

Source: [https://ttechcrunch2011.files.wordpress.com/2016/02/shutterstock\\_147776027.jpg?w=1279&h=727&crop=1](https://ttechcrunch2011.files.wordpress.com/2016/02/shutterstock_147776027.jpg?w=1279&h=727&crop=1)

# Robotics for Future Industrial Applications

## Modelbased Reinforcement Learning

Philipp Ennen, M.Sc.

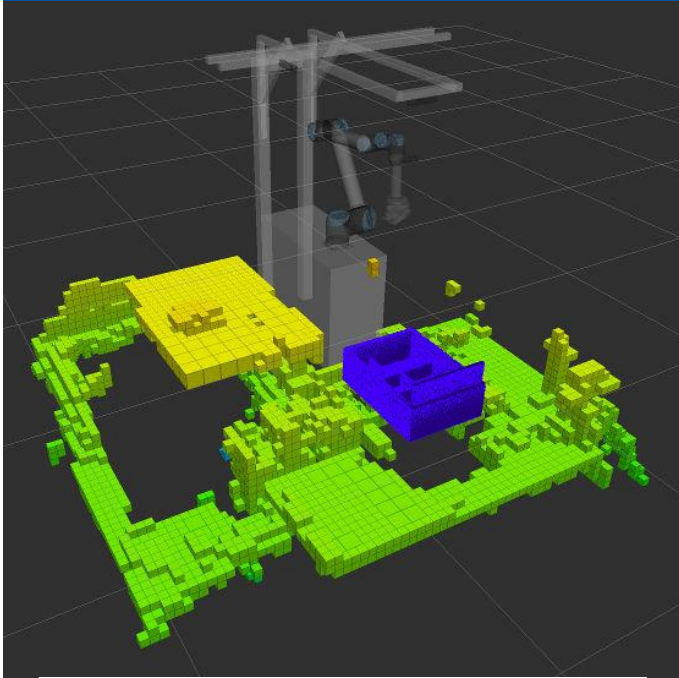




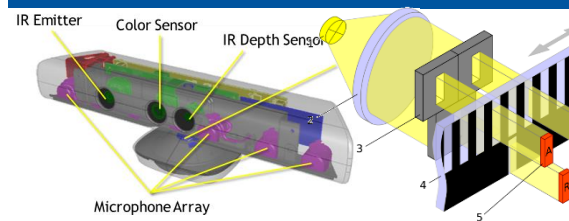
# Introduction

## How do I (the robot) go there?

### Model of the Robot and Environment



### Sensing

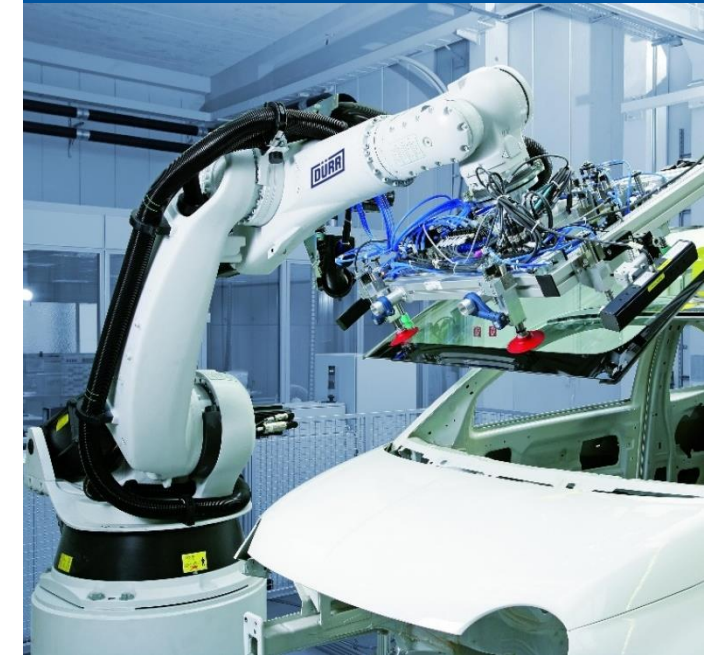


### Motion Planning with iLQR

Requires Goalstate:

- i.e. hand-engineered
- i.e. via a cost function

### Motion Control and Execution

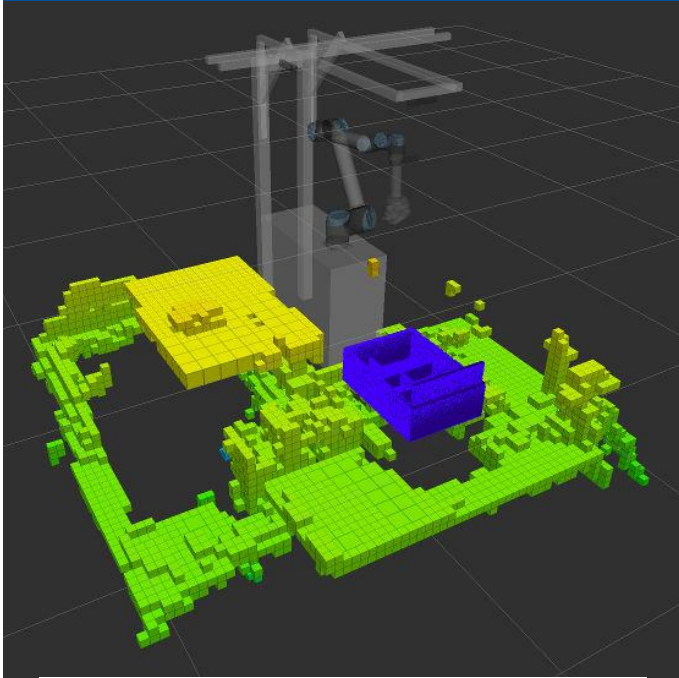


$$\begin{bmatrix} \ddot{x} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{-(I + ml^2)b}{I(M+m) + Mml^2} & \frac{m^2 gl^2}{I(M+m) + Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m) + Mml^2} & \frac{mgl(M+m)}{I(M+m) + Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I + ml^2}{I(M+m) + Mml^2} \\ 0 \\ \frac{ml}{I(M+m) + Mml^2} \end{bmatrix} u$$

# Introduction

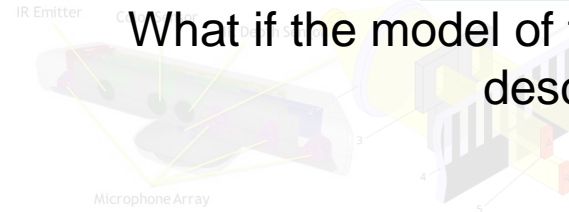
## How do I (the robot) go there?

### Model of the Robot and Environment



$$\begin{bmatrix} \ddot{x} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{-(I + ml^2)b}{I(M+m) + Mmi^2} & \frac{m^2 gl^2}{I(M+m) + Mmi^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m) + Mmi^2} & \frac{mgl(M+m)}{I(M+m) + Mmi^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I + ml^2}{I(M+m) + Mmi^2} \\ 0 \\ \frac{ml}{I(M+m) + Mmi^2} \end{bmatrix} u$$

### Sensing



What if the model of the robot and environment is hard to describe (or unknown)?

Think about flexible objects!

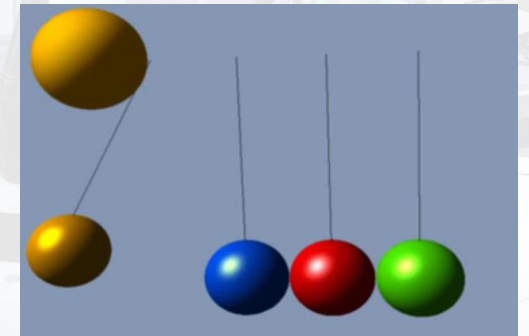


Requires Goalstate:

- i.e. hand-engineered
- i.e. via a cost function

### Motion Control and Execution

Think about contact-situations!



## Dynamics of unstructured environments



## Trajectory Optimization

- Trajectory Optimization: Calculates **optimal sequence of actions** using **cost-function** and **dynamics**

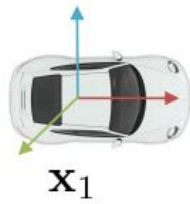
$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t) \quad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

- Today: How can optimal action sequences be calculated if a dynamic model does not exist?
- Today we learn an algorithm based on
  - Learning a global dynamic model (“model-based reinforcement learning”)
  - Learning a local dynamic model

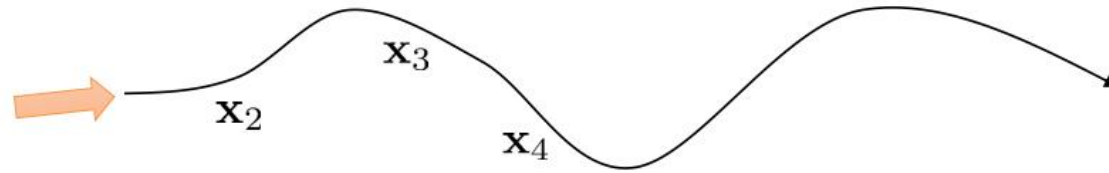


## Real-World Dynamics are Complex!

Often dynamic models exist



Dynamics of a car



Dynamic models usually do not exist



Dynamics of contacts



Dynamics of flexible objects



Dynamics of unstructured environments

## **I. Modelbased Reinforcement Learning**

- I. Learning of dynamic models
- II. Learning of dynamic models and policies

## **II. Representing a dynamic model**

## **III. Global and local dynamic model**

## **IV. Learning with local dynamic models with „Trust Regions“**

Why do we want to learn the dynamics?

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t) \quad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$



$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$

Usual procedure: Differentiate via Backpropagation and optimize (i.e. iLQR)

Requires:  $\frac{\partial f}{\partial \mathbf{x}_t}, \frac{\partial f}{\partial \mathbf{u}_t}, \frac{\partial c}{\partial \mathbf{x}_t}, \frac{\partial c}{\partial \mathbf{u}_t}$



### Why do we want to learn the dynamics?

- If  $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$  is known, we can do trajectory optimization
  - In the stochastic case  $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$



Learn  $f(\mathbf{x}_t, \mathbf{u}_t)$  with subsequent backpropagation (i.e. iLQR)

### Modelbased Reinforcement Learning Version 0.5

1. Execute initial policy  $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$  (i.e. a random policy) and collect data  $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. Learn dynamics  $f(\mathbf{x}, \mathbf{u})$  that minimizes  $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. Backpropagate  $f(\mathbf{x}, \mathbf{u})$  and calculate sequence of actions (i.e. iLQR)

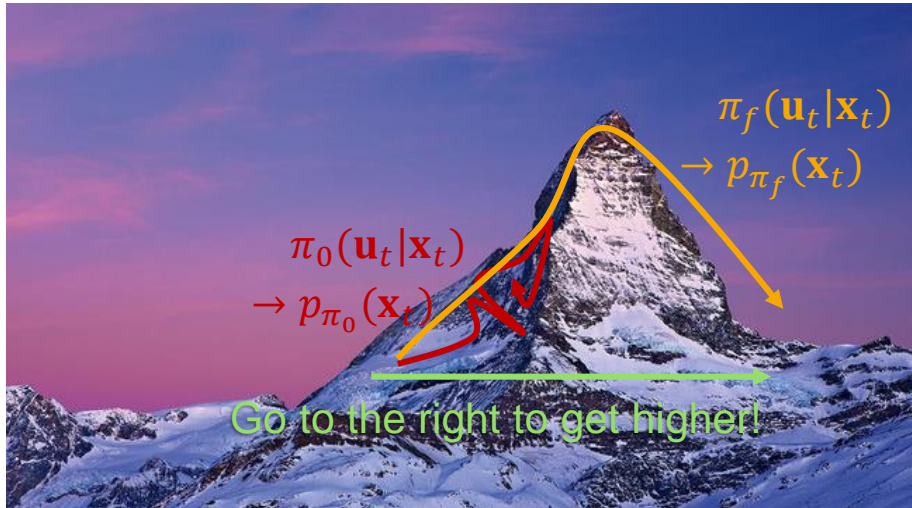
### Does Version 0.5 work?

(often) **YES!**

- Traditional system identification uses this method (control theory)
- Initial policy must be chosen with caution
- Version 0.5 is very effective
  - If a representation of the dynamics based on physical laws exists
  - If only a few parameters must be learned

## Does Version 0.5 work?

(in general) **NO!**



1. Execute initial policy  $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$  (i.e. a random policy) and collect data  $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. Learn dynamics  $f(\mathbf{x}, \mathbf{u})$  that minimizes  $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. Backpropagate  $f(\mathbf{x}, \mathbf{u})$  and calculate sequence of actions (i.e. iLQR)  $\rightarrow \pi_f(\mathbf{u}_t|\mathbf{x}_t)$

$$p_{\pi_0}(\mathbf{x}_t) \neq p_{\pi_f}(\mathbf{x}_t)$$

(Distribution Mismatch Problem)



Distribution Mismatch Problem increases if expressive classes of models are used (i.e. neural networks)



Can we do better?

Can we make  $p_{\pi_0}(\mathbf{x}_t) = p_{\pi_f}(\mathbf{x}_t)$ ?

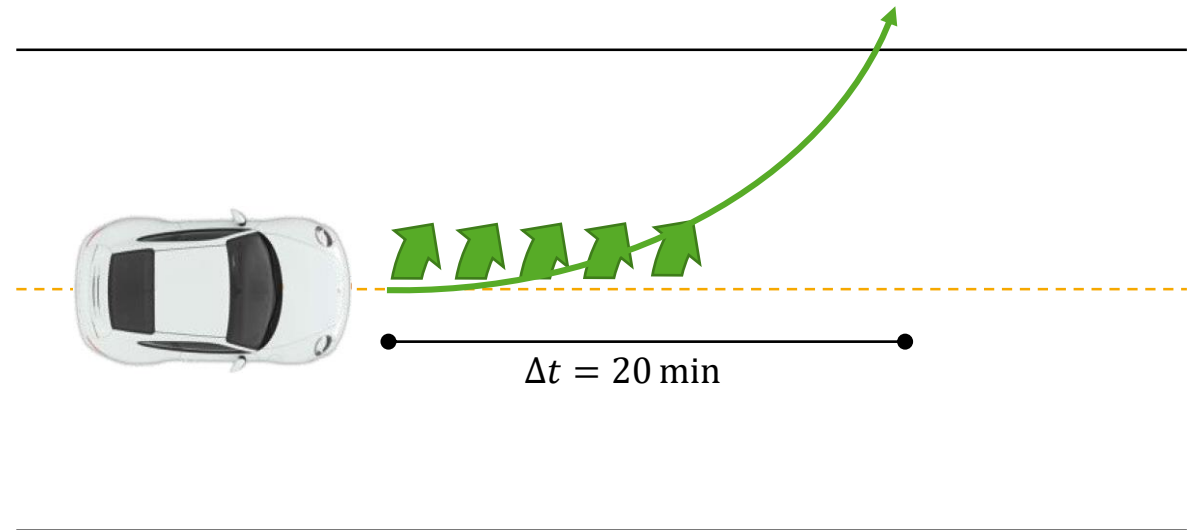


Need to collect data from  $p_{\pi_f}(\mathbf{x}_t)$ !

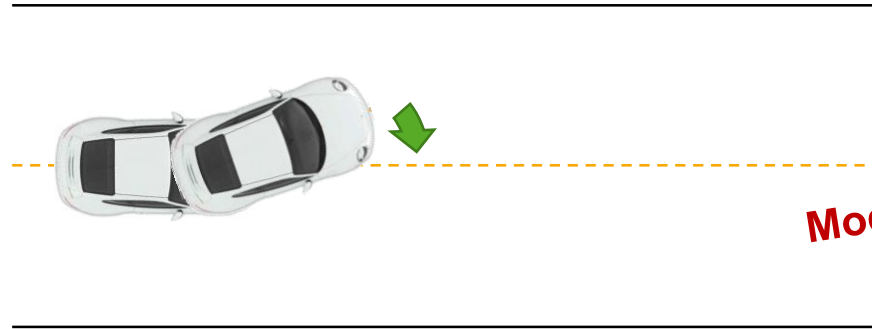
### Modellbasiertes Reinforcement Learning Version 1.0

1. Execute initial policy  $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$  (i.e. a random policy) and collect data  $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. Learn dynamics  $f(\mathbf{x}, \mathbf{u})$  that minimizes  $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
3. Backpropagate  $f(\mathbf{x}, \mathbf{u})$  and calculate sequence of actions (i.e. iLQR)
4. Execute those actions and add the resulting data  $\{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$  to  $\mathcal{D}$

What happens if the dynamic models contains little error?



Can we do better?



**Model predictive control  
helps with error**

## Modellbasiertes Reinforcement Learning Version 1.5

1. Execute initial policy  $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$  (i.e. a random policy) and collect data  $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
  2. Learn dynamics  $f(\mathbf{x}, \mathbf{u})$  that minimizes  $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
  3. Backpropagate  $f(\mathbf{x}, \mathbf{u})$  and calculate sequence of actions (i.e. iLQR)
  4. Execute the first planned action, observe resulting state  $\mathbf{x}'$  (MPC)
  5. Append  $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$  to dataset  $\mathcal{D}$
- every N steps



## Summary

- Version 0.5: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem
- Version 1.0: iteratively collect data, replan, collect data
  - Pro: simple, solves distribution mismatch
  - Con: open loop plan might perform poorly, exp. in stochastic domains
- Version 1.5: iteratively collect data using MPC (replan in each step)
  - Pro: robust to small model errors
  - Con: computationally expensive, but have planning algorithm available

## **I. Modelbased Reinforcement Learning**

- I. Learning of dynamic models
- II. Learning of dynamic models and policies

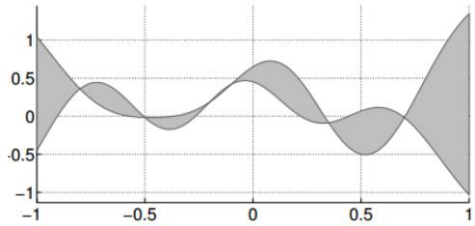
## **II. Representing a dynamic model**

## **III. Global and local dynamic model**

## **IV. Learning with local dynamic models with „Trust Regions“**

# What kind of models can we use?

## Gaussian process



GP with input  $(\mathbf{x}, \mathbf{u})$  and output  $\mathbf{x}'$

Pro: very data-efficient

Con: not great with non-smooth dynamics

Con: very slow when dataset is big

## Neural Network

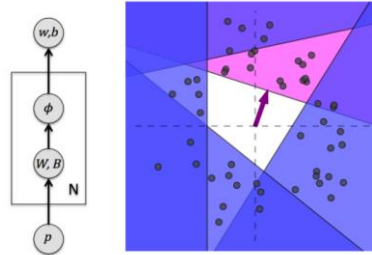


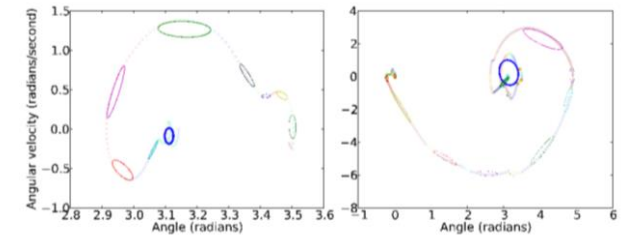
image: Punjani & Abbeel '14

Input is  $(\mathbf{x}, \mathbf{u})$ , output is  $\mathbf{x}'$

Pro: very expressive, can use lots of data

Con: not so great in low data regimes

## Gaussian Mixture Model



GMM over  $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$  tuples

Train on  $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ , condition to get  $p(\mathbf{x}'|\mathbf{x}, \mathbf{u})$

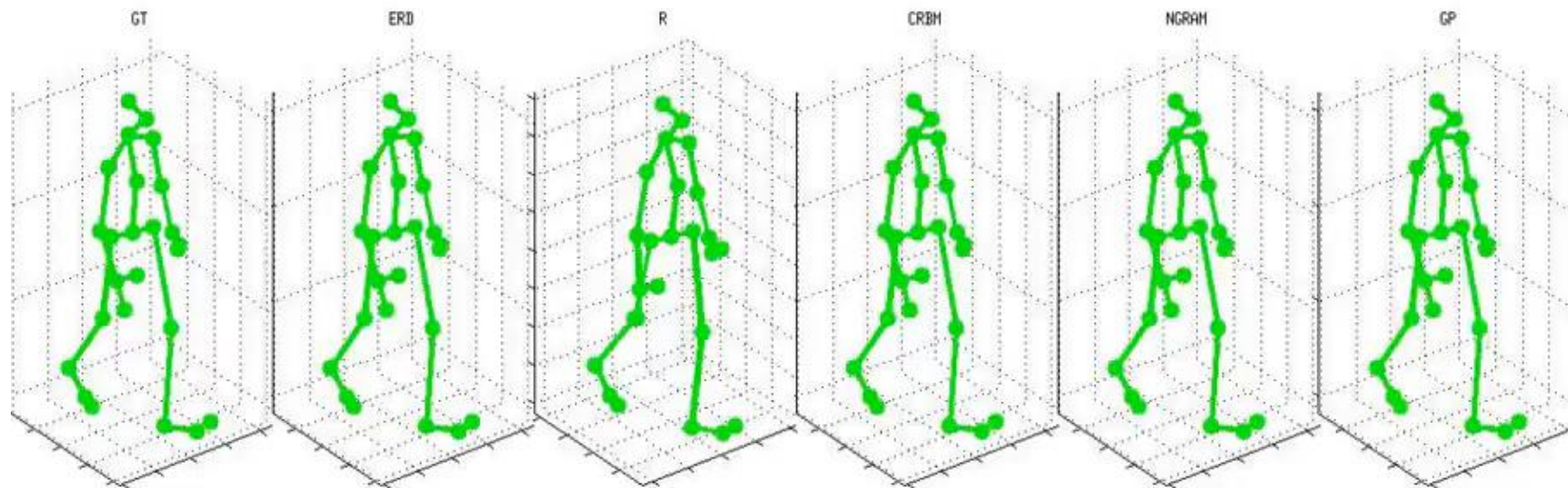
For  $i$ 'th mixture element,  $p_i(\mathbf{x}, \mathbf{u})$  gives region where the mode  $p_i(\mathbf{x}'|\mathbf{x}, \mathbf{u})$  holds

Pro: very expressive, if the dynamics can be assumed as piecewise linear



# Representation of Dynamic Models

---



## **I. Modelbased Reinforcement Learning**

- I. Learning of dynamic models
- II. Learning of dynamic models and policies

## **II. Representing a dynamic model**


## **III. Global and local dynamic model**

## **IV. Learning with local dynamic models with „Trust Regions“**

## Challenges

Example: Global dynamic model  $f(\mathbf{x}_t, \mathbf{u}_t)$  is represented by a neural network

### Modellbasiertes Reinforcement Learning Version 1.0

- 
1. Execute initial policy  $\pi_0(\mathbf{u}_t | \mathbf{x}_t)$  (i.e. a random policy) and collect data  $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
  2. Learn dynamics  $f(\mathbf{x}, \mathbf{u})$  that minimizes  $\sum_i \|f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}'_i\|^2$
  3. Backpropagate  $f(\mathbf{x}, \mathbf{u})$  and calculate sequence of actions (i.e. iLQR)
  4. Execute those actions and add the resulting data  $\{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$  to  $\mathcal{D}$

- Planner will seek out regions where the model is erroneously optimistic
- Need to find a very good model in most of the state space to converge on a good solution



## The trouble with global models

- Planner will seek out regions where the model is erroneously optimistic
- Need to find a very good model in most of the state space to converge on a good solution
- In some tasks, the model is much more complex than the policy



### Motivation

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t) \quad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$



$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$

Usual story: differentiate via backpropagation and optimize (i.e. iLQR)

need:  $\frac{\partial f}{\partial \mathbf{x}_t}, \frac{\partial f}{\partial \mathbf{u}_t}, \frac{\partial c}{\partial \mathbf{x}_t}, \frac{\partial c}{\partial \mathbf{u}_t}$

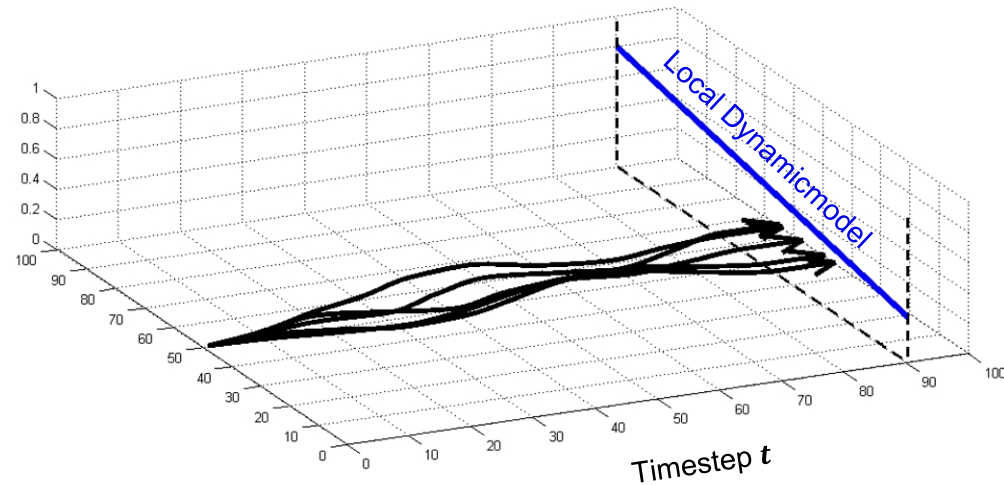
### Approach

need:

$$\frac{\partial f}{\partial \mathbf{x}_t}, \frac{\partial f}{\partial \mathbf{u}_t}, \frac{\partial c}{\partial \mathbf{x}_t}, \frac{\partial c}{\partial \mathbf{u}_t}$$

idea: just fit  $\frac{\partial f}{\partial \mathbf{x}_t}, \frac{\partial f}{\partial \mathbf{u}_t}$  around current trajectory or policy

$p(\mathbf{u}_t | \mathbf{x}_t)$  – time-varying linear-Gaussian controller –  
can **execute** on the robot and produces  
**trajectory distribution**

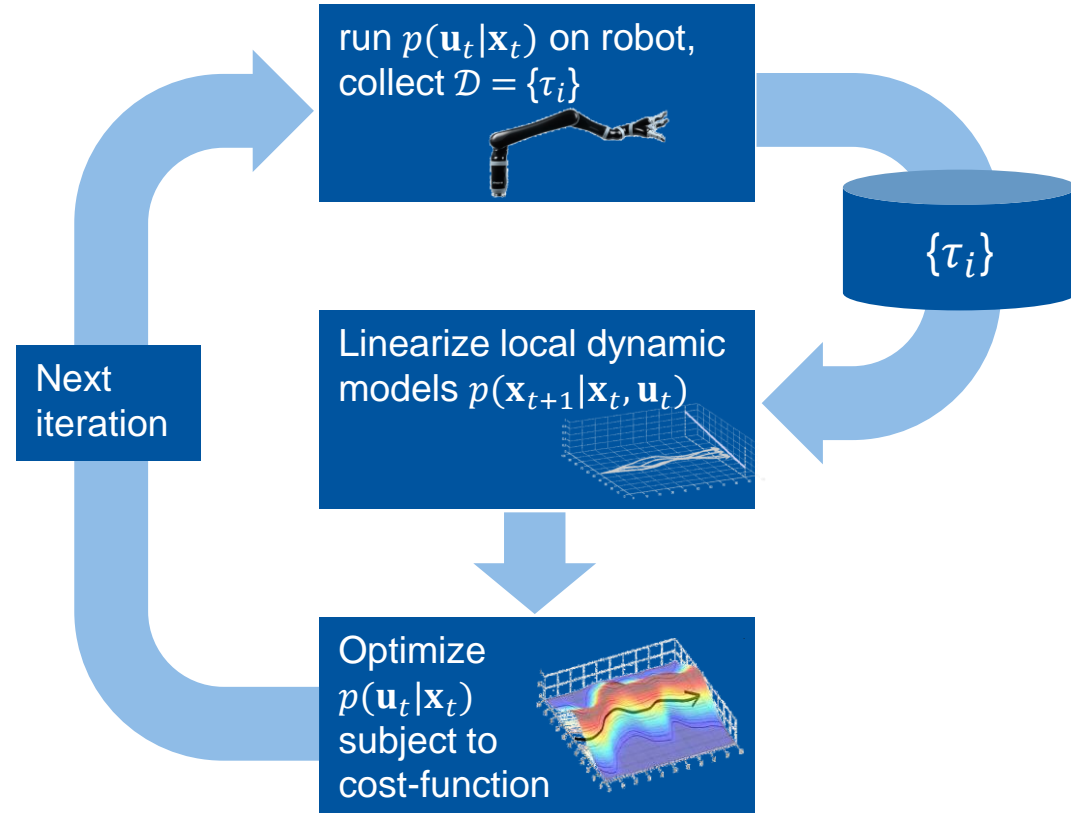


### Learning a policy

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t), \Sigma)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

$$\mathbf{A}_t = \frac{\partial f}{\partial \mathbf{x}_t} \quad \mathbf{B}_t = \frac{\partial f}{\partial \mathbf{u}_t}$$





### Time-dependent Linear Gaussian Controller

run  $p(\mathbf{u}_t|\mathbf{x}_t)$  on robot,  
collect  $\mathcal{D} = \{\tau_i\}$



iLQR produces:  $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t, \mathbf{K}_t, \mathbf{k}_t$

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

Set  $\Sigma_t = \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1}$

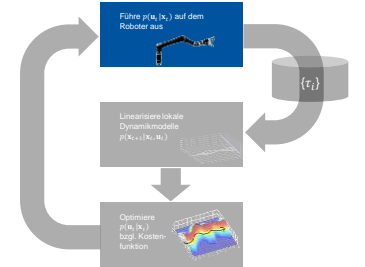
$Q(\mathbf{x}_t, \mathbf{u}_t)$  is the cost to go: total cost we get after taking an action  $\mathbf{u}_t$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}$  is big, if changing  $\mathbf{u}_t$   
changes the Q-value a lot!



If  $\mathbf{u}_t$  changes Q-value a lot, don't  
vary  $\mathbf{u}_t$  so much. Exploration  
noise  $\Sigma_t$  must be low



### Time-dependent Linear Gaussian Controller

run  $p(\mathbf{u}_t|\mathbf{x}_t)$  on robot,  
collect  $\mathcal{D} = \{\tau_i\}$



iLQR produces:  $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t, \mathbf{K}_t, \mathbf{k}_t$

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

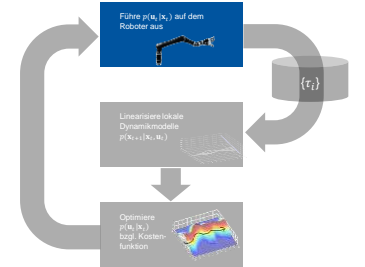
Set  $\Sigma_t = \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1}$

Standard LQR solves  $\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t)$

Linear-Gaussian solution solves  $\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} \sum_{t=1}^T E[c(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{H}(p(\mathbf{u}_t|\mathbf{x}_t))]$

**Maximum Entropy:** act as randomly as possible while minimizing cost

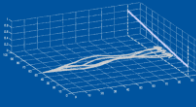
– Entropy: A measure for the average information content



### Linearize local dynamics

Linearize local dynamic models

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$



$$\{(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})_i\}$$

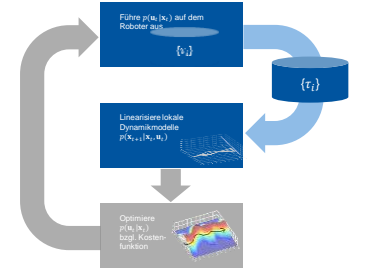
Version 1.0: Linearize  $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$  at each time step using linear regression

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \mathbf{c}_t, \mathbf{N}_t) \quad \mathbf{A}_t \approx \frac{\partial f}{\partial \mathbf{x}_t} \quad \mathbf{B}_t \approx \frac{\partial f}{\partial \mathbf{u}_t}$$

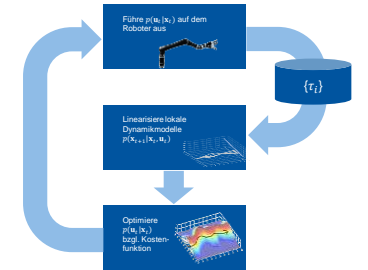
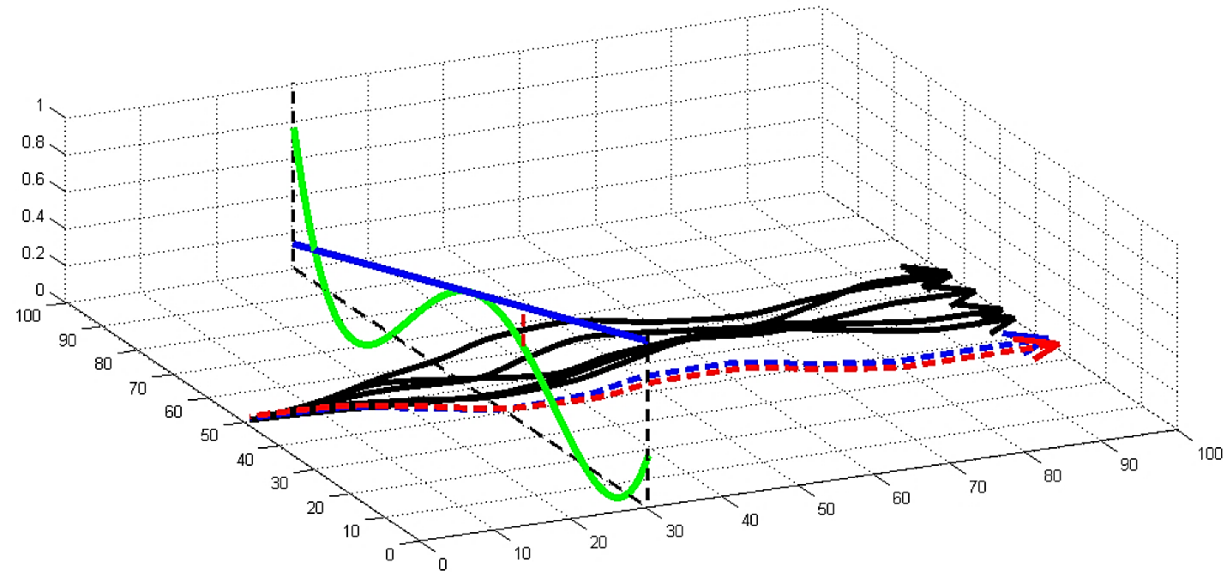
### Can we do better?

Version 2.0: Linearize  $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$  using *Bayesian* linear regression

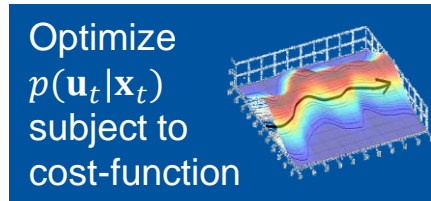
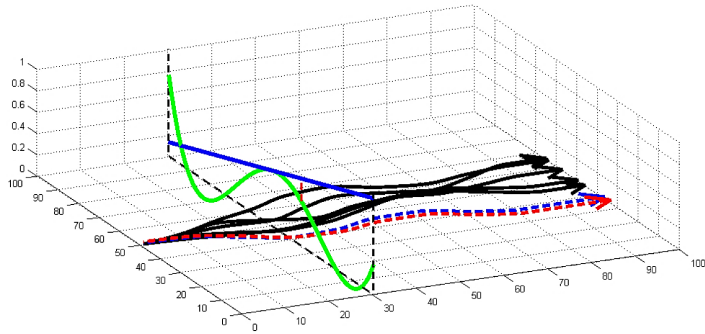
- Bayesian linear regression uses prior:  $p(\mathbf{x}_t, \mathbf{u}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = p(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$
- Use your favourite global model as a prior (GP, deep net, GMM)



## How to stay close to old controller?



### How to stay close to old controller?



$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

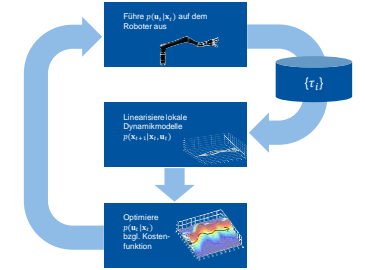
$$p(\tau) = p(\mathbf{x}_1) \prod_{t=1}^T p(\mathbf{u}_t|\mathbf{x}_t) p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$

New trajectory distribution  $p(\tau)$  must be similar to the old one  $\bar{p}(\tau)$

If trajectory distribution is close, the dynamics will be close too!

What does “close” mean?

Kullback-Leibler divergence:  $D_{KL}(p(\tau) || \bar{p}(\tau)) < \varepsilon$  **From here comes a lot of mathematics!**





# Its your turn!

---

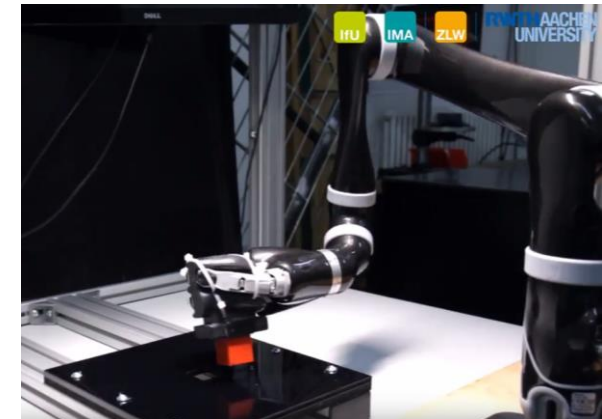
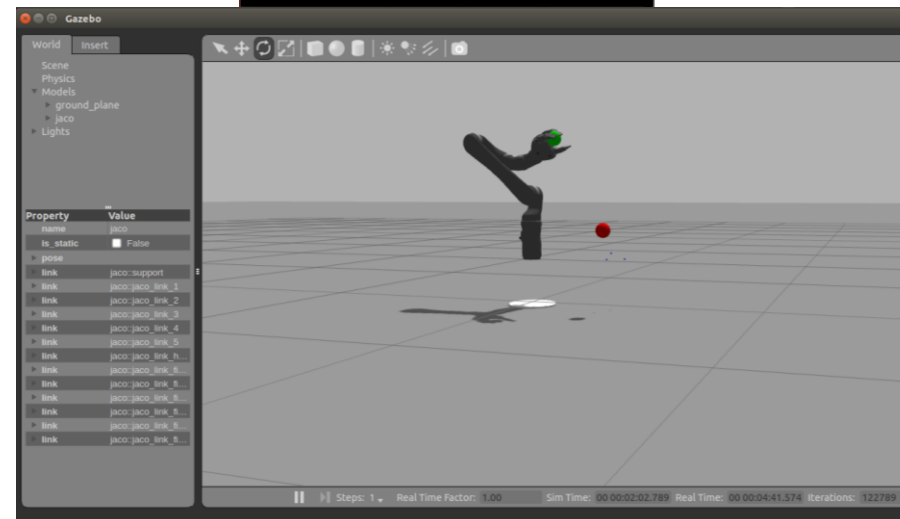
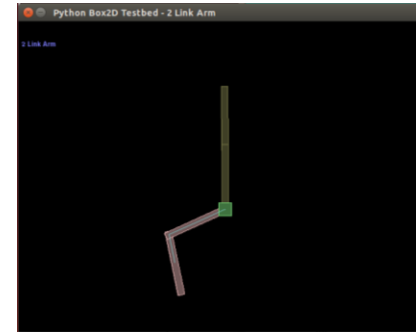
## Visit the website and implement it!



# Introduction to the tasks

## Tasks for today and tomorrow

- Task 1:
  - Implement an LQR Backward and Forward pass
  - Try to understand it!
  - Test it with our test method
- Task 2:
  - Implement linearization of the dynamic model
  - Try to understand it!
  - Test it with our test-method
  - Test it on the Box2D Scenario
- Task 3:
  - Test it with Kinova Jaco 2 in simulation
  - Adjust cost function
- Task 4:
  - Test it with real Kinova Jaco 2
  - Adjust cost function



# Task 1 – Installation procedure

---

Download source code (do it in your home directory: `cd ~`):

```
git clone https://github.com/philippente/ss2017\_task1\_lqr.git
```

Edit `.bashrc` to set environment variables:

```
gedit ~/.bashrc
```

At the end of file, the lines should look like this:

```
source /opt/ros/indigo/setup.bash
source /home/useradmin/catkin_ws/devel/setup.bash
export
ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:/opt/ros/indigo/share:/opt/ros/indigo/stacks:/home/useradmin/ss2017_task1_lqr:/home/useradmin/ss2017_task1_lqr/src/gps_agent_pkg
```

**Check if the blue part of the source folder and `ROS_PACKAGE_PATH` is correct!**

Then save it and close it. Source the `.bashrc` (load the environment variables):

```
source ~/.bashrc
```

Now, compile some stuff:

```
cd ss2017_task1_lqr
sh compile_proto.sh
cd /src/gps_agent_pkg
cmake .
make -j
```

## Task 1 – Installation procedure

---

- Open PyCharm
- Import the folder *ss2017\_task1\_lqr* as a new project
- Open within PyCharm: `python/gps/algorithm/algorithm_traj_opt.py`
- **Task: Implement the forward and backward pass of an LQR! Look at the website for advices` : <https://goo.gl/X5twgi>**
- You can test your implementation with a little test program
  - using a terminal, open the directory *ss2017\_task1\_lqr*
  - Start the program with: `python python/gps/lqr_test.py`
  - Was it successful?

**Thanks for your attention!**

- Univariate Gaussian
  - Multivariate Gaussian
  - Law of Total Probability
  - Conditioning (Bayes' rule)
- 
- Disclaimer: lots of linear algebra in next few lectures. In fact, pretty much all computations with Gaussians will be reduced to linear algebra!