LINAC4 beyond classical control

V. Kain, N. Bruchon, S. Hirlander, N. Madysa, G. Valentino, S. Tomin, I. Vojskovic, P. Skowronski

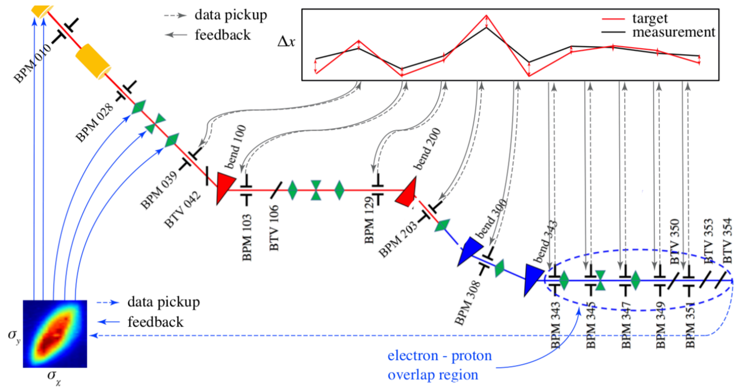
Disclaimer

LINAC4 test bed for advanced algorithms during CERN Long Shutdown 2 (2019/20)

But: limited time due to commissioning tasks.

→ many tests carried out at other facility: e^- line of AWAKE.

- ★ AWAKE: proton-driven plasma wakefield test facility.
- ★ e^- line: 20 MeV RF station, ~ 15 m transport to plasma cell
- ★ AWAKE R&D program for advanced algorithms



Courtesy A. Scheinker

Motivation

Our goal for accelerator operation: maximum efficiency and maximum flexibility while achieving maximum performance

→ physics based deterministic operation of accelerators, no trial and error.

→ = classical control (albeit not standard approach yet either)

Not always possible:

- ★ need models, and models online available; models can be very complicated
  - ✳ LINAC modelling not supported directly by current implementation in CERN control system
- ★ there are drifts → modelling even more complicated
- ★ need sufficient beam instrumentation
- ★ need algorithms on top of models; models not always easily invertible

One way out → automated and sample-efficient numerical optimisation

Reinforcement Learning (RL)

Numerical optimisation needs exploration phase at each deployment.

With RL (after training) exploration phase is reduced to a minimum → one iteration in the best case.

The reason:

- ★ it learns underlying **dynamics of the problem**
- ★ but needs additional input: ***state*** information
    - ✳ Given the ***state***, it applies the ***action*** to achieve maximum ***reward***

→ Controllers like with model-predictive control.

No reinventing the wheel

→ exploit results from python based scientific and industrial community.

CERN has python interface to accelerator control system: `pyjapc`

depend on: `scipy, pymoo, py-bobyqa`,... for numerical optimisation.

depend on: `spinningup` and `standard-baselines` for RL

Key component for algorithm development and comparing algorithms:

★ decision to implement all our problems as ***OpenAI Gym environments*** for RL.

★ extended to also cover numerical optimisation at CERN: ***SingleOptimizable, FunctionOptimizable, OptEnv***

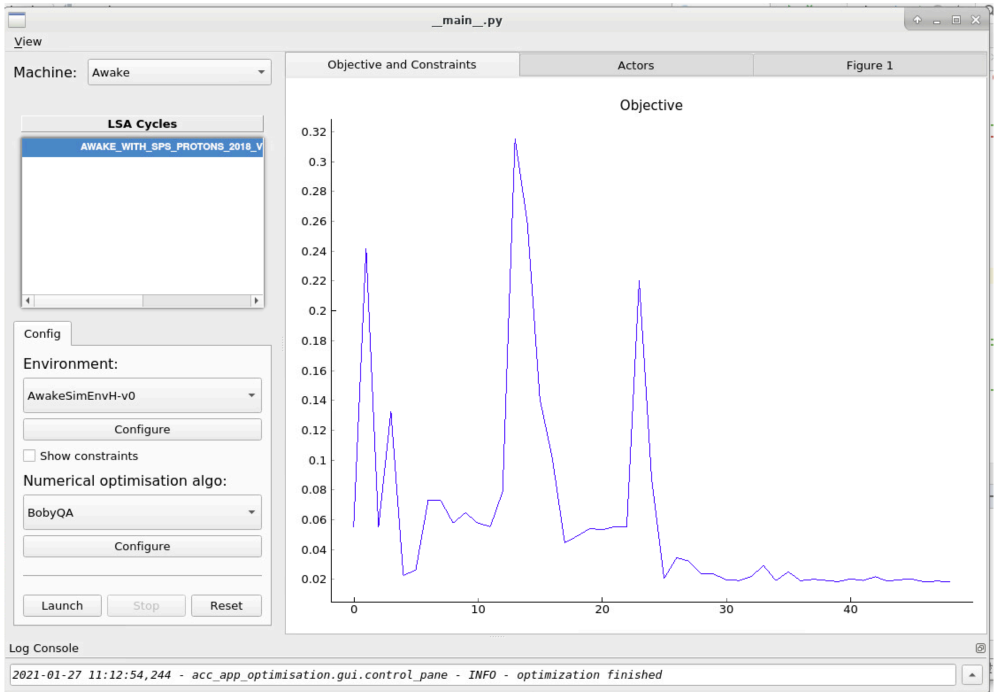
★→ separation of domain specific knowledge in problems by clients from algorithms and GUI

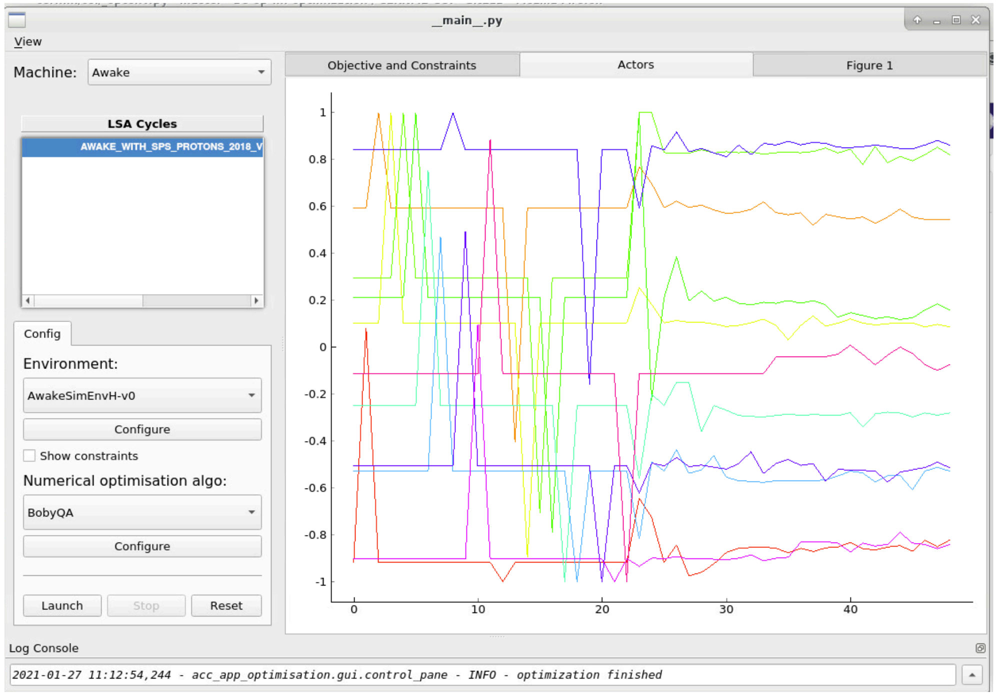
- ✳ Easier algorithm development, easy to switch algorithm for one problem

Plug & Play Optimisation for the control room

GUI for control room based on principles above.

Choice of problems (i.e. *environments),* choice of algorithms



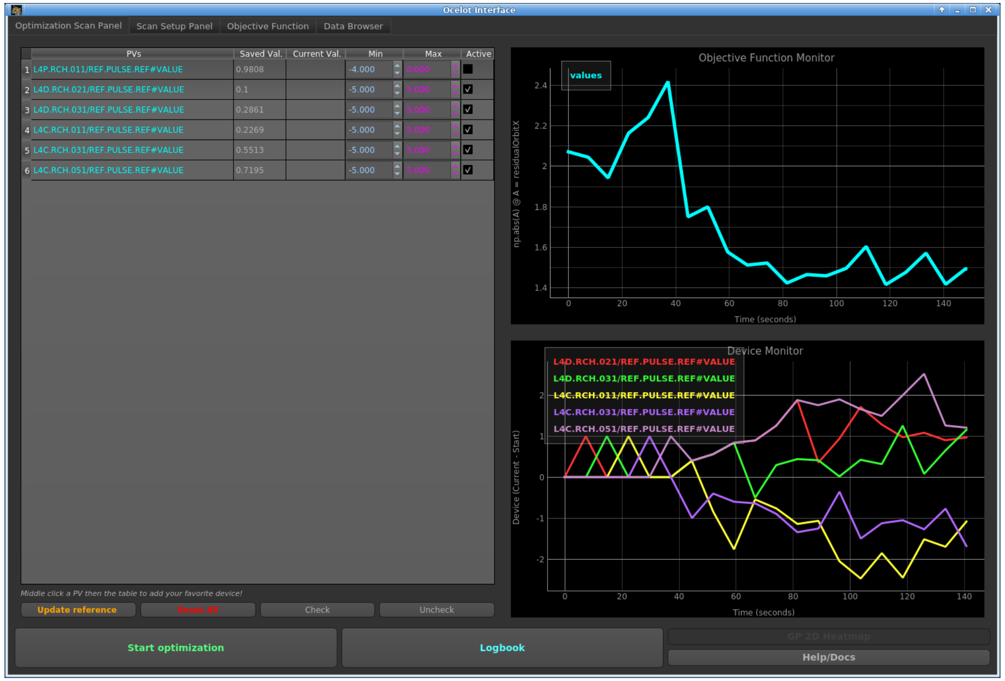


cernml.coi.SingleOptimizable
get_initial_params() compute_single_objective()
optimization_space objective_range constraints

Plug & Play Optimisation for the control room

Investigated OCELOT: https://github.com/ocelot-collab/ocelot

- ★ Used by several labs across world.
- ★ Optimisation tools developed by European XFEL, DESY and SLAC
- ★ More than just numerical optimisation tool: multi-physics software toolkit
  - ✳ also provides modelling and simulation; comes with GUI,…



Nelder-Mead optimisation of trajectory at LINAC4 with 5 correctors in H

not directly fulfilling our use case

→ collaboration to define common interface based on concept of OpenAI Gym

Numerical Optimisation

Many numerical optimisation algorithms available.

Use mainly: Derivative-Free Optimisation or Black Box Optimisation ≡ Model Unaware Algorithms

- ★ model does not have to be available
- ★ least investment necessary upfront

Task:
solve $\overrightarrow{x_0} = \mathbf{argmin}\, f(\overrightarrow{x})$

Not all derivative-free algorithms suitable for all optimisation problems.

- ★ Is problem convex? Need Bounds or constraints? What about noise?

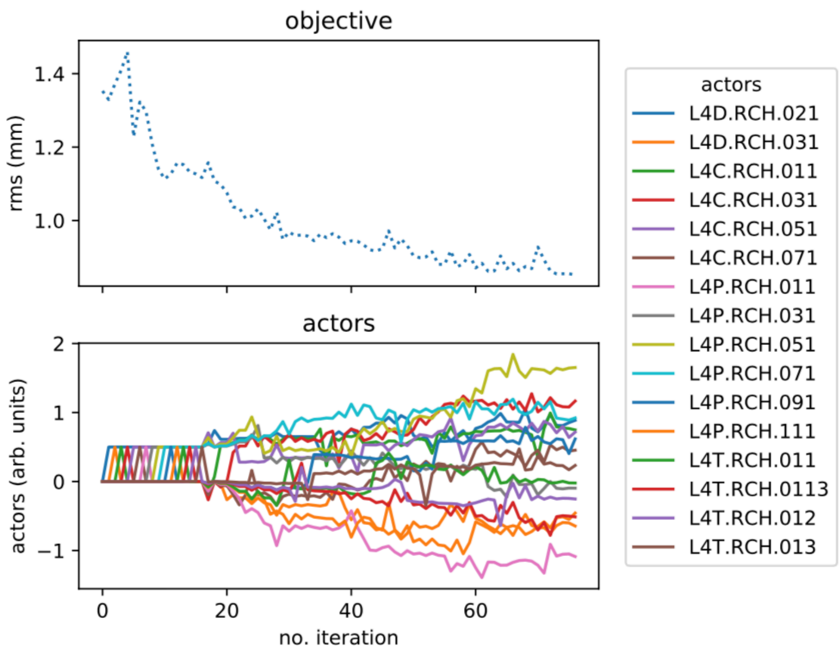
Our favourites: `COBYLA, BOBYQA`

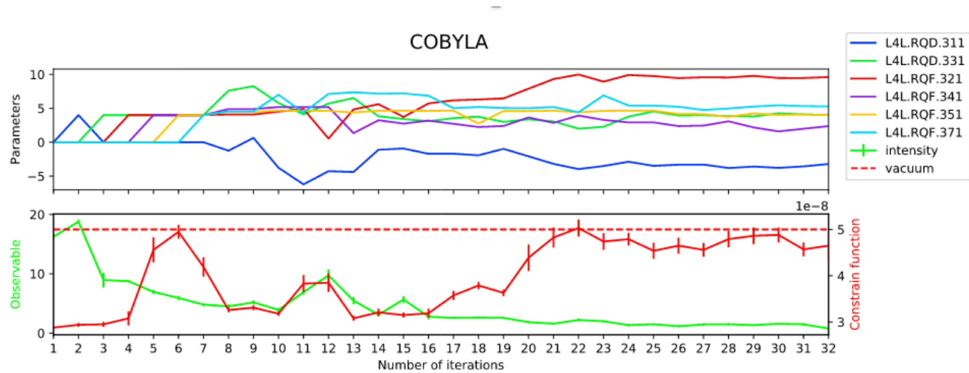
Next: provide for Bayesian Optimisation

Examples @ LINAC4

Examples for online numerical optimisation @ LINAC4

- ★ Trajectory steering in LINAC, 16 degrees of freedom; `COBYLA`
- ★ Chopping efficiency optimisation with constraints; `COBYLA`





Vacuum readings before chopper dump as BLMs → constraint for minimising pulse shape error while adjusting 6 quadrupoles

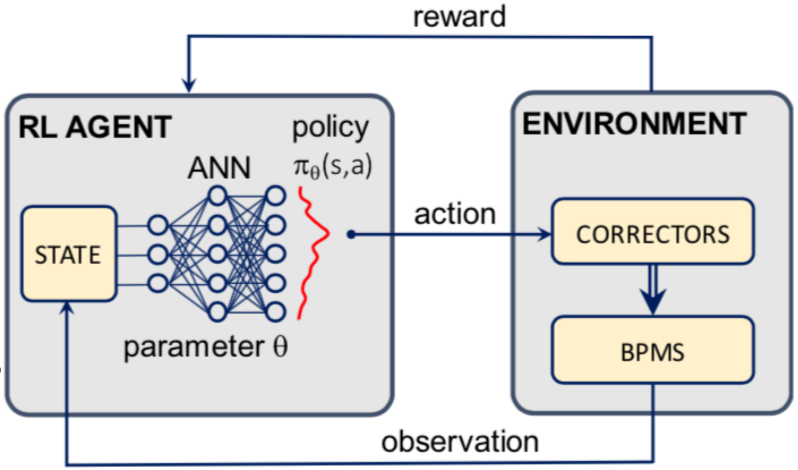
Basics of Reinforcement Learning

RL: learning how to **act** given a certain state to maximise cumulative reward.

Simple example: trajectory steering

State s :
- reading of BPMs

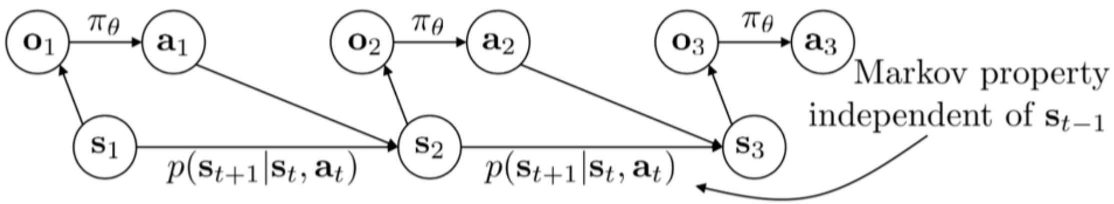
Action a :
- dipole corrector settings



Reward r could be:
- intensity on target
- RMS of trajectory $\times(-1)$
- losses $\times(-1)$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

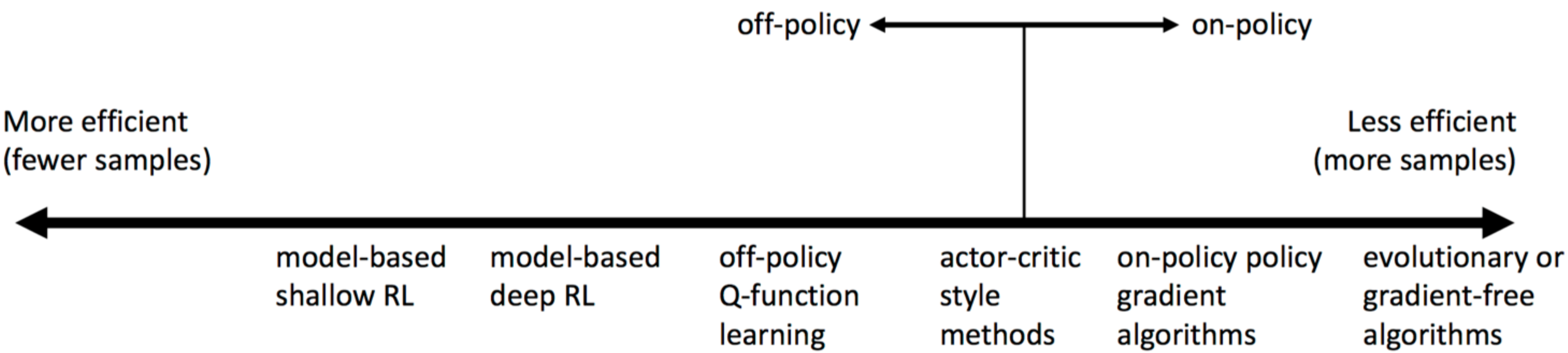
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



Partly from course "Deep Reinforcement Learning", Sergey Levine

Sample efficiency

How many interactions does RL algorithm need until it has learned the optimal policy/Q-function/…?



From course "Deep Reinforcement Learning", Sergey Levine

Machine time is expensive. Some algorithms are excluded on the machine (PPO,…)

→ because of algorithm simplicity started with: Q-learning and Actor-critic methods

→ then moved to model-based RL: albeit only some methods studied so far

Model-free RL test bed 2019

AWAKE e^- line and commissioning run of H^- LINAC4

Initial test cases on AWAKE and later for LINAC4: **trajectory correction**

- ★ **ideal test case**
- ★ well defined state s
- ★ high dimensional action and state space
- ★ can compare with existing algorithms and can solve the problem analytically.

Goal: train controller that corrects as well as SVD → similar RMS and ideally within 1 iteration.

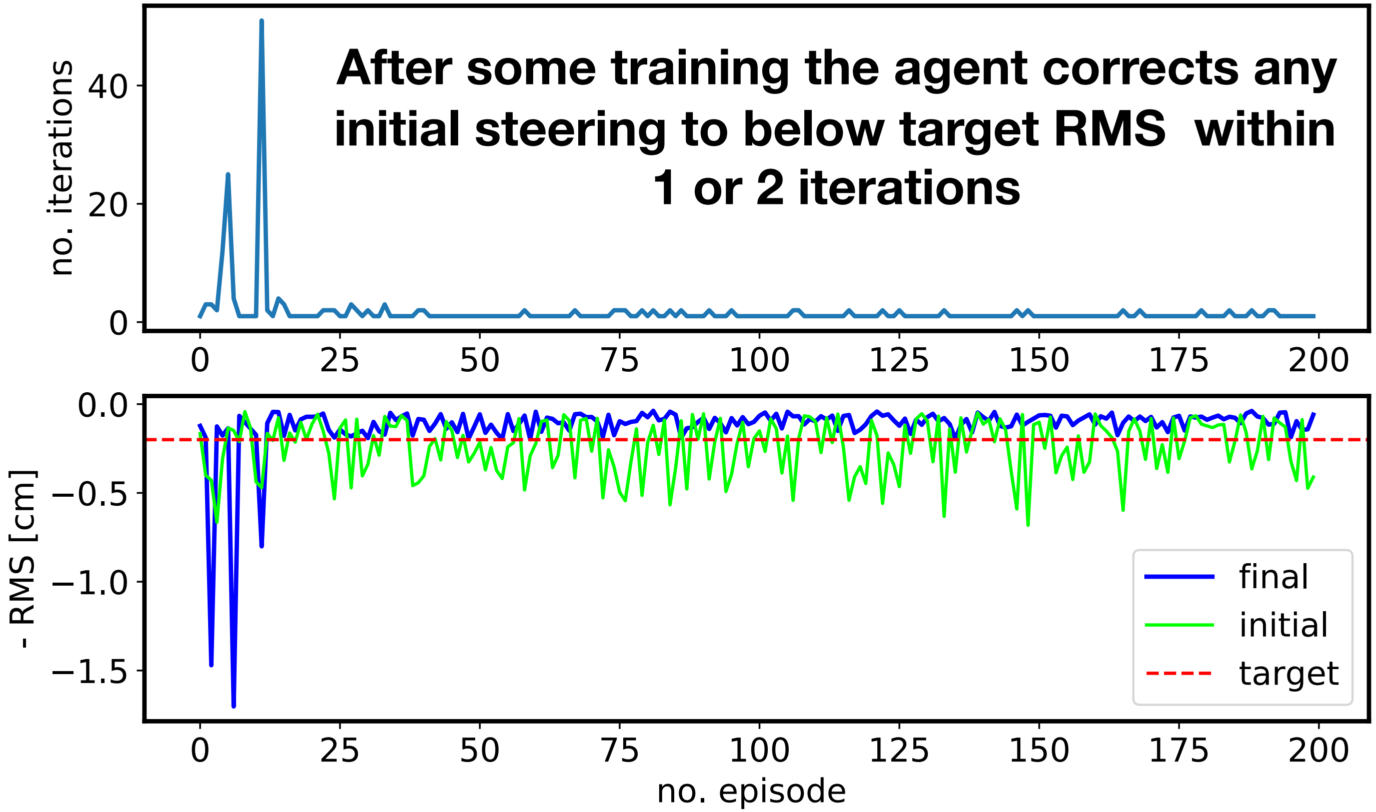
Implemented NAF *arXiv: 1603.00748* with *Prioritised Experience Replay: arXiv:1511.05952*

Also used DDPG variant TD3 from package `stable-baselines`

Model-free online learning for AWAKE trajectory steering

Proof-of-principle: learn how to steer AWAKE e^- - line in H

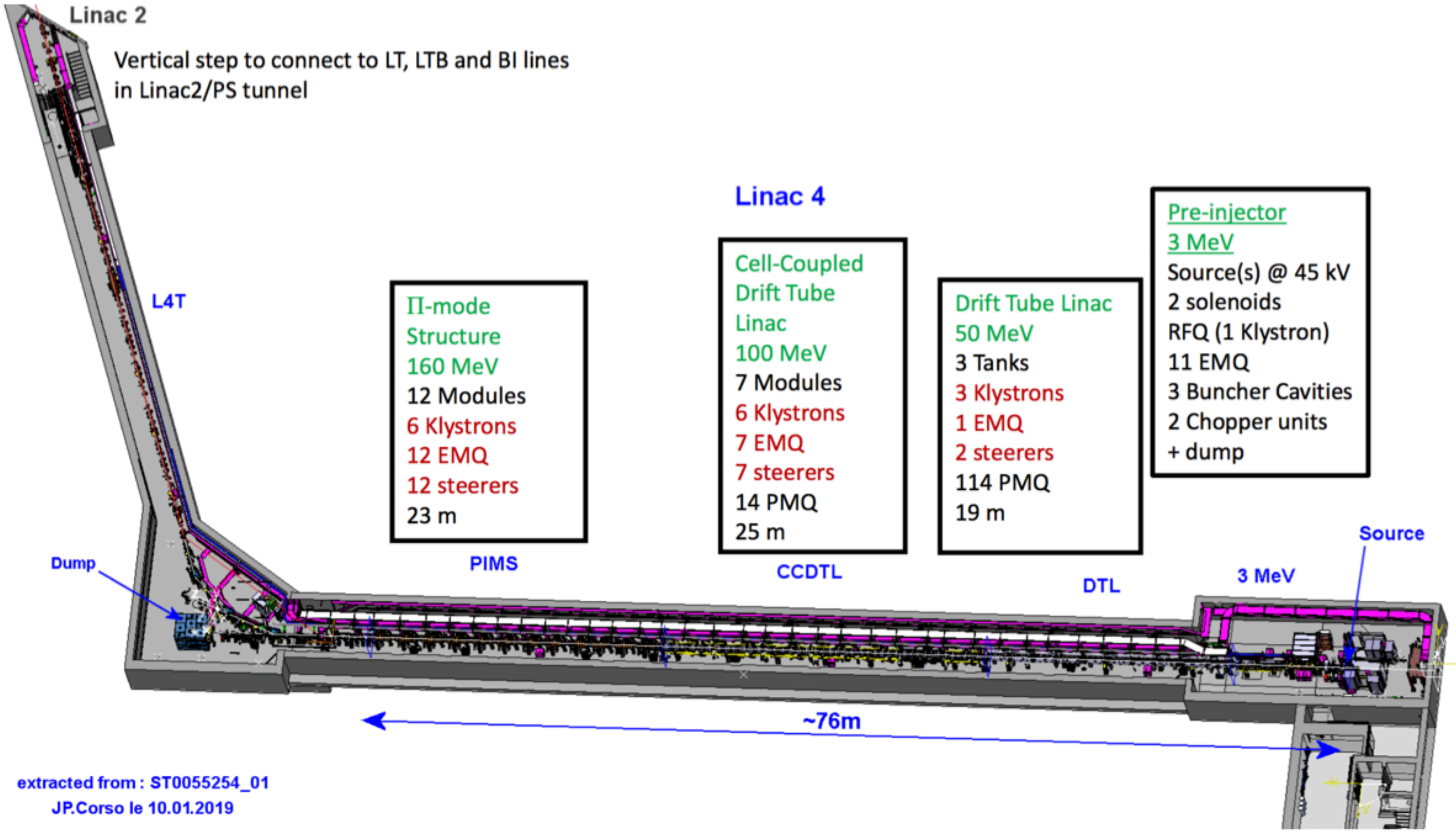
Q-learning with very sample-efficient NAF algorithm



Problem with 10 DOF

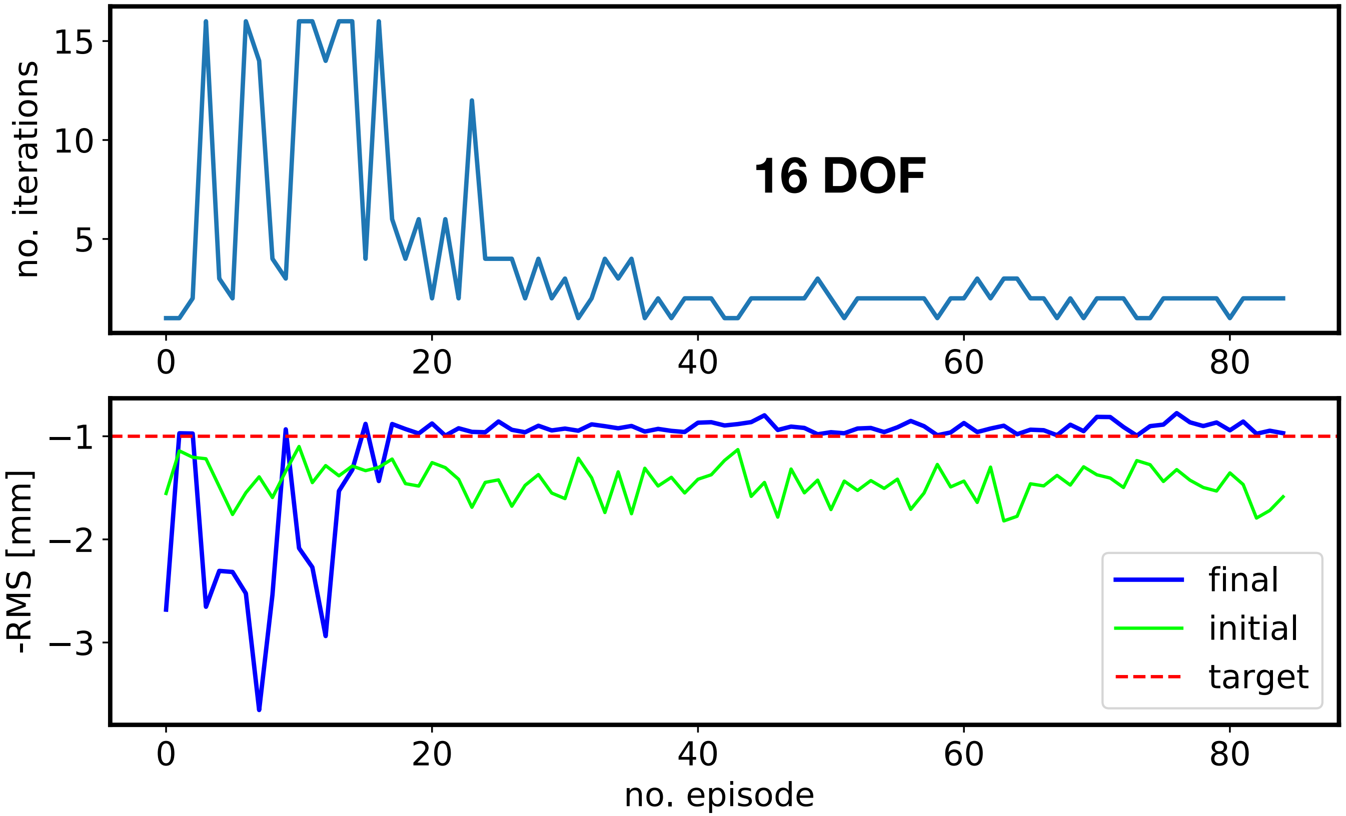
Other example with NAF: agent for LINAC4 steering

17 BPMs and actions possible on 16 correctors, through DTL, CCDTL, PIMS and start of the transfer line in the horizontal plane



Other example with NAF: agent for LINAC4 steering

Inexpensive way of learning any (also non-linear) response and solve control problem.



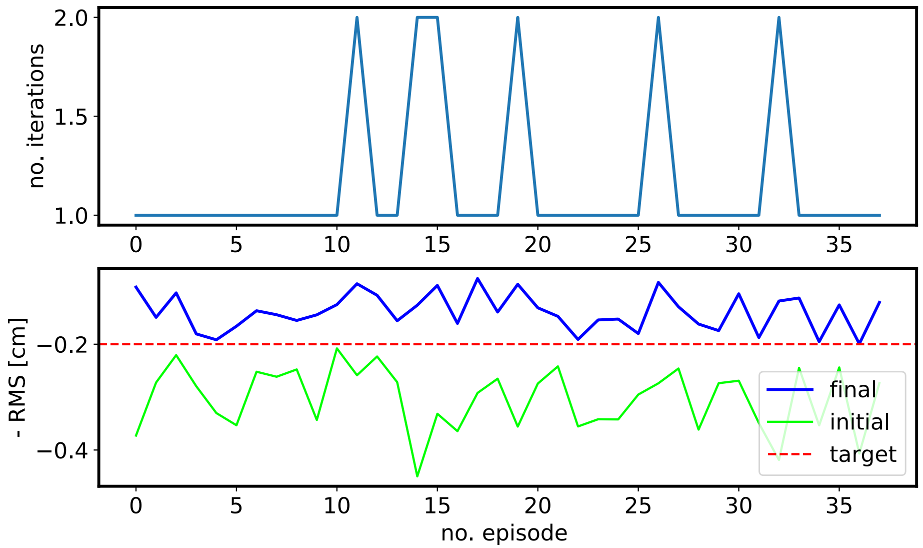
How often does one have to re-train?

Depends in general on

- ★ the problem: e.g. trajectory steering will need re-training if lattice is changed. No difference to SVD.
- ★ hidden state information
- ★ ...

The training time of NAF on the examples shown is acceptable if training remains valid for a long time (e.g. a run)

- ★ ~ 300 iterations: ~ 30 minutes for AWAKE trajectory steering agent
- ★ Test in September 2020 of agent that was trained in June 2020. **No degradation of correction performance**



Agent trained on June 10
Validation September 22

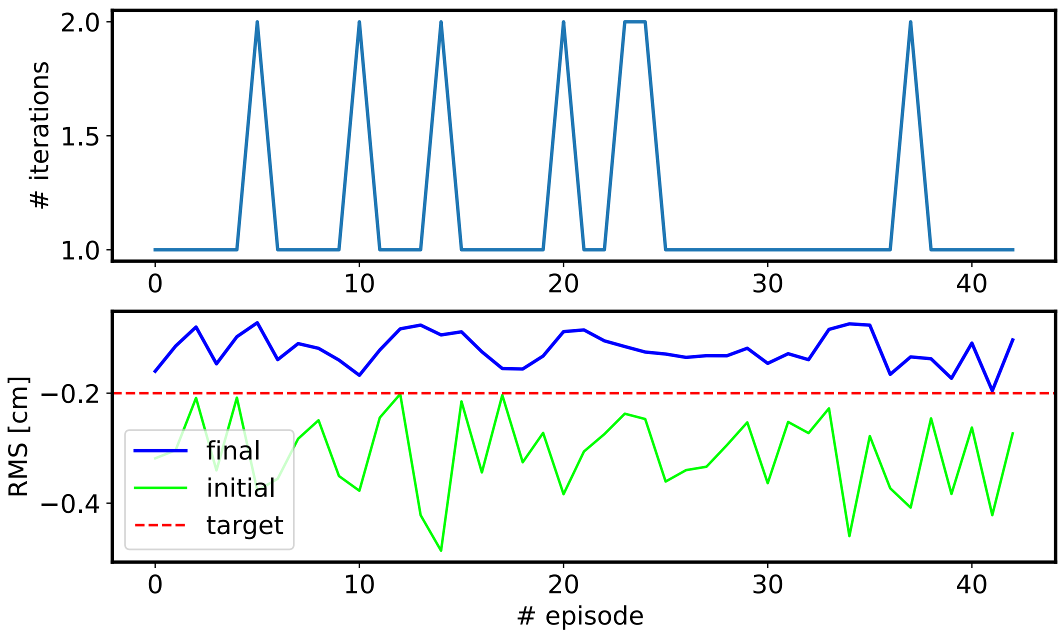
Train on simulation and apply on machine?

2 ways to circumvent the *sample efficiency* issue even further

→ Model-based RL: learn explicitly the model and train agent at the same time; see this talk

→ Train on simulation, apply on machine (transfer learning): typically relies on high level parameter control system

AWAKE training on simulation for trajectory steering; validation of trained agent on machine



If simulation and machine not perfect match, could use "residual physics"

Model-based RL

Learn model of dynamics explicitly and use it to train agent, instead of machine directly.

Many variants.

Used the DYNA-style MBRL (Sutton)

Dyna-Q algorithm:

Train dynamics model with supervised learning

Train model-free agent

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow \underline{Model(S,A)}$
$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$

...Our code is using `stable-baselines` model-free agents

Model-predictive control: iLQR

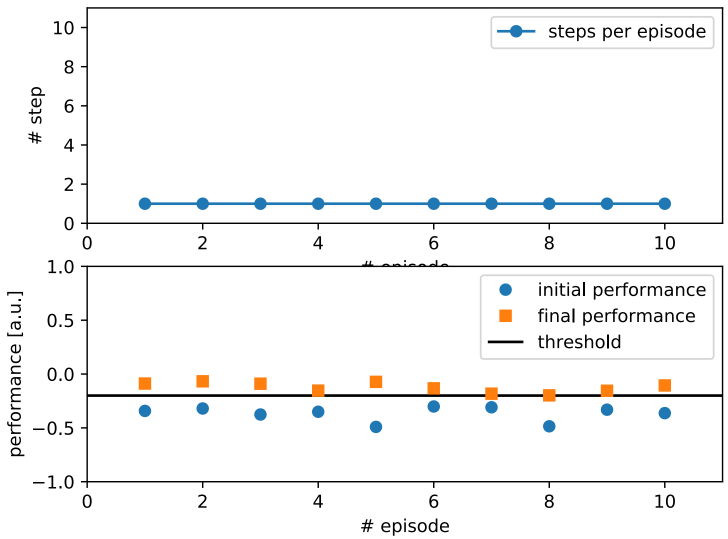
What if go through the loop only once and use MPC?

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

iLQR on the dynamics model for AWAKE trajectory correction.

Problem statement: Find corrector settings (10) to flatten trajectory from any initial trajectory (10 BPMs).

Results with dynamics training on 200 data points



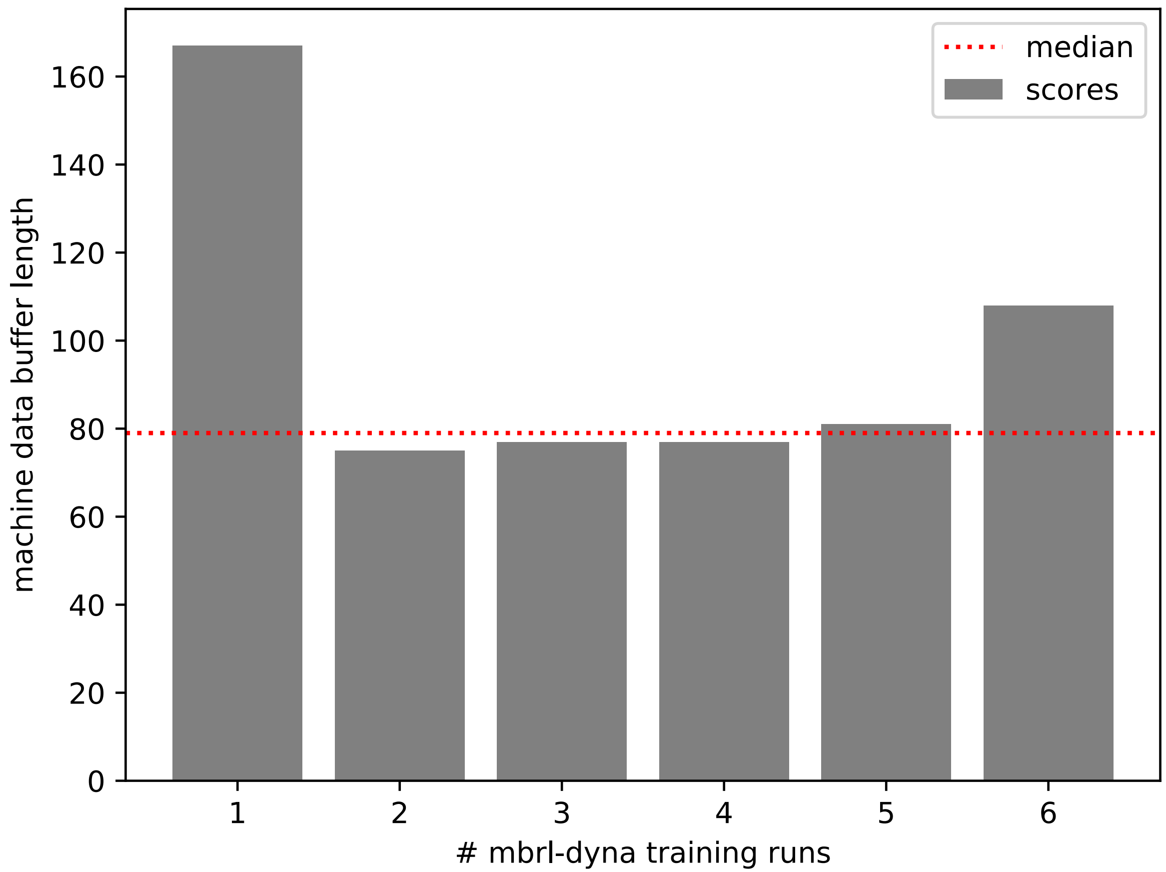
N. Bruchon et al.

MB-RL performance

Repeated agent training for AWAKE trajectory steering.

For comparison: model-free training ~ 300 iterations

Model-based RL:
Median ~ 80 data sets



Concluding words

Next step in efficiency, reproducibility and performance of our accelerators will include machine learning and other advanced algorithms

- ★ Algorithms for parameter tuning such as presented in this talk
- ★ But also ML for: forecasting (hysteresis correction,…), virtual diagnostics, de-noising, computer vision,…

These algorithms are mature enough now to really profit from them.

Generic optimisation and Machine Learning framework: key ingredient for successful exploitation

- ★ We are working on it for the CERN complex. First version already available
- ★ Also includes storing and loading neural nets

LINAC4 was a test bed for some of the algorithm developments in 2019.

- ★Many potential applications; first one in the making: RL for dispersion free steering.

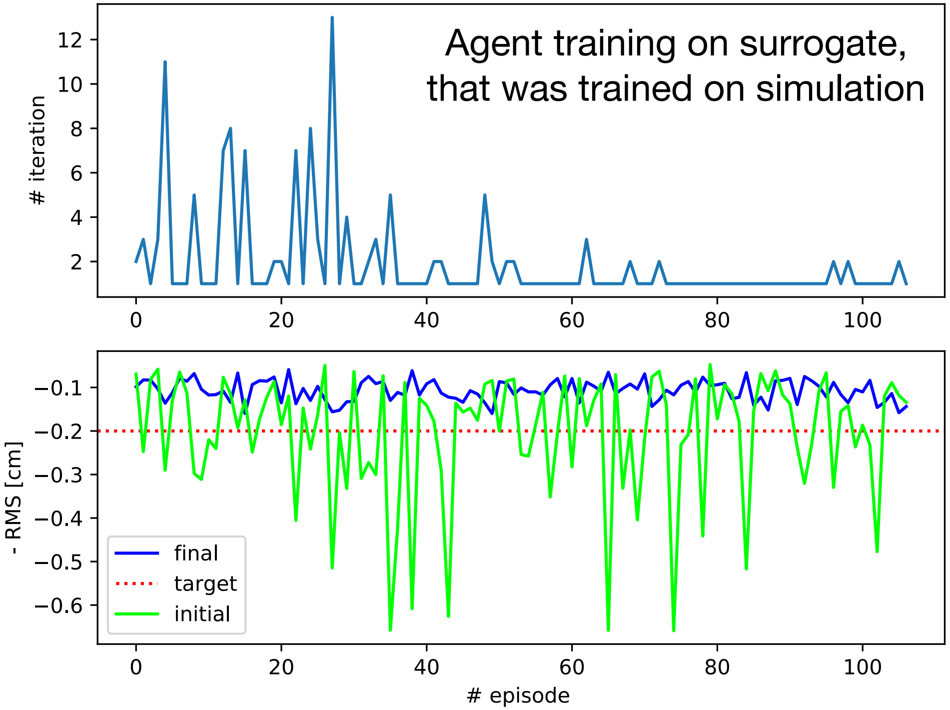
Many CERN accelerators are planning ML tools for the coming start-up.

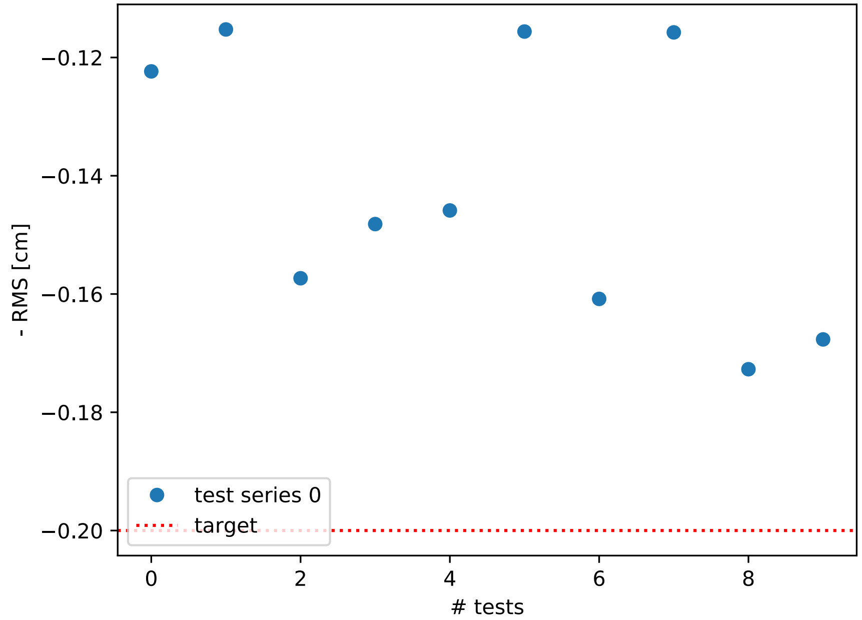
Extra

Model-based RL exploiting simulation

Only needed 15 data samples on machine.

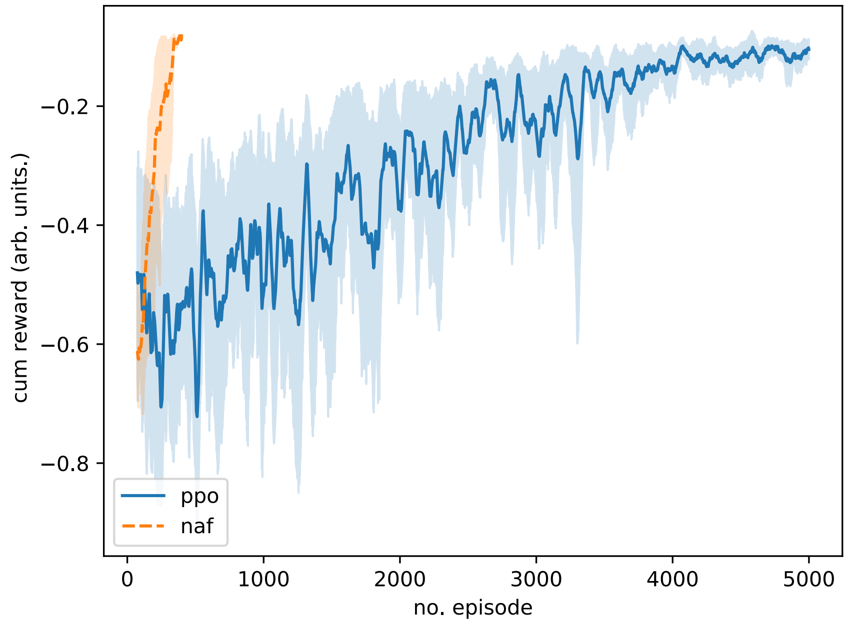
Data: 2020_08_12_16_45_46



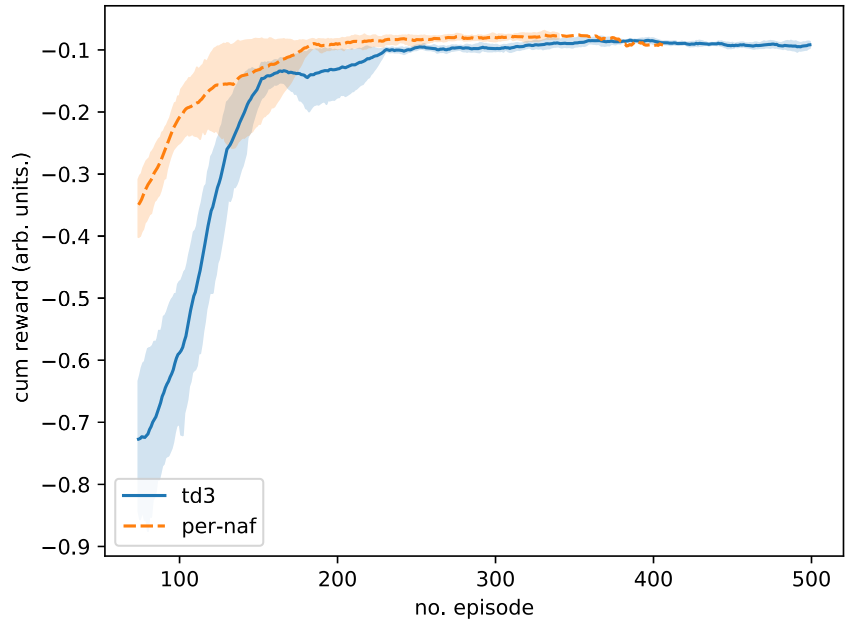


Comparison with other algorithms

Policy-gradient algorithm PPO versus NAF for AWAKE steering problem in simulation:



TD3 versus NAF for AWAKE steering problem in simulation: similar performance



→ Q **- learning much more sample efficient than policy gradient algorithms**