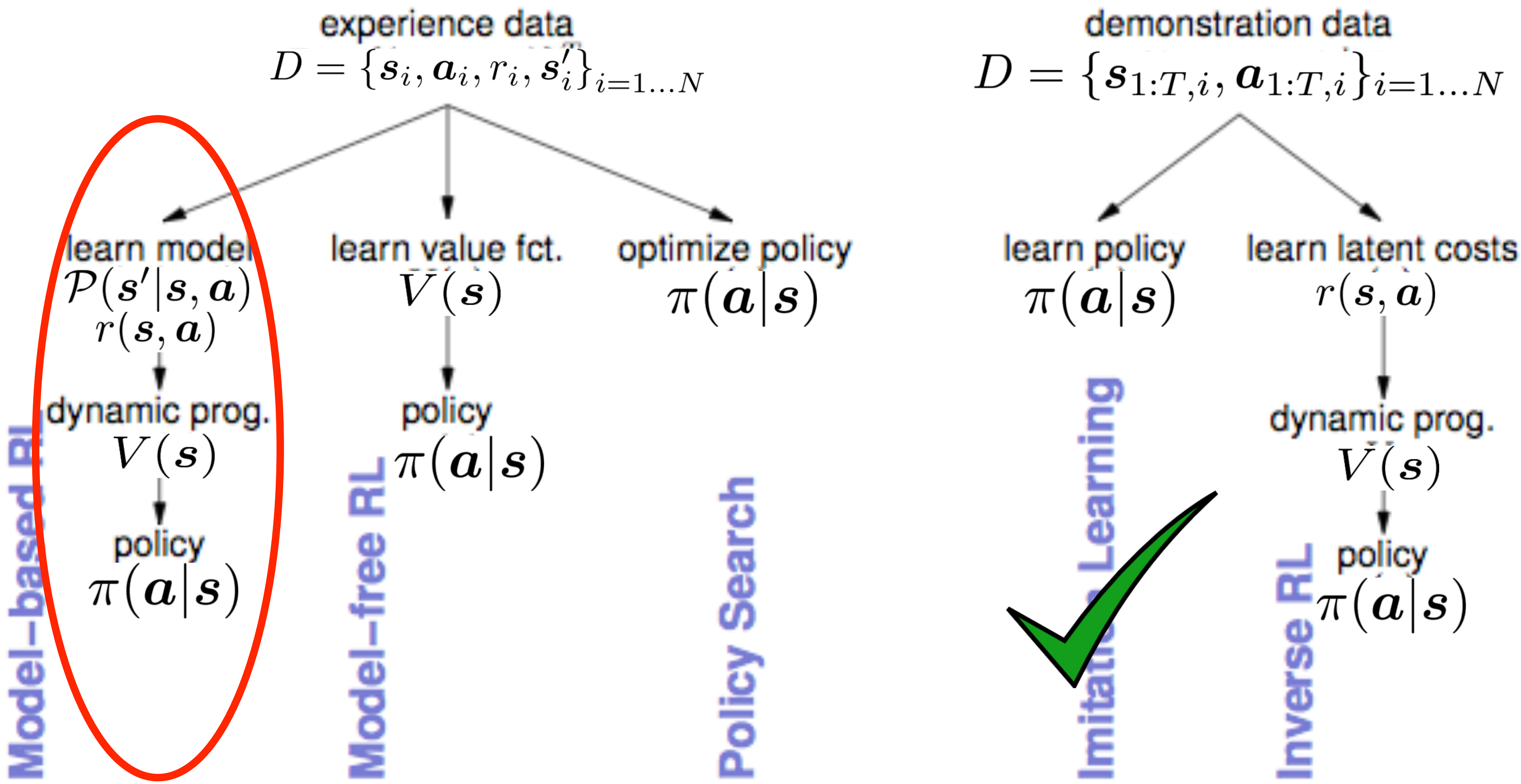
Reinforcement Learning Part I: Optimal Control ...with Learned Models

Jan Peters
Gerhard Neumann

The Bigger Picture: How to learn policies



1. This Lecture 2. 3. 4.

Motivation

Today we want to use optimal decision making for a specific system

- Linear system, Quadratic Reward, Gaussian Noise
  - The optimal policy is called Linear Quadratic Regulator (LQR)
- Optimal Decision making for continuous dynamical systems is also called Optimal Control

Why? Its the only continuous case where we can do it analytically…

If we do not know the model, we can learn it!

If the (learned) model is not linear, we can linearize it!

Outline of the Lecture

1. Optimal Control

2. Solving the Optimal Control for LQR systems

3. Approximating Non-Linear Systems

4. Optimal Control with Learned Models

5. Final Remarks

Optimal Decision Making in Continuous Systems

In continuous systems we call it Optimal Control

- continuous state space $\boldsymbol{s} \in \mathbb{R}^n$ (note: will be called $\boldsymbol{x}$)
- continuous action space $\boldsymbol{a} \in \mathbb{R}^m$ (note: same as $\boldsymbol{u}$)
- its transition dynamics as density

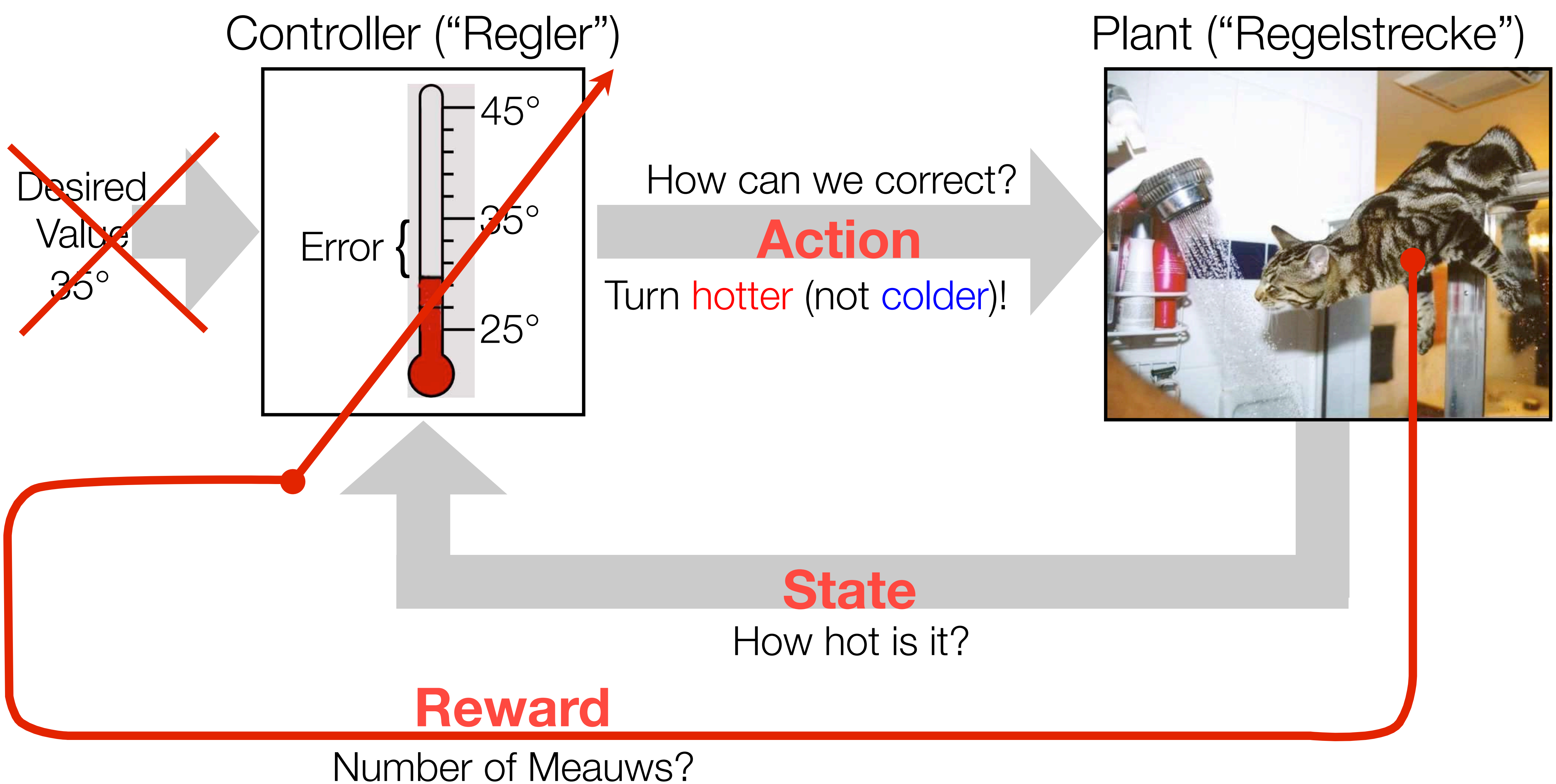
$$\mathcal{P}_t(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) = p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$$

➡ We use the more common optimal control notation

First question:

➡ How to define a reward for continuous systems?

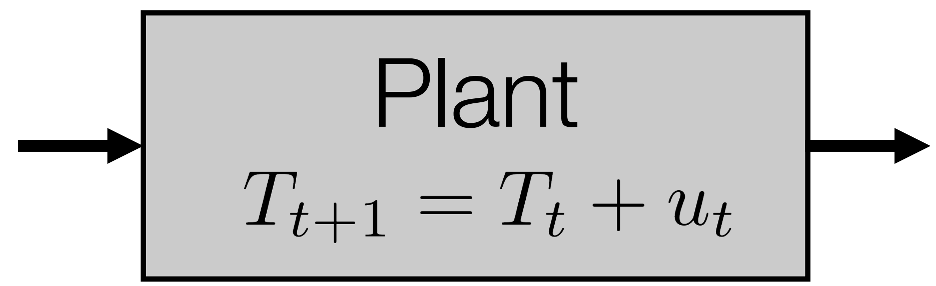
Illustration: Remember our “Showering Example”?



...but what would be a good reward?

Let's Model the System

The system



can be modeled as with

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = p(T_{t+1}|T_t, u_t) = \mathcal{N}(T_{t+1}|T_t + u_t, \sigma^2)$$

$$\text{with } \boldsymbol{x} = T, \ldots, \boldsymbol{u} = u$$

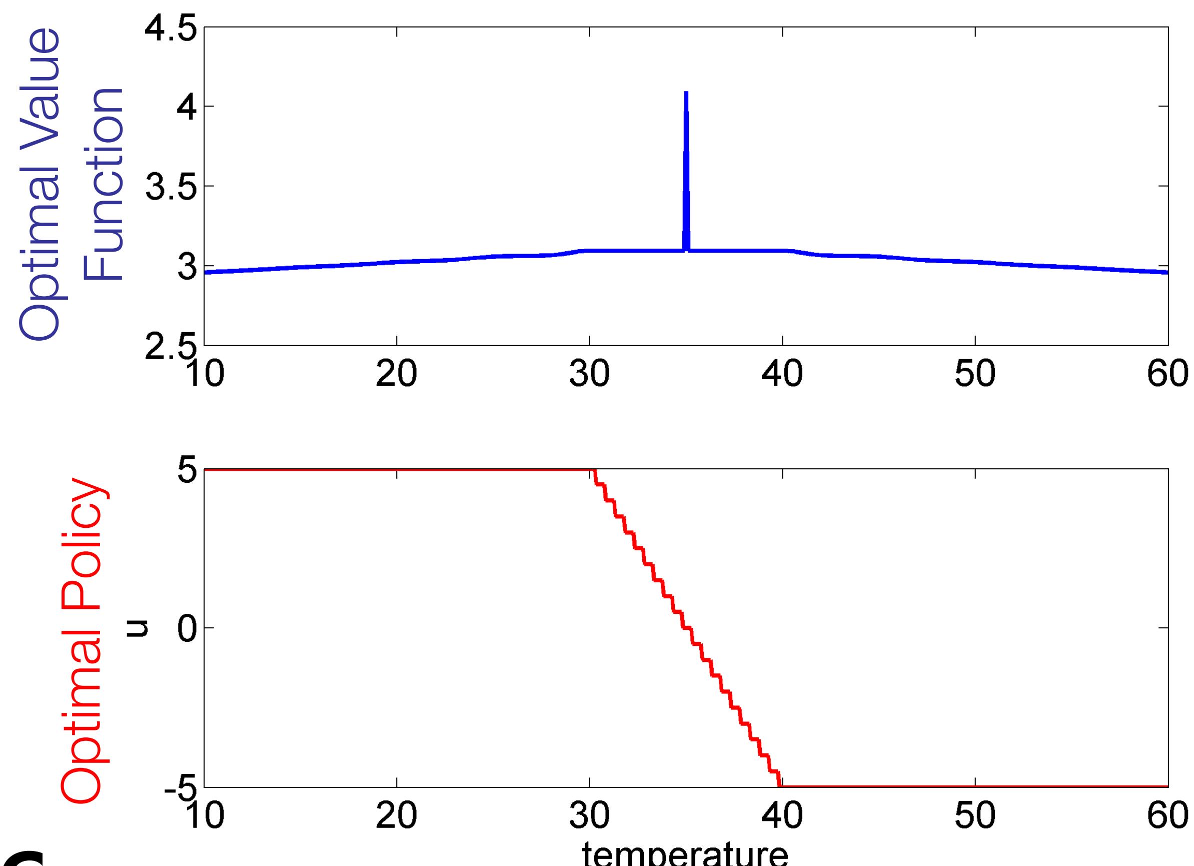
What kind of rewards induce which behavior?

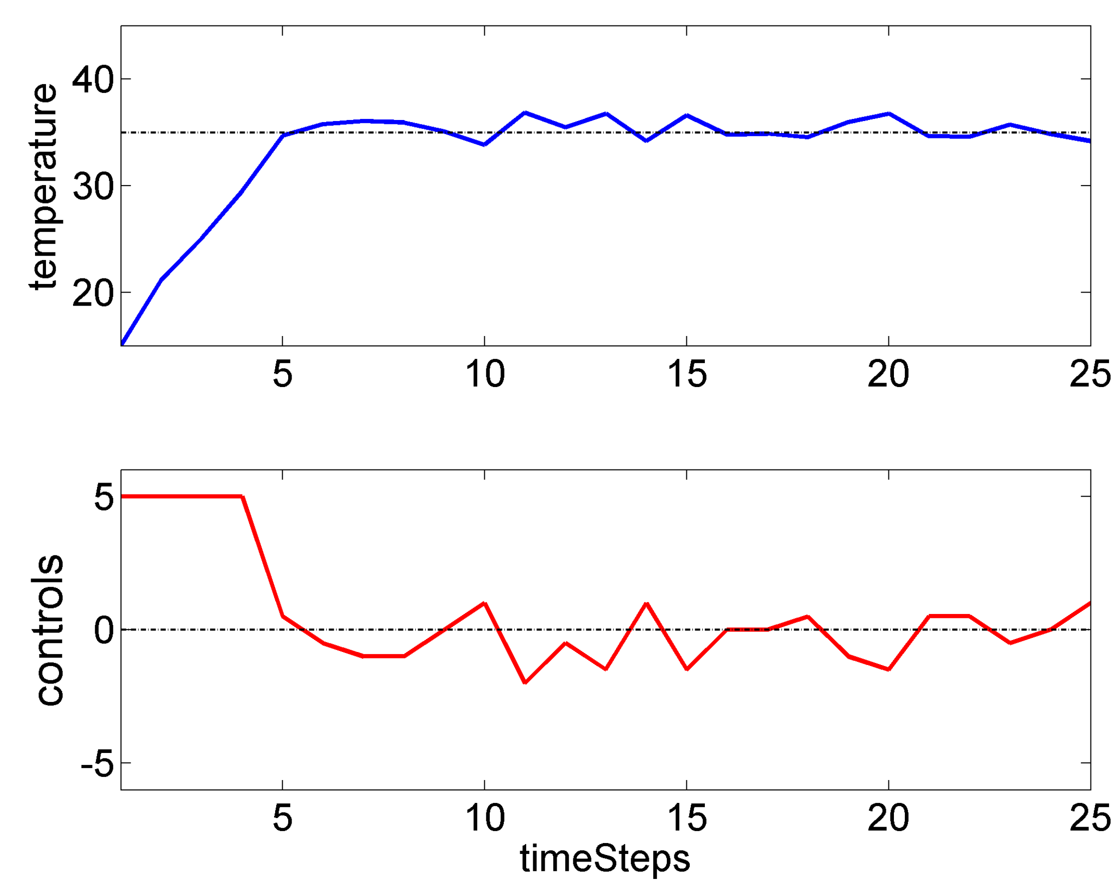
Let us find a reward!

Only give rewards to good temperatures:

$$r(T,u) = \begin{cases} 1, & \text{if } T = T_{\text{des}} \\ 0, & \text{otherwise} \end{cases}$$

How does the controller look? ➡ Rather jerky controls



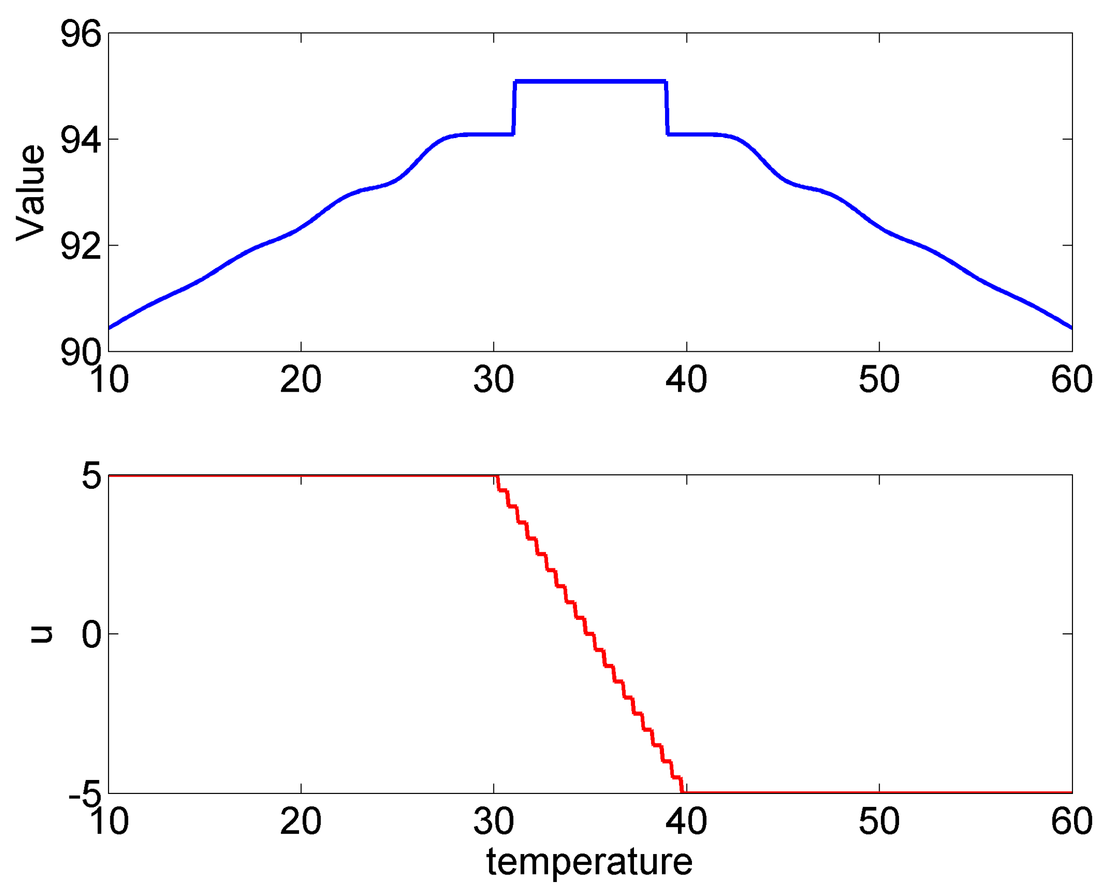


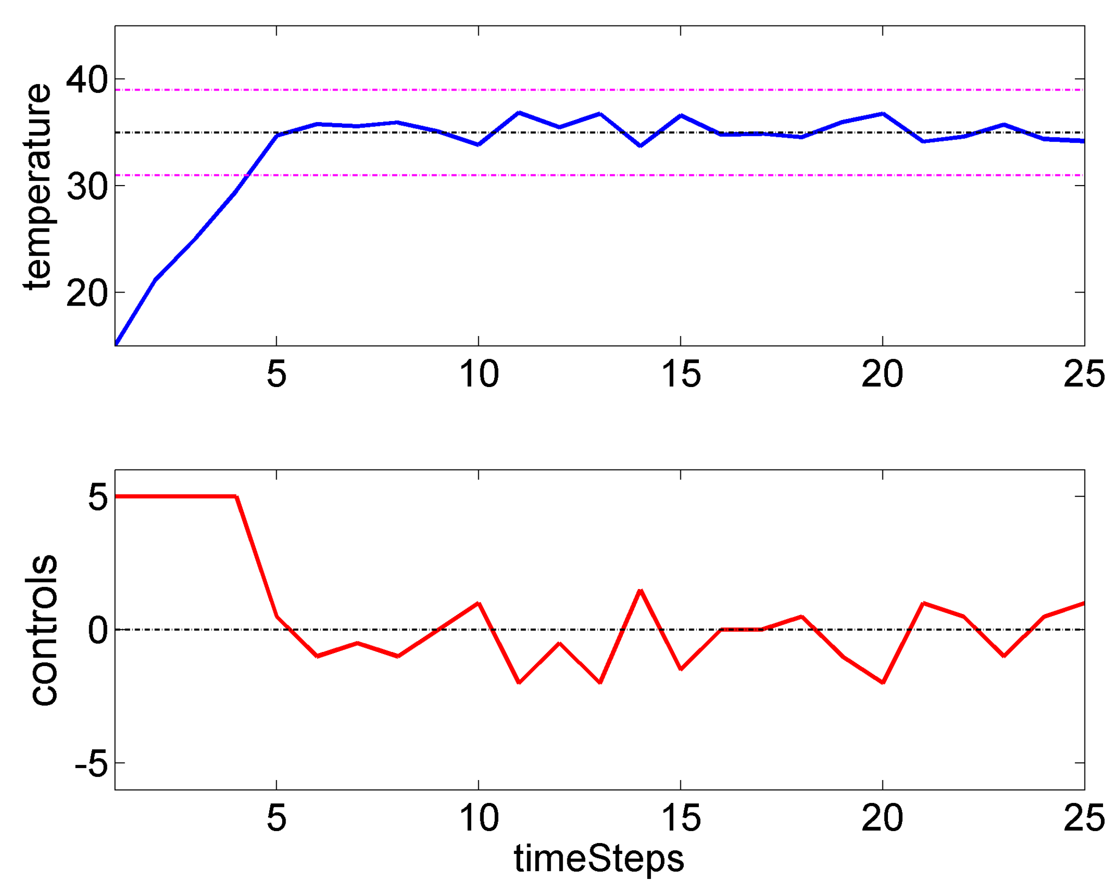
Let us find a reward!

We could enlarge the region:

$$r(T,u) = \begin{cases} 1, & \text{if } |T - T_{\text{des}}| < 4 \\ 0, & \text{otherwise} \end{cases}$$

How does the controller look? ➡ Rather jerky controls



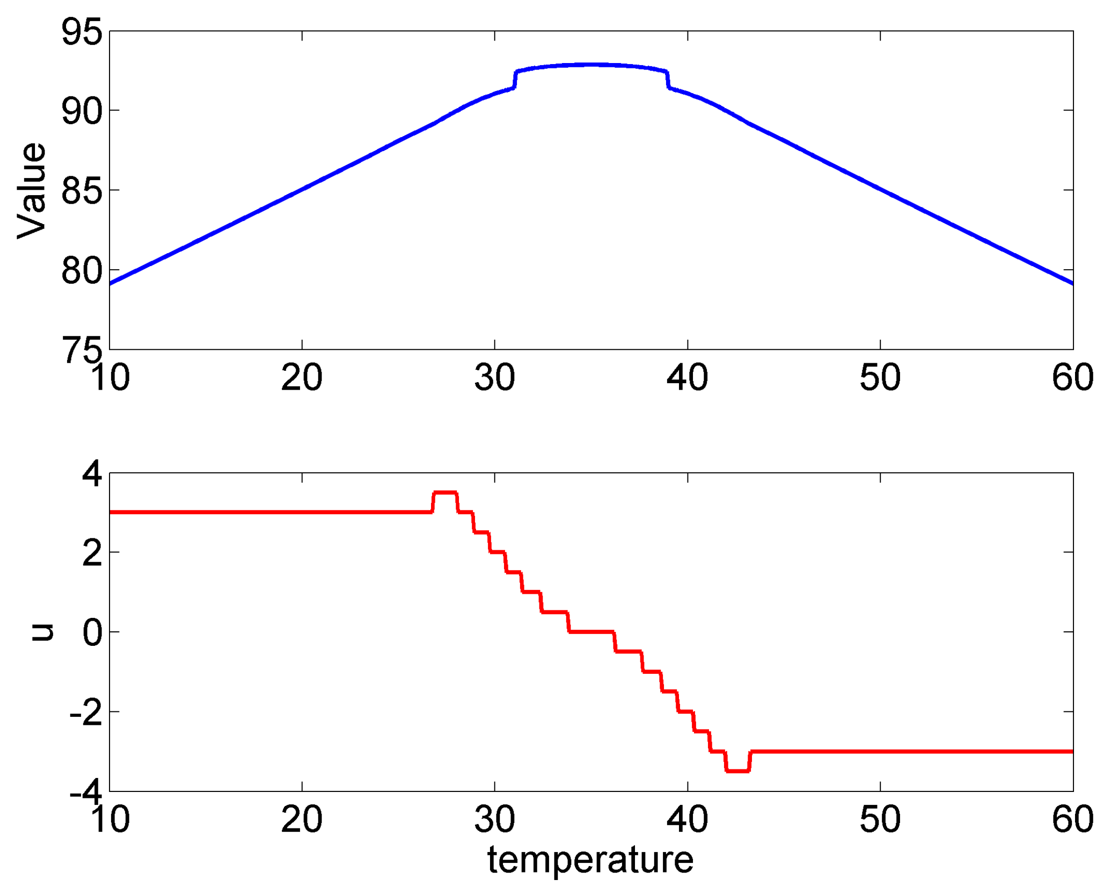


Let us find a reward!

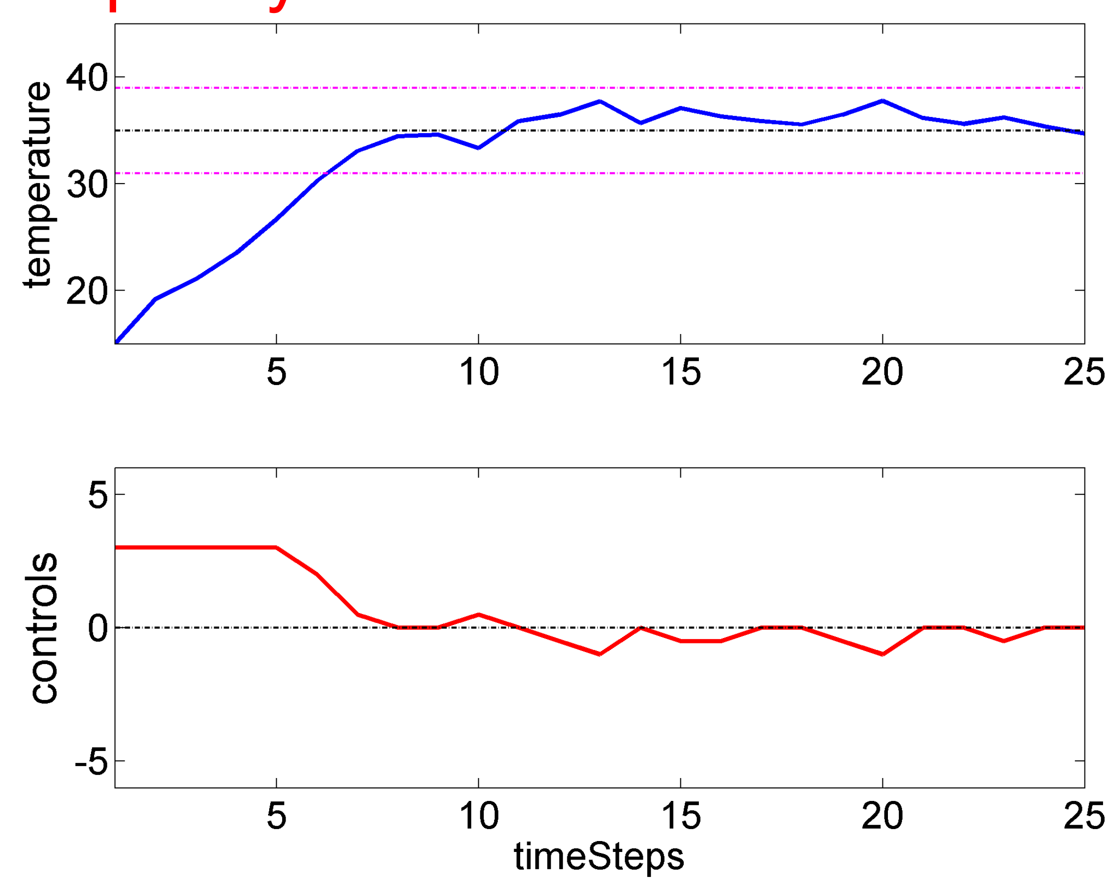
Punish if we need to turn the nob too much:

$$r(T,u) = \begin{cases} 1, & \text{if } |T - T_{\text{des}}| < 4 \\ -0.1u^2, & \text{otherwise} \end{cases}$$

How does the controller look?



Still complex value function and policy



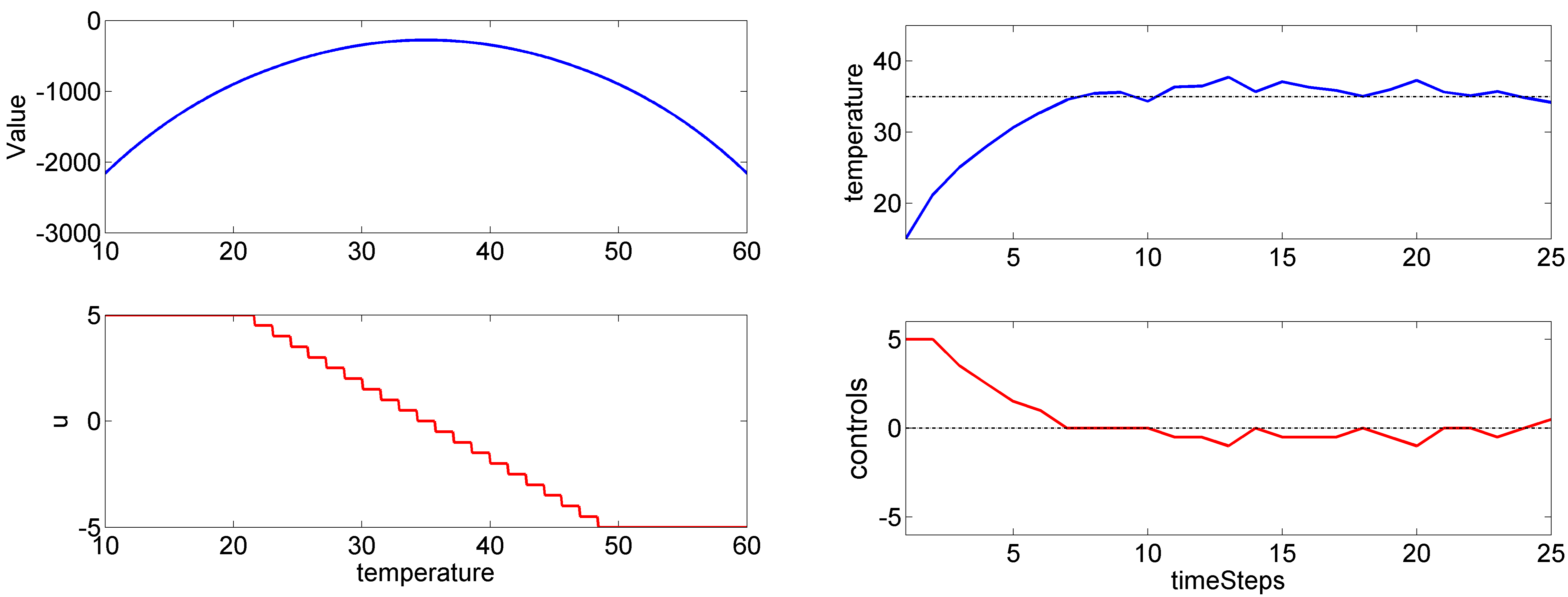
Let us find a reward!

Punish turning the nob and high deviations:

$$r(T, u) = -(T - T_{\text{des}})^2 - 5u^2$$

How does the controller look?

- Linear optimal controller
- Quadratic Value Function



Outline of the Lecture

1. **Optimal Control**
2. **Solving the Optimal Control for LQR systems**
3. **Approximating Non-Linear Systems**
4. **Optimal Control with Learned Models**
5. **Final Remarks**

Finite Horizon Objectives

The goal of the agent is to find a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ that maximizes its expected return $J_{\boldsymbol{\pi}}$ **for a finite time horizon**

Finite Horizon T: Accumulated expected reward for T steps

$$J_{\boldsymbol{\pi}} = \mathbb{E}_{\mu_0, \mathcal{P}, \pi}\left[\sum_{t=1}^{T-1} r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + r_T(\boldsymbol{s}_T)\right]$$

$r_T(\boldsymbol{s}_T)$ … final reward

Linear Quadratic Gaussian systems

An LQR system is defined as

- its state space $\boldsymbol{x} \in \mathbb{R}^n$ (note: same as $\boldsymbol{s}$)
- its action space $\boldsymbol{u} \in \mathbb{R}^m$ (note: same as $\boldsymbol{u}$)
- its (possibly time-dependent) linear transition dynamics with Gaussian noise

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

- its quadratic reward function

$$r_t(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{r}_t)^T \boldsymbol{R}_t (\boldsymbol{x} - \boldsymbol{r}_t) + \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

$$r_T(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{r}_T)^T \boldsymbol{R}_T (\boldsymbol{x} - \boldsymbol{r}_T)$$

- and its initial state density

$$\mu_0(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$$

Optimal Control for LQR systems

Linear systems with Gaussian Noise

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

$\boldsymbol{A}_t$ … system matrix, $\boldsymbol{B}_t$ … control matrix, $\boldsymbol{b}_t$ … drift term

$\boldsymbol{\Sigma}_t$ … system noise

Quadratic reward functions

$$r_t(\boldsymbol{x}, \boldsymbol{u}) = -(\boldsymbol{x} - \boldsymbol{r}_t)^T \boldsymbol{R}_t (\boldsymbol{x} - \boldsymbol{r}_t) - \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

$$r_T(\boldsymbol{x}) = -(\boldsymbol{x} - \boldsymbol{r}_T)^T \boldsymbol{R}_T (\boldsymbol{x} - \boldsymbol{r}_T)$$

$\boldsymbol{r}_t$ … desired state, $\boldsymbol{R}_t$ … state metric for reward

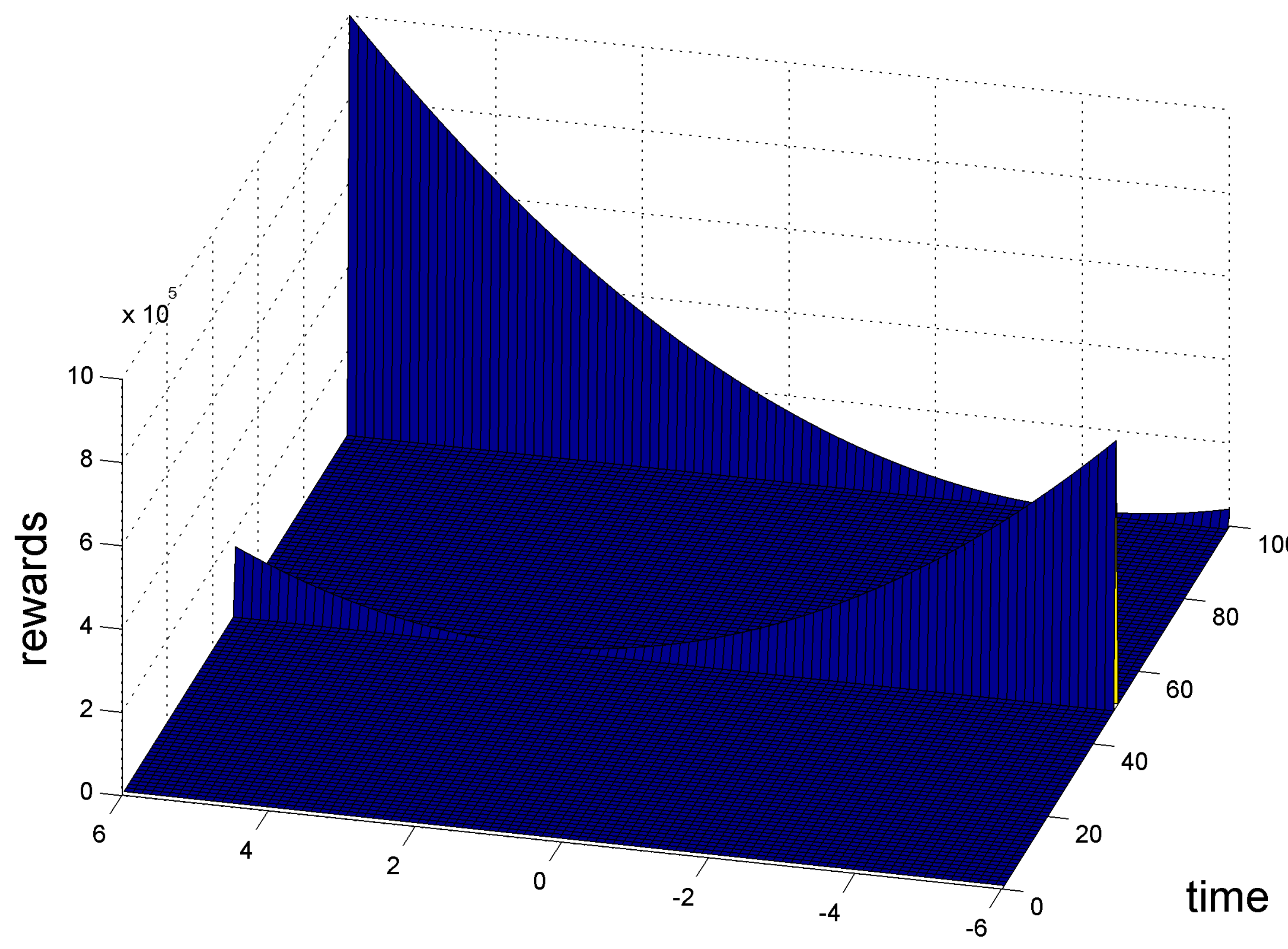
$\boldsymbol{H}_t$ … control metric for reward

Example

RewardFunction: Reach 2 Via-Points at $t_1 = 50$ and $t_2 = 100$

$$\boldsymbol{R}_{t_1,t_2} = \begin{bmatrix} 10^4 & 0 \\ 0 & 10^{-6} \end{bmatrix}, \text{ for all other } \tilde{t}, \boldsymbol{R}_{\tilde{t}} = \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}$$

$$\boldsymbol{r}_{t_1} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \boldsymbol{r}_{t_2} = \begin{bmatrix} -4 \\ 0 \end{bmatrix}$$



Optimal Control for Finite Horizon Objectives

We will look at the **simpler finite horizon case**

Short refresher from last lecture:

Start with last layer…

$$V_T^*(\boldsymbol{x}) = r_T(\boldsymbol{x})$$

Iterate **backwards in time**

$$V_t^*(\boldsymbol{x}) = \max_{\boldsymbol{u}} \left(r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \mathbb{E}_p \left[V_{t+1}^*(\boldsymbol{x}_{t+1}) | \boldsymbol{x}_t, \boldsymbol{u}_t \right] \right)$$

The optimal value function/policy for time step t is obtained after $T - t + 1$ iterations

$$V_T^*(\boldsymbol{x}) \Rightarrow V_{T-1}^*(\boldsymbol{x}) \Rightarrow \dots \Rightarrow V_1^*(\boldsymbol{x})$$

Optimal Control for LQR systems

We have to solve…

- Expectation over the next value:

$$\mathbb{E}_{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{u}_t)}\left[V^*_{t+1}(\boldsymbol{x}_{t+1})|\boldsymbol{x}_t,\boldsymbol{u}_t\right]$$

- Maximum operator in continuous action spaces:

$$\max_{\boldsymbol{u}}\left(r_t(\boldsymbol{x}_t,\boldsymbol{u}_t)+\mathbb{E}_p\left[V^*_{t+1}(\boldsymbol{x}_{t+1})|\boldsymbol{x}_t,\boldsymbol{u}_t\right]\right)$$

When can we do that?

In continuous systems: only for LQR systems!

Ok, lets solve the optimal control problem

For illustration, lets make it simpler (without any linear terms)…

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \cancel{\boldsymbol{b}_t}, \boldsymbol{\Sigma}_t)$$

$$r_t(\boldsymbol{x}, \boldsymbol{u}) = -(\boldsymbol{x} \cancel{- \boldsymbol{r}_t})^T \boldsymbol{R}_t(\boldsymbol{x} \cancel{- \boldsymbol{r}_t}) - \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

$$r_T(\boldsymbol{x}) = -(\boldsymbol{x} \cancel{- \boldsymbol{r}_T})^T \boldsymbol{R}_T(\boldsymbol{x} \cancel{- \boldsymbol{r}_T})$$

⇨

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t, \boldsymbol{\Sigma}_t)$$

$$r_t(\boldsymbol{x}, \boldsymbol{u}) = -\boldsymbol{x}^T \boldsymbol{R}_t \boldsymbol{x} - \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

$$r_T(\boldsymbol{x}) = -\boldsymbol{x}^T \boldsymbol{R}_T \boldsymbol{x}$$

For the derivation of the full problem including the drift and linear terms in the reward, see
http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/soc.pdf

Bellman's Recipe

1. At the last step, the value function is given as

$$V_T^*(\boldsymbol{x}) = r_T(\boldsymbol{x}) = -\boldsymbol{x}^T \boldsymbol{R}_T \boldsymbol{x} = -\boldsymbol{x}^T \boldsymbol{V}_T \boldsymbol{x}$$

$$\Rightarrow \boldsymbol{V}_T = \boldsymbol{R}_T$$

2. To get from *t+1* to *t*, first compute the Q-Function

$$Q_t^*(\boldsymbol{x}_t, \boldsymbol{u}_t) = r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \mathbb{E}_p\left[V_{t+1}^*(\boldsymbol{x}_{t+1}) | \boldsymbol{x}_t, \boldsymbol{u}_t\right]$$

3. then compute optimal policy π_t^*

$$\pi_t^*(\boldsymbol{x}) = \operatorname{argmax}_{\boldsymbol{u}} Q_t^*(\boldsymbol{x}, \boldsymbol{u})$$

4. compute optimal value function for time step t

$$V_t^*(\boldsymbol{x}) = Q_t^*(\boldsymbol{x}_t, \pi^*(\boldsymbol{x}))$$

Bellman's Recipe

Step 2a: compute expectation for value of the next state

- Lets assume for now a quadratic structure for V-function of next time step

$$V^*_{t+1}(\boldsymbol{x}) = -\boldsymbol{x}^T \boldsymbol{V}_{t+1} \boldsymbol{x}$$

- We need to compute:

$$\mathbb{E}_p\left[V^*_{t+1}(\boldsymbol{x}_{t+1}) | \boldsymbol{x}_t, \boldsymbol{u}_t\right] = -\int \mathcal{N}(\boldsymbol{x}_{t+1} | \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t, \boldsymbol{\Sigma}_t) \boldsymbol{x}^T_{t+1} \boldsymbol{V}_{t+1} \boldsymbol{x}_{t+1} d\boldsymbol{x}_{t+1}$$

Yet another useful Gaussian identity: 2nd order expectation

if $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ than $\mathbb{E}_p[\boldsymbol{x}^T \boldsymbol{M} \boldsymbol{x}] = \boldsymbol{\mu}^T \boldsymbol{M} \boldsymbol{\mu} + \mathrm{Tr}(\boldsymbol{M}\boldsymbol{\Sigma})$

- This identity yields

$$\mathbb{E}_p\left[V^*_{t+1}(\boldsymbol{x}_{t+1}) | \boldsymbol{x}_t, \boldsymbol{u}_t\right] = -(\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t)^T \boldsymbol{V}_{t+1} (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B} \boldsymbol{u}_t) + \mathrm{Tr}(\boldsymbol{V}_{t+1} \boldsymbol{\Sigma}_t)$$

Bellman's Recipe

Step 2b: Compute Q-function

$$
\begin{aligned}
Q_t^*(\boldsymbol{x}_t, \boldsymbol{u}_t) =& r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \mathbb{E}_p\left[V_{t+1}^*(\boldsymbol{x}_{t+1}) | \boldsymbol{x}_t, \boldsymbol{u}_t\right] \\
=& -\boldsymbol{x}^T \boldsymbol{R}_t \boldsymbol{x} - \boldsymbol{u}^T \boldsymbol{H}_t \boldsymbol{u} \\
& - (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t)^T \boldsymbol{V}_{t+1} (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B} \boldsymbol{u}_t) + \mathrm{Tr}(\boldsymbol{V}_{t+1} \boldsymbol{\Sigma}_t)
\end{aligned}
$$

Not state or action dependent

Also the Q-function is quadratic in state and action!

Bellman's Recipe

Step 3: compute optimal policy

$$\pi_t^*(\boldsymbol{x}) = \operatorname{argmax}_{\boldsymbol{u}} Q_t^*(\boldsymbol{x}, \boldsymbol{u})$$

Set **derivation to zero…**

$$\mathbf{0}^T = \frac{d}{d\boldsymbol{u}} \left(-\boldsymbol{x}_t^T \boldsymbol{R}_t \boldsymbol{x}_t - \boldsymbol{u}^T \boldsymbol{H}_t \boldsymbol{u} - (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u})^T \boldsymbol{V}_{t+1} (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}) \right)$$

Remember **matrix calculus…?**

$$\mathbf{0}^T = -2\boldsymbol{u}^T \boldsymbol{H}_t - 2(\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t)^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t$$

$$\mathbf{0}^T = -\boldsymbol{u}^T (\boldsymbol{H}_t + \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t) - \boldsymbol{x}_t^T \boldsymbol{A}^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t$$

And solve for u

$$\pi^*(\boldsymbol{x}_t) = \boldsymbol{u}^* = -(\boldsymbol{H}_t + \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t)^{-1} \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t \boldsymbol{x}_t = \boldsymbol{K}_t \boldsymbol{x}_t$$

➡ The optimal policy a time-varying linear (PD) controller!

Bellman's Recipe

Step 4: compute value function

$$
\begin{aligned}
V_t^*(\boldsymbol{x}) = &- \boldsymbol{x}_t^T \boldsymbol{R}_t \boldsymbol{x}_t - \boldsymbol{u}_t^{*T} \boldsymbol{H}_t \boldsymbol{u}_t^* - (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t^*)^T \boldsymbol{V}_{t+1} (\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t^*) \\
= &- \boldsymbol{x}_t^T (\boldsymbol{R}_t + \boldsymbol{A}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t) \boldsymbol{x}_t - \boldsymbol{u}_t^{*T} (\boldsymbol{H}_t + \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t) \boldsymbol{u}_t^* \\
&- 2 \boldsymbol{u}_t^{*T} \boldsymbol{B}_t \boldsymbol{V}_{t+1} \boldsymbol{A}_t \boldsymbol{x}_t
\end{aligned}
$$

We first set in the optimal action for $\boldsymbol{u}_t^* = -(\boldsymbol{H}_t + \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t)^{-1} \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t \boldsymbol{x}_t$

$$
V_t^*(\boldsymbol{x}) = - \boldsymbol{x}_t^T (\boldsymbol{R}_t + \boldsymbol{A}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t) \boldsymbol{x}_t - \boldsymbol{u}_t^{*T} \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t \boldsymbol{x}_t
$$

Now we can substitute $\boldsymbol{u}_t^* = \boldsymbol{K}_t \boldsymbol{x}_t$

$$
V_t^*(\boldsymbol{x}) = - \boldsymbol{x}_t^T (\boldsymbol{R}_t + \boldsymbol{A}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t + \boldsymbol{K}_t^T \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t) \boldsymbol{x}_t
$$

Note: this derivation works only if the matrices $\boldsymbol{V}_{t+1}, \boldsymbol{H}_t$ and $\boldsymbol{R}_t$ are positve definite (and hence symmetric), can always be garantueed

Bellman's Recipe

Step 4: compute value function

- The optimal value function $\boldsymbol{V}_t$ for time step *t+1* is also quadratic

$$V_t^*(\boldsymbol{x}) = -\boldsymbol{x}_t^T \boldsymbol{V}_t \boldsymbol{x}_t$$

- we ended up in a recursive update equation for

$$\begin{aligned} \boldsymbol{V}_t &= \boldsymbol{R}_t + \boldsymbol{A}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t + \boldsymbol{K}_t^T \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t \\ &= \boldsymbol{R}_t + (\boldsymbol{A}_t + \boldsymbol{B}_t \boldsymbol{K}_t)^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t \end{aligned}$$

$$\text{with } \boldsymbol{K}_t = -(\boldsymbol{H}_t + \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{B}_t)^{-1} \boldsymbol{B}_t^T \boldsymbol{V}_{t+1} \boldsymbol{A}_t$$

- if $V_{t+1}(\boldsymbol{s})$ is in quadratic form, $V_t(\boldsymbol{s})$ also is

- since $V_T(\boldsymbol{s})$ is quadratic, all $V_t(\boldsymbol{s})$ are quadratic

Solving optimal control

So how does the full case look like?

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

$$r_t(\boldsymbol{x}, \boldsymbol{u}) = -(\boldsymbol{x} - \boldsymbol{r}_t)^T \boldsymbol{R}_t(\boldsymbol{x} - \boldsymbol{r}_t) - \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

The optimal value function has a quadratic and linear form

$$V_t(\boldsymbol{x}_t) = -\boldsymbol{x}_t^T \boldsymbol{V}_t \boldsymbol{x}_t + 2\boldsymbol{v}_t^T \boldsymbol{x}_t + \text{const}$$

With the update rules:

$$\boldsymbol{V}_t = \boldsymbol{R}_t + (\boldsymbol{A}_t + \boldsymbol{B}_t\boldsymbol{K}_t)^T \boldsymbol{V}_{t+1}\boldsymbol{A}_t \text{ (same as before)}$$

$$\boldsymbol{v}_t = \tilde{\boldsymbol{r}}_t + (\boldsymbol{A}_t + \boldsymbol{B}_t\boldsymbol{K}_t)^T (\boldsymbol{v}_{t+1} - \boldsymbol{V}_{t+1}\boldsymbol{b}_t)$$

with $\tilde{\boldsymbol{r}}_t = \boldsymbol{r}_t^T \boldsymbol{R}_t$ and $\boldsymbol{K}_t = -(\boldsymbol{H}_t + \boldsymbol{B}_t^T \boldsymbol{V}_{t+1}\boldsymbol{B}_t)^{-1} \boldsymbol{B}_t^T \boldsymbol{V}_{t+1}\boldsymbol{A}_t$

For the derivation of the full problem including the drift and linear terms in the reward, see http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/soc.pdf

Solving optimal control

So how does the full case look like?

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

$$r_t(\boldsymbol{x},\boldsymbol{u}) = -(\boldsymbol{x} - \boldsymbol{r}_t)^T\boldsymbol{R}_t(\boldsymbol{x} - \boldsymbol{r}_t) - \boldsymbol{u}_t^T\boldsymbol{H}_t\boldsymbol{u}_t$$

The optimal policy is given by

$$\begin{aligned}\boldsymbol{u}^* &= -\left(\boldsymbol{H}_t + \boldsymbol{B}_t^T\boldsymbol{V}_{t+1}\boldsymbol{B}_t\right)^{-1}\boldsymbol{B}_t^T\left(\boldsymbol{V}_{t+1}(\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{b}_t) - \boldsymbol{v}_{t+1}\right) \\ &= \boldsymbol{K}_t\boldsymbol{x}_t + \boldsymbol{k}_t\end{aligned}$$

with $\boldsymbol{K}_t = -(\boldsymbol{H}_t + \boldsymbol{B}_t^T\boldsymbol{V}_{t+1}\boldsymbol{B}_t)^{-1}\boldsymbol{B}_t^T\boldsymbol{V}_{t+1}\boldsymbol{A}_t$

and $\boldsymbol{k}_t = -(\boldsymbol{H}_t + \boldsymbol{B}_t^T\boldsymbol{V}_{t+1}\boldsymbol{B}_t)^{-1}\boldsymbol{B}_t^T(\boldsymbol{V}_{t+1}\boldsymbol{b}_t - \boldsymbol{v}_{t+1})$

I.e. the optimal policy is a time-dependent linear feedback controller with time dependent offset

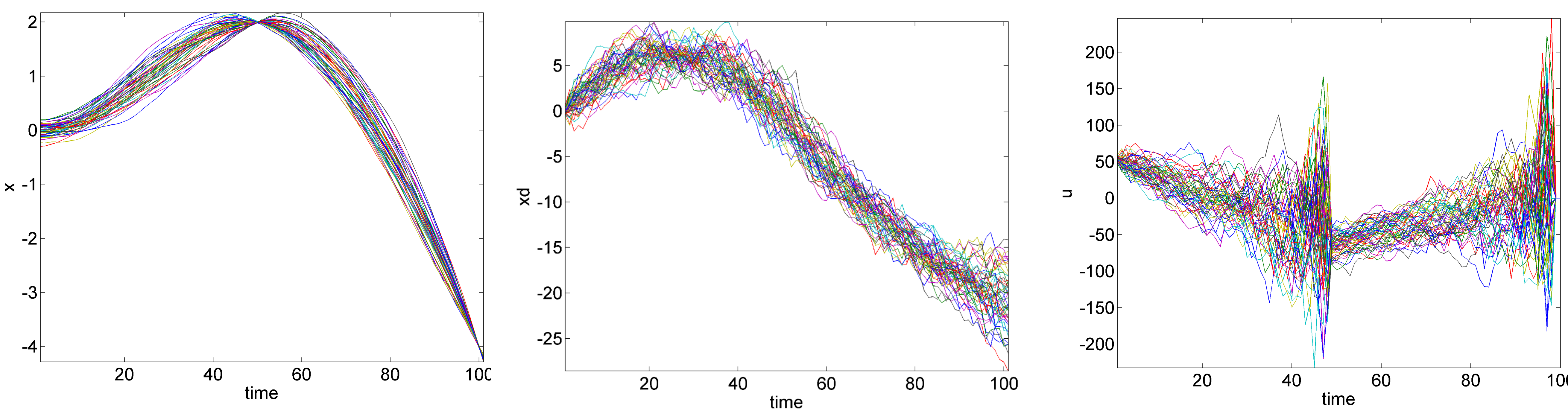
For the derivation of the full problem including the drift and linear terms in the reward, see http://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/soc.pdf

Back to the Example:

System:

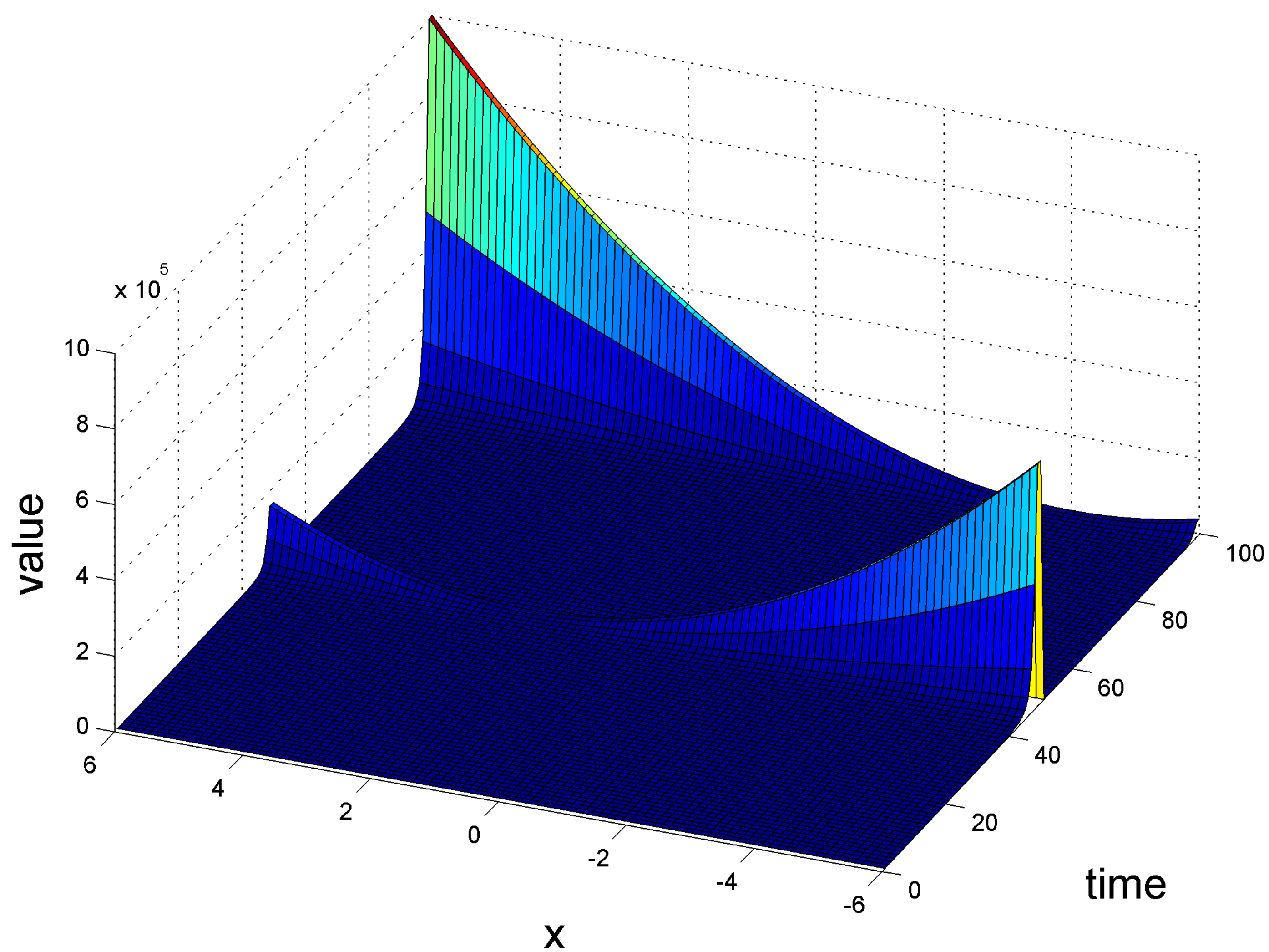
Second order integrator (we directly set accelerations)

$$\boldsymbol{x}_{t+1} = \underbrace{\begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}}_{\boldsymbol{A}} \boldsymbol{x}_t + \underbrace{\begin{bmatrix} 0 \\ dt \end{bmatrix}}_{\boldsymbol{B}} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} 0 & 0 \\ 0 & 0.5dt^2 \end{bmatrix}\right)$$



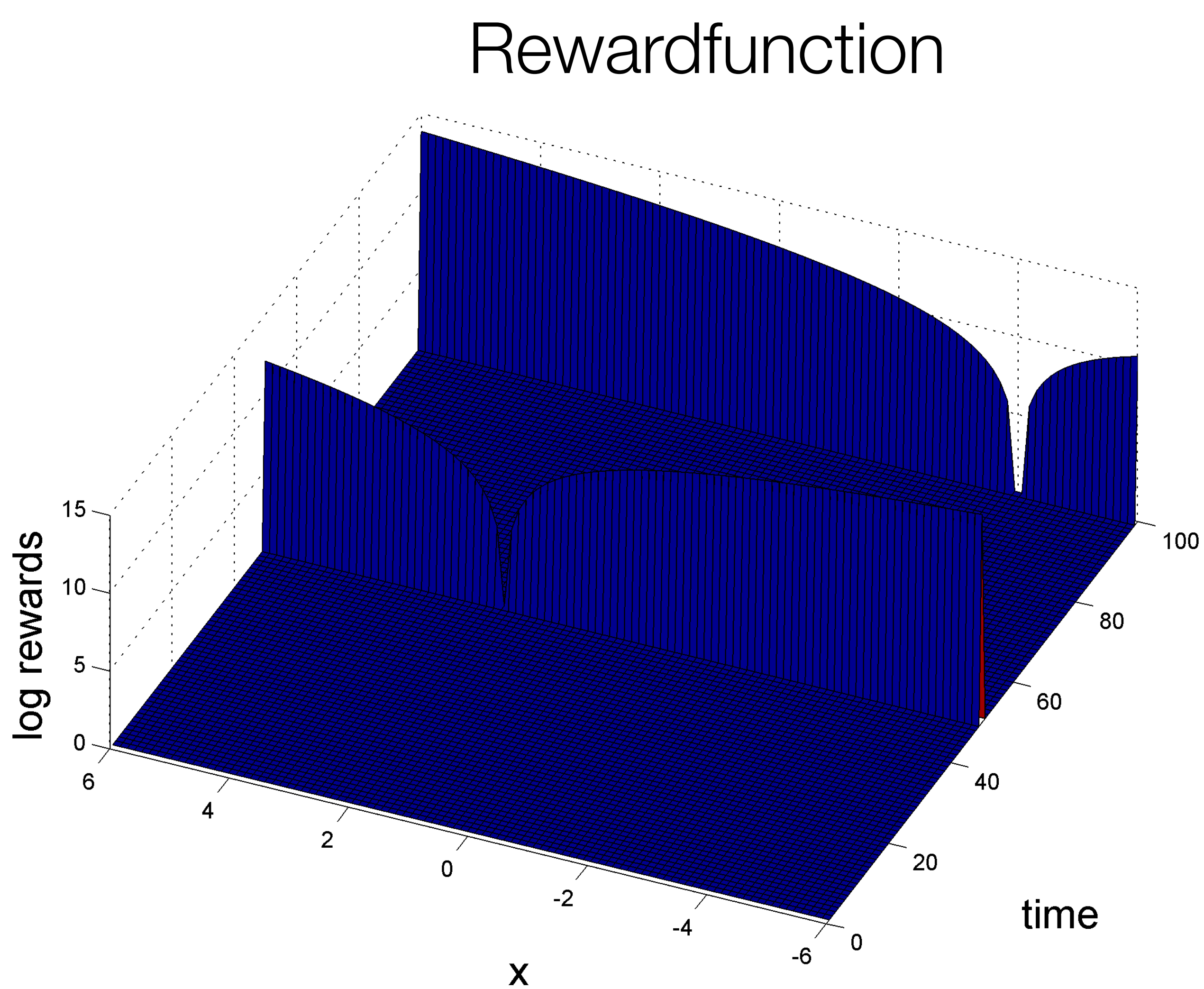
Optimal Control for LQR systems

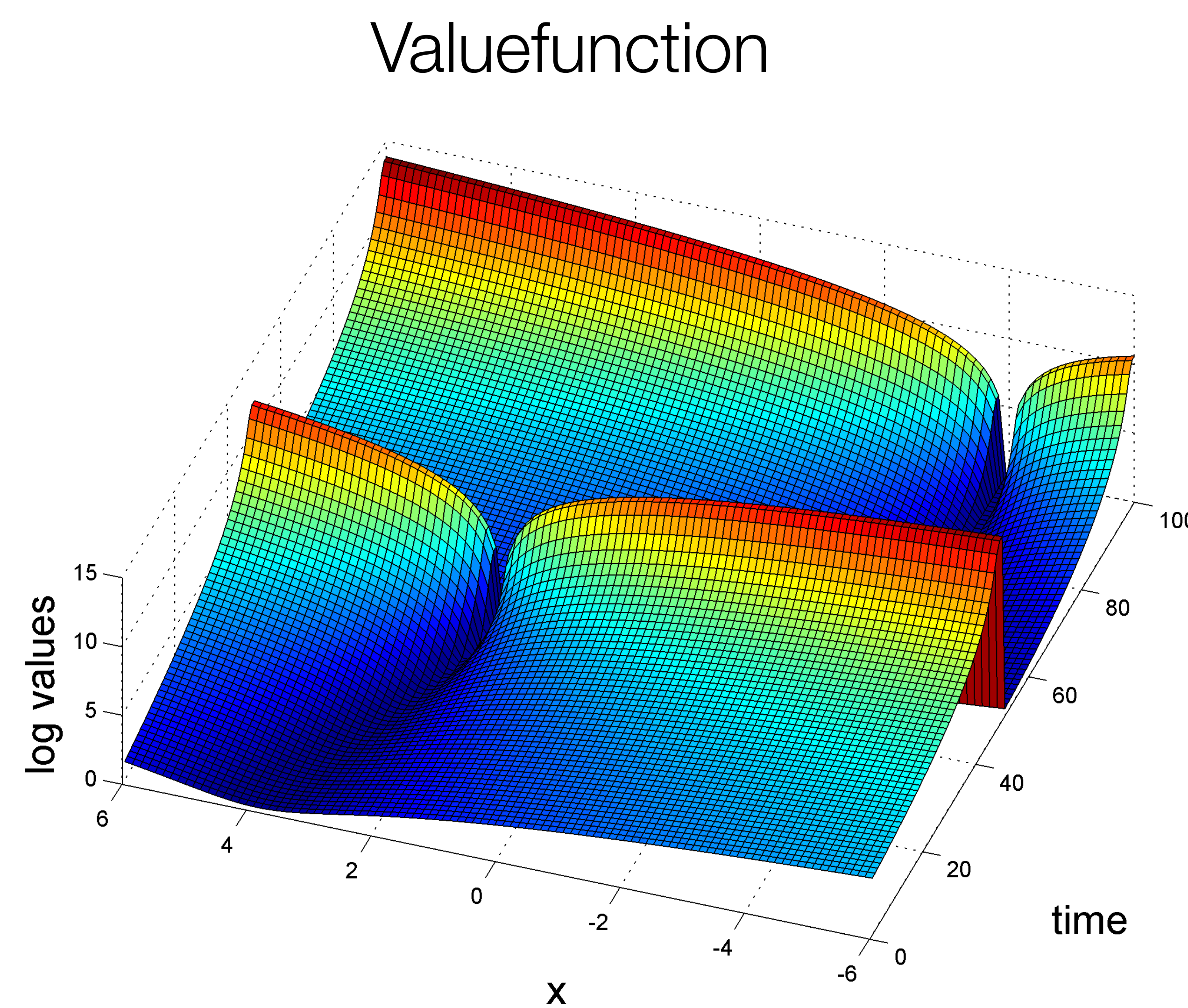
Illustration of the Value Function



Optimal Control for LQR systems

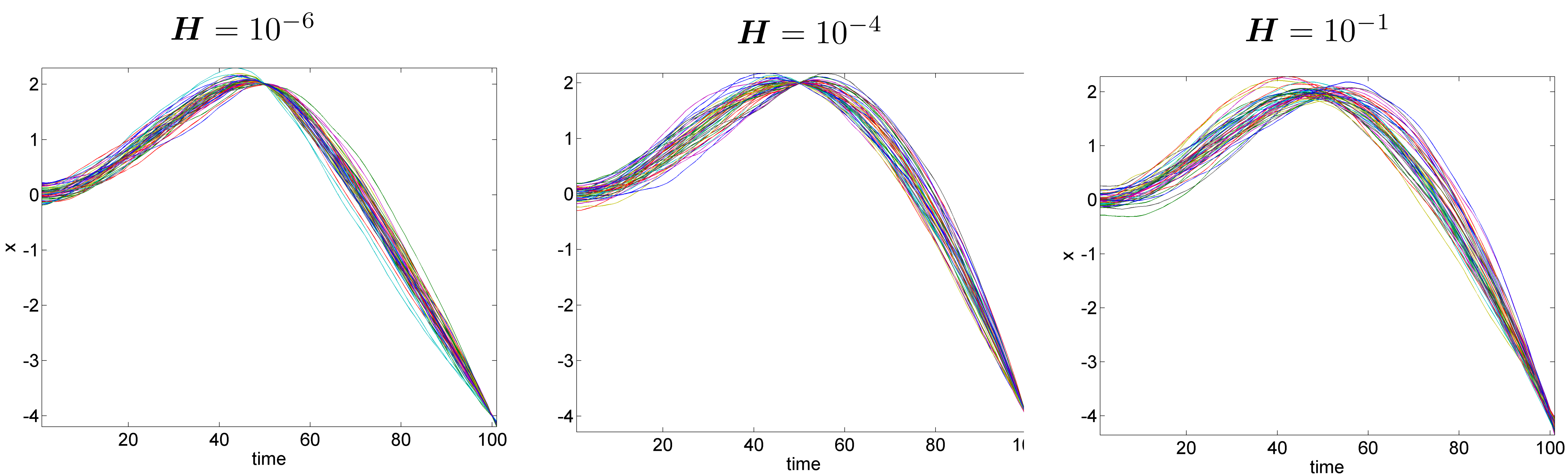
Comparison of Value and Reward Function (log domain)





Optimal Control for LQR systems

Different Control Costs



Outline of the Lecture

1. Optimal Control

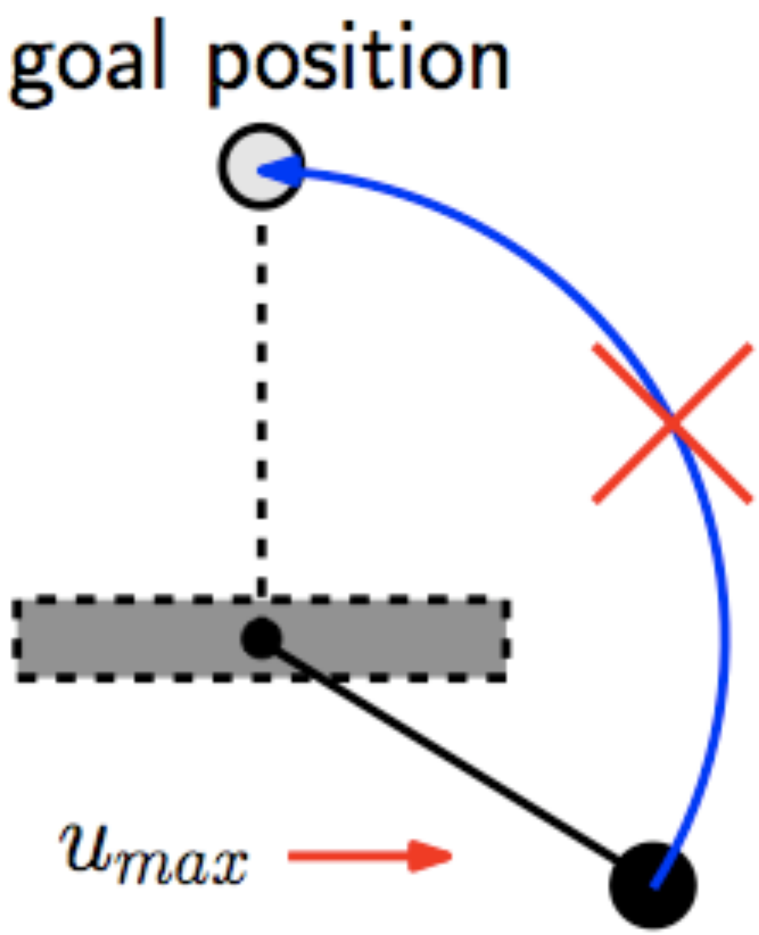
2. Solving the Optimal Control for LQR systems

3. Approximating Non-Linear Systems

4. Optimal Control with Learned Models

5. Final Remarks

Example for non-linear dynamics: Swing-Up



System

$$\ddot{\varphi}(t) = \frac{-\mu\dot{\varphi}(t) + mgl\sin(\varphi(t)) + u(t)}{ml^2}$$

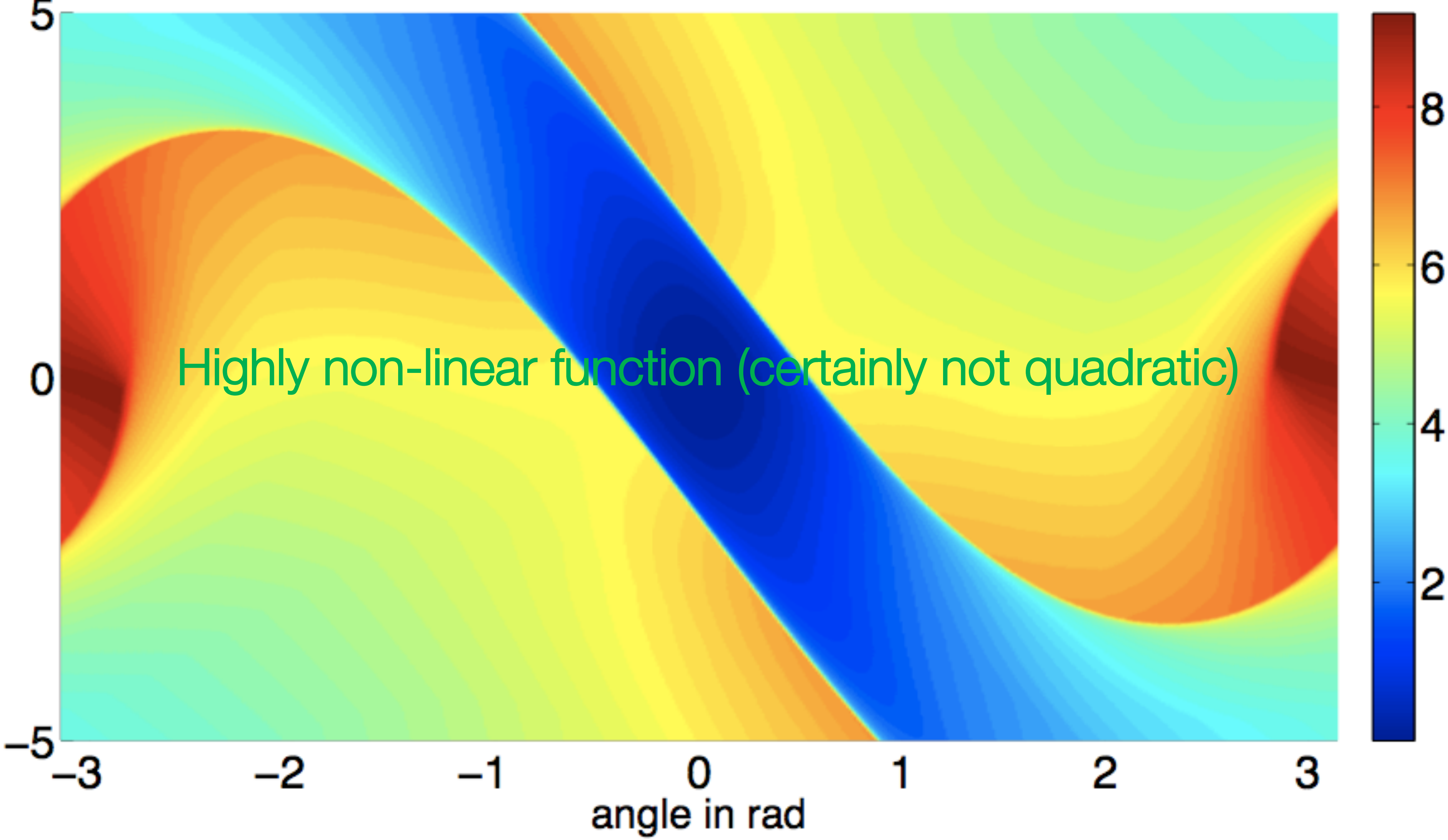
$$\mathbf{x}_{k+1} := \begin{bmatrix} \varphi_{k+1} \\ \dot{\varphi}_{k+1} \end{bmatrix} = \begin{bmatrix} \varphi_k + \Delta_t \dot{\varphi}_k + \frac{\Delta_t^2}{2}\ddot{\varphi}_k \\ \dot{\varphi}_k + \Delta_t \ddot{\varphi}_k \end{bmatrix}$$

Reward

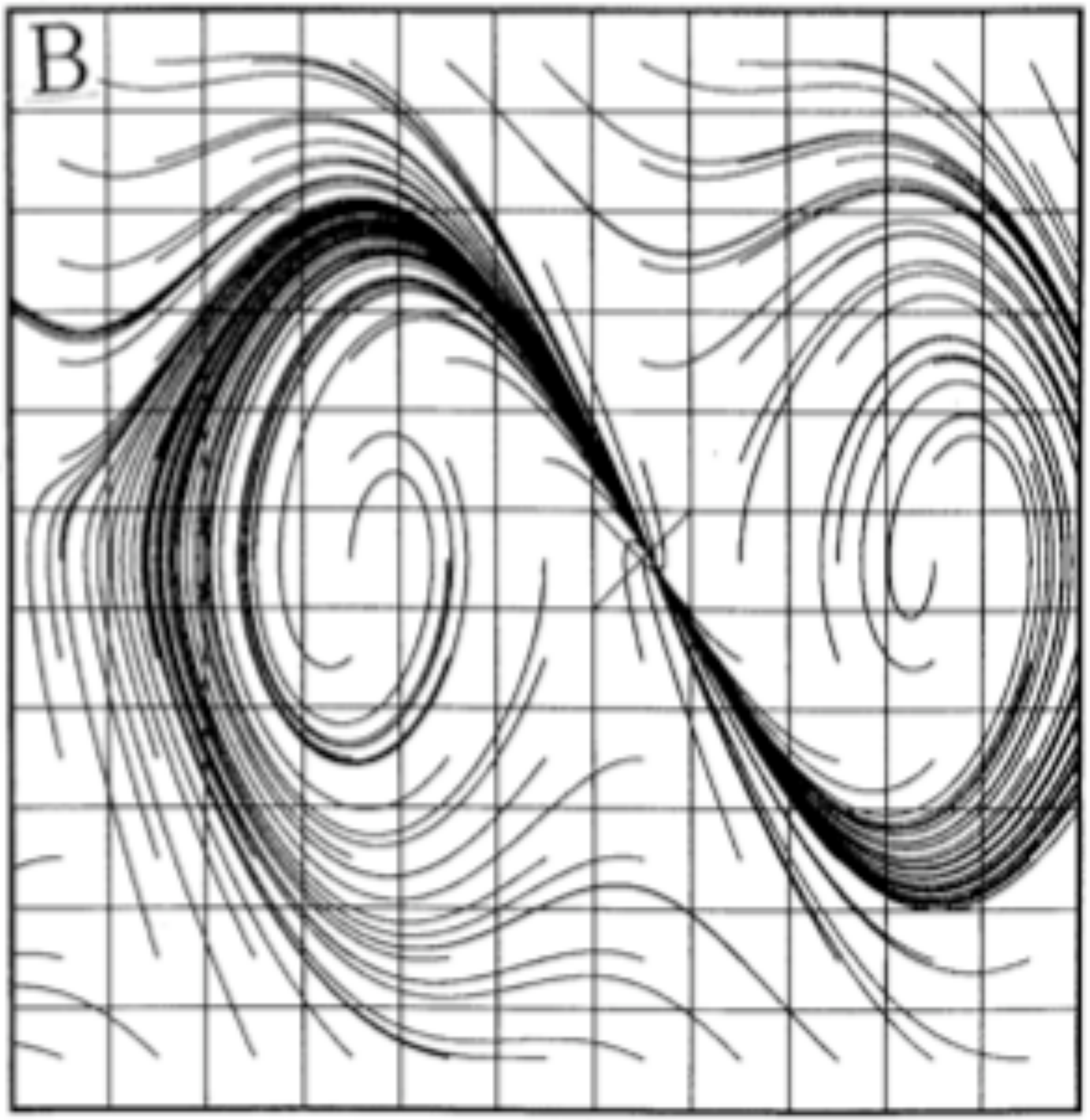
$$r(\boldsymbol{s}, a) = -\boldsymbol{s}^T \operatorname{diag}(1, 0.1)\boldsymbol{s} - 0.2a^2$$

Example: Value Function of Inverted Pendulum

Value function for the expected costs (negative reward)



Possible: Learn Solutions only where needed!



If you know places where we start...

... we can just look ahead and approximate the solution locally around an initial trajectory

Local Solutions by Linearizations

Every smooth function can be modeled with a Taylor expansion

$$f(\mathbf{x}) = f(\mathbf{a}) + \left.\frac{df}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}} (\mathbf{x}-\mathbf{a}) + \frac{1}{2}(\mathbf{x}-\mathbf{a})^T \left.\frac{d^2 f}{d\mathbf{x}^2}\right|_{\mathbf{x}=\mathbf{a}} (\mathbf{x}-\mathbf{a}) + \dots$$

Hence, we can also **approximate the (learned) forward dynamics by linearizing** at the point $(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t)$

$$\boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx f(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t) + \frac{df}{d\boldsymbol{s}}(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) + \frac{df}{d\boldsymbol{u}}(\boldsymbol{u}_t - \tilde{\boldsymbol{u}}_t)$$

$$= \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t + \boldsymbol{b}_t$$

with $\boldsymbol{A}_t = \left.\frac{df}{d\boldsymbol{x}}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}_t, \boldsymbol{u}=\tilde{\boldsymbol{u}}_t}$ and $\boldsymbol{B}_t = \left.\frac{df}{d\boldsymbol{u}}\right|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}_t, \boldsymbol{u}=\tilde{\boldsymbol{u}}_t}$

and $\boldsymbol{b}_t = f(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t) - \boldsymbol{A}_t \tilde{\boldsymbol{x}}_t - \boldsymbol{B}_t \tilde{\boldsymbol{u}}_t$

Local Solutions by Linearizations

Similarly, we can **approximate the (learned) reward function by a second order approximation** at the point $(\tilde{\boldsymbol{s}}_t, \tilde{\boldsymbol{a}}_t)$ (only shown for states)

$$r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) \approx r(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{u}}_t) + \frac{dr}{d\boldsymbol{x}}(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) + (\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t)^T \frac{dr}{d\boldsymbol{x} d\boldsymbol{x}}(\boldsymbol{x}_t - \tilde{\boldsymbol{x}}_t) - \boldsymbol{u}_t^T \boldsymbol{H}_t \boldsymbol{u}_t$$

$$= -\boldsymbol{x}^T \boldsymbol{R}_t \boldsymbol{x} + 2\boldsymbol{r}_t^T \boldsymbol{x} - \boldsymbol{u}^T \boldsymbol{H}_t \boldsymbol{u} + \text{const}$$

with $\boldsymbol{R}_t = -\frac{dr}{d\boldsymbol{x} d\boldsymbol{x}}$ and $\boldsymbol{r}_t = 0.5\frac{dr}{d\boldsymbol{x}} - \frac{dr}{d\boldsymbol{x} d\boldsymbol{x}}\tilde{\boldsymbol{x}}_t$

Local Solutions by Linearizations

So we are back to the **full linear optimal control case with**…

$$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t + \boldsymbol{b}_t, \boldsymbol{\Sigma}_t)$$

$$r(\boldsymbol{x}, \boldsymbol{u}) = -\boldsymbol{x}^T\boldsymbol{R}_t\boldsymbol{x} + 2\boldsymbol{r}_t^T\boldsymbol{x} - \boldsymbol{u}^T\boldsymbol{H}_t\boldsymbol{u} + \mathrm{const}$$

that we know how to solve…

Hence our algorithm for **solving non-linear optimal control** is…

1. **Backward Solution:** Compute optimal control law (i.e. Gains $\boldsymbol{K}_t$ and offsets $\boldsymbol{k}_t$

2. **Forward Propagation:** Run simulator with optimal control law to obtain linearization points $(\tilde{\boldsymbol{x}}_{1:T}, \tilde{\boldsymbol{u}}_{1:T})$

1.If not converged, go to 1.

Some interesting results (only in simulation)

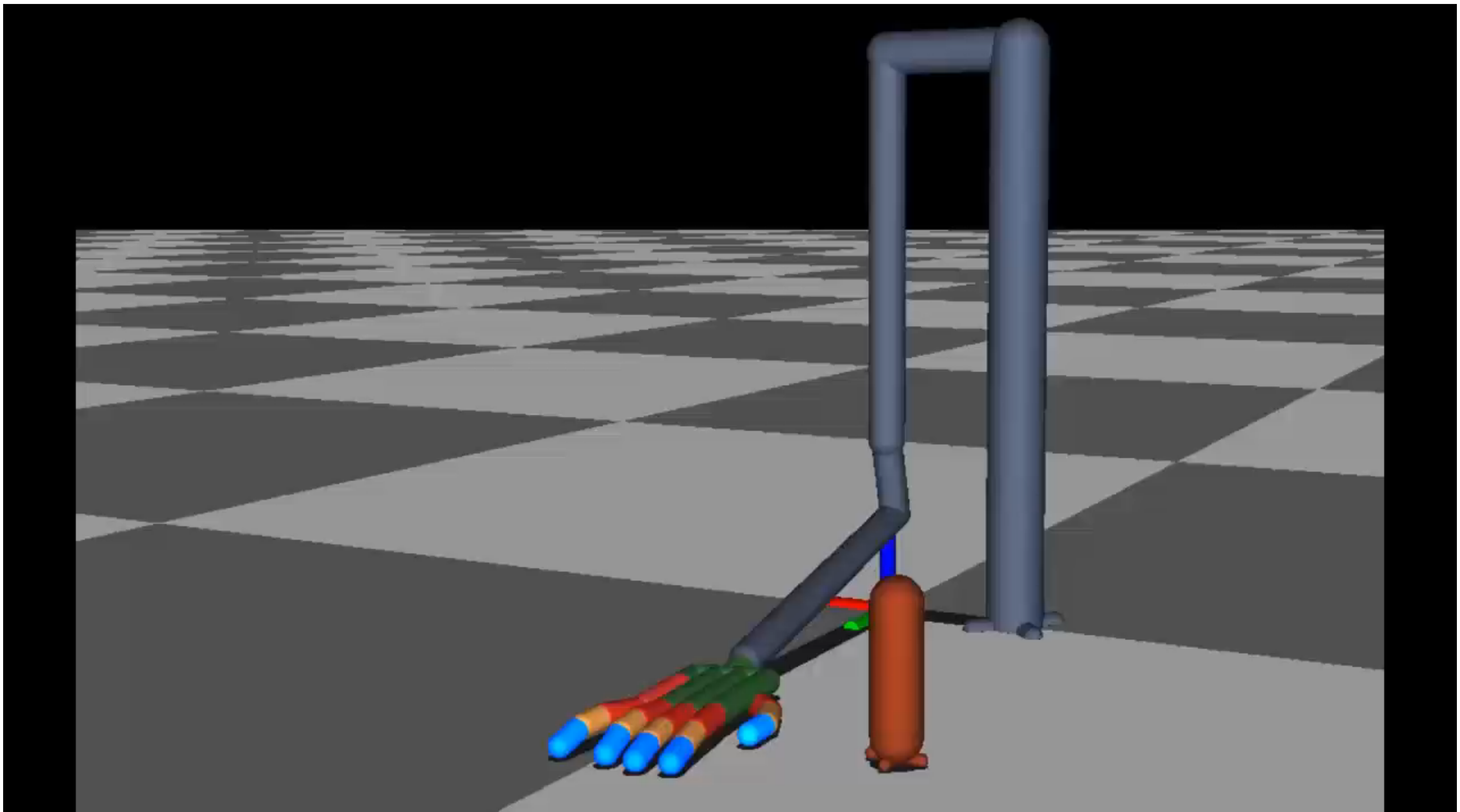
Work by Emo Todorov and Yuval Tassa
(They call basically the same algorithm incremential LQG, iLQG)

Synthesis of Complex Behaviors
with
Online Trajectory Optimization

(under review)

Application to the Swing-Up

Some interesting results (only in simulation)



Outline of the Lecture

1. Optimal Control

2. Solving the Optimal Control for LQR systems

3. Approximating Non-Linear Systems

4. Optimal Control with Learned Models

5. Final Remarks

Model Learning…

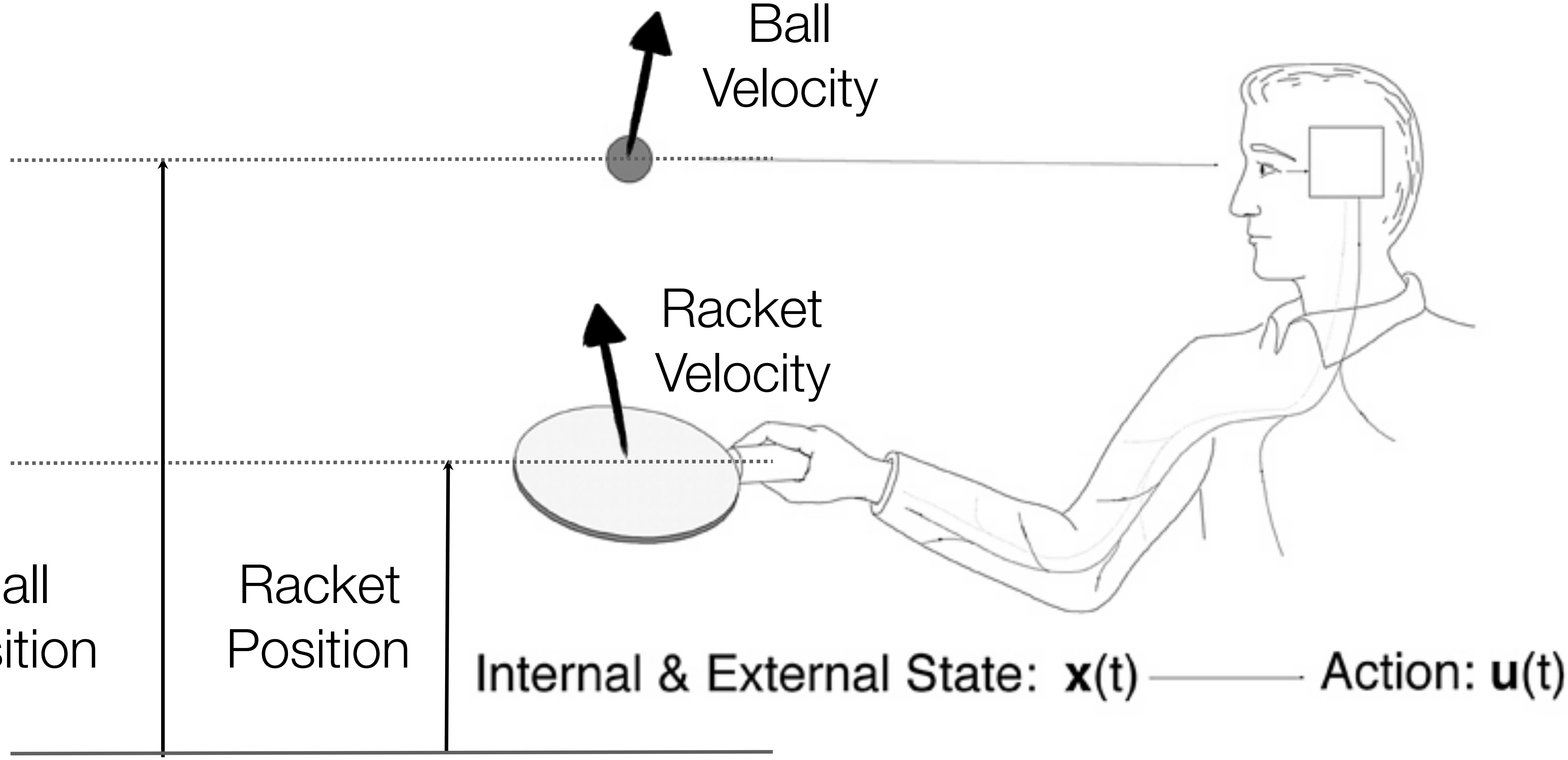
Why does this work only in simulation?

The models we have for such complex robots are… crap

We need to learn the models !

Example: Ball Paddling

What are the states x?



Example: Ball Paddling

What are the actions **u**?

All motor torques?

If you do not have an inverse model ...

Joint Accelerations?

Perfect, if you have a good inverse model ...

Maybe identify the proper degrees of freedom?

Accelerations in Task Space?

Ideally!

... but only if you have a good operational space control law!

Example: Ball Paddling

What are good rewards *r*?

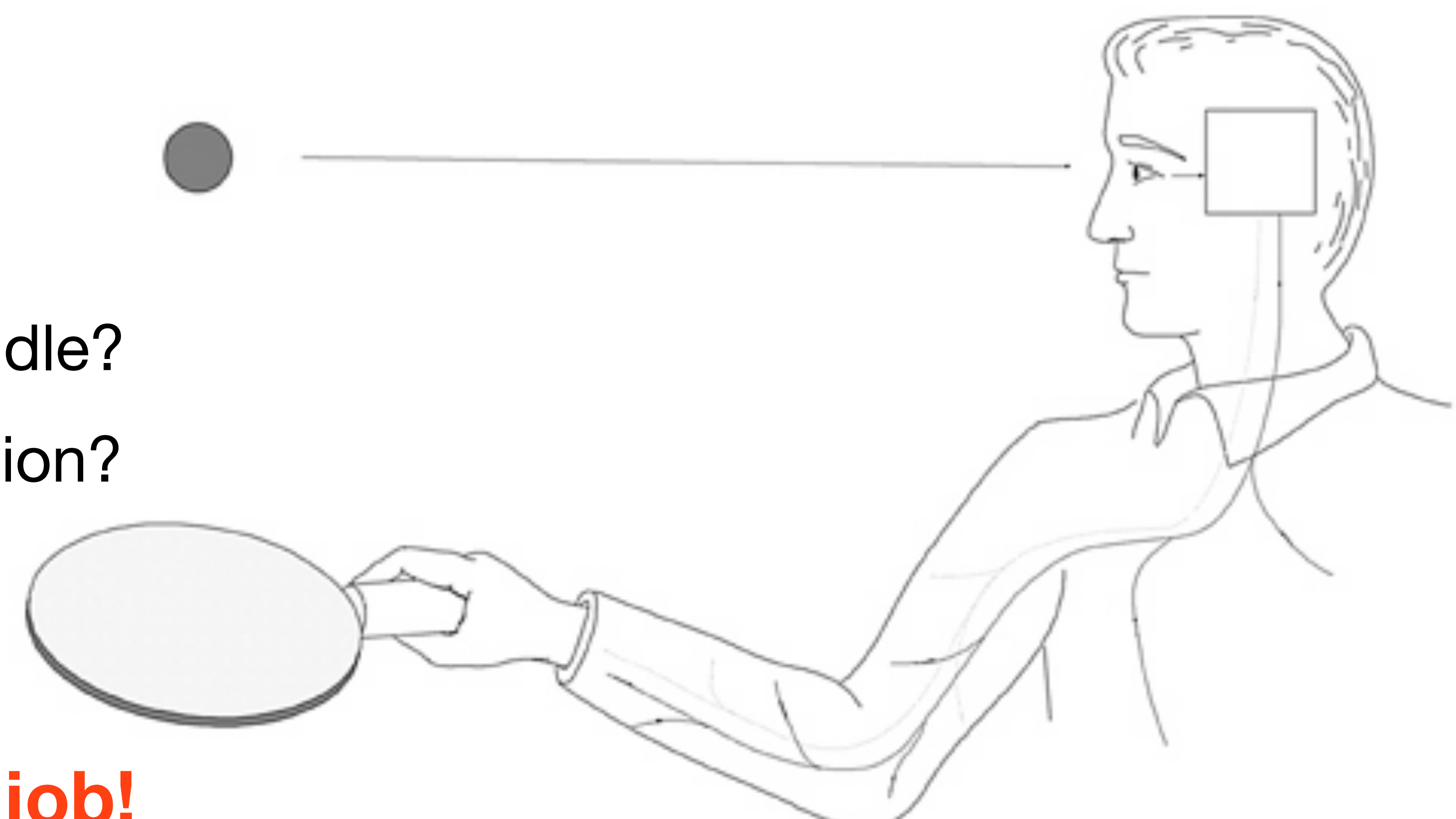
Task knowledge or success/failure?

- For some algorithms rewards in {1,0} are perfect ...
- Real problems often require *reward shaping*...

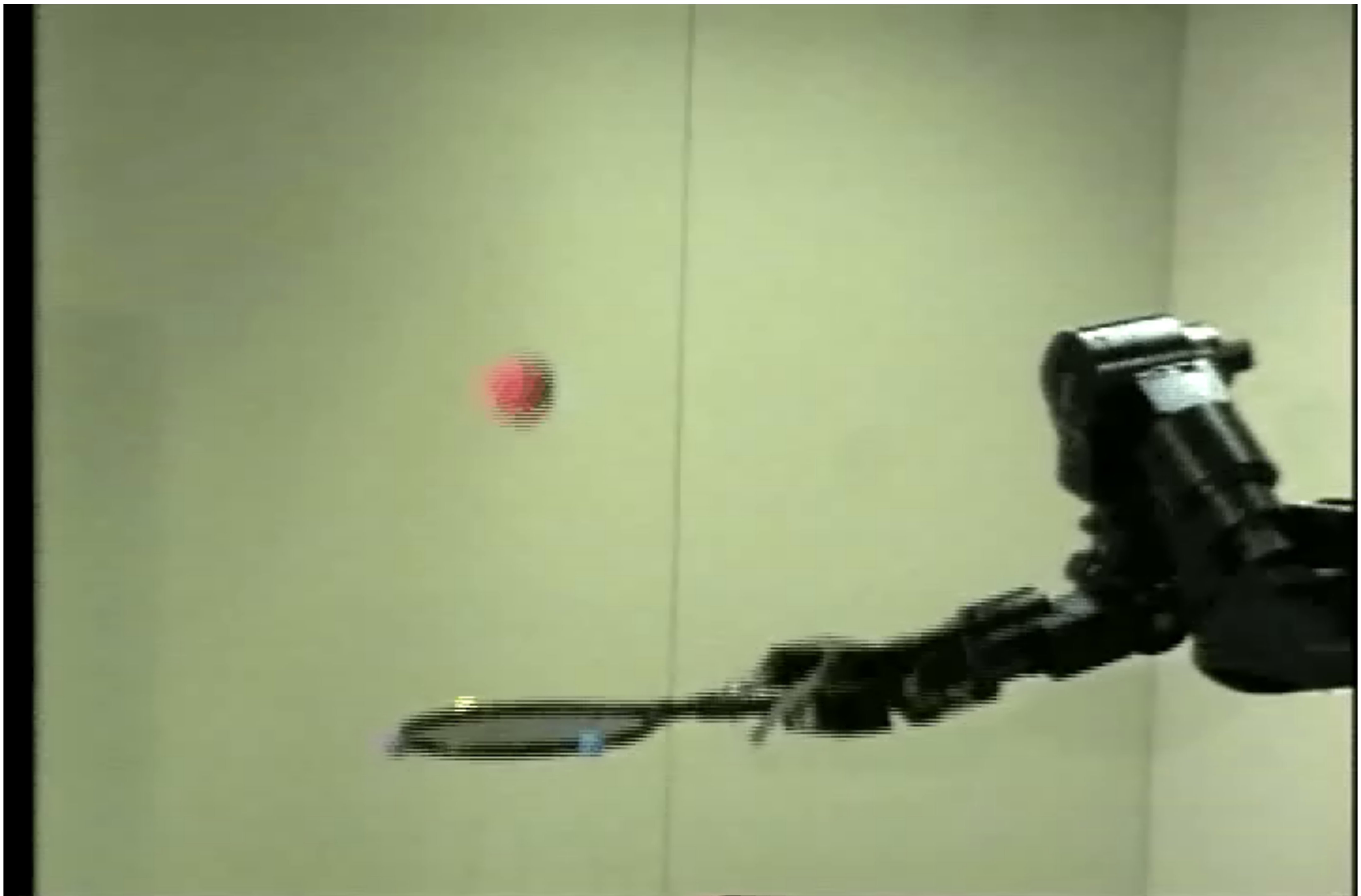
What's a good reward for our problem?

- Height of the ball?
- Distance between ball and the paddle?
- Ball needs to move in a certain region?
- All of the above?
- Additional punishments?

⇨ **All of these together do the job!**



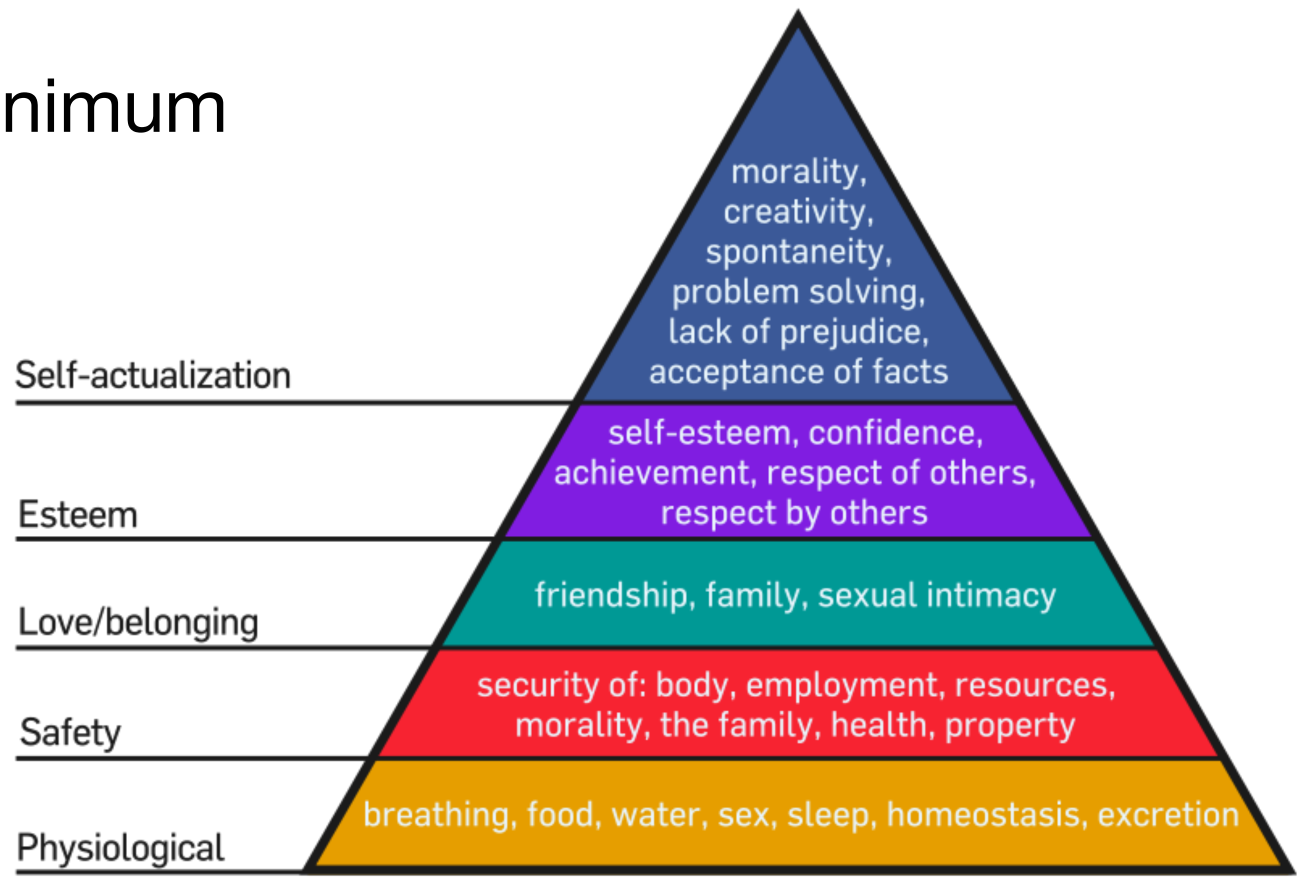
Example: Real world application...



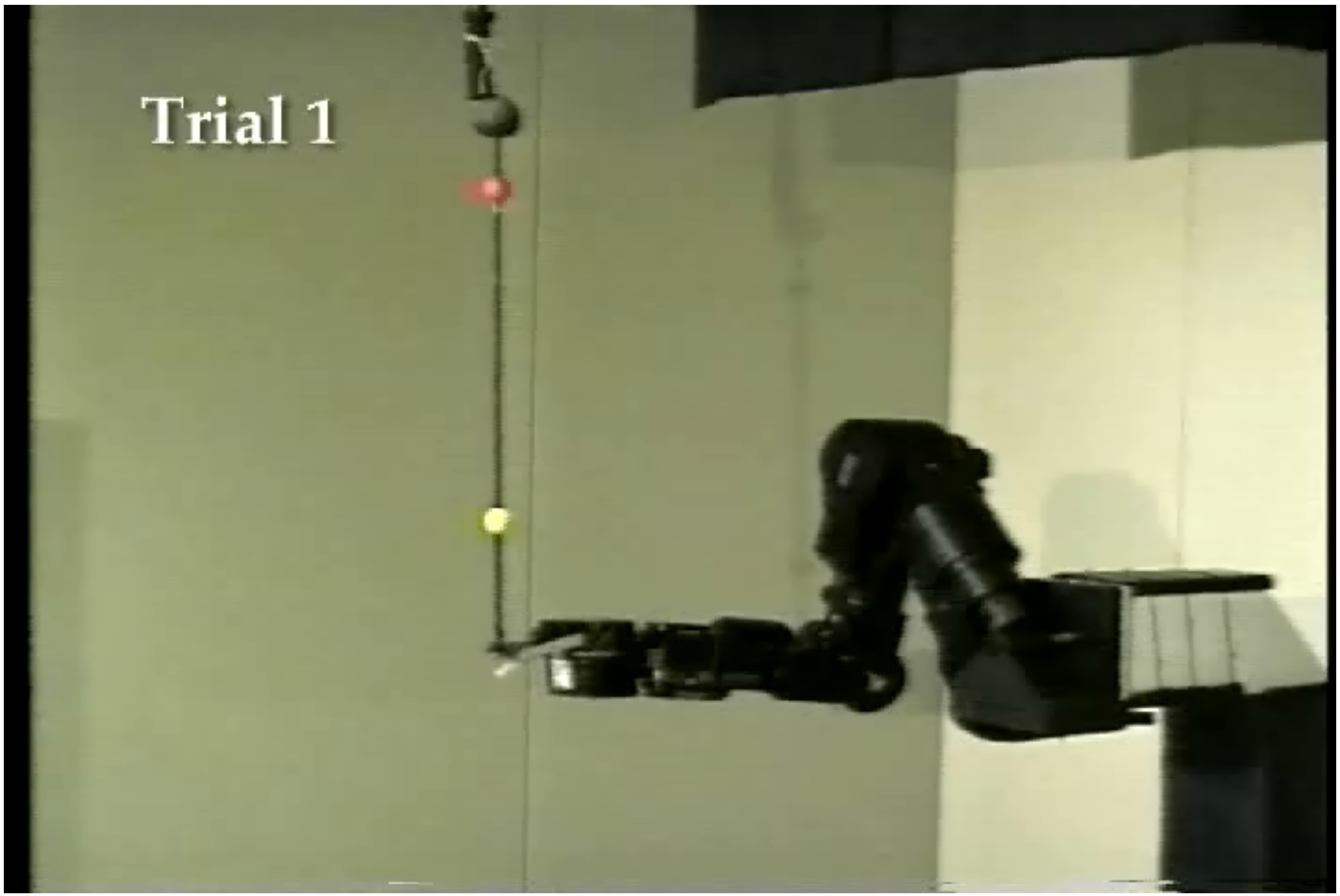
Human Motor Cost Functions?

Does this relate to Human Learning?

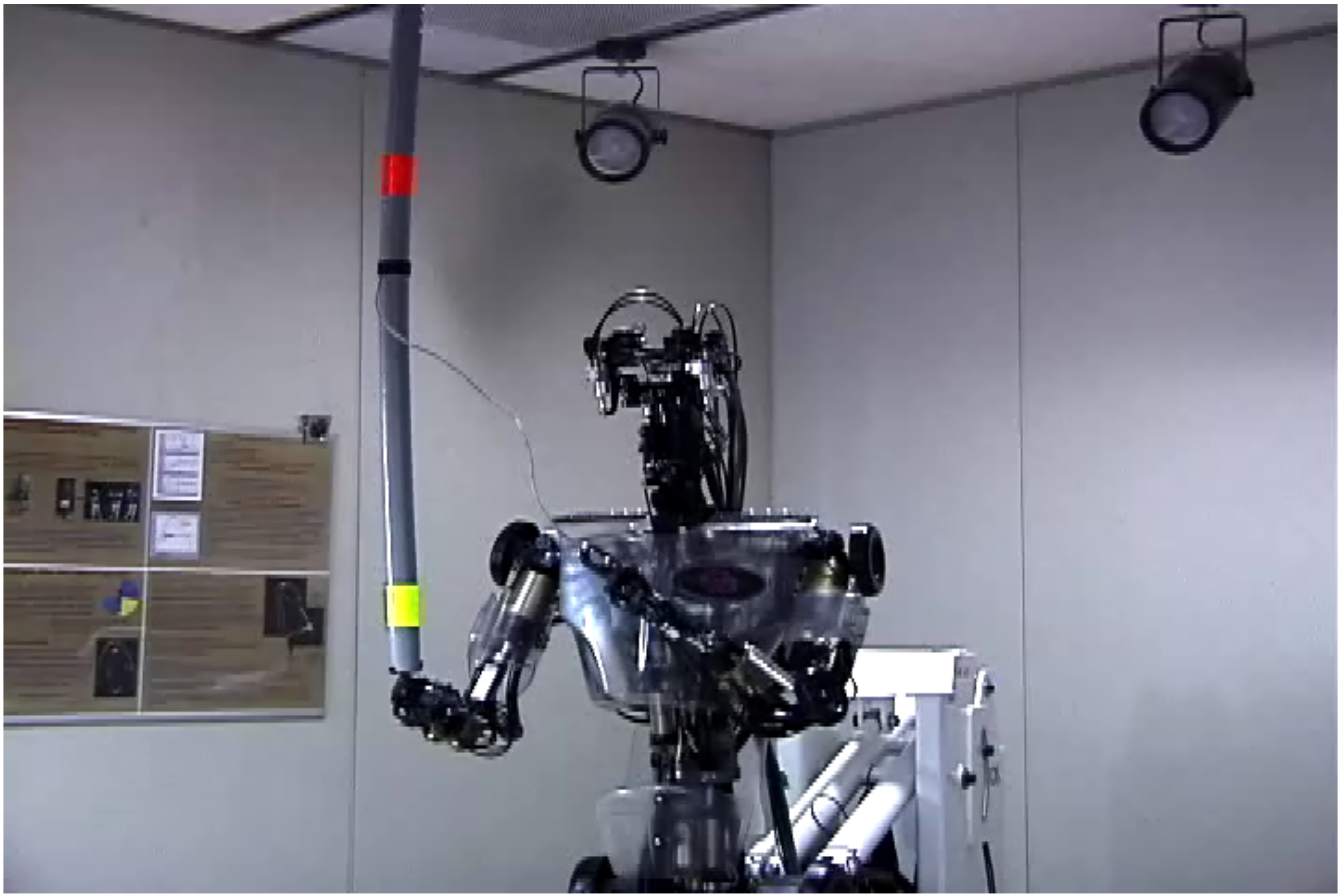
- ***Maybe!*** Many models from cognitive science are cost function based...
- *Reaching movements* can be explained by
  - Minimum jerk, Minimum torque change, Minimum end-point variance
- *Locomotion* can be explained by minimum metabolic energy consumption.
- Maslow's *Hierarchy of rewards* psychology ...



Model Learning with subsequent Policy Optimization



Model Learning with subsequent Policy Optimization



Challenges in model-based policy learning

This mainly works for balancing tasks where we live in a restricted state space

For more complex problems, using learned models becomes really hard

- ➡The model is likely to be inaccurate
- ➡Inaccuracies could be exploited by optimizer such that the policy on the real system performs bad
- ➡If we fully exploit an inaccurate model, we might jump into an area of the state space that we have not seen before
  - ➡inherently unstable

2 recent approaches…

1. PILCO (probabilistic inference for learning control)

Learn GP forward models

Use uncertainty of the GP-model for the long-term reward prediction

Policy Optimization with analytic gradient of expected reward

M. Deisenroth et. al.: “PILCO: A Model-Based and Data-Efficient Approach to Policy Search”, ICML 2011

Policy Optimization with PILCO

2 state of the art approaches…

2. Policy Search guided by trajectory optimization

Learn time dependent linear forward models

Trajectory Optimization: LQR like algorithm, additional constraint that new trajectory should stay close to the data → increase stability

Use optimized trajectories to learn a generalizing neural network policy

S. Levine et. al.: "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics.", NIPS 2014

Guided Policy Search

Learning Neural Network

Policies with Guided Policy Search

under Unknown Dynamics

Guided Policy Search

Conclusions

- You have solved an **(stochastic) optimal control** problem today!

- Only two cased are solvable: **linear & discrete**!

- The **optimal policy** for a **LQG system is a time-varying linear feedback** controller

- Linearizations can be problematic ⇨ lead to **oscillations** (but can be made more stable)

- Works well if the system is **not too non-linear and model can be learned accurately!**

- We will continue with **Value Function** and **Policy Search Methods.**