Model Learning

Gerhard Neumann
Jan Peters

Purpose of this Lecture

- Show different applications of supervised learning in robot learning.
- We can observe a lot of information, and model learning directly allows us to make use of it...
- Learning models *can* be easier than physical modeling as well as of learning control policies.
- **Model-based learning:** Using learned models to obtain a new policy is typically very data efficient!

Outline of the Lecture

1. An Example

2. Types of Models and Learning Architectures

3. Case Study A: *Inverse Dynamics & Forward Kinematics*

4. Case Study B: *Model Learning for Operational Space Control*

5. Case Study C: *Model Learning for Controller Falsification*

6. Final Remarks

Example: Mars Rover

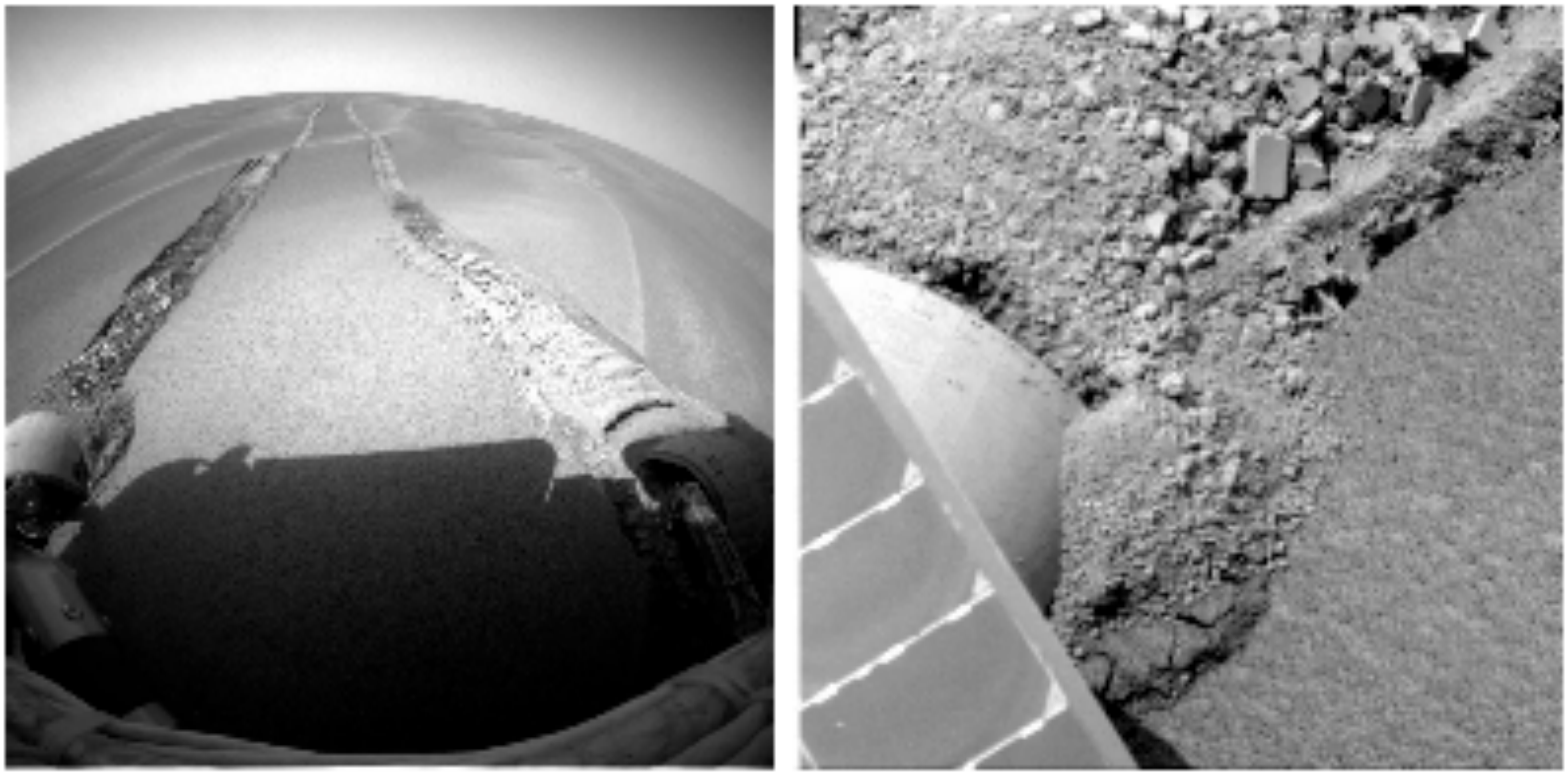
- Teleoperated System 1.5 AU (1 AU = 8min) away.
- Most intelligence was still on earth.
- Key problems:

  i) getting stuck,

  ii) coping with delays

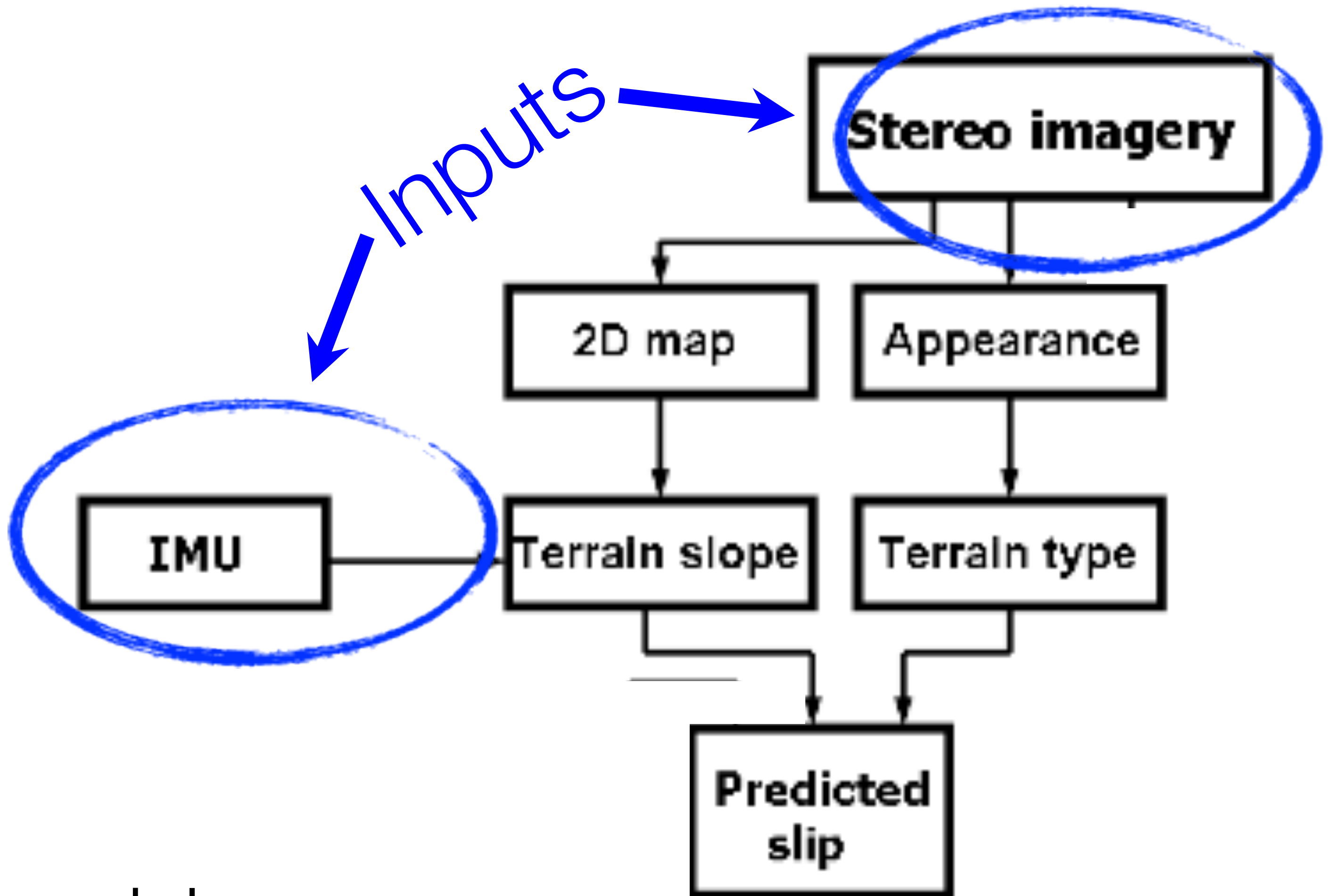
Hence, we need good models...

Learning to Predict Slip



The Mars Exploration Rover Opportunity trapped in the Purgatory dune on sol 447. A similar slip condition can lead to mission failure.

A Model Learning Architecture



Underlying model:

Geometry

$$p(S|A,G) = \sum_T p(T|A,G)p(S|T,A,G)$$

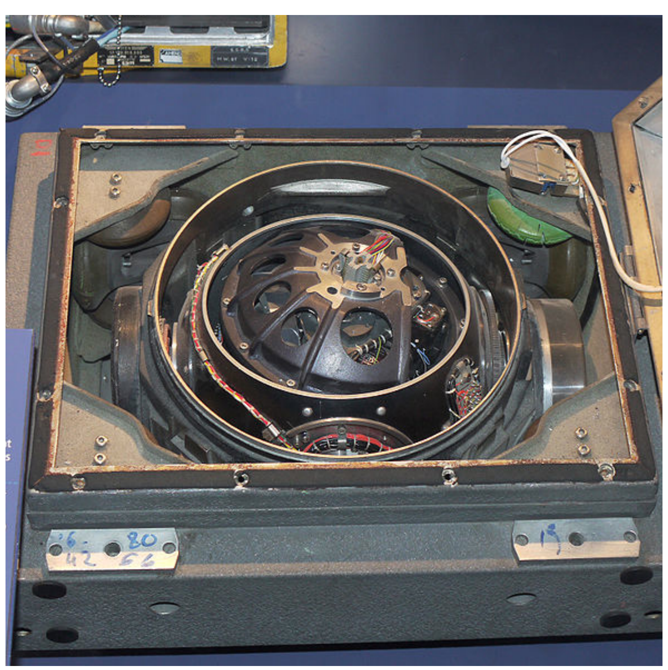
Slip Terrain Appearance

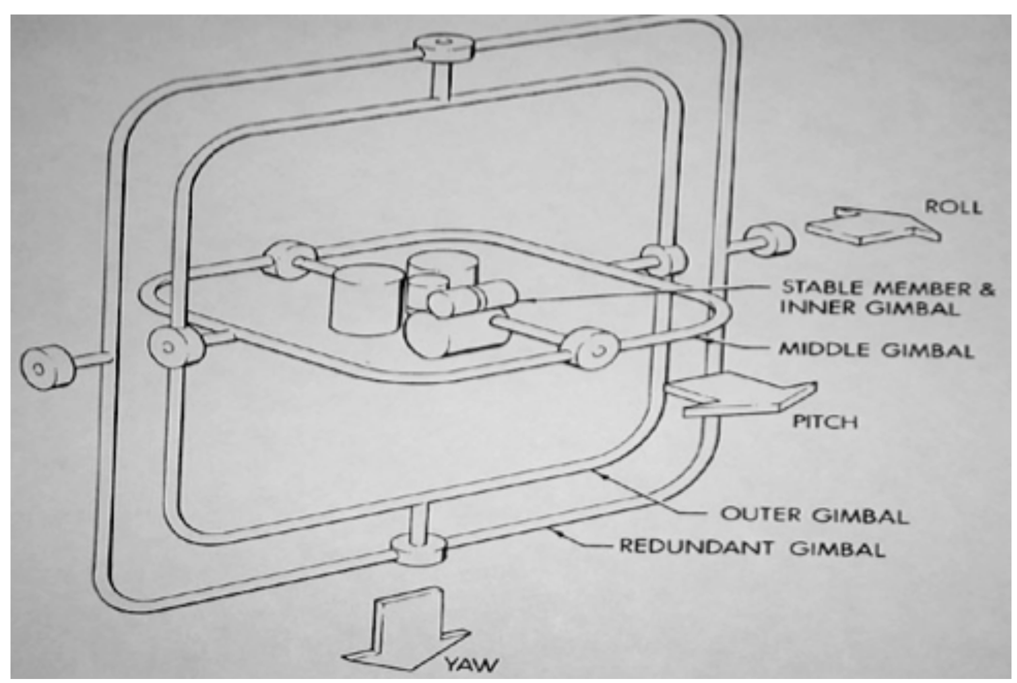
A. Angelova, L. Matthies, D. Helmick, P. Perona, Slip Prediction Using Visual Information, Robotics: Science and Systems (RSS), 2006

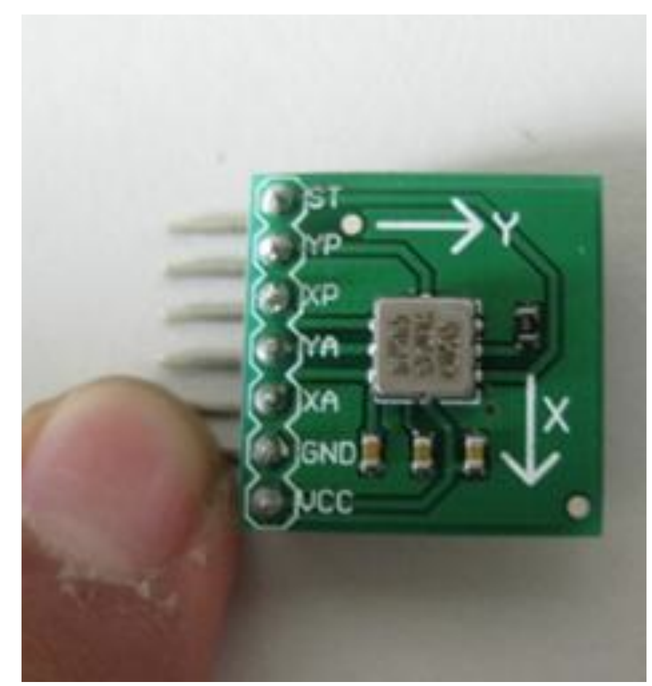
Inputs

Images

Orientation



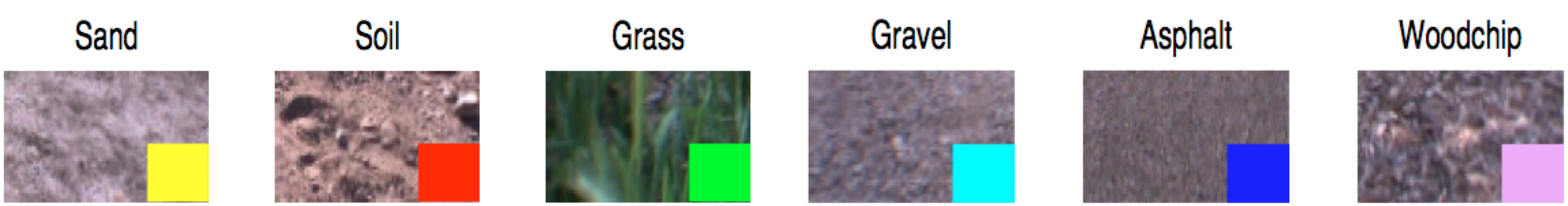




Features

Terrain Slope

Terrain Type



Legend

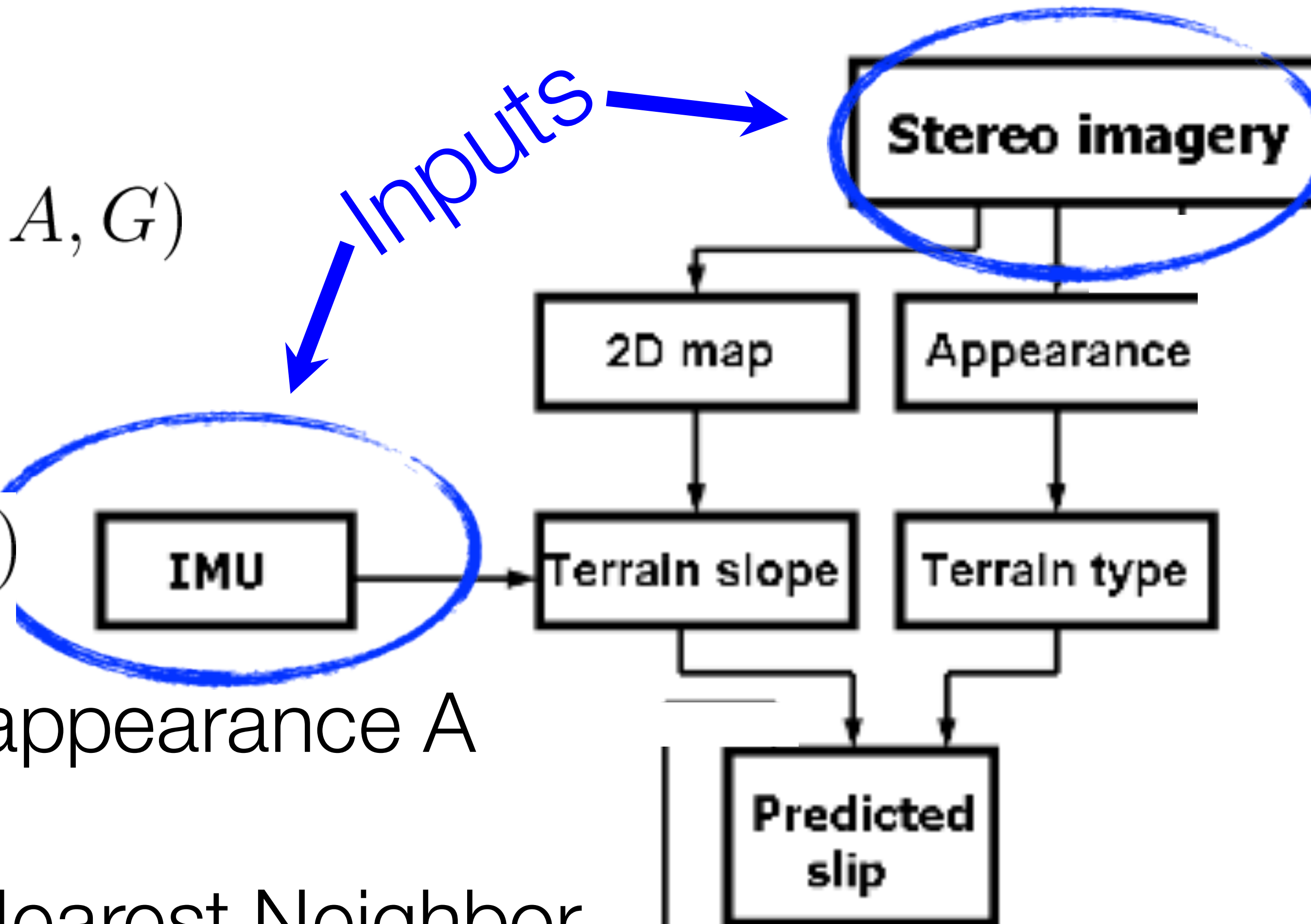
A Model Learning Architecture

Underlying model:

$$p(S|A,G) = \sum_T p(T|A,G)p(S|T,A,G)$$

Simplification:

$$p(S|A,G) = \sum_T p(T|A)p(S|T,G)$$



- $p(T|A)$ terrain prediction from appearance A

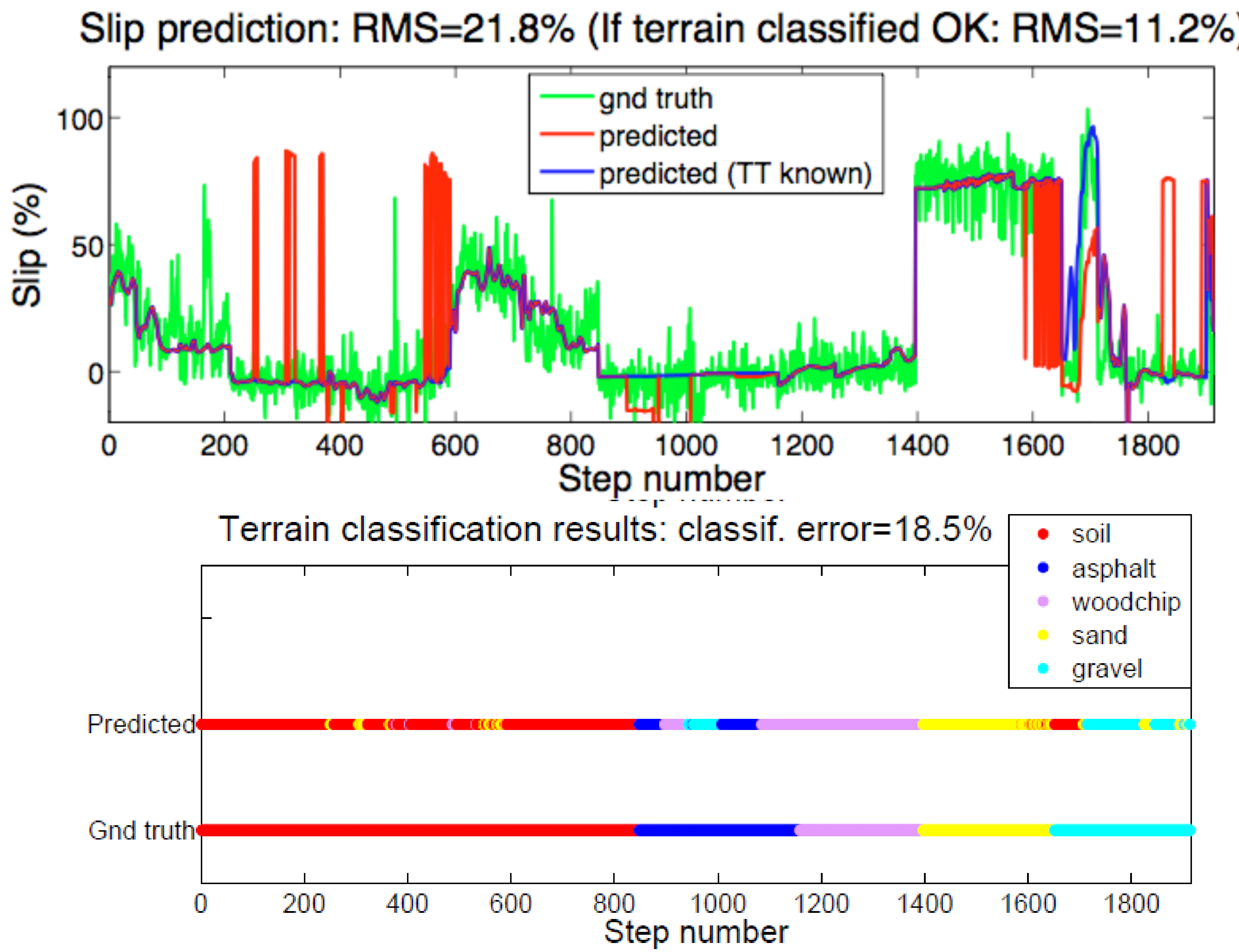
  Classification: Clustering + Nearest Neighbor

- $p(S|T,G)$ slip prediction from slopes G for each terrain

  Regression: 2 slopes -> slip, locally weighted regression

A. Angelova, L. Matthies, D. Helmick, P. Perona, [Slip Prediction Using Visual Information](), Robotics: Science and Systems (RSS), 2006

Outputs



If terrain type is known, prediction is almost spot on!

Outline of the Lecture

1. An Example
2. Types of Models and Learning Architectures
3. Case Study A: *Inverse Dynamics & Forward Kinematics*
4. Case Study B: *Model Learning for Operational Space Control*
5. Case Study C: *Model Learning for Controller Falsification*
6. Final Remarks

Types of Models

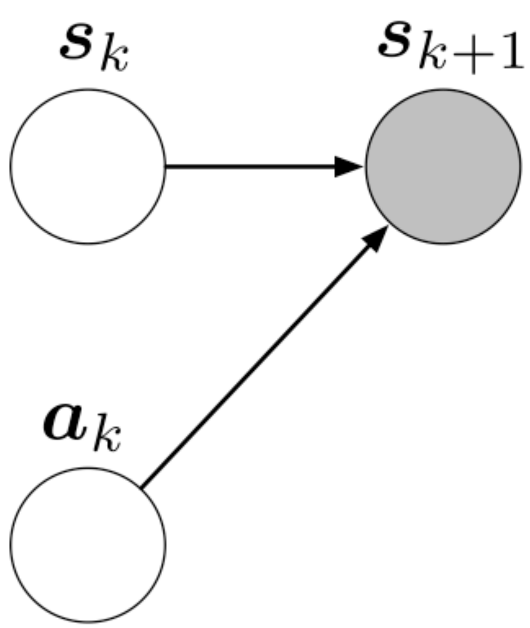
Assume our system has the functional form

- Discrete time: $$\boldsymbol{s}_{k+1} = f_D(\boldsymbol{s}_k, \boldsymbol{u}_k) + \boldsymbol{\epsilon}$$
- Continuous time: $$\dot{\boldsymbol{s}}_k = f_C(\boldsymbol{s}_k, \boldsymbol{u}_k) + \boldsymbol{\epsilon}$$
- Discrete time often easier to use $\Rightarrow$ no integration needed

Four types of models become useful:

- Forward Models: Predict the future state.
- Inverse Models: Predict the action needed to reach a state.
- Mixed Models: Predict required task elements with a forward model and use an inverse model for control.
- Multi-Step Models: Predict far in the future what will happen...

Forward Models



- **Predict next state:** $\boldsymbol{s}_{k+1} = f_D(\boldsymbol{s}_k, \boldsymbol{u}_k) + \boldsymbol{\epsilon}$

- **Dataset:**

$$\boldsymbol{X} = \{\boldsymbol{s}_k, \boldsymbol{u}_k\}_{k=1\ldots N}$$

$$\boldsymbol{Y} = \{\boldsymbol{s}_{k+1}\}_{k=1\ldots N}$$

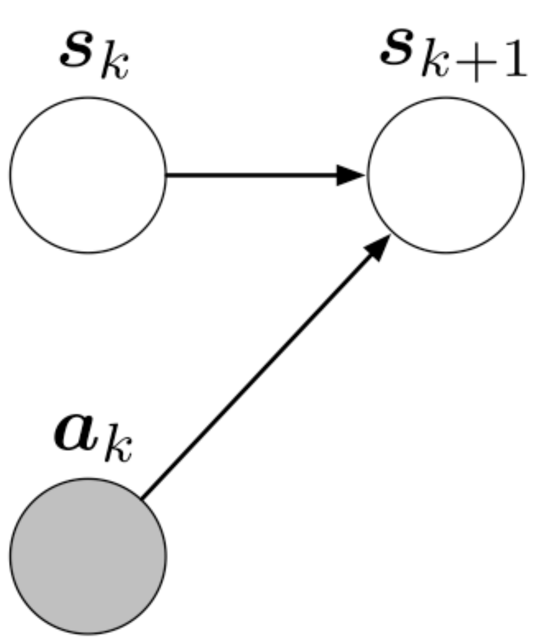
- Can be used for direct action generation:

$$\pi(\boldsymbol{s}_t) = \mathrm{argmin}_{\boldsymbol{a}} ||f(\boldsymbol{s}_t, \boldsymbol{a}) - \boldsymbol{s}_{t+1}^{\mathrm{des}}||$$

- Forward model is a simulator! $\Rightarrow$ can be used for long-term prediction!

Note: typically: $\boldsymbol{s} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}$

Inverse Models



- Predict the **action needed to reach a desired** state or any other desired outcome:

$$\boldsymbol{u} = \pi(\boldsymbol{s}_t) = f(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}^{\text{des}})$$

- **Dataset:**

$$\boldsymbol{X} = \{\boldsymbol{s}_k, \boldsymbol{s}_{k+1}\}_{k=1\dots N}$$
$$\boldsymbol{Y} = \{\boldsymbol{u}_k\}_{k=1\dots N}$$

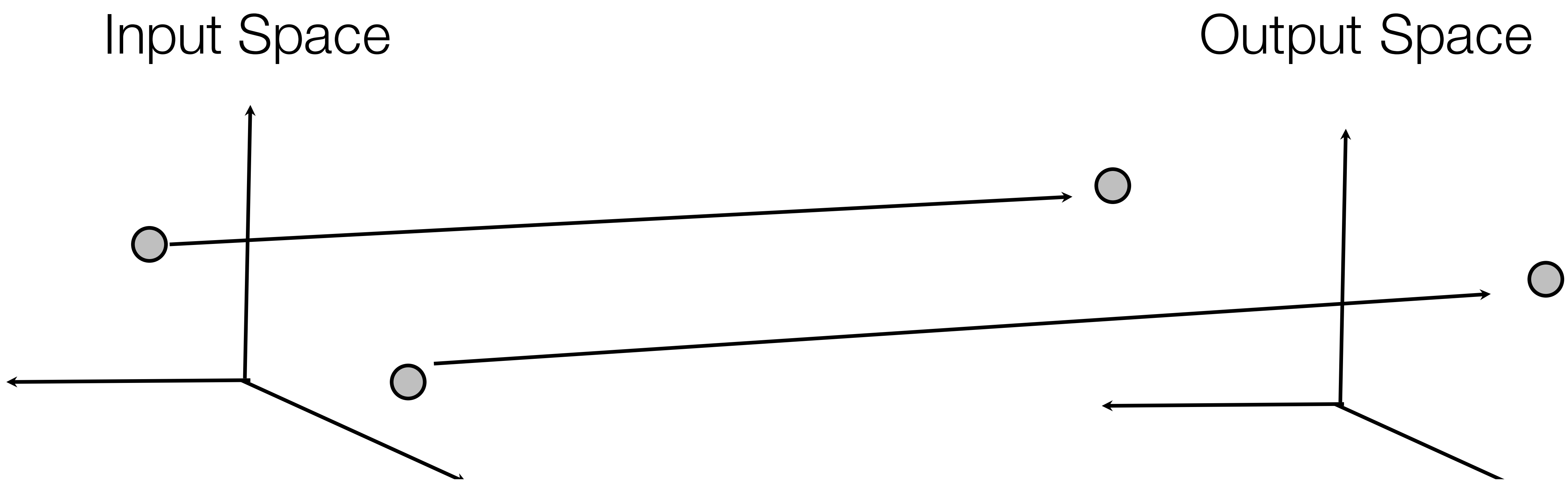
- Can be used directly in control, e.g**., inverse dynamics control:**

$$\boldsymbol{u} = f(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \ddot{\boldsymbol{q}}_t^{\text{des}})$$
$$\ddot{\boldsymbol{q}}_t^{\text{des}} = \boldsymbol{K}_P(\boldsymbol{q}_t^{\text{des}} - \boldsymbol{q}_t) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_t^{\text{des}} - \dot{\boldsymbol{q}}_t)$$

Next desired state is represented by the **desired acceleration** $\ddot{\boldsymbol{q}}_t^{\text{des}}$

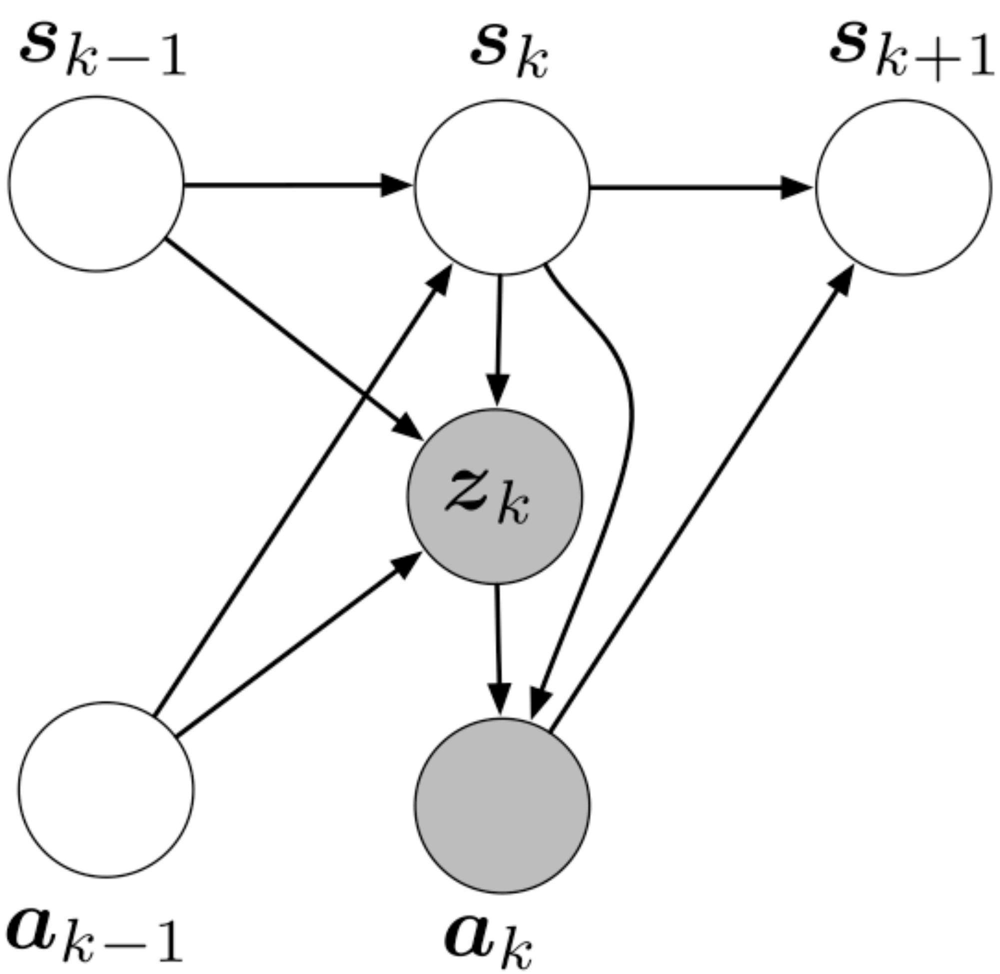
Inverse Model Learning ...

As long as our system is an **invertible function**, inverse model learning will be useful!



...but is that is not true for many problems!
Why? Redundancy!!

Mixed Model

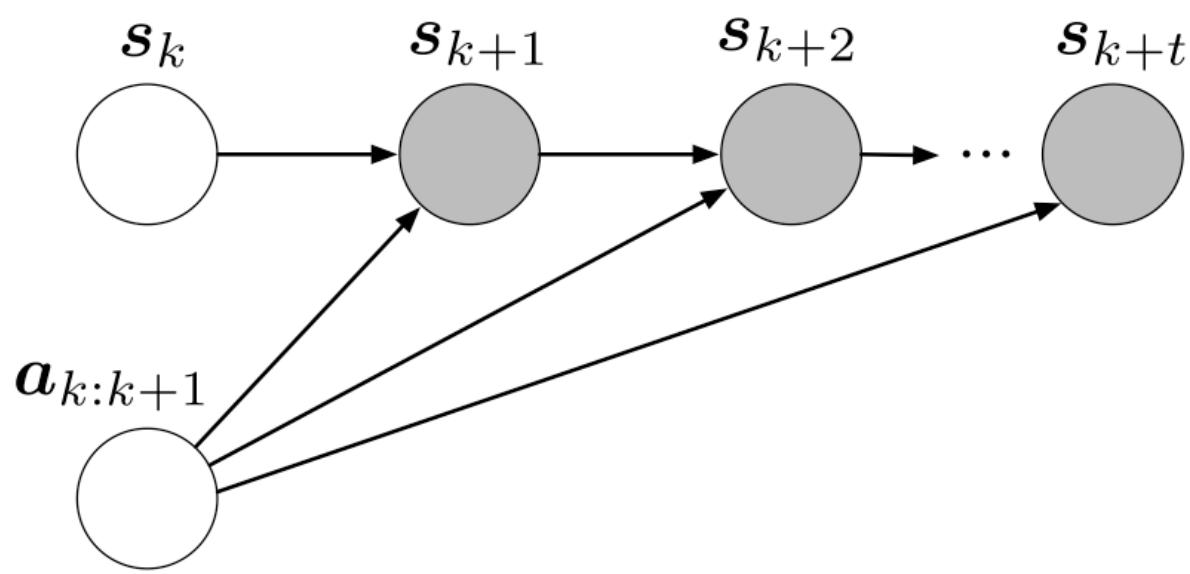


- Assume that we can use our *forward model* to predict quantity *z*.
- Based on *z*, our model can determine the action *a* with an inverse model.
- Examples are:

  i) Systems with Hysteresis

  ii) Inverse Kinematics

Multi-Step Prediction Models

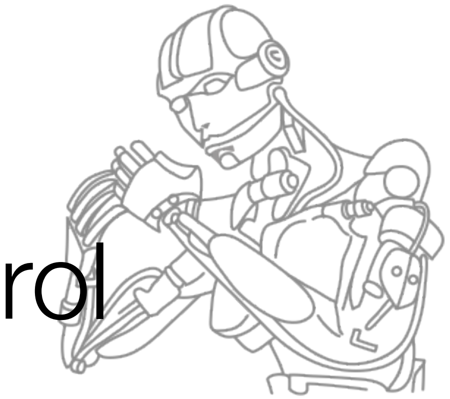


Example: Imagine you are controlling the Mars Rover. In that case, you need to predict the effect of your actions many states ahead such that you can cope with the delays in the system.

Multi-step prediction vs. **iterative one step prediction**?

- **Multi-step:** only for open loop control
- **Single step:** error accumulates!

Motivation for Model Learning in Robot Control



Why learn (Inverse) Kinematics Models?

- Kinematics can be measured nearly perfectly
- but Inverse Kinematics are expensive.

Why learn Dynamics Models:

- Dynamics parameters are terrible to estimate for interesting systems.
- Rigid Body Dynamics are inherently incomplete.

Example Problems in Robot Control

Forward Kinematics:

$$\mathbf{x} = f(\mathbf{q})$$

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$$

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})\ddot{\mathbf{q}}$$

Inverse Kinematics:

$$\mathbf{q} = f^{-1}(\mathbf{x})$$

$$\dot{\mathbf{q}} = \mathbf{J}^T(\mathbf{q})(\mathbf{J}(\mathbf{q})\mathbf{J}^T(\mathbf{q}))^{-1}\dot{\mathbf{x}} = \mathbf{J}^+\dot{\mathbf{x}}$$

Which one is not a regression model?

Example Problems in Robot Control

Forward Dynamics:

Continuous Time: $$\ddot{\boldsymbol{q}} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$$

Discrete Time: $$[\boldsymbol{q}_{t+1}, \dot{\boldsymbol{q}}_{t+1}] = f(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t, \boldsymbol{u})$$

Discrete time vs. continuous time forward models

+ Easier to learn, less noisy data
+ Model learns non-linear effects due to integration
- only works for constant control action and fixed time step

Example Problems in Robot Control

Inverse Dynamics:

$$\boldsymbol{u} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_{\text{ref}})$$

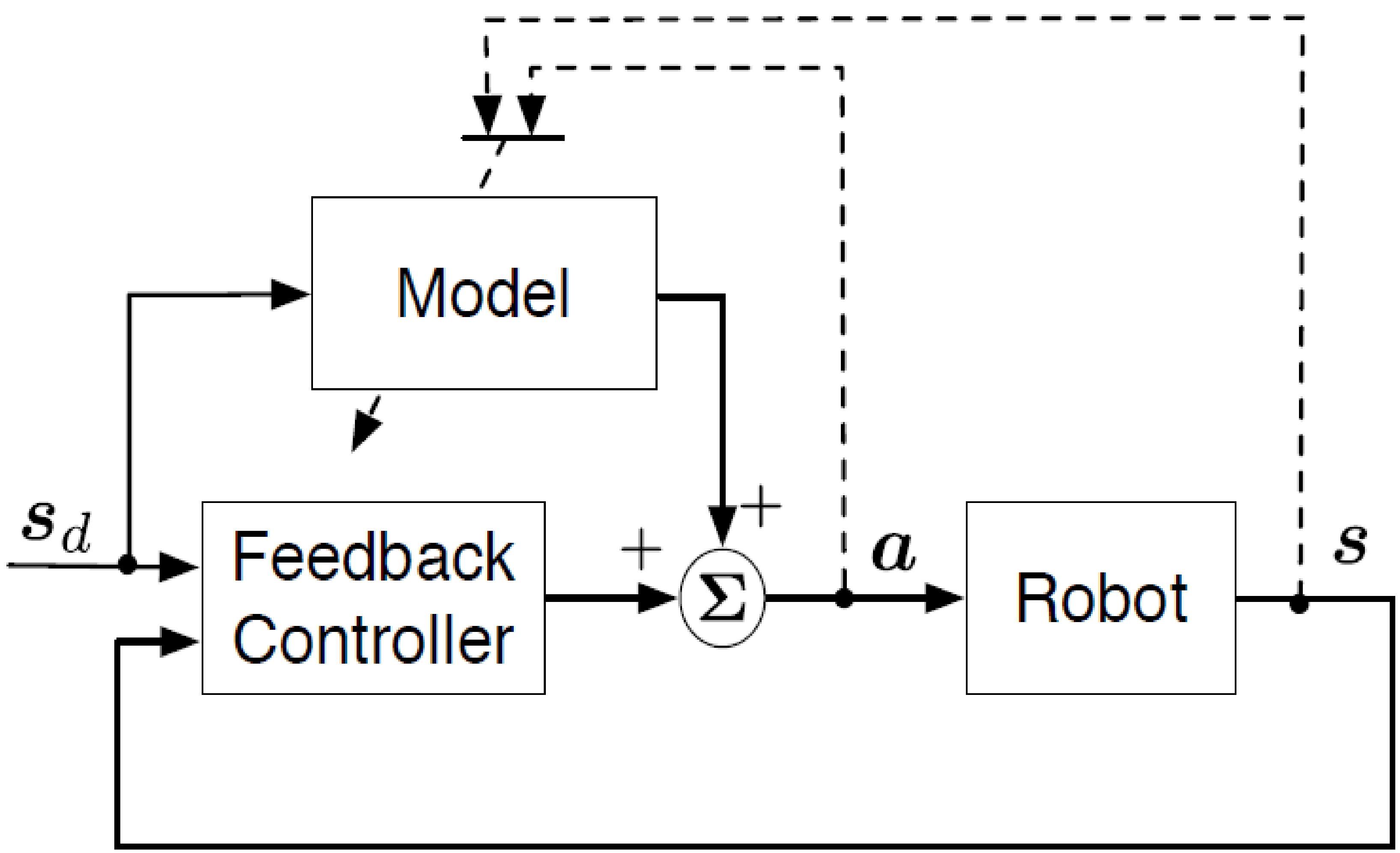
Operational/Task Space Control:

$$\boldsymbol{u} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_{\text{ref}})$$

Which one is not a regression model?

Model Learning Architectures

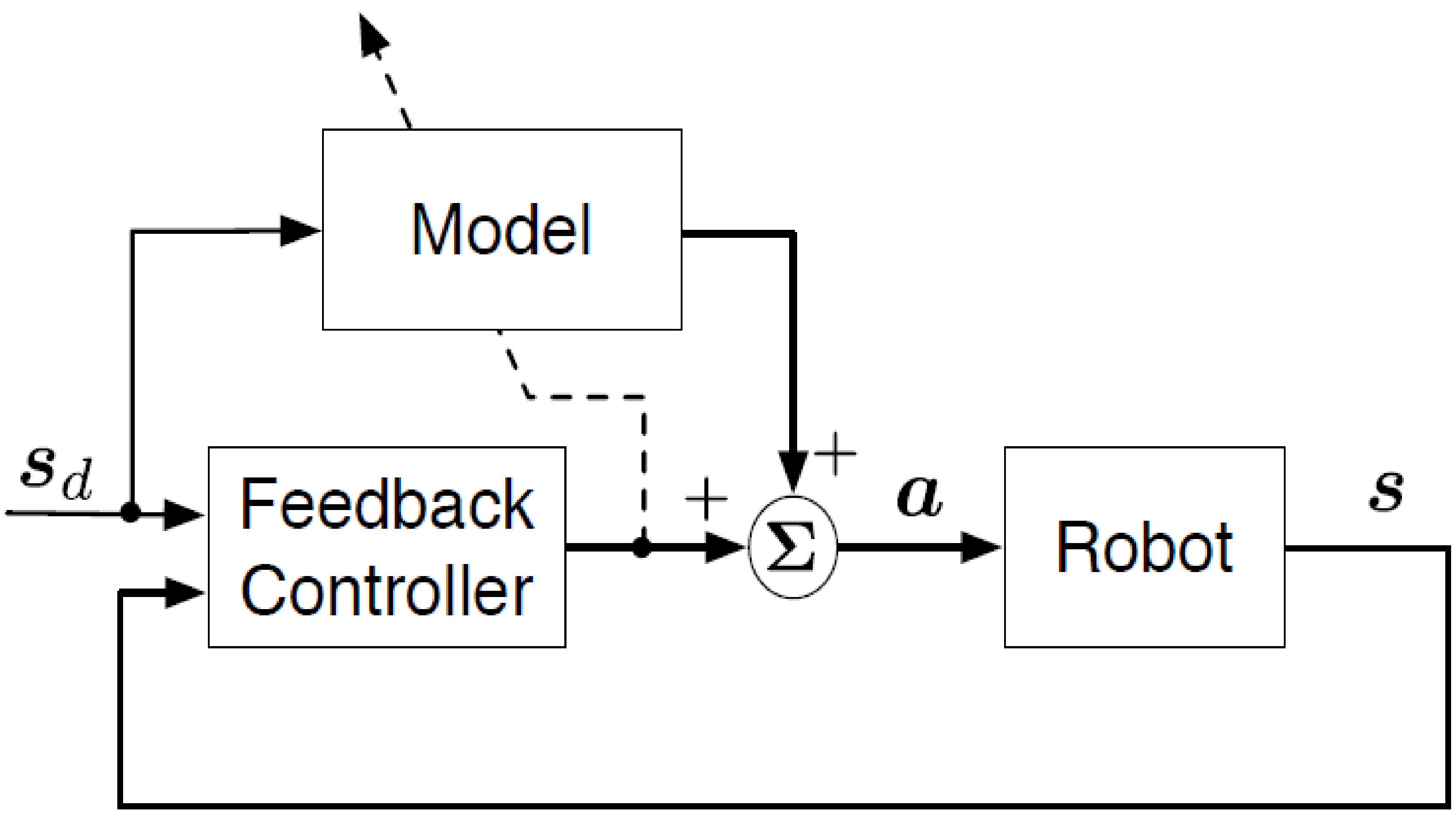
Direct Modeling



- Learning is directly formulated as regression problem
- Works for well defined input-output relationship

Model Learning Architectures

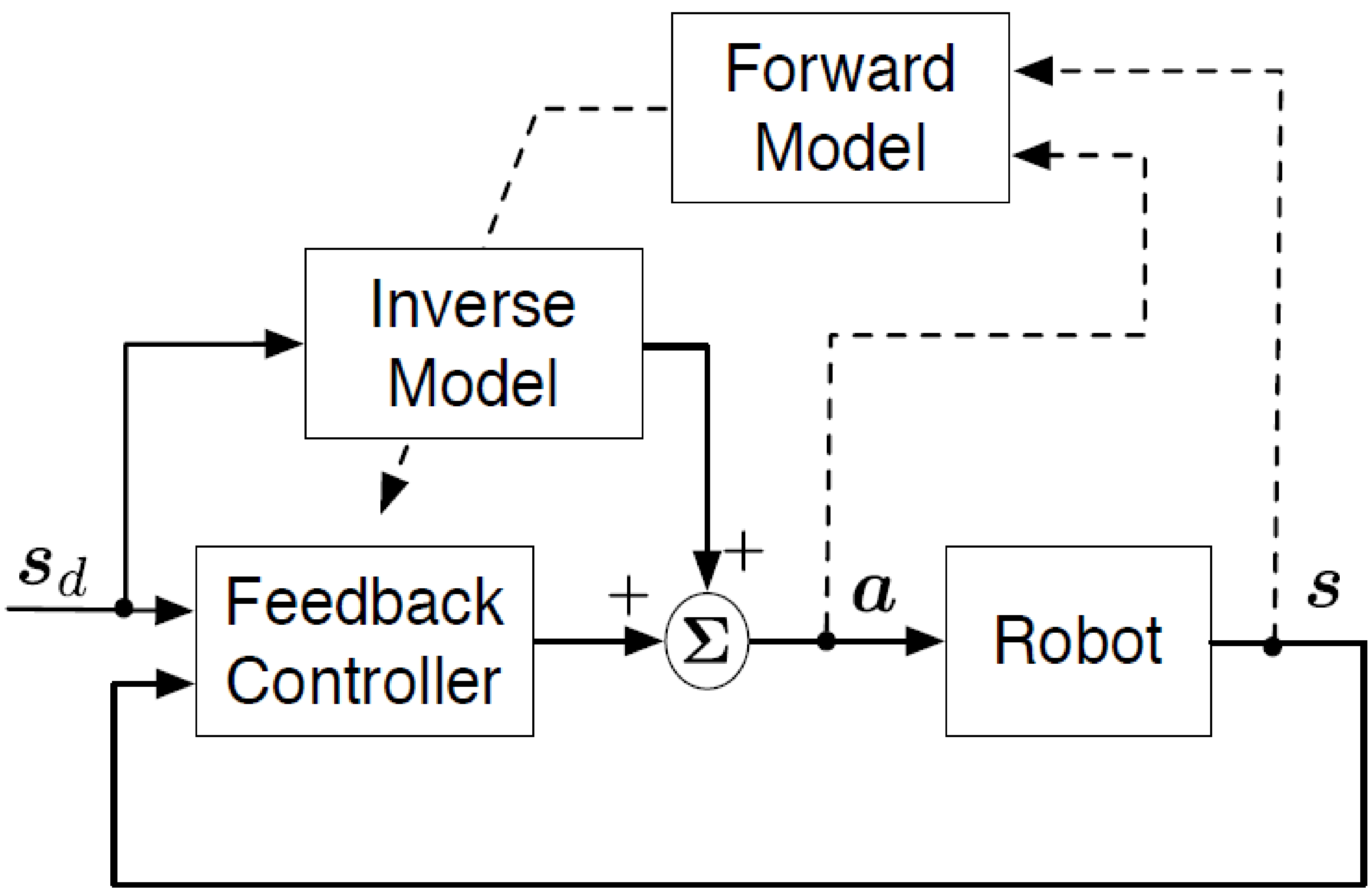
Indirect Modeling



- Works also for ill-defined problems (e.g., differential inverse kinematics)
- Learning is modulated by a the feedback error
- Goal oriented, learns for a specific task s_d

Model Learning Architectures

Distal Teacher Learning



- Designed for ill-defined problem of learning inverse models
- Learn unique forward and and inverse models
- Forward-model guides learning of the inverse model

Challenges in Model Learning

- High-dimensionality
- Smoothness
- Discontinuities (E.g., stiction, contacts)
- Noise/Outliers
- Missing Data
- Too large or too small datasets
- Online updates
- Incorporation of prior knowledge
- Robustness and Safety

Outline of the Lecture

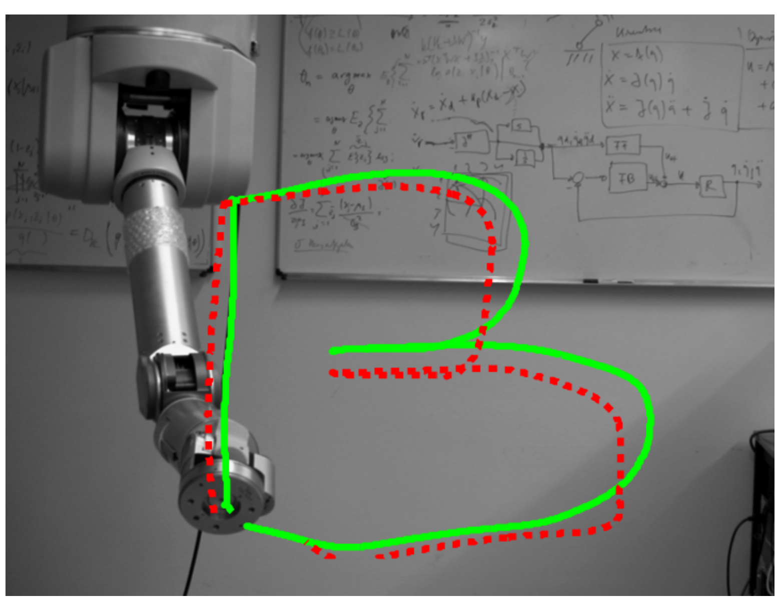
1. An Example
2. Types of Models and Learning Architectures
3. Case Study A: *Inverse Dynamics & Forward Kinematics*
4. Case Study B: *Model Learning for Operational Space Control*
5. Case Study C: *Model Learning for Controller Falsification*
6. Final Remarks

Learning to Control with Models

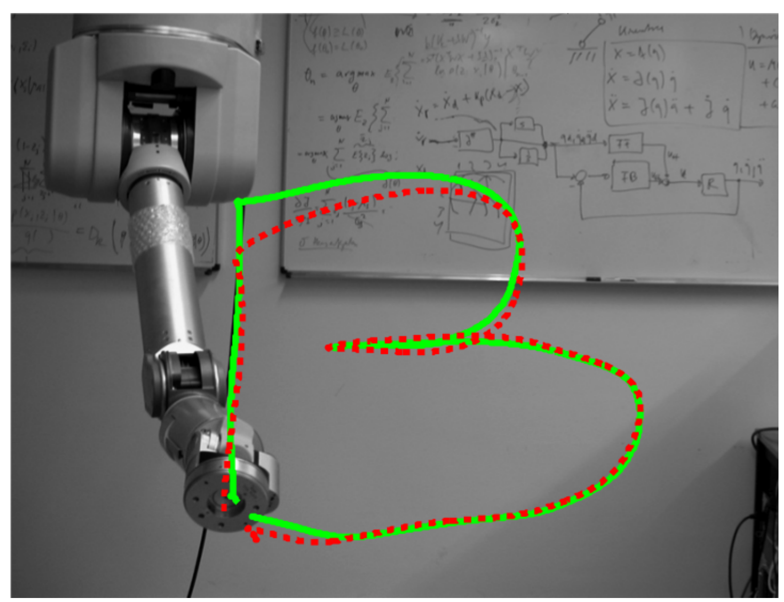
Compliant, low-gain control of fast & accurate movements requires precise models.

- A changing world requires adaption to altered dynamics.
- Control both directly in joint (here) and task space (next)

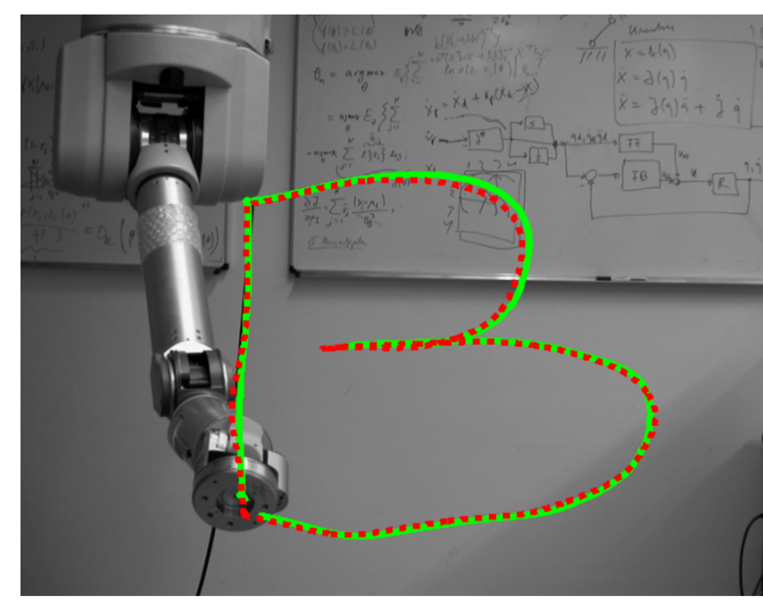
Analytical Rigid-Body Model with CAD data



Offline Trained



Online Trained



Nguyen-Tuong, Peters, IROS 2008 (Finalist for Best Paper Award)

Function Approximation Problem

Joint Accelerations, Velocities, Positions

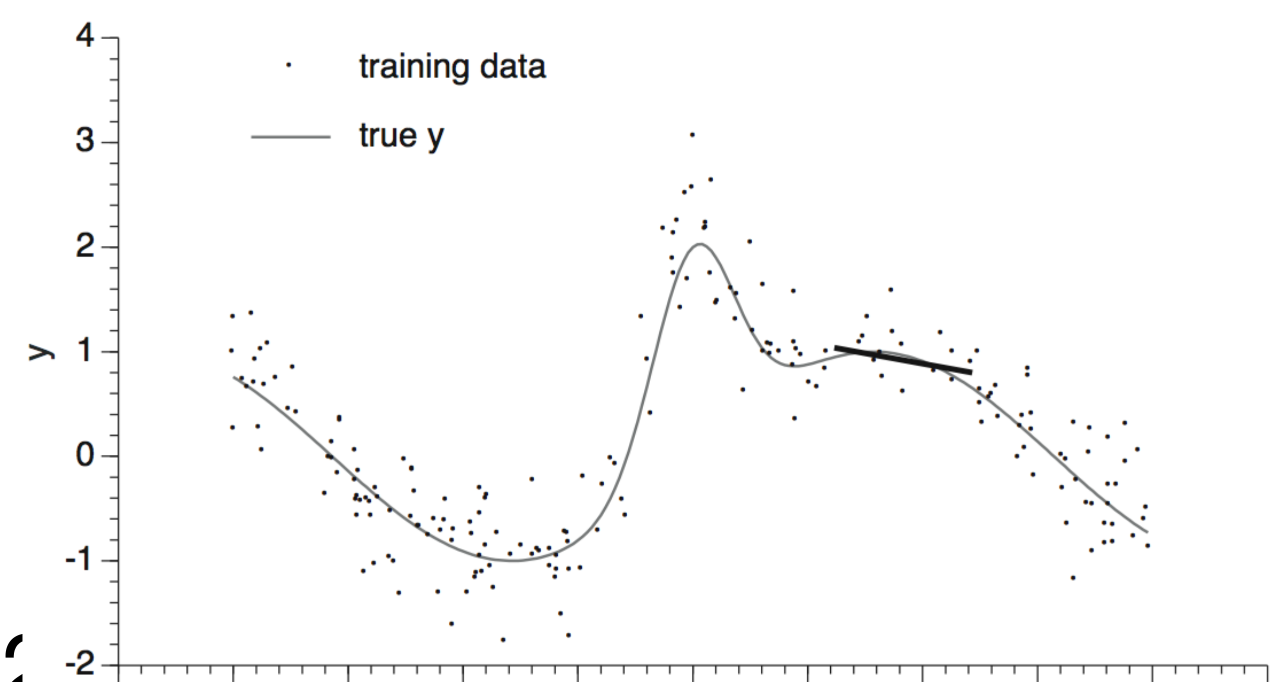
Torques

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

Mass Matrix

Gravity

Coriolis & Centripetal Forces



Inverse Dynamics is a giant *function approximation* problem

- **Robot arm**
  - 3 x 7 = 21 state dimensions,
  - 7 action dimensions
- **Humanoid**
  - 3 x 30 = 90 state dimensions
  - 30 action dimensions
- Learning in real-time!
  - **Online Adaptation** is needed for unexplored areas
- Unlimited continuous stream of data...

Function Approximation Problem

Joint Accelerations, Velocities, Positions

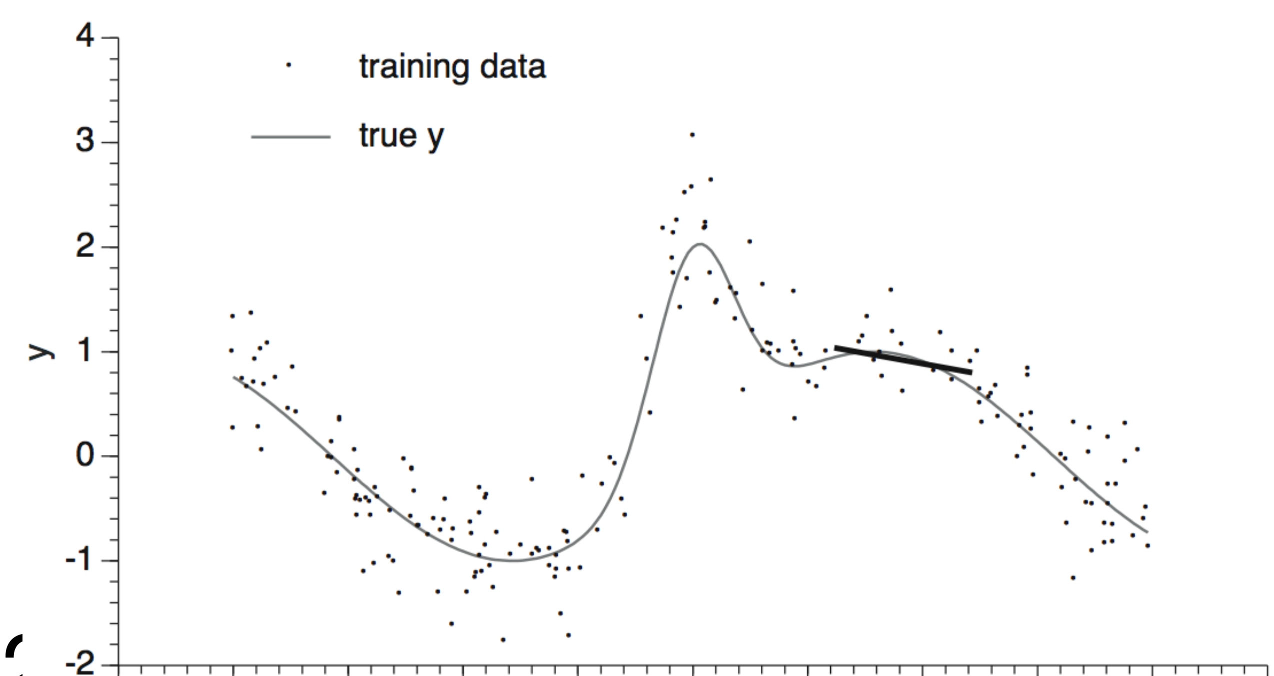
Torques

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

Mass Matrix

Gravity

Coriolis & Centripetal Forces



What methods can deal with this problem?

- **Neural networks?**
- **Kernel Regression? GPs?**
- **Computationally expensive:** only in offline settings

Local methods can perform online:

- **Locally Weighted PLS Regression (LWPR)** (Schaal, Atkeson & Vijayakumar, 2002)
- **Local Gaussian Processes (LGP)** (Nguyen-Tuong, Peters, 2008)

Local Gaussian Processes

Gaussian Processes are **typically slow:** $\mathcal{O}(N^3)$computing the inverse of kernel matrix

Use **Local GP Models**:

- Use centers $\boldsymbol{c}_k$ with activation function $w_k(\boldsymbol{x}) = \exp\left(-0.5 \sum_i \frac{(x_i - c_{ik})^2}{h_i}\right)$
- Whenever $w_k(\boldsymbol{x}) \leq w_{\text{thresh}}, \forall k$ create new center at location $\boldsymbol{x}$
- Output function: $\mu(\boldsymbol{x}) = \frac{\sum_k w_k(\boldsymbol{x}) \mu_k(\boldsymbol{x})}{\sum_k w_k(\boldsymbol{x})}$
- Add data only to nearest center

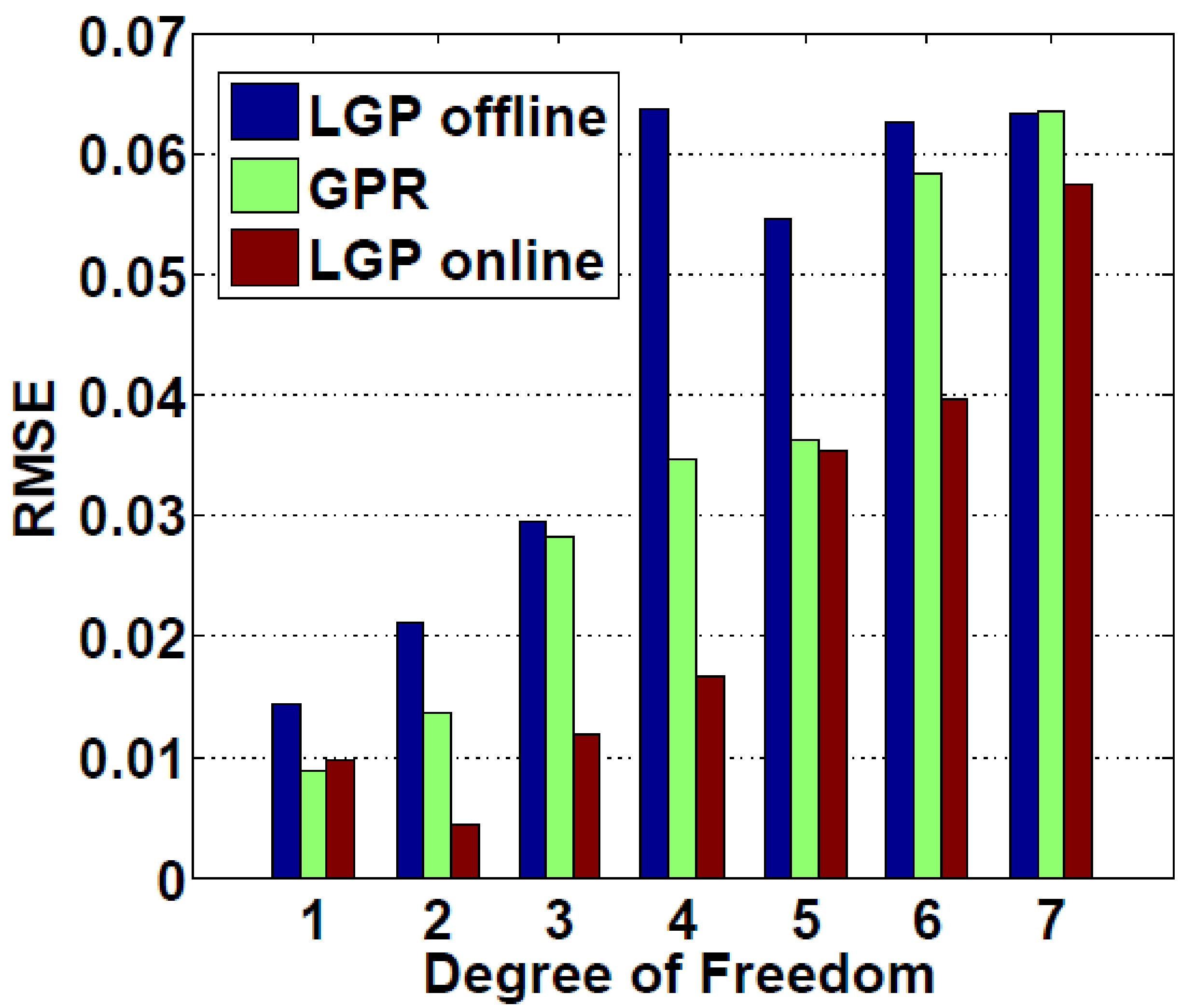
Local Gaussian Processes

Computational Complexity: $\mathcal{O}(L^2K)$

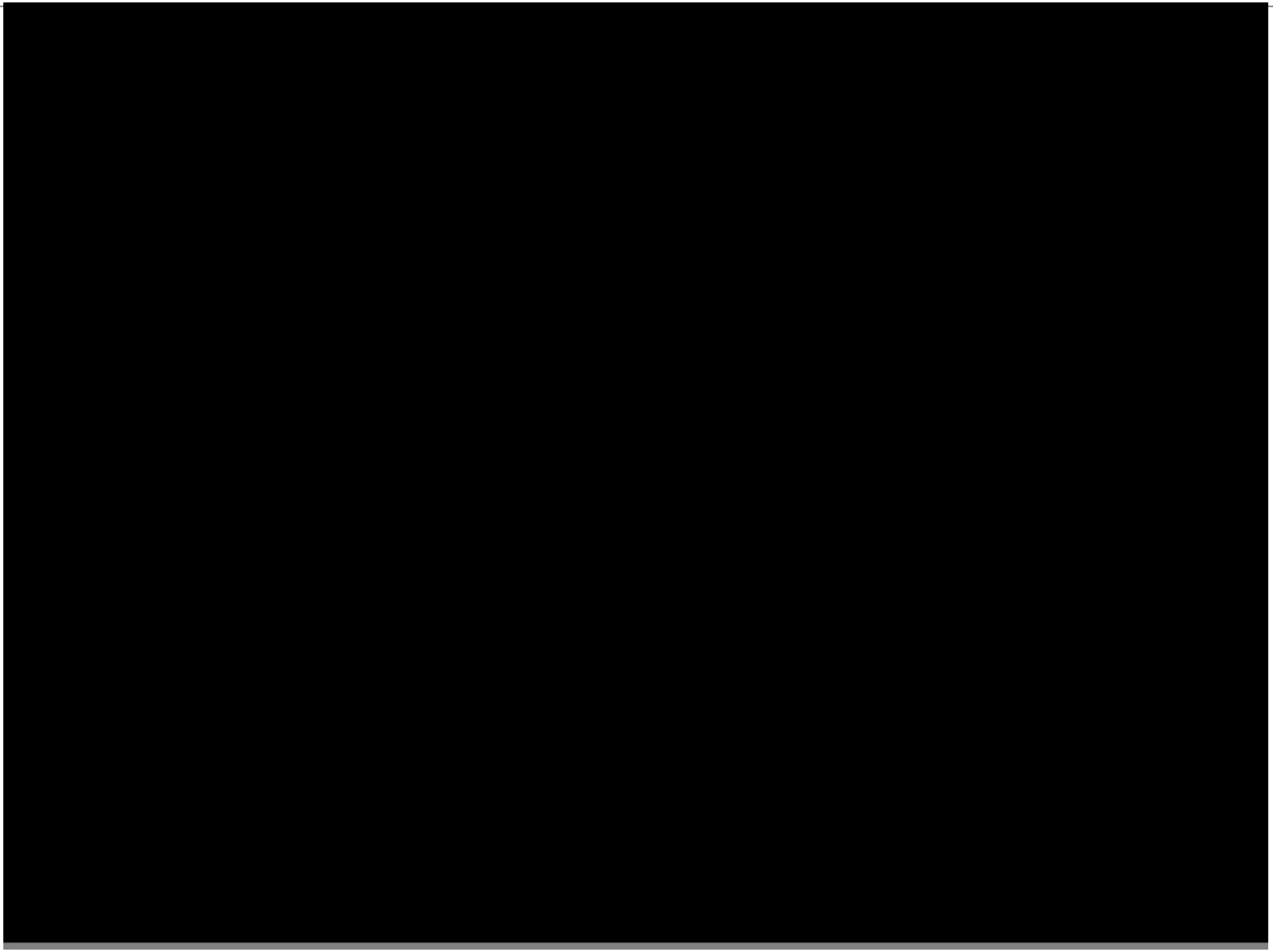
- L … number of samples in local models
- K number of local models

Fast rank-one updates of the covariance

Improved performance due to online updates!



Learning to Control: Inverse Dynamics



Outline of the Lecture

1. An Example

2. Types of Models and Learning Architectures

3. Case Study A: *Inverse Dynamics & Forward Kinematics*

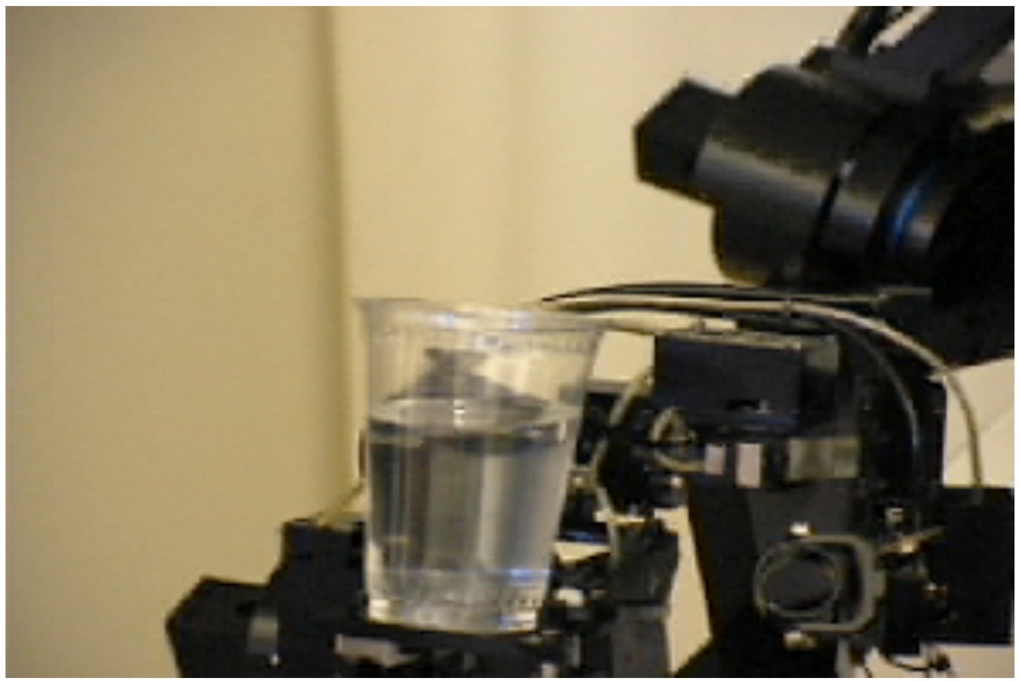
4. Case Study B: *Model Learning for Operational Space Control*

5. Case Study C: *Model Learning for Controller Falsification*

6. Final Remarks

Motivation

End-effector Position and Orientation



Balance Control

Operational space control:
learn to control in task-space

$$\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{x}}_{\mathrm{ref}}) \rightarrow \mathbf{u}$$

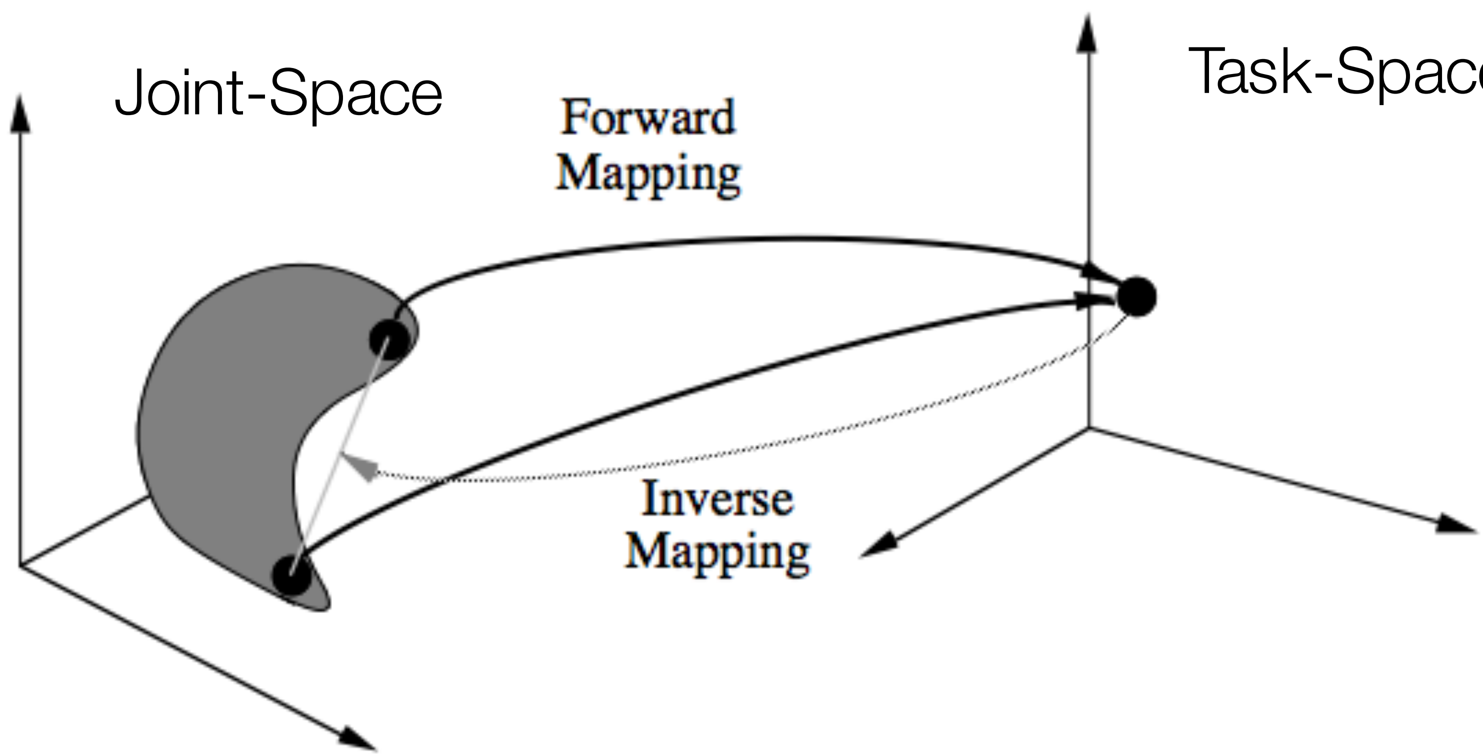
- It requires very precise analytical models!
- Complex robots can often not be modeled sufficiently accurate using rigid-body models.
- **We need to learn the models**

Peters & Schaal, R:SS 2005

Learning Operational Space Control

Why is learning the mapping $\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{x}}_{\mathrm{ref}}) \rightarrow \mathbf{u}$ difficult ?

- It requires averaging over non-convex data!



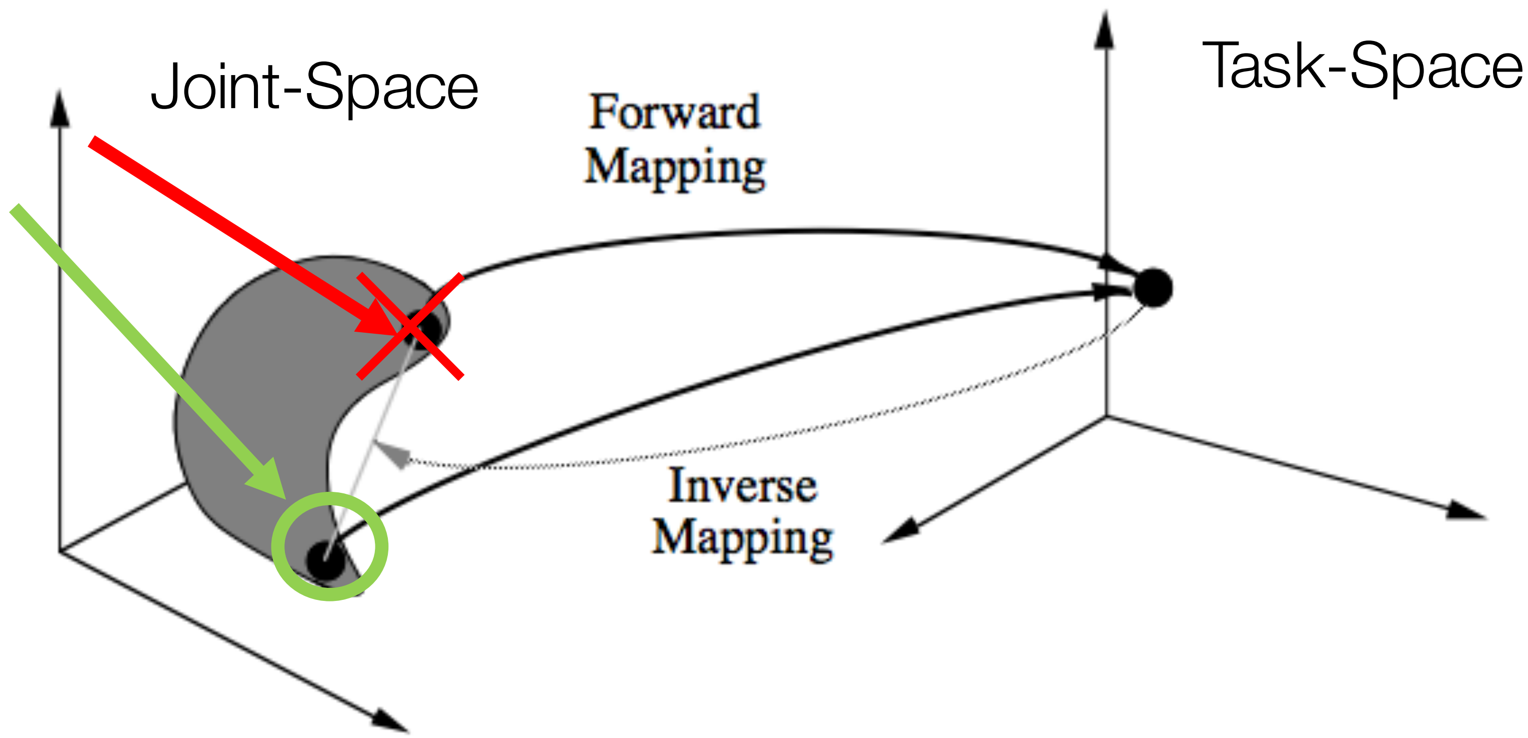
Possible Solutions:

- ➡ Linearize learned forward kinematics model
- ➡ Bias training data to come from only one mode
- ➡ Additional Regularization term to select desired solution

Peters & Schaal, ICML 2007

Compute Controllers: Basic Idea

Select one solution/mode with an additional regularization



Select solution that minimizes effort

$$\operatorname{argmax}_{\boldsymbol{u}}\, r(\boldsymbol{u}), \quad r(\boldsymbol{u}) = -\boldsymbol{u}^T \boldsymbol{H} \boldsymbol{u}$$

But still fulfills the control task

$$\ddot{\boldsymbol{x}}_{\mathrm{ref}} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$$

Peters & Schaal, ICML 2007

Compute Controllers: Basic Idea

Formalize this selection of the solution as weighted regression problem

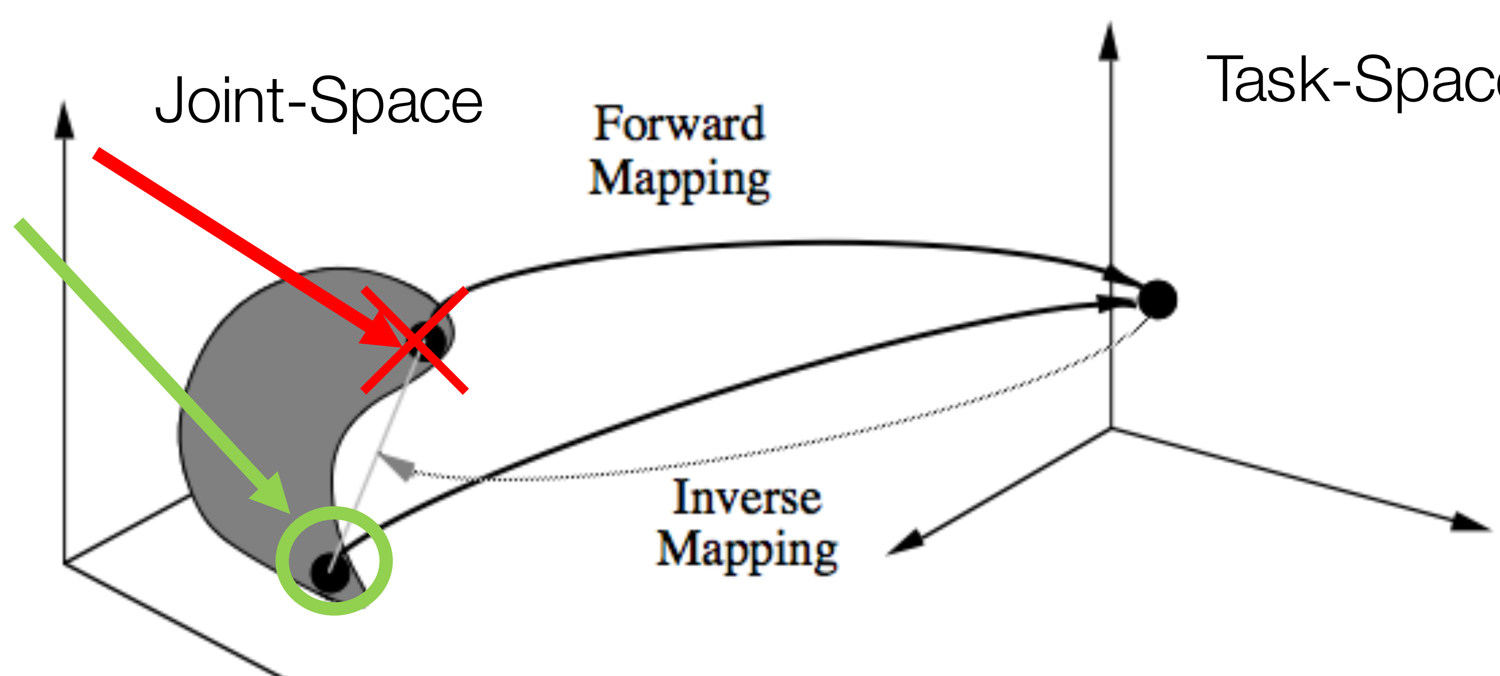
weighting $w_i \propto \exp\left(\eta r(\boldsymbol{u}_i)\right)$

$$\boldsymbol{\theta} = \operatorname{argmax}_{\boldsymbol{\theta}} \sum_i w_i \log \pi(\boldsymbol{u}_i | \boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_{\text{des}})$$

Weighted maximum likelihood!

The weighting is smaller for data from suboptimal modes

➡ Only one mode remains



Compute Controllers: Weighted Regression

Use several local linear models m_j

For each model, we use a local data-set

$$\boldsymbol{x}_i = [1, \boldsymbol{q}_i^T, \dot{\boldsymbol{q}}_i^T, \ddot{\boldsymbol{x}}_{\mathrm{des},i}^T]^T \text{ and } \boldsymbol{y}_i = \boldsymbol{u}_i$$

... where we use a reward-weighting w_i for each data point

$$w_i = \exp(-\tau \boldsymbol{u}_i^T \boldsymbol{u}_i)$$

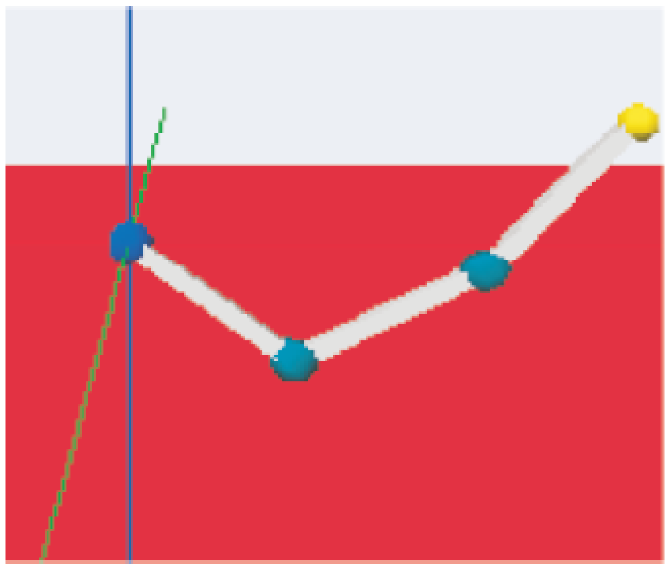
The solution for $\boldsymbol{\theta}_j$ of the local models is given by a weighted linear regression

$$\boldsymbol{\theta}_j = (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{Y}$$

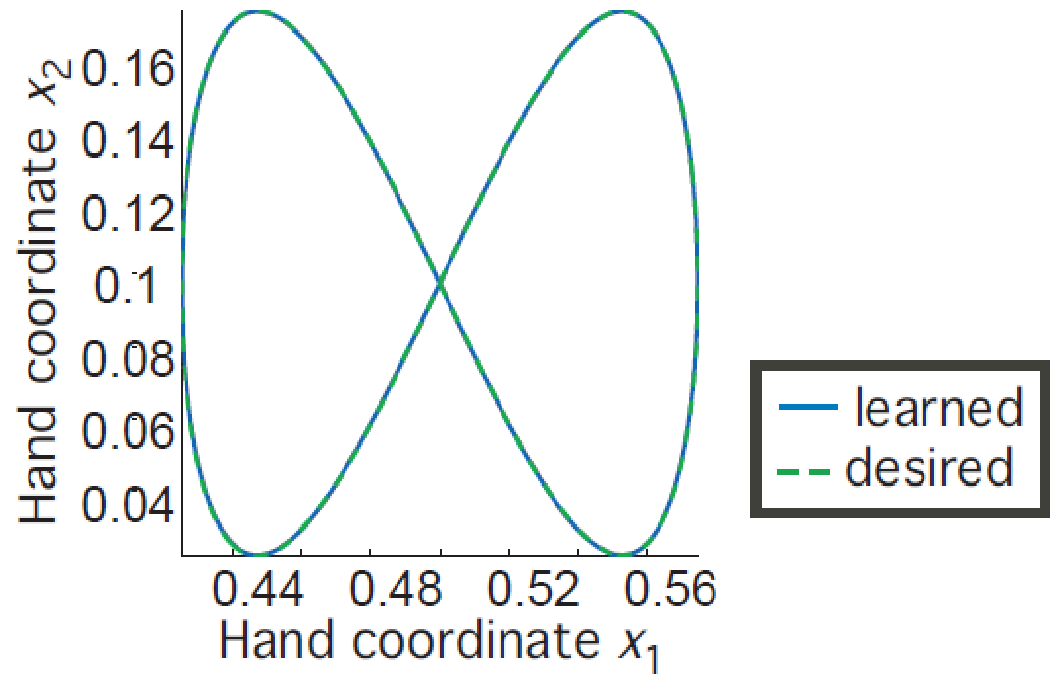
The controls provided by the local model: $\boldsymbol{u}_{t,j} = \boldsymbol{\theta}_j^T \begin{bmatrix} 1 \\ \boldsymbol{q}_t \\ \dot{\boldsymbol{q}}_t \\ \ddot{\boldsymbol{x}}_{\mathrm{des}} \end{bmatrix}$

Results: Learning Operational Space Control

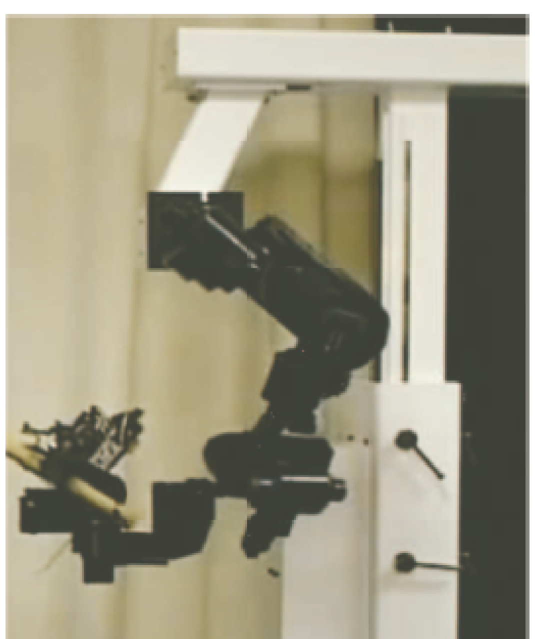
(a) 3 DoF Robot Arm



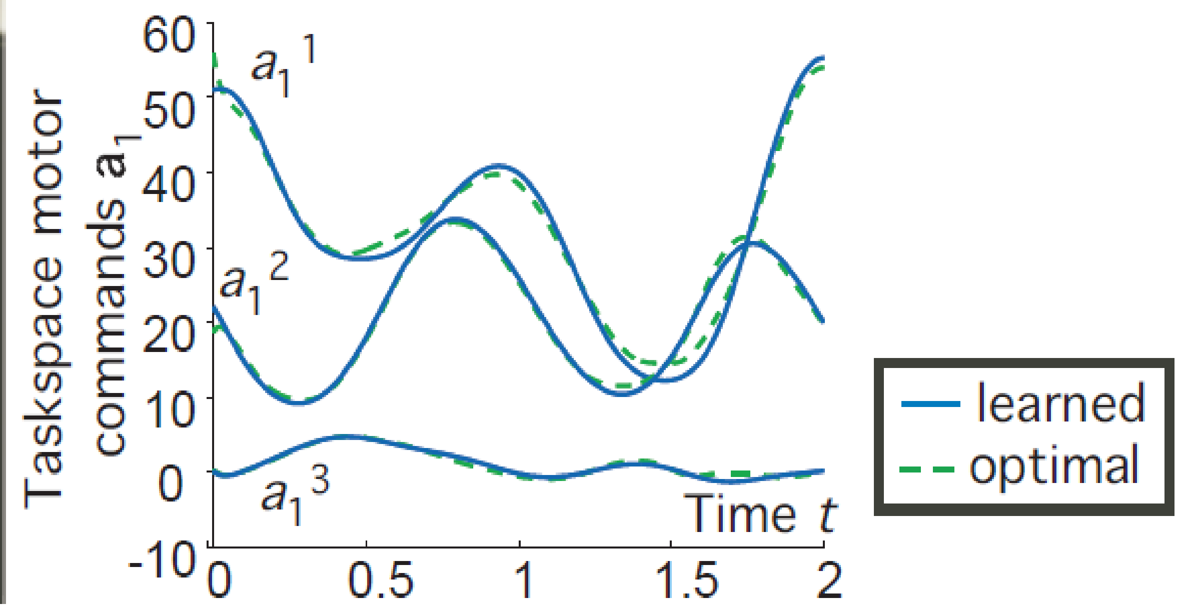
(b) Tracking Performance



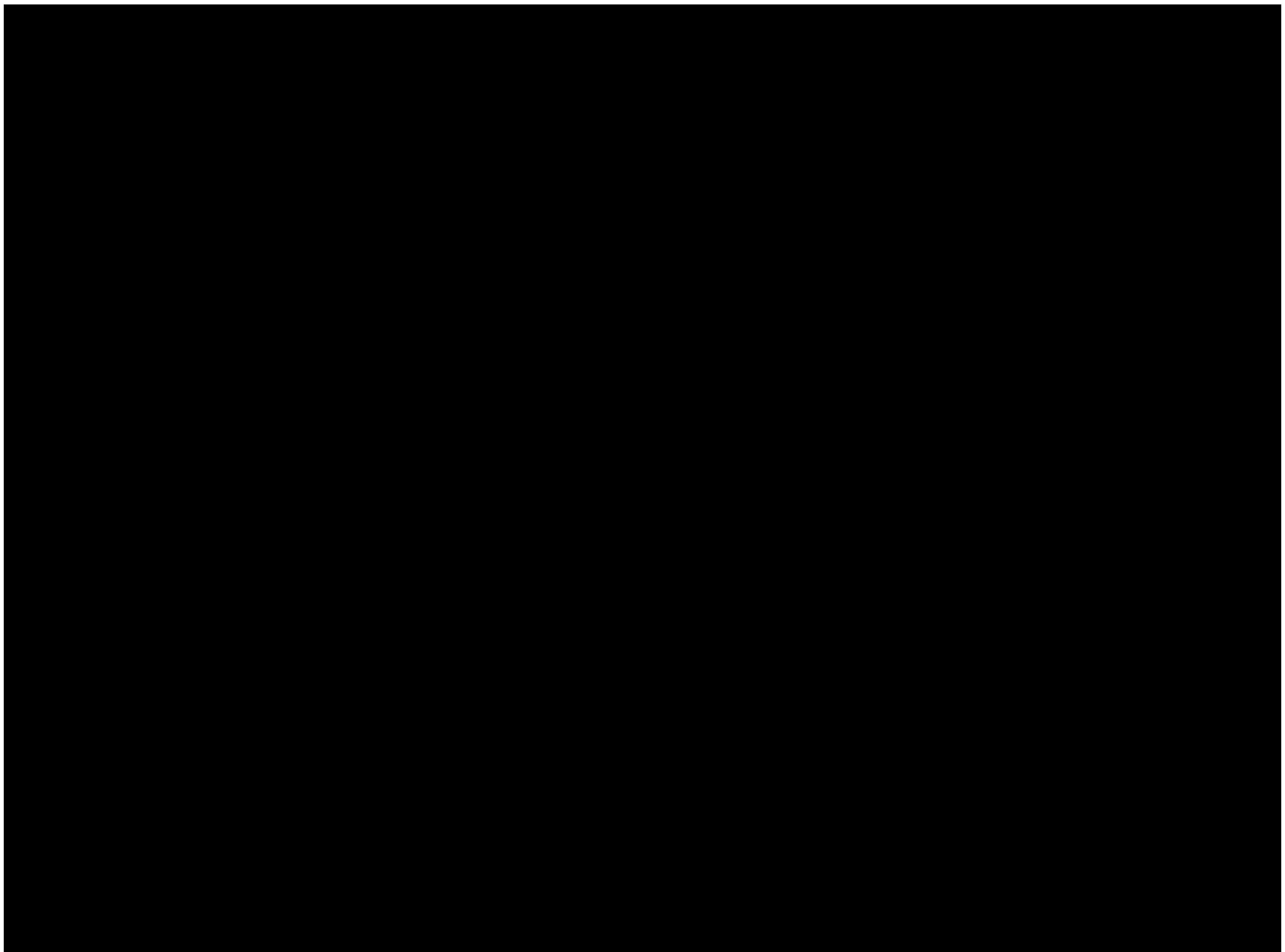
(c) SARCOS Master Robot Arm



(d) Optimal vs Learned Motor Command



Results: Learning Operational Space Control



Outline of the Lecture

1. An Example

2. Types of Models and Learning Architectures

3. Case Study A: *Inverse Dynamics & Forward Kinematics*

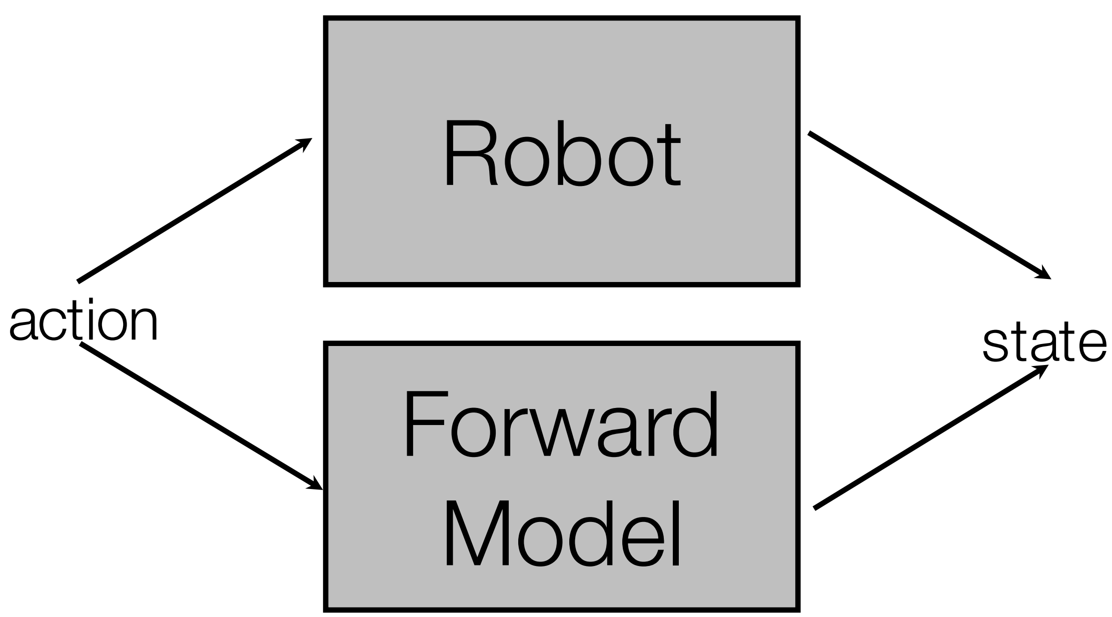
4. Case Study B: *Model Learning for Operational Space Control*

5. Case Study C: *Model Learning for Controller Falsification*

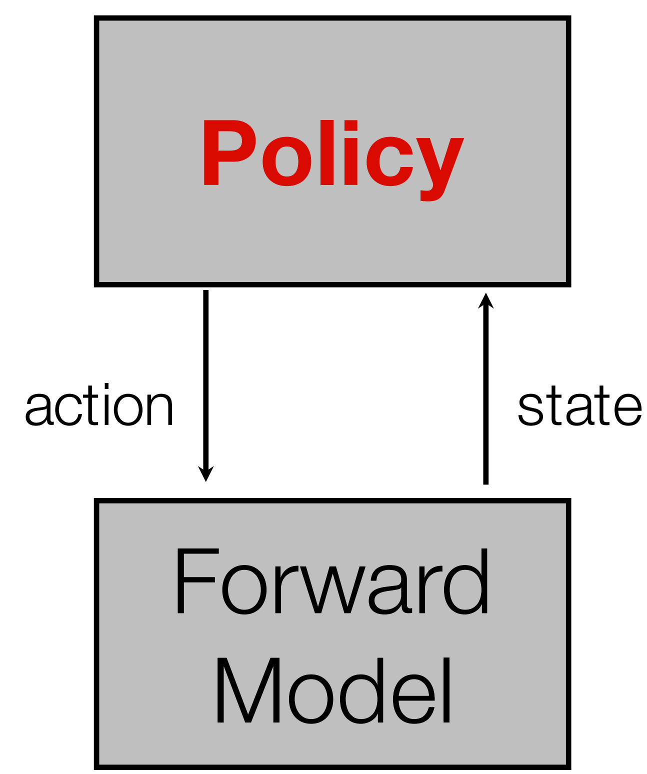
6. Final Remarks

With a Learned Simulator

1. Step: Learn an Forward Model

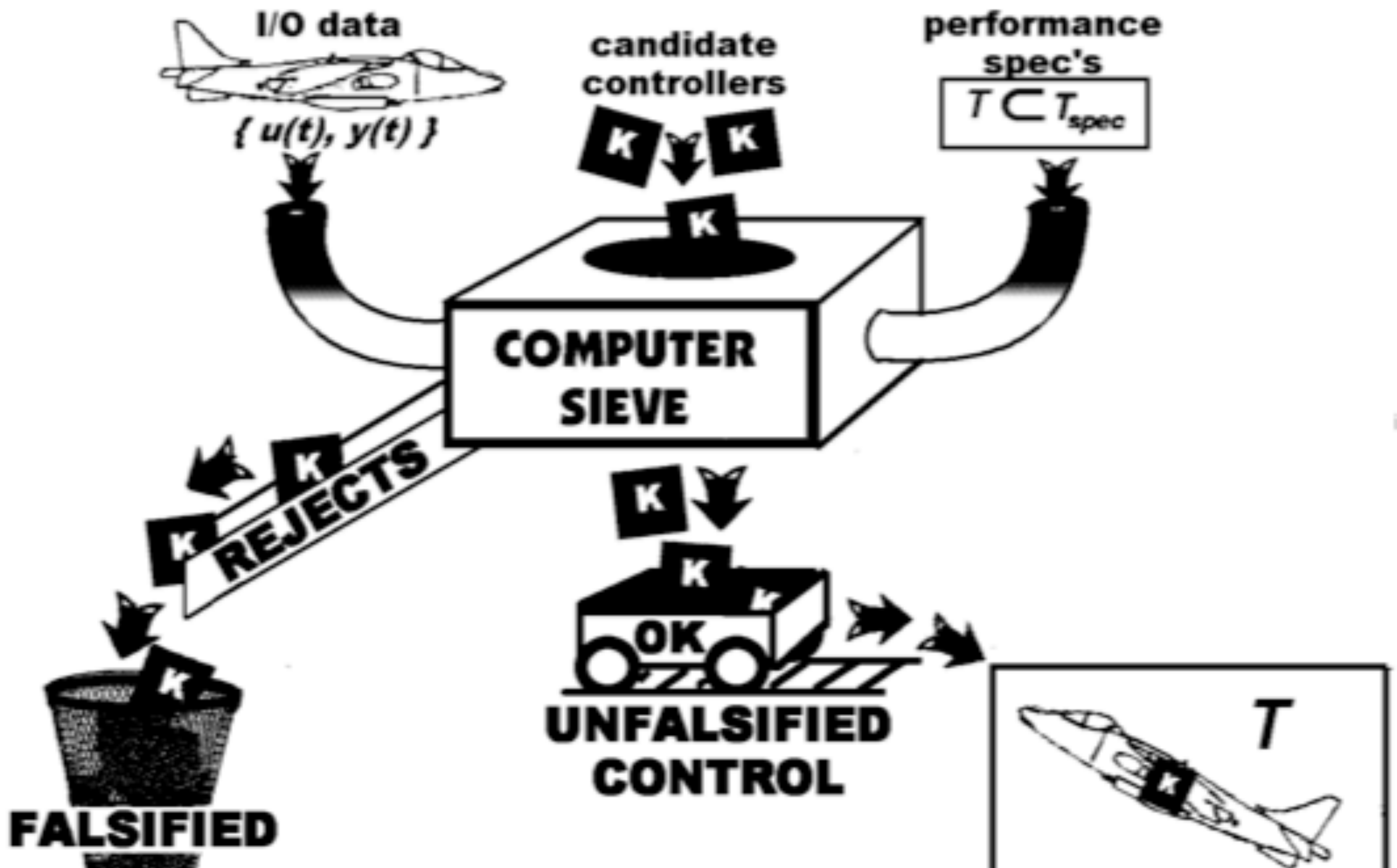


2. Step: Use your favorite *Control Method* to get an good policy



Here, a we learn a simulator - more later in RL!

Controller Falsification



Safonov: Controller Invalidation and Learning

General Recipe: Fulfill Specification

1. Learn a Forward Model

2. **Repeat** until at least one good control law was generated:

   1. Generate Control Law randomly.

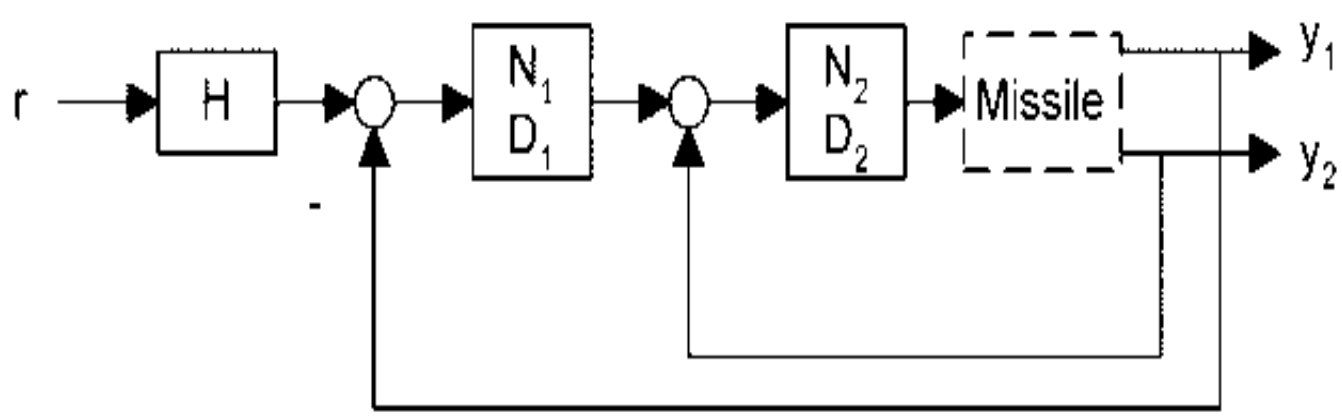
   2. Run Control Law in Learned Simulator

   3. If Control Law fails, throw it away. Otherwise: break!

3. Output

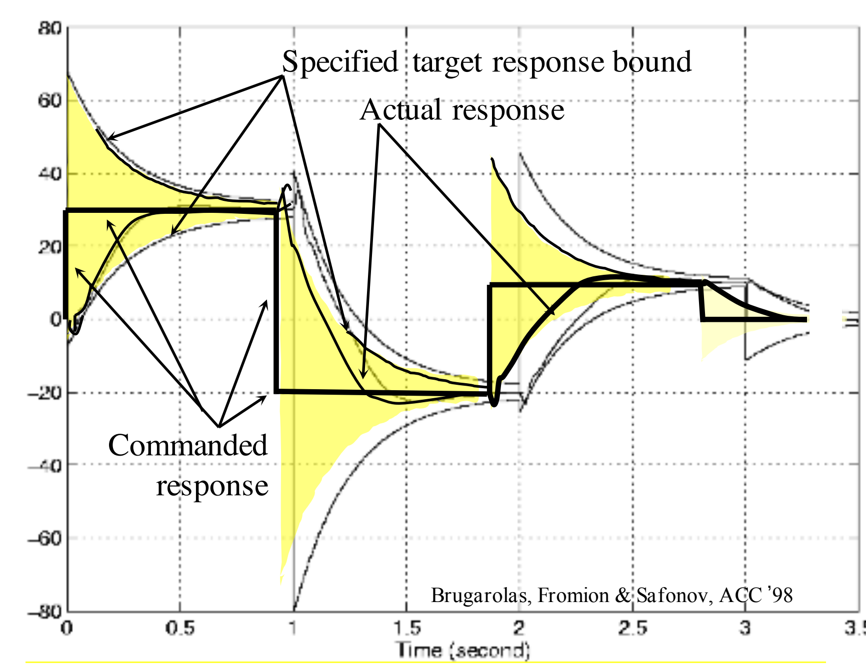
Example: Missile (M. Safonov)

- Learns control gains
- Adapts quickly to compensate for damage & failures
- Superior performance



Unfalsified adaptive missile autopilot:

- discovers stabilizing control gains as it flies, nearly instantaneously
- maintains precise sure-footed control



Brugarolas, Fromion and Safonov, ACC98

General Recipe: Fulfill Specification Perfectly

1. Learn a Forward Model

2. **Repeat** until at least one good control law was generated:

    1. Generate Control Law randomly.

    2. Run Control Law in Learned Simulator

    3. If control law does better on the metric then the last, keep it!

3. Output

Problems

- **Model Errors:**
  - If we have any errors in our model, this approach will exploit them.
- **Local Minima:**
  - We are prone to get stuck in partially fulfilled specifications if we do anything smarter than brute force sampling.
- **Stochasticity:**
  - You cannot compare trials well if they are random.

Solution

- **Add random noise:**
  - Loads of noise require more robustness and simulator errors cannot be exploited that easily.
  - Noise “washes out” the local minima.
  - BUT: the Stochasticity increases ...

- **Easy Fix**
  - Test the policy on quasi-random scenarios.
  - Can be achieved by re-using the random numbers!
    - Long known in the simulation community...
  - E..g., by resetting the random seed of the simulator to always the same value when testing a policy.
    - Known as Pegasus (Ng et al.) but much older.

Video: Inverted Helicopter

QuickTime™ and a
mpeg4 decompressor
are needed to see this picture.

Further Reading

- M. G. Safonov. Focusing on the knowable: Controller invalidation and learning. In A. S. Morse, editor, Control Using Logic-Based Switching, pages 224–233. Springer-Verlag, Berlin, 1996.
- M. G. Safonov and T. C. Tsao. The unfalsified control concept and learning. IEEE Trans. Autom. Control, AC-42(6):843–847, June 1997.
- Unfalsified direct adapative control of a two-link robot arm. T.-C. Tsao and M. G. Safonov, Proc. IEEE CCA/CACSD, Kohala Coast–Island of Hawaii, HI, August 22-27, 1999.
- Inverted autonomous helicopter flight via reinforcement learning, Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger and Eric Liang. ISER 2004.
- PEGASUS: A policy search method for large MDPs and POMDPs, Andrew Y. Ng and Michael Jordan. UAI 2000.

Outline of the Lecture

1. An Example
2. Types of Models and Learning Architectures
3. Case Study A: *Inverse Dynamics & Forward Kinematics*
4. Case Study B: *Model Learning for Operational Space Control*
5. Case Study C: *Model Learning for Controller Falsification*
6. Final Remarks

Conclusion

- When directly learnable, **learn the model**!
- Learning inverse models often requires learning from multiple non-convex solutions
- Inverse models are useful, if you can, learn them
- **Learning good models** can sometimes be very hard