A closer look at RL for beam-based feedback systems

Leander Grech
Simon Hirlaender
Gianluca Valentino

L-Università ta' Malta
Xjenza — The Malta Council for Science & Technology
IDA LAB — INTELLIGENT DATA ANALYTICS SALZBURG
PARIS LODRON UNIVERSITÄT SALZBURG

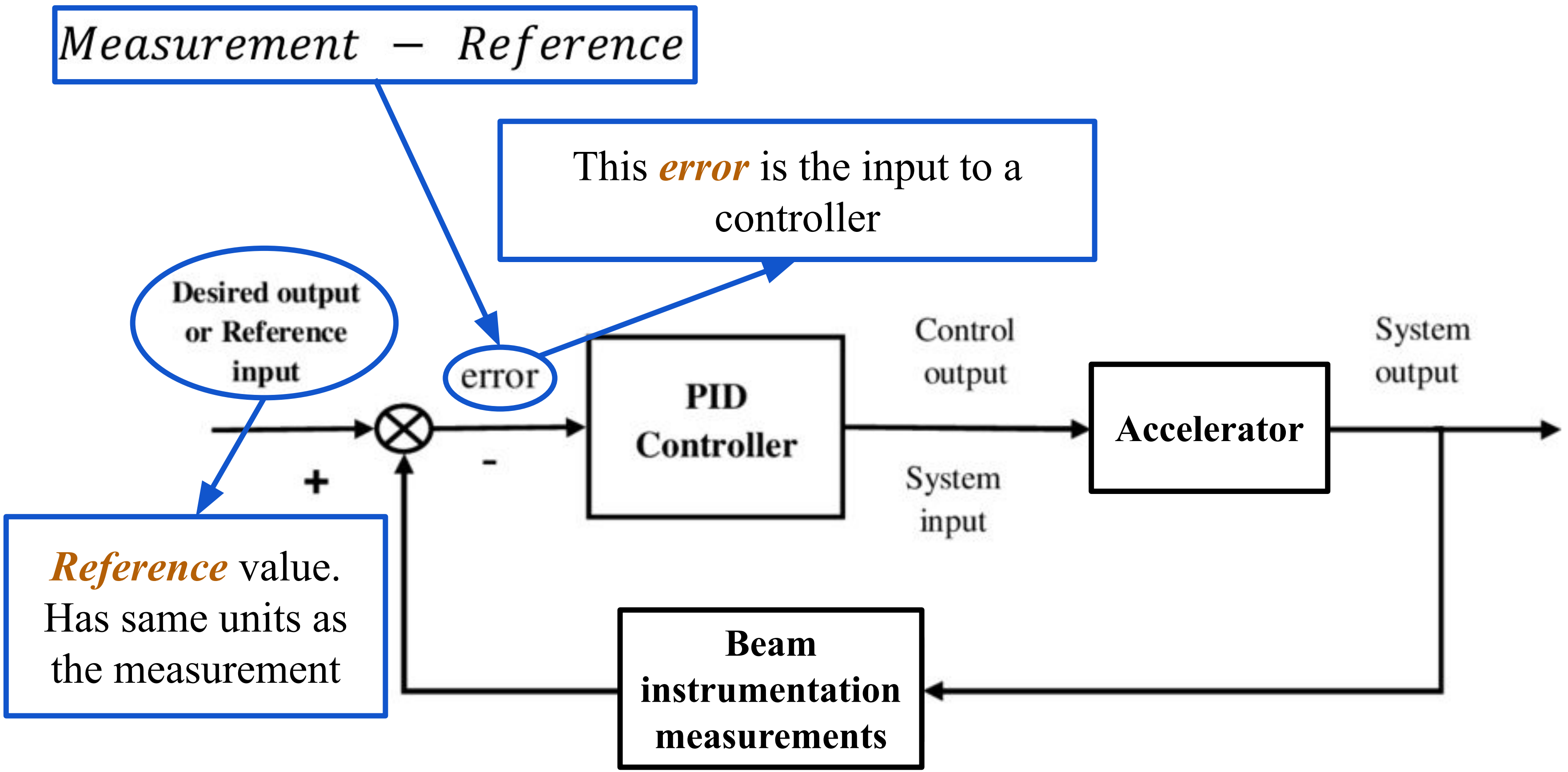
Outline

1. What are **beam-based feedback (BBF)** systems?
2. Feasibility study on the **application of RL on QFB**
3. Development & testing of **RandomEnv (RE)**
4. Testing **state-of-the-art RL** algorithms
5. Simplification of **RE into discrete actions**
    a. Tabular RL approach
    b. Linear RL approach

What are beam-based feedback (BBF) systems?

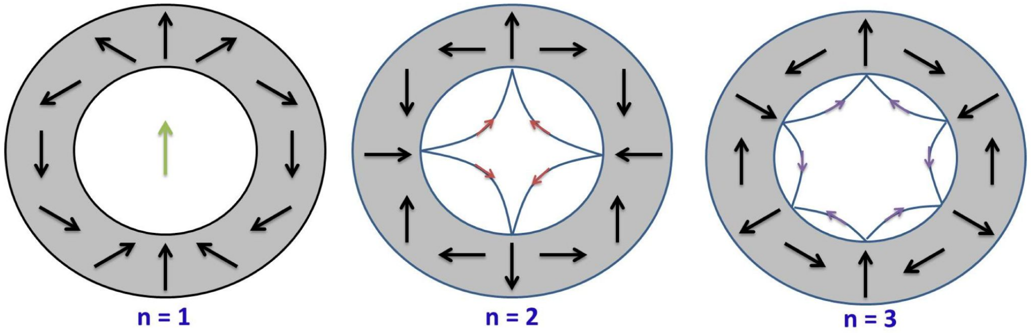
1/5

Beam-based feedback (BBF) systems



Beam-based feedback (BBF) systems

- Beam and machine parameters are modelled quite accurately
    - **Linear models** - transfer matrices
- PID controllers use **inverse transfer matrix** to correct magnet currents
- LHC was the first accelerator to **require** automatic beam-based feedback controller systems
- Different types of **magnets** are used to correct these parameters



- One of these BBF systems was considered for RL tests

Feasibility study on the application of RL on QFB

2/5

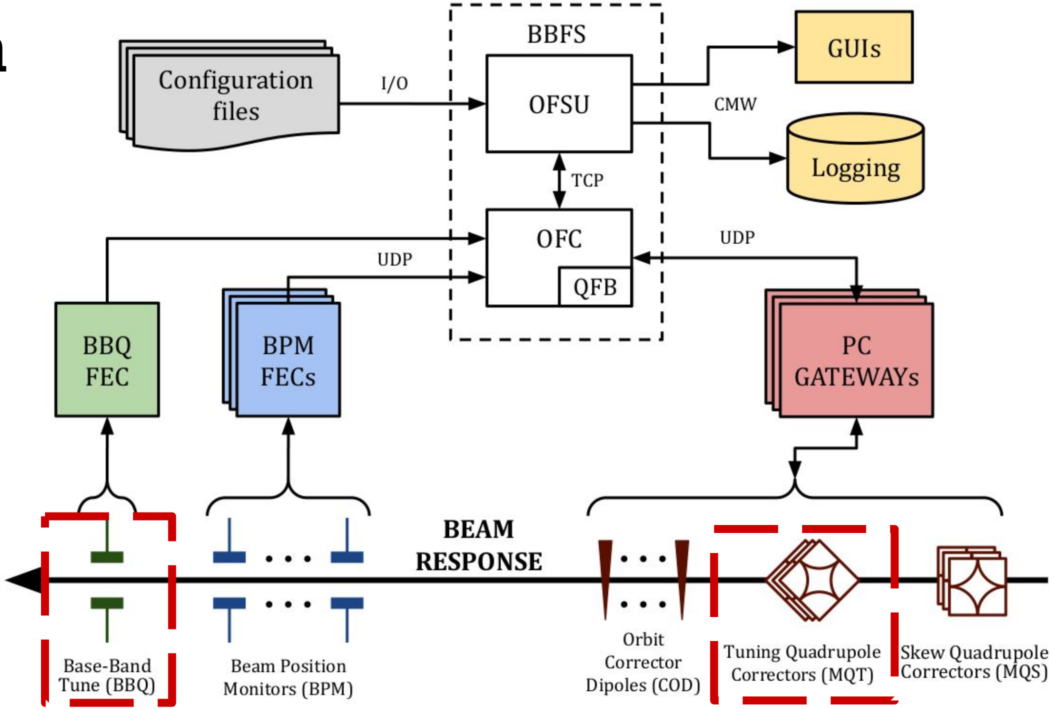
Tune feedback (QFB) system

- QFB system operates on both beams but each beam is corrected independently by 16 quadrupoles
  - Assume no coupling

- Therefore the QFB operation can be simplified into one system with one beam:

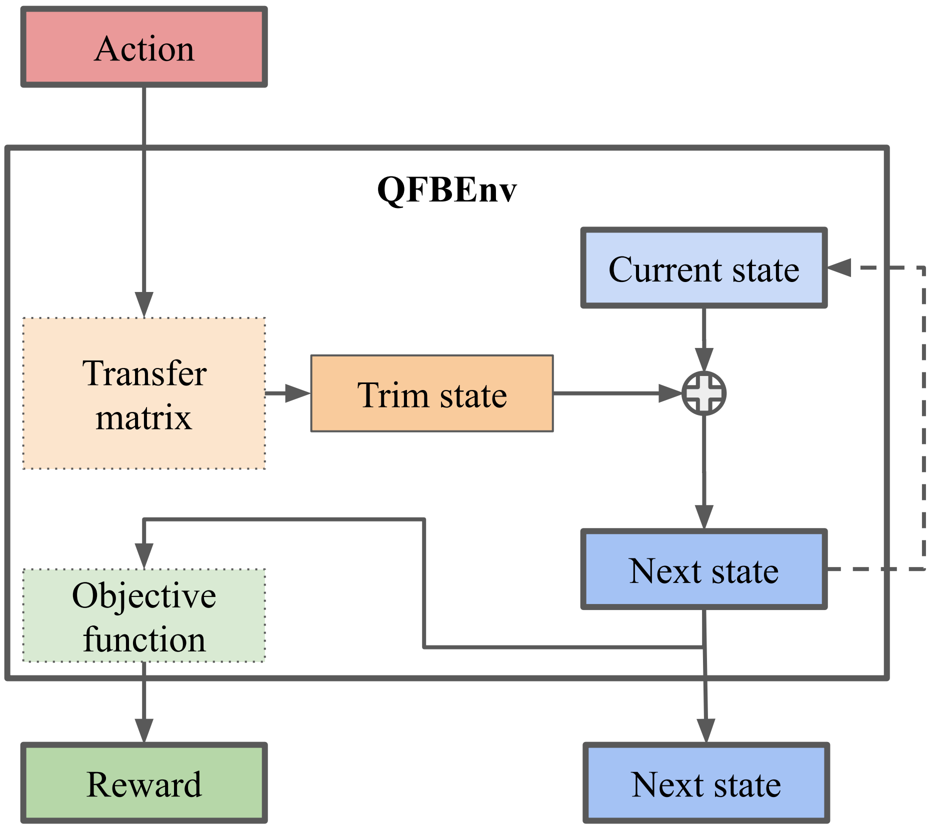
  - Input: **2 continuous state dimensions**:
    - Horizontal tune (H)
    - Vertical tune (V)

  - Output: **16 continuous action dimensions**:
    - Error in magnet deflection (radians)
    - 16 total correcting quadrupoles



Preparing QFB for RL

- Created simulation environment (**QFBEnv**)
  - Using transfer matrix to calculate forward dynamics
  - **Show equation of dynamics**
- **Optimal policy available** for normal operating QFB
  - Inverse of transfer matrix
- State & Actions
  - $\Delta\vec{Q}_{t+1} \doteq \Delta\vec{\sigma}_t \cdot R + \Delta\vec{Q}_t$
  - Q is the state - tune
- **Reward**/Objective function
  - $r \doteq -\sqrt{\frac{1}{M}\left(\sum_{i=1}^{M} s_i^2\right)}$
- Terminal states
  - **Max**(**abs**(**state**)) < **threshold** used in real operation



Training on QFBEnv

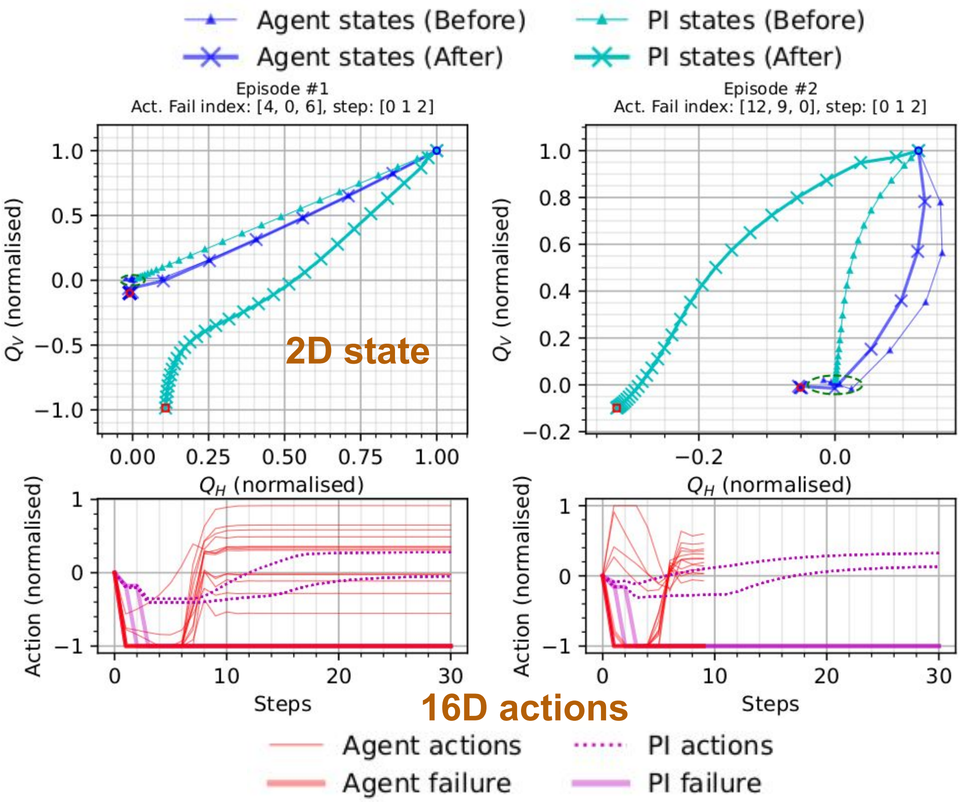
- Trained various types of state-of-the-art RL algorithms
- Used deep networks for policy and value functions
  - Hidden layers: *[64, 64]* each
- Proximal Policy Optimization (PPO) provided best results (on-policy method)
  - Actions decayed to zero
  - **Comparable behaviour to PI controller**
- Normalized Advantage Function with double Q (NAF2) (off-policy method)
  - Also performed well empirically
  - But achieved a sub-optimal policy
  - Hard to tune the hyperparameters

Testing trained agents on QFBEnv

- One example of a testing scenario:

- Magnet malfunction
  - Magnet(s) **chosen at random and turned off** for the duration of the episode
  - **PPO agent outperformed PI controller** showing that it managed to generalise well during training

E.g. 3 magnets failures

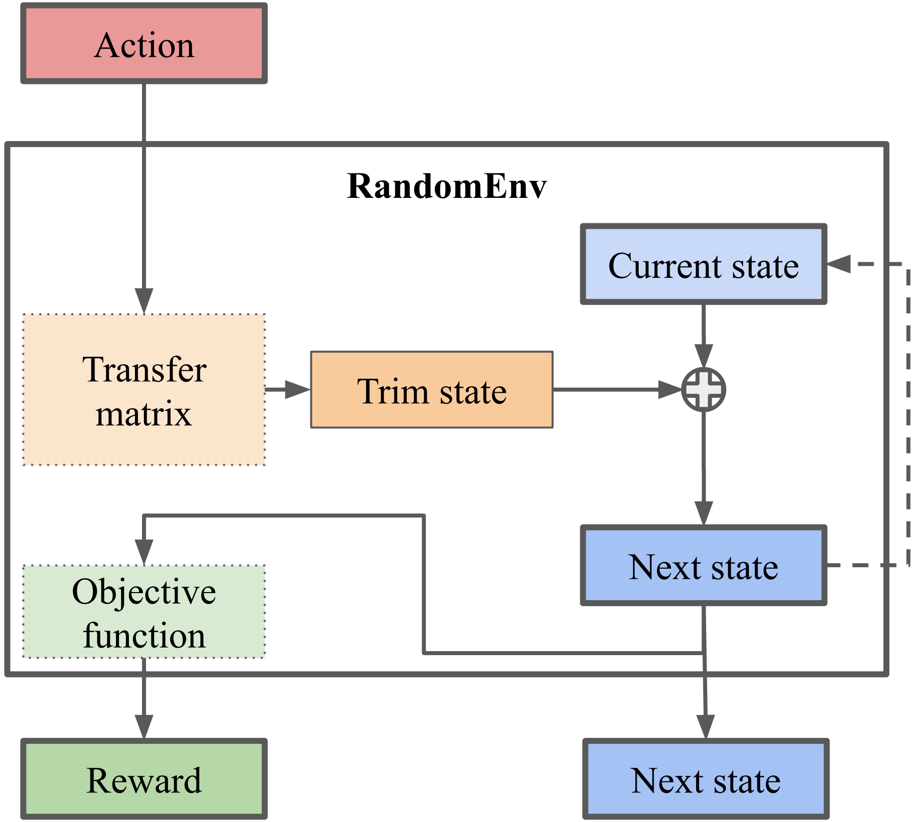


Development & testing of RandomEnv (RE)

3/5

RandomEnv (RE)

- First goal was to develop a general environment
  - **Solving this environment means solving any BBF with similar attributes**
  - Attributes related to state and action spaces
    - *Size*
    - *Discrete or continuous*
- What we want to study is how well we can train RL agents on BBF-type environments
- Dynamics can be *fictitious* but must have certain properties
  - **Invertible**, i.e. an optimal response is possible
  - **Scalable**, i.e. different number of states/actions
  - **Randomly generated** dynamics to allow for multi-seed tests



Testing state-of-the-art RL algorithms

4/5

State-of-the-art tests

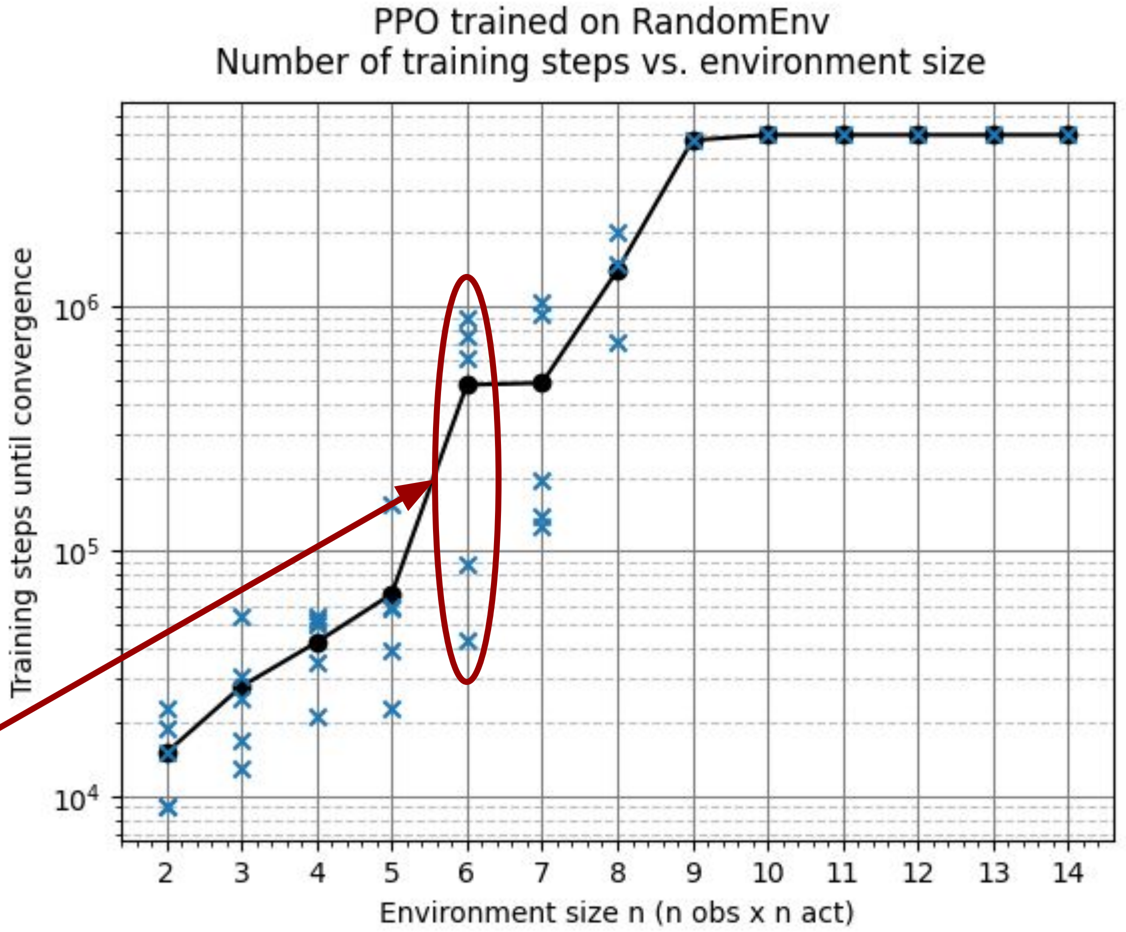
- In QFB study, Proximal Policy Optimization (PPO) provided best results
- Closely related to Trust-Region Policy Optimization (TRPO)
  - Provides **theoretical guarantees** on how policy is optimised
  - *(See extra slides for more PPO vs TRPO info)*
- Studies show both are highly susceptible to code-level optimizations
  - Some RL libraries provide good Python implementations
- PPO and TRPO agents were trained on RE of varying sizes
  - Square dynamics →**Nb. State dimensions = Nb. Action dimensions → M=N**
  - Up to 5M training steps
  - M: 2→15
- Training convergence
  - How long until training produces successful policy
  - Success = Reaching optimal state
    - *Measurement - Reference = Error → 0*

Training PPO on RandomEnv with increasing complexity

- RandomEnv: creation of linear dynamics →Can be set to an Identity matrix
  - Why Identity matrix?
    - *One-to-one linear orthogonal mapping between the state and actions*
    - *Most intuitive when debugging*
    - *When **deep networks** are involved, having "simple" dynamics, does not imply better training*
      - Deep networks have non-linear mappings within and is dependent solely on initial weight initialization
- Let M = Number of state dimensions
- Ler N = Number of action dimensions
- Set M=N for square dynamics - Benchmark for each RL algorithm
- Instantiate 5 separate environment-agent instances for every $M = N, M \in [2, 3, \dots, 14, 15]$
- Set the default hyperparameters to PPO algorithm
  - Stable-baselines3 initial hyper-parameters
- Train PPO agent on 5 x 14 = **70 agents**

PPO as RE gets more complex

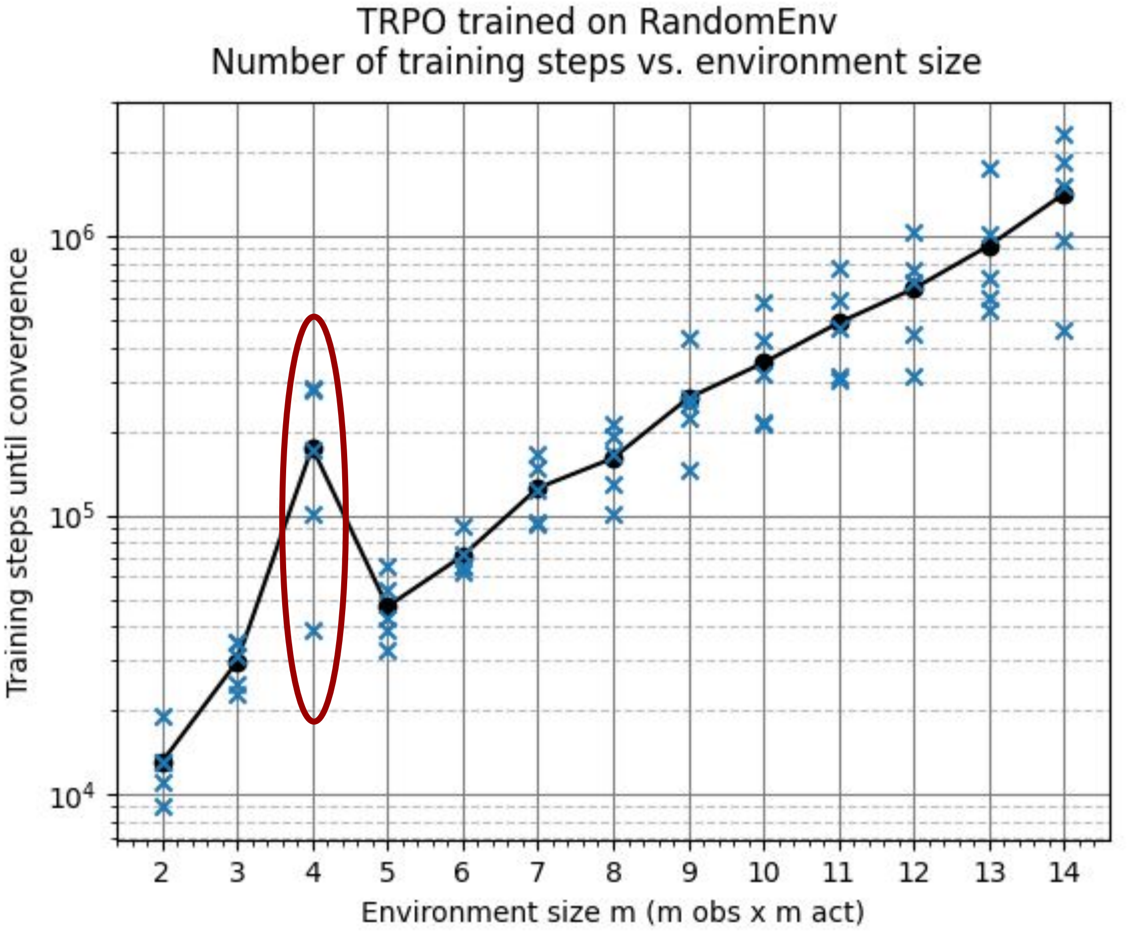
- M: 2⟶ 14
- **Initial network weights** contribute to the differences in each agent of size
- Environment **dynamics fixed** for all env sizes $m \in M$
  - Transfer matrix = I (mxm)
- PPO training times blows up exponentially with env size
- **Sporadic spread** in training times



- RE 6x6
  - Training time varies by 1 order of magnitude
  - 100Hz system, training can take between **15 minutes and 3 hours**!

TRPO trained on RE with identity transfer matrix

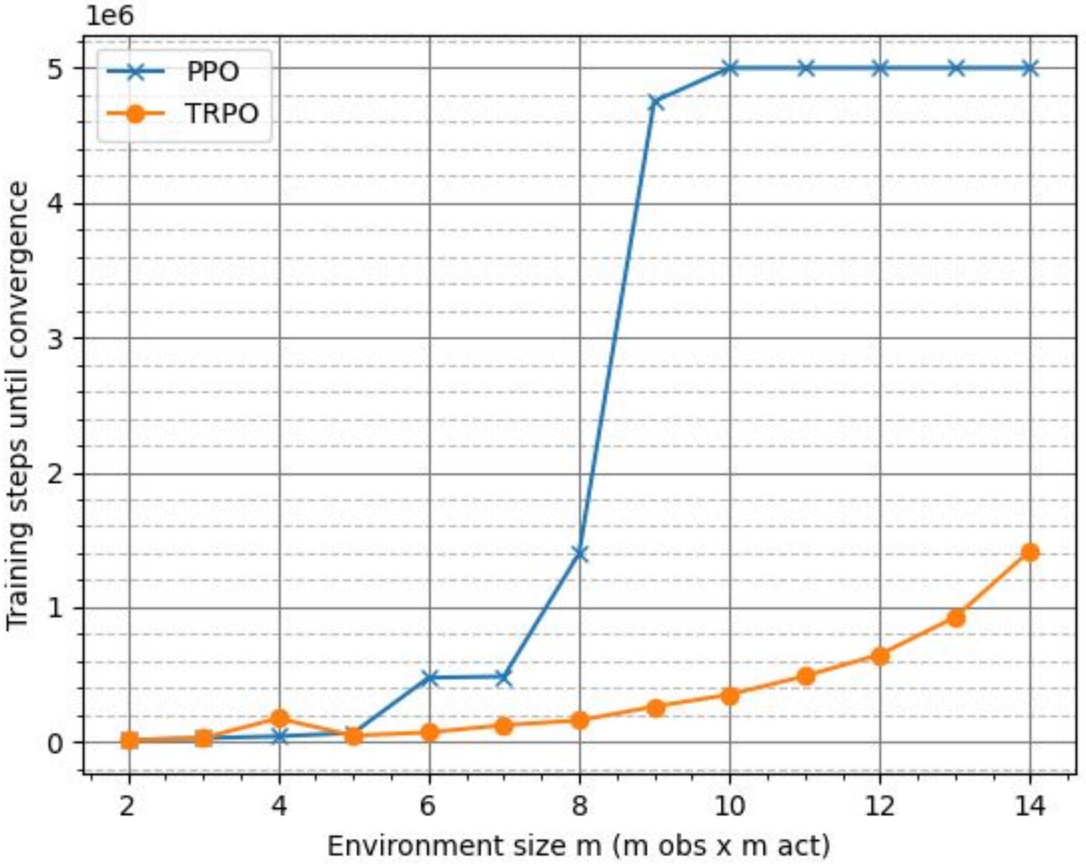
- M: 5→14
- Using same hyperparameters for all runs
- Environment **dynamics fixed** for all env sizes
  - Transfer matrix = I (mxm)
- Trains more predictably than PPO
- Spread in training time increases with larger environments
- Can solve RE 14x14 in approximately 2M training steps



Comparing PPO and TRPO

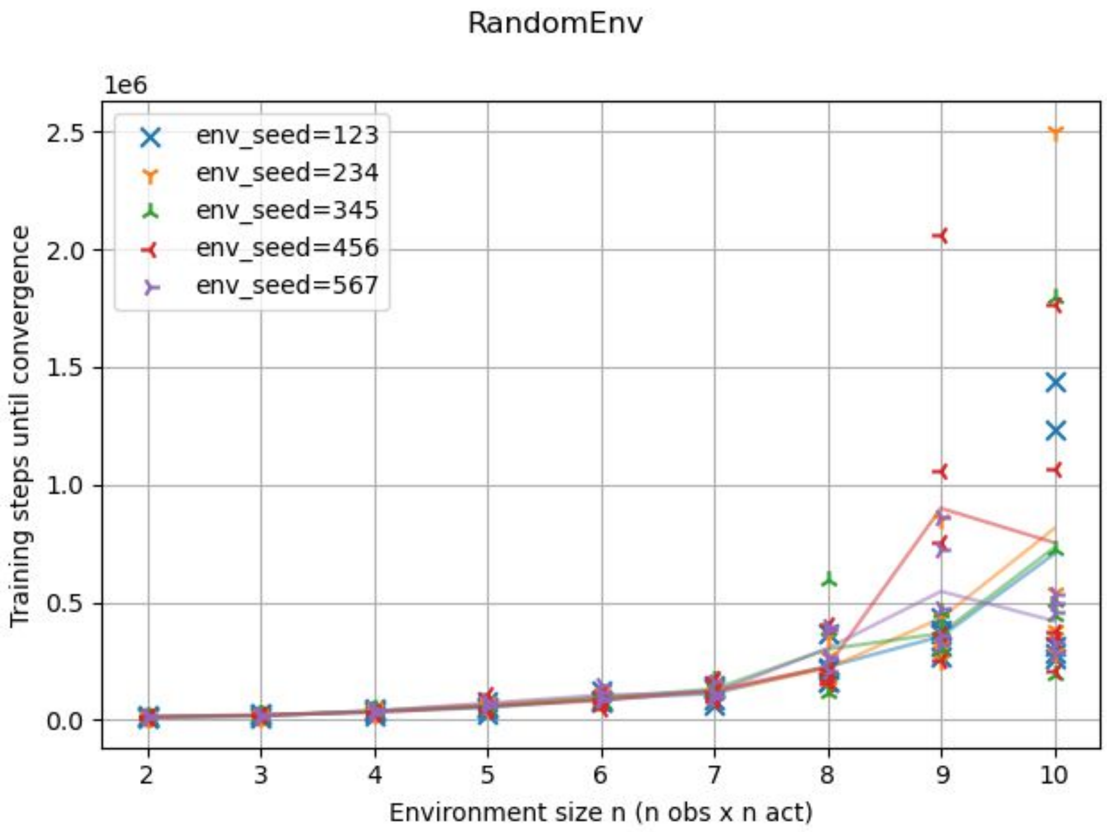
- Goal <u>was not</u> to find the optimal hyper parameters
  - That would be done by a grid-search
  - PPO might have performed better
- How easy is it to use state-of-the-art RL algorithms ‘*out of the box*’?
  - TRPO seems to be a better choice
- **BUT…**
  - Implementation of PPO really matters!
- In fact this issue is studied closely, e.g.:

“Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO”
Logan Engstrom, et al.
ICLR 2020



TRPO trained on different dynamics

- 5 random seeds per environment size
  - Generate 5 different dynamics
- Train 5 TRPO agents, per environment
  - Networks intialised with different seeds
- Larger environments, training time spread can increase significantly
  - *Unlucky* dynamics
  - Might be fixed with proper hyper parameter tuning, <u>per environment</u>



Key takeaways from these tests

- Results show that expected training time until convergence to an optimal policy increases exponentially with environment size
    - But **training becomes unstable** in large environments
    - TRPO is reasonably robust to hyperparameter tuning, but suffers in large environments
- State-of-the-art RL algorithms might not be suitable for online BBF systems
    - Too **sample inefficient**
    - Difficult to use and tune
    - Highly susceptible to code-level design choices
- Do we really need deep networks in BBF systems?

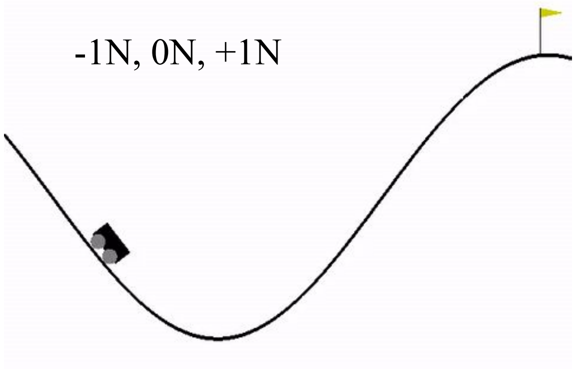
Simplification of RE into discrete actions

5/5

Back to basics

- RL without deep networks?
  - Tabular methods
  - Linear function approximation
- **Strong theoretical convergence guarantees only exist for tabular RL**
  - Can only be used with discrete state-action space
  - Finite number of state-action pairs possible
  - Very limited environment size
- RL with **linear approximation** also has some guarantees
  - Hand-designed features
  - Continuous states possible
- Using simpler agents might be useful for real operation in BBF systems
  - Deterministic training time/number of training interactions
  - Monotonic policy improvement
  - Safe policies
  - Meaningful exploration

RE with Discrete Actions (REDA)



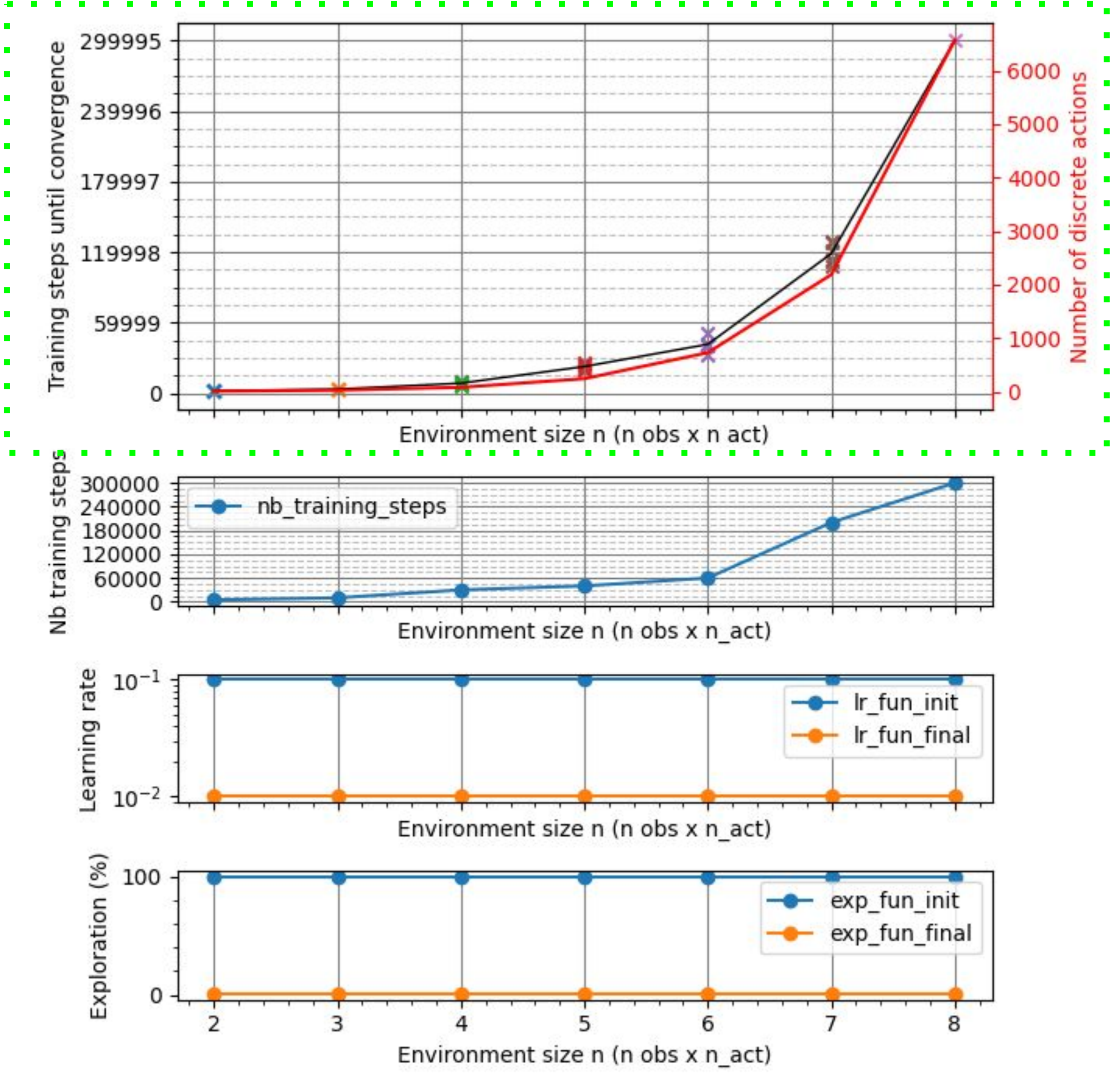
- RE was converted to use **discrete actions** (REDA)
  - Same action strategy as OpenAI Gym MountainCar environment
  - Each action dimension can be one of three values **[-ε,0,+ε]**
  - **ε** → Tuned such that at least an episode has more than 1 step
    - *i.e. one-step solutions are made less likely*
    - *Enforcing a precision with which the policy can change the state*
    - *Makes the MDP much simpler to solve*
- We can use REDA to analyse **fundamental RL ideas**
  - Episodic vs infinite-horizon environment
  - Epsilon-greedy vs Boltzmann policies
  - Regret upper confidence bounds
- Action space can be represented by a **finite set of choices**. We have some options:
  - **Policy type 1: All action permutations** →Cardinality(A) = **3^N**
    - *E.g.* ***REDA2x2: card({{-ε,-ε}, {-ε,0}, {-ε,+ε}, {0,-ε}, … , {+ε,0}, {+ε,+ε}}) =27 possible actions***
  - **Policy type 2: Canonical vectors + do nothing action** →Cardinality(A) = **2*N + 1**
    - *E.g.* ***REDA2x2: card({{-ε,0},{+ε,0},{0,-ε},{0,+ε},{0,0}}) = 5 possible actions***
- The simplest stable RL algorithm using temporal difference learning
  - State-Action-Reward-State-Action (SARSA)
- SARSA + REDA tests follow

Linear RL on REDA

- REDA has continuous states and discrete actions
- Discrete actions either **policy type 1** (all permutations) or **policy type 2** (canonical actions)
- What does **linear** mean?
  - V and Q functions are a linear in the **features of the state**
  - In deep networks, **last layer is linear** on output of last hidden layer
- What are features of the state?
  - Recall:
    - *Policy function maps state to action*
    - *When applying deep networks for the policy of an agent in Deep RL:*
      - All layers before the last linear layer represent the features of the state
- If we have a feature function: $\Phi(s) \in R^d$
  - Value can be estimated with: $v_\pi(s) = \vec{w}^\top \cdot \Phi(s)$
  - Greedy policy: $\pi(s) \doteq \arg\max_a q_\pi(s,a) = \arg\max_a w_a^\top \cdot \Phi(s,a)$

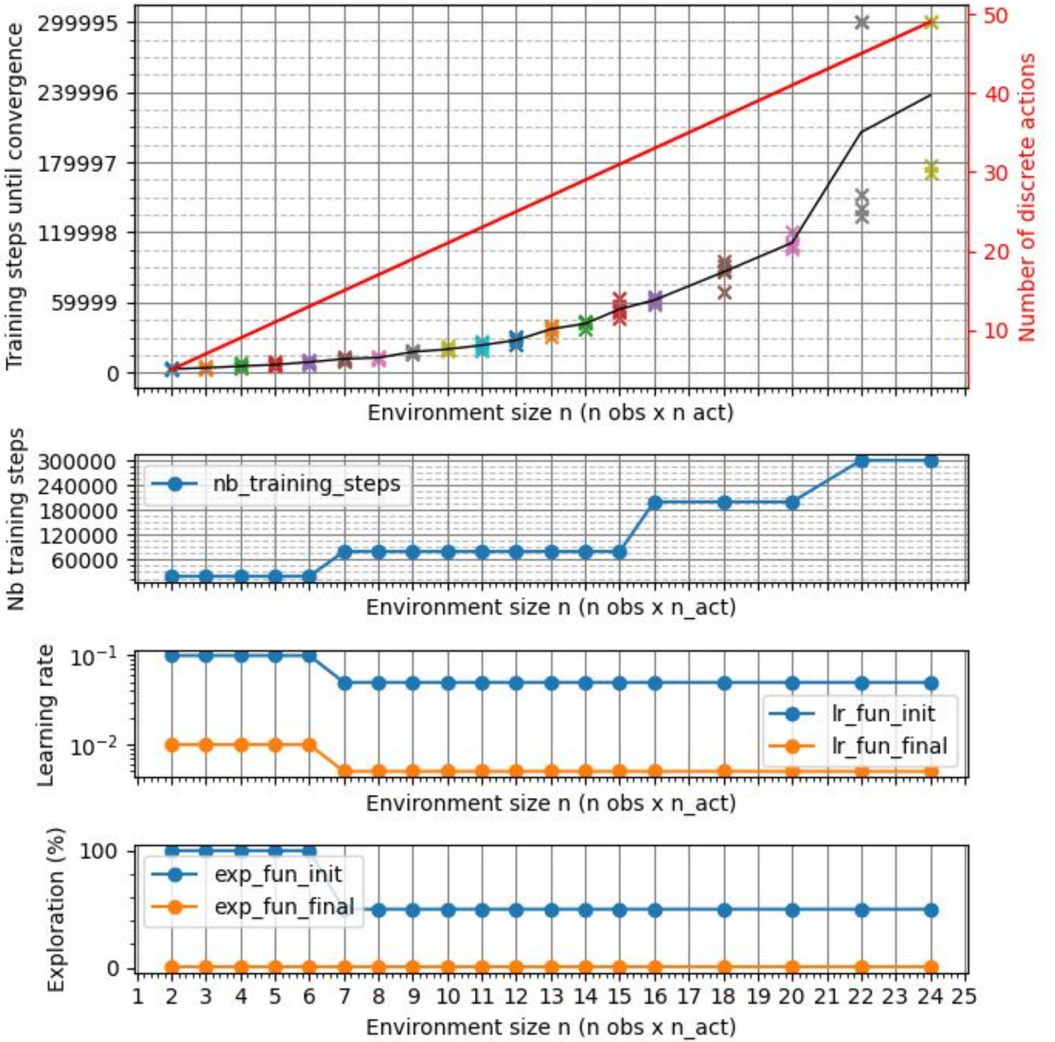
REDA with all action permutations set

- The training times match closely to the training of PPO and TRPO
    - Order of 10^5 steps
- Environments larger than 7x7 suffer from bad sample efficiency
- REDA 7x7 = 2187 actions
- REDA 8x8 = 6561 actions



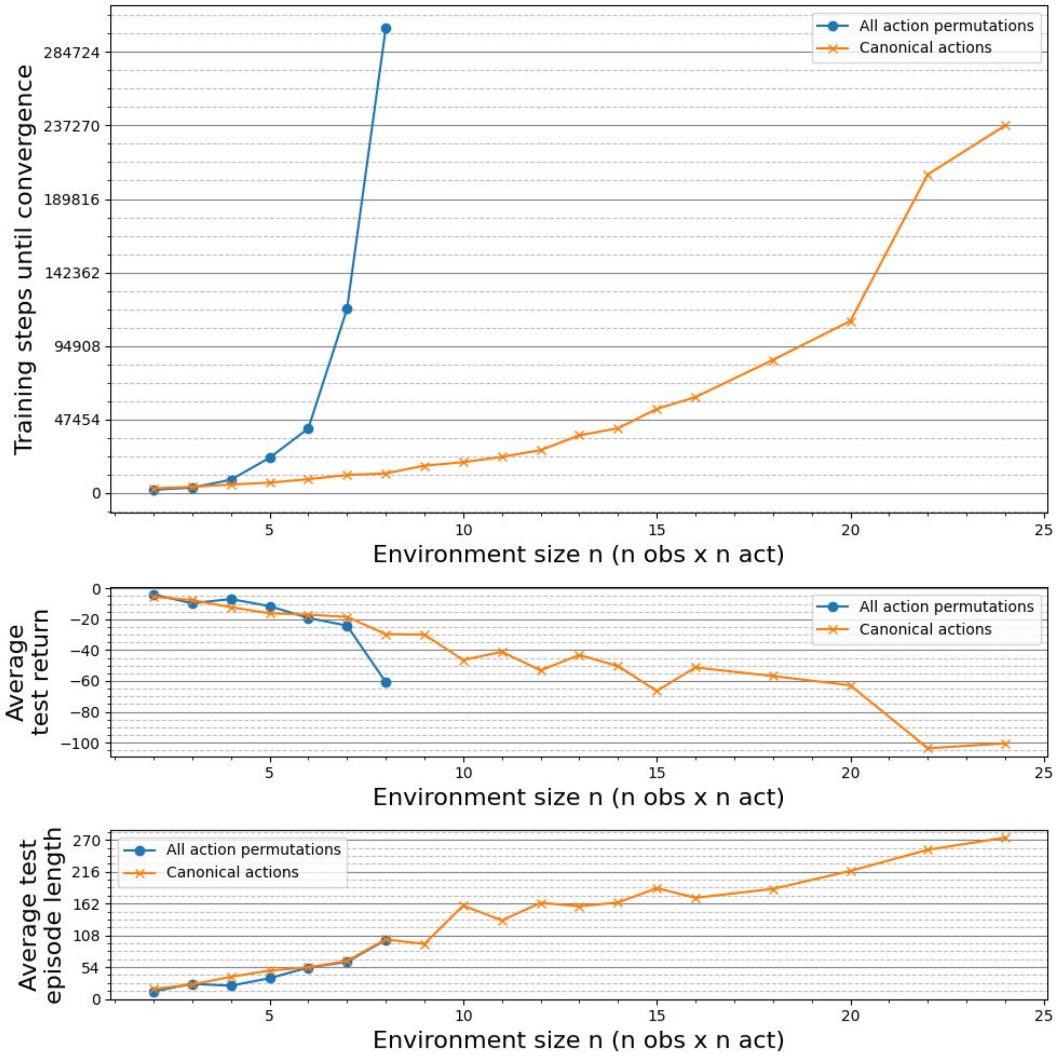
REDA with canonical action set

- Learning rate might have to be reduced with large environments since weights might explode
- Learning and exploration decay was linear and should be adjusted for larger environments
- REDA 7x7 →REDA 15x15
  - Same hyperparameters



Comparing all action permutations vs canonical action set

- Canonical action set more sample efficient
- Optimal policy with canonical action set has similar episode lengths to policy with all action permutations
- So we can use REDA with very low number of actions
- Note episodes become longer with larger env size



Comparing the two REDA approaches

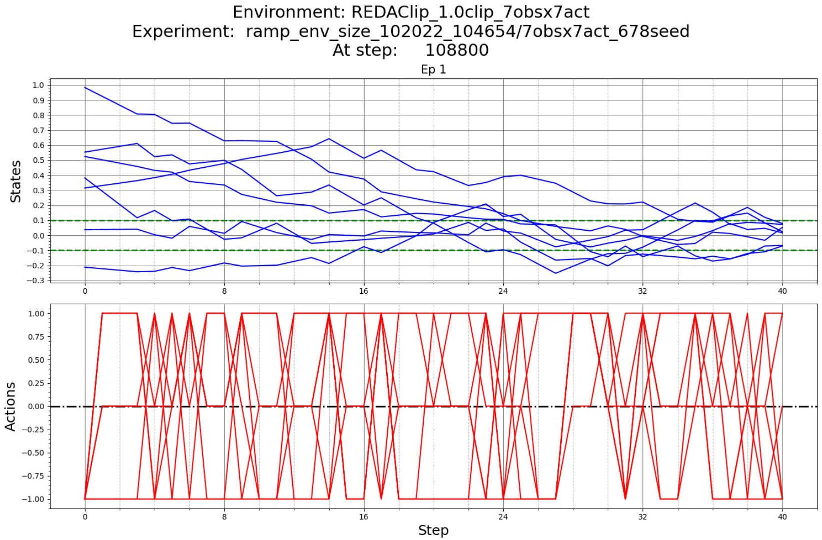
- By playing some episodes we can look at the two different type of policies
  - Both policies reach optimal states after approx. the same number of steps
- Showing agents trained on **REDA 7x7**
- By using canonical action set: ~10x faster to train

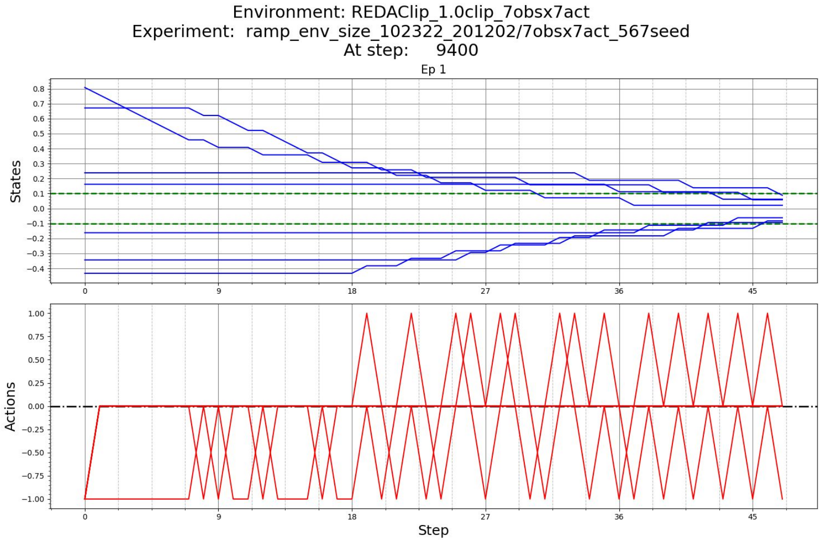
ALL ACTION PERMUTATIONS @ TRAINING STEP 108800

CANONICAL ACTION SET @ TRAINING STEP 9400

CONTINUOUS STATE

DISCRETE ACTION





Time steps in one episode of REDA 7obsX7act

Summary and future work

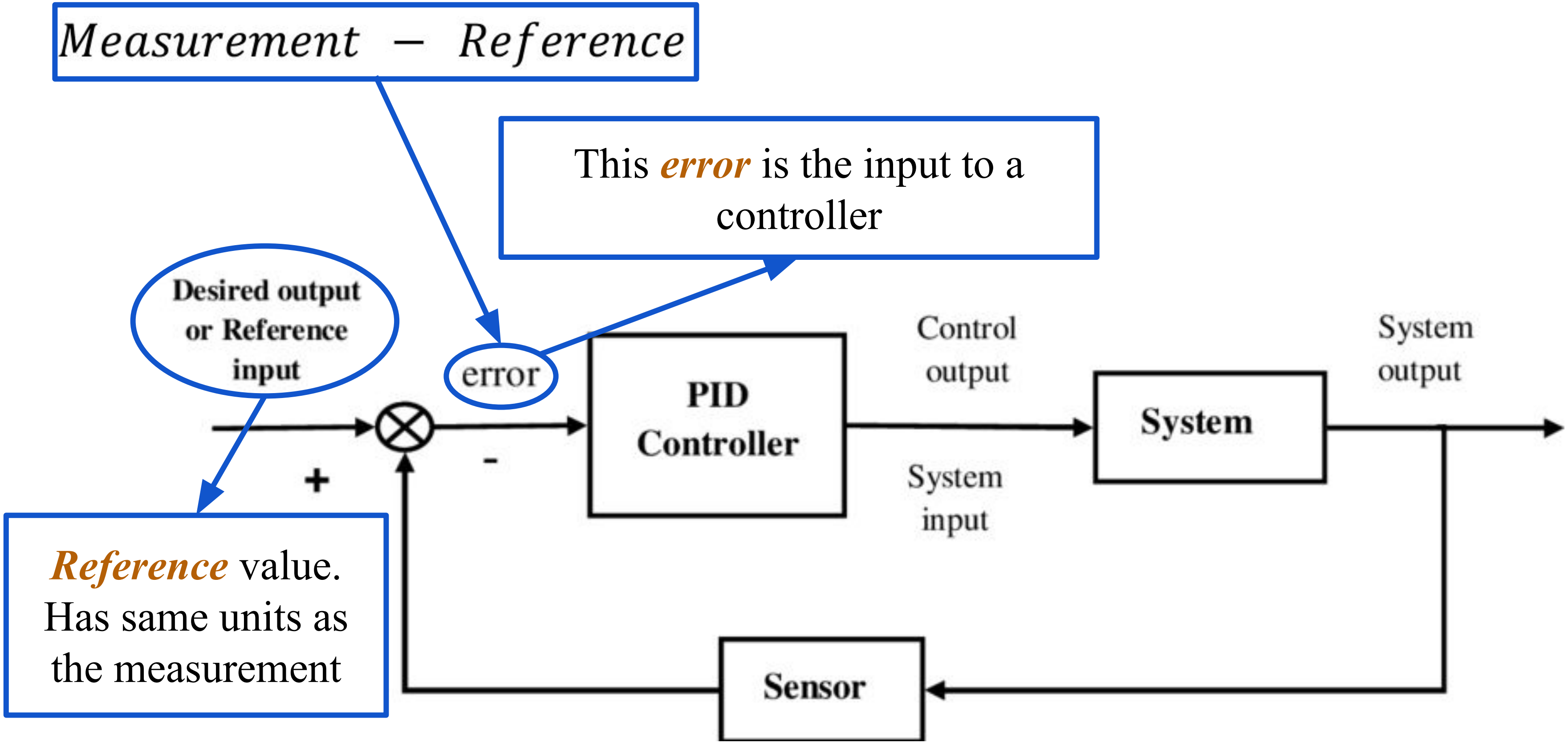
- Started with feasibility study of RL on Tune feedback system
- Created a generalised environment which matches the operation of a BBF system
- Tested performance of state-of-the-art deep RL algorithms on REs of different complexities
    - Found to be unreliable to train
    - Difficult to make them work in real operation
- Since guarantees only exist only for tabular RL and linear function approximation
    - We use SARSA to train on REDAs of different complexities
- For small enough problems (~< REDA 25x25), you can achieve good sample efficiency compared to state-of-the-art
- Any suggestions are welcome!

Thank you!

Questions?

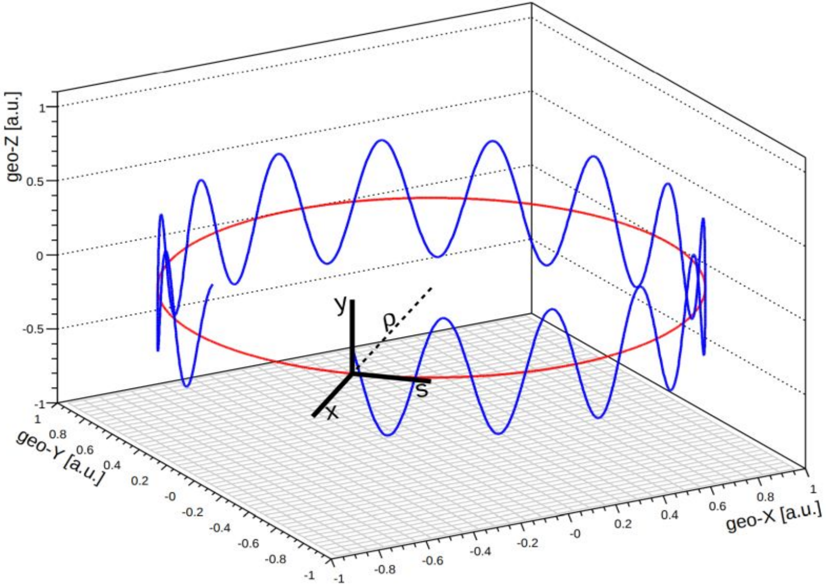
Extra slides

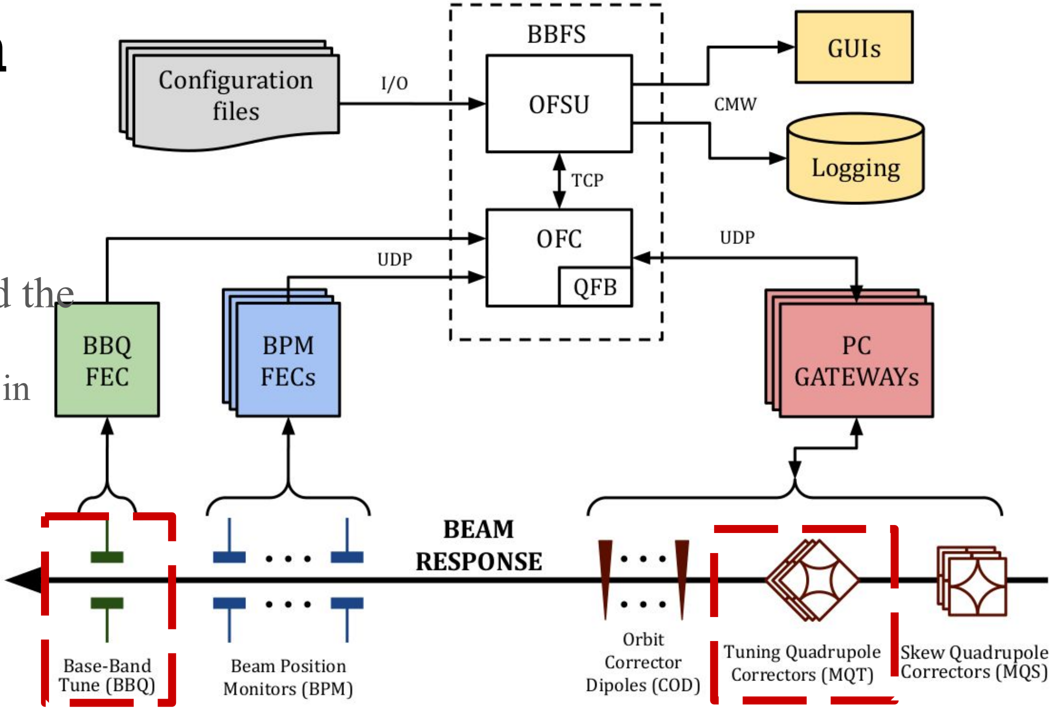
What are feedback systems: *E.g. PID Controller*



Tune feedback (QFB) system

- **Tune** - related to **number of transverse oscillations** of a particle per turn in the accelerator
- Tune is measured for the horizontal (x) and the vertical (y) plane respectively
  - 2 measurements of tune per beam (2 beams in LHC)





- The Large Hadron Collider (LHC) BBF system by LHC Long Shutdown 2
  - Highlighted systems are part of QFB

DeepREL

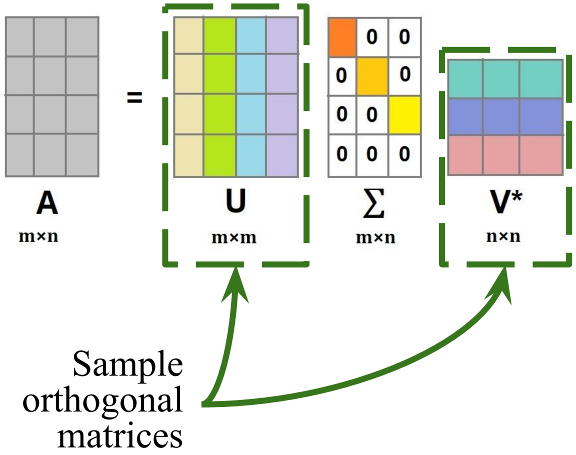
- Continue looking at the performance of RL agents on BBF controllers
- All beam-based feedback systems rely on 2D matrix
    - Linear dynamics model
    - Sometimes number of eigenvalues is controlled to globalise correction - localised correction results in dramatic corrections to magnets
- Some BBF systems in the LHC have 1000s of states and 100s of actions
    - E.g. Orbit Feedback (OFB) system controlling position of beam in the beam pipe
- Can state-of-the-art RL handle such a system with linear dynamics?

RL motivation

- Started as an **explorative study** on Tune feedback system
- Finding **optimal response** in an unknown environment
- Preliminary offline tests show potential improvement of RL agent compared to standard controller
- This work looks at expanding the use of RL to larger systems

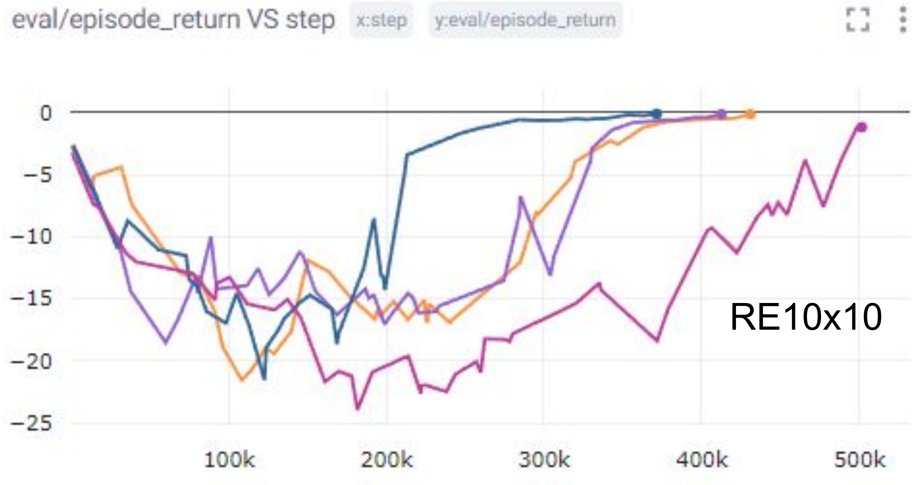
RE dynamics

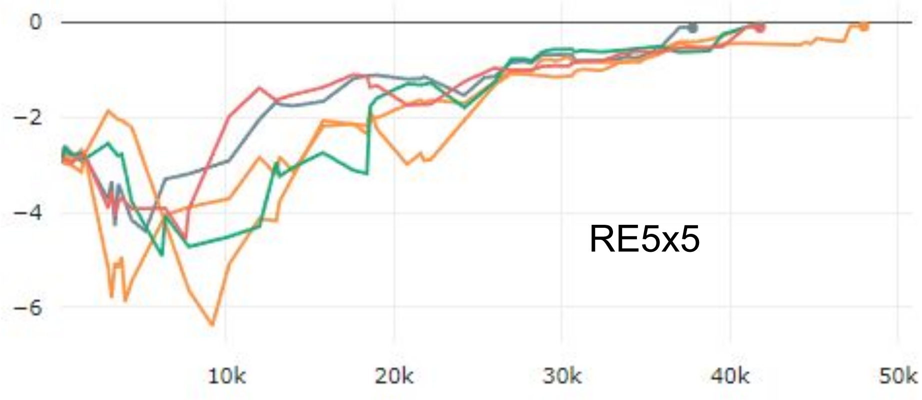
- When modelling a physical system **linearly** we obtain a dynamics matrix
- We can use Singular Value Decomposition (**SVD**) to find inverse matrix
    - Pseudo-inverse if number of states != number of actions: $A \cdot A^{-1} = I$
    - Gives control on localisation/globalisation of corrections
    - Used in LHC BBFs
- Linear model is **factorised**
    - $A = U\Sigma V^{\top}$, where U & V are unitary matrices
    - I.e. inverse becomes trivial: $A^{-1} = V\Sigma^{-1}U^{\top}$
    - Factored matrices are real-valued; i.e. U & V are orthogonal matrices
- We can **sample** new linear systems
    - Invertible dynamics
    - Easy to create different size environments
    - Change random seed to create different environments



Training plots from TRPO

- Showing average return obtained greedily during evalutation





Different types of RL algorithms

- Trust region algorithms
  - Updates to the parameters occur **within local neighbourhood** to ensure smooth policy transitions
- Trust-Region Policy Optimization (TRPO)
  - Constrains the **action conditional probability distribution** from the policy
  - **Kullbeck-Leibler divergence constraint** between policy updates
  - Uses an approximation of the lower bound expected return from a policy
- Proximal Policy Optimization (PPO)
  - Attempts to **simplify** TRPO
  - **Constrains policy parameter space** between updates
  - *With deep networks there may be non-linear dependencies between policy parameters and outputs*
- Normalised Advantage Functions (NAF)
  - Analytical formulation of Q-Function
  - Off-Policy algorithm - can use experience obtained from an unknown policy
  - Can be combined with advances made in deep off-policy RL algorithms;
    e.g. Twin-Delayed Deep Deterministic policy gradient (TD3)
  - Promising results from QFB study

Different types of RL algorithms

- Deterministic policy gradient theorem
- Deep Deterministic Policy Gradient (DDPG)
  - Deterministic Policy Gradient theorem: Equivalent to stochastic policy gradient as the noise goes to zero
- Twin-Delayed Deep Deterministic policy gradient (TD3)
  - Advances to DDPG algorithm
  - Train two Q networks; choose the smallest; minimise overestimation bias
  - Delayed target Q network updates
  - Target smoothing action noise
- Soft Actor-Critic (SAC)
  - Adding an entropy regularisation term to the PG loss
  - Maximise trade-off between exploration and exploitation
  - Off-policy algorithm

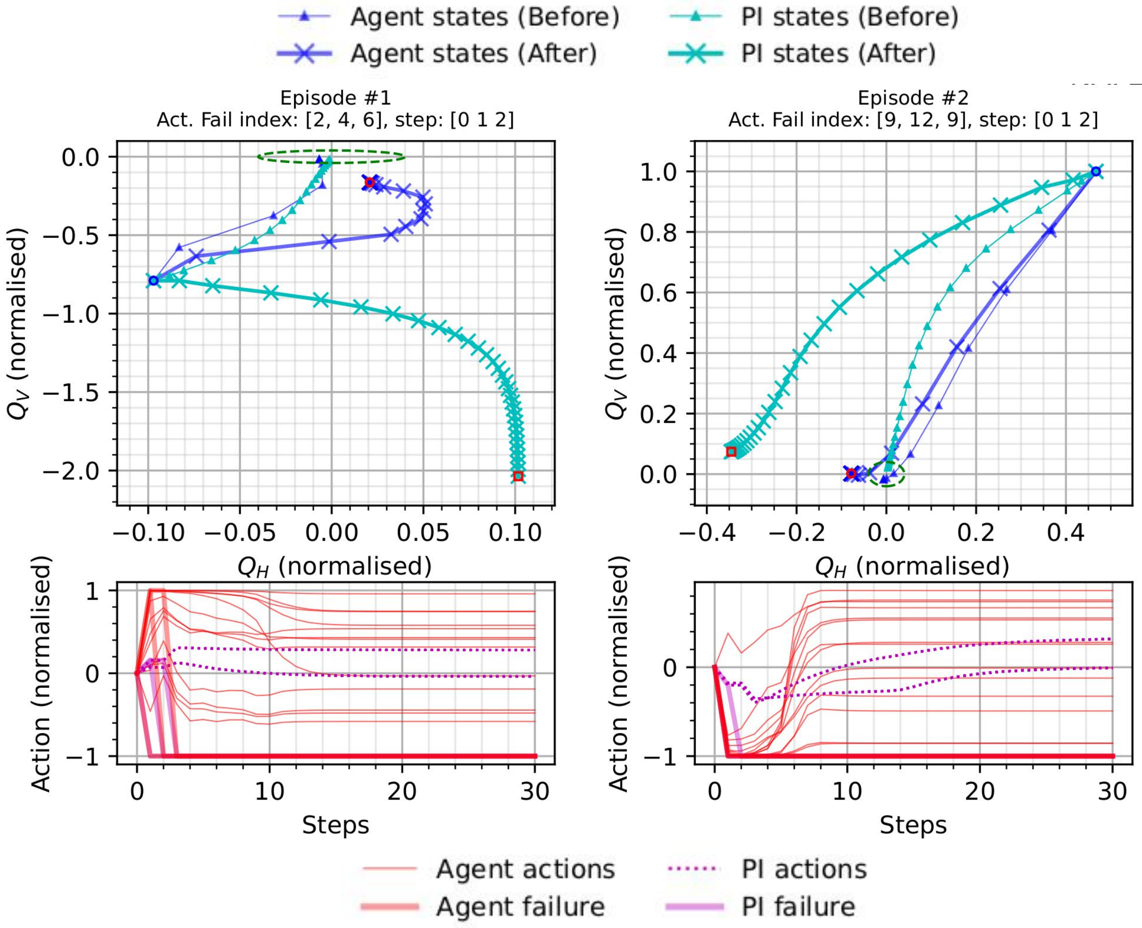
Other tests on QFBEnv

- Normal action noise with varying magnitude:
  - A sweep from 0% → 50% action noise
  - Generated test episodes with greedy policy at 10%, 25% and 50% action noise
- Applying systematic perturbations to the tune
  - 50Hz noise harmonics were messing with the tune estimation
  - Sporadic estimates mislabeled to occur at these harmonics
  - Using worst-case realistic perturbations observed in real tune estimates
  - PPO, NAF2 outperform the PI controller again
    - *Deep RL trained agents manage to keep the state from drifiting*

NAF2 Best Policy Evaluation

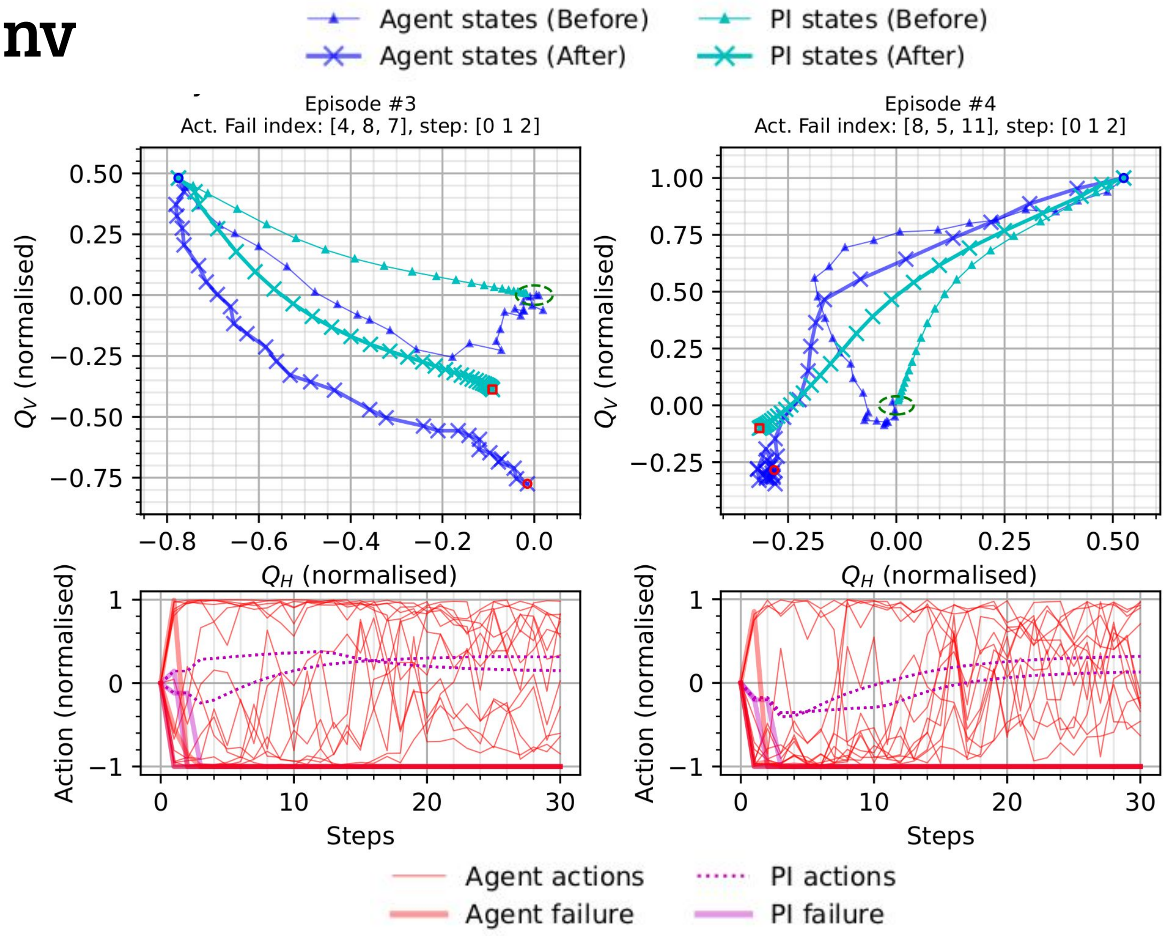
3 actuator failures

- Off-policy algorithm
- PPO equivalent - **good sample efficiency**
- Undesirable performance compared to optimal policy
  - Optimal - derived from transition matrix
  - Actions do not decay proportionally to state: $||A|| \propto ||S||$
- Needs **well-tuned hyper parameters**
- Best policy trained by NAF algorithm is **not the optimal policy**



Model-based RL on QFBEnv

- Training of an uncertainty aware model with a crude approximation through network ensembles
- To the right: AE-DYNA with three actuator failures
- Interesting observations:
  - Model-Based RL (MBRL) agents fail similarly to the optimal controller, indicating strong dependence of policy on model
  - Remember: Model-Free RL (MFRL) agents on rely on estimation of expected return



How do you implement linear RL?

- Find a suitable feature representation of the state. This is a good time to introduce prior knowledge about the environment
  - E.g. I know that REDA has an objective proportional to the root mean square (RMS) objective
    Therefore a good feature selection for REDA_MxN would be:

$$\Phi(s_t) \doteq \left(s_t,\ RMS(s)\right)^{\top}$$

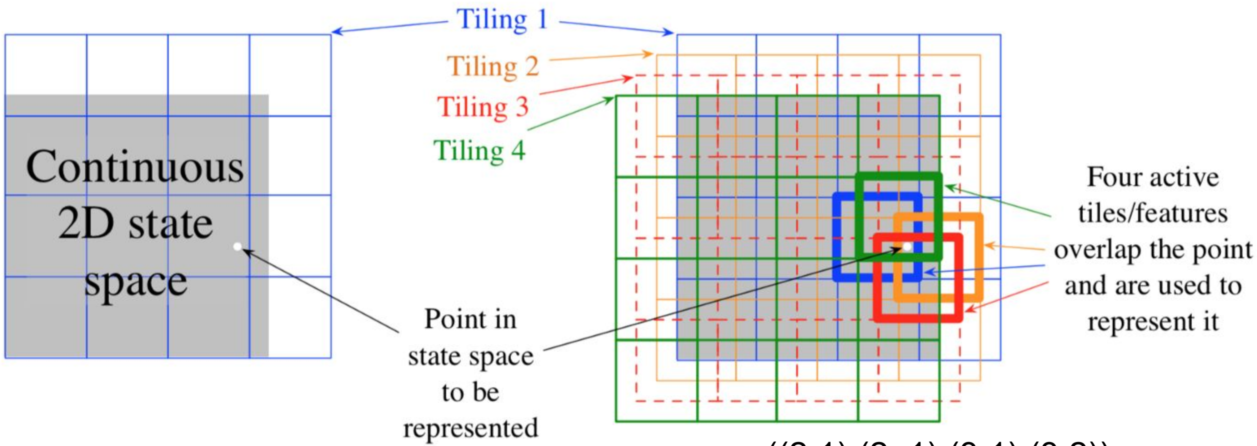
  - Therefore we would only require |S| + 1 number of weights for this feature selection:

$$|W| \doteq |S| + 1$$

RE as an MDP

- Markov Decision Processes (MDP) are used to formulate RL problems
  - Discrete time, stochastic control process
- How do we design environment?
  - This is critical to what we want to achieve
- Use case 1: BBF is turned on when needed
  - Episodic MDP
  - You can omit discount factor for fixed time episodes
- Use case 2: BBF is on continuously
  - Infinite horizon MDP
  - Discount factor required
  - Numerical problems when exploration is unbounded
- Exploration in a real system needs to be bounded otherwise we risk breaking the machine
  - E.g. Orbit of the beam cannot exist outside the beam pipe!
- Initial states need to be realistic
  - E.g. if the state is 2D and we have 1 action, all the states have to be reachable with the dynamics

Tabular RL on REDA

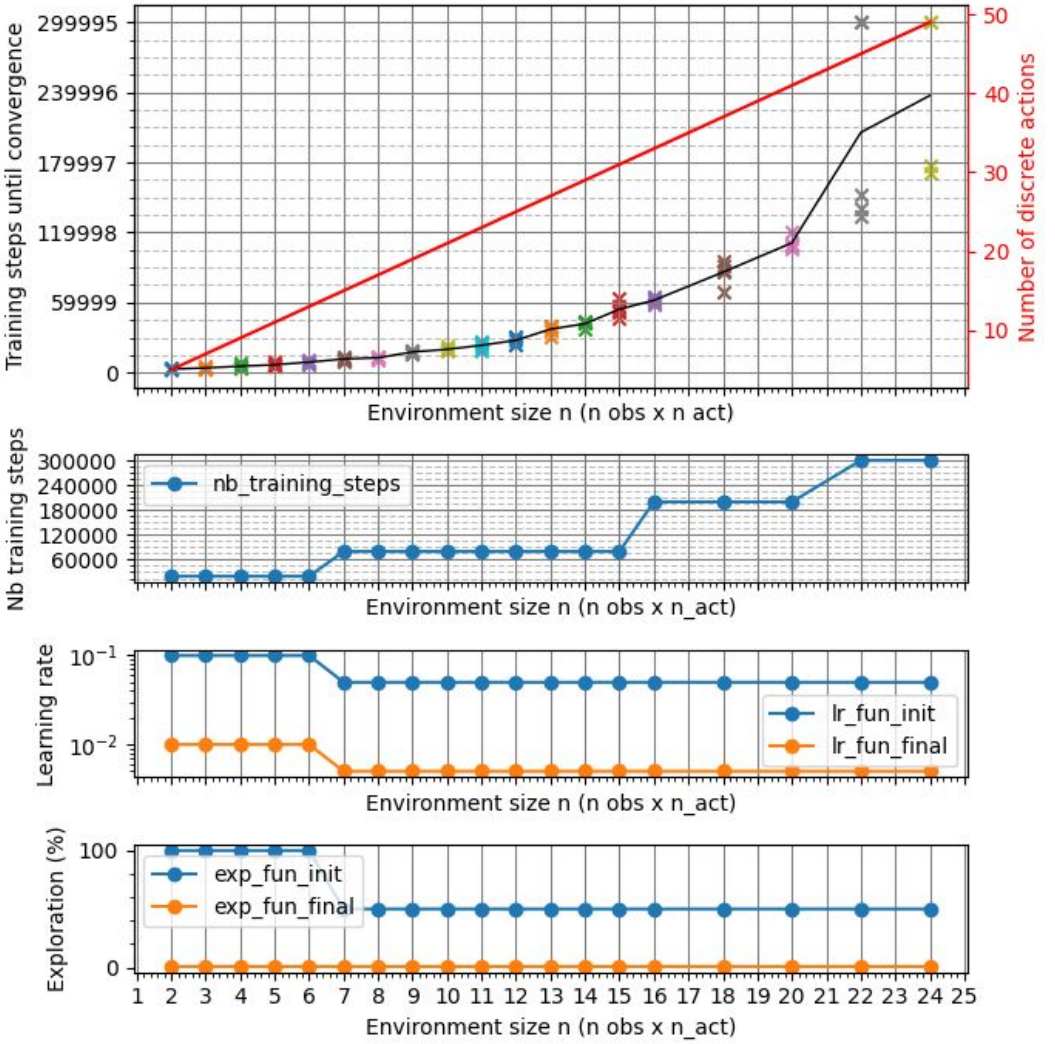


((2,1),(2, 1),(3,1),(3,2))

- To use tabular RL methods we also need:
    - Discrete representation of the state
- Tile-coding is a good candidate:
    - Non-parametric function approximator
- Memory complexity blows up quickly
    - **O(NB_TILINGS x NB_BINS ^ M x |A|)**
- Limited to very small environments (< REDA_5x5)

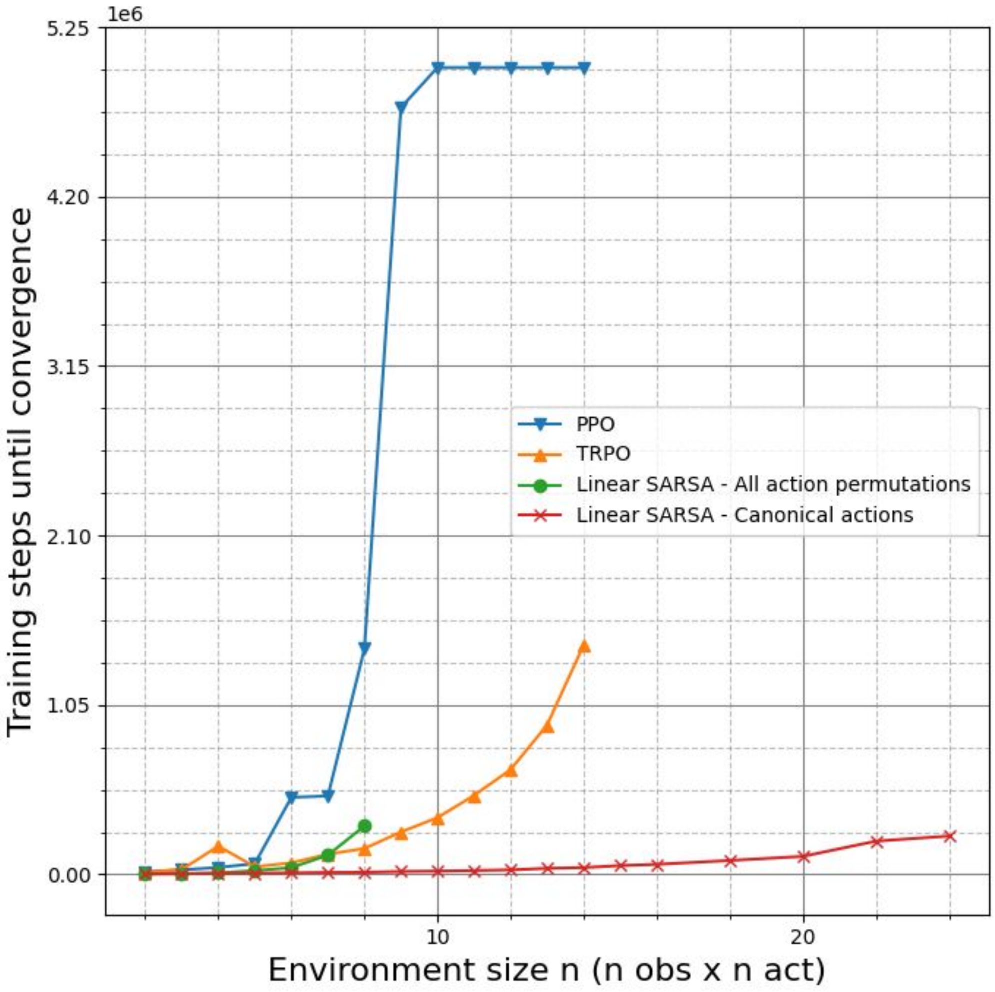
REDA with canonical action set

- Large environments (>REDA_10x10) require more hyperparameter tuning

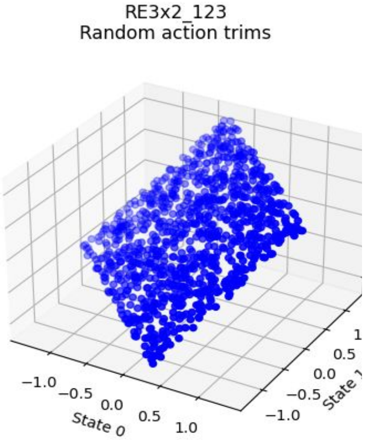


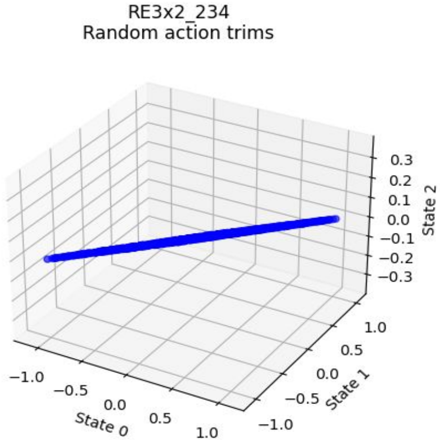
Comparing Deep RL & Linear RL

- We can improve sample efficiency on large environments
  - Using SARSA
  - Hand-designed features
  - Limiting number of actions

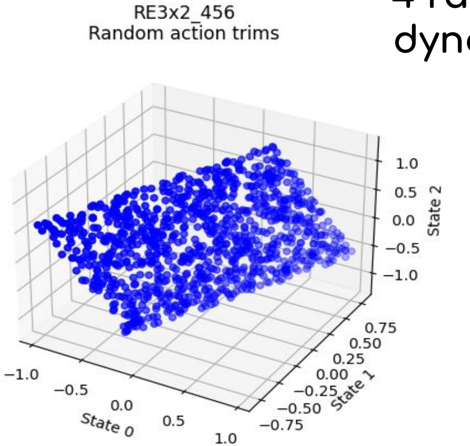


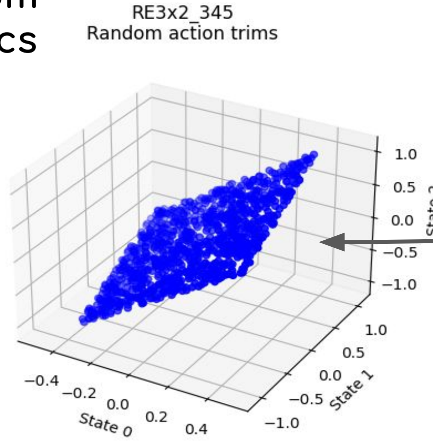
Initialising RE MXN where M < N





4 random
dynamics





- Having less actions than observations results in unreachable states
- The planes show the the direction of the possible trims using the forward dynamics only

$$\begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ r_{31} & r_{32} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} r_{11}a_1 & r_{12}a_2 \\ r_{21}a_1 & r_{22}a_2 \\ r_{31}a_1 & r_{32}a_2 \end{bmatrix} \triangleq \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

Resultant plane obtained by Monte Carlo simulation containing all plausible initial states for each resp. env.