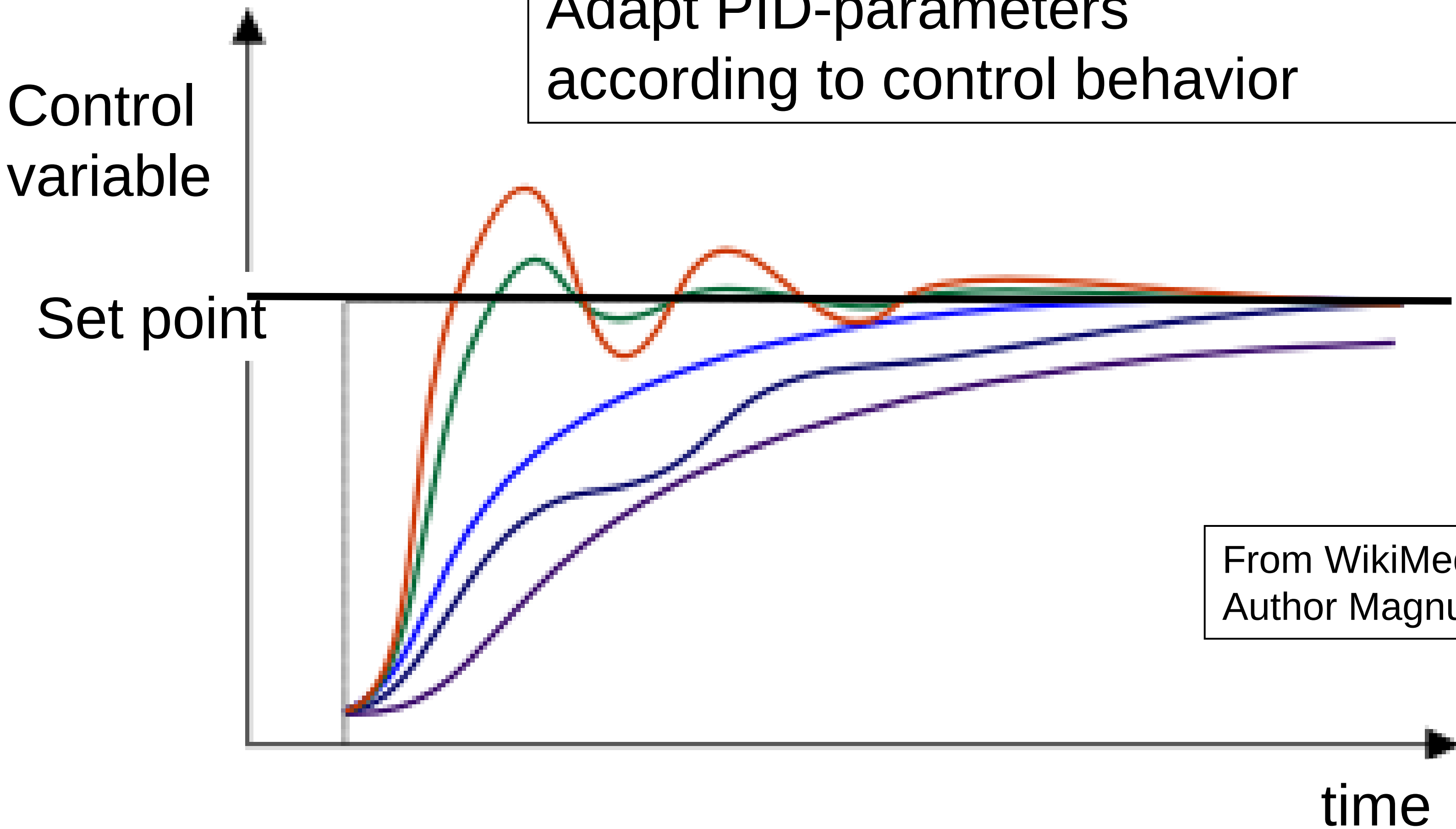
$$u(t) = K_P \cdot e(t) + K_I \sum_{i=1}^{l} e(t_i)\Delta t_i + K_D/\Delta t \cdot [e(t)-e(t-1)]$$

with appropriately chosen constants K_P, K_I and K_D



(Empirical) Design

Adapt PID-parameters
according to control behavior



From WikiMedia,
Author Magnus Manske

Further Controllers

- Fuzzy-Control:
  - Fuzzification:

    Transformation of controlled values y(t) to linguistic terms
  - Application of Fuzzy-rules for linguistic terms
  - Defuzzification:

    Transformation of linguistic terms to control values u(t)
- Neural Networks etc.

Keyframe Controller

Fixed time to arrive at target keyframe.

(Linear) interpolation according to time.

Some smoothness by inertia of limbs/motors.

Customized motors have their own controllers …

RoboNewbie uses some kind of proportional controller (difference to target angles)

Jacobi-Matrix

Relation between workspace with pose**s** $p=(p_1,\ldots,p_m)$ and configuration space with configurations $q=(q_1,\ldots,q_n)$ is given by Kinematics: ***p=f(q)***

Kinematics of motions (velocities) with control parameters q:

$$\boldsymbol{dp/dt = df(q)/dt = \partial f(q)/\partial q \cdot dq/dt = J\, dq/dt}$$

Jacobi-Matrix: $J = \partial f(q)/\partial q = [\partial f_i / \partial q_j]_{ij}$

$$J = \begin{pmatrix} \partial f_1/\partial q_1 & \ldots & \partial f_1/\partial q_n \\ & & \\ \partial f_m/\partial q_1 & \ldots & \partial f_m/\partial q_n \end{pmatrix}$$

Jacobi-Matrix

Approximation of small deviations Dp near p=f(q) is given by

$$\boldsymbol{Dp \approx J(p)\ Dq}$$

To reach a position p‘= p+Dp from p=f(q)

the control can calculate Dq such that

$$\boldsymbol{p' = p + Dp \approx f(q) + J(p)\ Dq}$$

and then perform Dq .

Inverse Jacobi-Matrix

Kinematics of motions:

$$dp/dt = J(p)\ dq/dt$$

Inverse Kinematics of motions:

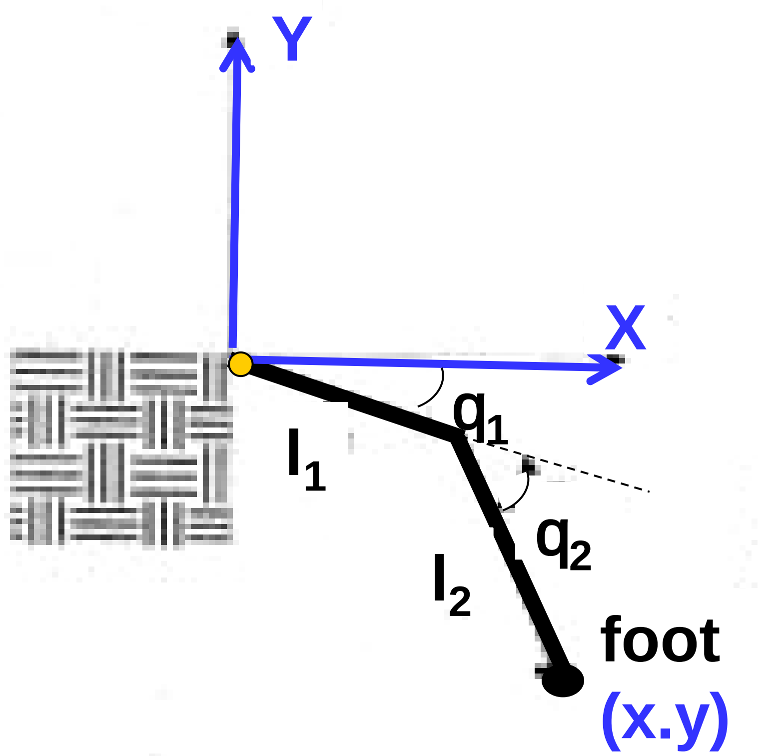
$$dq/dt = J^{-1}(p)\ dp/dt$$

The change Dq of control parameters q
for change Dp of position p is approximated by:

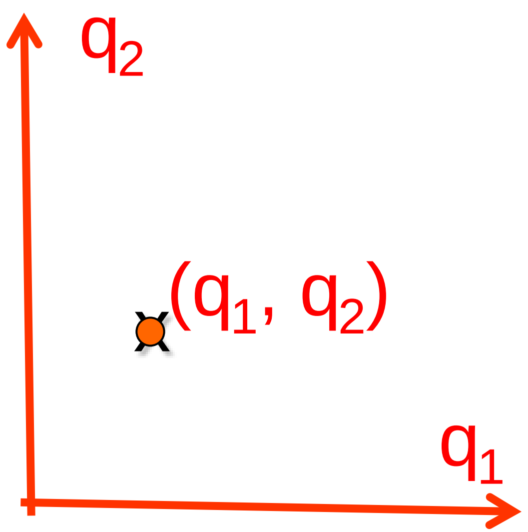
$$Dq = J^{-1}(p)\ Dp$$

Example „Planar Leg“

Work space x,y



Control space q_1, q_2



Example „Planar Leg“

$$\begin{bmatrix} x \\ y \end{bmatrix} = l_1 \begin{bmatrix} \cos(\theta_1) \\ \sin(\theta_1) \end{bmatrix} + l_2 \begin{bmatrix} \cos(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2) \end{bmatrix}$$

$$p = f(q) = \begin{bmatrix} f_x(Q_1, Q_2) \\ f_y(Q_1, Q_2) \end{bmatrix} = \begin{bmatrix} l_1 \cos(Q_1) + l_2 \cos(Q_1 + Q_2) \\ l_1 \sin(Q_1) + l_2 \sin(Q_1 + Q_2) \end{bmatrix}$$

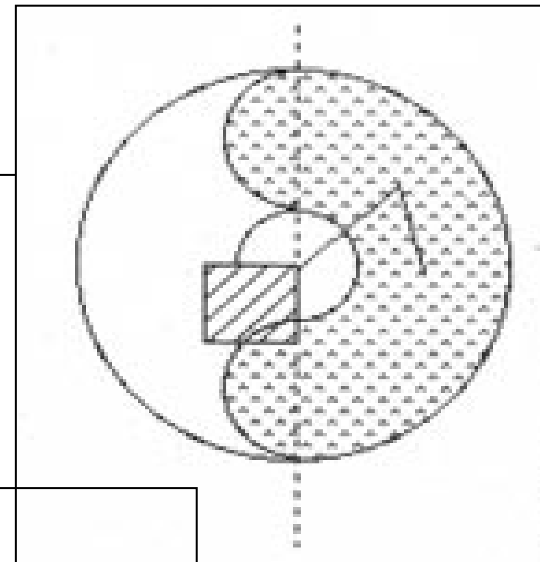
$$J = \partial f(q) / \partial q = \begin{bmatrix} -l_1 \sin(Q_1) - l_2 \sin(Q_1 + Q_2) & -l_2 \sin(Q_1 + Q_2) \\ l_1 \cos(Q_1) + l_2 \cos(Q_1 + Q_2) & l_2 \cos(Q_1 + Q_2) \end{bmatrix}$$

Example „Planar Leg"

Determinant of Jacobi Matrix:

$$\left| \partial f(q) / \partial q \right| = \left| \begin{bmatrix} -l_1 \sin(Q_1) - l_2 \sin(Q_1 + Q_2) & -l_2 \sin(Q_1 + Q_2) \\ l_1 \cos(Q_1) + l_2 \cos(Q_1 + Q_2) & l_2 \cos(Q_1 + Q_2) \end{bmatrix} \right|$$

$$= l_1 l_2 \sin(Q_2) = 0 \quad \textit{for } Q_2 = 0, p, -p$$



Restricted motion
for $Q_2 = 0, p, -p$
(singularities)

Example from Dudek/Jenkin:
Computational Principles of Mobile Robotics

Singularities of Jacobi Matrix

p = f(q) is not invertible at p if $|J(p)| = 0$:

Some points in the neighborhood of p are not reachable.
Values of control parameters can become very high in the neighborhood of p.

Controls avoid neighborhood of p because of problems for control.

Pseudo Inverse of Jacobian Matrix

(Moore-Penrose-Inverse)

Pseudo-Inverse J^+ can be used

instead of J^{-1} for non-quadratic m´ n - matrices J :

(n = number of control parameters)

If rank J(p) = n then

- Pseudo-Inverse $J^+ = (J^t \, J)^{-1} \, J^t$
- J^+ is Left-Inverse of J

$$Dp \approx J(p)\, Dq$$

$$J^+(p)\, Dp \approx J^+(p)\, J(p)\, Dq = (J^t(p)\, J(p))^{-1} J^t(p)\, J(p)\, Dq$$

$$= (J^t(p)\, J(p))^{-1} (J^t(p) J(p))\, Dq = Dq$$

$$Dq \approx J(p)^+ Dp$$

Pseudo Inverse of Jacobian Matrix

(Moore-Penrose-Inverse)

Problems near singularities at p (rank J(p)<n):

- Several neighboring points are not reachable from exactly p (no motion into that direction)
- Small changes of Dp lead to very huge changes Dq of control parameters in the neighborhood of p

More complex calculation of $J(p)^+$ if rank J(p) < n :

$Dq \approx J(p)^+ Dp$

gives best possible solution Dq,

i.e. minimizes the quadratic error $(Dp - J(p)\, Dq)^2$

Control by Keyframes

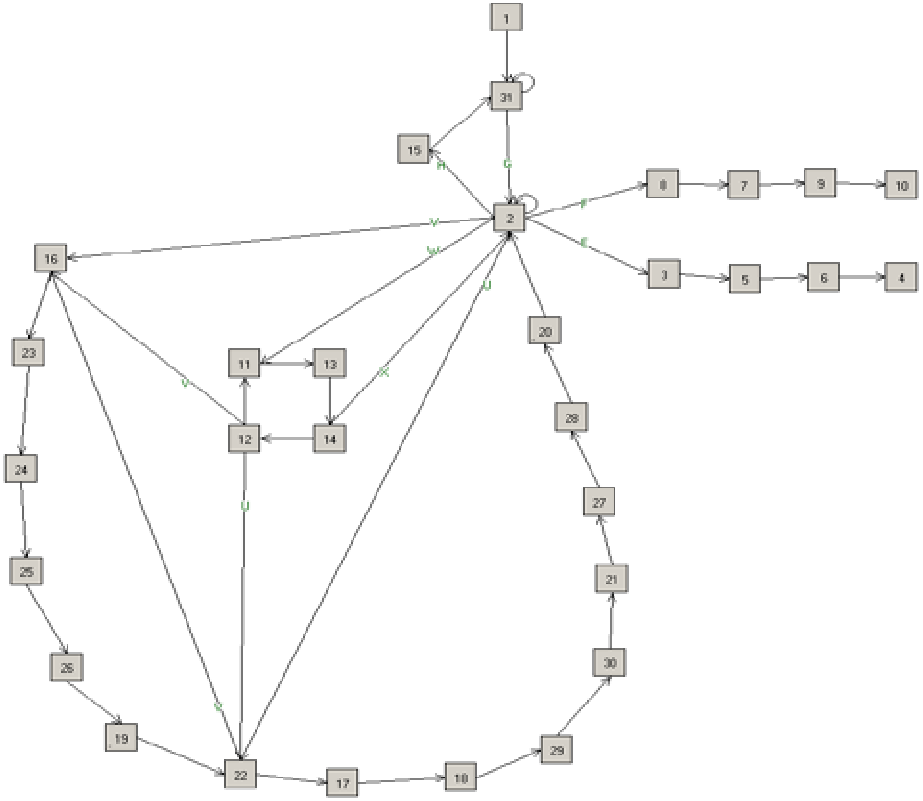
Keyframes define „characteristic“ poses of a trajectory.

Keyframe Motions:

Trajectories are traversed by transitions between keyframes (predefined poses) in predefined times.

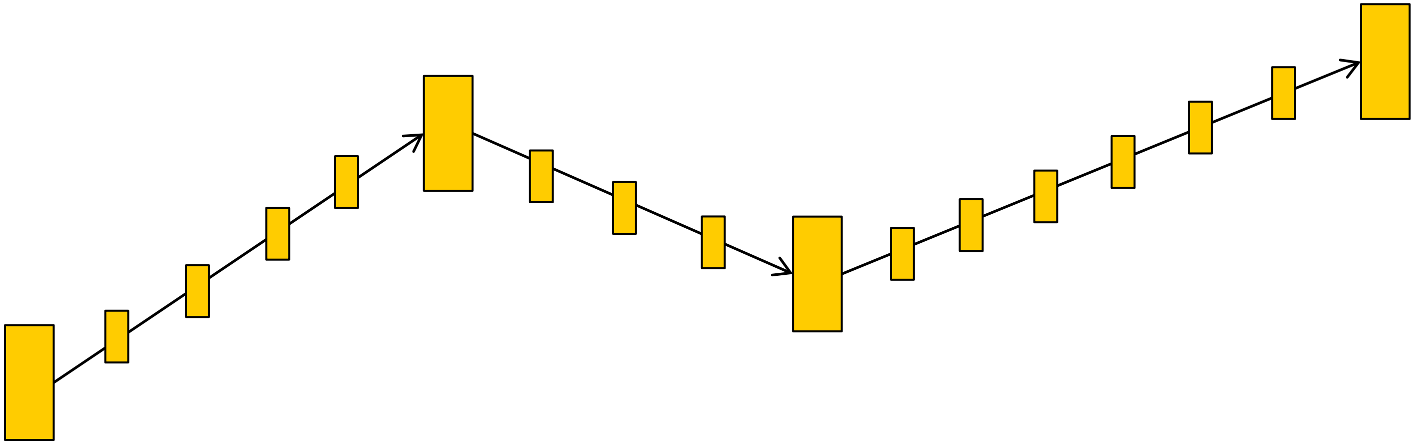
They are given as sequences or nets of keyframes.

Branching in nets according to different situations e.g. user commands or sensor inputs.



Control by Keyframes

For control of a keyframe motion,
the actuators are controlled accordingly by a “*keyframe player*”,
e.g. interpolation by automatically calculated intermediate poses.



Sensor feed back can be used to adapt the interpolated poses.

Usually, keyframes are not changed during motion.

Smoothness of keyframe motions

Smoothness of keyframe motion is influenced

By physical properties of (real) robots and environment,
e.g. inertia, friction, backlash, parameters of motors, …
(servo motors have separate controllers)

By keyframe player:

- Splines etc. instead of linear interpolation
  can be used for smoothing (especially in simulation)

By design of keyframes:

- Designer of keyframes can introduce more keyframes at „critical“ parts of the desired trajectory.
- Machine learning can be used to optimize keyframes
  (resp. the common result of keyframe and keyframe player)

Simple Physical Controls

Control by simple physical processes without calculations, e.g.

- Thermostat
- Braitenberg vehicle
- Dynamic Passive Walker (see below)

Model Based Motion Control

Actuation for next pose(s) determined by some model:

Calculation by some criteria to be maintained,

e.g. stability/balance by CoM, ZMP (see below).

Actuator commands by Inverse Kinematics

(for drives, for limbs …)

Outline

Introduction

Kinematics of Poses

Kinematics of Drive Systems

Trajectories

Motion Planning

Motion Control

Motions of Legged Robots

Optimization/Learning of Motions

Biologically Inspired Motions

Statically Stable Balance

Projection of center of mass (CoM) within the convex hull of the ground contact points ("support-polygon")

- Stable walk with 4 legs:
  Only 1 leg lifted with shift of weight
- Stable walk with 6 legs:
  Simultaneous movement of 3 legs without shift of weight

Dynamic Balance

Projection of CoM may be outside of support polygon

Appropriate movements prevent falling over

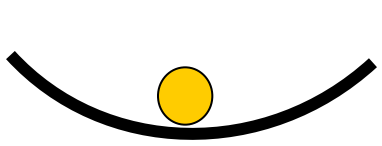
Equilibrium/Balance

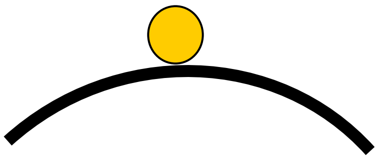
Static equilibrium: Robot in persistent state (e.g. standing)

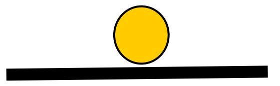
Dynamic equilibrium: Robot in persistent motion (e.g. walking)

After disturbance:

- Return to equilibrium by itself: *Stable equilibrium*
- Further departure from equilibrium: *Unstable equilibrium*
- Indifference: *Indifferent equilibrium*







Running patterns

Complete cycle of all leg movements:

2 phases for each leg:

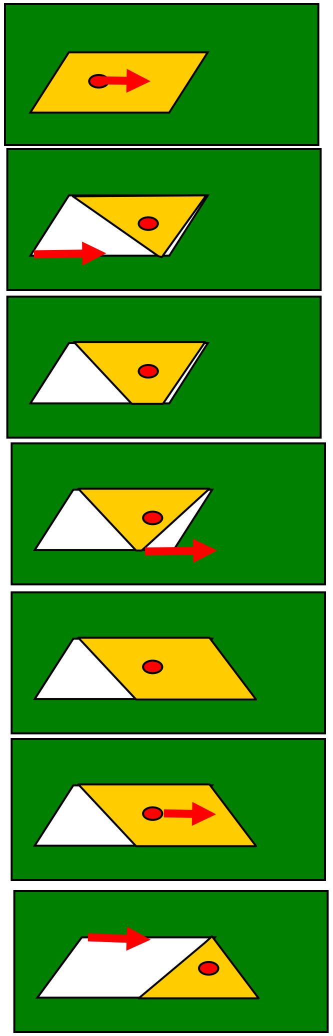
- Support phase (stance): ground contact
  contact points to body and ground determine joint angles
- Transfer phase (swing): Free movement
  trajectory to next attachment point determines joint angles

Duty-factor = Percentage of the ground contact time

e.g. Trot (always 2 of 4 feet on the ground): Duty factor = 0.5

Further details with more phases, e.g.:
lift - move forward – put down – roll off

Statically Stable 4 Legged Walk



R1. Shift CoM

R2. Right hind leg in the air

R3. Right hind leg on ground

R4. Right front leg in the air

R5. Right front leg on ground

L1. Shift CoM

L2. Left hind leg in the air

…

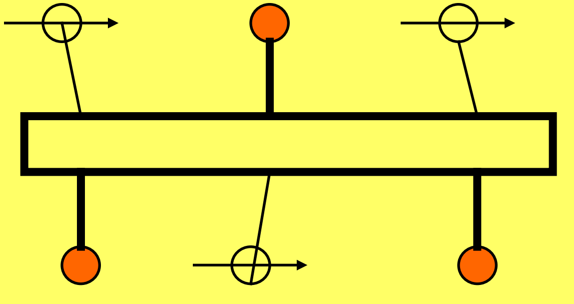
Statically Stable Walk

Robot can stop at any given time in statically stable balance.

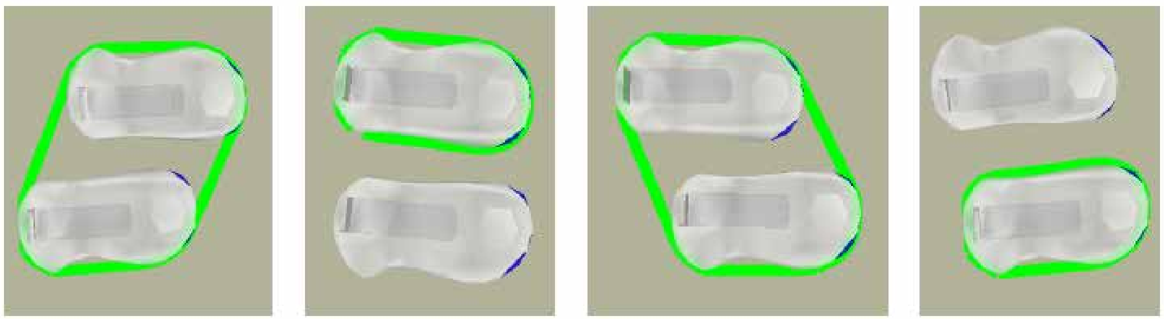
Transitions between statically stable balance states.

CoM always above support polygon.

Statically stable walk with 6 legs:
Always 3 feet on the ground
CoM above support triangle

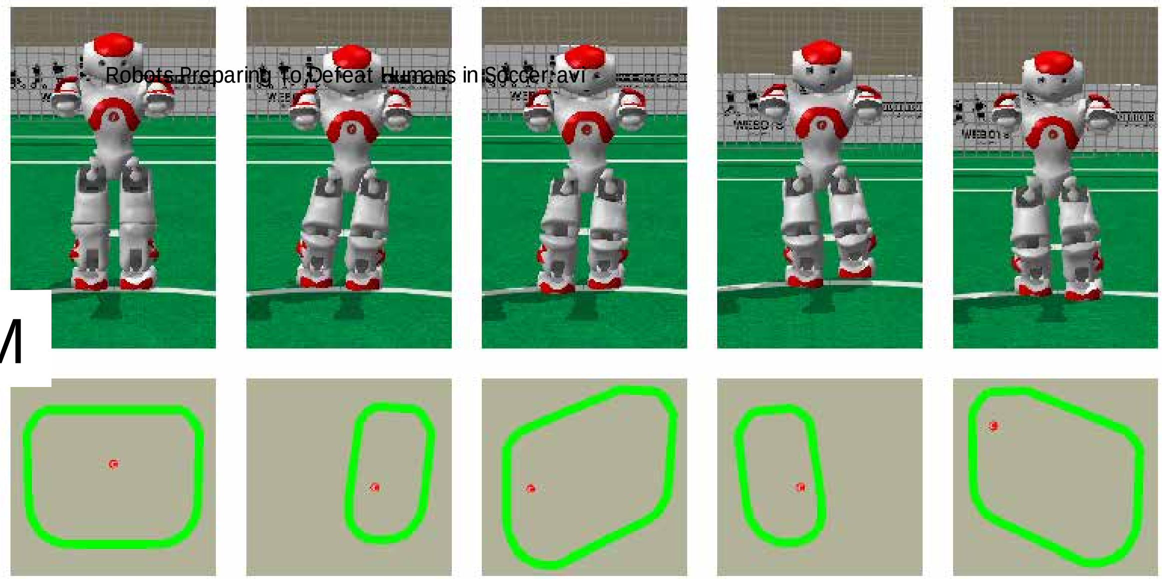


Statically Stable Walk of Humanoid



(a) Double Support Left (b) Single Support Left (c) Double Support Right (d) Single Support Right

Diploma Thesis
Oliver Welter



Projection of CoM

Design of Static Stable Walk

Motion by transitions between static stable equilibriums

Control of the legs by means of inverse kinematics, calculation along the kinematic chains:

- Define path of CoM
- This defines connection points between body and legs
- Foot point of standing legs
- Trajectories of moving legs

Further parameters by optimization methods

Dynamic Walk

„prevented falling over“

No universally accepted definition
(“not statically stable walk”)

Unlike stable running:
CoM at least temporarily outside support polygon
(not statically stable equilibrium when interrupting)

Possible definiton by
"dynamically stable equilibrium" for trajectory

Outline

Introduction

Kinematics of Poses

Kinematics of Drive Systems

Trajectories

Motion Planning

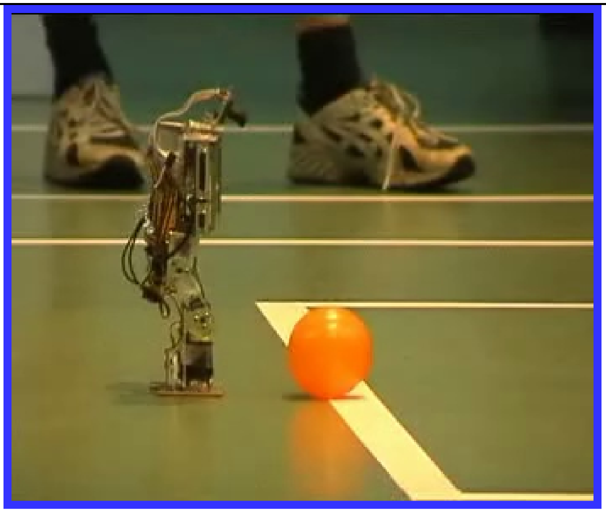
Motion Control

Motions of Legged Robots**/Humanoid Robots**

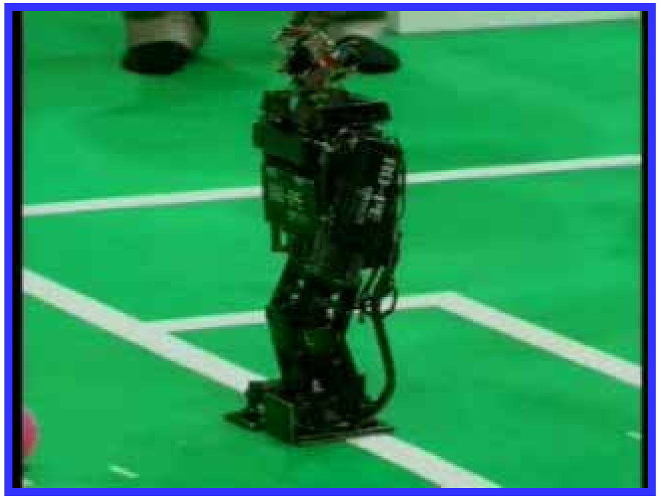
Optimization/Learning of Motions

Biologically Inspired Motions

Humanoid Robots: RoboCup



2002



2005

2010

2011

Humanoid Robots: BOSTON DYNAMICS

Petman

"ATLAS"

Humanoid Robots: Avatars

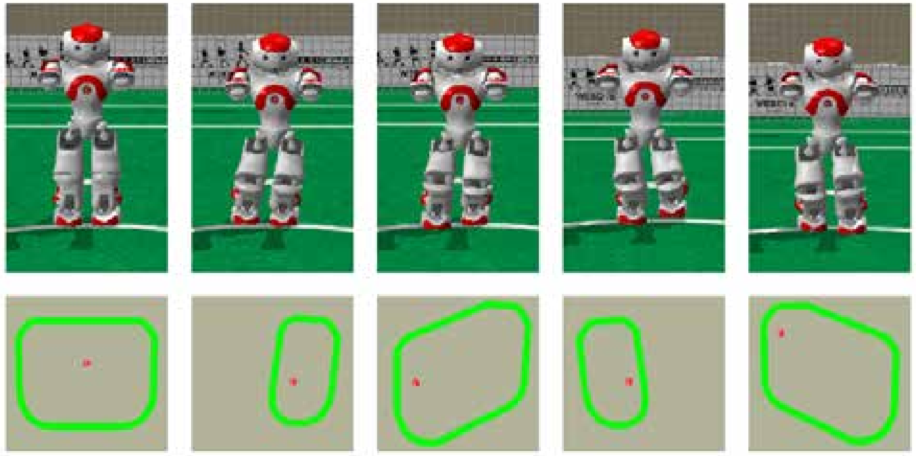
Asimo + Hanson Robotics

Ishiguro: Androids

Biped Walk (Humanoid Robots)

Statically Stable Walk: Projection of CoM inside support area

- slow "walk"



Diploma Thesis
Oliver Welter

Dynamic Walk: Projection of CoM may be outside support area

- faster walk
- problem: how to prevent from falling

Model Based Dynamic Walk

Calculate trajectories by physical models like

- Inverted pendulum for stand leg
- Pendulum for swing leg
- Center of Mass
- Zero Moment Point (ZMP)

Problem:
Model based control needs precise hardware.

No elasticity as in nature.

Zero Moment Point (ZMP)

ZERO-MOMENT POINT — THIRTY FIVE YEARS OF ITS LIFE

MIOMIR VUKOBRATOVIĆ

Institute Mihajlo Pupin,Volgina 15
11000-Belgrade, Serbia and Montenegro
vuk@robot.imp.bg.ac.yu

BRANISLAV BOROVAC

University of Novi Sad, Faculty of Technical Sciences
21000-Novi Sad, Trg D, Obradovica 6, Serbia and Montenegro
borovac@uns.ns.ac.yu

Received 24 October 2003
Accepted 8 January 2004

Zero Moment Point (ZMP)

„Its first practical demonstration took place in Japan in 1984, at Waseda University, Laboratory of Ichiro Kato, in the **first dynamically balanced** robot WL-10RD of the robotic family WABOT. The paper gives an in-depth discussion of source results concerning ZMP, paying particular attention to some delicate issues that may lead to confusion if this method is applied in a mechanistic manner onto irregularcases of artificial gait, i.e. in the case of loss of dynamic balance of a humanoid robot.“

(Introduction M.Vucobratovic, B.Borovac:

„Zero-Moment Point: 35 Years of its Life“)

Zero Moment Point (ZMP)

Forces and moments in single support phase are considered:

Forces/moments acting on the support foot:

Influence of body to ankle, gravity, ground reaction, friction.

Dynamic equilibrium:

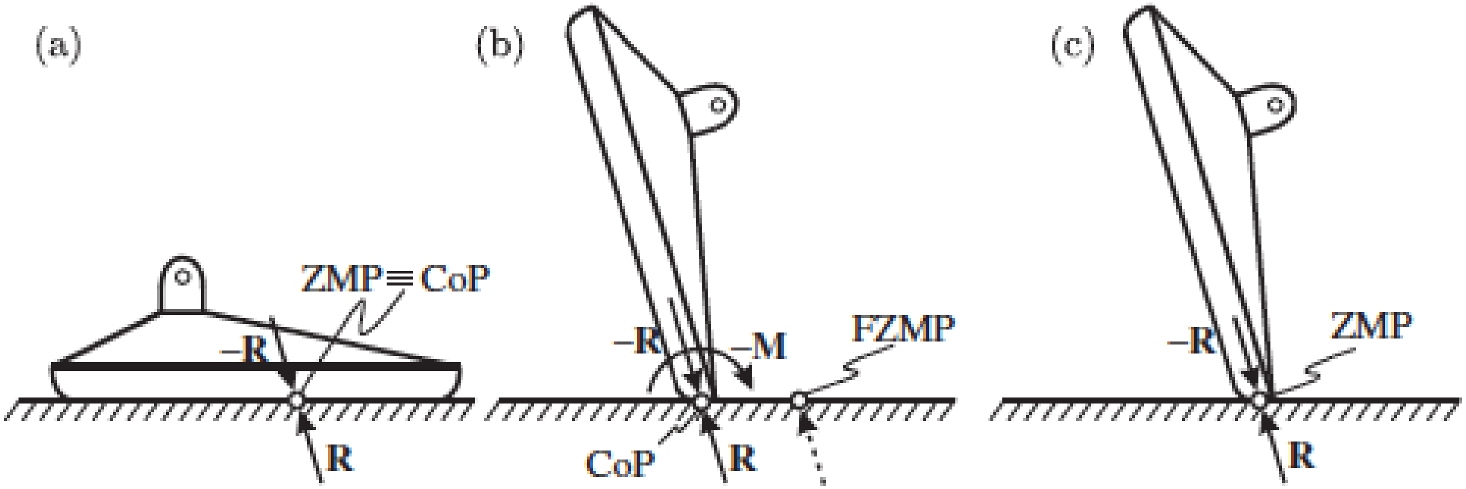
- horizontal moments $M_x = M_y = 0$
  at CoP (= center of pressure) of foot

If such a point does not exist inside support polygon, the robot will rotate over the foot edge and overturn.

"*Zero-Moment-Point*" if inside (!) support polygon.

Different (sometimes conflicting) definitions in the literature.

M.Vucobratovic, B.Borovac:
„Zero-Moment Point: 35 Years of its Life“



Possible relations between ZMP and CoP :

(a) dynamically balanced gait,

(b) unbalanced gait where ZMP does not exist and the ground reaction force acting point is CoP while the point where *Mx = 0 and My = 0 is outside the support polygon (FZMP). The* system as a whole rotates about the foot edge and overturns,

(c) tiptoe dynamic balance (“balletic motion”).

ZMP Control

Condition for dynamically stable walk:
ZMP within support polygon (projection of CoM may be outside)

Conditions for control using ZMP:
- Keep ZMP of stand leg inside support polygon
- ZMP of swing leg inside support polygon at touch down

Define Trajectories (e.g. by forward simulation):
- Maintain conditions
  (e.g. by related shift of CoM using hip)

Different implementations.

Approximated Calculation of ZMP

Calculate ZMP = CoP (Center of Pressure) on feet

(as long as not on the foot edge)

ZMP as result of measured forces at the feet

(cf. FRP in SimSpark)

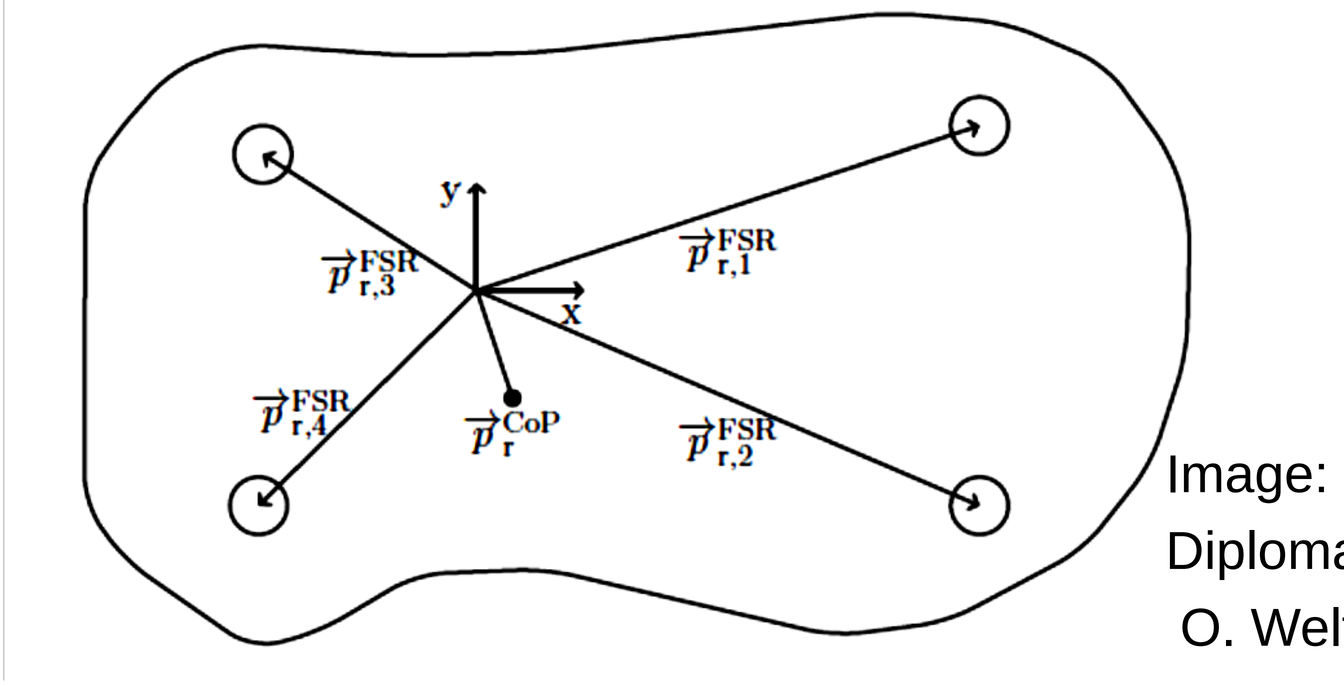


Image:
Diploma thesis
O. Welter

Approximated Calculation of ZMP

Calculate ZMP from CoM by physical model:

CoM at the top of stand leg as inverted pendulum

Forward simulation for optimal ZMP positions

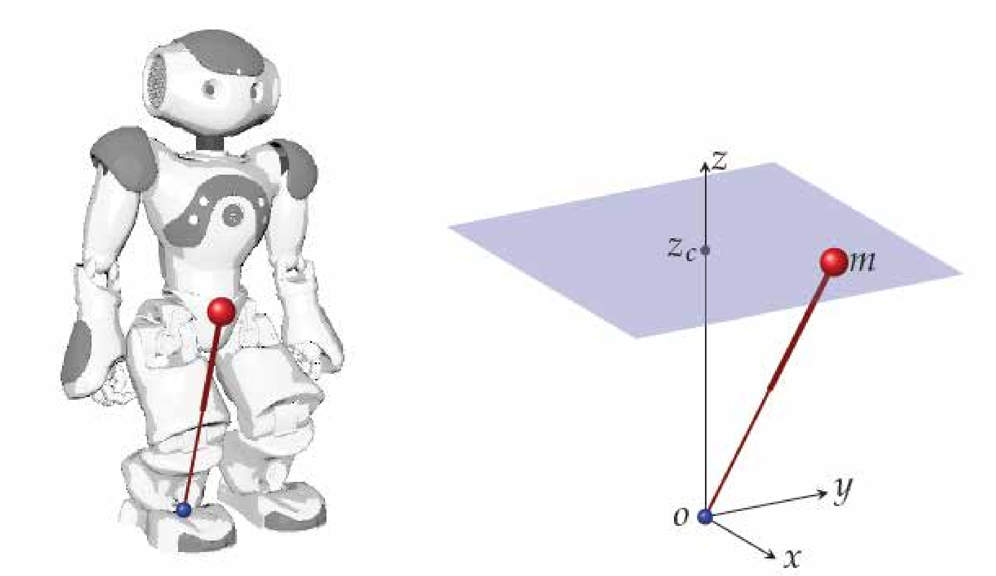
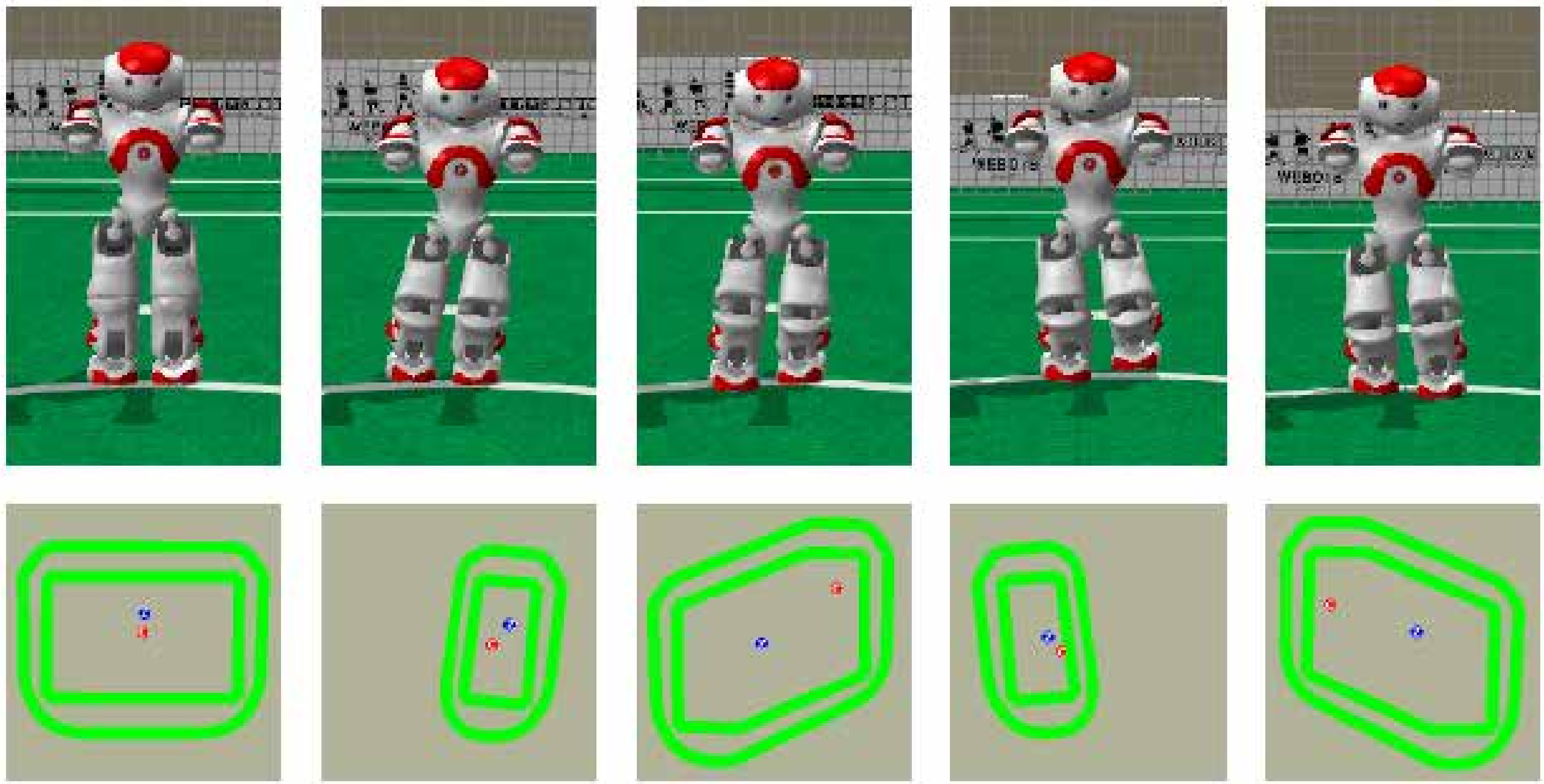


Image:
Yuan Xu (NaoTH)

Zero Moment Point (ZMP)



Displacement of the projections of CoM (red) and ZMP (blue) while walking

(Diploma thesis O. Welter)

Walk for Nao

Walk by Yuan Xu
NaoTeam Humboldt

Outline

Introduction

Kinematics of Poses

Kinematics of Drive Systems

Trajectories

Motion Planning

Motion Control

Motions of Legged Robots

Optimization/Learning of Motions

Biologically Inspired Motions

Machine Learning, Optimization

Many parameters are used for control.

Problem of optimal choice, optimization

e.g. by

- Gradient descent
- Evolutionary methods
- Reinforcement learning

Fitness (Quality) of walk:

- duration
- speed
- accuracy of path
- energy consumption
- aesthetics

- Experiments with real robots are expensive
- Experiments with simulated robots are not strictly equivalent

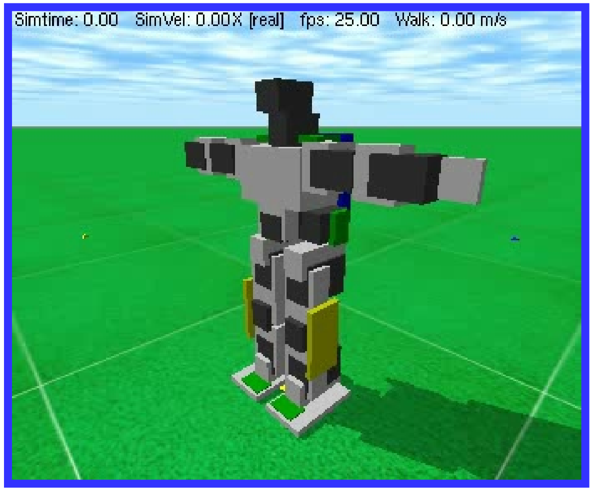
Ø Combination of both.

Ø PhD thesis of Yuan Xu

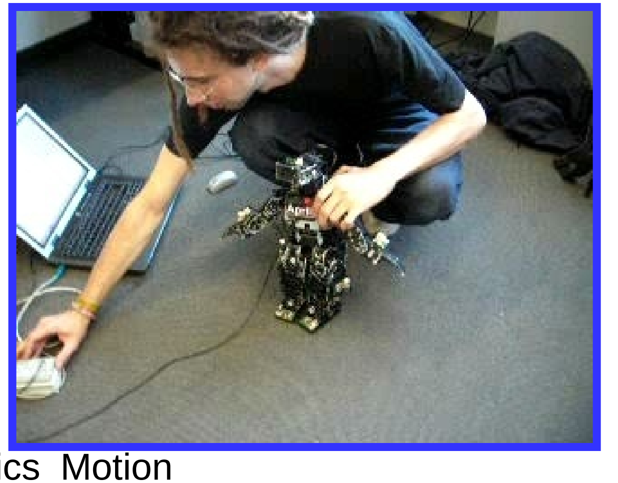
Machine Learning, Optimization

Simloid (Diploma thesis Daniel Hein):

Evolved walks of simulated Bioloid



Transfer to real Bioloid



Case Study: Optimized walk for AIBO

Diploma thesis Uwe Düffert 2004

- Optimize omnidirectional walk
- Calibrating the running movements (correct control)

Walk parameters:

Forward velocity *dx/dt*

Sideward velocity *dy/dt*

Rotation velocity *df/dt*

Automate:

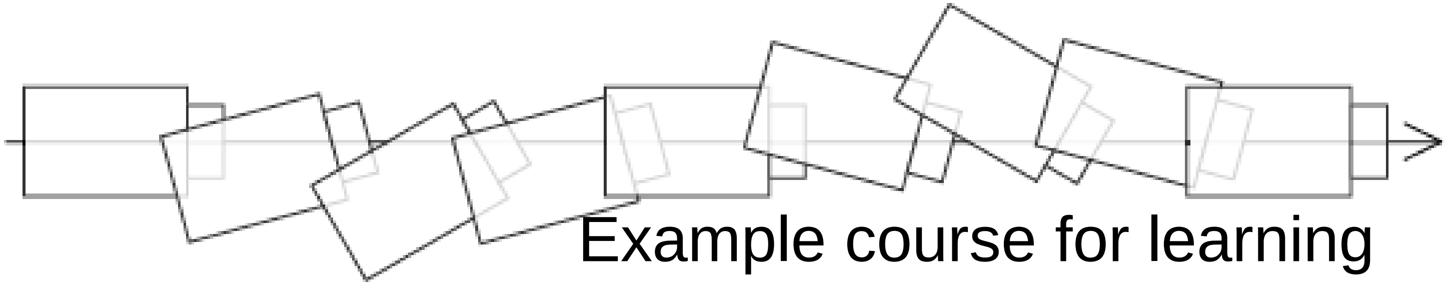
- Learning process
- Tests
- Evaluation (test environment)

AIBO: Requirements for Walk

Optimization: Find optimal walk

Omnidirectional walk:

- walk in any direction
  (forward, backward, sideways, diagonally)
- rotate while walking
- smooth transitions between the directions
  (Without "stop" or “switch")
- high speeds possible
- correct implementation of the required movements
- aesthetics



Example course for learning

AIBO: Basic Design Decisions

Define trajectory of CoM (according to desired path).

This defines coordinates of shoulders.

Define foot positions by “Wheel model”

according to desired path

(maybe with slipping during changes).

Duty factor = 0.5:

Only the 2 diagonally positioned feet have ground contact (not statically stable).

Define trajectory of feet according to given curve template.

AIBO: Parameters for Optimization

Reduce parameters to few parameters which

- have great impact
- can be predefined

1. Rest position of feet relative to the body
2. Trajectory of the legs (height, length)
3. Gait: time points for swing and stance

AIBO: Decomposition of Task

Experience: Optimal parameter sets for fast forward walk are not optimal for fast backward etc.

Consequently: Different parameter sets $P_i = (p_{i1},...,p_{in})$ for different requirements A_i :

In total: 127 different requirements for

- Direction (8 values)
- Ratio Walk/Turn (7 levels)
- Speed (3 levels)

Not all combinations are used. The combinations are more uniform than a combination by forward/sideways/turning speed.

AIBO: Decomposition of the Task

and restriction to discrete values

Direction of Walk ; $\alpha = \arctan(\dot{x}, \dot{y})$

$$\rightarrow \left[-\pi, -\frac{3\pi}{4}, -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\right]$$

Ratio Walk/Turn $\delta = \frac{2}{\pi} \arctan\left(\frac{v}{v_{max}}, \frac{\dot{\varphi}}{\dot{\varphi}_{max}}\right)$

$$\rightarrow \left[\underbrace{-1}_{\text{right}}, -\frac{3}{10}, -\frac{1}{10}, 0, \frac{1}{10}, \frac{3}{10}, \underbrace{1}_{\text{left}} \right]$$

Speed ; $r = \sqrt{\left(\frac{v}{v_{max}}\right)^2 + \left(\frac{\dot{\varphi}}{\dot{\varphi}_{max}}\right)^2}$

$\rightarrow$ slow middle fast .

AIBO: Setup of Experiments

Optimization by evolutionary methods:

- Fitness of parameter sets (individuals) $P = (p_1,\ldots,p_n)$ evaluated by walks in real environment
- Fitness by correspondence to required path and time

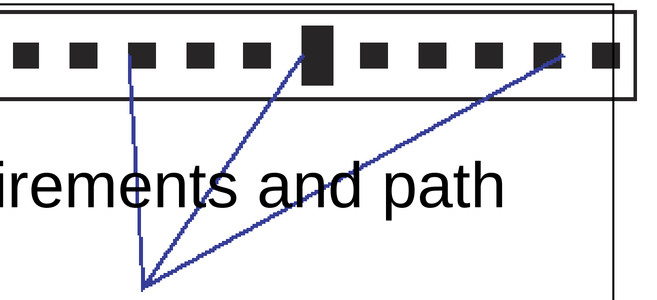
Automatization of experiments

by appropriately designed environment:

- Robot tries to walk according to required path and time
- Robot measures path and time using special landmarks
- Robot evaluates fitness by comparing actual with requested path and time

AIBO: Setup of Experiments

Landmarks for orientation



- Used for determining control requirements and path corrections
- Used for evaluation of actual path (fitness)

Image by AIBO Camera

with identified landmarks

AIBO: Fitness F(P)

$$F(P) \;=\; \dot{x} - \Delta y/6 - 33\Delta\varphi - \left(10^{-5}\ddot{z} - 5\right) - 40p_{\mathrm{blind}}$$

dx/dt : average speed in x direction (along the course)

Dy : average deviation of y-position (distance to requested line)

Df : averaged deviation from requested direction

d^2z/dt^2 : averaged acceleration in z direction
(unpleasant hard pounding)

p_{blind} : percentage of images where landmarks are not identified
(strong deviation or strong vibration)

AIBO: Experiments

Optimal parameter $P_i = (p_{i1},...,p_{in})$

were determined for the 127 walk requirements A_i by

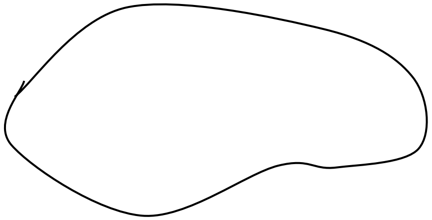
- Evolutionary methods for some (not all) requirements
- Good parameter sets already known and evaluated
- Regarding good transitions between adjacent requirements

AIBO: Further Options:

Pre-evaluation and selection of parameter sets before test with real robot by

- Comparison with similar known parameter sets
- Simulation
- Hill Climbing in parameter space

More complex trajectories of feet



Much efforts in RoboCup:

- Dortmund (Ingo Dahm and others)
- NuBots (Michael Quinlan)
- Austin (Peter Stone)

Speeds of up to 50cm/sec (2-times length of body)

Outline

Introduction

Kinematics of Poses

Kinematics of Drive Systems

Trajectories

Motion Planning

Motion Control

Motions of Legged Robots

Optimization/Learning of Motions

Biologically Inspired Motions

Outline

Introduction

Kinematics of Poses

Kinematics of Drive Systems

Trajectories

Motion Planning

Motion Control

Motions of Legged Robots

Optimization/Learning of Motions

Biologically Inspired Motions

Biological Models

Can be exploited for

Hardware, e.g.:

- Mechanical design (legs, elasticity,…)
- Actuators (muscles, tendons, springs…)
- Sensors (skin sensors, …)

Software, e.g.

- Control loops
- Local/distributed control
- Dynamic systems control
- Perception, sensor data integration

Biological Models

Emergence:

Complex behavior **emerges** from

simple principles by clever design

Situatedness:

Appropriate behavior emerges by

Appropriate interaction with the environment

Examples:

- Put the foot down until ground reaction is sensed on foot (knee, hip, proprioceptive sensors …)
- Move the arms, the upper body etc.

  to compensate acceleration (prevent from falling)
- Shift of CoM at slopes

Mechanical Design

Passive walker

- Inverse pendulum (Stand leg) + Pendulum (Swing leg)
- High center of gravity (hip)
- Additional compensation by arms
- Energy-efficiency

Cornell University

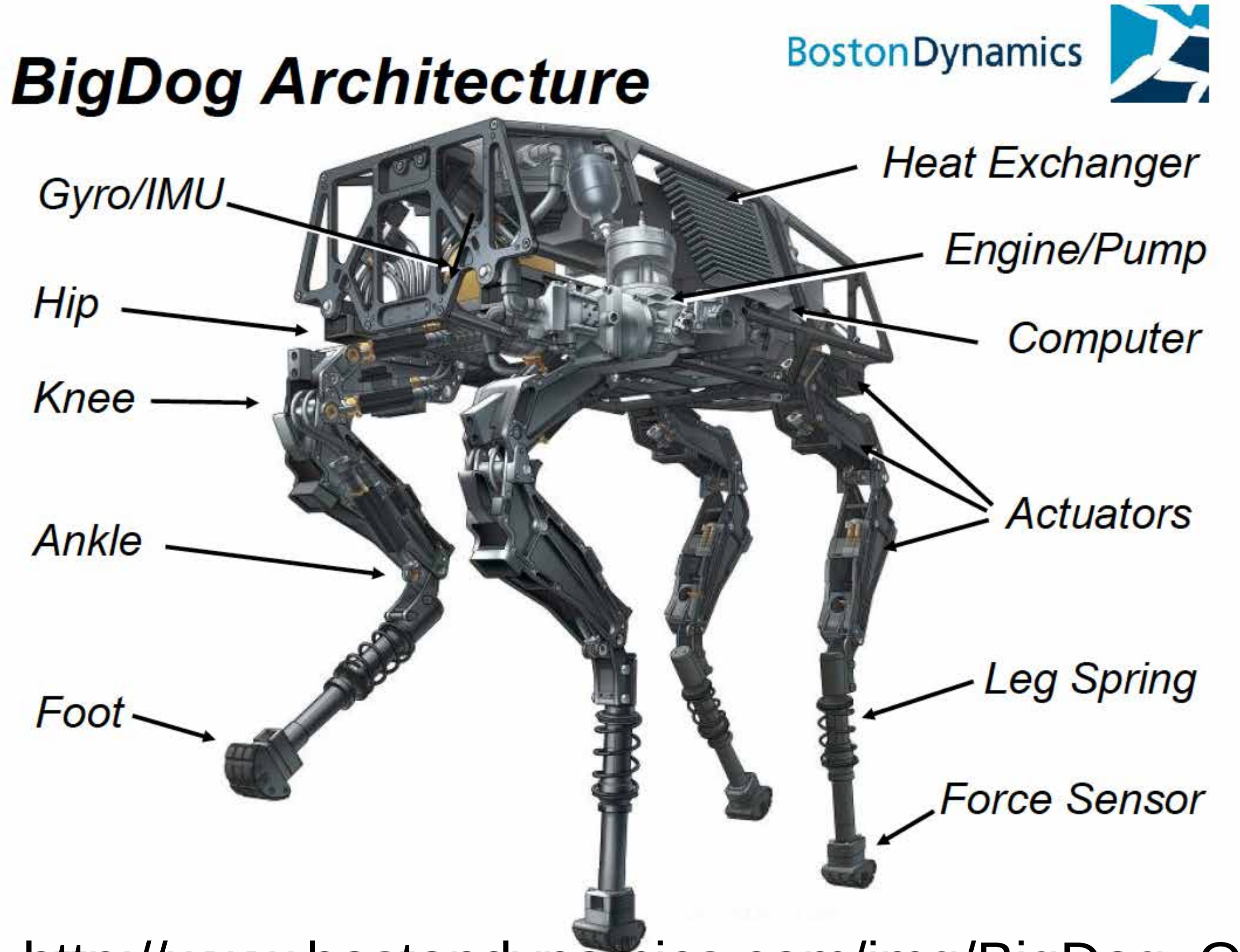
Mechanical Design

Body Shape:

Walking emerges from well designed shape

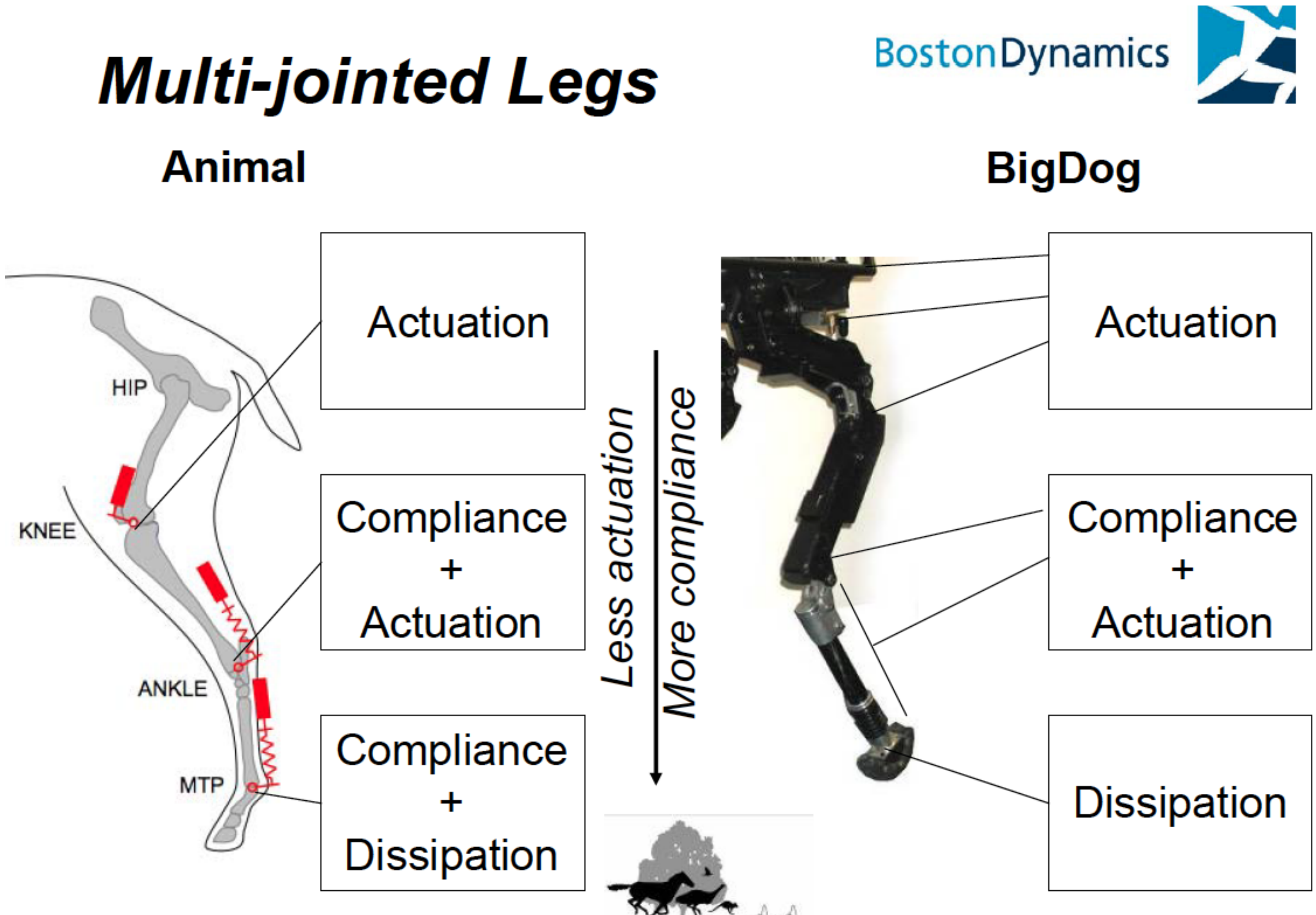
Blickhan, Seyfarth (Jena)

Mechanical Design



http://www.bostondynamics.com/img/BigDog_Overview.pdf

Mechanical Design



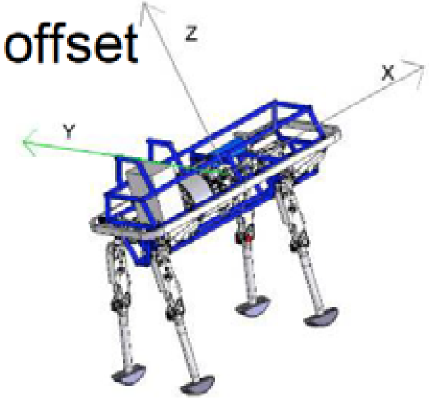
http://www.bostondynamics.com/img/BigDog_Overview.pdf

Mechanical Design and Control

Trot Control

BostonDynamics

- **X** – Closed loop. Speed error corrected by x direction foot forces.
- **Y** – Lateral foot position chosen to offset unwanted lateral body velocity.



- **Z**
- **Roll**
- **Pitch**
- **Yaw**

Coupled Controller. Corrections for height and Euler errors map to y and z direction foot forces.

http://www.bostondynamics.com/img/BigDog_Overview.pdf

Central Pattern Generator (CPG)

Hypothesis:

Cyclic motions of animals (walk, fly, swim, wind, …)

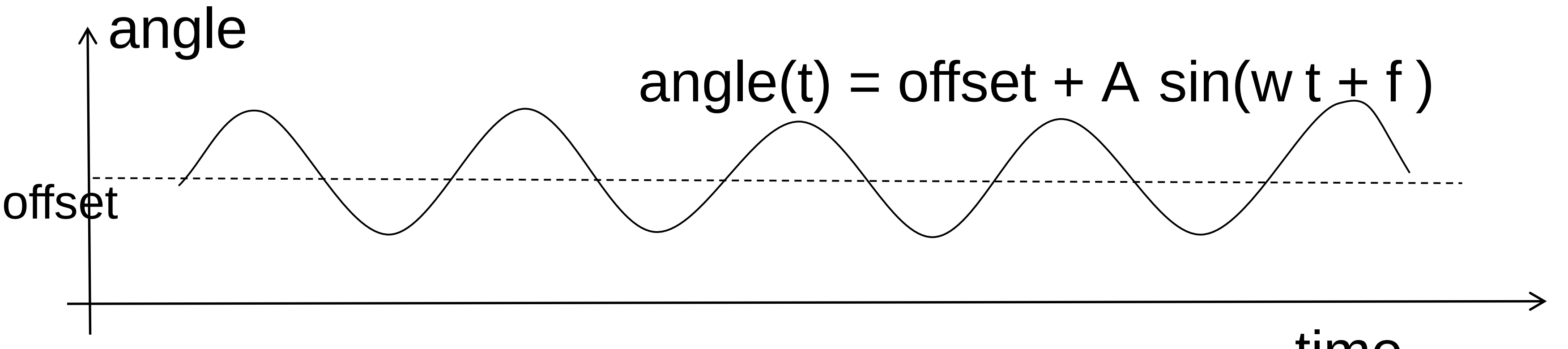
are controlled by oscillating CPG.

Oscillations can be produced by

- Sine-Function(s)
- Recurrent Neural Networks

Oscillations by Sine Function(s)

The trajectory of a joint (e.g. knee joint) oscillates while following the sine function as motor control:

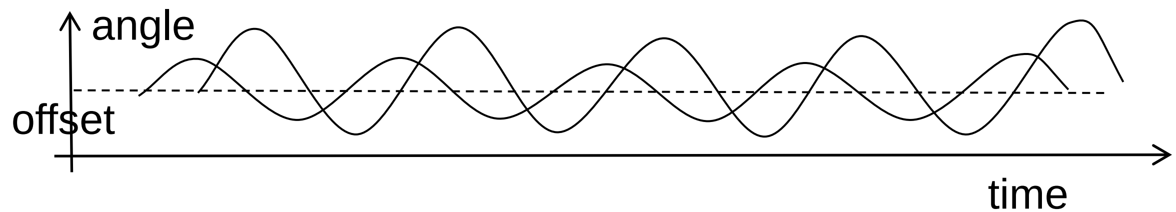


A = amplitude (vertical scaling)
w = angular frequence (horicontal scaling)
f = phase (horicontal shift)
offset (vertical shift)

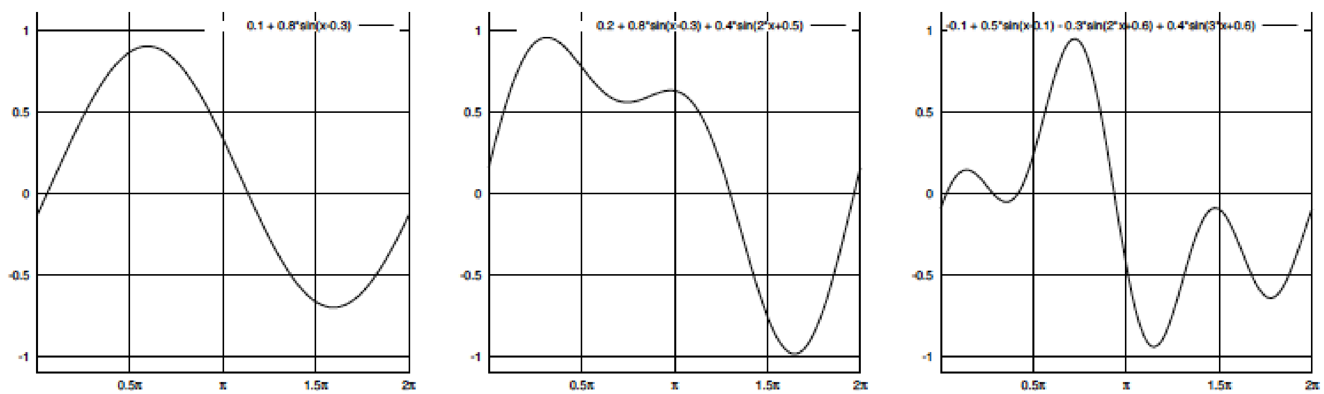
Oscillations by Sine Function(s)

More complex oscillations are performed by combinations of different sine functions (cf. Fourier-series)

$$angle(t) = offset + A_1 \sin(w_1 t + f_1) + A_2 \sin(w_2 t + f_2)$$



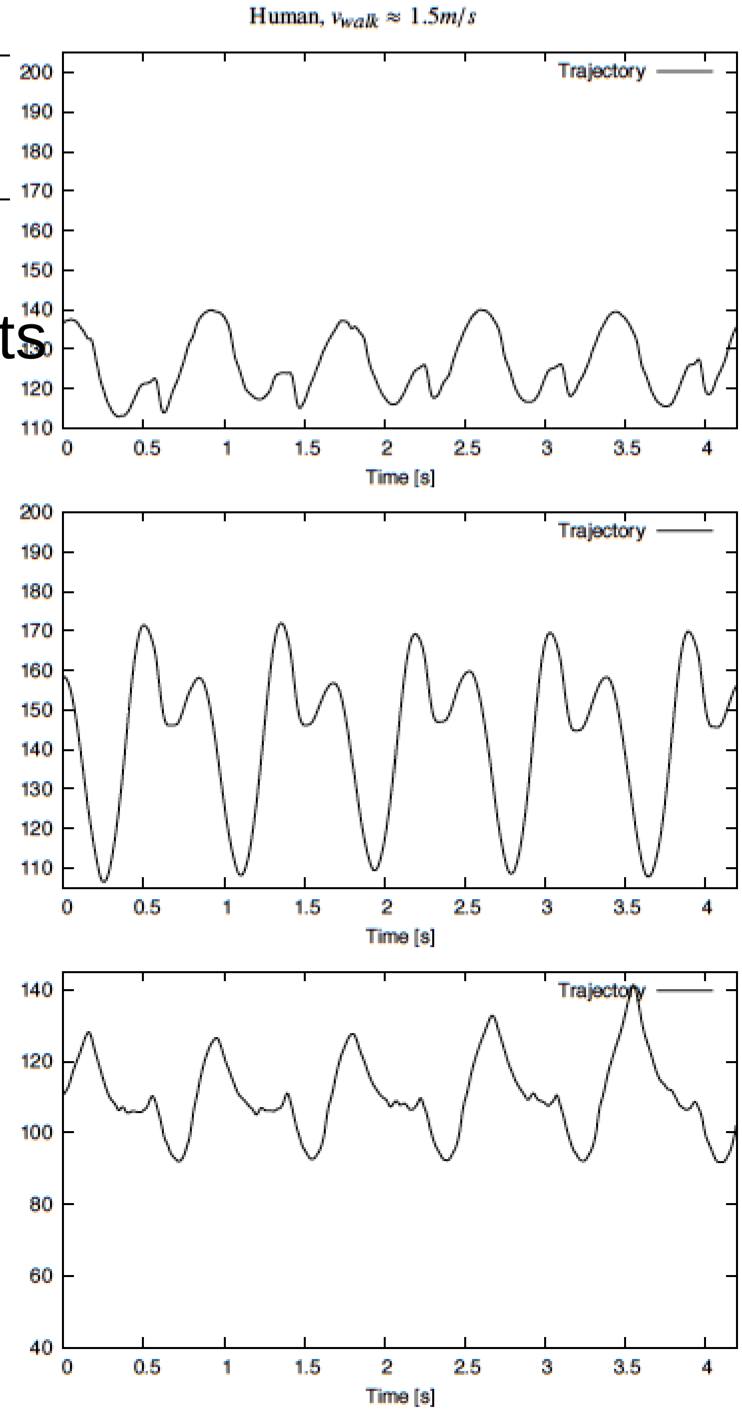
Examples of more complex curves (from Dipl.Thesis D. Hein):



Biology

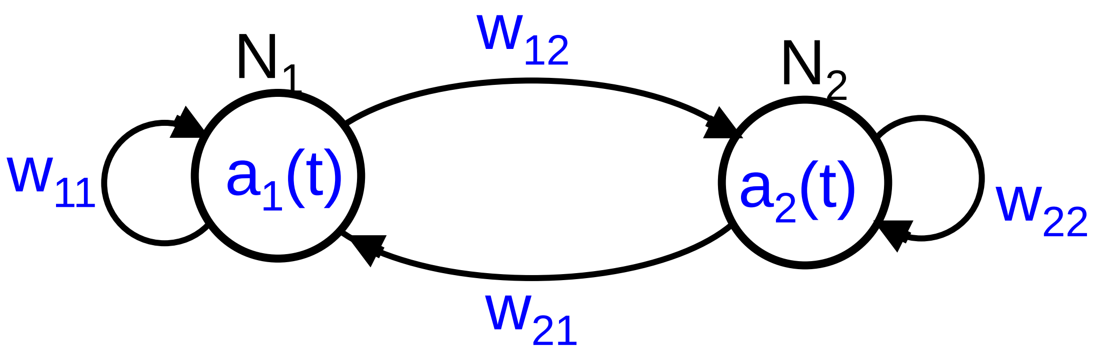
Trajectories of human joints during walk (1,5 m/sec):

- Right hip
- Right knee
- Right ankle



Images: S.Lipfert
Locomotion Lab. Jena

Neural Network Oscillators: Example

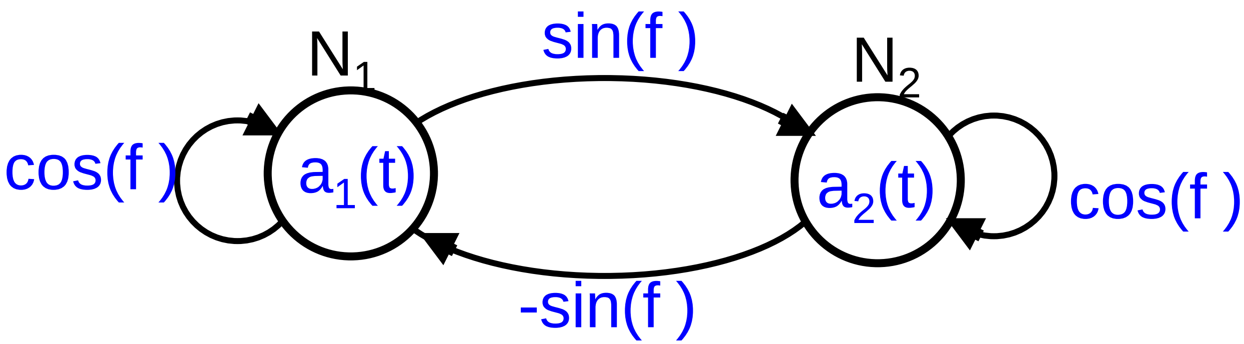


At each time t, the neurons N_1 and $N_2(t)$ are activated by $a_1(t)$ resp. $a_2(t)$ which are recursively computed:

$$a_1(t+1) = a \tanh(w_{11} a_1(t) + w_{21} a_2(t))$$
$$a_2(t+1) = a \tanh(w_{12} a_1(t) + w_{22} a_2(t))$$

$$\begin{pmatrix} a_1(t+1) \\ a_2(t+1) \end{pmatrix} = a \cdot \tanh \cdot \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{pmatrix} \cdot \begin{pmatrix} a_1(t) \\ a_2(t) \end{pmatrix}$$

Neural Network Oscillators: Example



Simplified special case **without tanh**, and

$w_{11} = w_{22} = \cos(\varphi)$, $w_{12} = \sin(\varphi)$, $w_{21} = -\sin(\varphi)$

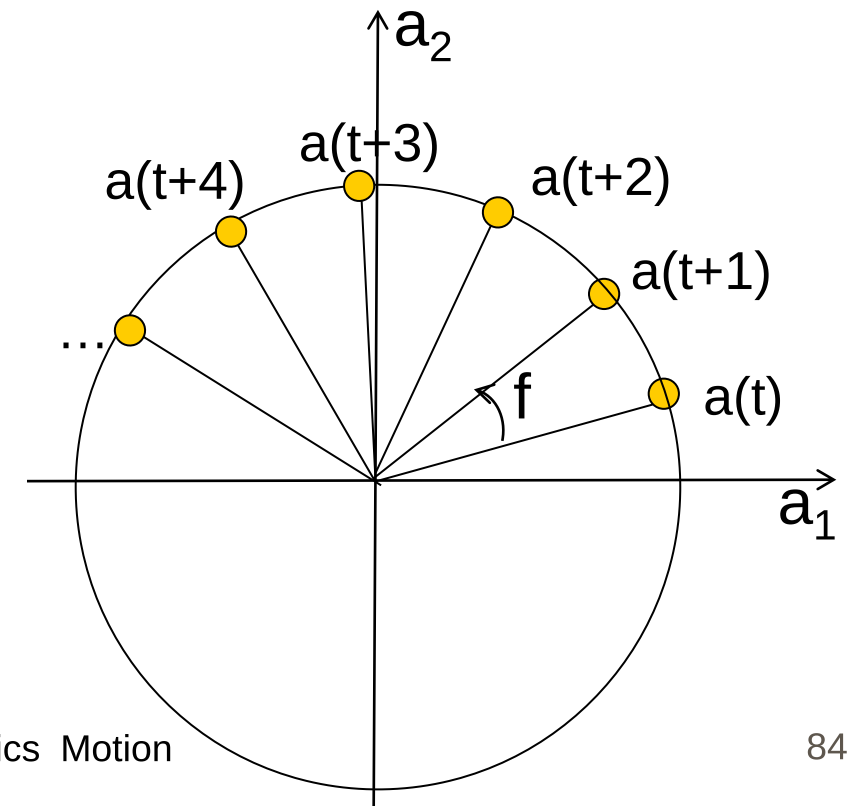
for some φ and $a = 1$:

$$\begin{pmatrix} a_1(t+1) \\ a_2(t+1) \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} \cdot \begin{pmatrix} a_1(t) \\ a_2(t) \end{pmatrix}$$

Neural Network Oscillators: Example

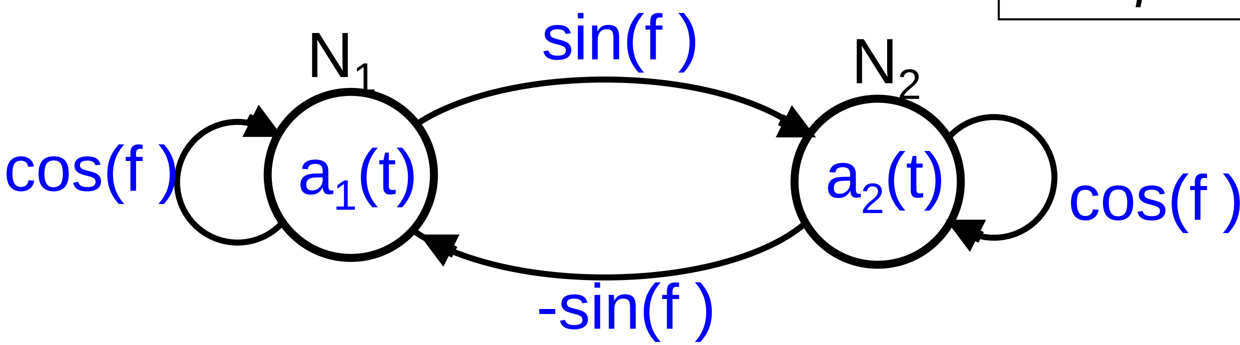
$$\begin{pmatrix} a_1(t+1) \\ a_2(t+1) \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} \cdot \begin{pmatrix} a_1(t) \\ a_2(t) \end{pmatrix}$$

The Matrix W= $\begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix}$ defines rotations of a(t) = $\begin{pmatrix} a_1(t) \\ a_2(t) \end{pmatrix}$
in the a_1-a_2-space,
i.e.
(quasi-)periodic behavior
of $a_1(t)$ and $a_2(t)$



Neural Network Oscillators: SO(2)-Network

"Special Orthogonal Group" SO(2)



with tanh

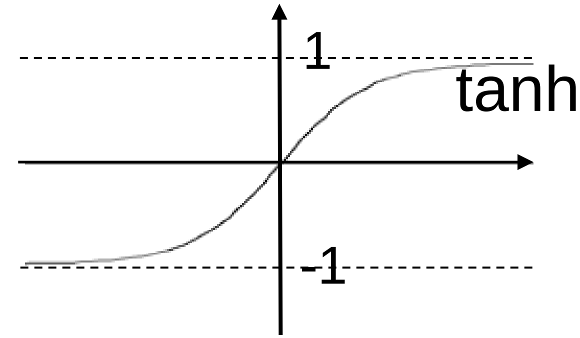
$w_{11} = w_{22} = \cos(f)$, $w_{12} = \sin(f)$, $w_{21} = -\sin(f)$

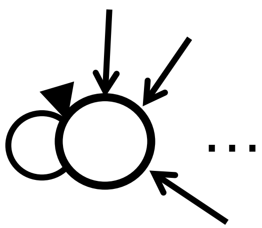
for some a , f :

$$a_1(t+1) = a \tanh(\cos(f)\, a_1(t) - \sin(f)\, a_2(t))$$
$$a_2(t+1) = a \tanh(-\sin(f)\, a_1(t) + \cos(f)\, a_2(t))$$

Neural Networks: tanh

$\tanh(x) = (e^{x} - e^{-x})/(e^{x}+e^{-x})$
$= (e^{2x} - 1)/(e^{2x}+1) = 1 - 2/(e^{2x}+1)$





The activation of a Neuron N_i is computed by

$a_i(t+1) = \tanh\,(S_{j=0,..,n}\; w_{ji}\; \mathbf{a}_j(t))$

where w_{ji} is the weight from Neuron N_j to N_i
(a can be integrated to weights w_{ji})

tanh and a gives more flexibility in the behaviors,
e.g. decreasing/increasing amplitudes (next slide).

Neural Network Oscillators: Example

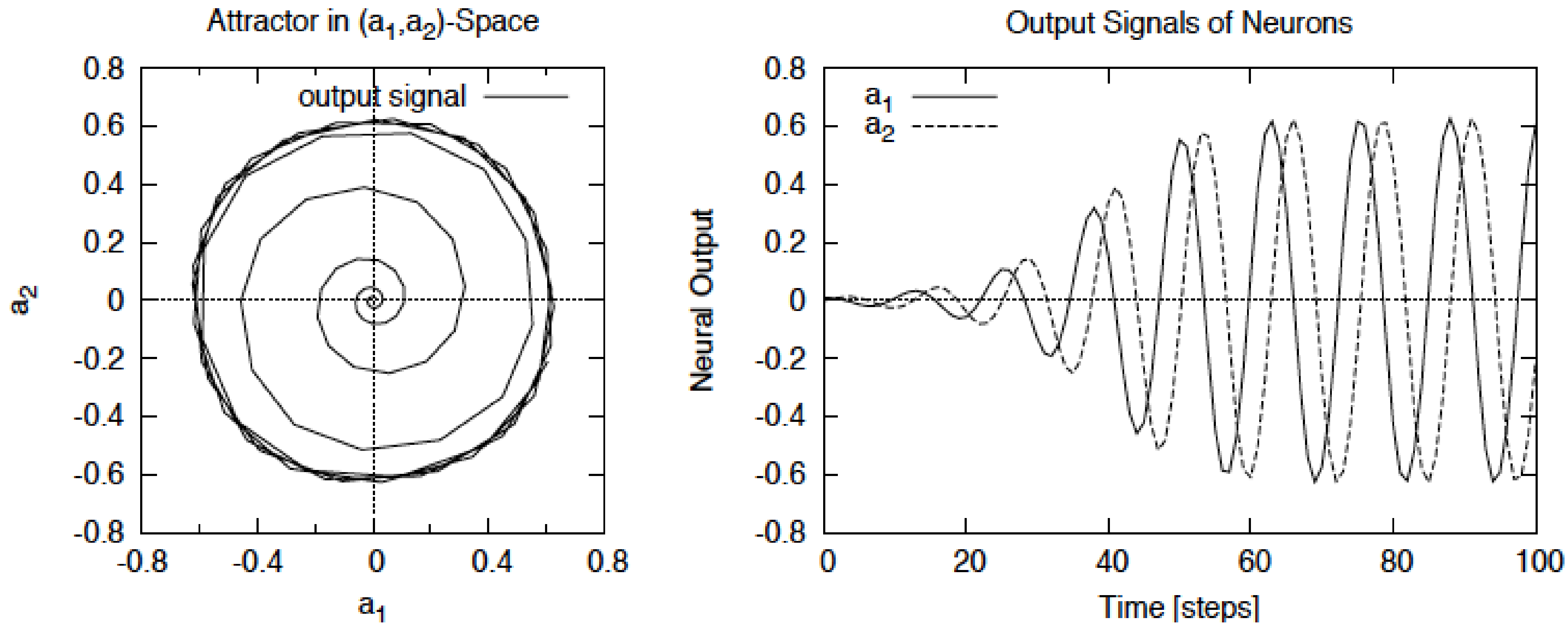
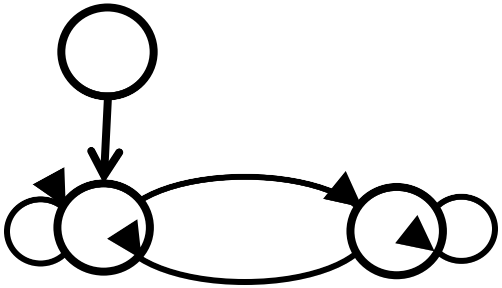


Figure 4.8: Example of a SO(2)-network output: Phase trajectory in (a_1, a_2)-space (left), and output signals of neuron 1 and 2 (right) for $\alpha = 1.1$, $\varphi = 0.5$. Graphs show the initial phase up to reaching the quasi-attractor range within the first 100 time steps. The initial activation was set to $a_1 = 0.01$, $a_2 = 0.0$.

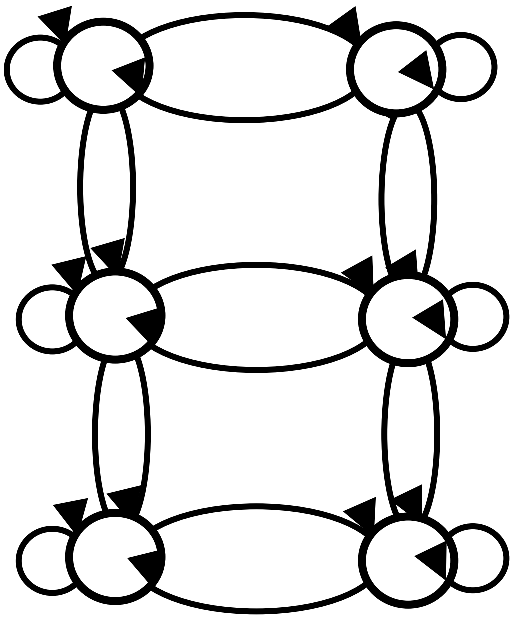
Diploma Thesis Daniel Hein

Neural Network Controllers



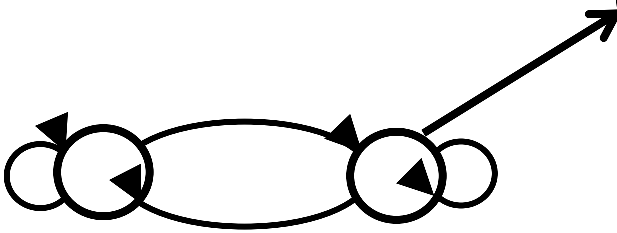
Nets can get inputs from other neurons,
e.g. sensor data which can
- start oscillations
- modify oscillations
- stop oscillations

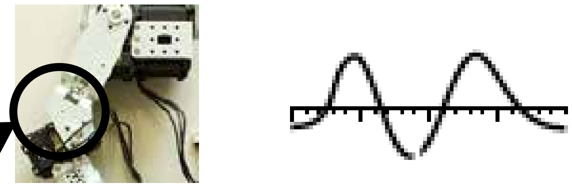
by changing the activations in the net.



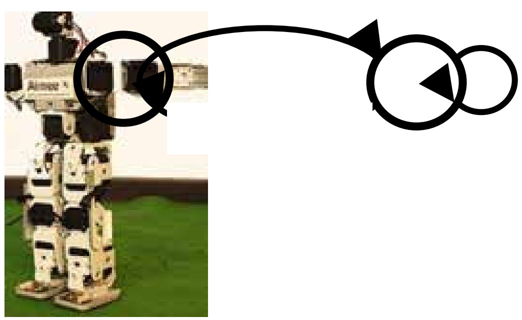
Nets can be connected with other nets,
e.g. for synchronizing pairs of joints

Neural Network Controllers



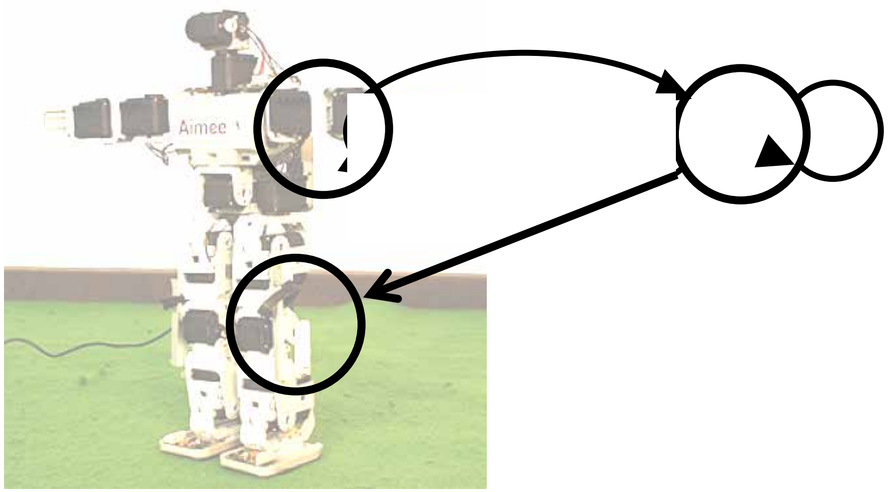


Nets can be connected with
a joint control
for oscillating movements



Motion measuring sensors
(e.g. acceleration sensors)
can be integrated directly

Neural Network Controllers



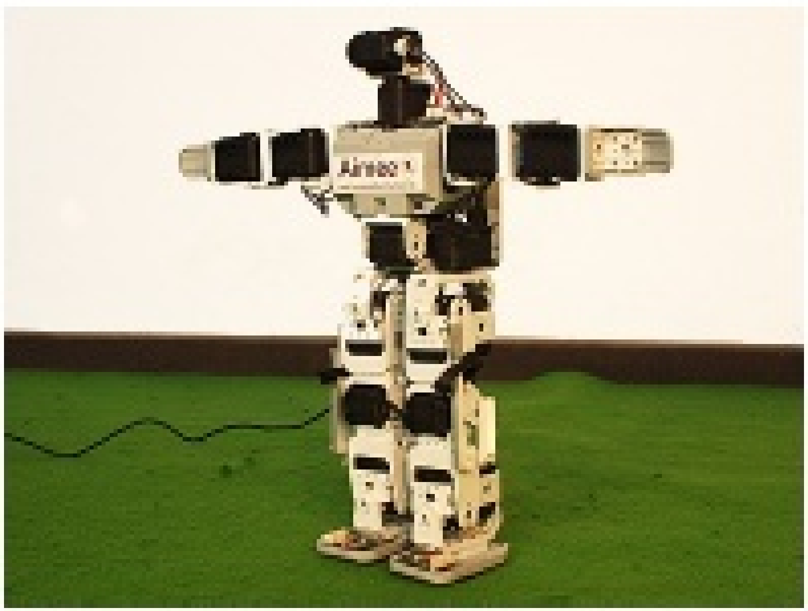
If weights are adjusted accordingly,
the acceleration sensors (in the shoulders)
and the motor control neurons
build an oscillating system

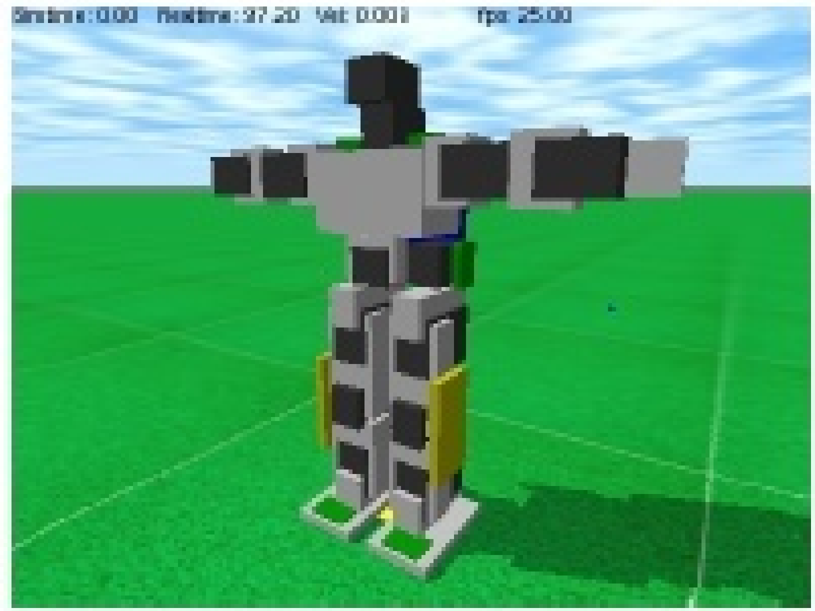
Case Study Simloid

Diploma Thesis
Daniel Hein

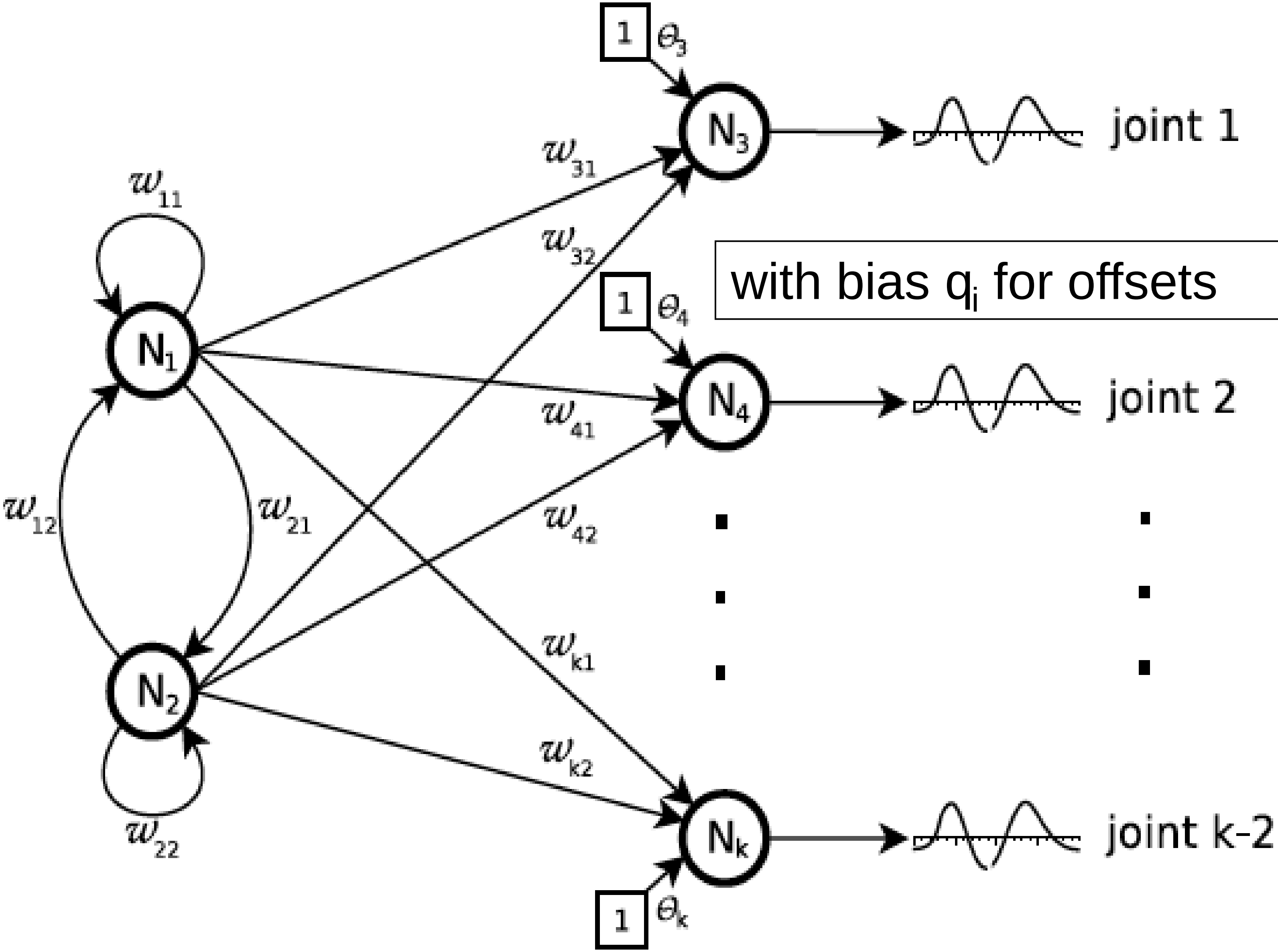
Neural Net Controller for Simloid

(= simulated robot Bioloid from Robotis)





Simloid: Neural Net controller



with bias q_i for offsets

Simloid: Evolution of Neural Net controller

Optimal weights w_{ij} of the Net were determined by evolution:

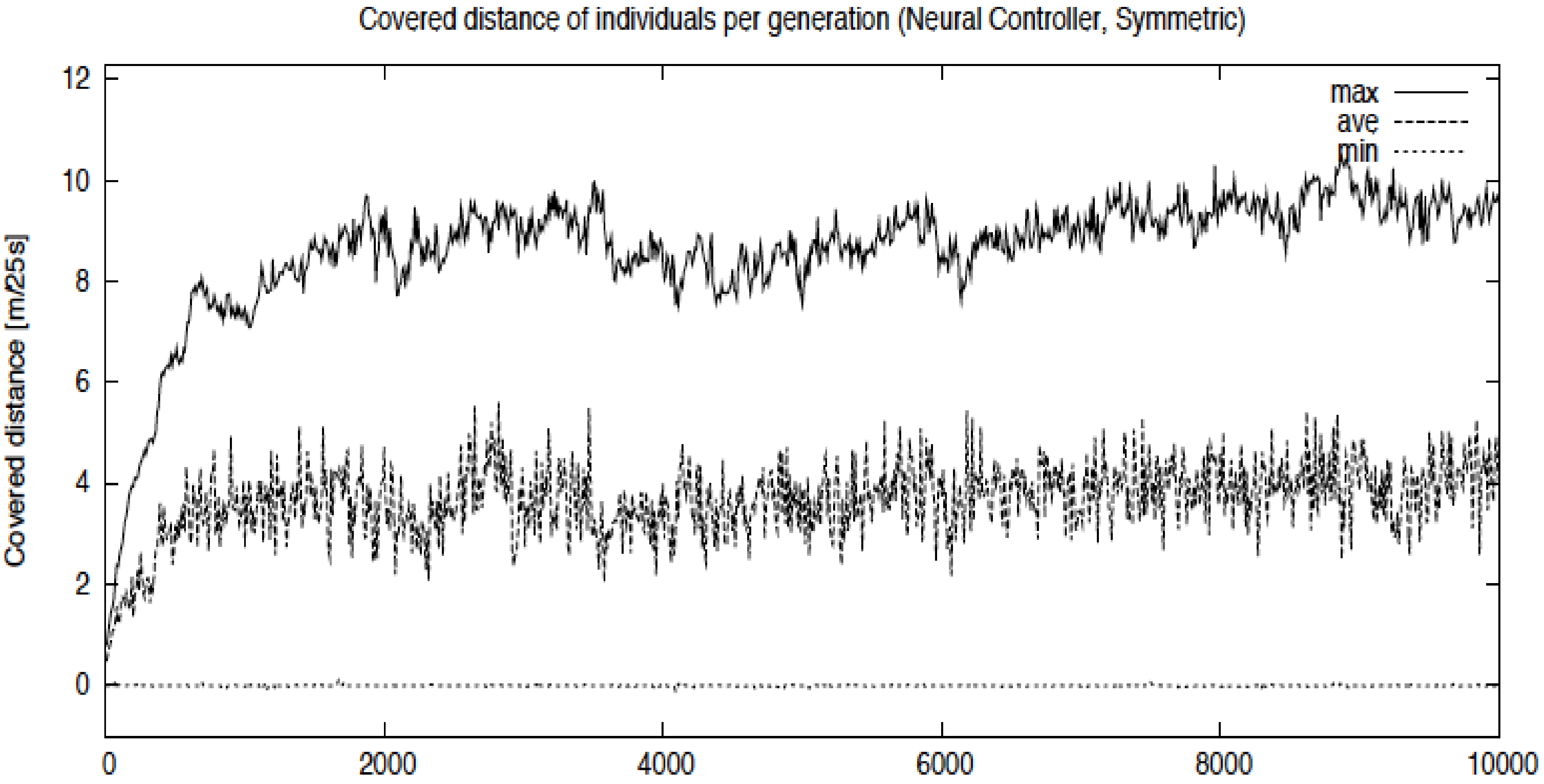
Parameters:

- 57 weights for 19 joints (6 per leg, 3 per arm, 1 waist)
  + 4 weights for oscillator
- Reduction by left/right symmetry assumption: 34 parameters

Individuals: $(p_1,\ldots,p_{34})$ with ranges (-4,4)

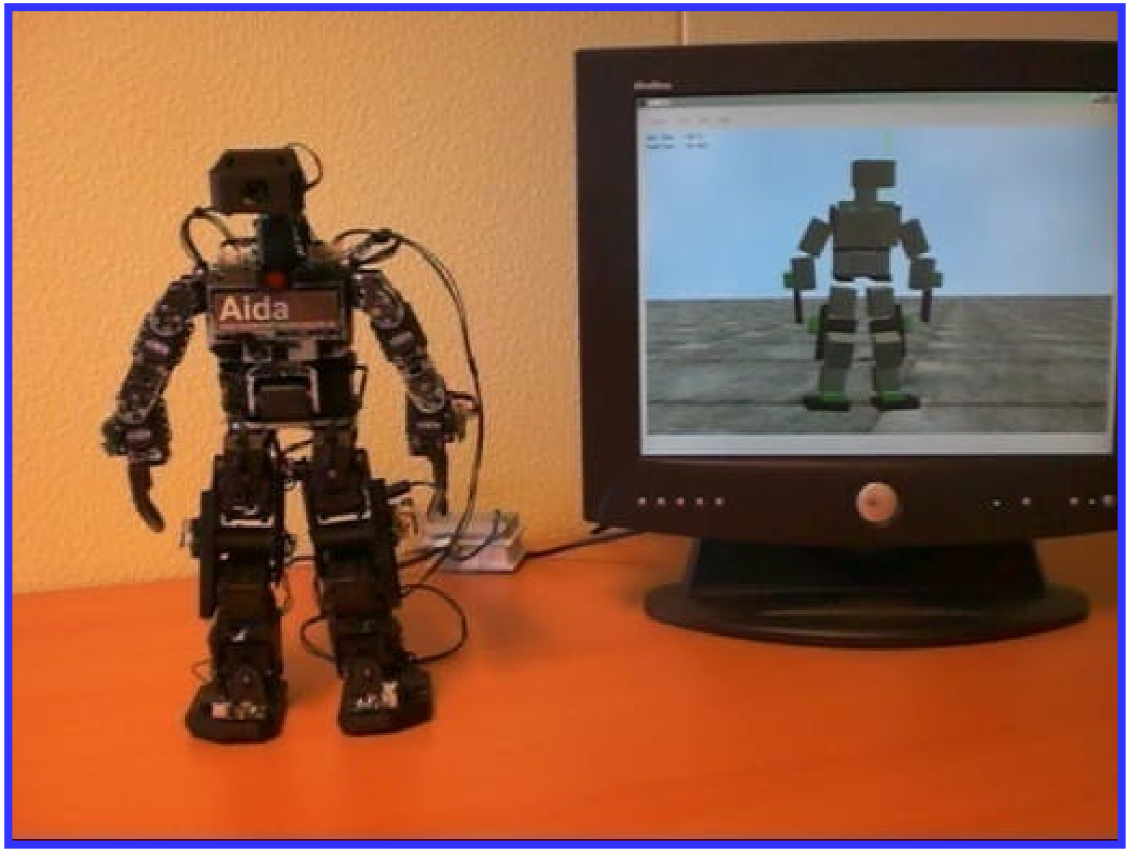
Fitness: Distance covered in a given constant time

Simloid: Evolution of Neural Net controller



Evolved Neural Nets for Bioloid (A-Series)

Evolved Neural Nets for Bioloid (A-Series)



(with another simulator from ALEAR project)