Imitation Learning by Behavioral Cloning

Jan Peters
Gerhard Neumann

Purpose of this Lecture

Learning From Demonstrations

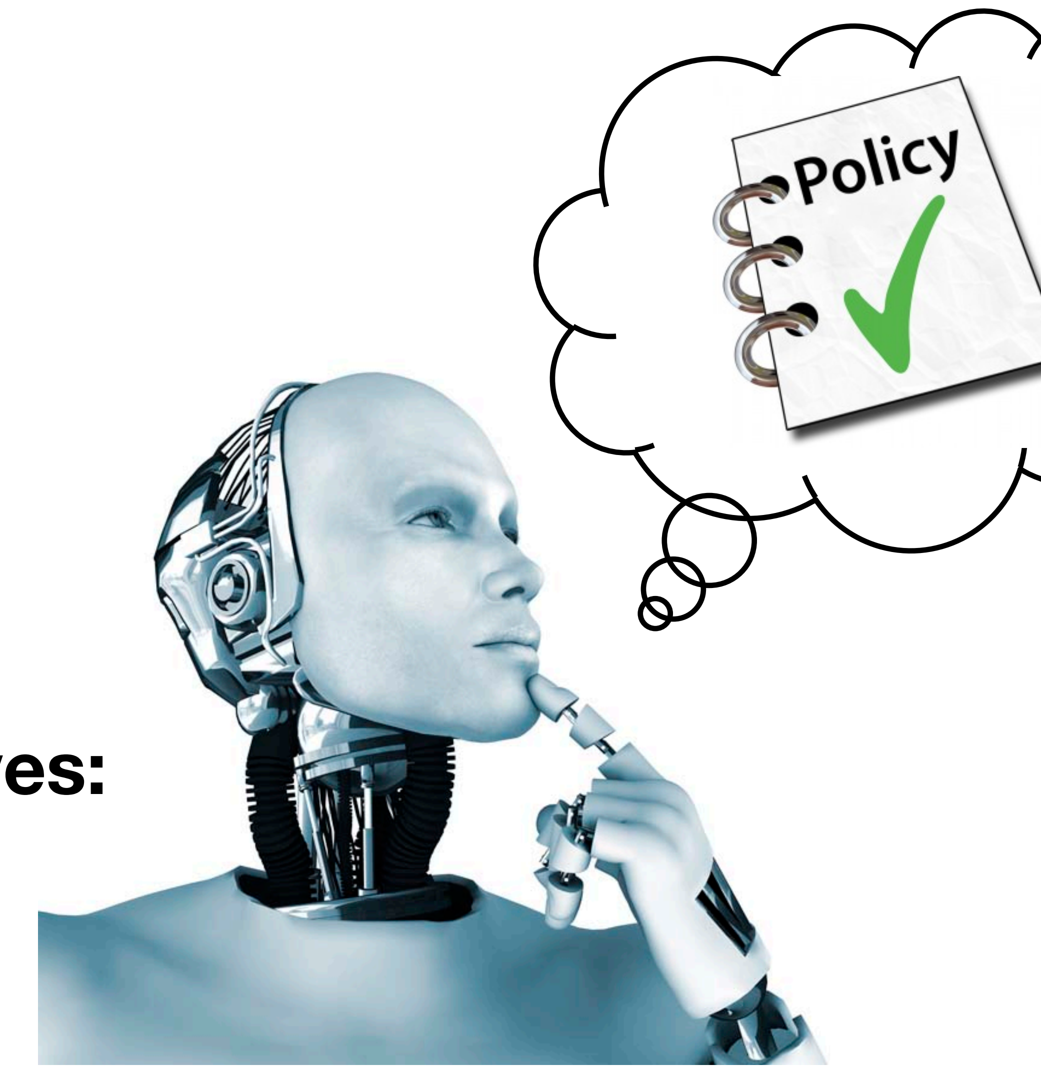
- How can we teach a robot without programming?

Policy Representations

- Show you important characteristics of commonly used policies
- State space vs. trajectory space view

Introduce the concept of Movement Primitives:

- How can we incorporate modularity?
- Data-driven acquisition of movements



Outline:

1. **Learning Policies from Demonstrations by Supervised Learning**
2. **Policy Representations**
    - State-space representations
    - Trajectory-based representations
3. **Imitation Learning with Movement Primitives**
    - Dynamic Movement Primitives
    - Probabilistic Movement Primitives
    - Beyond a single primitive

Why do we need imitation?

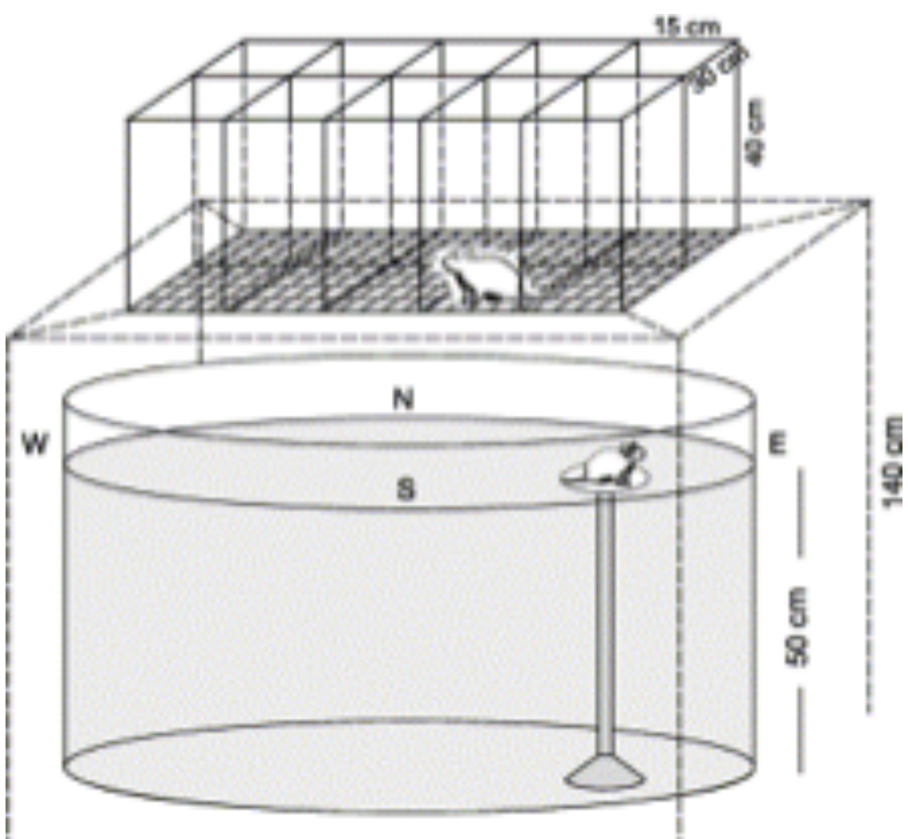
- Very successful strategy for humans

- Learning controllers from scratch by reinforcement learning is often very time consuming or even too difficult

- the search space may frequently be way too large for the agent to explore it in its lifetime

- an expert takes many years to optimize his policy and a robot could avoid his expensive training by cloning his policy

Already rats can imitate!



- Student rats observe companion actor rats performing different spatial tasks differing according to the experimental requirements.
- After the observational training, surgical ablation to block any further learning in the student rat.
- The observer rats displayed exploration abilities that closely matched the previously observed behaviors.

Legio et al, Brain Res. Protocols, 2003
Heyes, Trends in Cog. Sciences, 2001

... and dolphins...

Infants have Imitation build in!

- Infants as young as 42 minutes old copy several facial actions (e.g., Meltzoff & Moore, 1977).

How to Demonstrate?

- **Teleoperation**: Use a joystick to train an RC car, a mouse for training a Quake III player, the steering wheel of the Navlab, data gloves, etc.
- **Kinesthetic Teach-In**: Take the robot by the hand like a tennis teacher teaches a tennis student.
- **Vision**: Video-based tracking of human beings.
- **Marker-based Tracking**: With markers and a basic skeleton, very precise human data can be obtained.
- **Sensuits**: Suits with encoders and accelerometers attached to human beings.

Basic Idea of Behavioral Cloning

- Behavioral Cloning is the simplest form of learning from demonstration

- An expert is available and supplies data traces:

$$\boldsymbol{s}_1 \rightarrow \boldsymbol{u}_1 \rightarrow \boldsymbol{s}_2 \rightarrow \boldsymbol{u}_2 \rightarrow \boldsymbol{s}_3 \rightarrow \boldsymbol{u}_3 \rightarrow$$

- In our case, often $\boldsymbol{s} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}$

- The student infers a policy from these data traces, i.e.,

$$\boldsymbol{u} = \pi(\boldsymbol{s}) \text{ or } \boldsymbol{u} \sim \pi(\boldsymbol{u}|\boldsymbol{s})$$

- In principle, this can be treated as a supervised learning problem.

Direct Behavioral Cloning

Standard ML techniques can simply be applied to the data set

$$\mathcal{D} = \{\boldsymbol{s}_i, \boldsymbol{u}_i\}$$

to extract a **policy**

$$\boldsymbol{u} = \pi(\boldsymbol{s}) = \boldsymbol{\phi}^T(\boldsymbol{s})\boldsymbol{\theta}, \quad \text{or}$$
$$\boldsymbol{u} \sim \pi(\boldsymbol{u}|\boldsymbol{s}) = \mathcal{N}\left(\boldsymbol{u}|\boldsymbol{\mu}(\boldsymbol{s}), \boldsymbol{\Sigma}(\boldsymbol{s})\right)$$

… the problem frequently boils down to a **regression problem**.

⇨ **The clean-up effect:** due to regularization, the noise in the demonstration is no longer exhibited by the reproduction and, hence, the clone often surpasses the quality of the expert.

Outline:

1. **Learning Policies from Demonstrations by Supervised Learning**
2. **Policy Representations**
    - State-space representations
    - Trajectory-based representations
3. **Imitation Learning with Movement Primitives**
    - Dynamic Movement Primitives
    - Probabilistic Movement Primitives
    - Beyond a single primitive

Why do we use parametric policies?

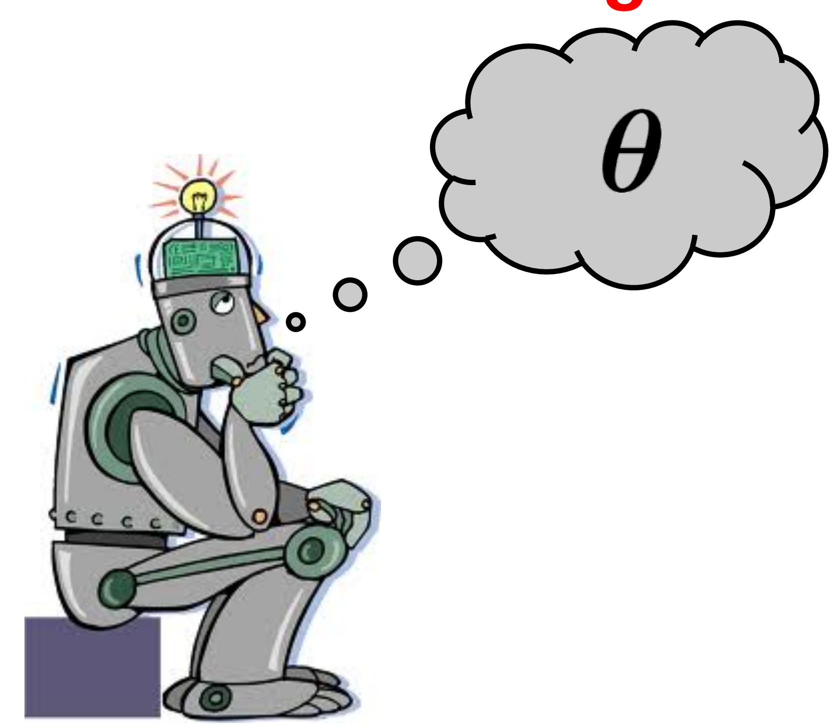
A parametric policy is a conditional probability distribution $\pi(\boldsymbol{u}|\boldsymbol{s};\boldsymbol{\theta})$ **that chooses the actions** $\boldsymbol{u}$ **depending on the state** $\boldsymbol{s}$ **of the robot**

- Parametric policy naturally incorporates continuous actions
- **Estimate from demonstration / imitation learning**

  ⇨ Generalize to unseen situations

- **Search for improved parameters / reinforcement learning**

  ⇨ Autonomous self improvement!



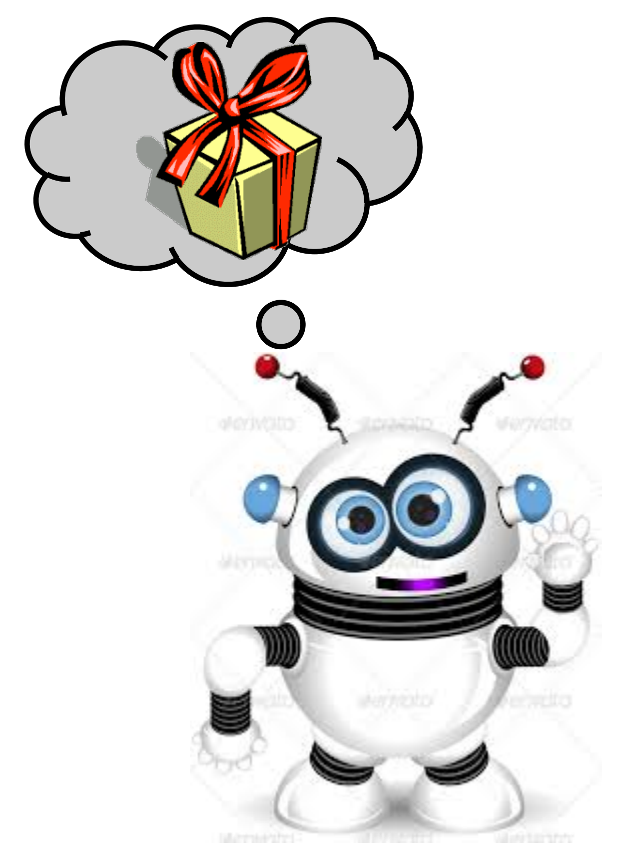
What are desirable properties?

- **Compactness:** Low number of parameters
- **Learn-ability**: Easy to learn from demonstration and by reinforcement learning
- **Stochasticity:** Can encode exploration and variability
- **Optimality:** Can encode optimal behavior?
- **Scalability:** Can be used for a high number of DoFs?
- **Modularity:**

    Adaptability: Reusable for new situations?

    Co-activation and Blending of movements

- **Useable** for **stroke-based** and **rhythmic movements**



Stochastic vs. deterministic policies

Why use a stochastic policy?

Used for exploration in reinforcement learning (later)

Can also capture variability of movements

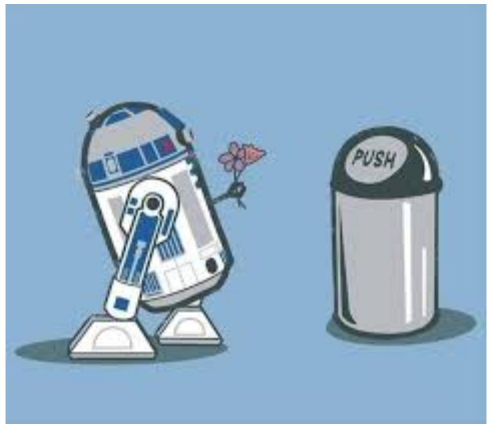
Exploration models:

No exploration: $\boldsymbol{u} = \pi(\boldsymbol{s}) = f_{\boldsymbol{w}}(\boldsymbol{s}), \quad \boldsymbol{\theta} = \boldsymbol{w}$

Uncorrelated Exploration: $\pi(\boldsymbol{u}|\boldsymbol{s};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{u}|f_{\boldsymbol{w}}(\boldsymbol{s}), \sigma^2 \boldsymbol{I}), \quad \boldsymbol{\theta} = \{\boldsymbol{w}, \sigma^2\}$

Correlated Exploration: $\pi(\boldsymbol{u}|\boldsymbol{s};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{u}|f_{\boldsymbol{w}}(\boldsymbol{s}), \boldsymbol{\Sigma}), \quad \boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{\Sigma}\}$

➡ We also have to learn the variances of the linear models



Exploration might also hurt

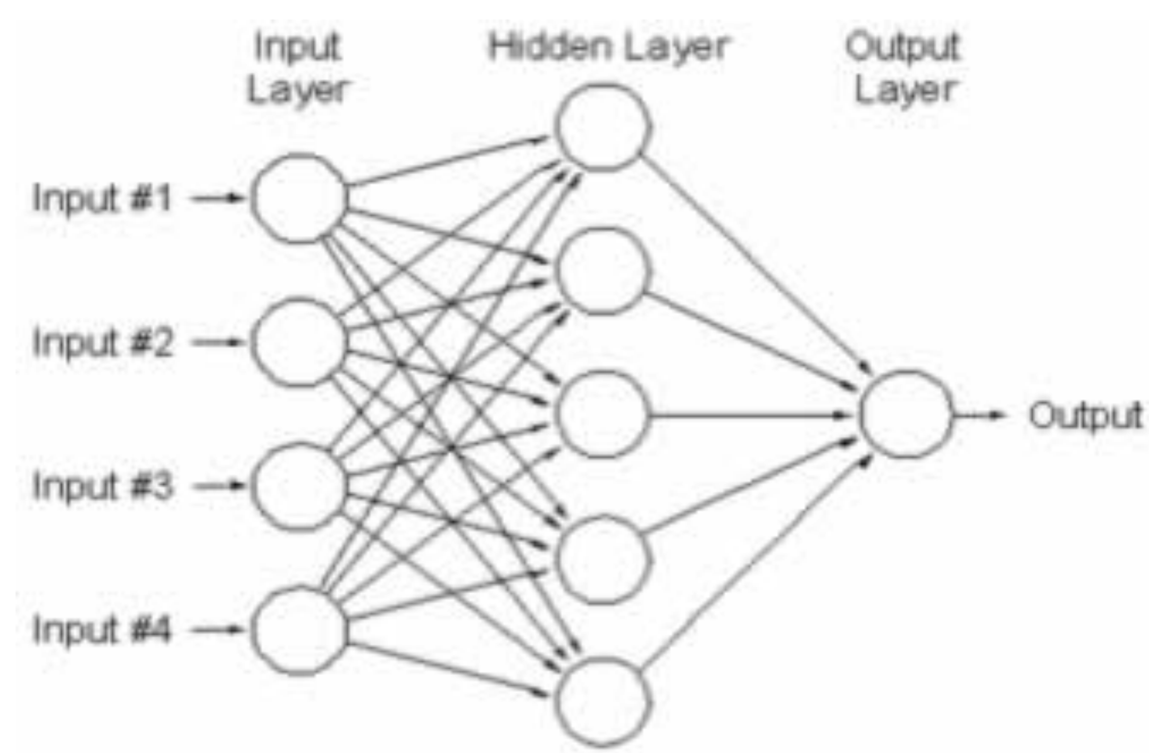
State space vs. trajectory space representations

State space representation: $\pi(\boldsymbol{u}|\boldsymbol{s};\boldsymbol{\theta})$

- Policy depends on the state and on the parameters
- Represents a globally valid policy
- Complex non-linear representations are needed

Examples:

- Neural Networks
- RBF Networks
- Gaussian Processes
- Locally Weighted Regression Models



State space representations

Linear controllers: $f_{\boldsymbol{w}}(\boldsymbol{s}) = \phi^T(\boldsymbol{s})\boldsymbol{w}$

Most simple case: linear PD controller

$$\phi(\boldsymbol{s}) = \begin{bmatrix} 1 \\ \boldsymbol{s} \end{bmatrix}, \quad f_{\boldsymbol{w}}(\boldsymbol{s}) = \boldsymbol{K}\boldsymbol{s} + \boldsymbol{k}$$

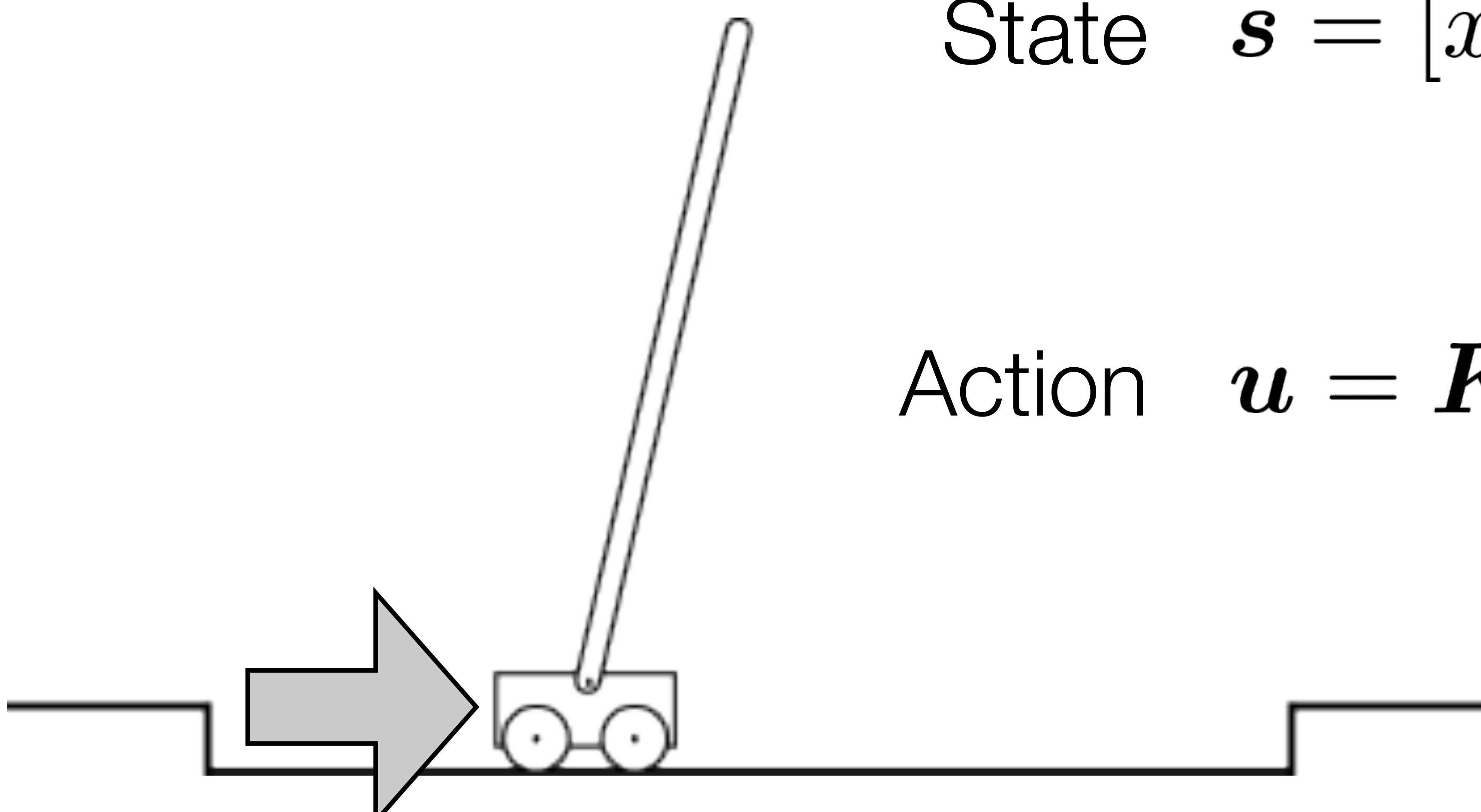
[-] Good feature representation needs to be known

[+] Very compact representation (low number of parameters)

[+] Easy to learn (linear regression)

Pole Balancing

Widrow and Smith (1964) used supervised learning to acquire a pole balancing policy.



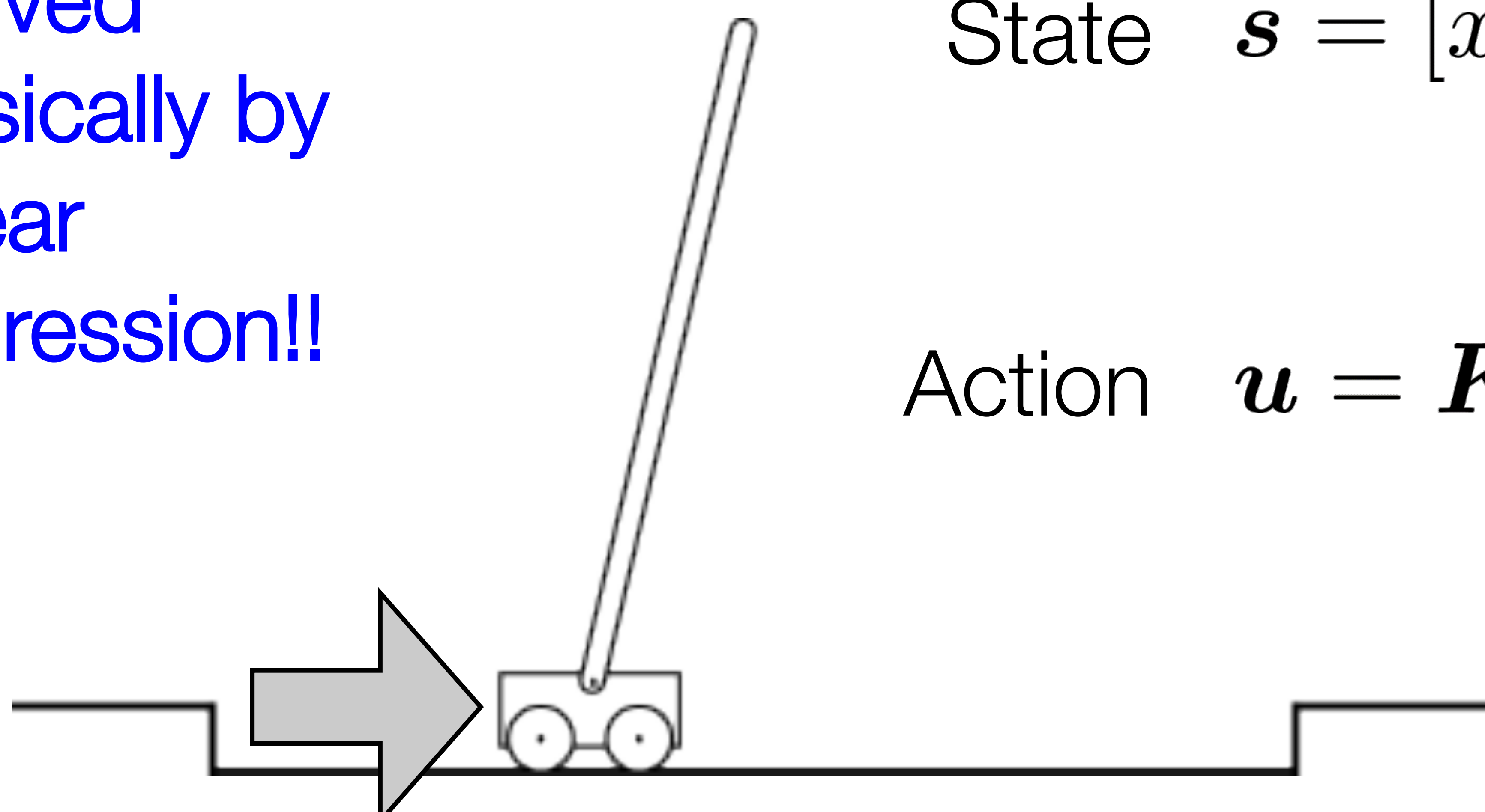
State $\boldsymbol{s} = [x, \dot{x}, \alpha, \dot{\alpha}]$

Action $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{s}$

Pole Balancing

Widrow and Smith (1964) used supervised learning to acquire a pole balancing policy.

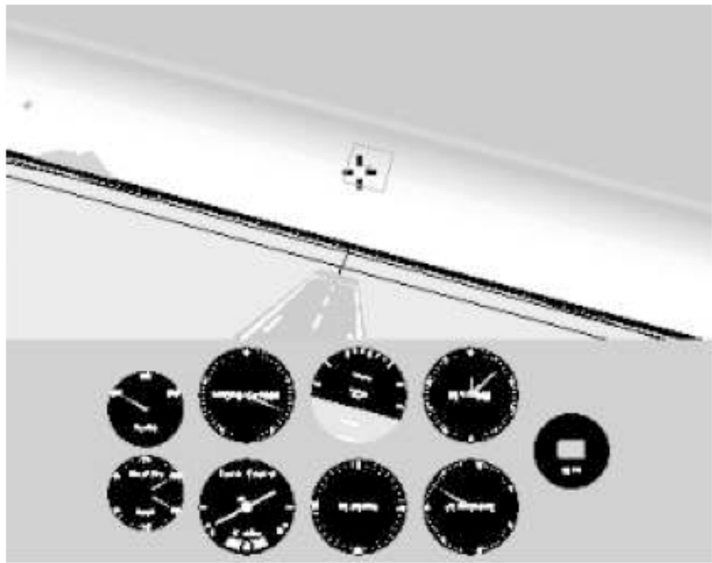
Solved basically by linear regression!!



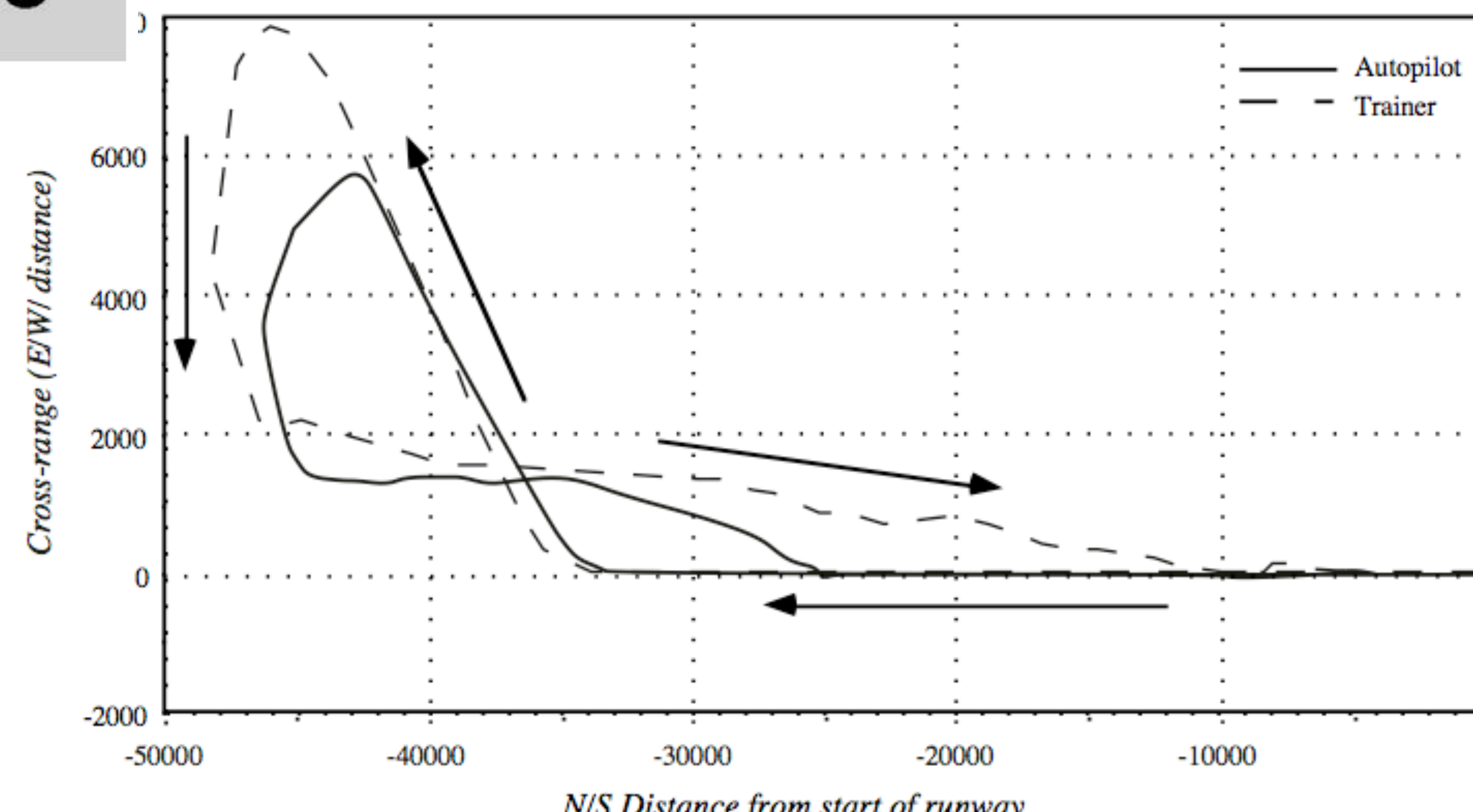
State $\boldsymbol{s} = [x, \dot{x}, \alpha, \dot{\alpha}]$

Action $\boldsymbol{u} = \boldsymbol{K}\boldsymbol{s}$

Sammut's Cessna Pilot



(Sammut et al., 1992)



Non-linear state space representations

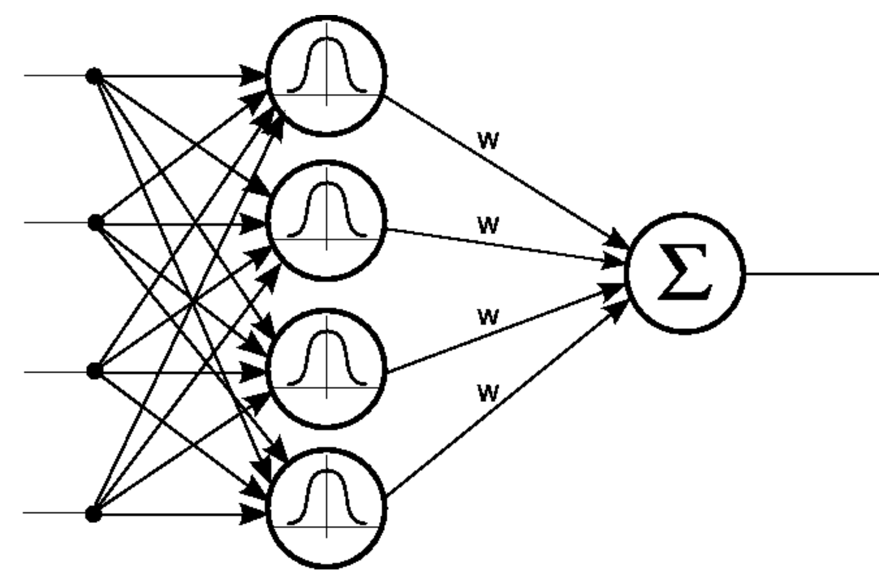
Radial Basis Function (RBF) networks:

$$f_{\boldsymbol{w}}(\boldsymbol{s}) = \phi^T(\boldsymbol{s})\boldsymbol{\beta}, \quad \phi_i(\boldsymbol{s}) = \exp\left(-0.5\sum_{j=1}^{D}(s_j - \mu_{ij})^2/h_{ij}\right)$$

$$\boldsymbol{w} = \{\boldsymbol{\beta}, \boldsymbol{\mu}_{1:K}, \boldsymbol{h}_{1:K}\}$$

Normalized RBF:

$$f_{\boldsymbol{w}}(\boldsymbol{s}) = \frac{\sum_{i=1}^{K}\phi_i(\boldsymbol{s})\beta_i}{\sum_{i=1}^{K}\phi_i(\boldsymbol{s})}$$



[-] A high number of parameters

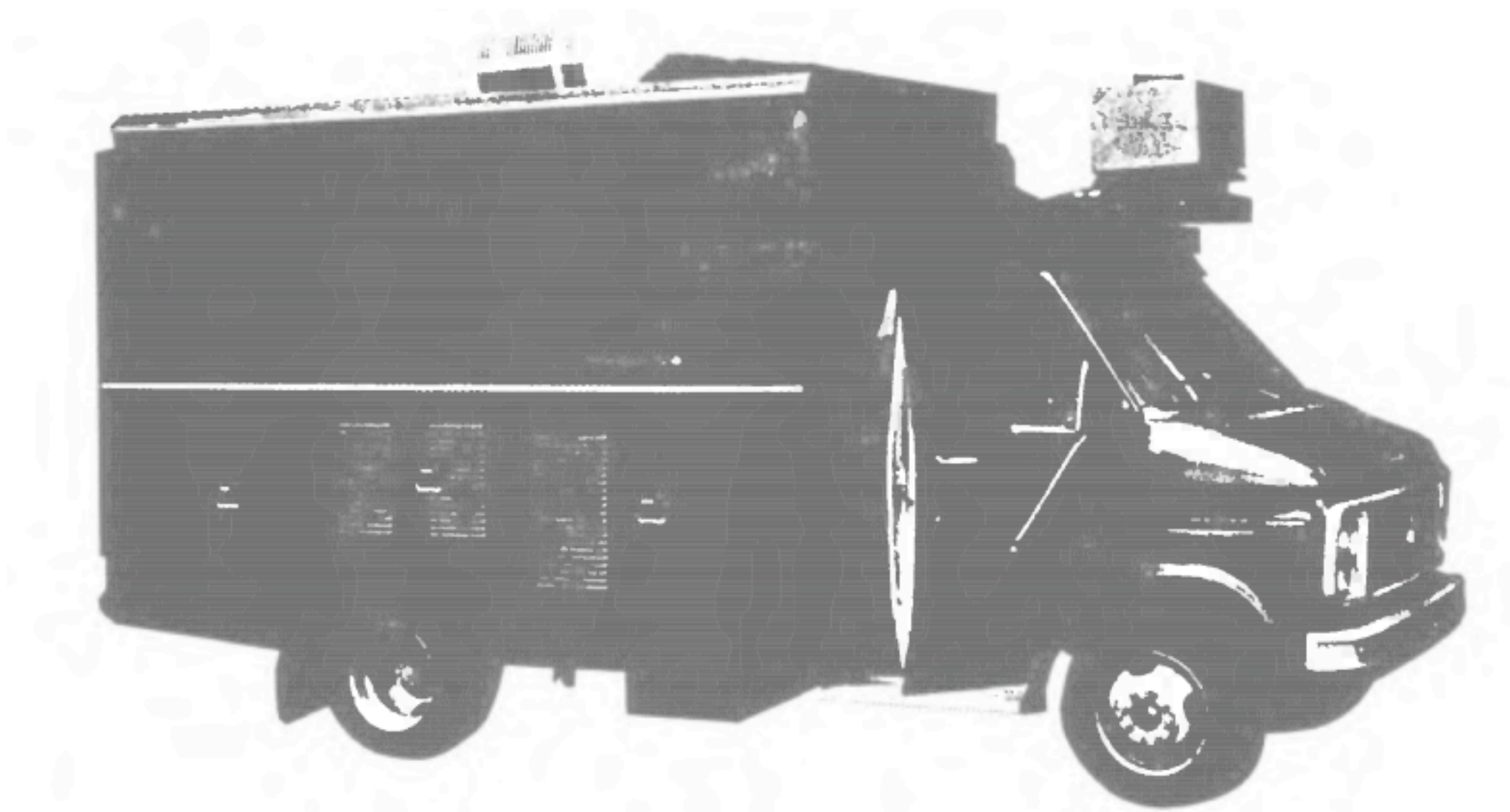
[-] Non-convex optimization

[-] Hard to scale ⇨ curse of dimensionality

[+] Automatic feature construction

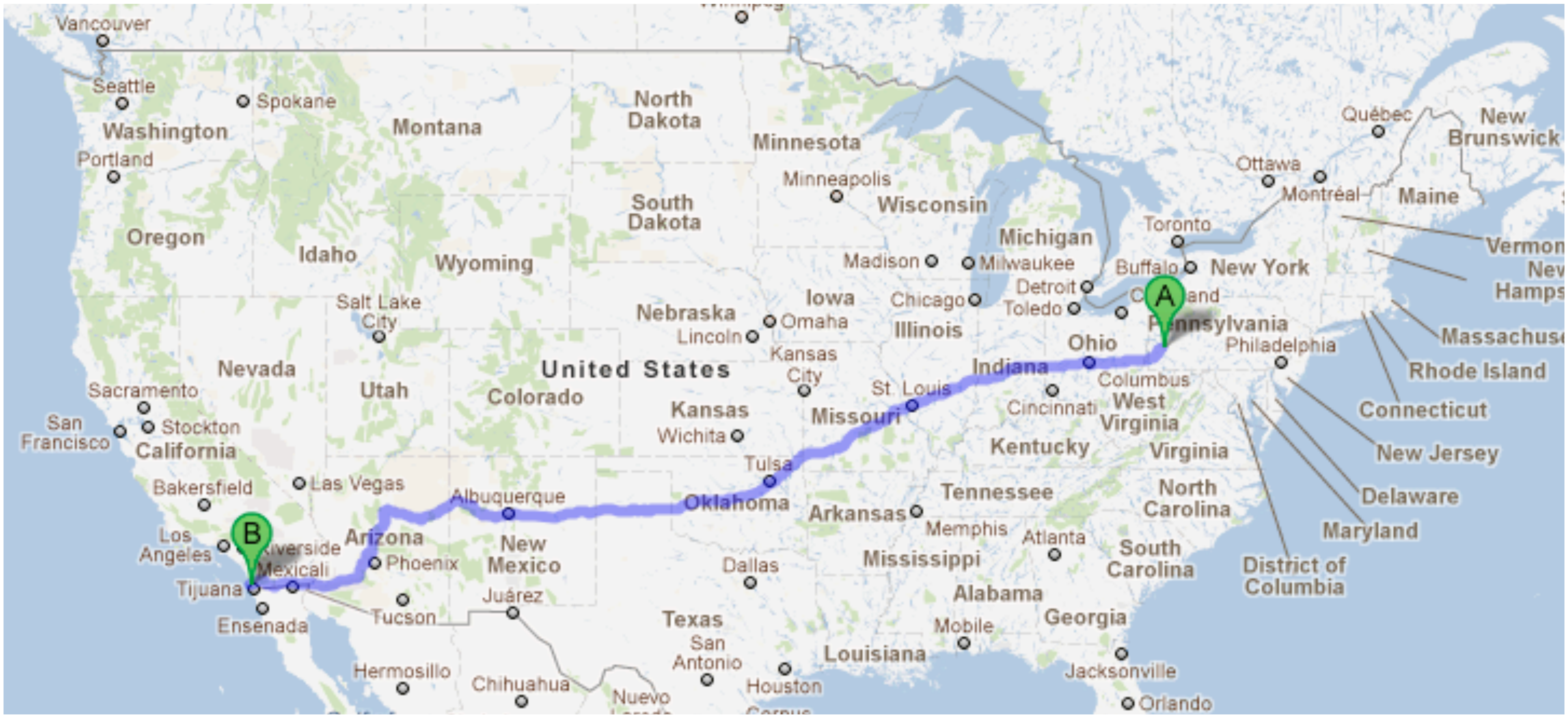
Alternatives: Gaussian Mixture Models (GMM), Neural Networks

ALVINN & Navlab in 1989-1995!



Pomerlau (1989-1995)

No-Hands-Across-America



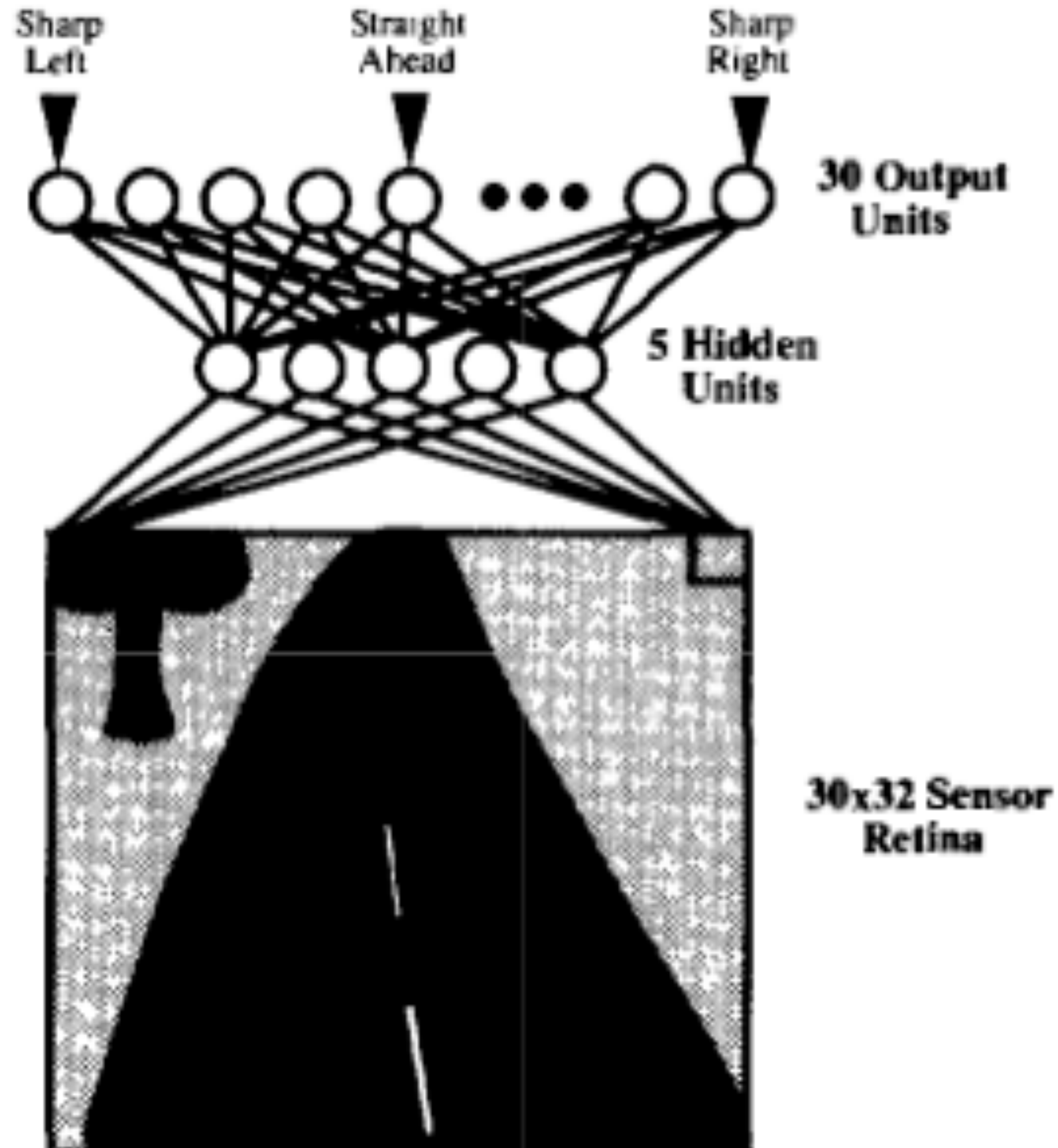
ALVINN allowed the Navlab vehicle of CMU’s robotics institute to drive 2796km autonomously as part of their ’No-Hands-Across-America’ Tour in 1995.

States and Actions

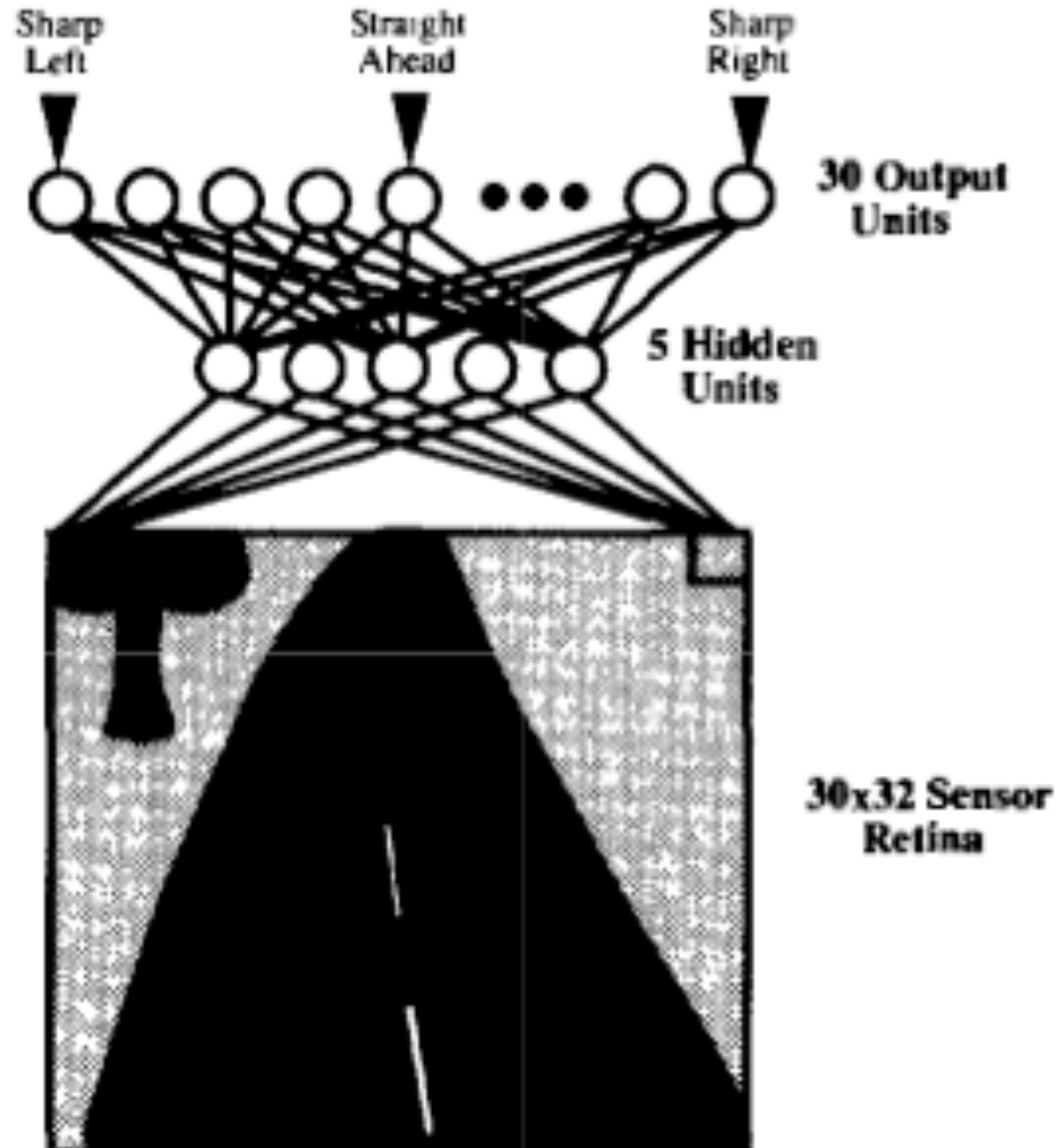
State:
Camera
Image

Action:
Steering
Wheel,
Brakes, Gas

Function Approximator:
A Two-Layered
Neural Network



Function Approximator:
A Two-Layered
Neural Network



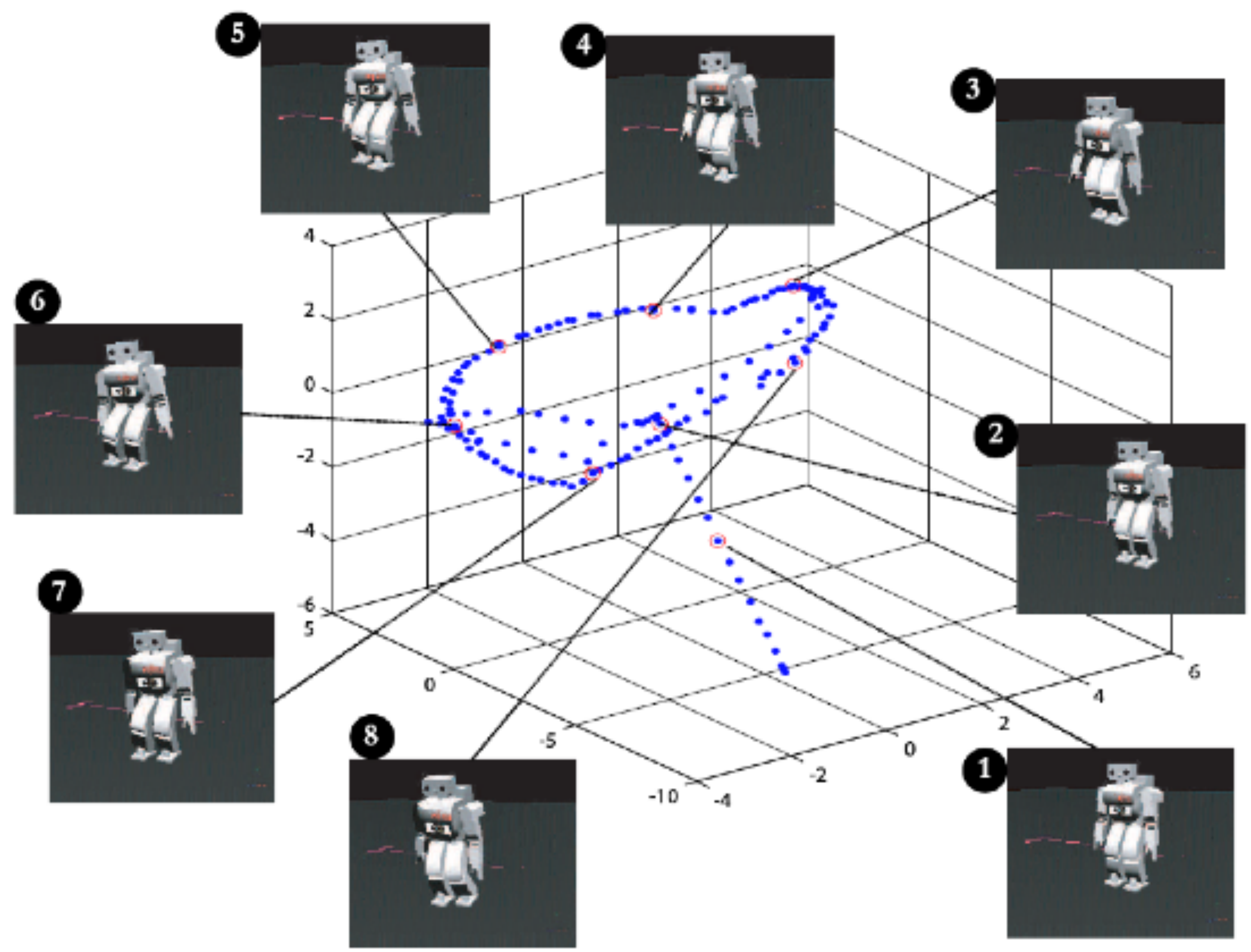
JUST (nonlinear) REGRESSION!

Video from ALVIN

State space representations

Represent controller in a **low-dimensional manifold**

⇨ E.g. Eigenpostures for walking

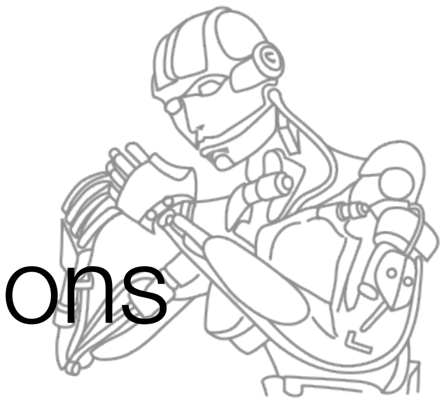


(Grimes, Rashid, & Rao, NIPS 2007)

Doubts on Direct Behavioral Cloning

- It becomes brittle for **larger state-spaces** unless you have a task-appropriate representation.
- Frequently leads to **catastrophic failures** if the controller has not been trained in this area of the state-action space (Sammut, 2010) or if there have been small changes in the system (Camacho & Michie, 1995).
- Reproduction of **single human teachers always works best** (Camacho & Michie, 1995).
- There is no **guarantee that the reproduction is meaningful**, nor an interpretation of behavior.
- The data is treated as **if it was i.i.d**.
- We do not know whether we can also reproduce **the long-term behavior**!
- Only learning of **individual motions is „easy“.**

State space vs. trajectory space representations



Time-dependent representation: $\pi(\boldsymbol{u}|\boldsymbol{s}, t; \boldsymbol{\theta})$

Policy also depends on time, e.g., follow a specific trajectory

For the same time step, the robot is often in similar states

Simple local models are often sufficient!

Time-dependend representations

For example: **Time-dependent linear feedback controllers**

$$f_{\boldsymbol{w}}(\boldsymbol{s}, t) = \sum_{i=1}^{K} \phi_i(t)(\boldsymbol{K}_i \boldsymbol{s} + \boldsymbol{k}_i)$$

- Time dependent basis functions, e.g., normalized RBF functions
- Scales quadratically with # DoF D: $KD/2(D+1)$
- Equivalent to PD-trajectory tracking with time-varying controller gains
  - Variable stiffness controllers
- Locally **optimal representation** (why we will see in the next lectures!)

Trajectory-based representations

Trajectory Generators:

Directly learn desired trajectory $\boldsymbol{q}^*(t;\boldsymbol{w})$

Use feedback controller to follow trajectory

$$\pi(\boldsymbol{s}, t; \boldsymbol{w}) = \boldsymbol{K}_P(\boldsymbol{q}^*(t;\boldsymbol{w}) - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}^*(t;\boldsymbol{w}) - \dot{\boldsymbol{q}})$$

where typically $\boldsymbol{K}_P$ and $\boldsymbol{K}_D$ are hand tuned diagonal matrices

Possible Trajectory Representations:

- Splines $\boldsymbol{q}^*(t;\boldsymbol{w}) = \sum_{i=0}^{5} w_i t^i$
- Linear basis function models (RBFs) $\boldsymbol{q}^*(t;\boldsymbol{w}) = \boldsymbol{\phi}^T(t)\boldsymbol{w}$
- Dynamical Systems

Outline:

1. **Learning Policies from Demonstrations by Supervised Learning**
2. **Policy Representations**
   - State-space representations
   - Trajectory-based representations
3. **Imitation Learning with Movement Primitives**
   - Dynamic Movement Primitives
   - Probabilistic Movement Primitives
   - Beyond a single primitive

Movement Primitives

What are movement primitives?

- Movement primitives are a compact representation of a movement
- Often represented as parametrized trajectory generator

$$\boldsymbol{\tau} = f(\boldsymbol{w}), \quad \boldsymbol{w} \ldots \text{ parameters of the primitive}$$

Imitation Learning with trajectory generators

- By learning the desired trajectory, we also learn the desired long term behavior!
- However, we still have to learn how to follow this trajectory
- If we do not have good trajectory tracking controllers, it does not work

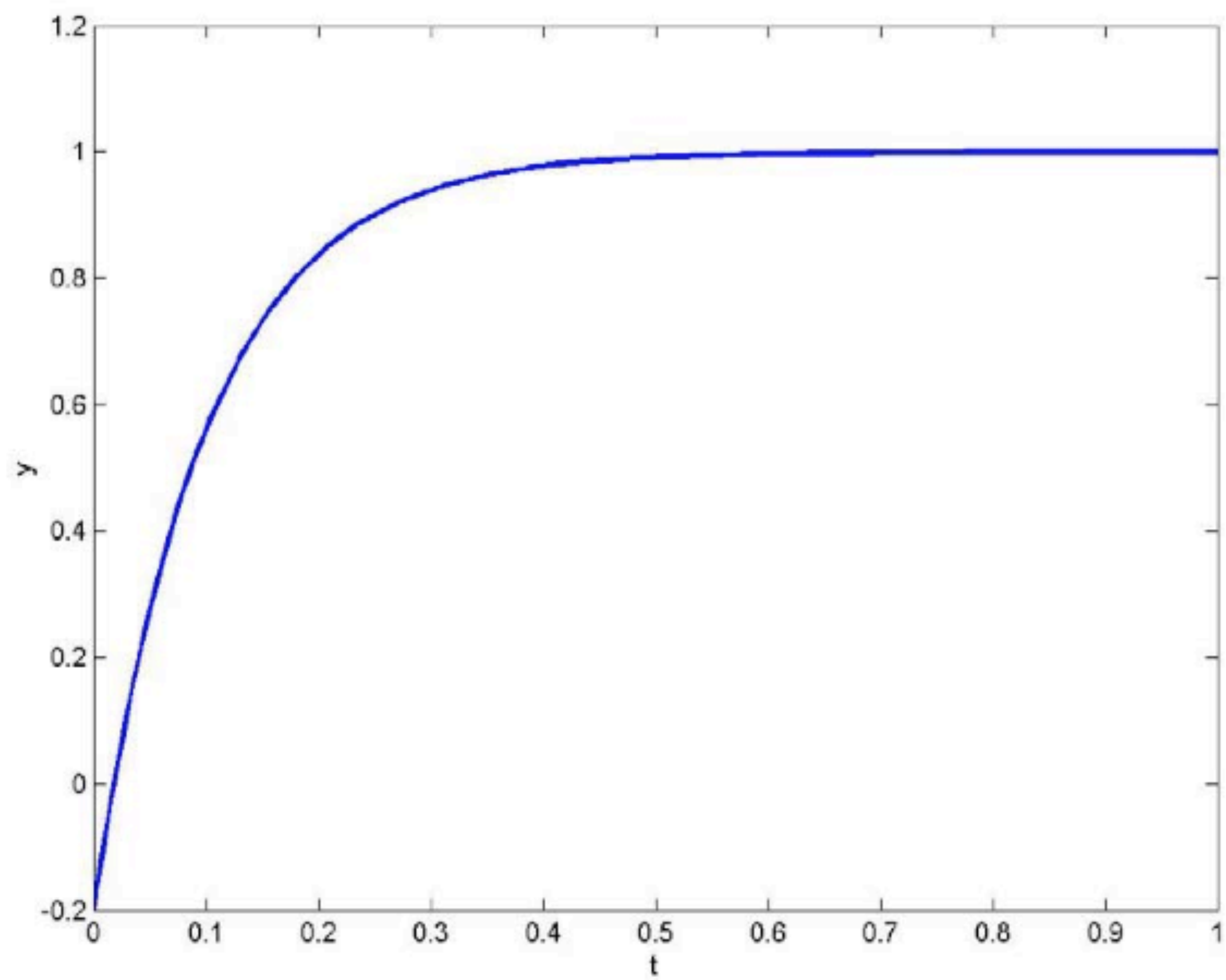
Dynamical systems as Trajectory Generators

Dynamical systems can be used to represent trajectories

➡ Integrating the dynamical system results in a trajectory

$$\dot{y} = \alpha(c - y)$$

- What movement can a differential equation encode?
- **Example:** First order linear dynamical system:



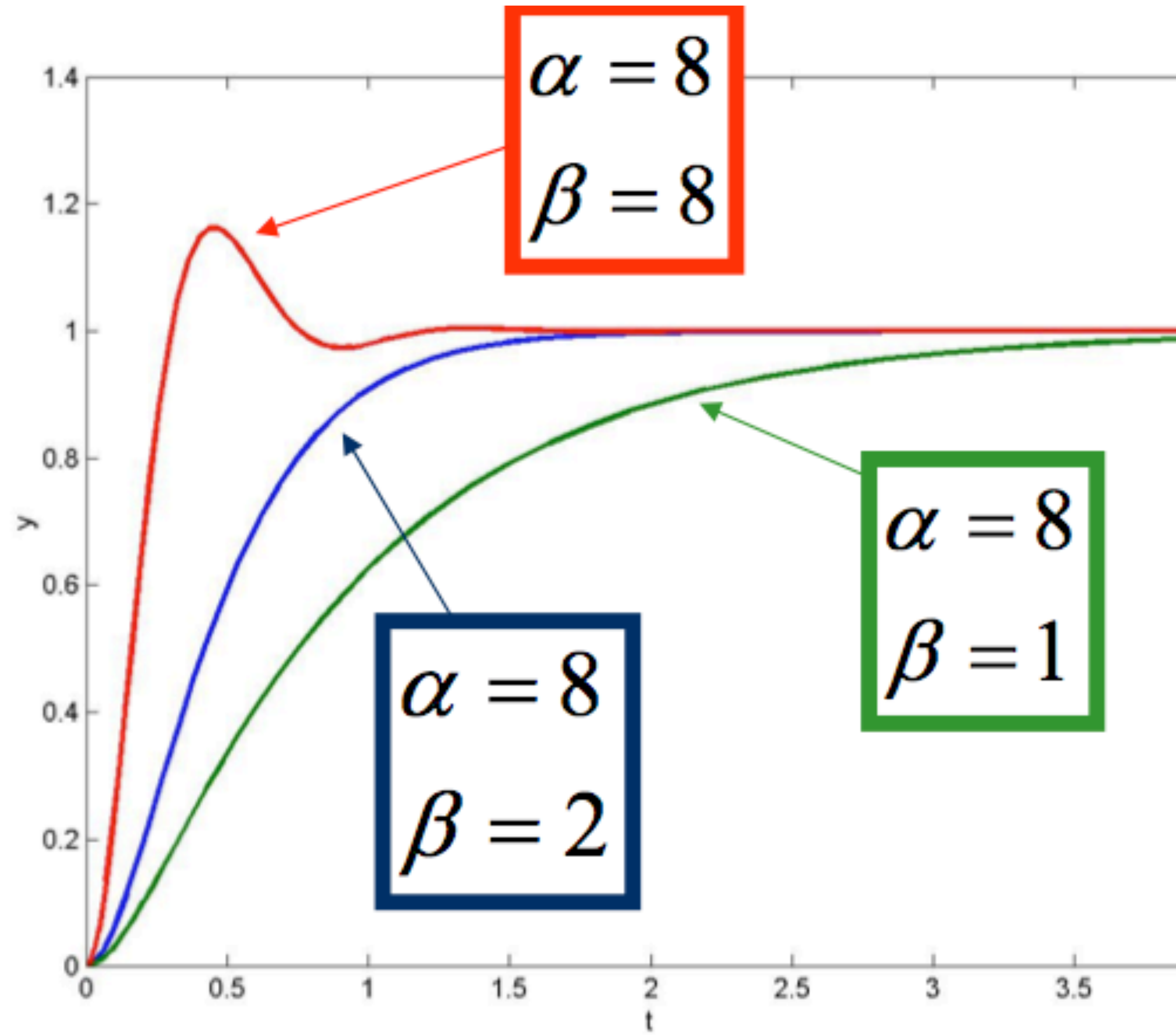
What movements can a differential equation encode?

Second order linear dynamical system:

$$\ddot{y} = \alpha(\beta(c - y) - \dot{y})$$

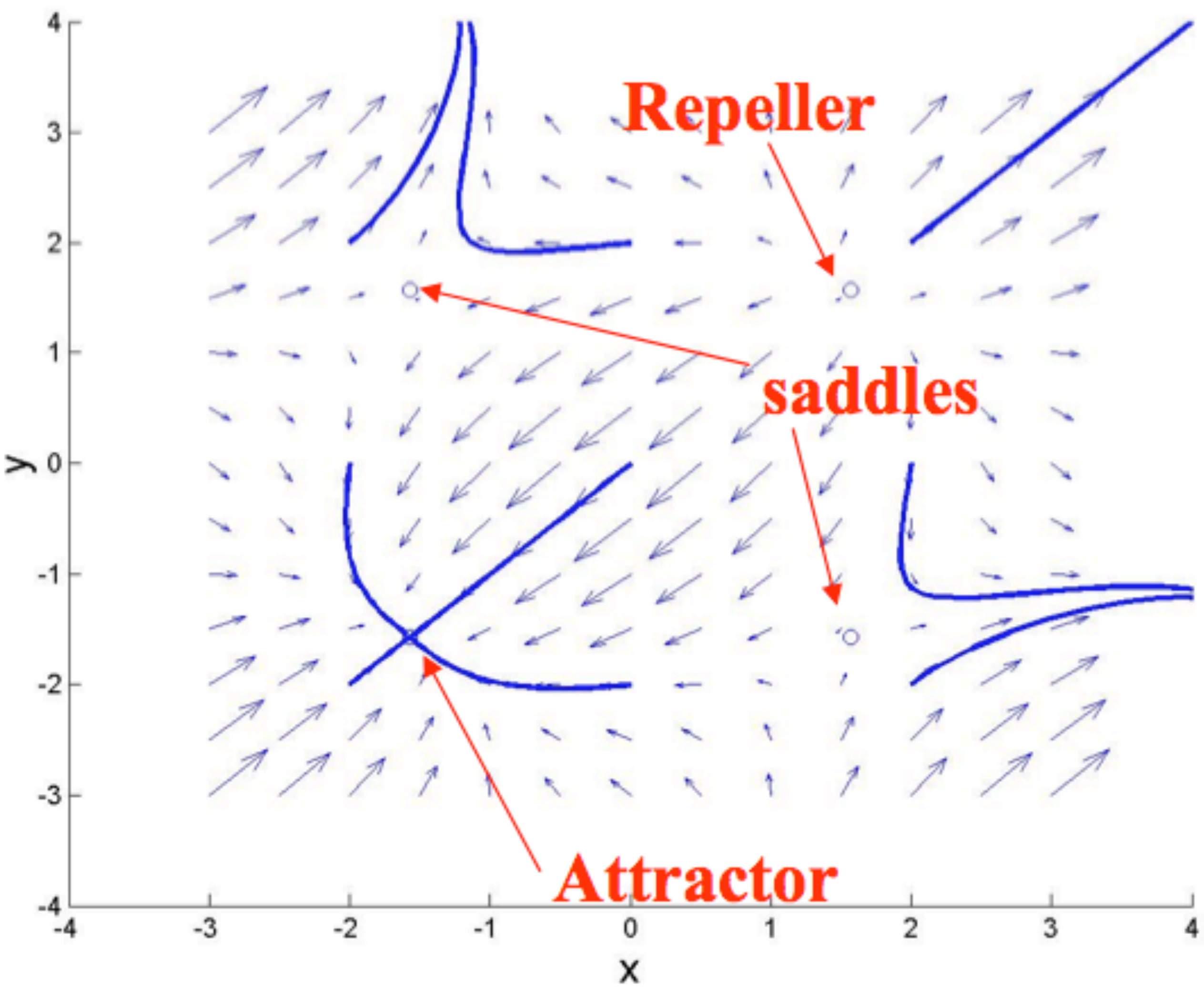
Linear differential equations:

- well-defined behavior
- But: limited class of movements



How can we make it more representative?

$$\dot{y} = -2\cos x - \cos y$$
$$\dot{x} = -2\cos y - \cos x$$

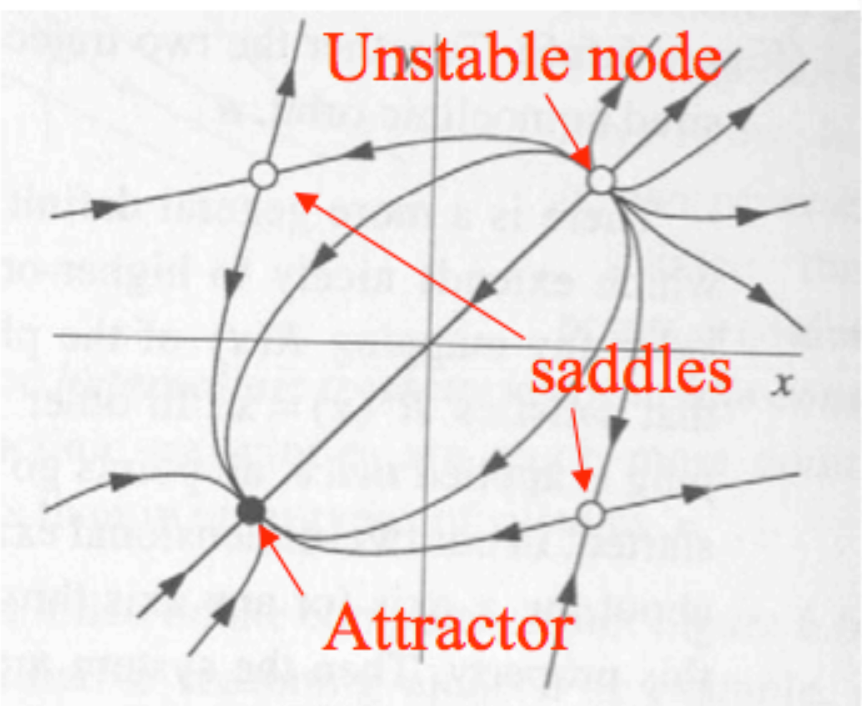


Use **non-linear dynamical systems** ?

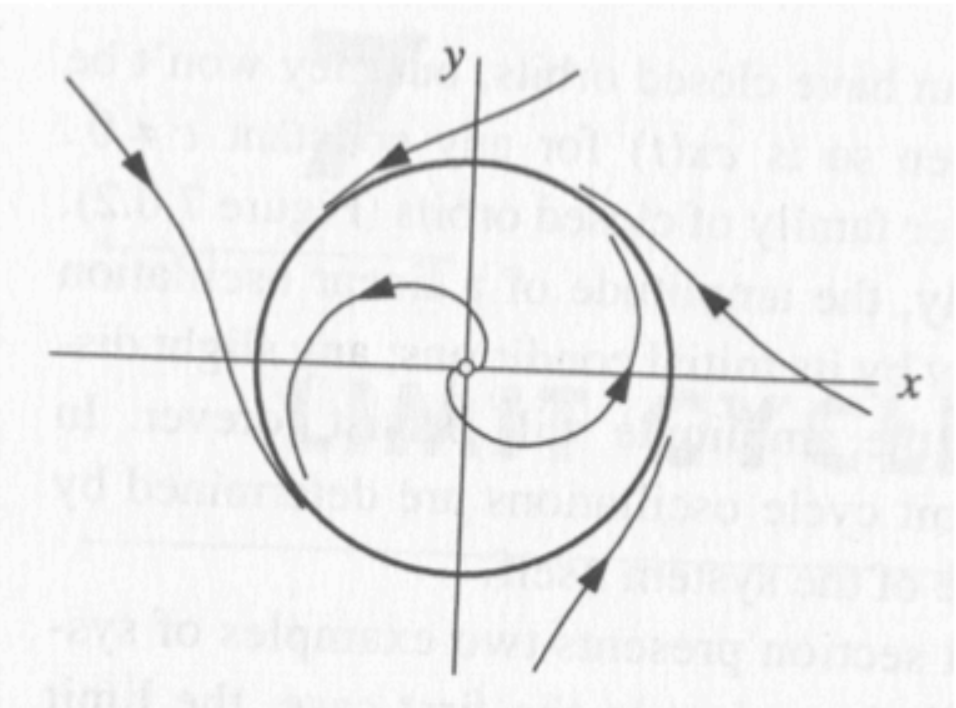
- Can represent more complex behavior
- Can also get unstable! ☹

Non-linear dynamical systems

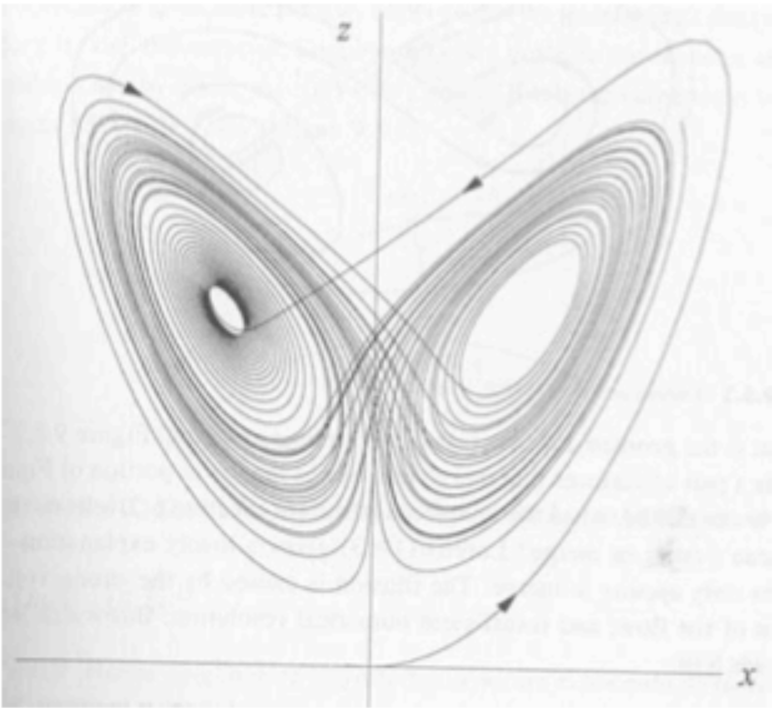
Different behaviors might emerge…



Attractors



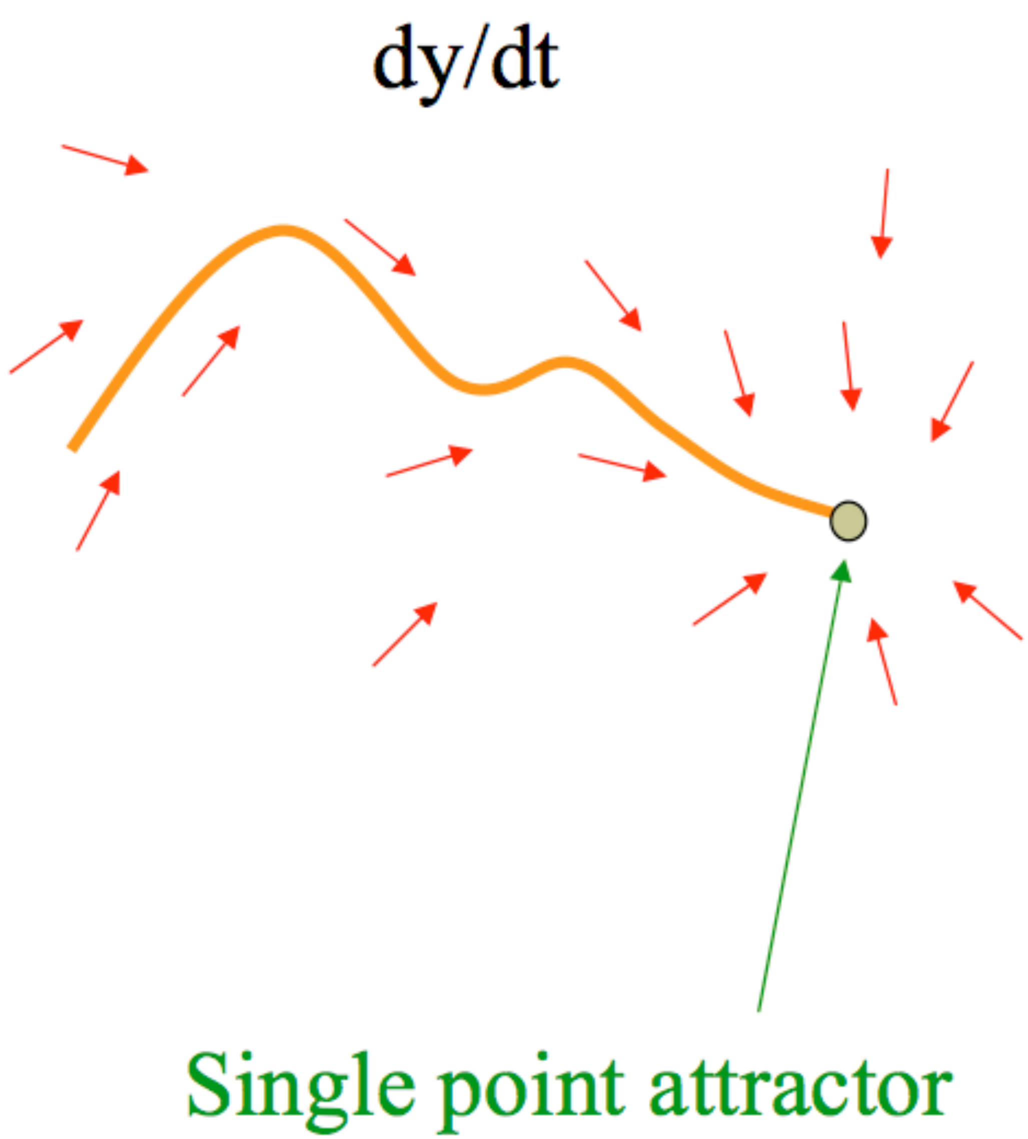
Limit cycles



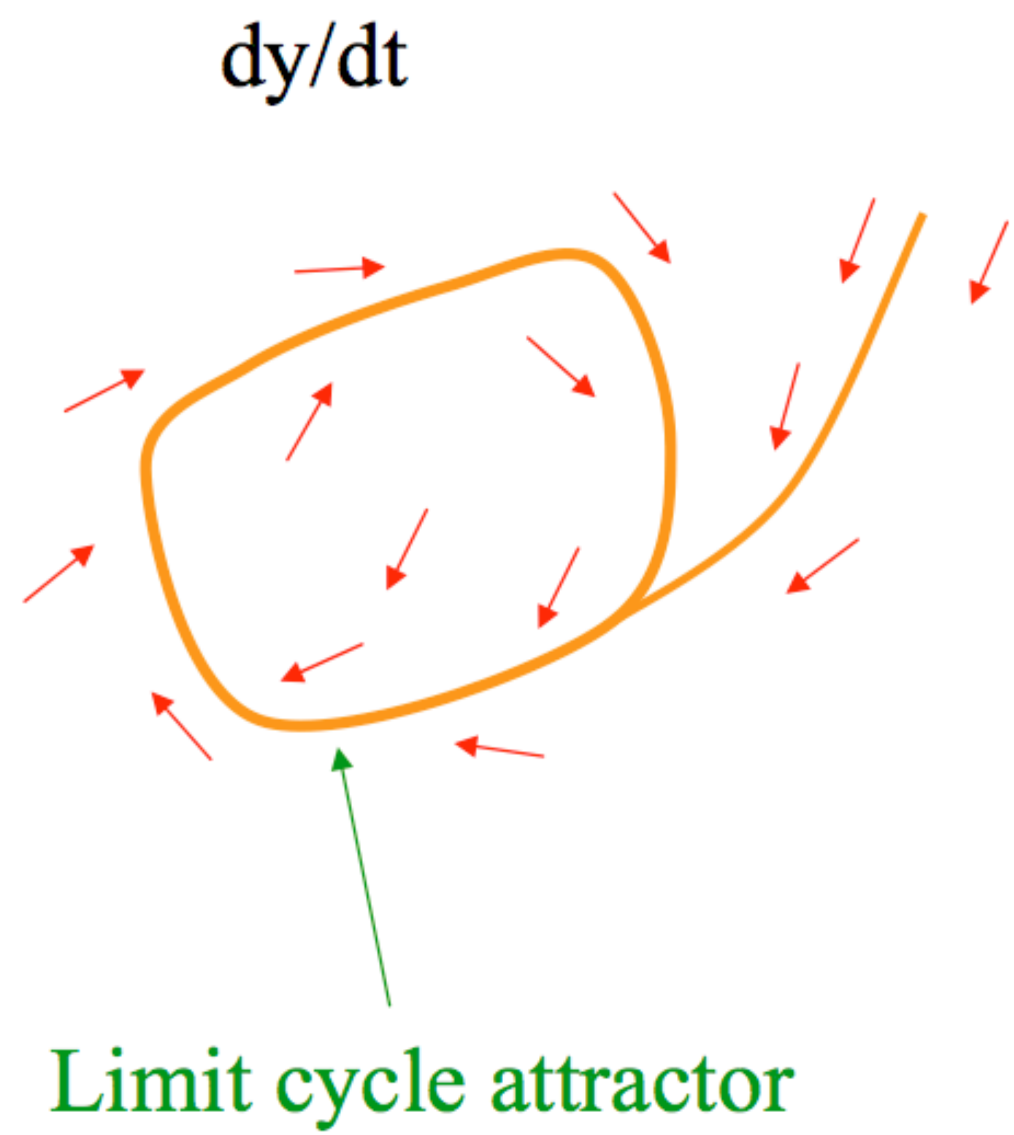
Chaos

Movements as dynamical systems

Discrete movements



Rhythmic movements



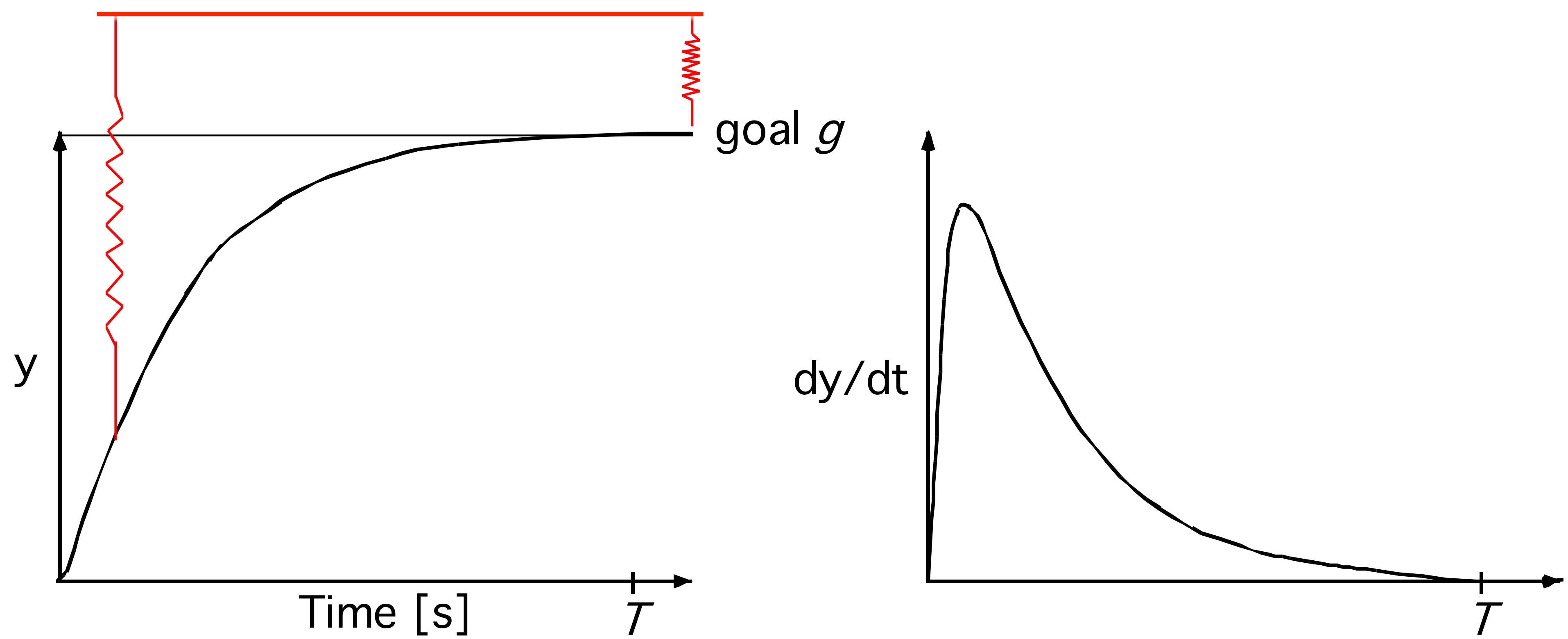
Dynamic Movement Primitives (DMPs)

We can encode desirable properties such as:

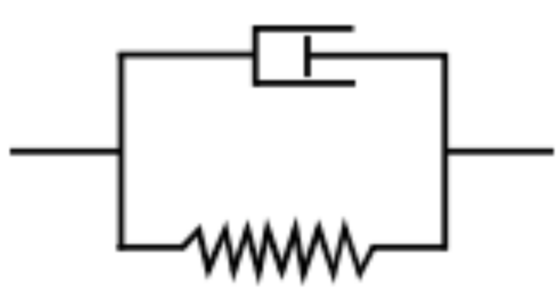
- stability
- perturbation robustness
- periodic and point-to-point behaviors
- Attractors that have rather complex shape
- Easy to learn
- Coupling of a high number of DoFs
- Timing, temporal scaling
- Generalization (structural equivalence for parameter changes)

DMPs: [Ijspeert, Nakanishi & Schaal, NIPS 2003, Schaal, Peters, Nakanishi, Ijspeert, ISRR 2003

Point-to-Point Movements as Dynamic Systems



E.g., for a one degree-of-freedom movement, start with a simple damped spring model



$$\equiv \quad \ddot{y} = \alpha(\beta(g - y) - \dot{y})$$

(Ijspeert, Nakanishi & Schaal, NIPS 2003; Schaal, Peters, Nakanishi, Ijspeert, ISRR 2003)

Dynamic Movement Primitives

How can we encode a desired behavior?

- Add a forcing function to obtain a moving attractor

$$\ddot{y} = \alpha(\beta(g - y) - \dot{y}) + f_{\boldsymbol{w}}(t)$$
$$= \alpha(\beta(g + f_{\boldsymbol{w}}(t)/(\alpha\beta) - y) - \dot{y})$$

- The forcing function encodes the desired additional acceleration profile
- $f_{\boldsymbol{w}}$ … learnable function

(Ijspeert, Nakanishi & Schaal, NIPS 2003; Schaal, Peters, Nakanishi, Ijspeert, ISRR 2003)

Dynamic Movement Primitives

How can we encode a temporal scaling?

Add a phase variable z_t to replace time

$$\ddot{y} = \tau^2 \alpha(\beta(g - y) - \dot{y}/\tau) + \tau^2 f_{\boldsymbol{w}}(z)$$

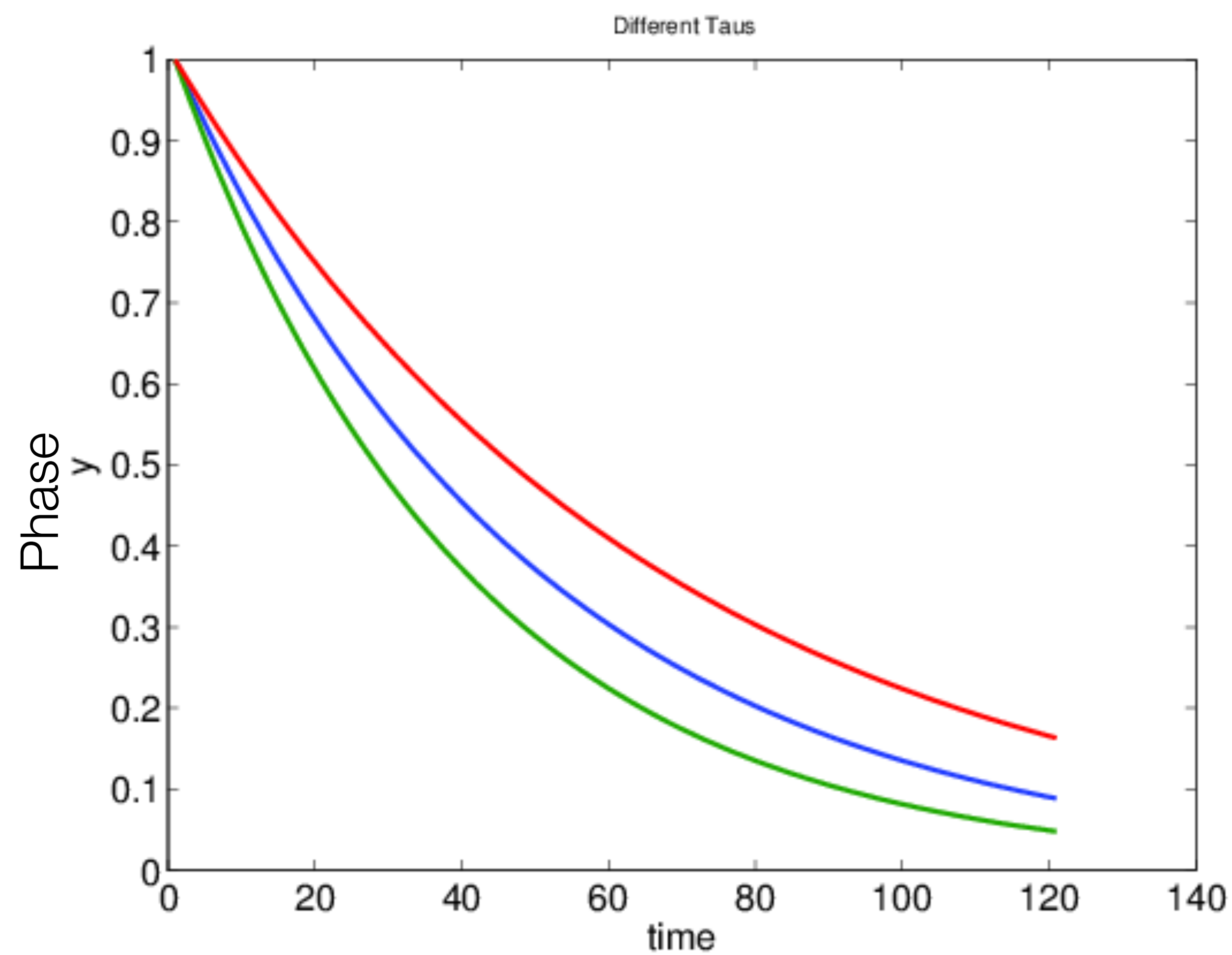
$$\dot{z} = -\tau \alpha_z z$$

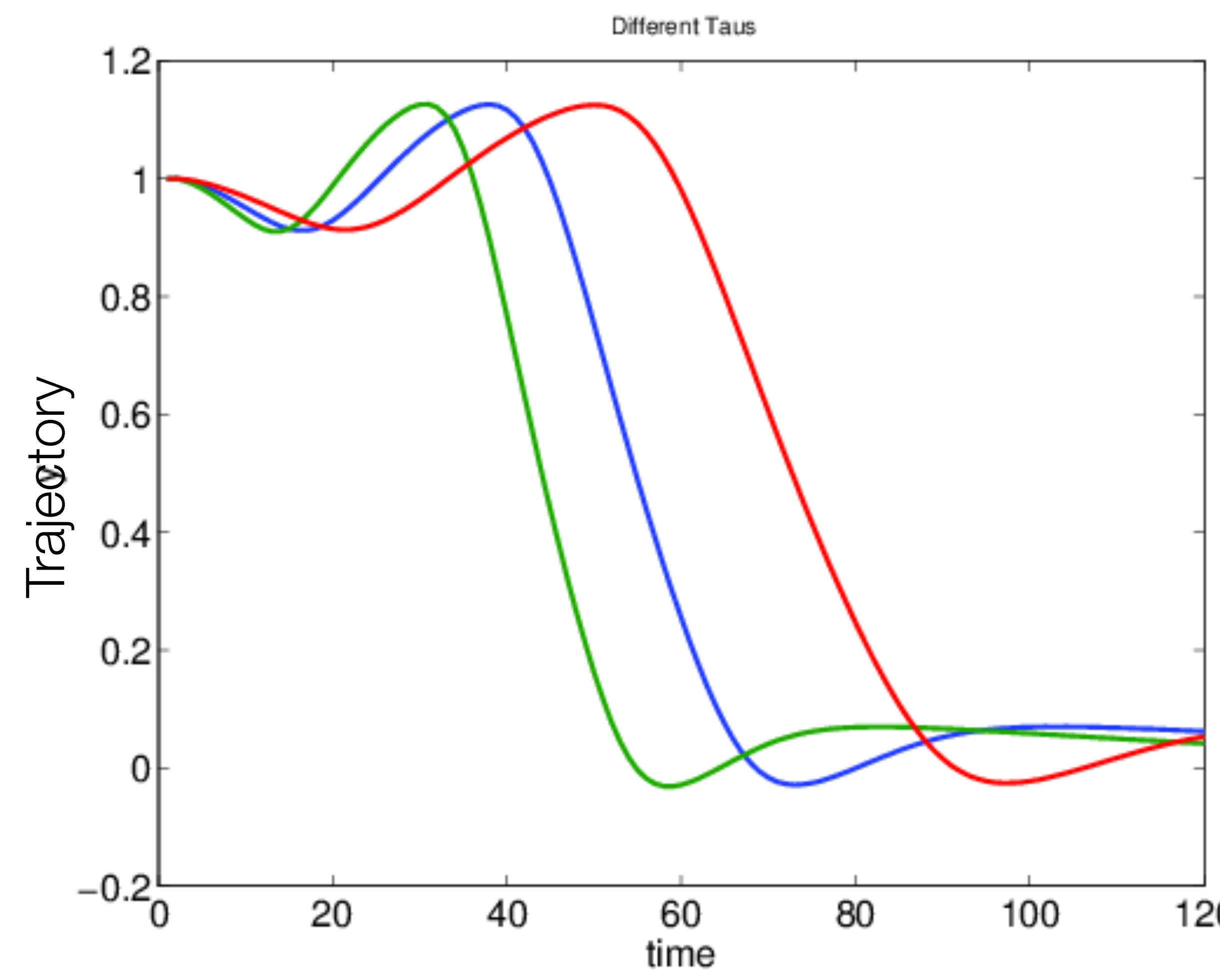
Also uses dynamical system to model phase z

τ ... temporal scaling variable

(Ijspeert, Nakanishi & Schaal, NIPS 2003; Schaal, Peters, Nakanishi, Ijspeert, ISRR 2003)

Adapting the temporal scaling…





higher τ → higher movement speed

Representation of the forcing function

How to represent f?

- Normalized RBF basis functions

$$\phi_i(z) = \exp(-0.5(z - c_i)^2/h_i)$$

$$f_{\boldsymbol{w}}(z) = \frac{\sum_{i=1}^{K} \phi_i(z) w_i z}{\sum_{j=1}^{K} \phi_j(z)}$$

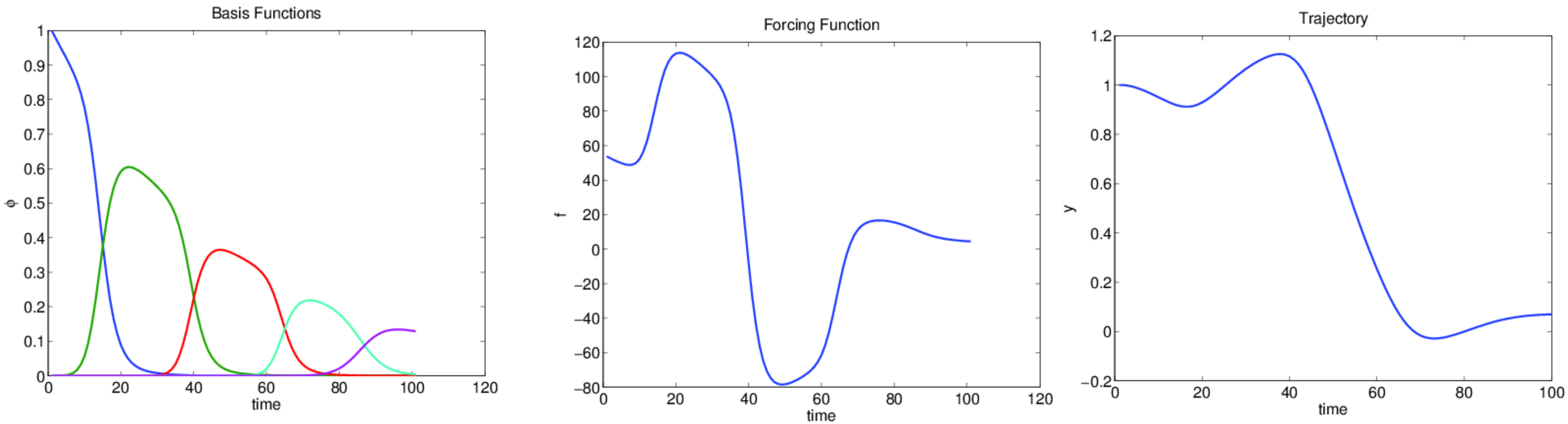
- Matrix Form:

$$f_{\boldsymbol{w}}(z) = \boldsymbol{\psi}^T(z)\boldsymbol{w}, \text{with } \psi_i(z) = \frac{\phi_i(z) z}{\sum_{j=1}^{K} \phi_j(z)}$$

- For $t \to \infty, f_{\boldsymbol{w}}(z) \to 0 \text{ as } z \to 0$

A DMP is stable per construction as the forcing function vanishes ➡ it is just a standard PD for $t \to \infty$

Representation of the forcing function



Integrating the dynamical system leads to the trajectory

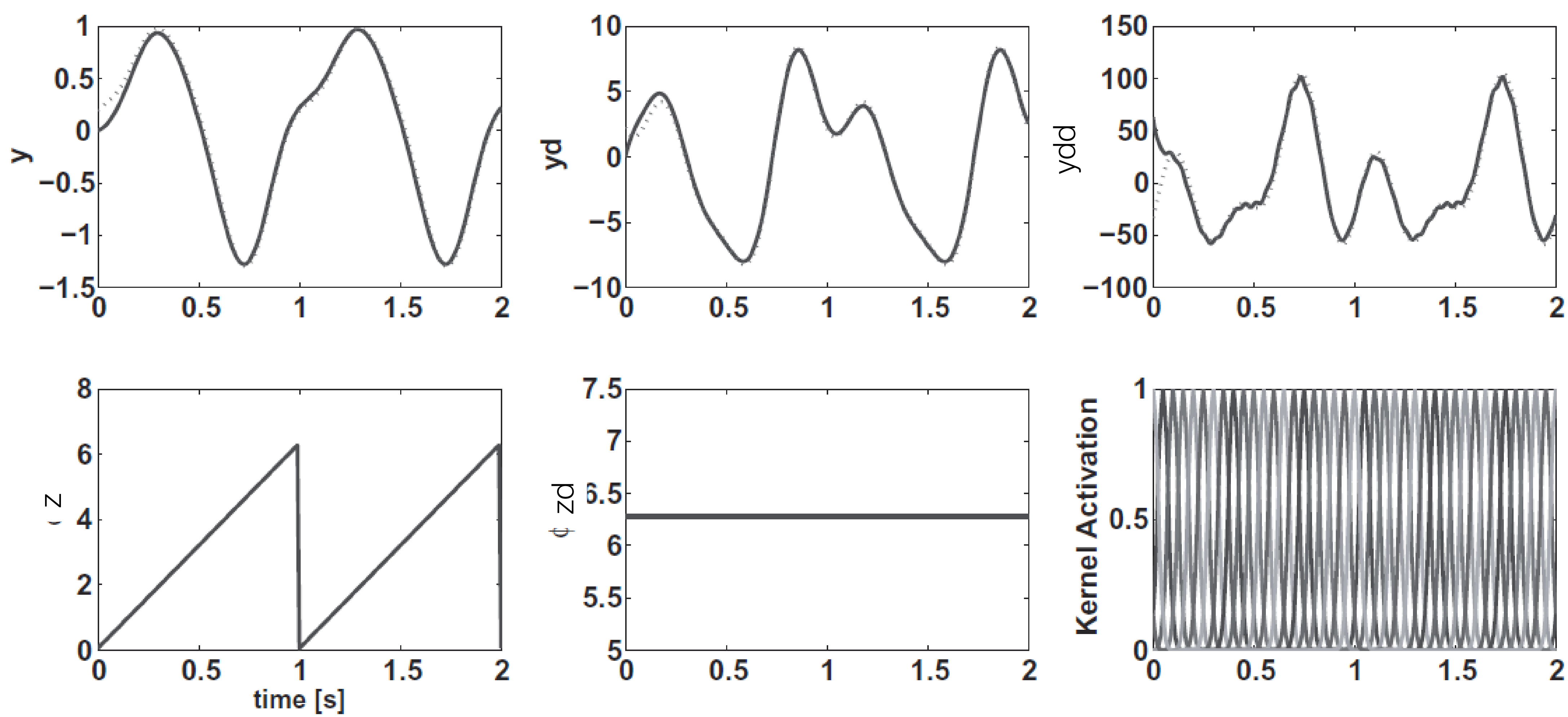
Dynamic Movement Primitives

For multi-DoF robots, we use an individual DMP per DoF

Phase variable z is shared

Coupling between joints due to the shared phase

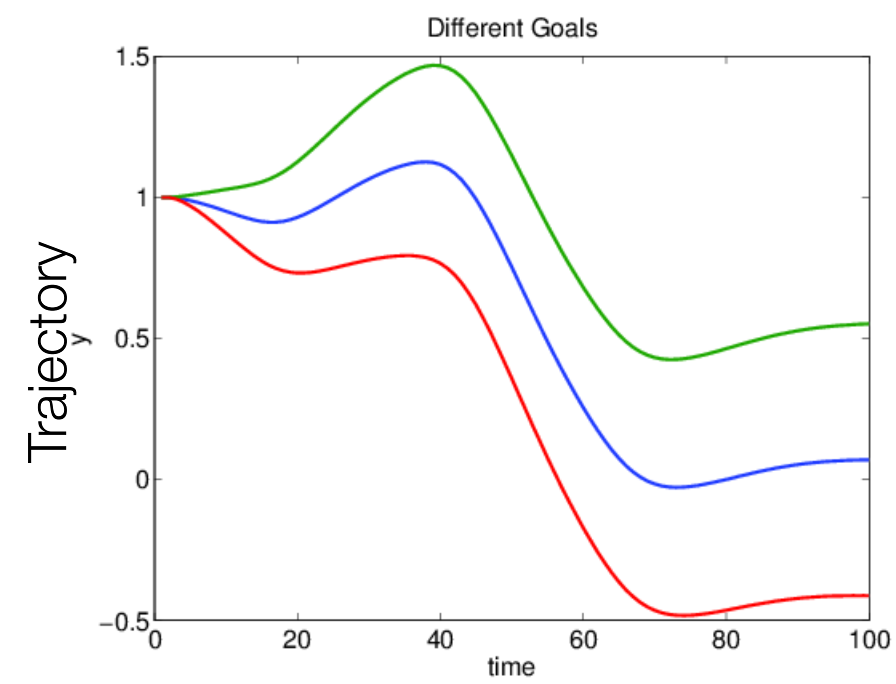
For periodic movements, we can use periodic phase variables



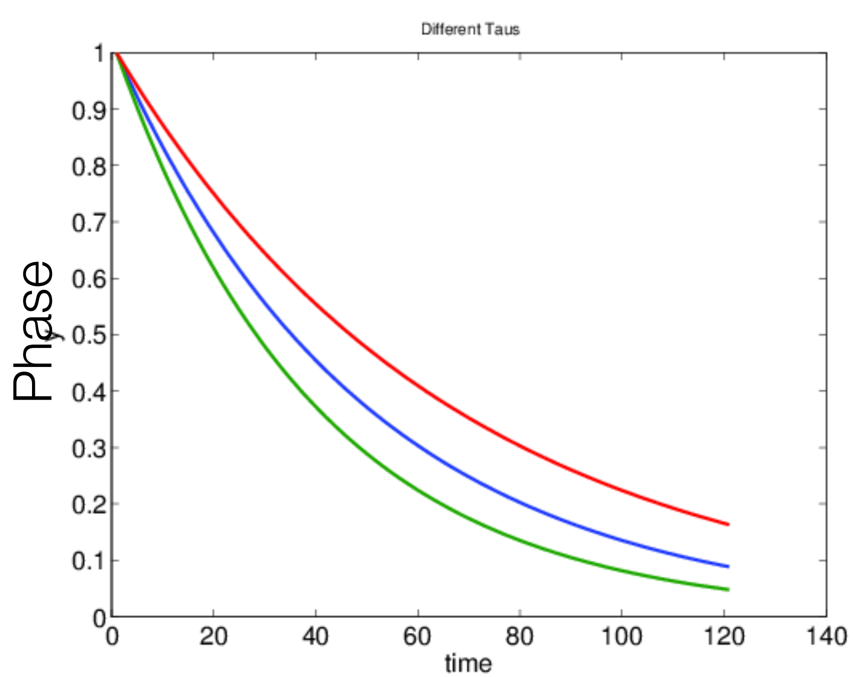
Adapting the meta-parameters…

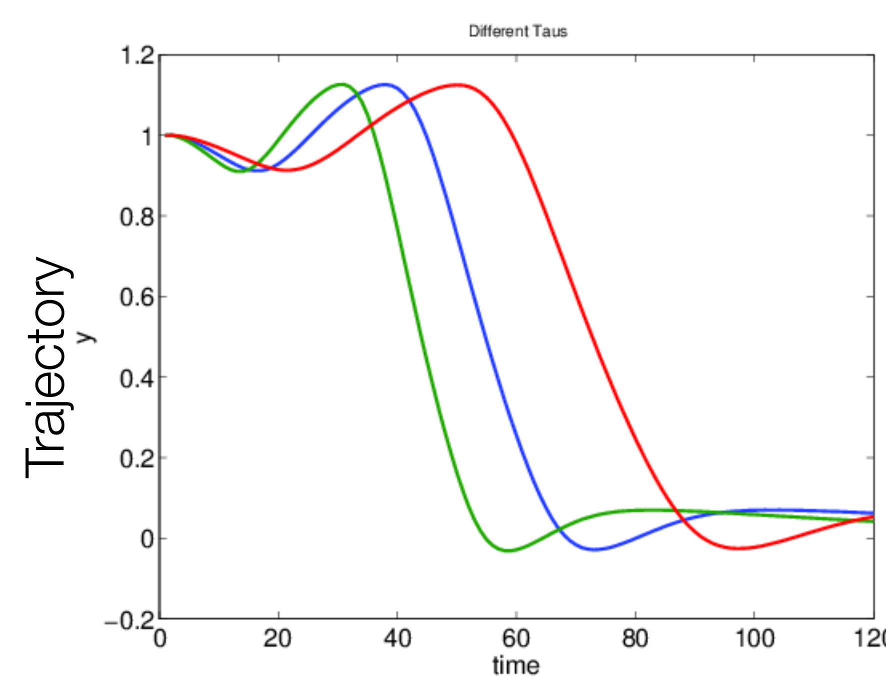
Adapting the goal attractor g

⇨ Changes final position



Adapting the temporal scaling τ





Imitation Learning with DMPs

Given:

- A desired trajectory and its derivatives $\boldsymbol{q}_{1:T}, \dot{\boldsymbol{q}}_{1:T}, \ddot{\boldsymbol{q}}_{1:T}$
- A goal attractor g (e.g. final position of trajectory)
- Parameters: α, β, α_z (typically fixed)
- Temporal Scaling τ: Adjusted to movement duration

Algorithm:

- Compute target values for each time step

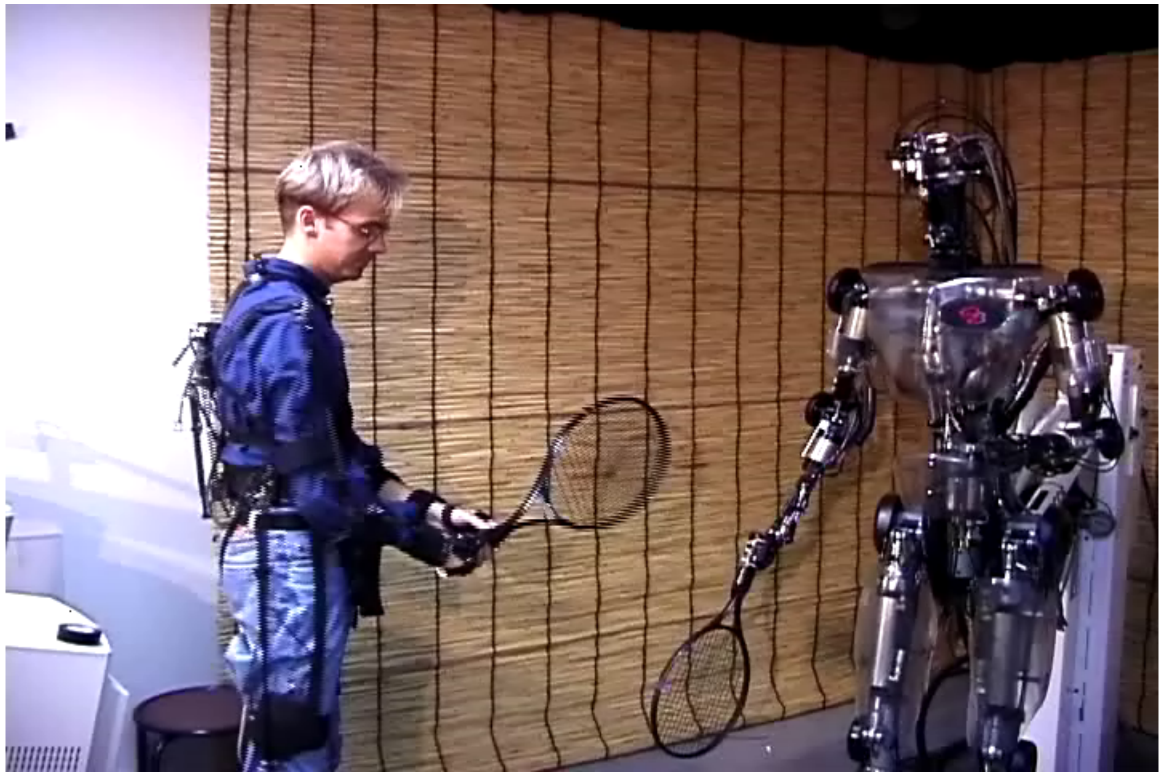
$$f_t = \ddot{q}_t/\tau^2 - \alpha(\beta(g - q_t) - \dot{q}/\tau)$$

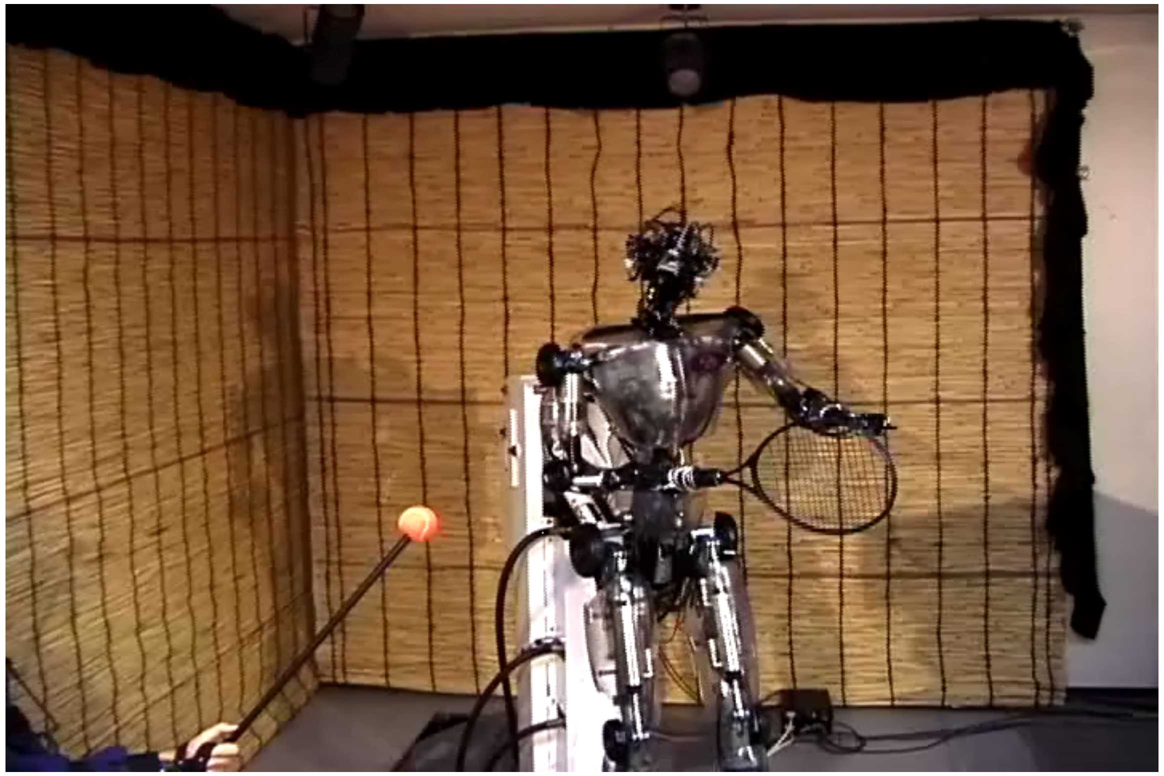
- Compute shape parameters $\boldsymbol{w}$ by linear (ridge) regression

$$\boldsymbol{w} = (\boldsymbol{\Psi}^T\boldsymbol{\Psi} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{\Psi}^T\boldsymbol{f}$$

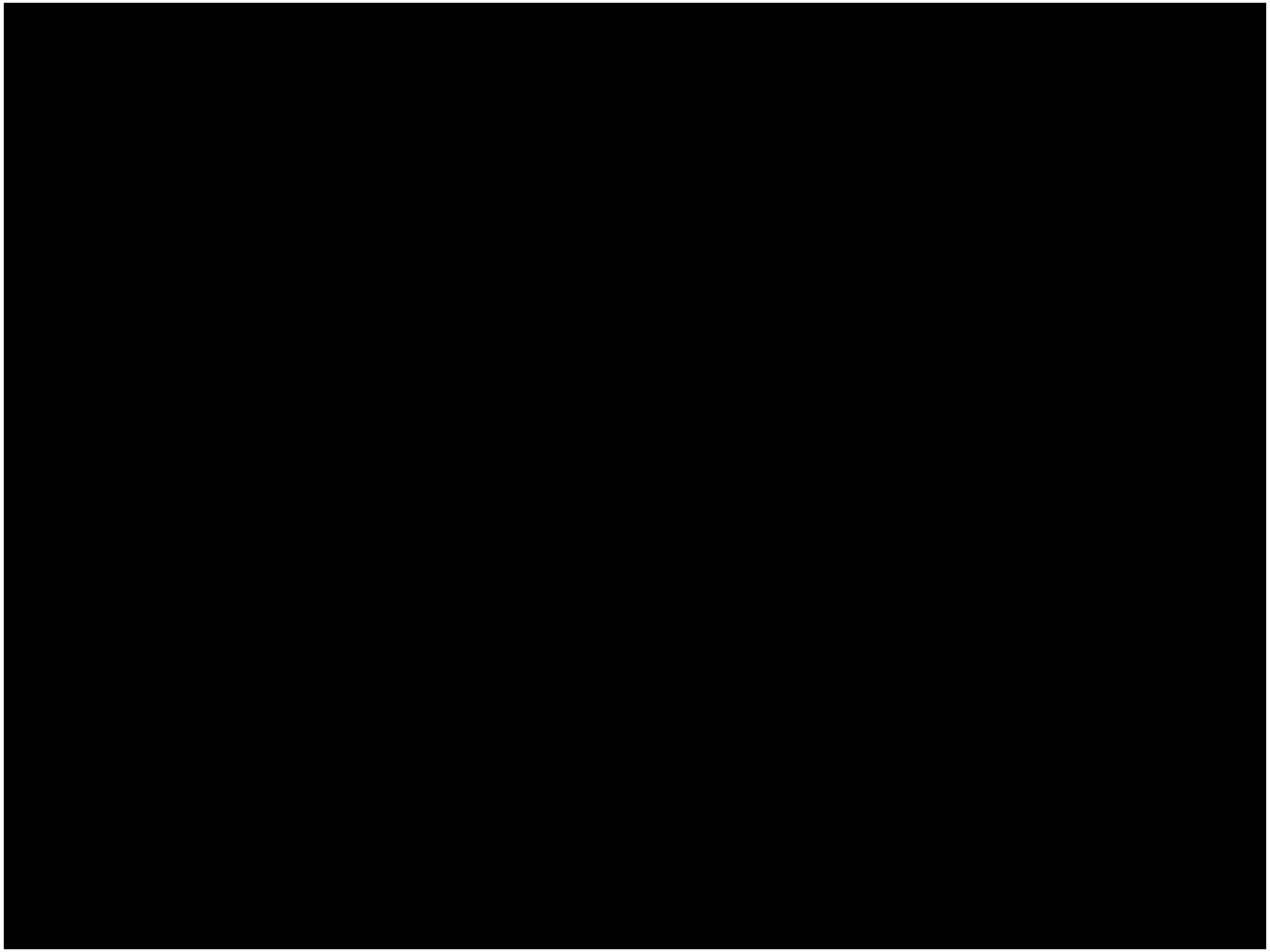
(Ijspeert, Nakanishi & Schaal, NIPS 2003; Schaal, Peters, Nakanishi, Ijspeert, ISRR 2003)

Example: A Tennis Backhand



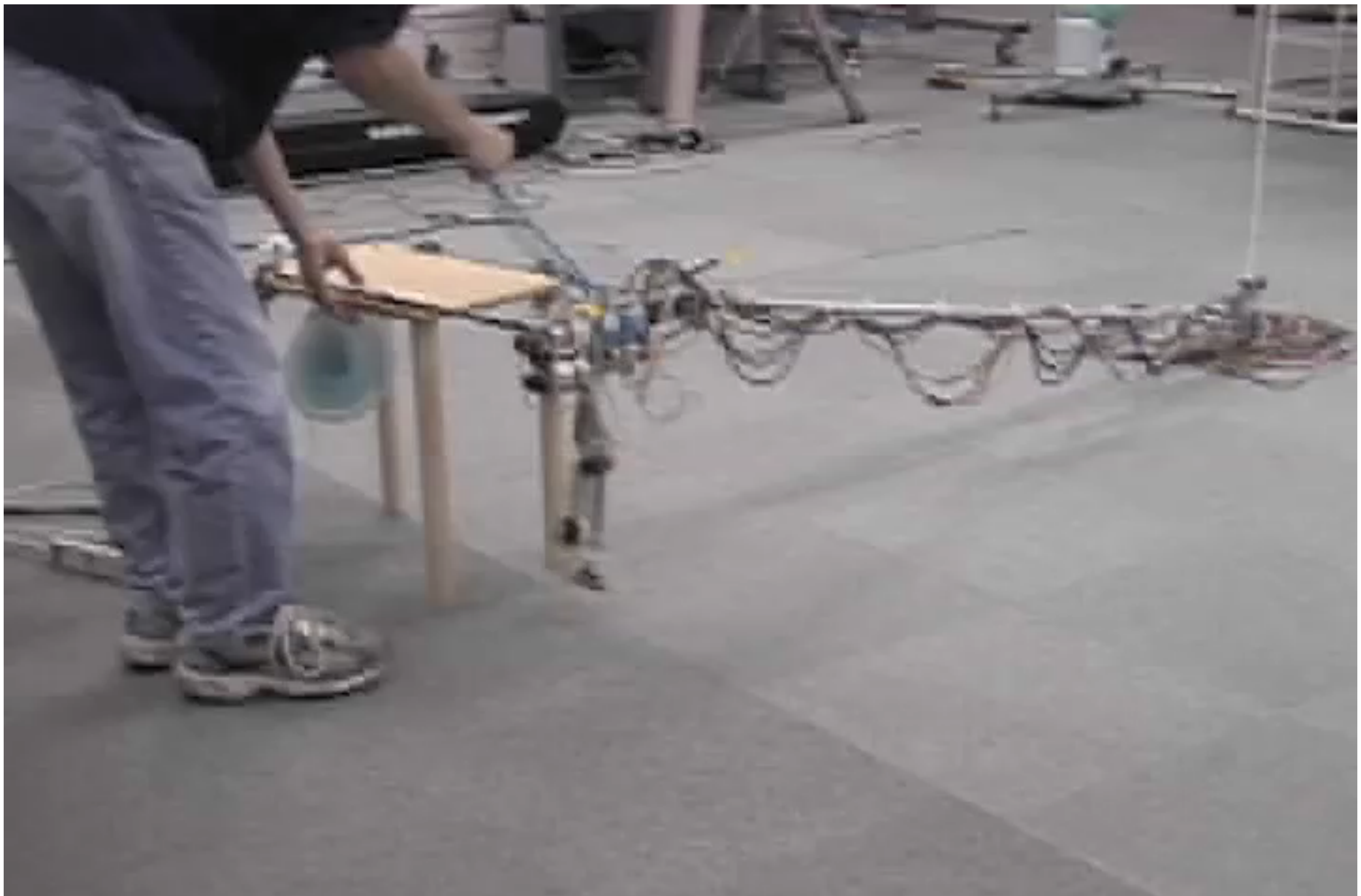


Rhythmic Motor Primitives



(Kober & Peters, ICRA 2009)

Fast Coupling between System and Gait



Movement Primitives

Important properties of movement primitive representations

- Data-Driven: easily learnable from demonstrations
- Generalization: easily adaptable to a new situation
- Combination: Co-activate multiple primitives to solve a combination of tasks
- Temporal Scaling: Modulate the execution speed of the movement
- Coupling: Represent the coupling between a high number of joints
- Variability: Reproduce the stochasticity in the demonstrations
- Optimality: Can we represent optimal behavior?
- Can be applied for rhythmic and stroke-based movements

Movement Primitives

What we have so far…

- Data-Driven: Yes
- Generalization: Only adapt final positions
- Combination: No idea how…
- Temporal Scaling: Yes
- Coupling: Yes, but only the mean is coupled, no correlations
- Variability: No
- Optimality: Is following a single trajectory really optimal? No
- Can be applied for rhythmic and stroke-based movements: Yes

Probabilistic Movement Primitives

TECHNISCHE UNIVERSITÄT DARMSTADT

Stochastic representation of trajectories:

$$\boldsymbol{\tau} \sim p(\boldsymbol{\tau})$$

- Use $\boldsymbol{w}$ to represent a single trajectory

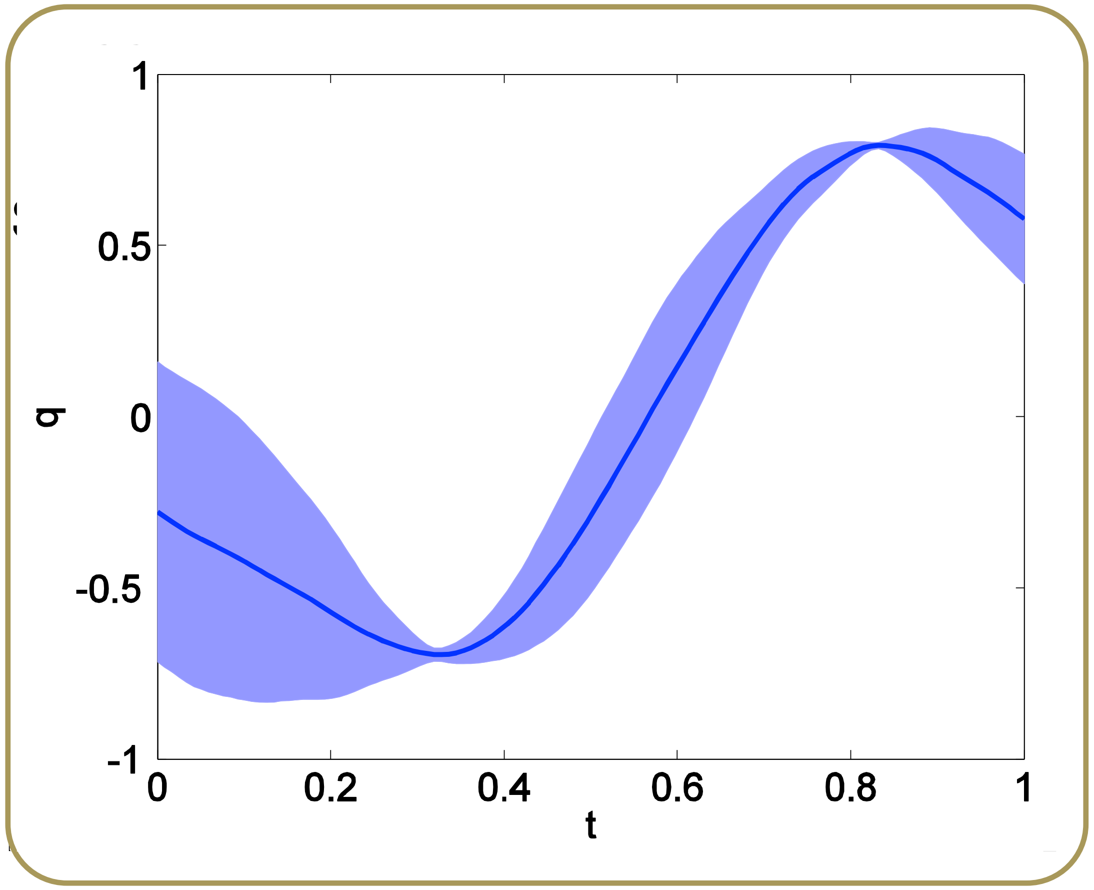
$$\boldsymbol{\tau} = f(\boldsymbol{w}) + \boldsymbol{\epsilon}$$

- Learn a distribution $p(\boldsymbol{w})$ over the $\boldsymbol{w}$ vecto

- Integrate out $\boldsymbol{w}$ to obtain $p(\boldsymbol{\tau})$

Why is this useful?

- We can also represent uncertainty
- Uncertainty gives us information on importance of time points
- We can apply probabilistic operations



How to represent trajectory distributions?

Representation of a single trajectory

$$y_t = \boldsymbol{\psi}_t^T \boldsymbol{w} + \epsilon_y \qquad \epsilon_y \sim \mathcal{N}(0, \sigma^2)$$

Phase-dependent basis: $\psi_t = \boldsymbol{\psi}(z_t)$

For example, normalized Gaussian basis functions

Probabilistic model:

$$p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_t \mathcal{N}(y_t|\boldsymbol{\psi}_t^T \boldsymbol{w}, \sigma^2) = \mathcal{N}(\boldsymbol{\tau}|\boldsymbol{\Psi}\boldsymbol{w}, \sigma^2 \boldsymbol{I}),$$

$$\text{with } \boldsymbol{\Psi} = [\boldsymbol{\psi}_1, \ldots, \psi_T]^T$$

Trajectory distribution: distribution over the parameters $p(\boldsymbol{w}|\boldsymbol{\theta})$

$$p(\boldsymbol{\tau}|\boldsymbol{\theta}) = \int_{\boldsymbol{w}} p(\boldsymbol{\tau}|\boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{\theta}) d\boldsymbol{w}$$

How to represent trajectory distributions?

You can always rely on old friends…

Lets use a **Gaussian**: $p(\boldsymbol{w}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$

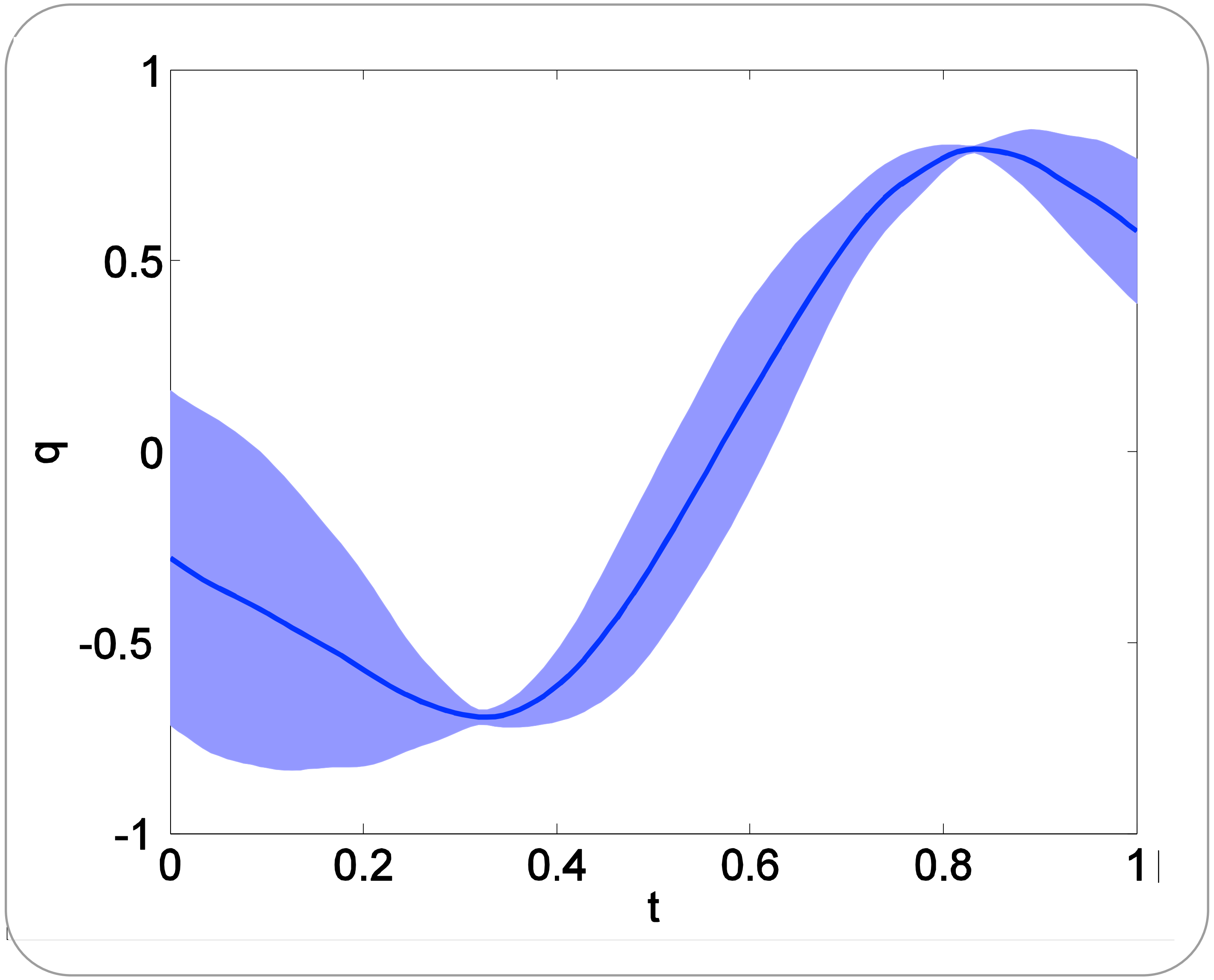
Computing the trajectory distribution is now easy

$$
\begin{aligned}
p(\boldsymbol{\tau}|\boldsymbol{\theta}) &= \int p(\boldsymbol{\tau}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{\theta})d\boldsymbol{w} \\
&= \int \mathcal{N}(\boldsymbol{\tau}|\boldsymbol{\Psi}\boldsymbol{w}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})d\boldsymbol{w} \\
&= \mathcal{N}(\boldsymbol{\tau}|\boldsymbol{\Psi}\boldsymbol{\mu}_{\boldsymbol{w}}, \sigma^2\boldsymbol{I} + \boldsymbol{\Psi}\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Psi}^T)
\end{aligned}
$$

Hence, we can easily **evaluate mean and variance** for any time point

How to represent trajectory distributions?

Hence, we can easily **evaluate mean and variance** for any time point



How to represent trajectory distributions?

How can we encode a distribution over multiple DoFs?

- Use a concatenated weight and trajectory vector and block-diagonal basis matrix

$$\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{\tau}_1, \\ \vdots \\ \boldsymbol{\tau}_D \end{bmatrix} \qquad \boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}_1, \\ \vdots \\ \boldsymbol{w}_D \end{bmatrix} \qquad \boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Psi} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{\Psi} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{\Psi} \end{bmatrix}$$

- The same linear relation holds: $\boldsymbol{\tau} = \boldsymbol{\Phi}\boldsymbol{w}$

- We use a distribution $p(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ over the parameters of all DoFs

For a single time step: $p(\boldsymbol{y}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\phi}_t\boldsymbol{\mu}_{\boldsymbol{w}}, \sigma^2\boldsymbol{I} + \boldsymbol{\phi}_t\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\phi}_t^T)$

Covariance matrix encodes correlation between the joints

Trajectory distribution tracking

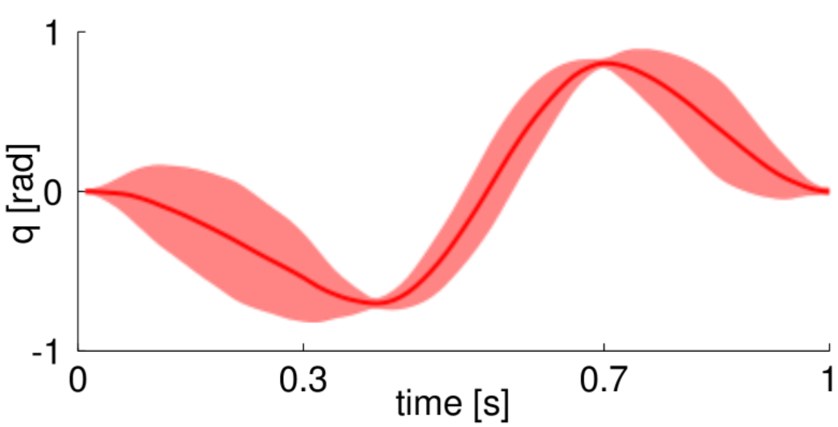
How do we use a trajectory distribution for robot control?

- We can obtain a time-varying stochastic feedback-controller in closed form

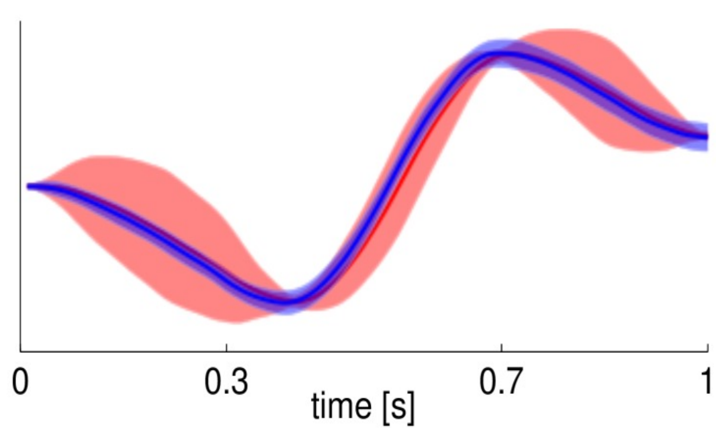
$$\boldsymbol{u}_t = \boldsymbol{k}_t + \boldsymbol{K}_t \boldsymbol{y}_t + \boldsymbol{\epsilon}_{\boldsymbol{u}}, \quad \boldsymbol{\epsilon}_u \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_u)$$

from $\boldsymbol{\tau} \sim p(\boldsymbol{\tau}|\boldsymbol{\theta})$ that exactly reproduces the given trajectory distribution (mean and variances)

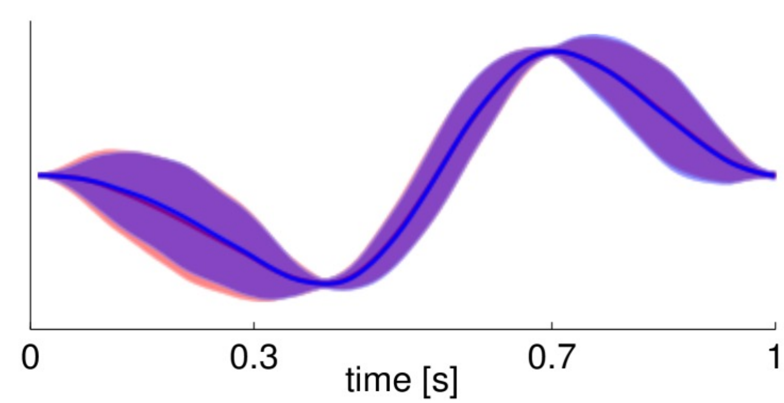
- Same structure as optimal controllers for linear(ized) systems
- But it needs an accurate model



Optimal control



DMPs



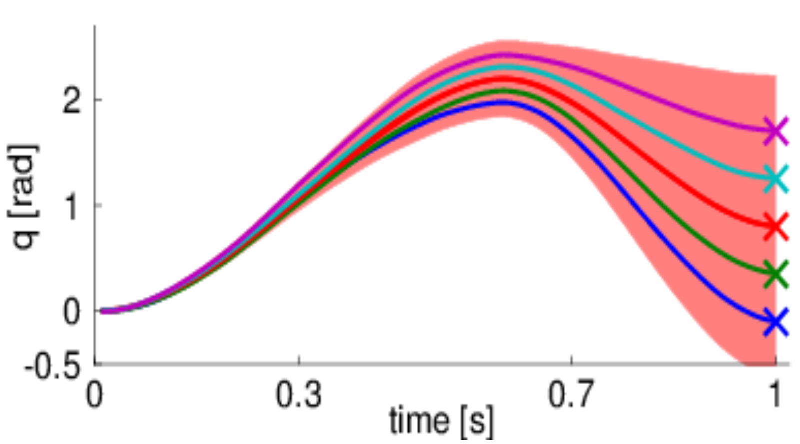
ProMPs

Generalization via Conditioning

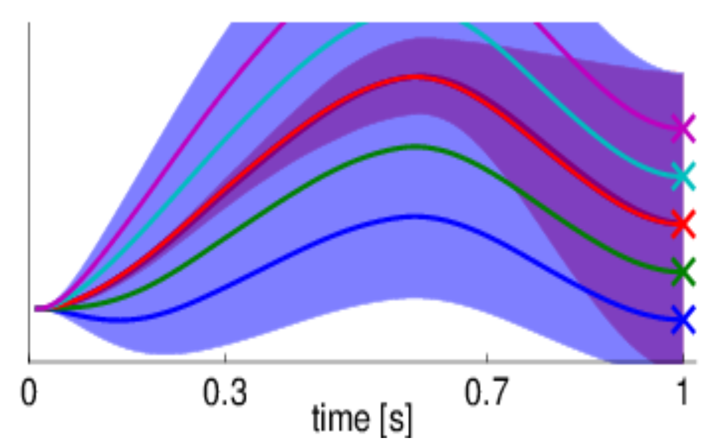
Generalization: Change intermediate or end-point of the movement

We can condition $p(\boldsymbol{w})$ on reaching position $\boldsymbol{q}_t^*$ at time-step t

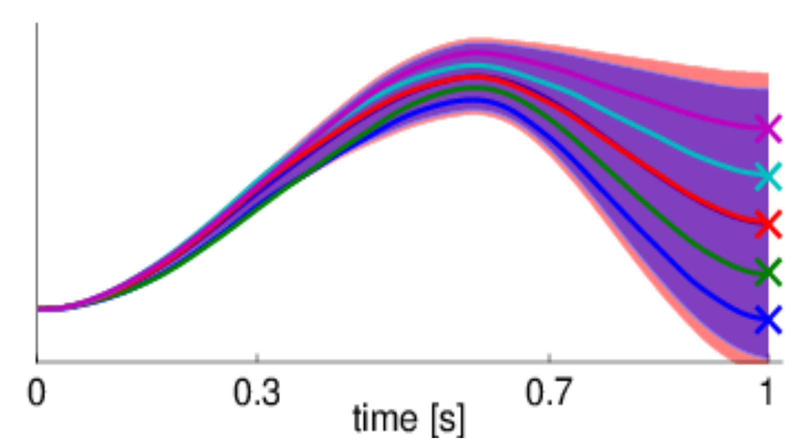
- ➡ New trajectory distribution $p(\boldsymbol{w}|\boldsymbol{q}_t = \boldsymbol{q}_t^*)$ is obtained by Bayes theorem
- ➡ Closed-form solution for Gaussian trajectory distributions



Demonstration



Dynamic MPs



ProMPs

Combination of Movement Primitives

Modularity: Combine primitives to solve a combination of tasks

Implemented as product of distributions:

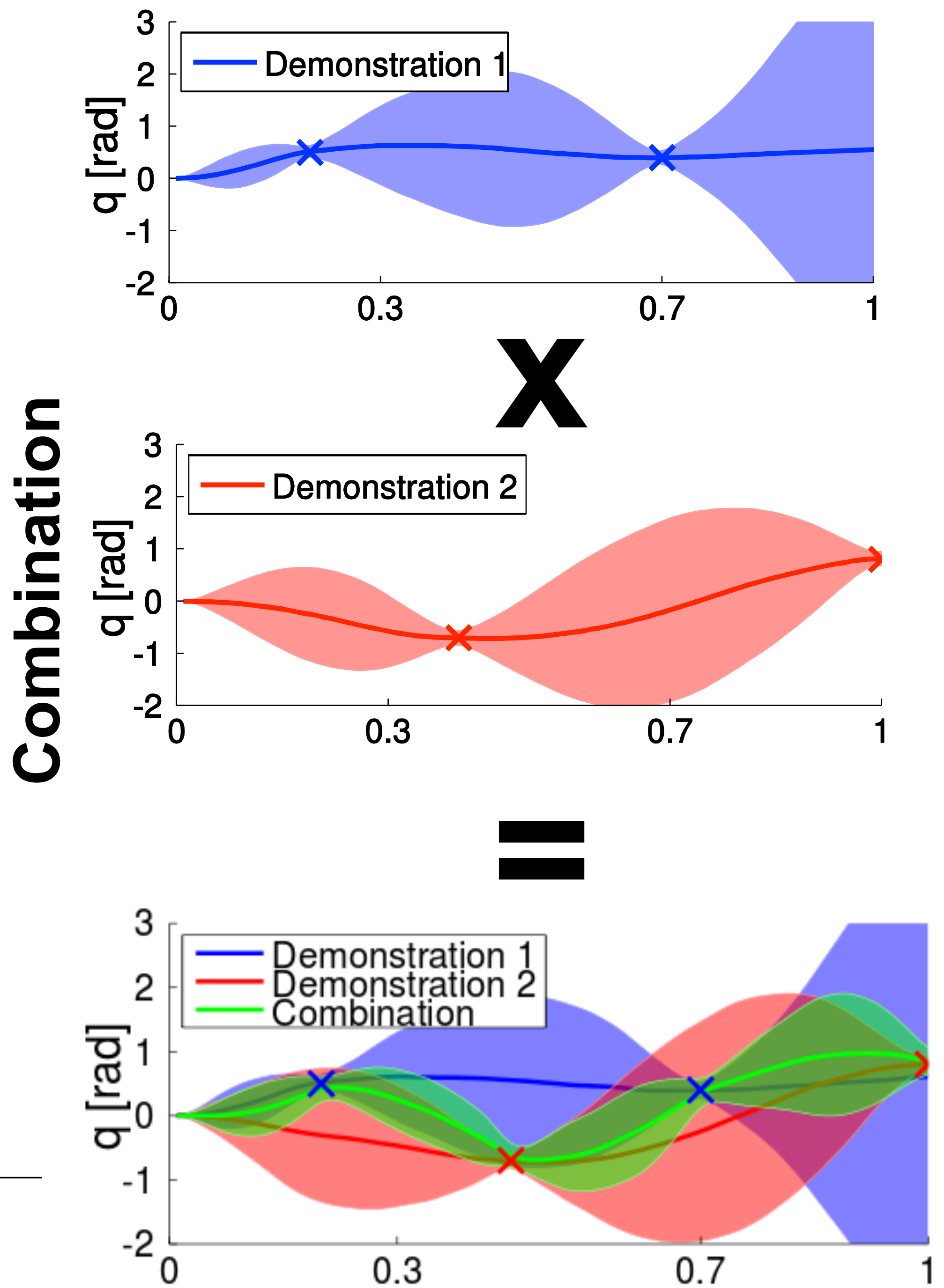
- „Intersection" of trajectory distributions
- Area, in which all distributions have high probability

$$p_{\text{co}}(\boldsymbol{q}_t) \propto \prod_{i=1}^{N} p_i(\boldsymbol{q}_t)^{\alpha_i(t)}$$

$p_i(\boldsymbol{q}_t) \ldots$ i-th movement primitive

$\alpha_i(t) \ldots$ activation factors

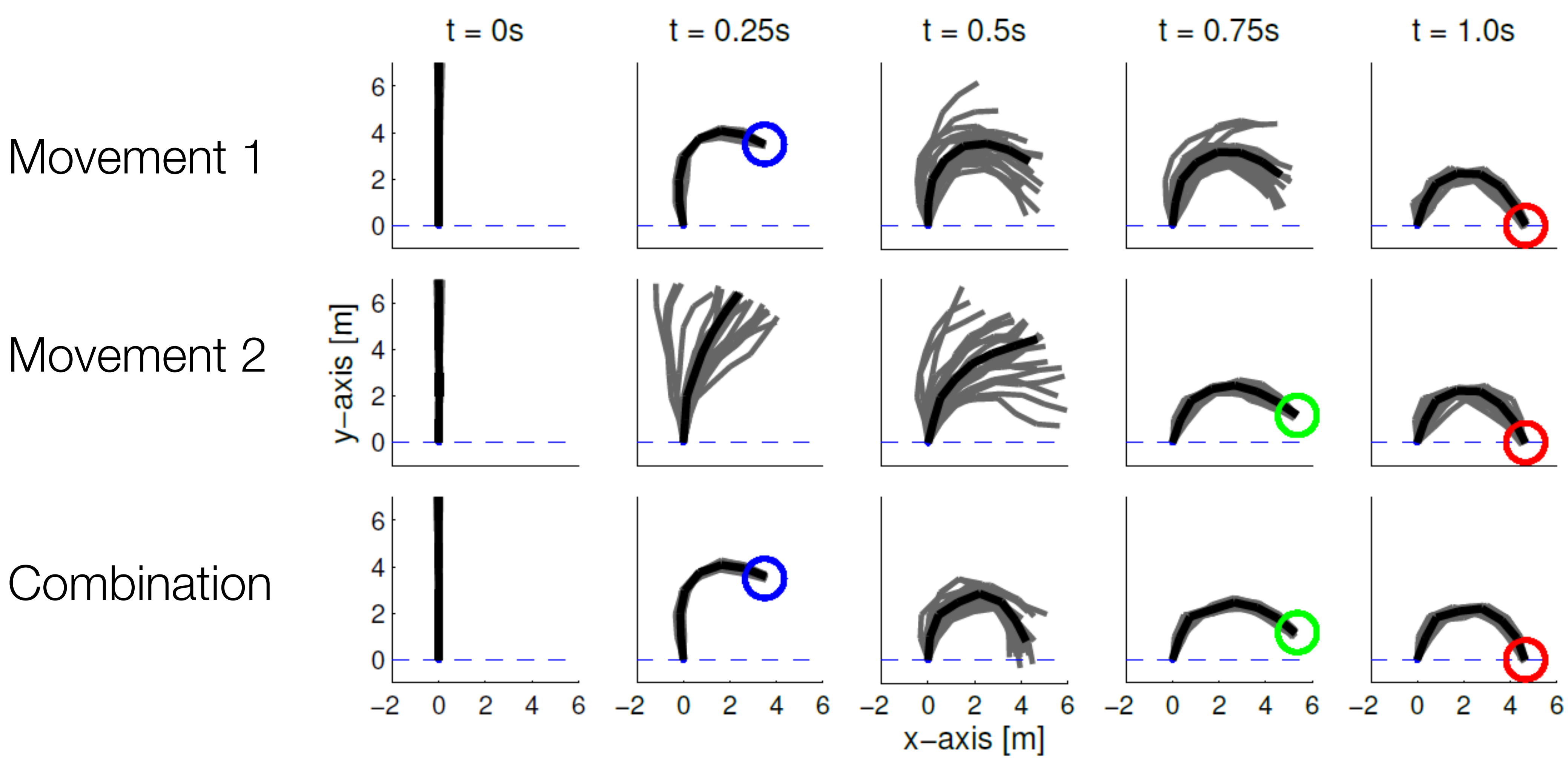
- Computed in closed-form for Gaussian distributions



Experiments: Co-Activation

7-link planar robot arm, controlled by inverse dynamics

- Trained 2 movements for reaching different via-points at different time steps
- Combination of the movements reaches all 2 via-points

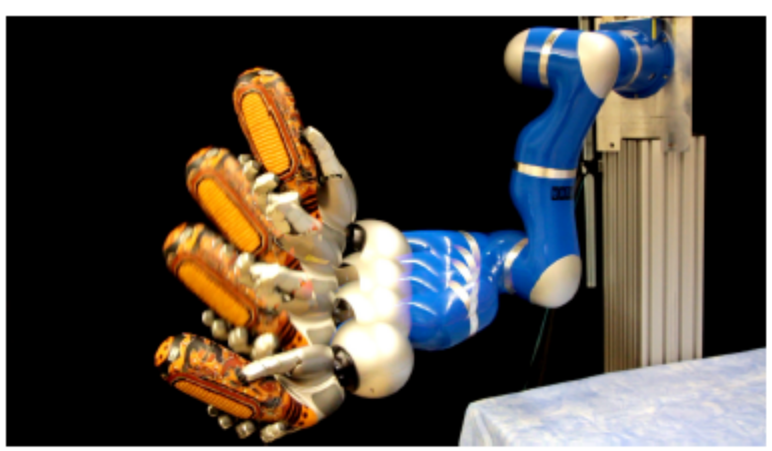


Experiments: Blending and temporal scaling

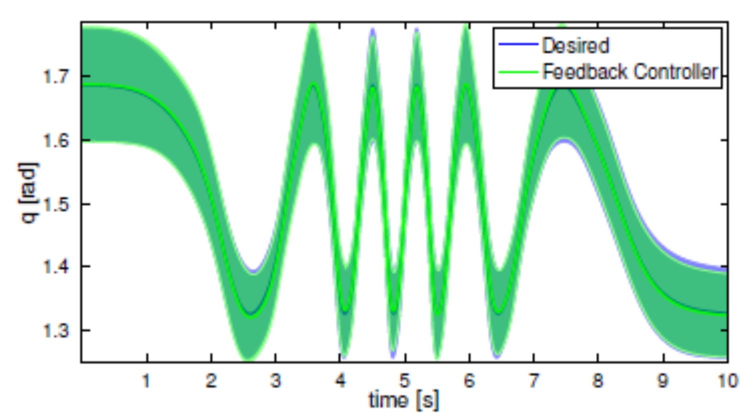
7-link KUKA robot arm, playing maracas

- Record rhythmic movements to produce sounds
- Blend between different rhythmic movements

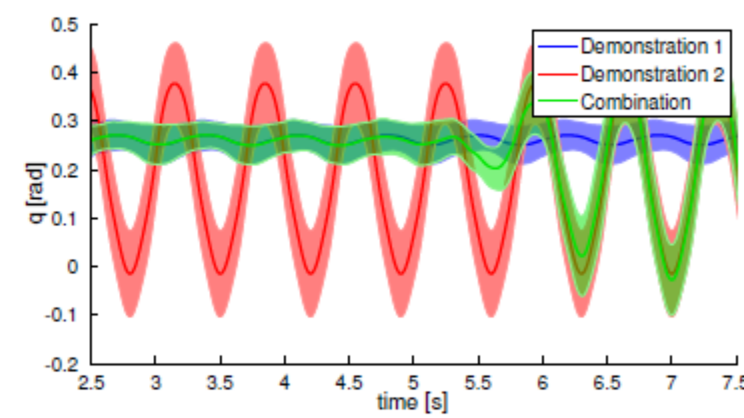
Maracas



Temporal scaling



Blending

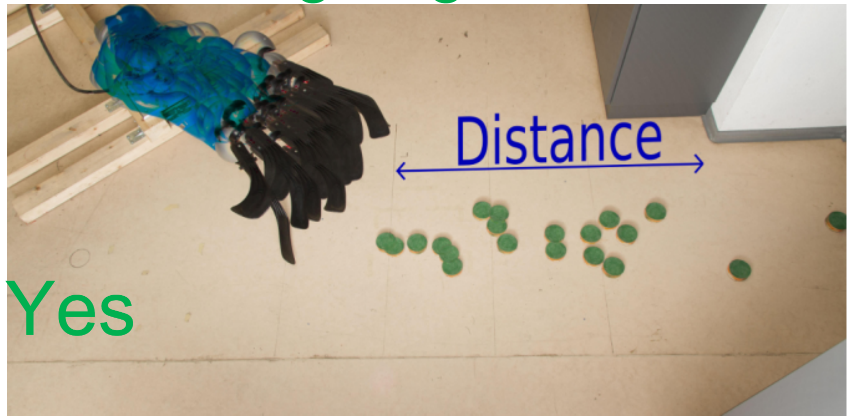


Case Study: Robot Hockey

7-link KUKA robot arm, playing hockey

➡ Train 2 primitives with high variance in shooting angle or in distance

Demonstration 1

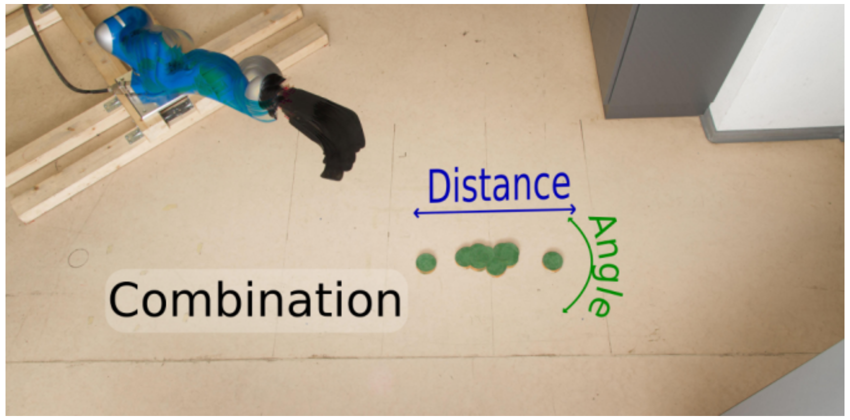


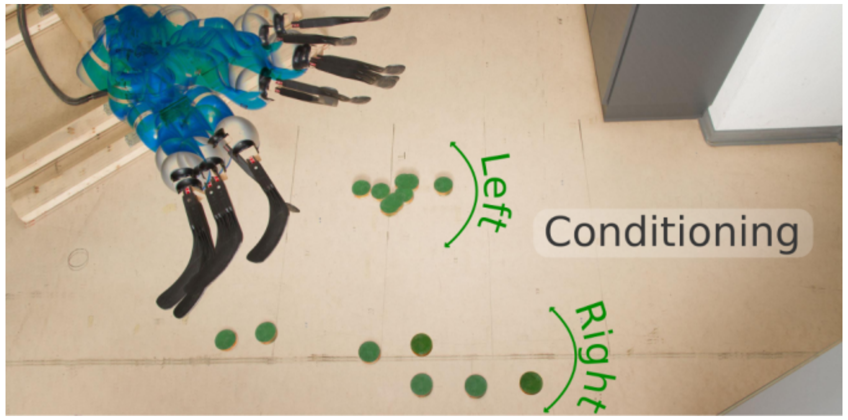
Demonstration 2

➡ Product of the primitives:

Combination of both tasks

➡ **Conditioning** to select the shooting angle





Movement Primitives

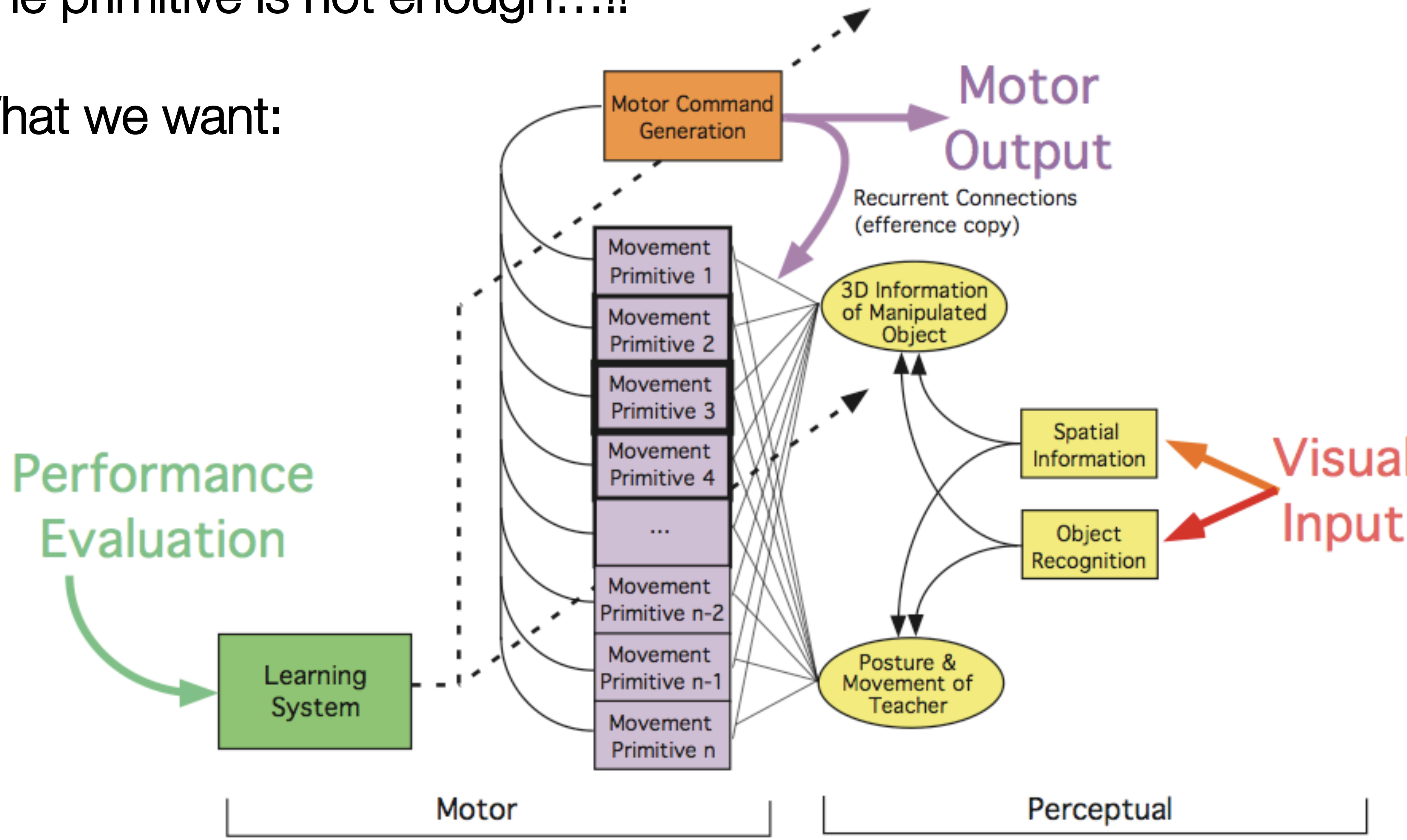
What we have so far…

- Data-Driven: Yes
- Generalization: Yes
- Combination: Yes
- Temporal Scaling: Yes
- Coupling: Yes
- Variability: Yes
- Optimality: Yes
- Can be applied for rhythmic and stroke-based movements: Yes

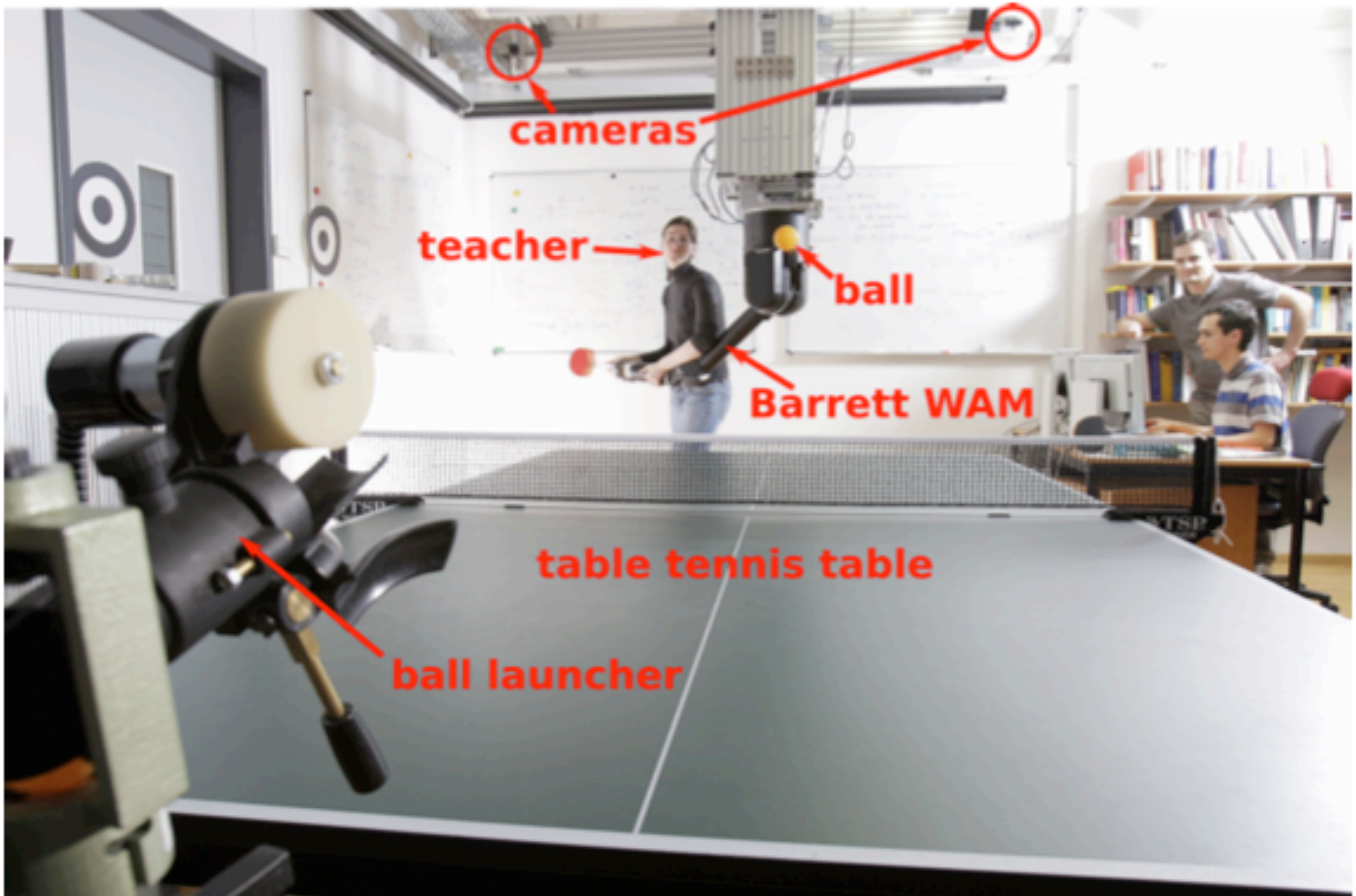
Libraries of Primitives

One primitive is not enough…!!

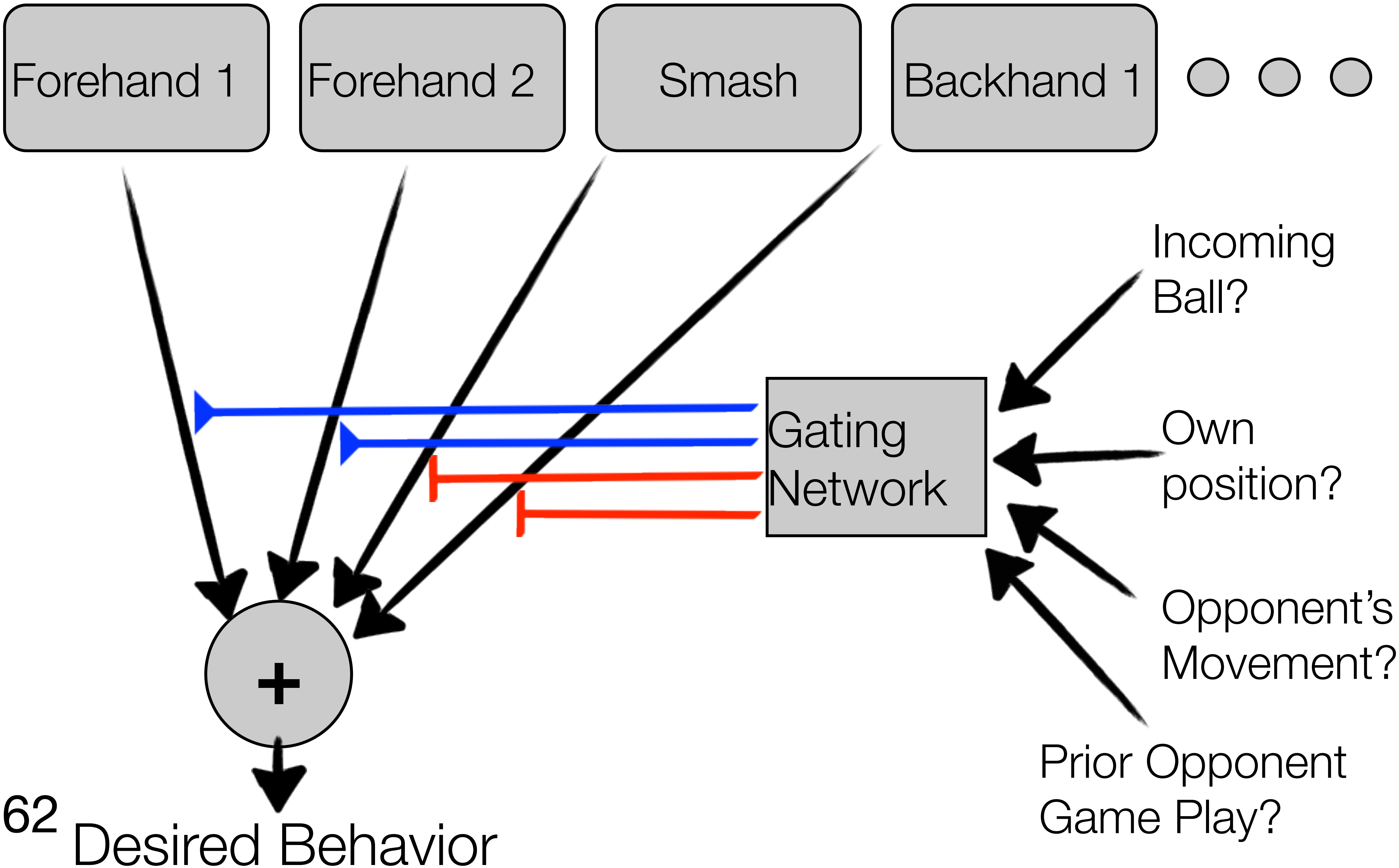
What we want:



Imagine the following situation...



What about many primitives?



What you can do with it...

Core Open Questions in Imitation Learning

- **What to Imitate?** The data traces will contain outliers, redundant data, data that is irrelevant to the task. How can the system extract the relevant components? Imitate on which **level of abstraction**?

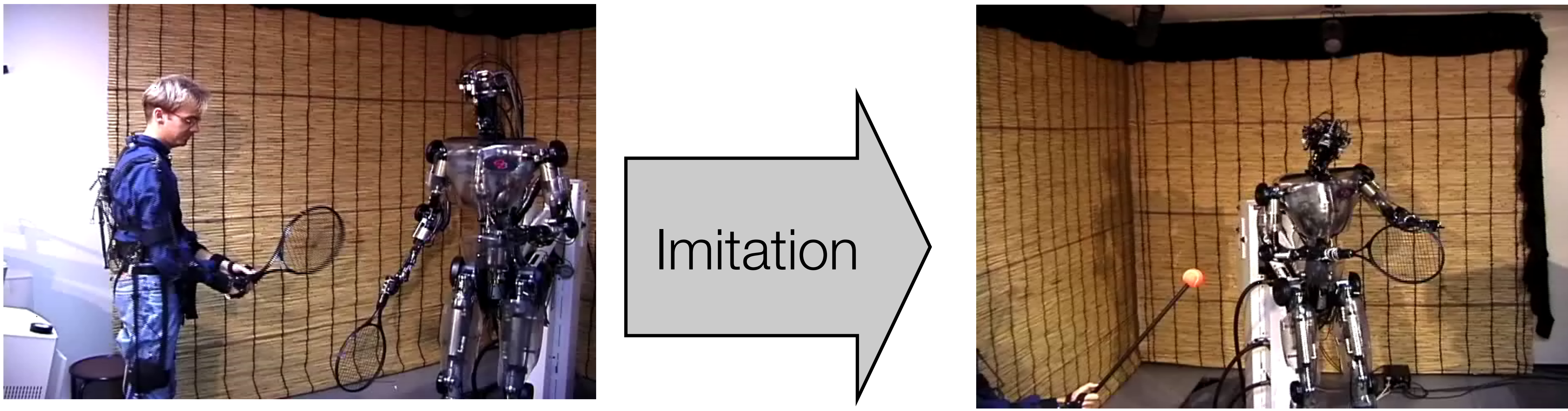
- **How to Imitate?** **body of the teacher** $\neq$ **body of the student**
  ⇨ „Correspondence Problem“.

- **When to Imitate?** Not all behavior in a data stream may be suited for imitation. Untackled questions

- **Whom to Imitate?** If a scene with several actors is observed, the correct one needs to be extracted. Untackled questions

(Nehaniv & Dautenhan, 2001)

Imitation Learning



Problems of Imitation Learning

- Correspondence Problem ➜ requires reinforcement learning
- Imperfect demonstrations ➜ require reinforcement learning
- Intent identification ➜ requires inverse reinforcement learning

Summary…

What you should know…

- State-space representations versus trajectory-based policy representations
- What is imitation learning and when does it fail?
- What are the main ideas of using movement primitives?
- Why do use dynamical systems? Advantages/Disadvantages?
- Why do use a probabilistic representation?