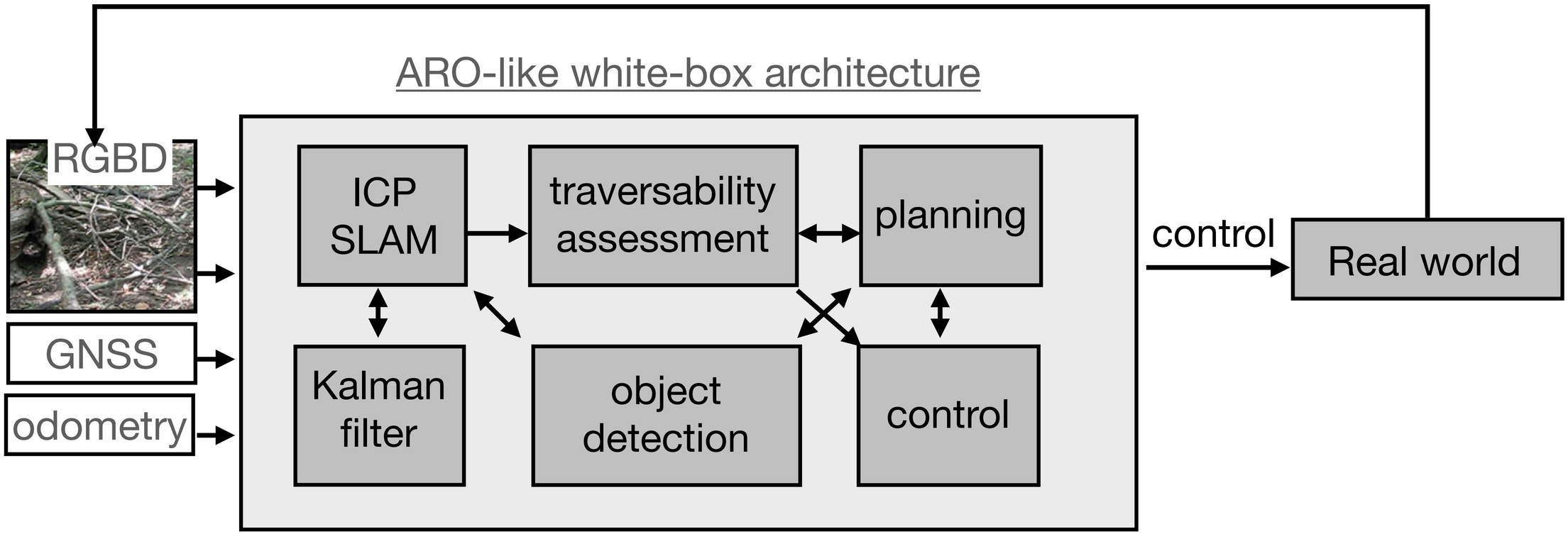
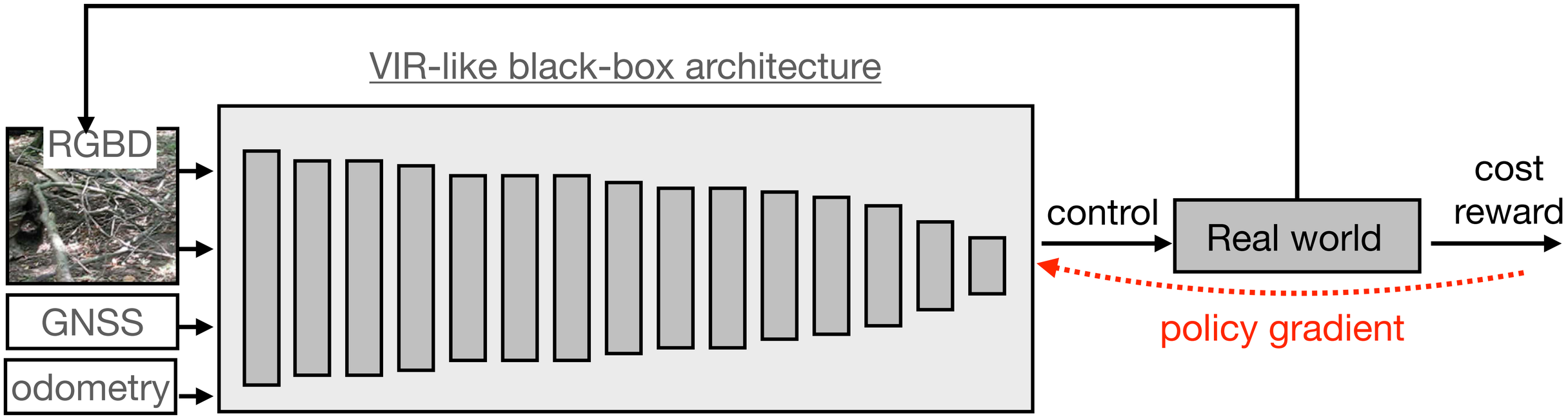
Take home message ARO

Karel Zimmermann

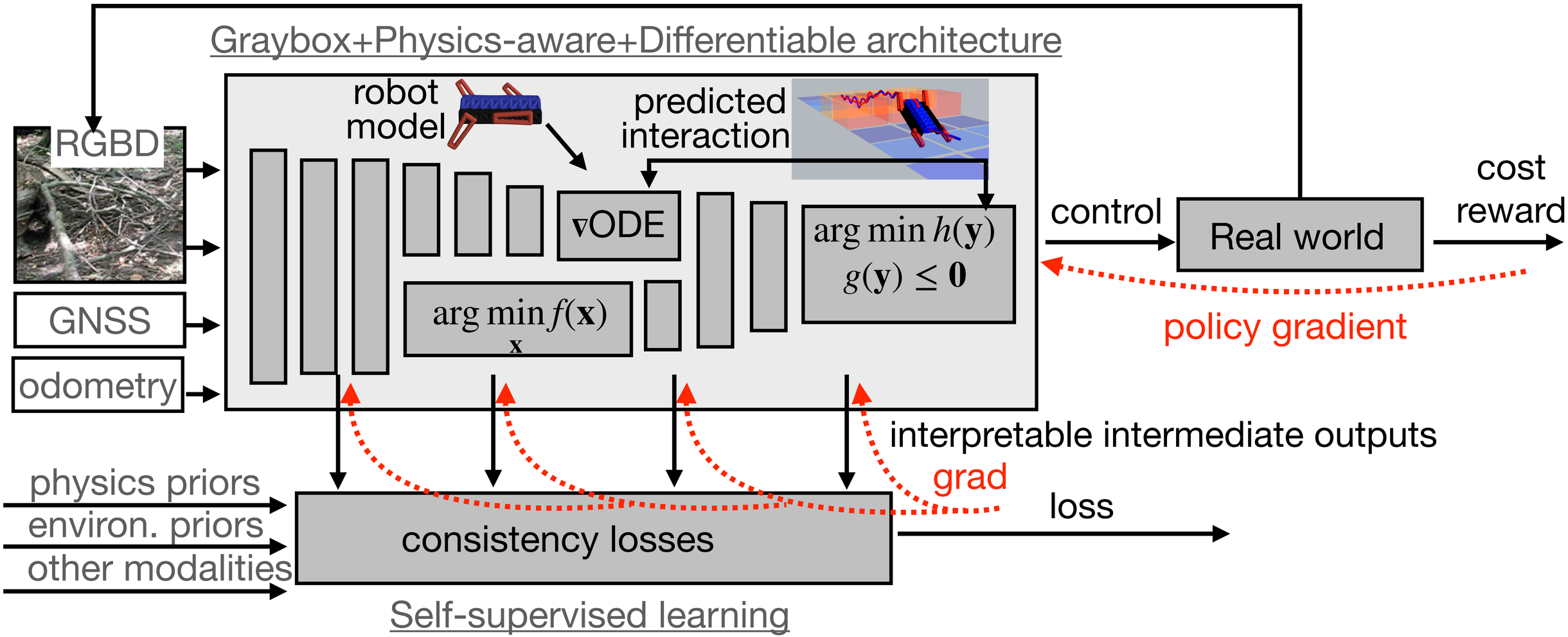


- Each component designed/learned/tuned independently:
  - huge amount of human-annotated data/human-feedbacks
    => tuning on other domains is extremely expensive
  - domain-specific data => transfer impossible



- Allows to learn from cost/reward function via policy gradient, but it suffers from:
  - Sample inefficiency => Often trained in simulation => Simulation bias
  - Black-box architecture => Poor generalization beyond training environment
  - Physical embodiment ignored => Impossibility to transfer on other robots
  - Hard to construct dense self-evaluating cost/reward function

Longterm goal: Explainable + self-learnable architecture

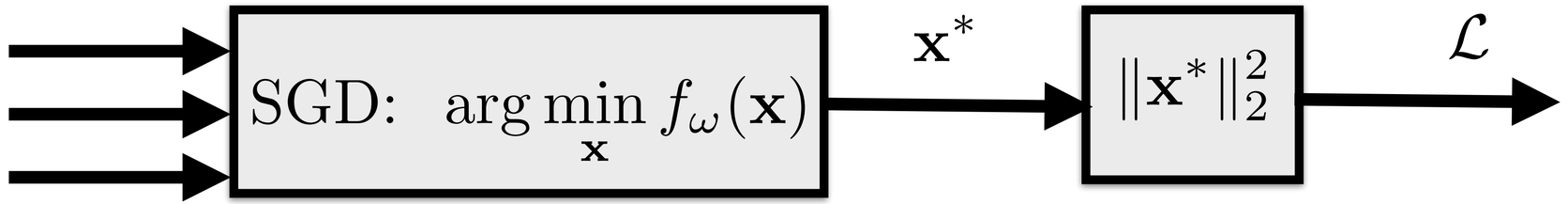


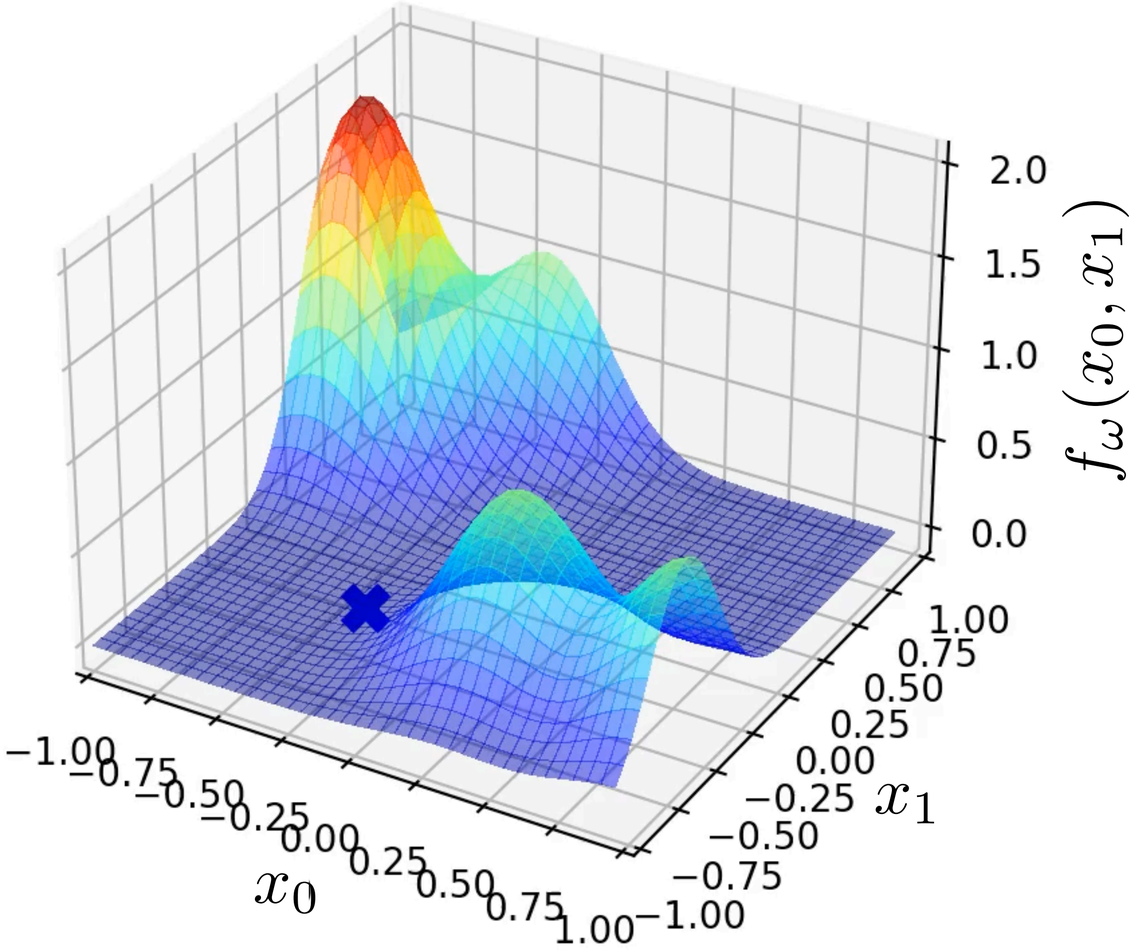
- Self-learning through “end-to-end differentiability” + “explainability” + “priors”
- Interesting issues:
  - backpropagation through non-trivial components
  - construction of consistency losses

Backpropagation through optimization problem

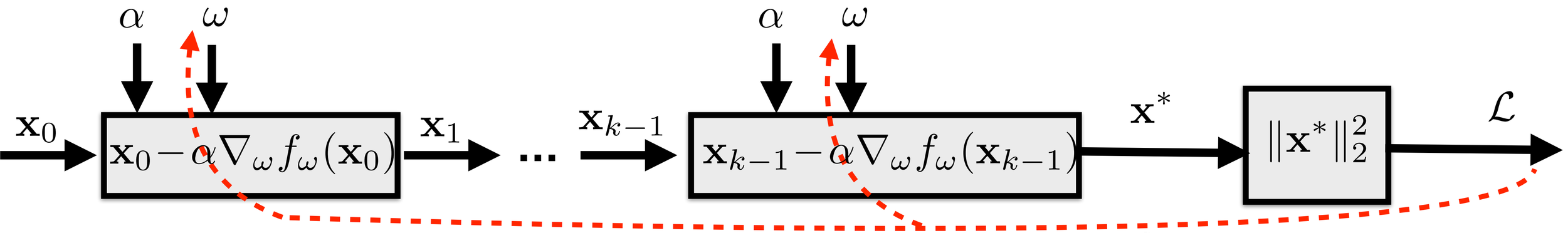
$$\omega^* = \arg\min_{\omega} \|\text{SGD:} \quad \arg\min_{\mathbf{x}} f_\omega(\mathbf{x})\|^2$$

$\mathbf{x}_0$… initial point
ω… criterion params
α, β… optimizer params



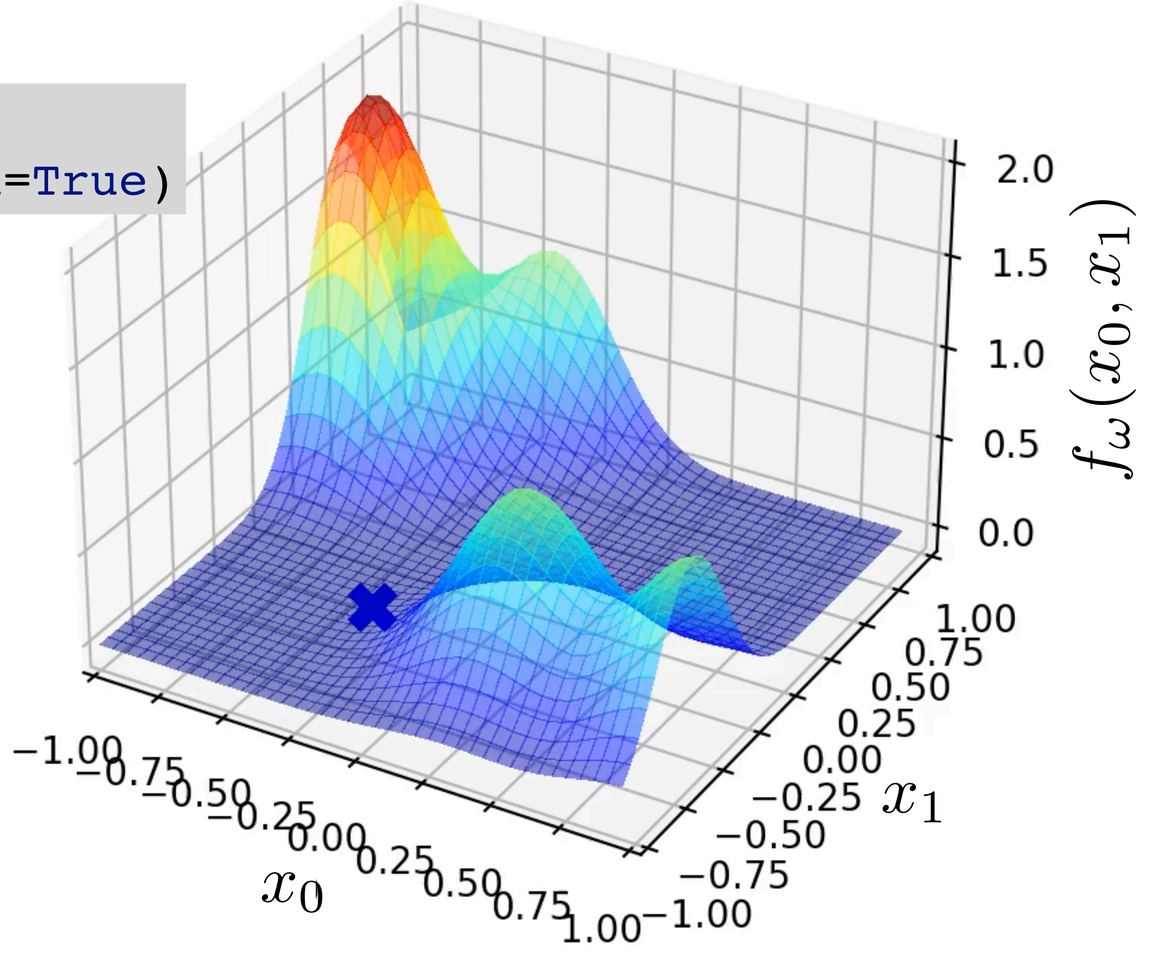


Backpropagation through optimization problem



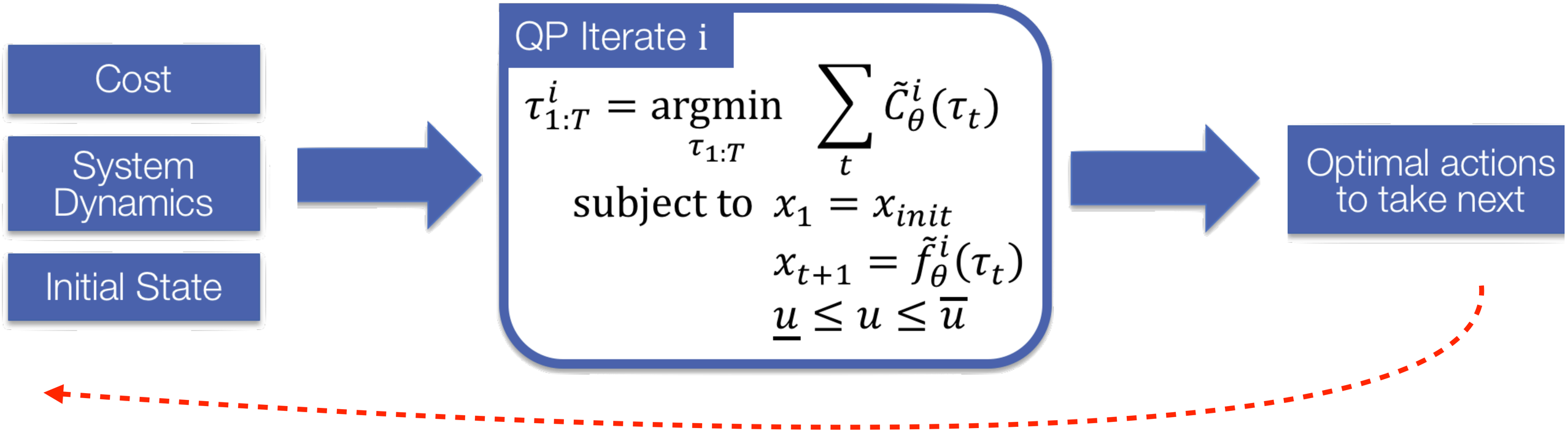
PyTorch:

```
for i in range(iter):
  x = x - torch.autograd.grad(f, x, retain_graph=True)
loss = x.norm()
torch.autograd.grad(loss, omega)
```



Differentiable MPC control, [Amos et al. NIPS 2018]

https://arxiv.org/abs/1810.13400

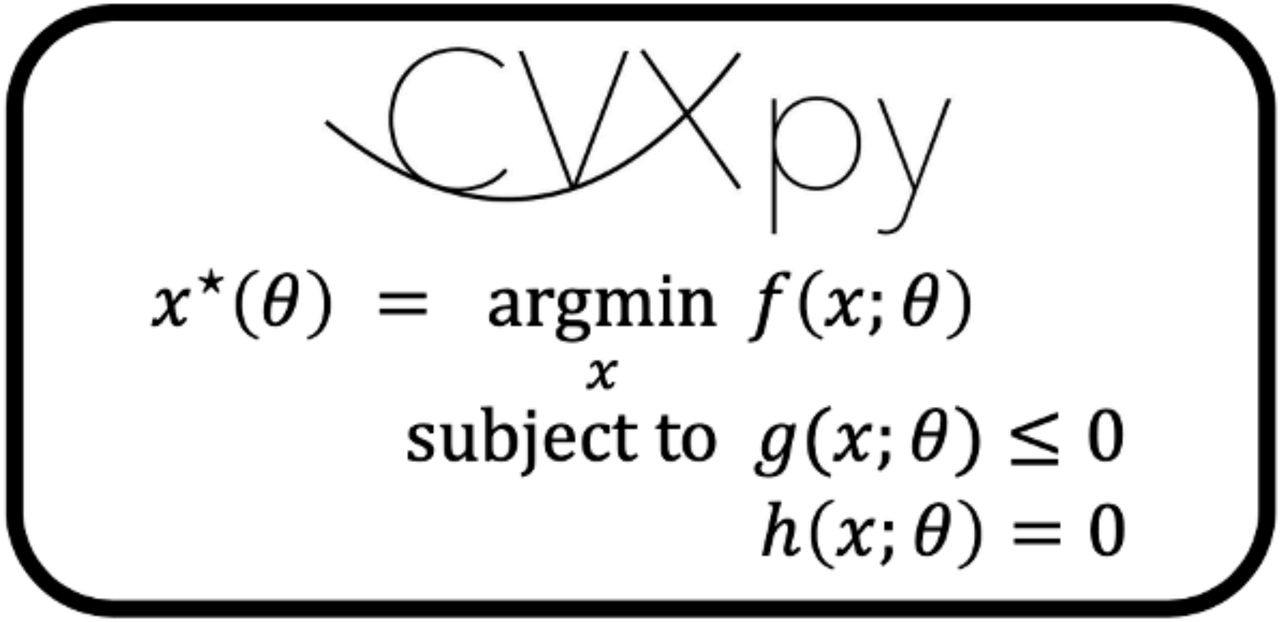


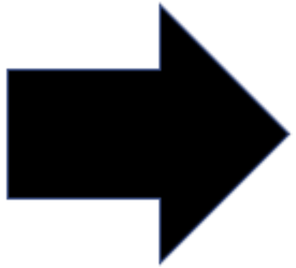
PyTorch embedded modeling language for convex optimization problems

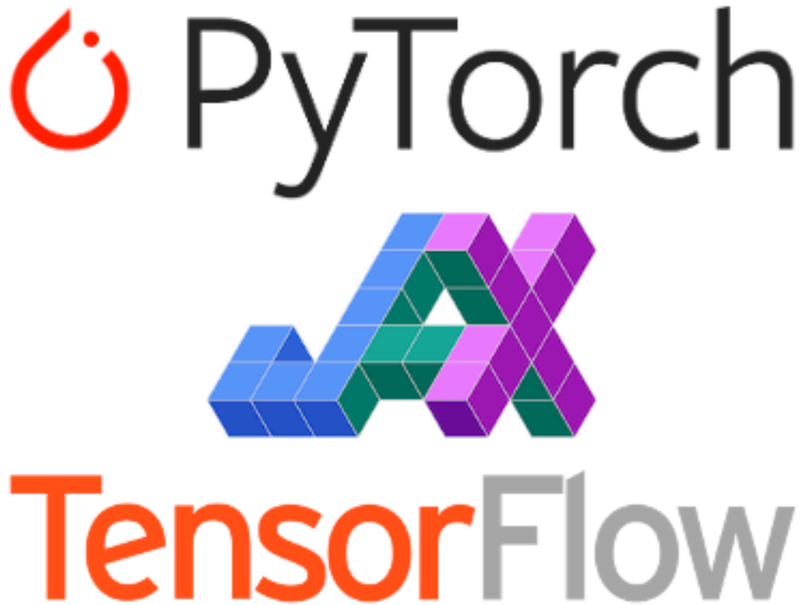
https://www.cvxpy.org/

[Boyd, et al, NIPS, 2019]

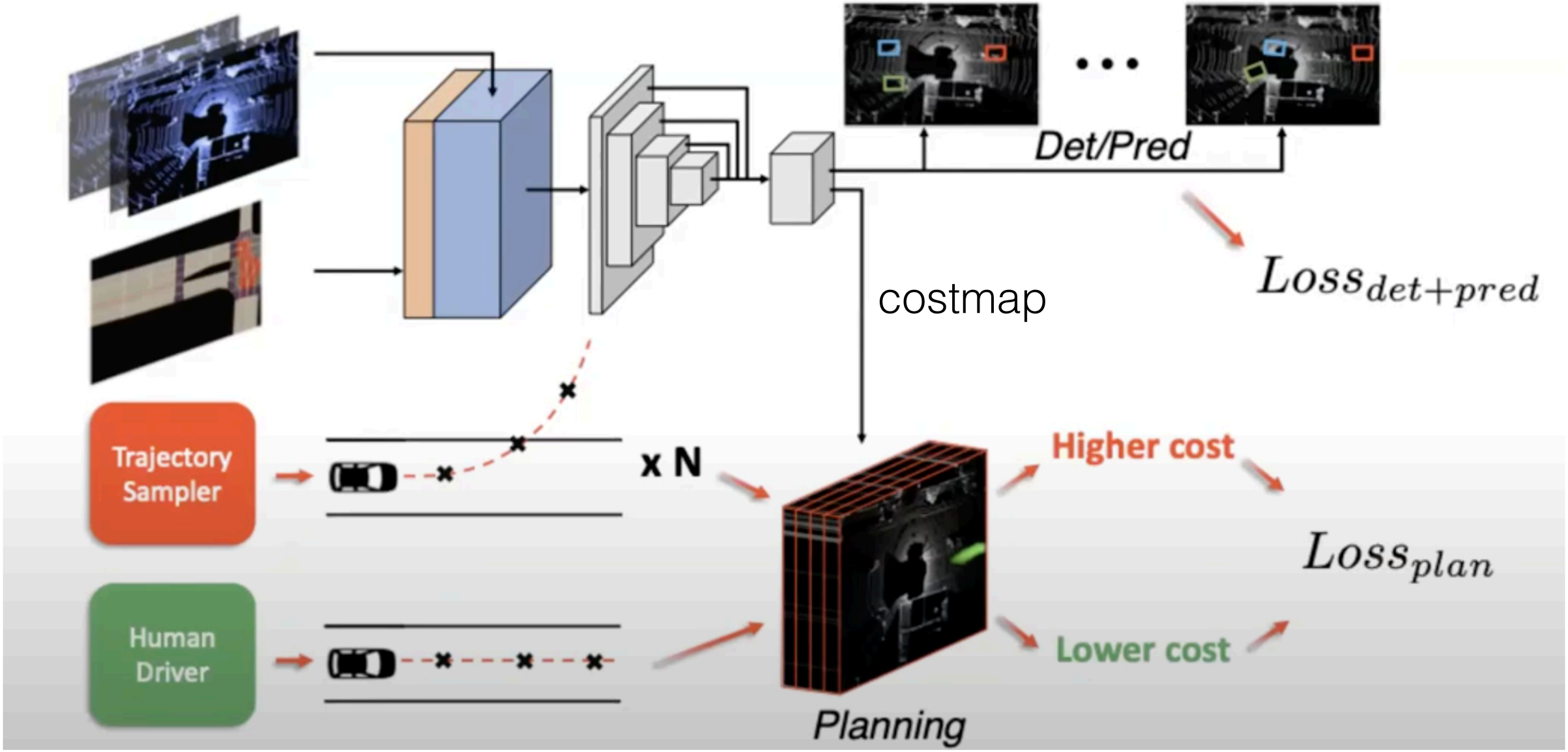
```
cvxpylayer = CvxpyLayer(problem, parameters=[A, b], variables=[x])
solution, = cvxpylayer(A, b) # feed-forward pass (solve the problem)
solution.sum().backward()    # backward pass (how A,b influnce solution)
```







Interpretable motion planning
[Zeng,.. Urtasun, Uber, CVPR, 2019]
http://www.cs.toronto.edu/~wenjie/



Grad SLAM [Murthy, ICRA, 2021]

https://gradslam.github.io/

SLAM

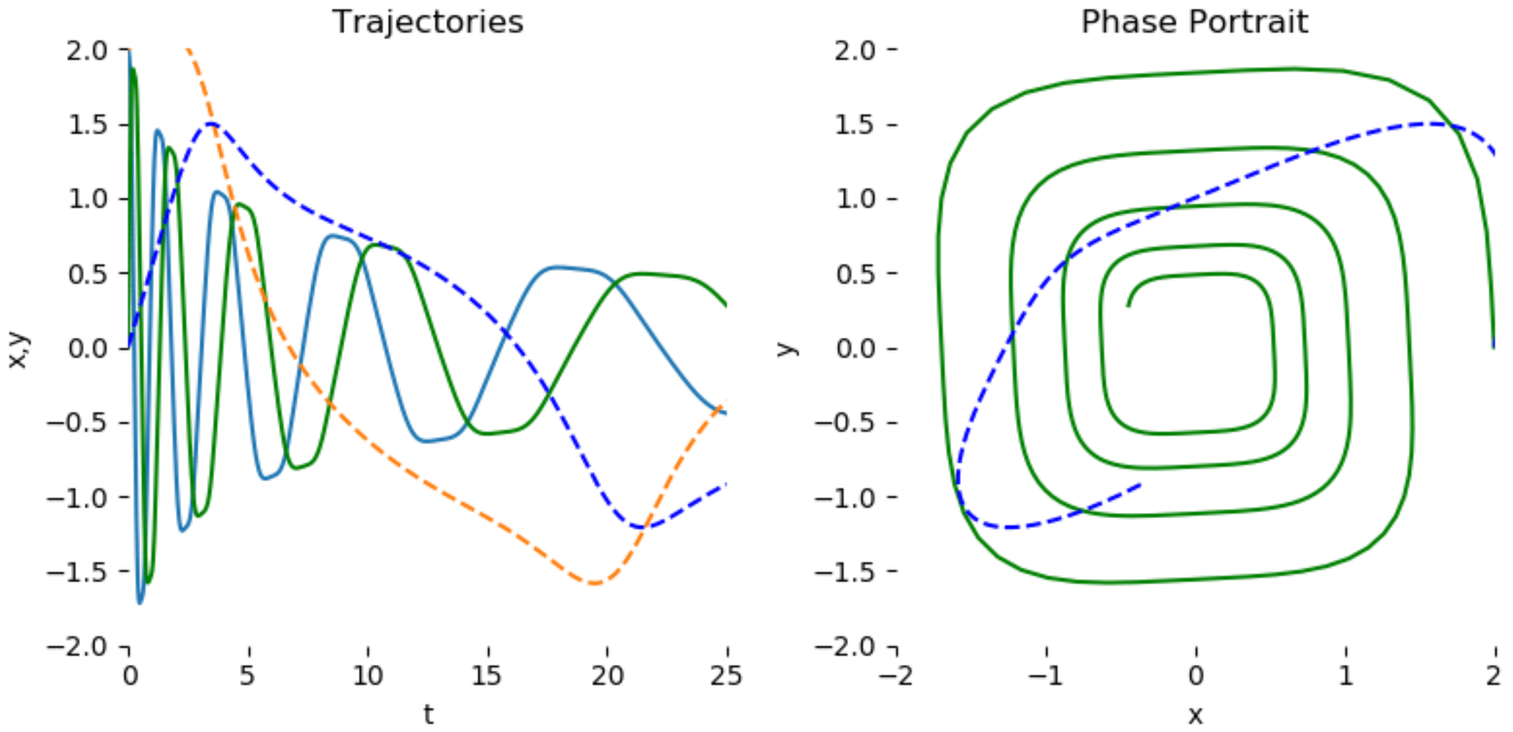
map:

pose: Φ

$$\frac{\partial \mathrm{SLAM}(\mathbf{x}_1, \ldots, \mathbf{x}_t)}{\partial \mathbf{x_1}, \ldots, \mathbf{x_t}} = ?$$

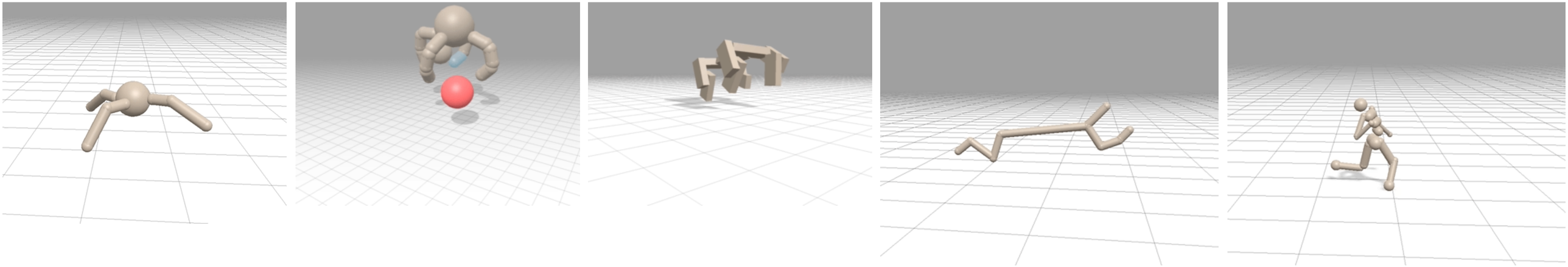
Pytorch differentiable ODE solver

https://github.com/rtqichen/torchdiffeq



Google’s BRAX/ Nvidia’s WARP - differentiable physics engine

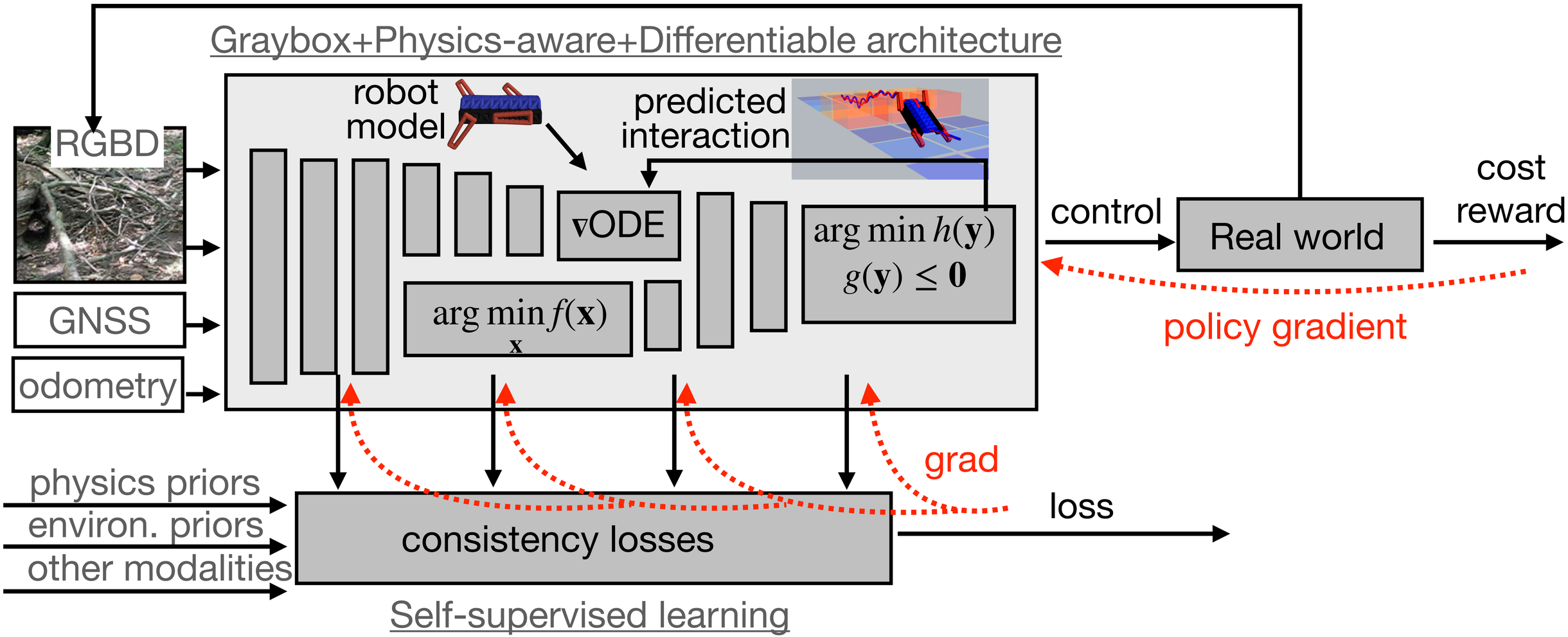
https://github.com/google/brax



Brax simulates these environments at millions of physics steps per second on TPU

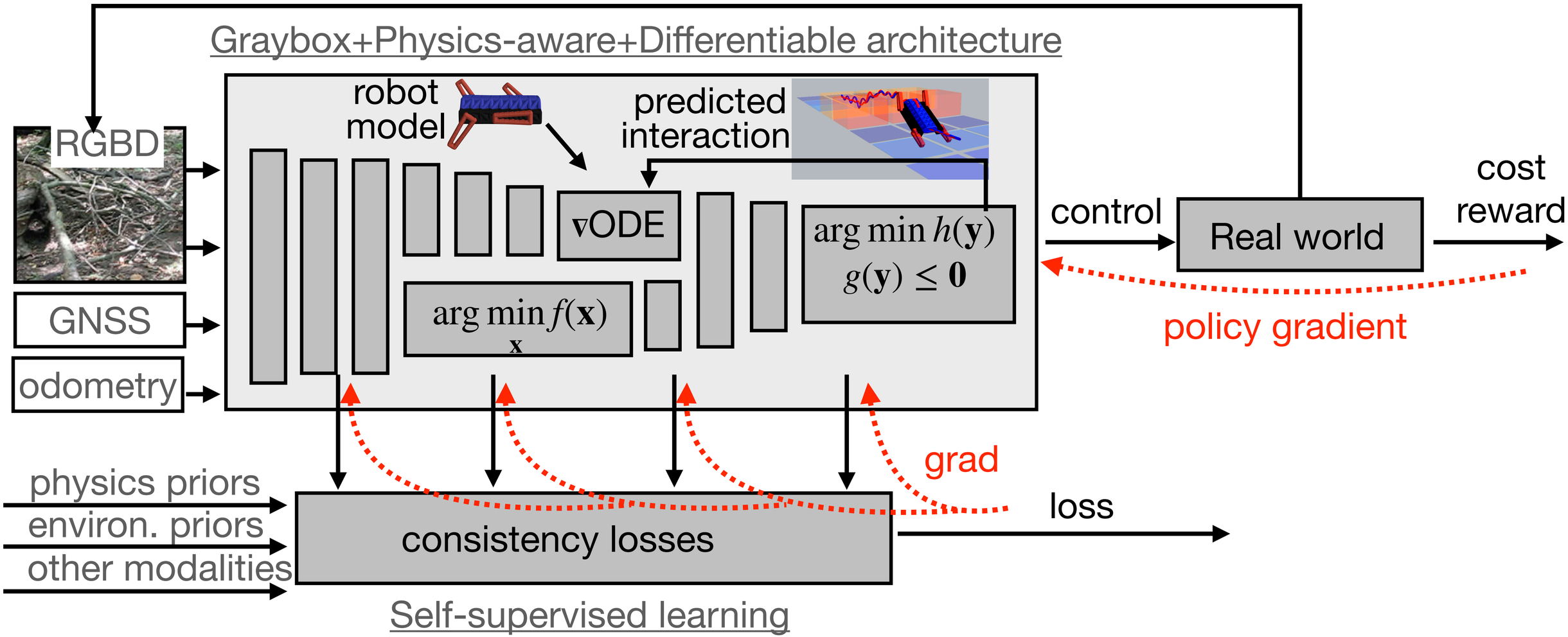
BRAX

Longterm goal: Explainable + self-learnable architecture



- Self-learning through “end-to-end differentiability” + “explainability” + “priors”
- Interesting issues:
  - backpropagation through non-trivial components
  - construction of consistency losses

Longterm goal: Explainable + self-learnable architecture

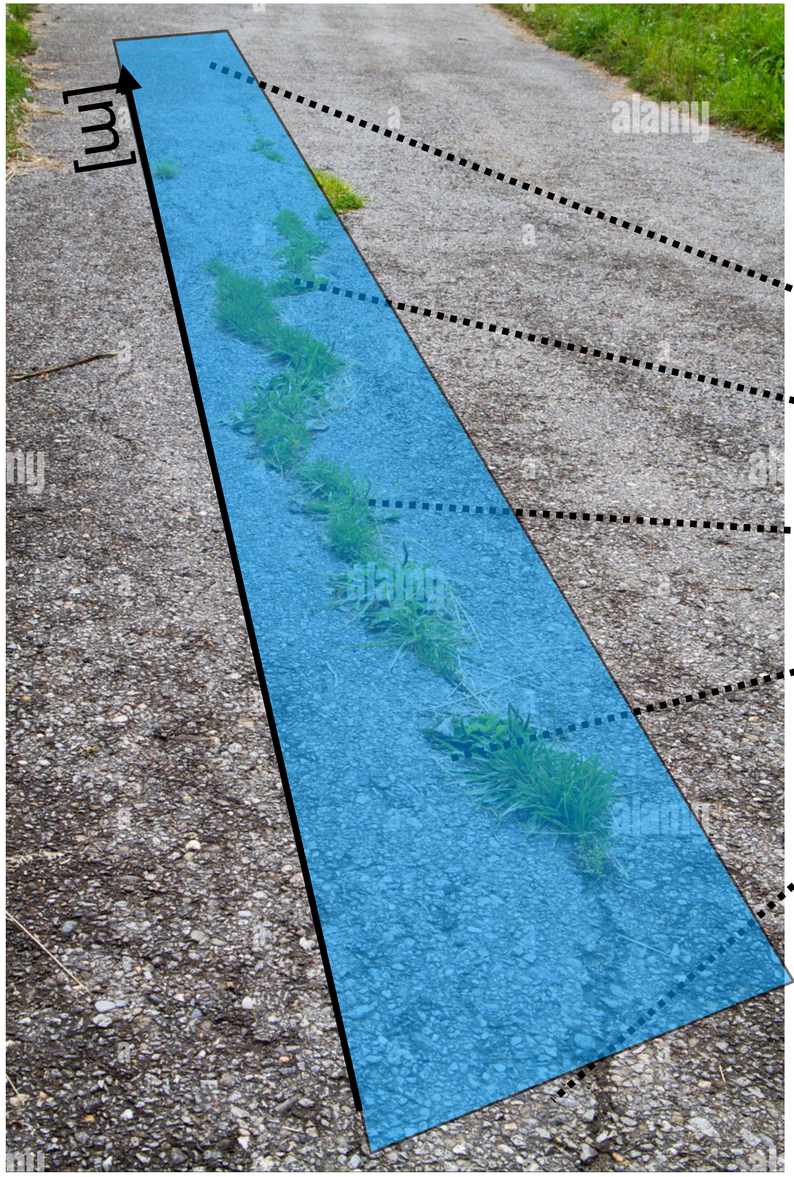


- Self-learning through “end-to-end differentiability” + “explainability” + “priors”
- Interesting issues:
  - backpropagation through non-trivial components
  - construction of consistency losses

Real robots

Traversability assessment: ability to predict robots behaviour on complex terrains perceived by exteroceptive onboard sensors such as camera or lidar.

Learning to predict robot-terrain interaction from RGB(D)
Trivial self-supervision



control

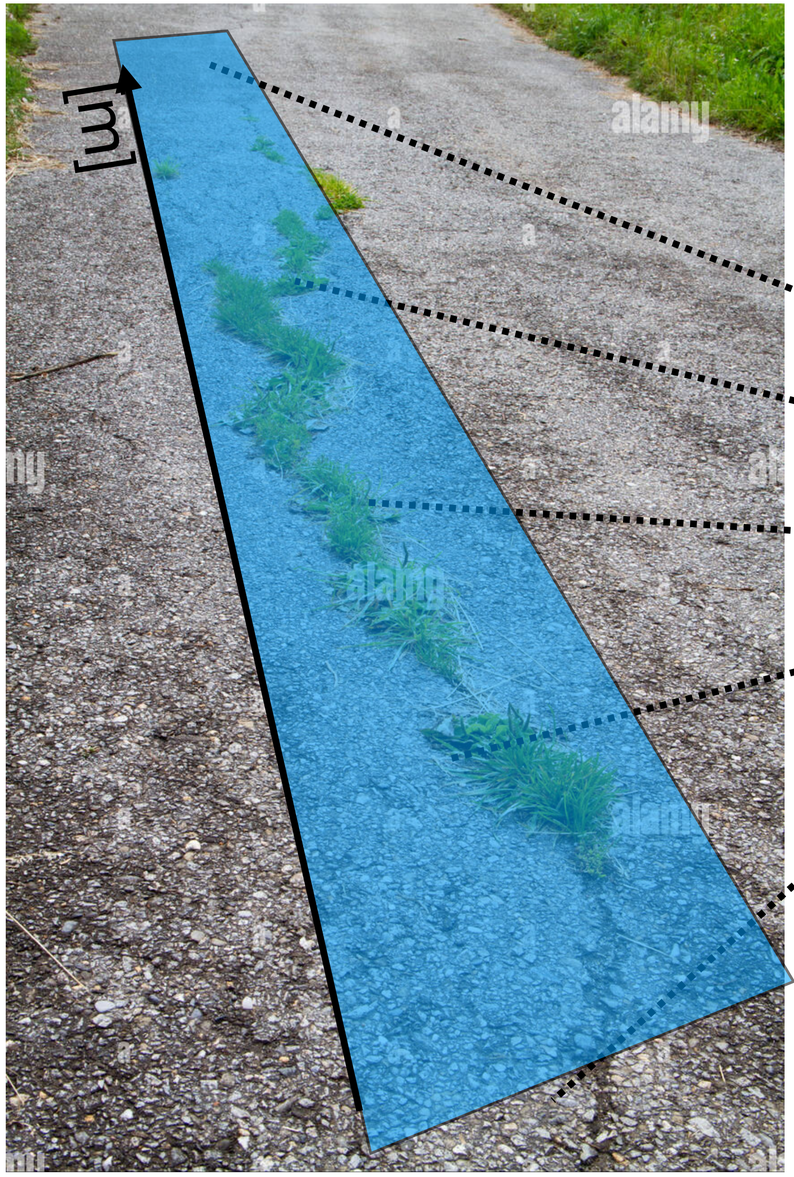
CNN

[m]

$\hat{\mathbf{x}}(\theta)$ . . . Predicted trajectory

trajectory

Learning to predict robot-terrain interaction from RGB(D)
Trivial self-supervision



control

CNN

[m]

trajectory

$\mathbf{x}^{\mathrm{gt}}$ ..... Ground truth trajectory

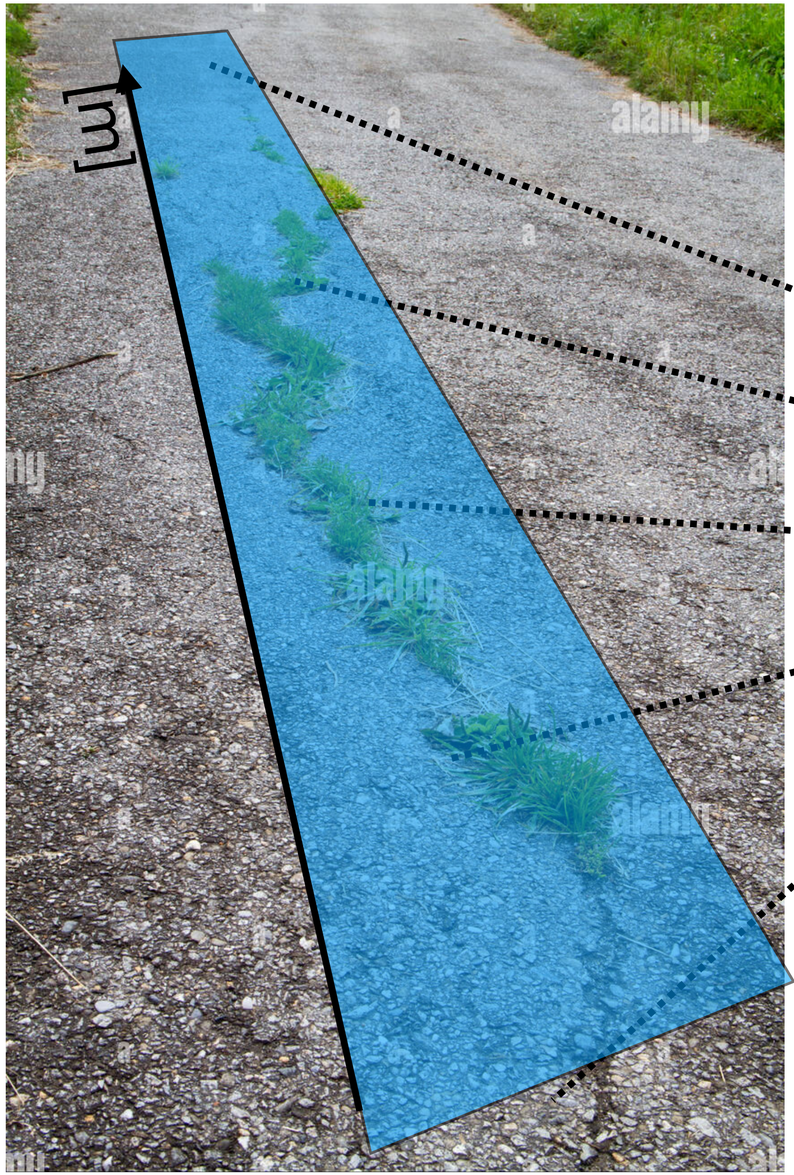
$\hat{\mathbf{x}}(\theta)$ .... Predicted trajectory

$$\mathscr{L}_2(\hat{\mathbf{x}}(\theta), \mathbf{x}^{\mathrm{gt}})$$

Blackbox model:

- Weak generalisation
- Unexplainable
- Untransferable

Learning to predict robot-terrain interaction from RGB(D)



terrain encoder

[m]

Elasticity
Damping
Height, Friction

control

robot

∇physics

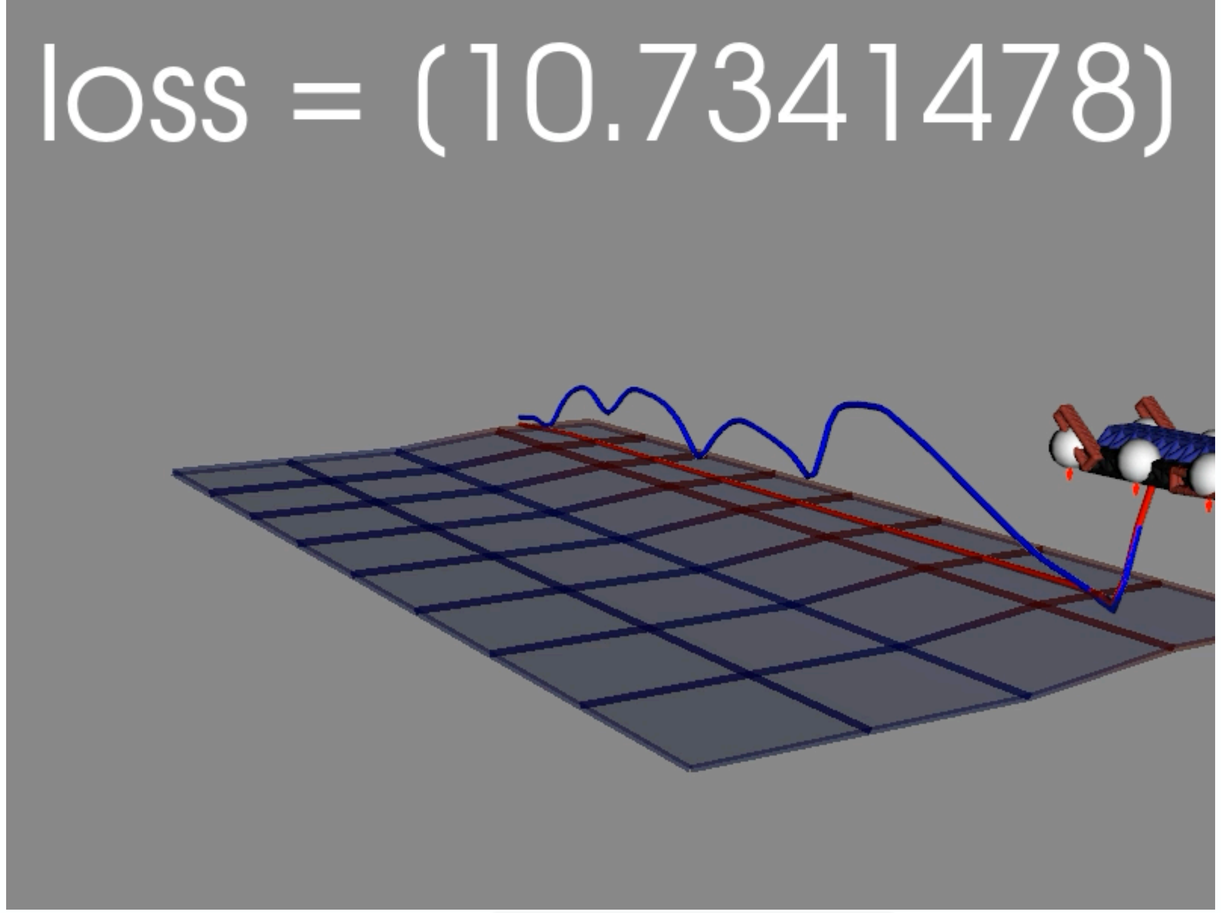
[m]

trajectory

$\mathbf{x}^{\text{gt}}$ ..... Ground truth trajectory

$\hat{\mathbf{x}}(\theta)$ .... Predicted trajectory

Learning to predict terrain properties



real robot trajectory

$\mathbf{x}^\star$

lidar camera

$\mathbf{x}$

convNet $h_\theta(\mathbf{x})$

estimated terrain $\hat{\mathbf{h}}$

heightmap, elasticity, damping, friction, interaction forces

∇physics($\hat{\mathbf{h}}$)

estim. trajectory

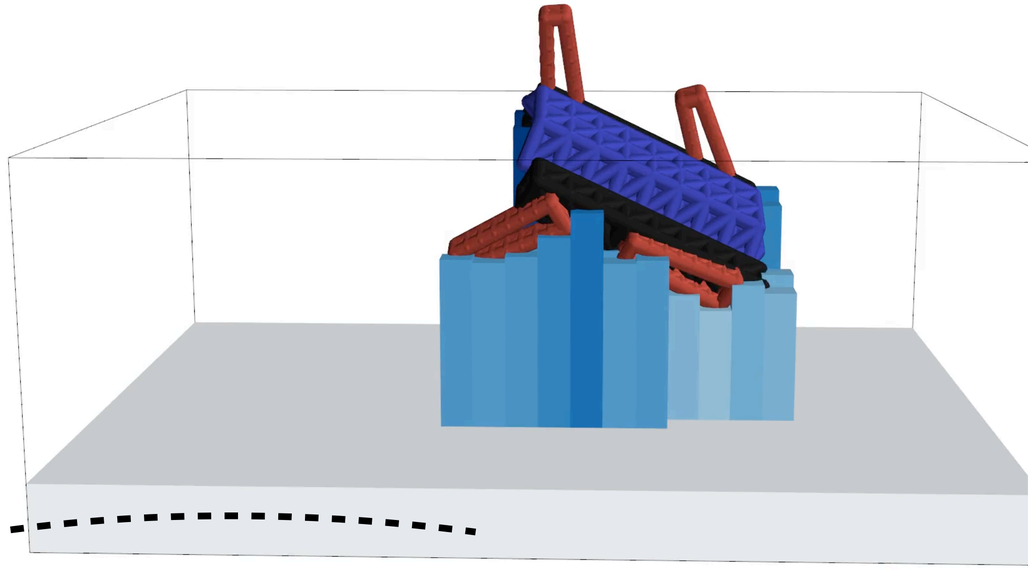
L2-loss

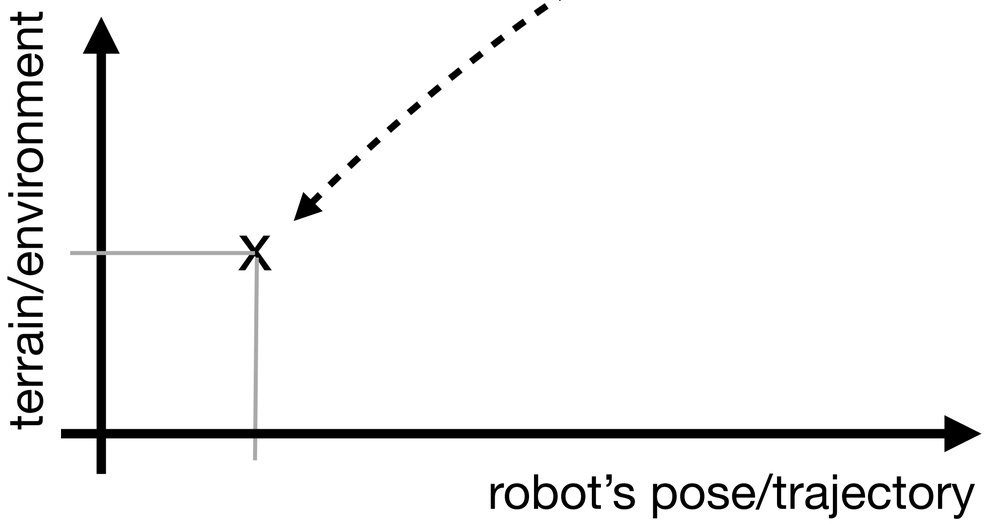
θ

$\frac{\partial h_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \boldsymbol{\theta}}$

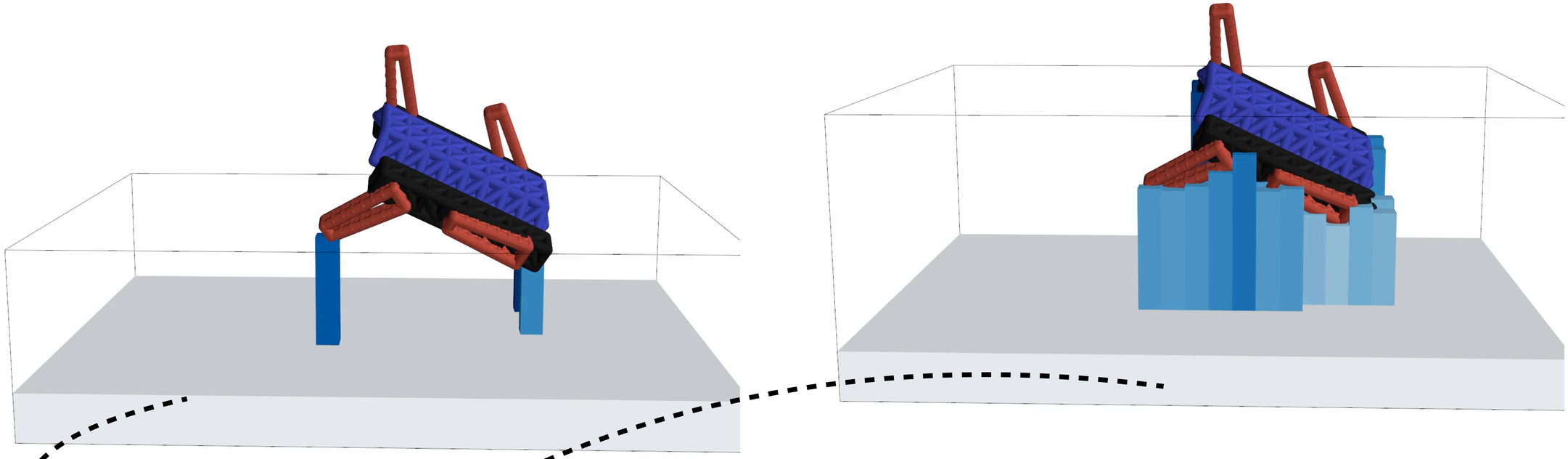
$\frac{\partial \nabla \text{physics}(\hat{\mathbf{h}})}{\partial \hat{\mathbf{h}}}$

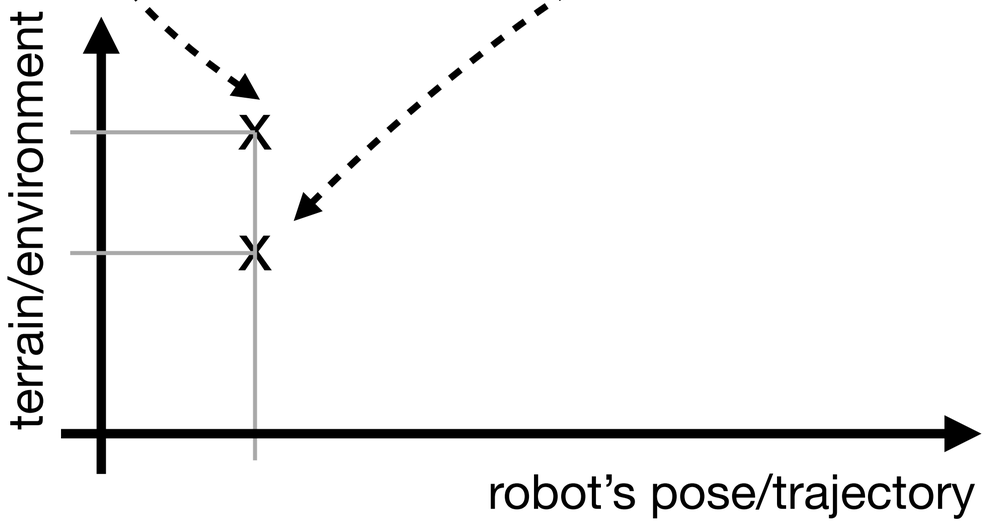
$\frac{\partial \|\mathbf{x} - \mathbf{x}^\star\|}{\partial \mathbf{x}}$



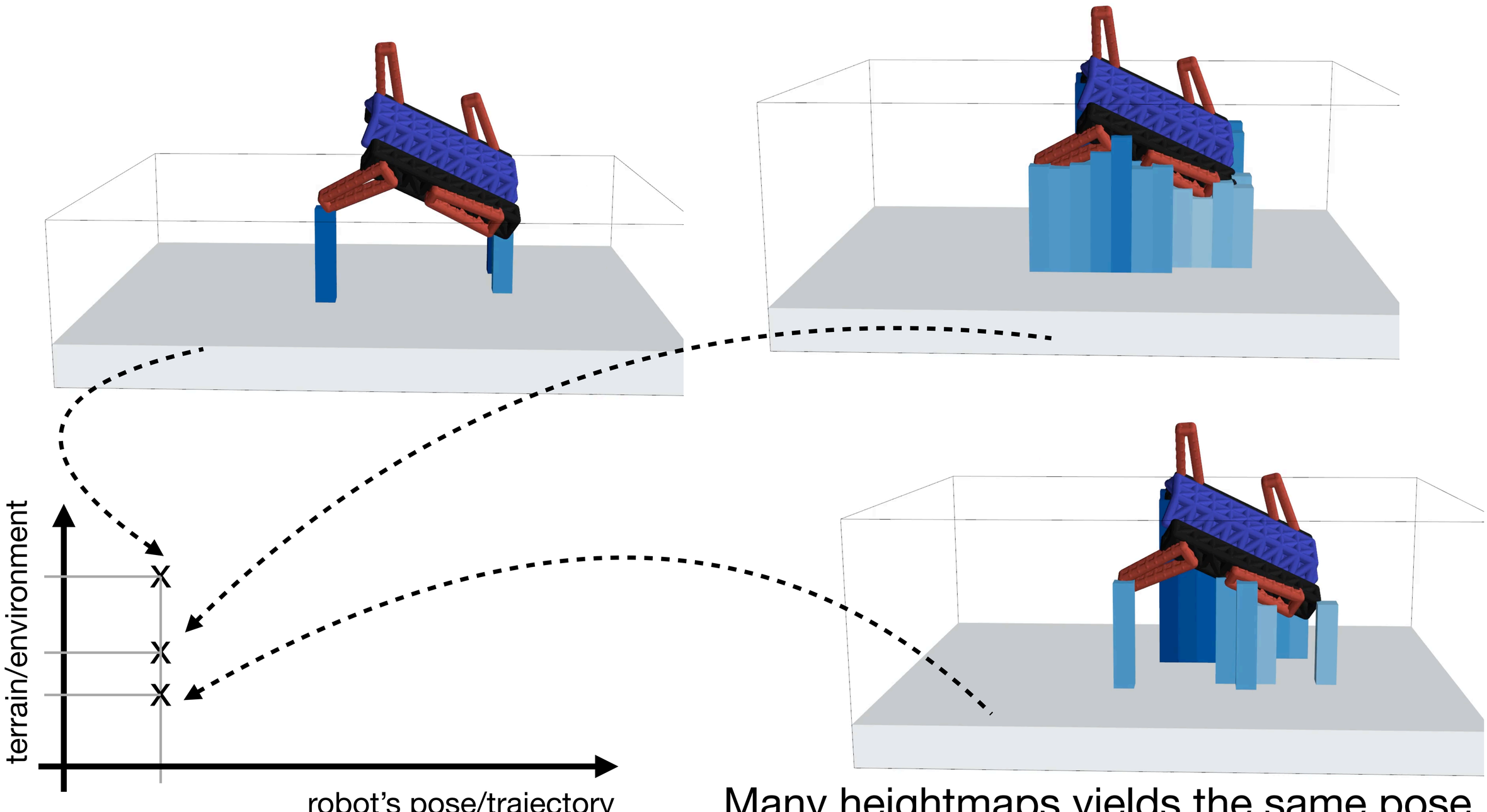


Many heightmaps yields the same pose

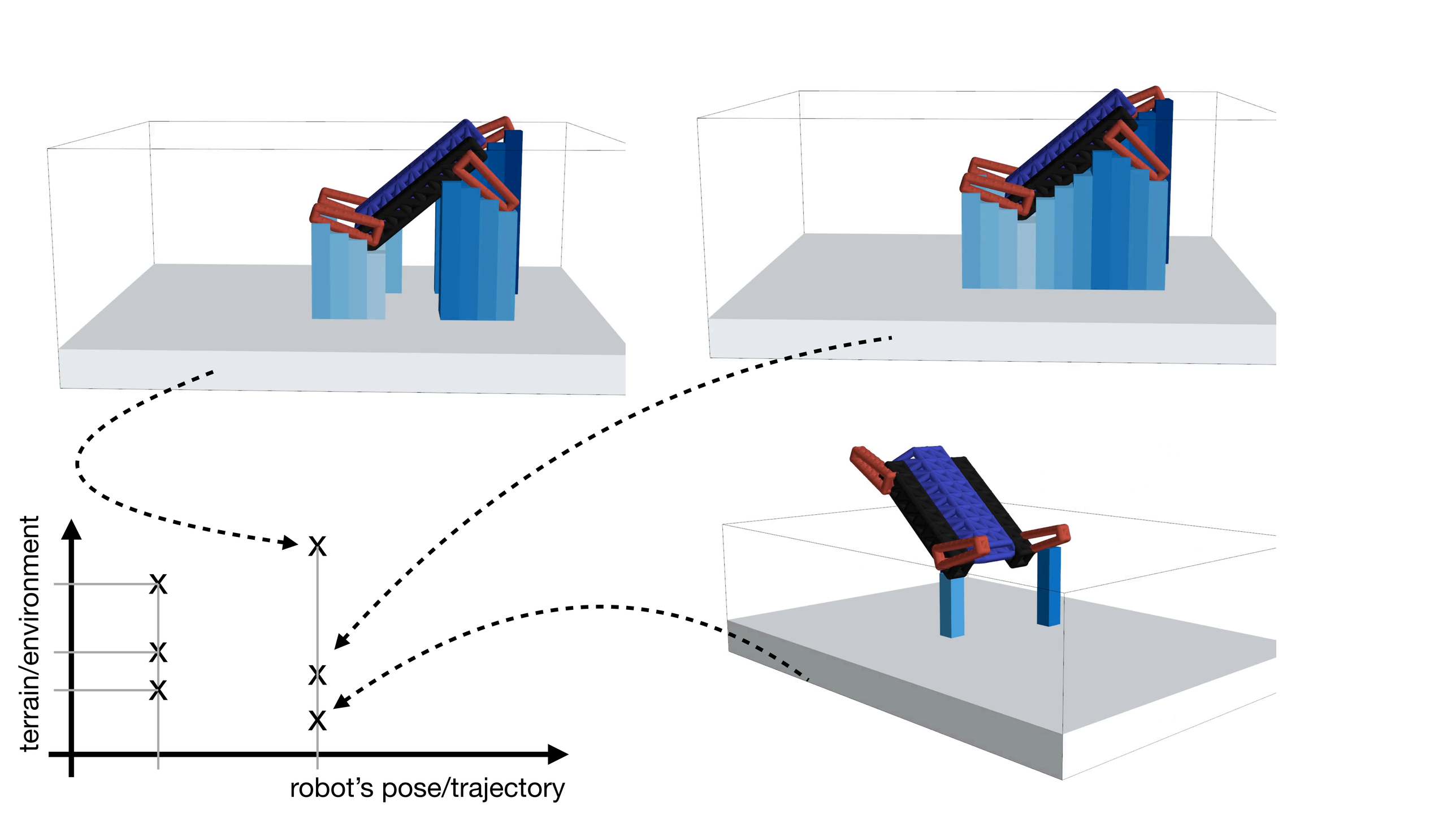


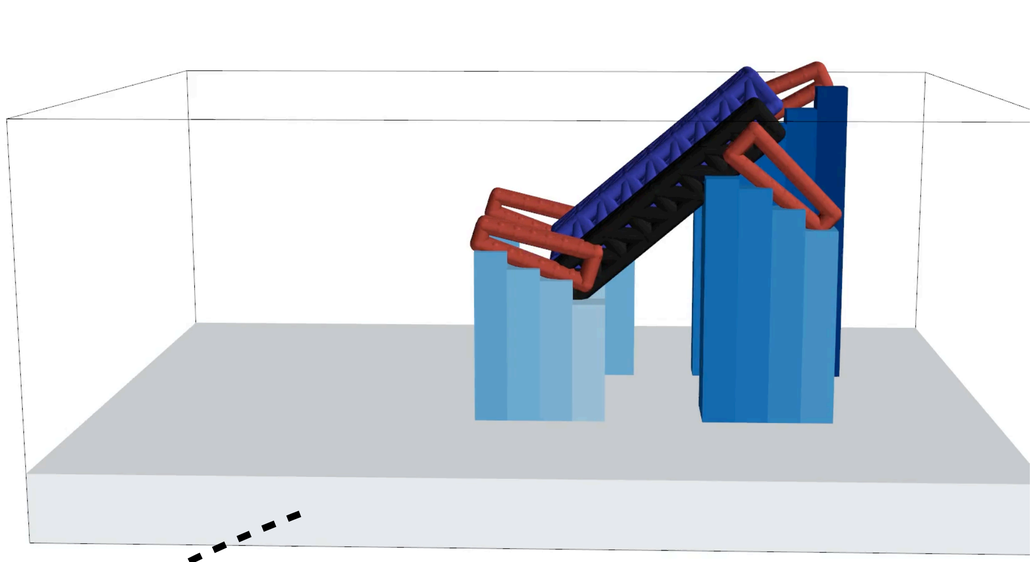


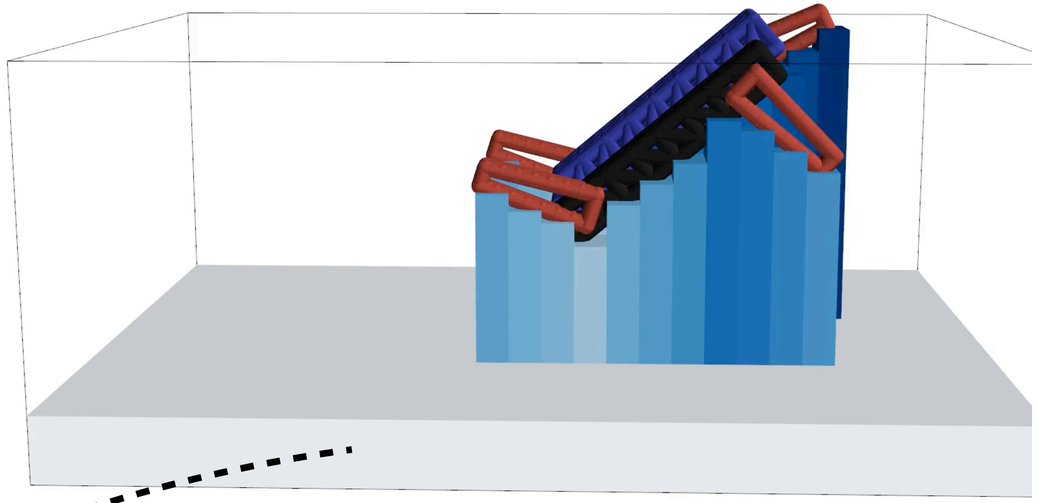
Many heightmaps yields the same pose

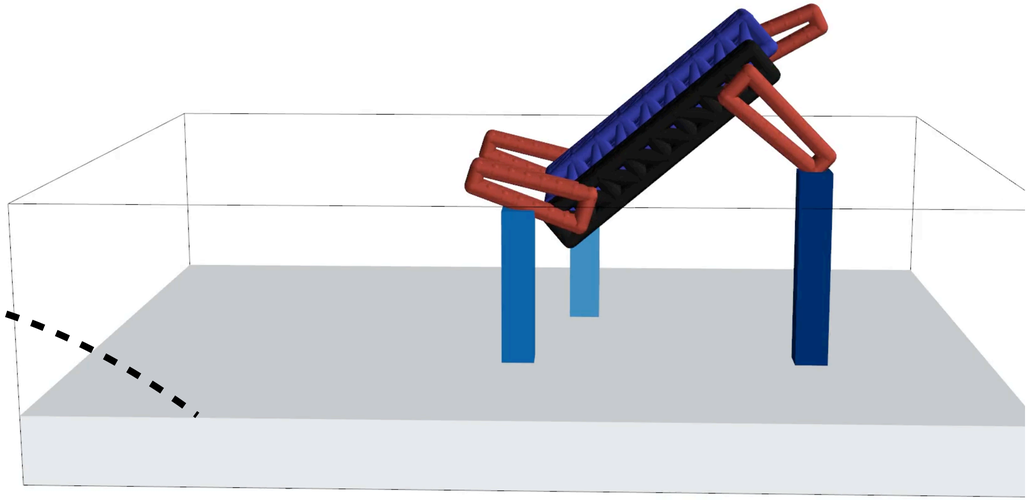


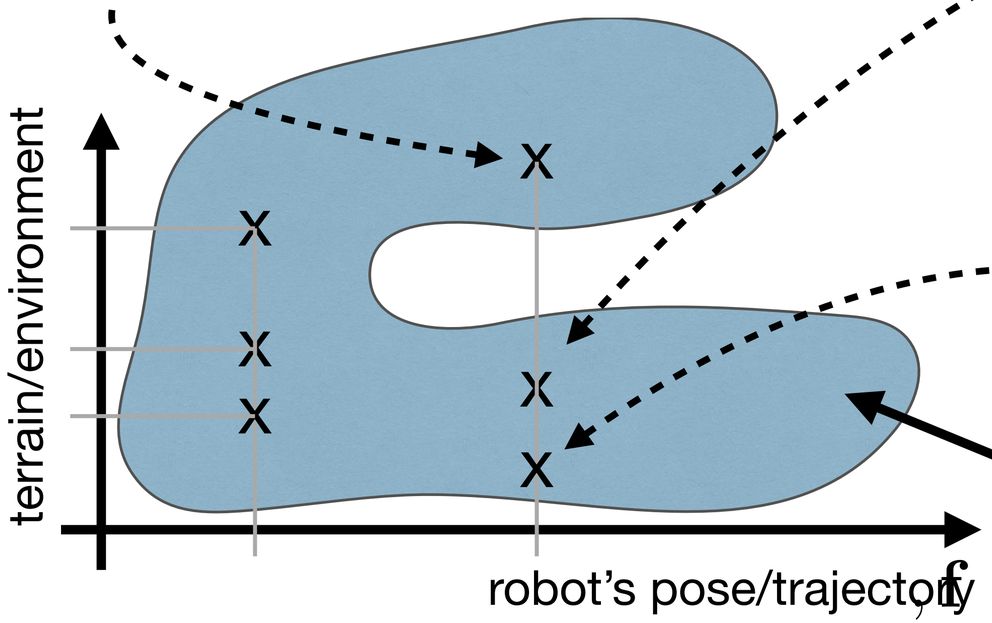
Many heightmaps yields the same pose


terrain/environment
robot's pose/trajectory



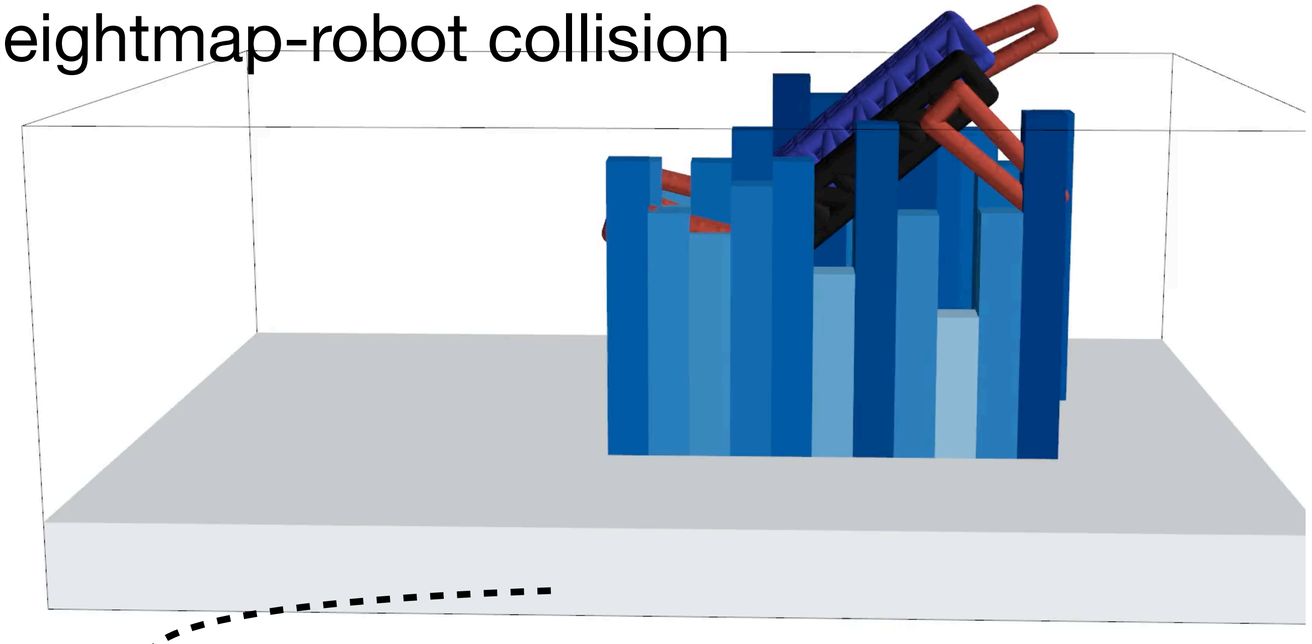


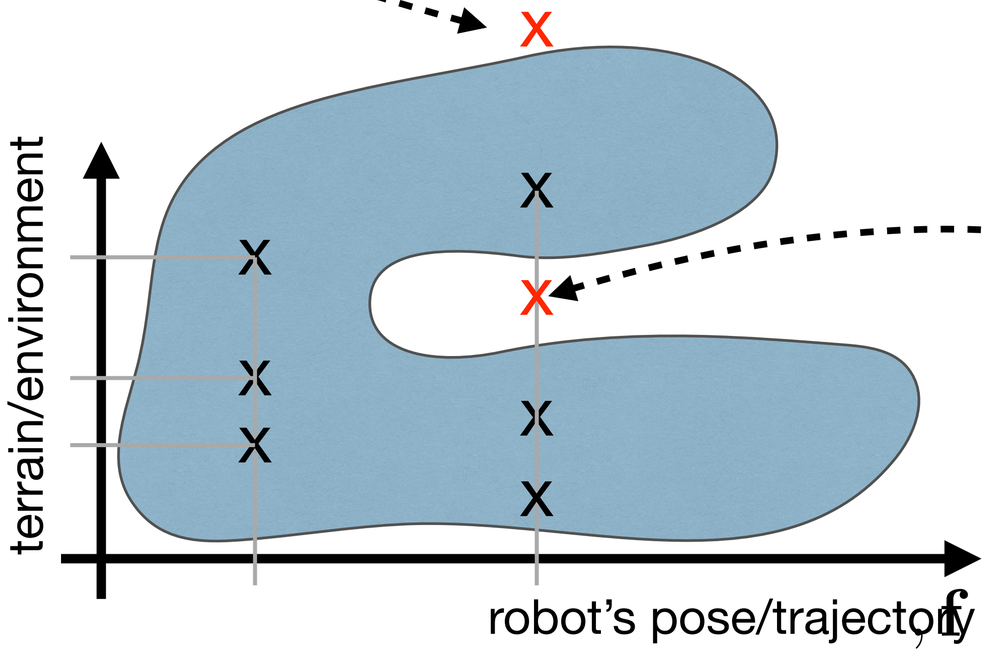




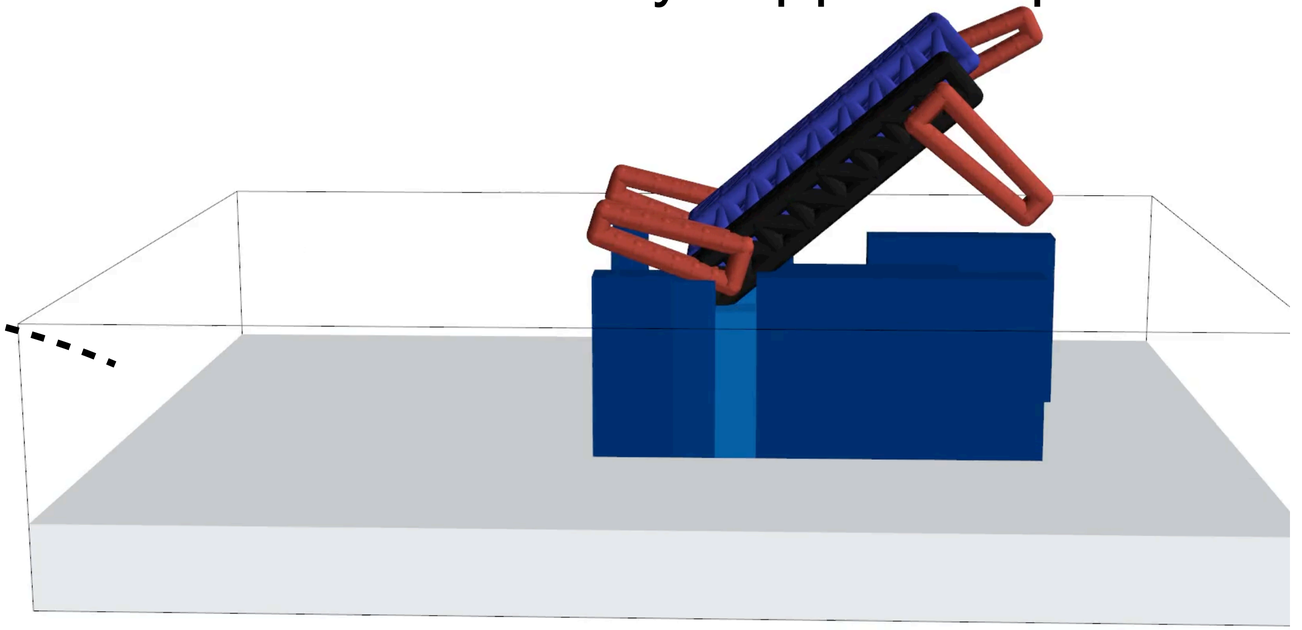
Tuples produce the physically consistent set

heightmap-robot collision



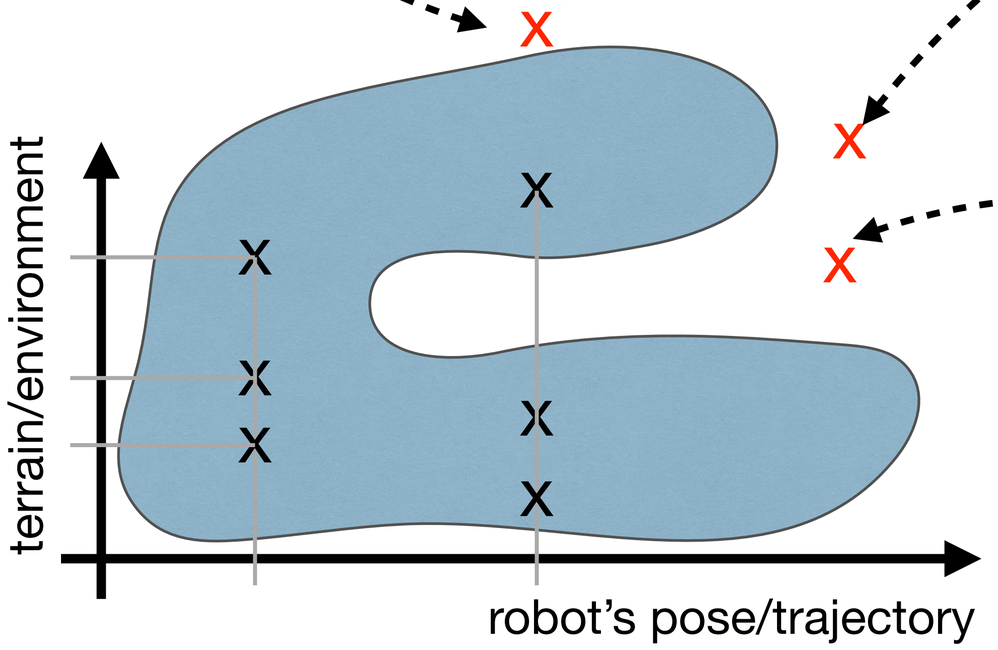


insufficiently supported pose

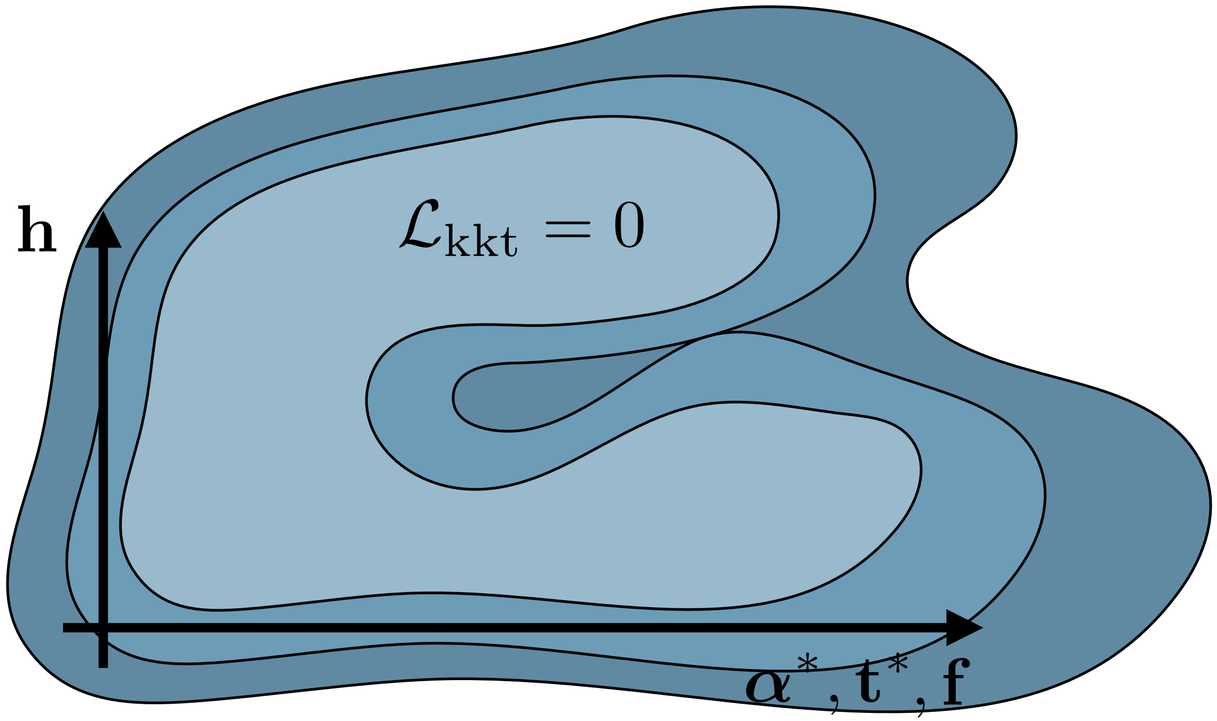


The remaing tuples are physically inconsistent

real world physical inconsistencies

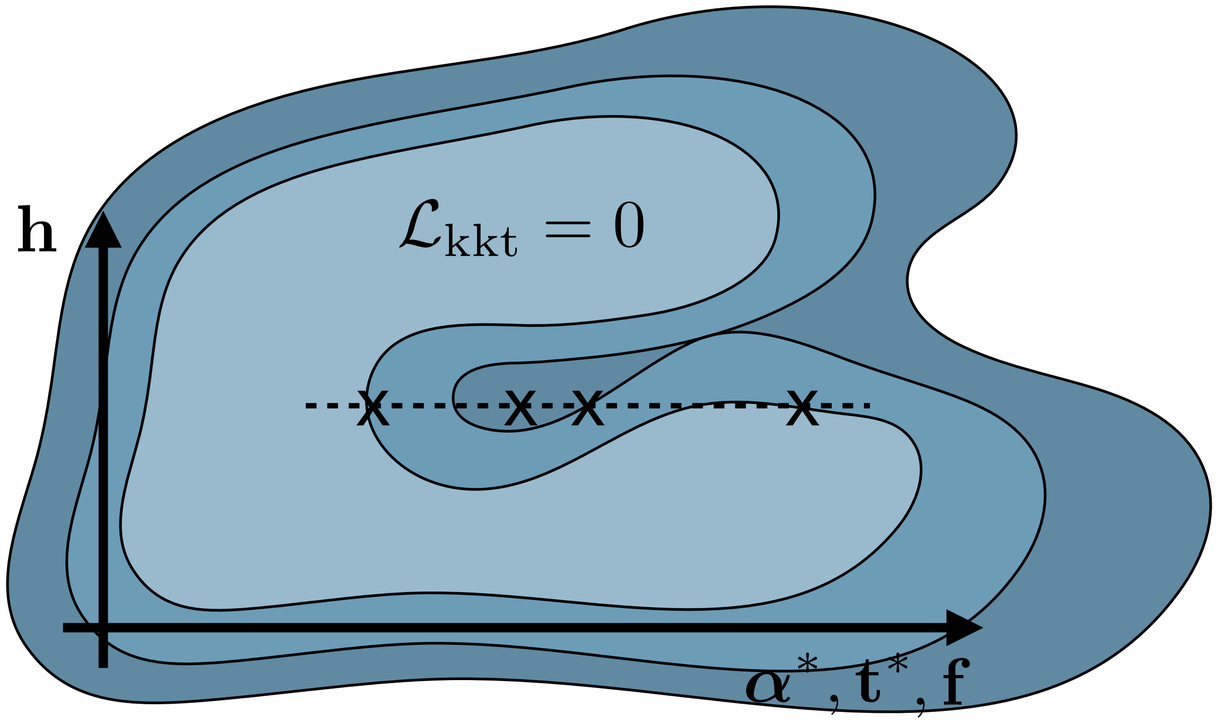


Construct differentiable loss that measures physical trajectory/terrain-consistency



Construct differentiable loss that measures physical trajectory/terrain-consistency

Given the ground truth trajectory one can optimize terrain

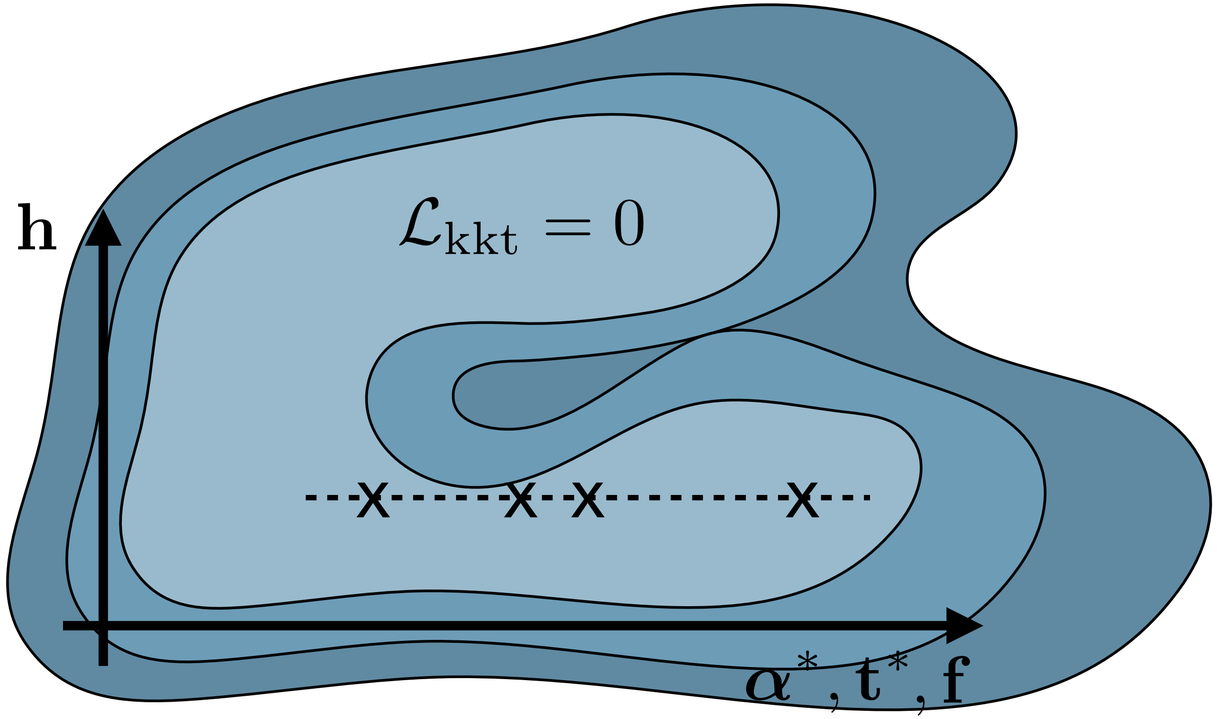


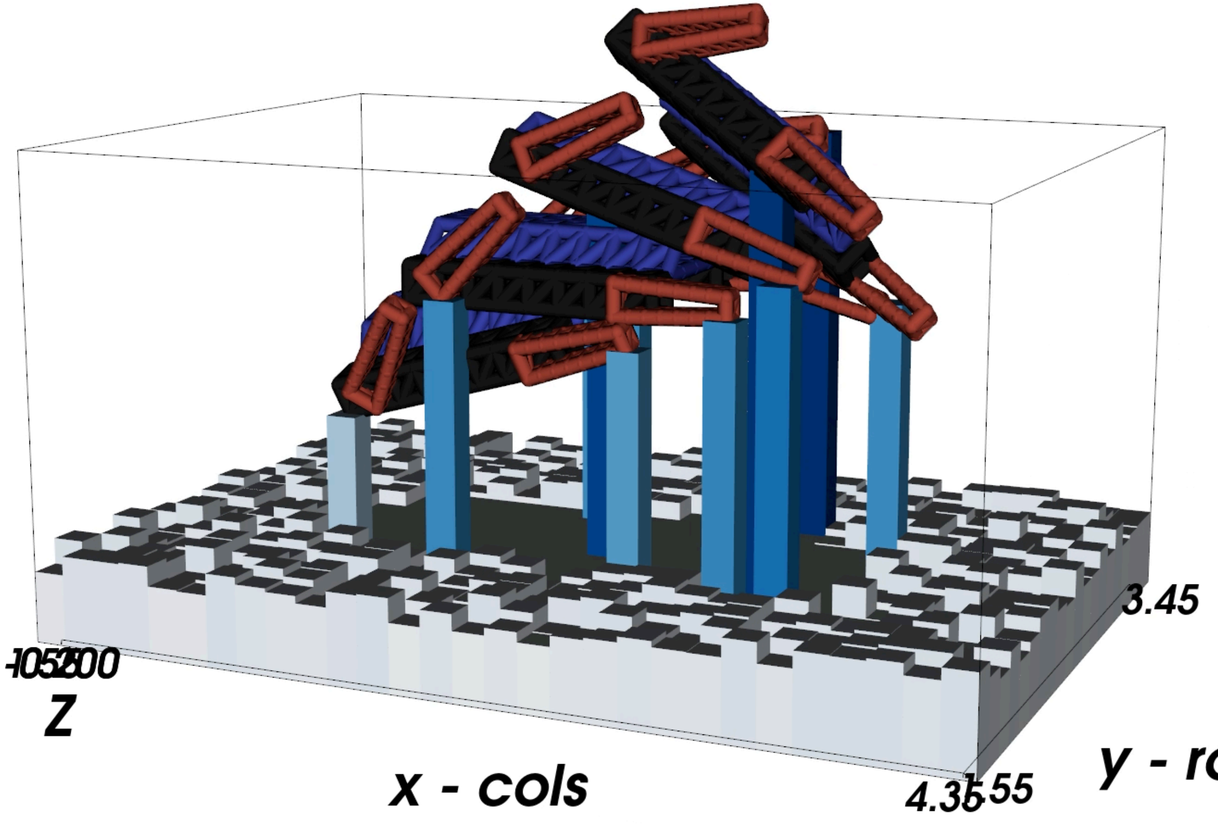
Construct differentiable loss that measures physical trajectory/terrain-consistency

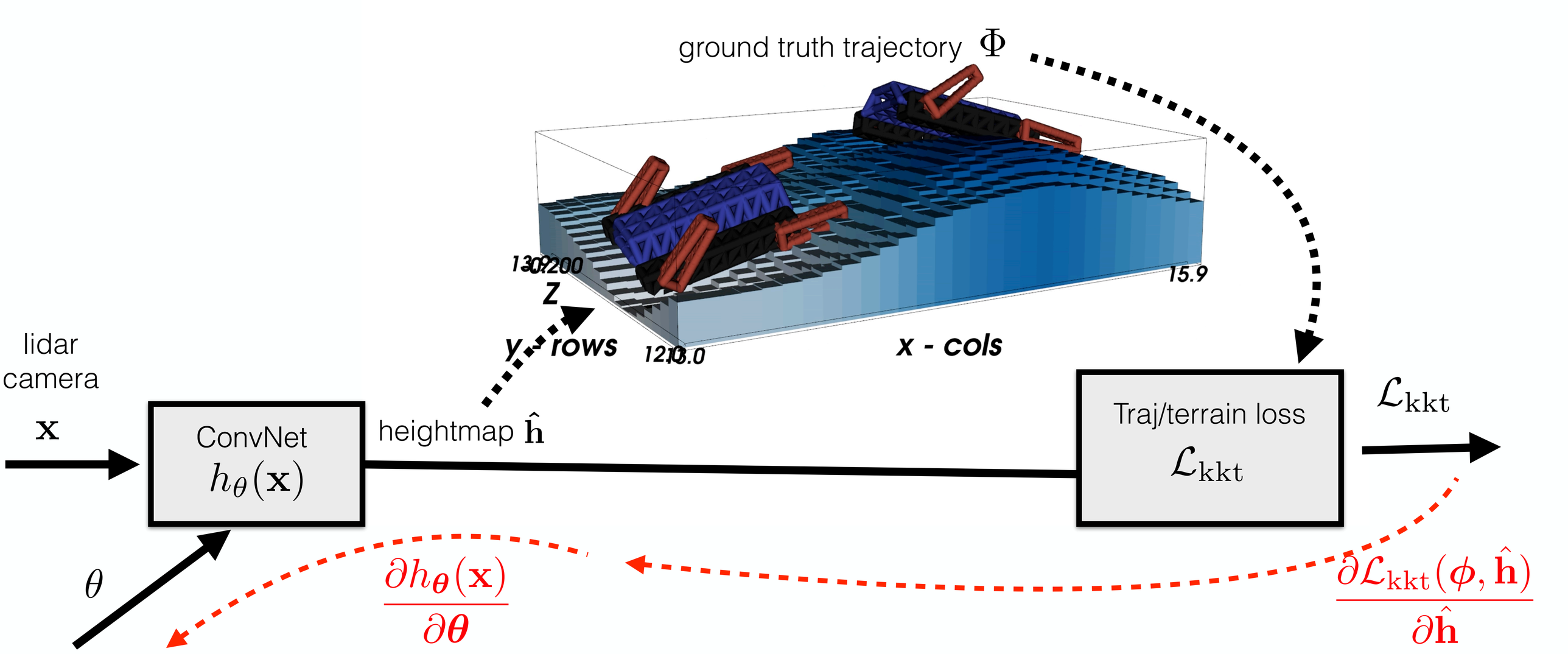
Given the ground truth trajectory one can optimize terrain

Obviously huge sub-space of physically consistent terrains

Regularization needed

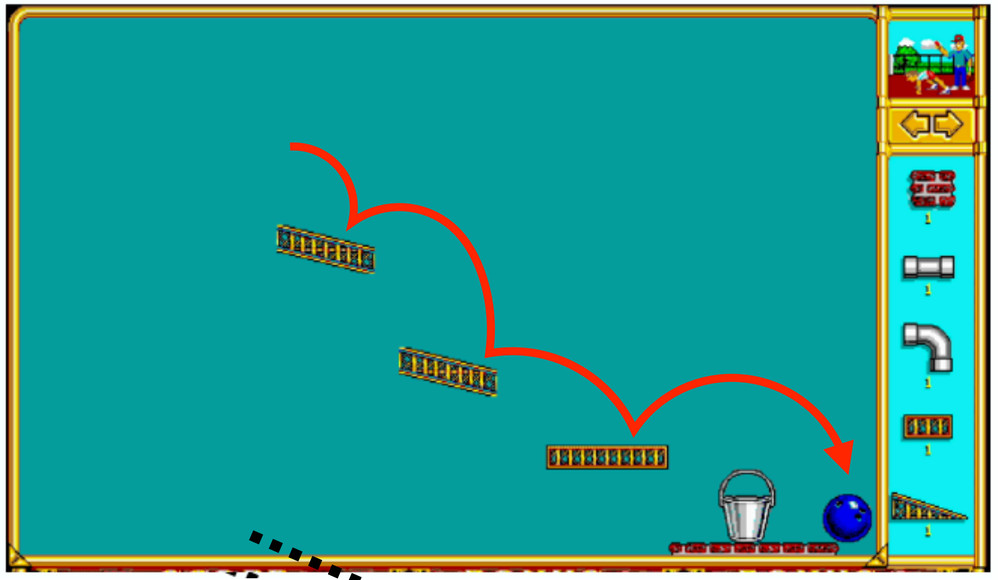


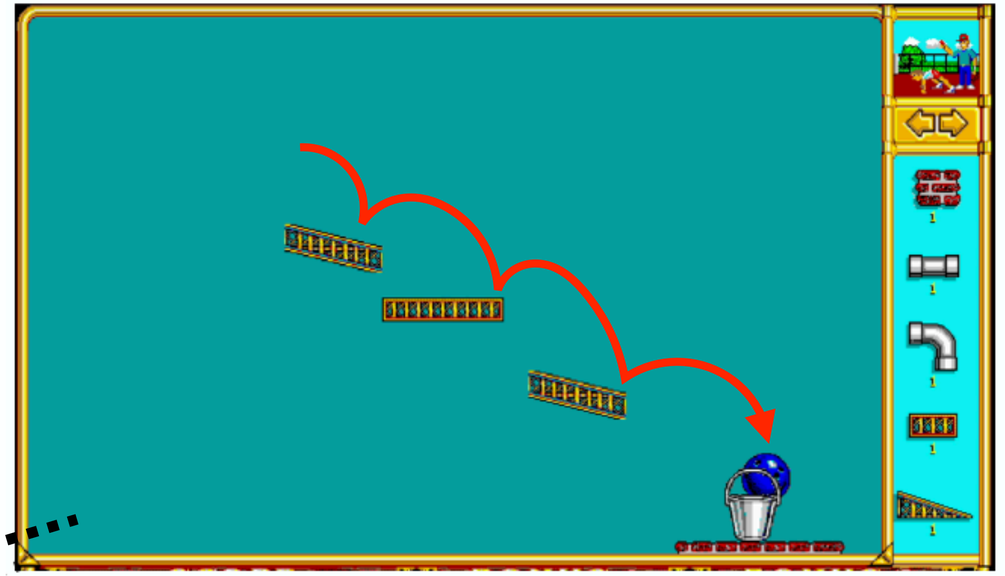


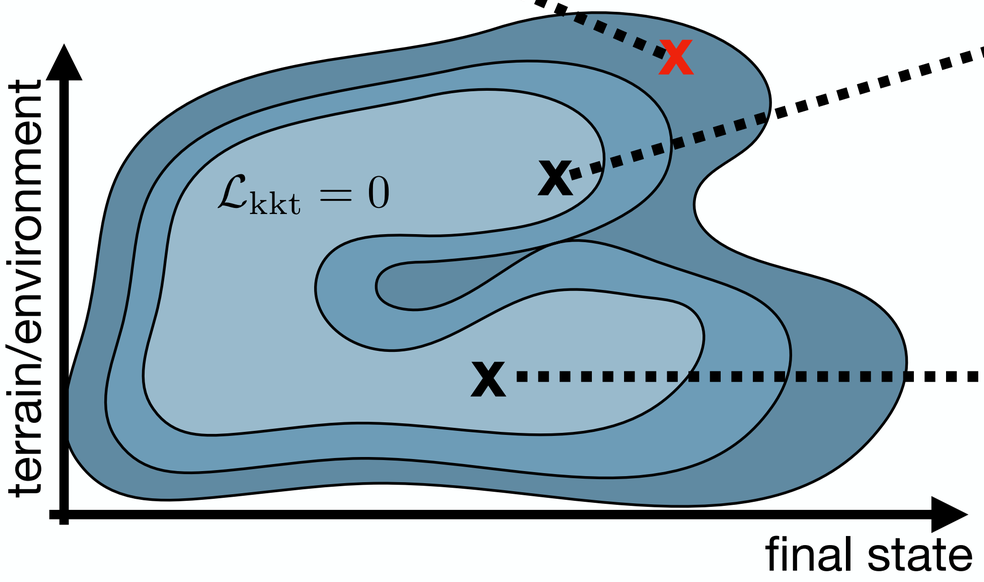


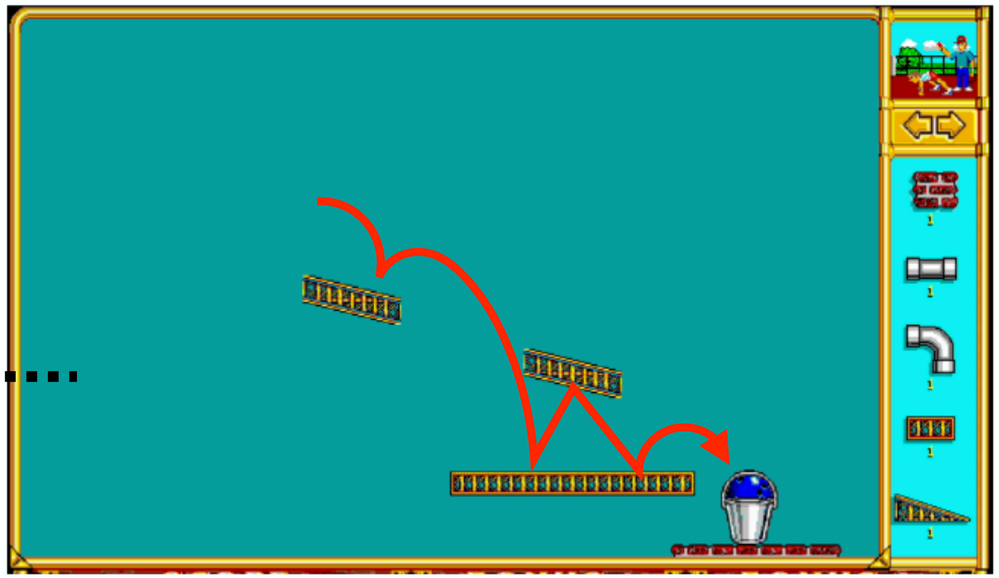

```
loss = loss_kkt(robot_pose, net(input), robot_model)
loss.backward()
```

Incredible Machine (DOS 1993)









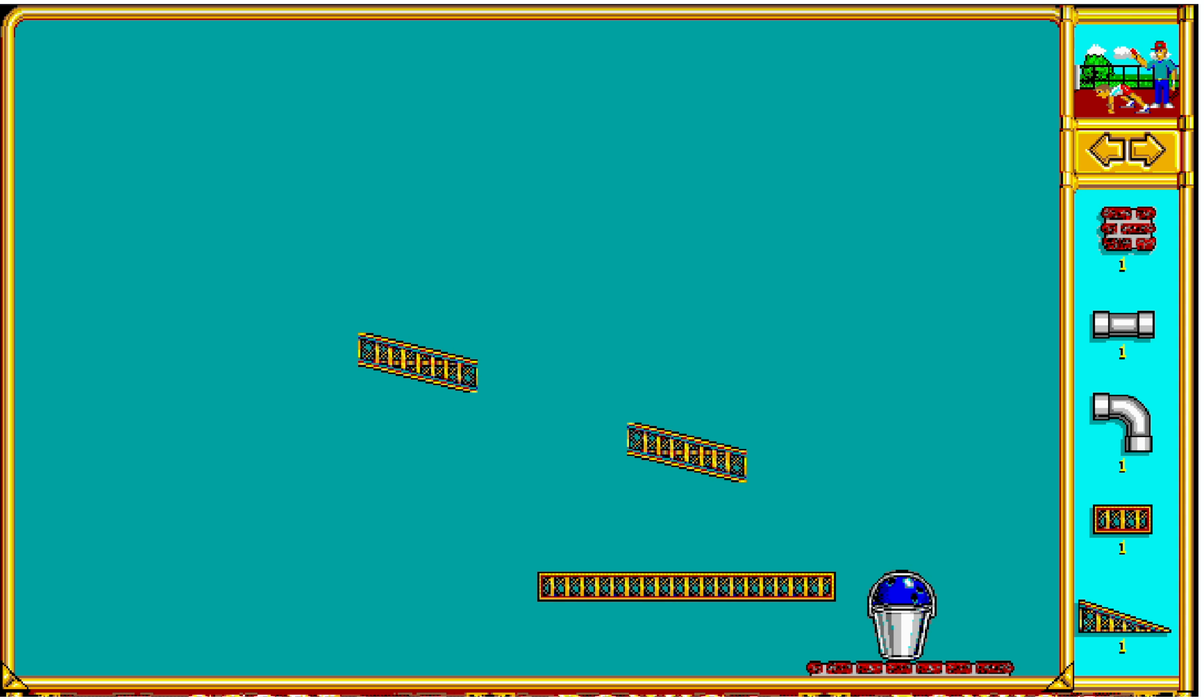
Huge subspace of terrains physically consistent with the final state (ball-in-bucket)

Solution => Environment: Hard, ill-conditioned, combinatorial optimisation problem

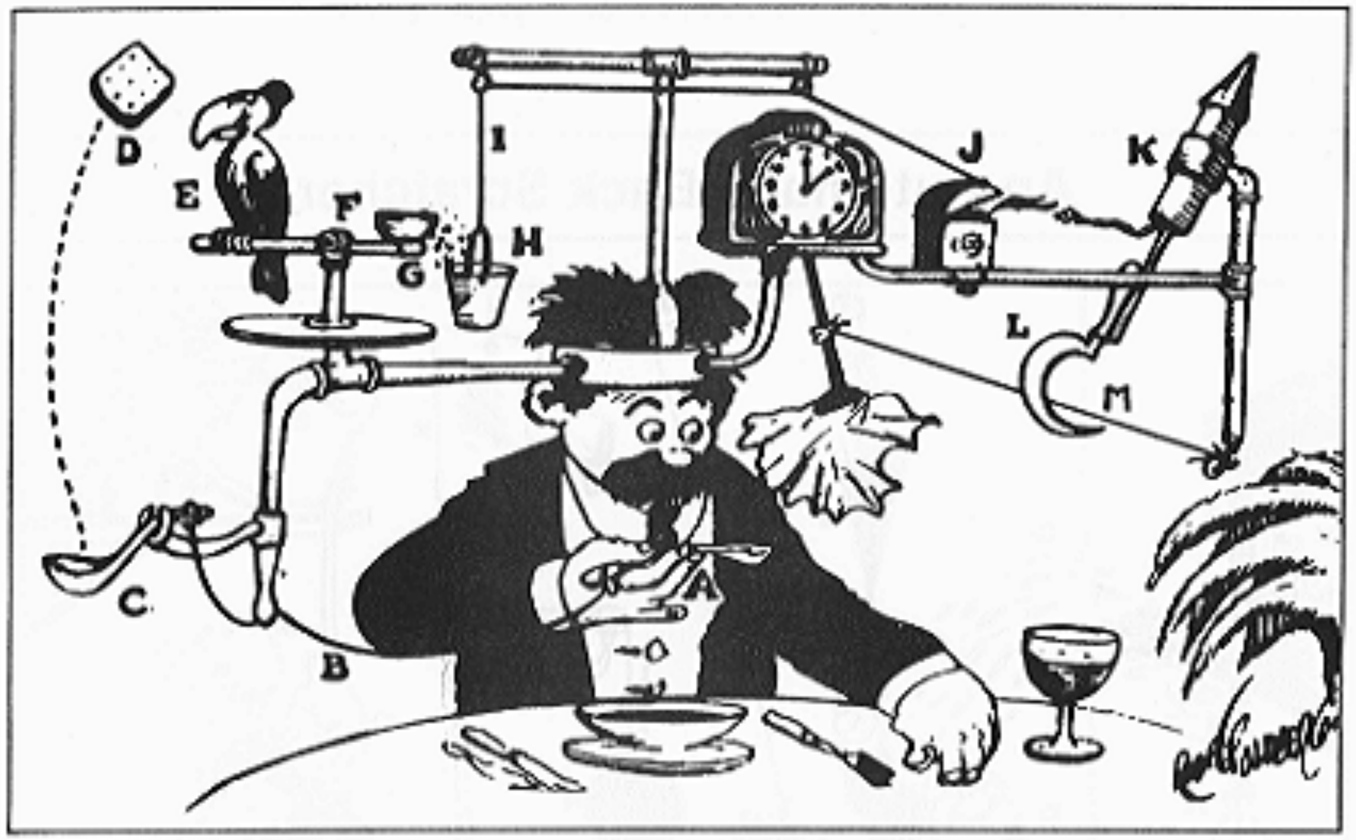
a **solution** has huge number of **ways** that leads into it.

e.g. all “Rube Goldberg’s Incredible Machine”

Incredible Machine (DOS 1993)



Rube Goldberg’s self-operating napkin (1931)



Application II: robot-terrain-interaction-aware SLAM

estimate this

given this

$$\text{MAP:}\quad \mathbf{x}^* = \arg\max_{\mathbf{x}_0 \dots \mathbf{x}_t, \mathbf{m}} p(\mathbf{x}_0 \dots \mathbf{x}_t, \mathbf{m} \mid \mathbf{z}_1 \dots \mathbf{z}_t, \mathbf{u}_1 \dots \mathbf{u}_t) = \arg\min_{\mathbf{x}_0, \dots \mathbf{x}_t, \mathbf{m}} \sum_j f_j(\mathbf{x}, \mathbf{m}, \mathbf{z})^2 \Rightarrow \text{LM}$$

Factorgraph: graphical model of conditional independence among unknown variables

unknown terrain properties (map)

$\mathbf{m}$

$\mathbf{x}_{t-1}$

unknown poses of the robot in time

$\mathbf{x}_{t+1}$

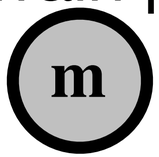
$\mathbf{x}_t$

Application II: robot-terrain-interaction-aware SLAM

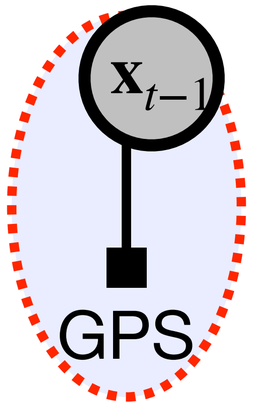
$$\text{MAP: } \mathbf{x}^* = \arg\max_{\mathbf{x}_0 \dots \mathbf{x}_t, \mathbf{m}} p(\underbrace{\mathbf{x}_0 \dots \mathbf{x}_t, \mathbf{m}}_{\text{estimate this}} \mid \underbrace{\mathbf{z}_1 \dots \mathbf{z}_t, \mathbf{u}_1 \dots \mathbf{u}_t}_{\text{given this}}) = \arg\min_{\mathbf{x}_0, \dots \mathbf{x}_t, \mathbf{m}} \sum_j f_j(\mathbf{x}, \mathbf{m}, \mathbf{z})^2 \text{ =>LM}$$

Factorgraph: graphical model of conditional independence among unknown variables

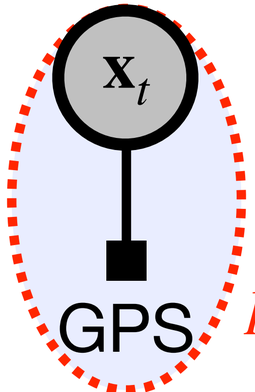
unknown terrain properties (map)



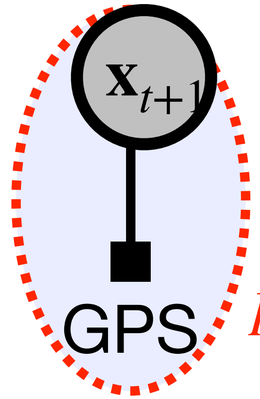
unknown poses of the robot in time



$p(\mathbf{z}_{t-1}^{GPS} \mid \mathbf{x}_{t-1})$



$p(\mathbf{z}_t^{GPS} \mid \mathbf{x}_t)$



$p(\mathbf{z}_{t+1}^{GPS} \mid \mathbf{x}_{t+1})$

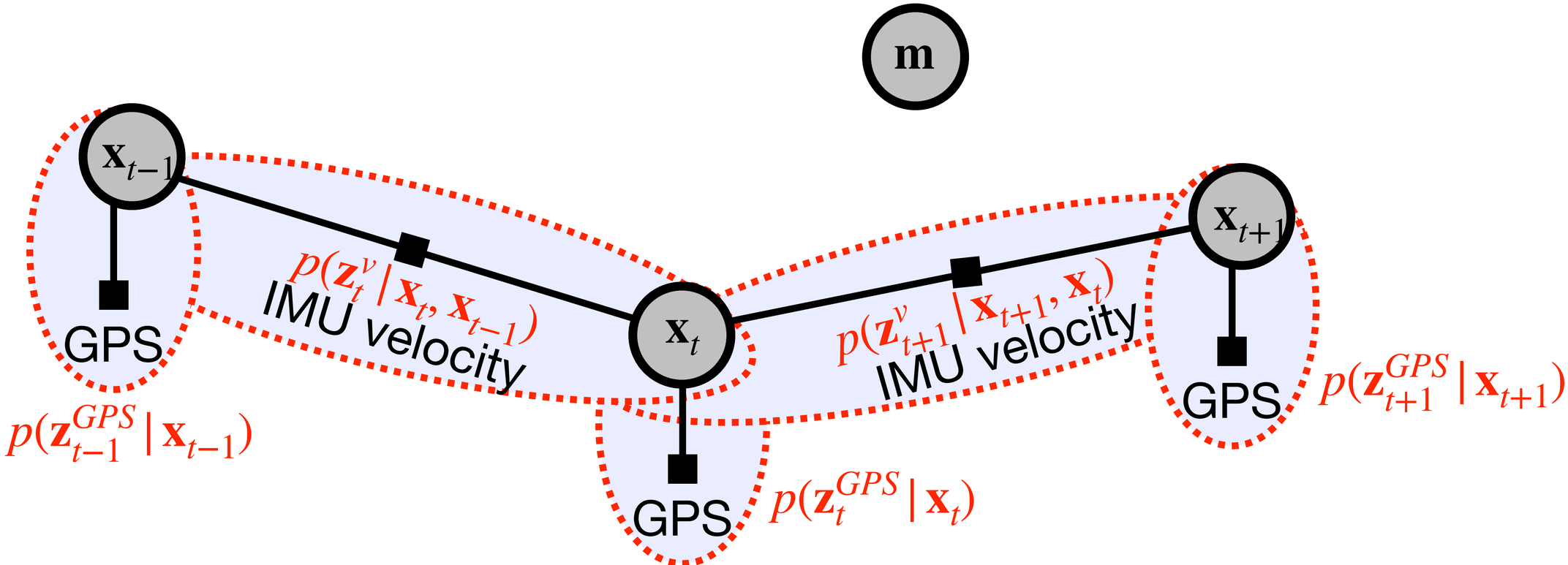
Application II: robot-terrain-interaction-aware SLAM

MAP: $\mathbf{x}^* = \arg\max_{\mathbf{x}_0 \dots \mathbf{x}_t, \mathbf{m}} p(\mathbf{x}_0 \dots \mathbf{x}_t, \mathbf{m} \mid \mathbf{z}_1 \dots \mathbf{z}_t, \mathbf{u}_1 \dots \mathbf{u}_t) = \arg\min_{\mathbf{x}_0, \dots \mathbf{x}_t, \mathbf{m}} \sum_j f_j(\mathbf{x}, \mathbf{m}, \mathbf{z})^2$ =>LM

estimate this

given this

Factorgraph: graphical model of conditional independence among unknown variables



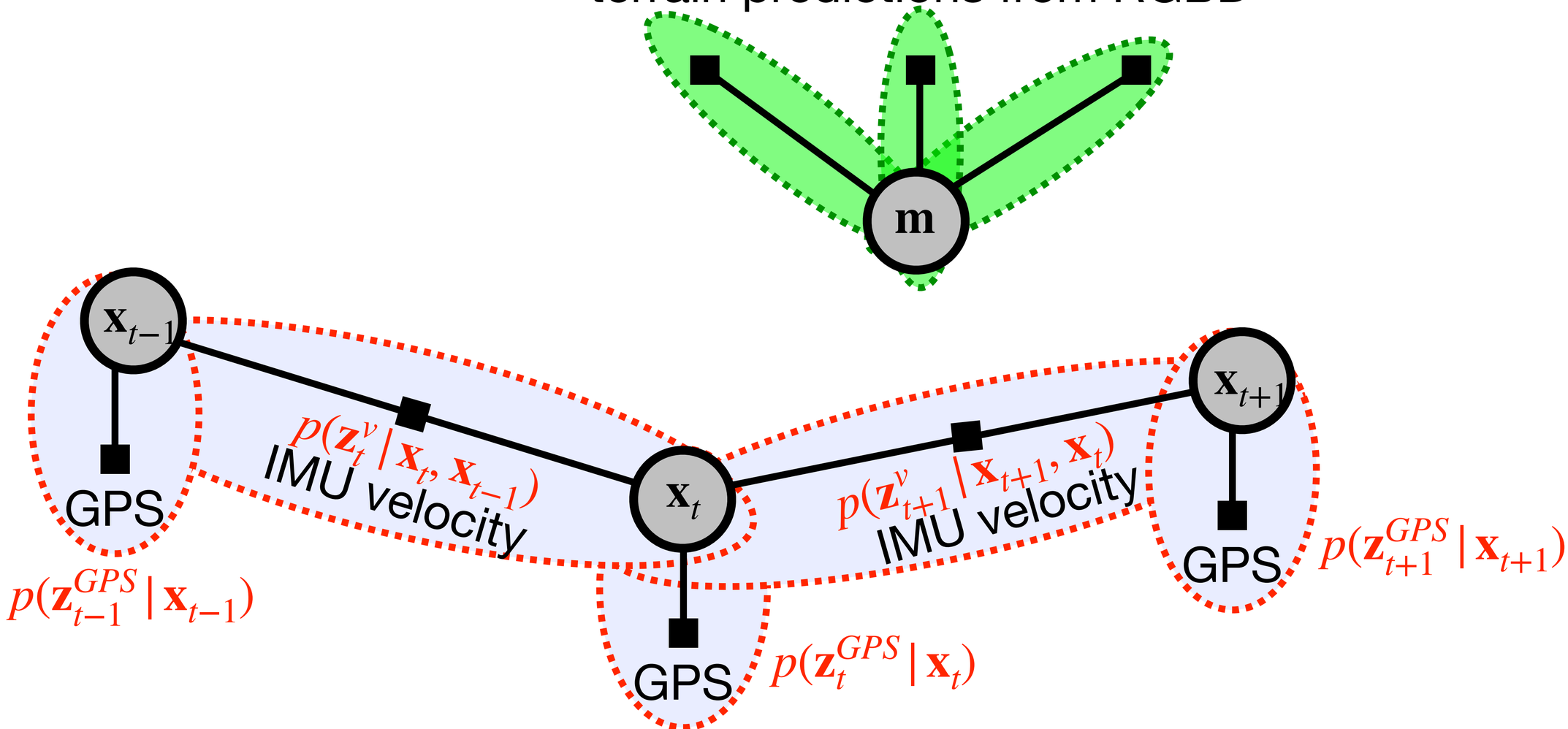
Application II: robot-terrain-interaction-aware SLAM

$$\text{MAP:}\quad \mathbf{x}^* = \underset{\mathbf{x}_0 \ldots \mathbf{x}_t, \mathbf{m}}{\arg\max}\, p(\mathbf{x}_0 \ldots \mathbf{x}_t, \mathbf{m} \mid \mathbf{z}_1 \ldots \mathbf{z}_t, \mathbf{u}_1 \ldots \mathbf{u}_t) = \underset{\mathbf{x}_0, \ldots \mathbf{x}_t, \mathbf{m}}{\arg\min} \sum_j f_j(\mathbf{x}, \mathbf{m}, \mathbf{z})^2 \quad \text{=>LM}$$

estimate this

given this

terrain predictions from RGBD



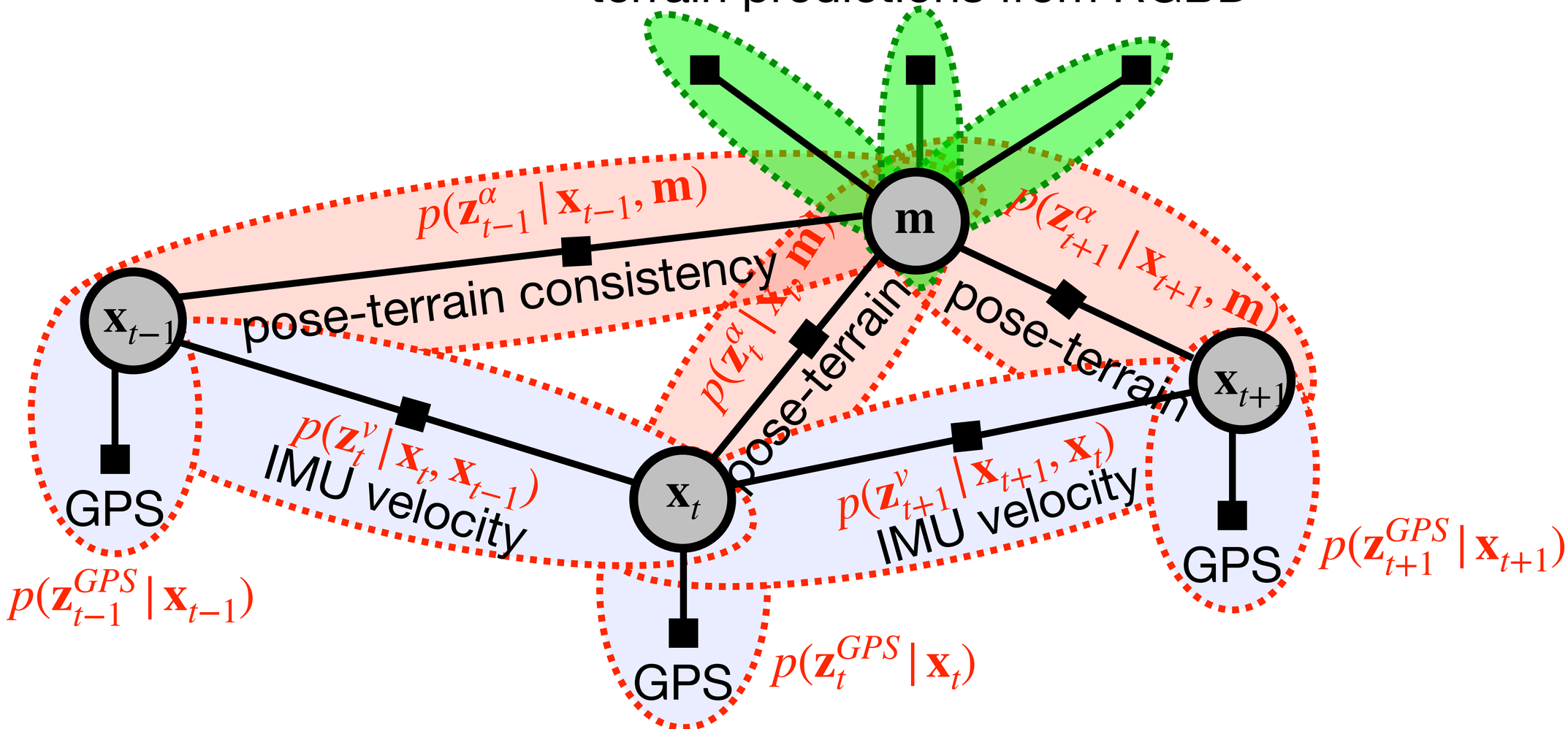
Application II: robot-terrain-interaction-aware SLAM

$$\text{MAP:}\quad \mathbf{x}^* = \arg\max_{\mathbf{x}_0 \ldots \mathbf{x}_t, \mathbf{m}} p(\mathbf{x}_0 \ldots \mathbf{x}_t, \mathbf{m} \mid \mathbf{z}_1 \ldots \mathbf{z}_t, \mathbf{u}_1 \ldots \mathbf{u}_t) = \arg\min_{\mathbf{x}_0, \ldots \mathbf{x}_t, \mathbf{m}} \sum_j f_j(\mathbf{x}, \mathbf{m}, \mathbf{z})^2 \Rightarrow \text{LM}$$

estimate this

given this

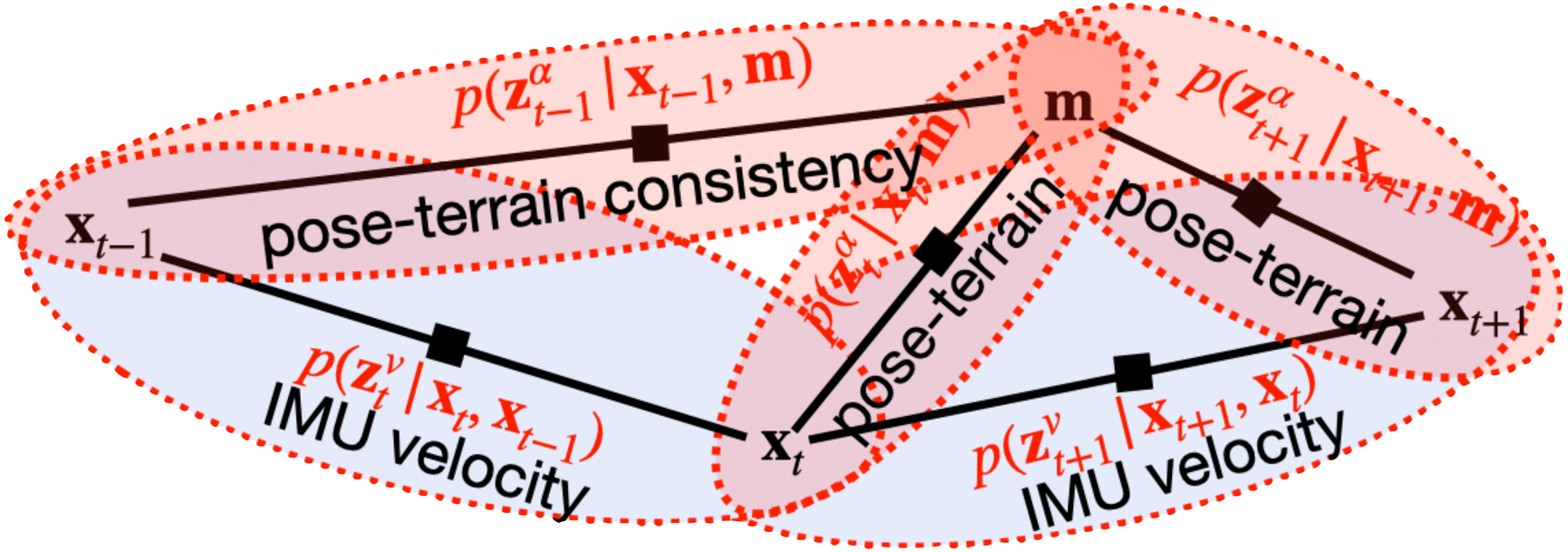
terrain predictions from RGBD



Application II: robot-terrain-interaction-aware SLAM

Given measurement:
roll, pitch, yaw,
track_vel, flippers
$\mathbf{z}_1^{\alpha}, \mathbf{z}_1^{v}, \ldots, \mathbf{z}_T^{\alpha}, \mathbf{z}_T^{v},$

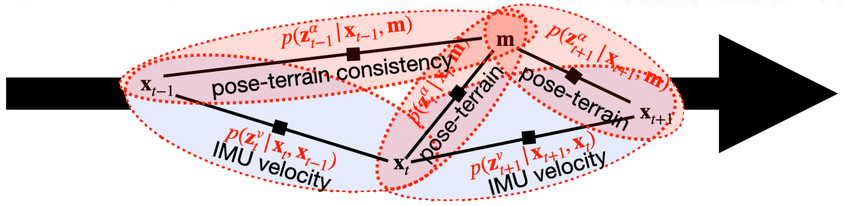
Joint optimization of
$\mathbf{m}, \mathbf{x}_1 \ldots, \mathbf{x}_T$:



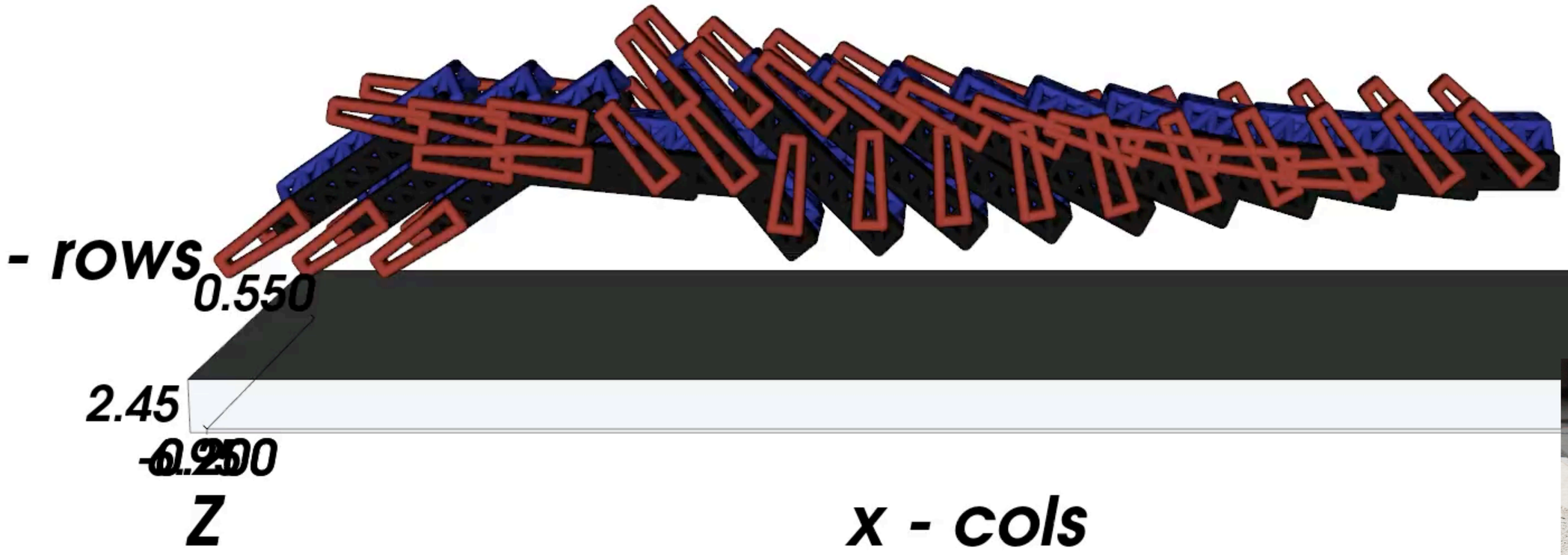
No GPS, no camera, no LIDAR
(just angular velocities and tracks velocities)

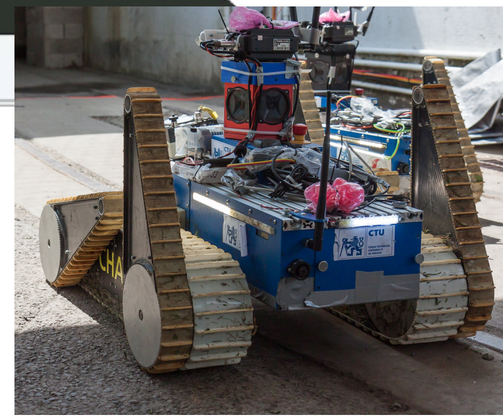
Given measurement:
roll, pitch, yaw,
track_vel, flippers

Lkkt=100.00000



Joint optimization of
$\mathbf{m}, \mathbf{x}_1 \ldots, \mathbf{x}_T$:

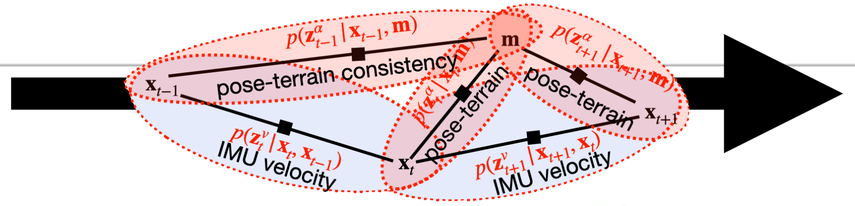




Blind SLAM with **local** loop-closure

Lkkt=0.05023

Given measurement:
roll, pitch, yaw,
track_vel, flippers



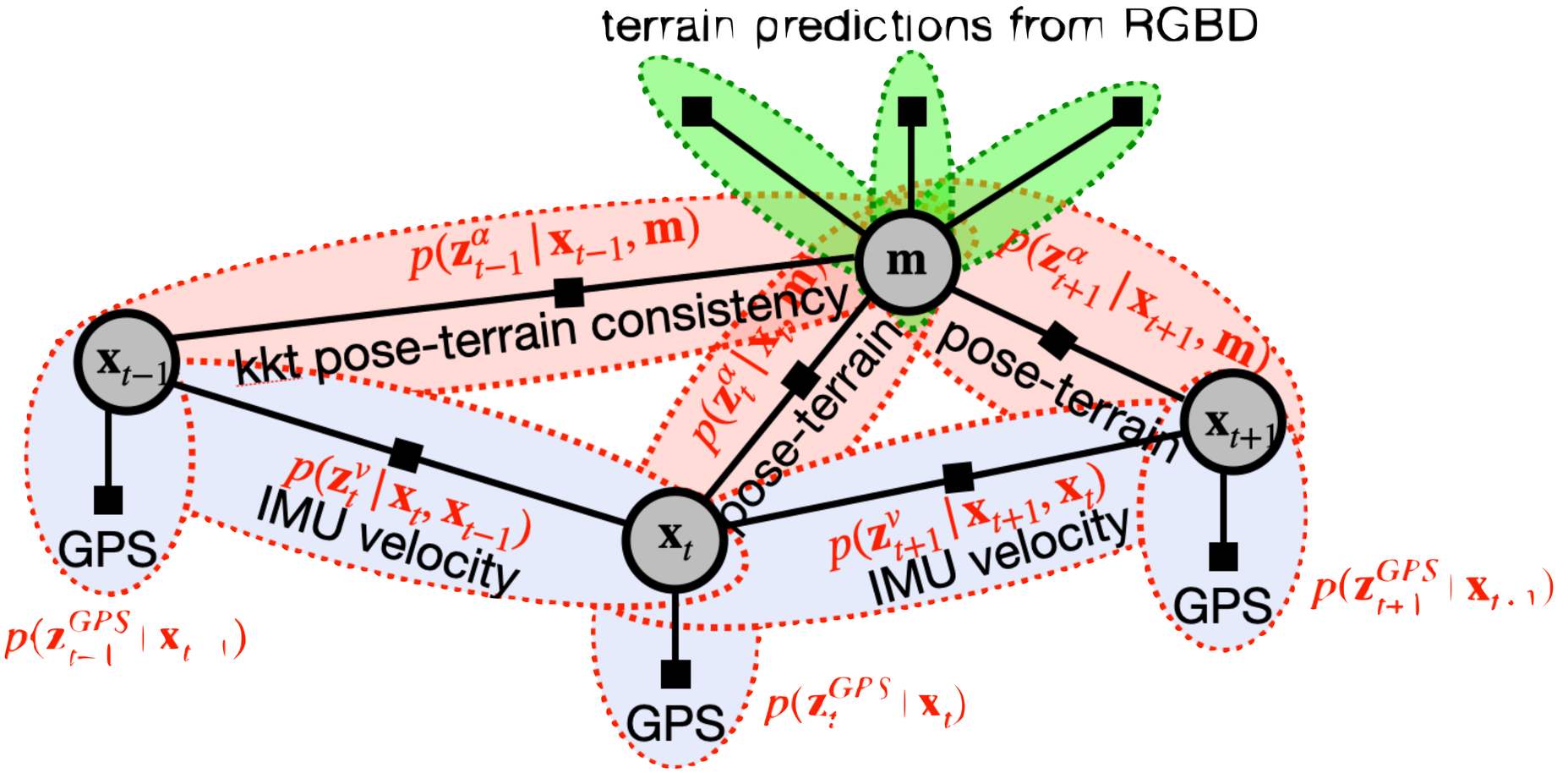
Joint optimization of
$\mathbf{m}, \mathbf{x}_1 \ldots, \mathbf{x}_T$:

0.0500

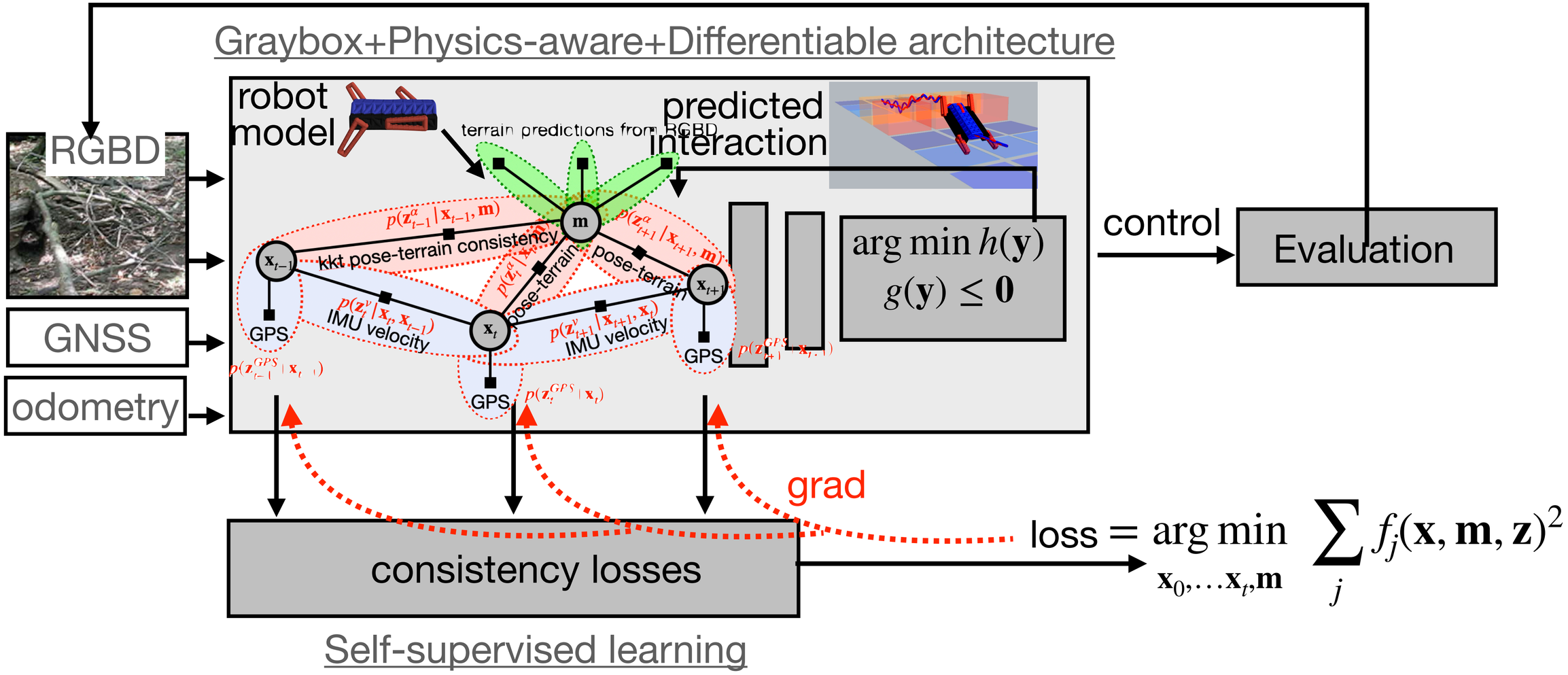
y - rows

Blind SLAM with **global** loop-closure

Application III: self-supervision from factorgraph

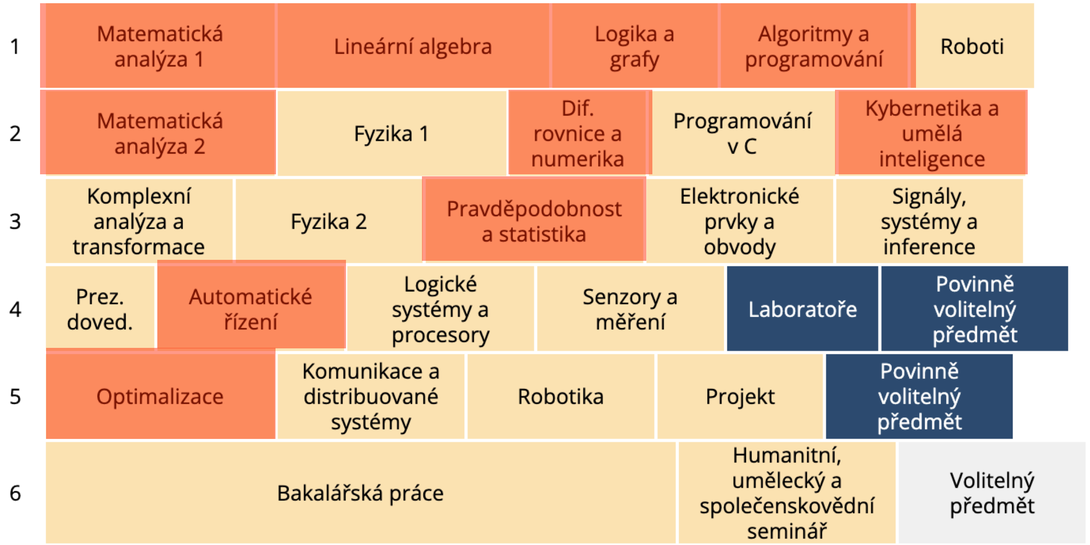


Application III: self-supervision from factorgraph



The take-home message

- Differentiable graybox architecture is **less prone to overfitting**
- Differentiable graybox architecture **enable self-supervision**
- ARO course:
  - builds on top of fundamental principles from KyR theoretical courses (LAG, LGR, MA1/2, PST, OPT, ALP, KUI, …)



- Build probabilistic model=> criterion => suggest solution => critically evaluate
- **Don’t be satisfied with a solution** => always try to find less Ruby Goldberg’s solution (more Interpretable/Explainable/Lepricon-free algorithm).