Accelerator Tuning with Deep Reinforcement Learning

David Yuchen Wang [1,2]
david311@student.ubc.ca

Harpriya Bagri[1]
harpriya.bagri@gmail.com

Calum Macdonald [1,3]
calum.macdonald@generalfusion.com

Spencer Kiy [1]
spencerkiy@triumf.ca

Paul Jung [1]
pjung@triumf.ca

Olivier Shelbaya [1]
oshelb@triumf.ca

Thomas Planche[1]
tplanche@triumf.ca

Wojciech Fedorko[1]
wfedorko@triumf.ca

Rick Baartman[1]
baartman@triumf.ca

Oliver Kester[1]
okester@triumf.ca

[1]TRIUMF, [2]The University of British Columbia, [3]General Fusion

Abstract

Particle accelerators require routine tuning during operation and when new isotope species are introduced. This is a complex process requiring many hours from experienced operators. The difficult control aspect of this problem is challenging for traditional approaches, but offers to be a promising candidate for reinforcement learning. We aim to develop an automated tuning procedure for the accelerators at TRIUMF, starting with the Off-Line Ion Source (OLIS) portion of the Isotope Separator and Accelerator (ISAC) facility. In this early stage of research, we show that the method of Recurrent Deep Deterministic Policy Gradients (RDPG) is successful in learning accelerator tuning procedures for a simple simulated environment representing the OLIS section.

1 Introduction

The TRIUMF accelerator complex [8] drives Canadian accelerator-based science programs enabling research in diverse fields ranging from nuclear physics to material sciences, life sciences and nuclear medicine. The TRIUMF cyclotron is the centerpiece of the complex, which, as the most powerful driver of an isotope separation on-line (ISOL) target in the world, gives the Isotope Separator and Accelerator (ISAC) [18] facility a worldwide lead in the production of many rare isotope beams. The short-lived radioactive isotopes are produced by impinging the primary proton beam from the cyclotron onto a target. Rare isotopes produced in the target are ionized to form an ion beam that can be separated by mass and then guided to the experimental facilities of the ISAC complex. The off-line ion source (OLIS) [11], [12] provides stable ions ranging from Helium to Xenon for accelerator tuning, commissioning, and stable beam experiments.

The procedure of tuning such accelerator beamlines is time-consuming and relies heavily on the intuition and experience of the operators. The uncertainties involved in tuning are in part due to unknown misalignments in the beamline components and a low level of observability of beam characteristics. The tuning of a section such as OLIS can take up to 4 hours, during which an operator

Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS 2021).

needs to constantly monitor various diagnostics and adjust the steering and focusing components in order to counteract the effects of those misalignments. We hope to improve the speed and efficiency of such tuning procedures by developing a reinforcement learning agent which can observe beam diagnostics and predict the optimal tuning parameters. OLIS is selected as a starting point for this endeavour due to its low operating current, and use of non-radioactive isotopes.

Since the full internal state of an operational particle beamline cannot be measured directly and must be estimated based on measurements, such an environment can be modeled as a partially observable Markov decision process (POMDP). Previous work [2] has shown that neural networks are effective in learning physics related tasks, and in particular reinforcement learning can be successfully used for beamline tuning [9]. Recurrent deep deterministic policy gradients (RDPG) [4] is an effective method for solving control tasks for POMDPs and has been successful across a wide domain of control tasks including driving [14], locomotion control [15], and frequency control [17]. We utilize the RDPG architecture and test its effectiveness on predicting tunes for simulated beamline environments. Further development of this work will hopefully lead to automation of beamline tuning in the real world. While this may mitigate the need for human operators, we do not think it will fully replace the efforts operators spend on tasks such as beamline development and problem diagnosis and mitigation.

2 Setup

TRANSOPTR [5] [7] is a beam envelope optimization capable of tracking the first moments (centroid) and second moments (envelope) of the particle distribution through a beamline. [19] and [20] detail the use of TRANSOPTR for the OLIS portion of ISAC. We create a simulated beamline environment from TRANSOPTR by computing the centroid and envelope along the beamline at every time step.

2.1 Simulated Beamline Environment

A simulation for the OLIS beam is generated from TRANSOPTR, and we train a RDPG agent on the simulation. The OLIS section (see **A.1**) consists of 8 electrostatic quadrupoles, 1 electrostatic bender, 5 electrostatic y-steerers, 2 electrostatic x-steerers, 2 Faraday Cups (FCs), and 1 Rotary Profile Monitor (RPM). The simulation starts just after the ion beam is extracted from the source, 71.6 mm upstream of the first x-steerer. The simulation ends right after the profile monitor IOS:RPM8.

We develop a Python wrapper for TRANSOPTR and construct 2 simulated environments compatible with OpenAI gym [1]. TRANSOPTR is configured to discretize the centroid and envelope to 55 points along the beamline. During each episode, a random misalignment can be generated at each of the optical components to produce a non-optimal beam centroid (see **A.2**). By training on this type of environment, we hope an agent will learn optimal procedures for tuning beamlines with an unknown combination of intrinsic misalignments. Transfer learning strategies [21] can then be applied to adapt the agent to the real machine, which possesses some unknown misalignments (which we assume to be constant throughout one tuning episode). At each time step t, an observation o_t is produced from a simulated measurement at each FC and RPM. The agent then predicts an action a_t, corresponding to the adjustment angles of each steering element, and receives a reward r_t and new observation o_{t+1}. The agent only has knowledge of the simulated measurements and therefore must learn to infer the underlying physics of the beam and explore the beam state to maximize reward received.

YBeamlineEnv provides a simplified simulation of a beamline in only the vertical (y) dimension. Transmission is calculated based on equation (2) in 1D for dimension y. The action space corresponds to steering angles for a y-bender and 4 y-steerers. Note that the y-bender does not exist in the physical beamline. However the number of tuneable parameters is preserved since the last physical y-steerer is omitted in the simulation.

BeamlineEnv provides a simulation of the beamline in both x and y dimensions. Transmission is calculated using equation (2) in 2D. A x-bender and 4 x-steerers are added to the environment, at the same locations as those of the y-components in YBeamlineEnv, to form an action space of 10 steering elements. In reality, x-steering on OLIS makes use of a magnetic dipole and fewer x-steerers, which differs from the setup presented in BeamlineEnv. We believe that BeamlineEnv sufficiently captures the relevant degrees of freedom for the control problem at OLIS and can be used to verify learning capabilities of the agent.

2.2 Simulated Measurement

To simulate a measurement, we assume a Gaussian distribution for the particles and calculate transmission using the centroid and envelope determined by TRANSOPTR. Locations and widths of slits are incorporated along the beamline simulation according to their design in OLIS. We approximate the upper bound for transmission by integrating the shifted Gaussian particle distribution at each slit location. The incremental transmission at location i along the beamline is:

$$\Theta_{i,q} = \frac{1}{2}\left[\text{Erf}\left(\frac{w_i - \mu_i}{\sqrt{2}\sigma_i}\right) + \text{Erf}\left(\frac{w_i + \mu_i}{\sqrt{2}\sigma_i}\right)\right] \tag{1}$$

where σ_i is the rms-envelope of the beam, μ_i is the centroid of the beam, and w_i is slit width, or the wall width of 2cm if no slit is present.

The upper bound of total transmission at location k in dimension q is approximated (assuming no x-y correlation in phase space) as:

$$T_{k,q} = \prod_{i=0}^{k} \Theta_{i,q} \quad \text{(1D)} \qquad \text{and} \qquad T_k = T_{k,x} \cdot T_{k,y} \quad \text{(2D)} \tag{2}$$

A nominal current I_0 of 10.0 + δ nA (with a noise δ randomly sampled between [-0.1, 0.1] at each step) is assumed. The simulated measurement of a Faraday cup at location k is:

$$I_k = I_0 \cdot T_k \quad \text{(Beam)} \qquad \text{or} \qquad I_{k,y} = I_0 \cdot T_{k,y} \quad \text{(YBeam)} \tag{3}$$

2.3 Reward Function

At each step in the episode, the agent receives a step reward $r_t = R(a_t, s_{t+1})$ based off of some reward function R for the current action a_t and next state of the environment s_{t+1} after taking this action. We consider two reward functions for our experimentation.

SSE Reward Function We construct an artificial reward function to incentivize a well-aligned beam. The sum of squared errors (SSE) of the beam centroid from centerline along the beamline is used along with the reciprocal term. This reward function gives a very good relation to beam characteristics, but cannot realistically be measured on the physical beamline. This function is described in detail in **A.3**.

Measurement Reward Function We construct a second reward function to be directly obtainable from measurements on the beamline, to allow a better representation of the real machine. High transmission at the last FC is rewarded while large adjustments of the steerer angles or large offsets of beam centroids at the RPM are penalized (see **A.3**).

3 Recurrent Deep Deterministic Policy Gradients Agent

Deep deterministic policy gradients (DDPG) implements an actor-critic architecture which has shown to be effective for high dimensional and continuous action spaces [13]. The addition of a long short-term memory (LSTM) network [6] to the actor and critic networks forms the architecture for recurrent deep deterministic policy gradients (RDPG) [4]. At each step, RDPG takes in a sequence of the last ℓ observations $(o_t, ..., o_{t-\ell})$ to predict the next action a_t. This structure allows the agent to keep a memory state from each successive term of the sequence of past observations and more effectively learn in partially observed environments.

3.1 Model Architecture

We utilize the architectures from [4] and [13] and develop a RDPG agent using PyTorch [16]. During prediction, we define the sequence of past observations $(o_{t-\ell-1}, o_{t-\ell}, ..., o_{t-1})$ for some memory length ℓ where o_τ is the zero padded vector if $\tau < 0$. This sequence is passed to the actor for predicting the next action. We found that including previous actions as a part of the observation space gave significantly better performance than only using the measurements as observation.

3.2 Training

During training, the agent samples m mini-batches of trajectories $(T_\ell^i)_{i=1,...,m} = (o_d^i, a_d^i, r_d^i, ..., o_{d+\ell}^i, a_{d+\ell}^i, r_{d+\ell}^i, o_{d+\ell+1}^i)_{i=1,...,m}$ (with random segments $d, ..., d+\ell$) from the episode sequences $(o_1, a_1, r_1, ...o_t, a_t, r_t, o_{t+1})_{i=1,...,m}^i$. The critic network is optimized from the Bellman equation and the actor network using deterministic policy gradients following the approaches of [13].

4 Experiments and Results

We train the RDPG agent on both YBeamline and Beamline environments, using SSE and Measurement reward functions. For each of these cases, the environment generates new random misalignments for the beamline at every episode, effectively allowing the agent to train on "new" beamlines. We also emulate training an agent from scratch directly on the physical beamline by creating a fixed version for both environments, called FixedYBeamlineEnv and FixedBeamlineEnv. For the fixed environments, misalignments in the optical components of the beam are kept constant across episodes. Hyperparameters for all experiments are described in **A.4**. We utilize an exponentially decaying exploration strategy which we termed Normal Noise Strategy to explore the action space (see **A.5**).

Results for training are shown in **A.6**. For each of YBeamline and Beamline environments, the RDPG agent was trained on 14 different random seeds, and we recorded the number of episodes trained before the agent successfully solves the environment. Each agent was trained for 160000 episodes with 10 steps per episode.

Figure 1 plots the number of training episodes required before solving for each environment and reward function. Due to the simplicity of YBeamlineEnv, it is easier to obtain high rewards using MSMT reward, hence its faster training times compared to SSE. The inverse is observed for the more complex BeamlineEnv, where the 2 dimensional problem increases the difficulty of obtaining a high MSMT reward. A demo of the trained agent for BeamlineEnv can be seen in **A.7**. Figure 2 shows number of training episodes to solve fixed environments. The results indicate training from scratch on a physical beamline would be impractical due to the large number of training episodes required.

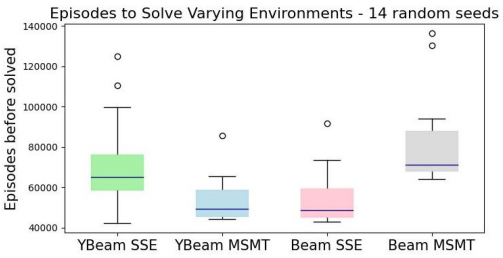

Figure 1: Box plot of YBeam and Beam training using SSE and MSMT reward functions. Solved criterion is an episode reward of > 60 for Beam using MSMT reward and > 100 for all others.

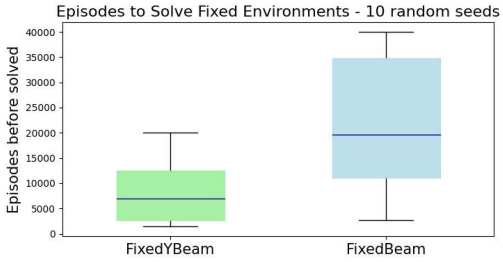

Figure 2: Box plot of FixedYBeam and FixedBeam training using MSMT reward function. Solved criterion is an episode reward of > 100 for FixedYBeam and > 60 for FixedBeam.

5 Limitations

Hyperparameters chosen for our experiments were determined from comparisons between various single-run trials which may have high variability and we believe a more thorough examination of the hyperparameter space will lead to improved agent performance. While RPDG was utilized for this work based off of its success from past literature, other reinforcement learning algorithms and methods should also be be explored for future iterations. From our current definition of the reward function (which incentivizes high transmission of the beam), it may be possible for the agent to develop a tune that is apparently optimal by maximizing only those characteristics, while producing undesired beam behaviour further downstream. Future work should perform additional analysis of the downstream impacts of tuning and may involve the inclusion of longer portions of the beamline or implementation of sophisticated reward functions.

6 Conclusion

The complex control problem of accelerator tuning can be tackled with methods of reinforcement learning. We have developed and demonstrated a framework for training RDPG agents on simulated beamlines. Our experiments indicate that training an agent from scratch directly on the physical beamline may not be feasible due to the large number of episodes required. However, there is much potential for transfer learning techniques as a trained agent is able to achieve excellent tuning results in very few steps on new randomly generated beamline episodes. The framework has the potential to be trained on a more accurately simulated environments and deployed to the physical beamline. This work shows promising applications of reinforcement learning in automating and improving tuning procedures for particle accelerators.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile. Neural networks for modeling and control of particle accelerators. *IEEE Transactions on Nuclear Science*, 63(2):878–879, 2016.

[3] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. *Proceedings of Machine Learning Research*, 15:315–323, 2011.

[4] N. Heess, J. J Hunt, T. P Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *avXiv:1512.04455*, 2015.

[5] E. A. Heighway and R. M. Hutcheon. Transoptr - a second order beam transport design code with optimization and constraints. *Nuclear Instruments and Methods in Physics Research*, 187(1), 1981.

[6] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[7] https://gitlab.triumf.ca/beamphys/transoptr.

[8] https://www.triumf.ca/.

[9] V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. Della Porta, N. Bruchon, and G. Valentino. Sample efficient reinforcement learning for cern accelerator control. *Phys. Rev. Accel. Beams*, 23(124801), 2020.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2017.

[11] R. E. Laxdal. Beam optics design for the isac off-line source. *http://lin12.triumf.ca/text/ISAC/ISAC-pre2004/laxdal/optics.html*, 1996.

[12] R. E. Laxdal. Corrections and additions to optics design for the isac off-line source. *http://lin12.triumf.ca/text/ISAC/ISAC-pre2004/laxdal/optic_corr.html*, 1996.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2019.

[14] K. Mani, M. Kaushik, N. Singhania, and K. M. Krishna. Learning adaptive driving behavior using recurrent deterministic policy gradients. *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2092–2098, 2019.

[15] D. R. Song C. Yang C. McGreavy and Z. Li. Recurrent deterministic policy gradient method for bipedal locomotion on rough terrain challenge. *2020 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 311–318, 2018.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[17] S. Rozada, D. Apostolopoulou, and E. Alonso. Load frequency control: A deep multi-agent reinforcement learning approach. *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5, 2020.

[18] P.W. Schmor, R. Baartman, P. Bricault, M. Dombsky, G. Dutto, S. Koscielniak, R.E. Laxdal, F. Mammarella, G.H. Mackenzie, R. Poirier, L. Root, G. Stanford, G. Stinson, I. Thorson, and J. Welz. Status of the triumf-isac facility for accelerating radioactive beams. *PAC'97 Proceedings*, 1997.

[19] O. Shelbaya. A Quick TRANSOPTR Primer. *Beam Physics Note TRI-BN-20-06R*, 2020.

[20] O. Shelbaya. OLIS to RFQ Beam Transport and Acceleration in TRANSOPTR. *Beam Physics Note TRI-BN-20-13R*, 2020.

[21] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *arXiv:1911.02685*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 1
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] The experiments performed in this paper do not pose any of the potential negative social impacts outlined in the ethics review guidelines, and adheres to the General Ethical Conduct.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code repo continues to be under development and we plan to release it to the public at a later date, once progress has matured.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For hyperparameters used in training, refer to Section A.4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Distributions from random seeds are described in Figures 1 and 2
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Compute and training times are described in A.6
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] All code libraries and assets were cited.
   (b) Did you mention the license of the assets? [Yes] We use TRANSOPTR for beam line simulation as mentioned in section 2. Access to the source code of TRANSOPTR is provided to the community by request via [7]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]
   No crowdsourcing or research with human subjects was conducted for this paper.

A Appendix

A.1 OLIS

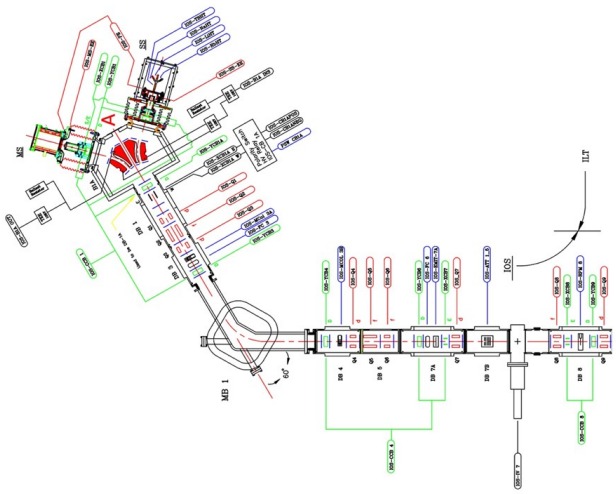

Figure 3: Sketch of the ISAC Offline Ion Source (OLIS)

A.2 Simulated Environments

Figures 4, 5, 6, and 7 show the simulated environments created from TRANSOPTR. The horizontal scale measures distance along beamline (s) in cm, and the vertical scale measures the width of the beamline wall (in x or y) in cm. The solid line represents the beam centroid and the dashed lines represent the envelope for the x (blue) and y (red) dimensions. The vertical dashed lines mark the locations of the bender (dark green), steering elements (light green), and quadrupoles (red). The thick solid lines marks the locations of Faraday Cups FC3 (blue) and FC6 (green), and RPM (red). For misalignment, a random offset (sampled from a normal distribution with standard deviation of 0.05 cm) in x and y was applied to each of the quadrupoles.

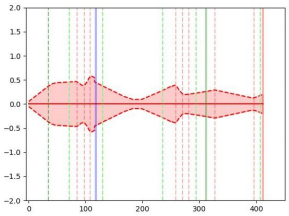

Figure 4: YBeam Environment with no misalignments

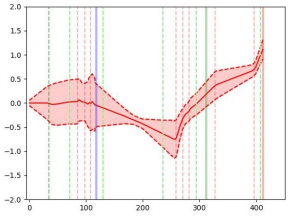

Figure 5: YBeam Environment with an instance of randomly generated misalignments

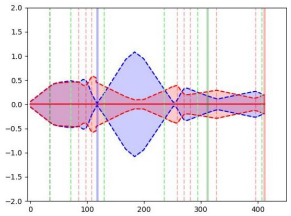

Figure 6: Beam Environment with no misalignments

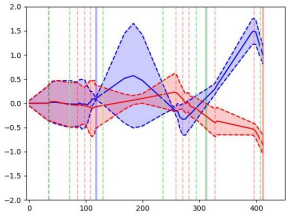

Figure 7: Beam Environment with an instance of randomly generated misalignments

A.3 Reward Functions

For each time step t, a reward r_t is obtained from $r_t = R(a_t, s_{t+1})$ for the action a_t taken at that step and next state of the environment s_{t+1}, using some reward function R.

SSE Reward Function uses the SSE along with the inverse term in order to give larger positive reward for the case of very well-aligned beams. This reward exploits the simulated environment, as it will not be reproducible through real measurements of a physical beamline. For YBeamlineEnv, it is defined as:

$$R_{t,y}^{\mathrm{SSE}} = R_{y}^{\mathrm{SSE}}(a_t, s_{t+1}) = -\sum_{i=0}^{M} \mu_{i,y}^2 + min(\frac{1}{\sum_{i=0}^{M} \mu_{i,y}^2 + \delta}, 100) \tag{4}$$

for every centroid $\mu_{i,y}$ for all M points along the beam, and $\delta = 0.0001$ to avoid zero division.

For BeamlineEnv:

$$R_{t}^{\mathrm{SSE}} = R^{\mathrm{SSE}}(a_t, s_{t+1}) = \frac{1}{2}(R_{t,x}^{\mathrm{SSE}} + R_{t,y}^{\mathrm{SSE}}) \tag{5}$$

Measurement Reward Function uses only measurable information from the FC, RPM, and steering positions. Separate terms are constructed to incentivize high transmission at FCs, and to penalize large offsets at the RPM and large steering angles.

The transmission reward was designed to give a reward of 10 per step for 85% transmission based on I_* in equation (3):

$$R_{t}^{\mathrm{FC}} = (\frac{I_*}{8.5})^4 \cdot 10 \tag{6}$$

The action penalty P_{actions} is defined as:

$$P_{\mathrm{actions}} = \alpha \cdot \sum_{i=0}^{n} |a_i|^3, \tag{7}$$

for a space of n actions and scaling factor α

The RPM penalty P_{q}^{RPM} is defined as:

$$P_{q}^{\mathrm{RPM}} = min(\beta \cdot \mu_q^2, 10) \tag{8}$$

for the beam centroid (in dimension q) μ_q at the RPM, and scaling factor β,

The measurement reward function is defined for YBeamlineEnv:

$$R_{t,y}^{\mathrm{MSMT}} = R_{y}^{\mathrm{MSMT}}(a_t, s_{t+1}) = R_{t}^{\mathrm{FC}} - P_{\mathrm{actions}} - P_{y}^{\mathrm{RPM}} \tag{9}$$

and for BeamlineEnv:

$$R_{t}^{\mathrm{MSMT}} = R^{\mathrm{MSMT}}(a_t, s_{t+1}) = R_{t}^{\mathrm{FC}} - \frac{1}{2}(P_{x}^{\mathrm{RPM}} + P_{y}^{\mathrm{RPM}} + P_{\mathrm{actions}}) \tag{10}$$

Scaling constants were used to appropriately scale episodic rewards:

$\alpha = 250000$ such that the upper bound of action magnitudes of 0.02 rad will give $P_{\mathrm{actions}} = -10$

$\beta = 250$ such that a RPM measurement of magnitude 0.2 cm or more will give $P_{q}^{\mathrm{RPM}} = -10$

A.4 Hyperparameters

We use a batch size of 64, discount factor $\gamma = 0.99$, soft target update ratio $\tau = 0.005$, actor learning rate $\alpha = 0.001$, critic learning rate $\beta = 0.002$, a replay buffer with capacity of 20000 episodes, and a memory length $\ell = 5$. The Adam [10] optimizer is used for all network optimizations. All hidden layers uses ReLU [3] activation, and the output layer of the actor applies a tanh to bound all actions. For all 1D environments, we use an actor RDPG network structure with two fully connected layers of 16 and 32 units, connected to a LSTM layer with hidden dimension of 32, connected to 2 fully connected layers of 64 and 32 units. The critic network has two fully connected layers of 16 and 32 units for the observation input and one fully connected layer of 32 units for the action input connected to a LSTM layer with hidden dimension of 64 units, connected to an output layer of 64 units. For 2D environments, all fully connected and LSTM layers are doubled in size.

A.5 Normal Noise Strategy

During training, an exponential decay strategy for exploration is defined with standard deviation σ_m at step m:

$$\sigma_m = \begin{cases} \sigma_0 \cdot (\frac{\sigma_0}{\sigma_f})^{-n/N} & \text{if } m \leq T \\ \sigma_f & \text{if } m > T \end{cases} \tag{11}$$

for an initial σ_0, final σ_f, and decay steps T.

Each selected action a_i is added with some noise $a_i \leftarrow a_i + \nu(\sigma_t) * a_{max}$ where a_{max} is the action bound and $\nu(\sigma_m)$ samples a random normal noise with standard deviation σ_m

For both varying environments, $\sigma_0 = 0.2$, $\sigma_f = 0.001$, and $T = 900000$.

For the fixed environments, T is set to 1000

A.6 Training Plots

Experiments were carried out for each environment and reward function. Reward vs steps was averaged over 14 random seeds and plotted in Figures 8, 9, 10, and 11. Baseline reward was generated by averaging the rewards obtained from running 100000 episodes in the environment and taking no action at each step.

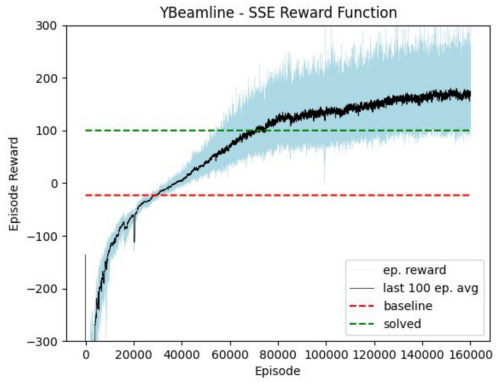

Figure 8: Average over 14 seeds of reward vs steps during training for YBeam with SSE reward function

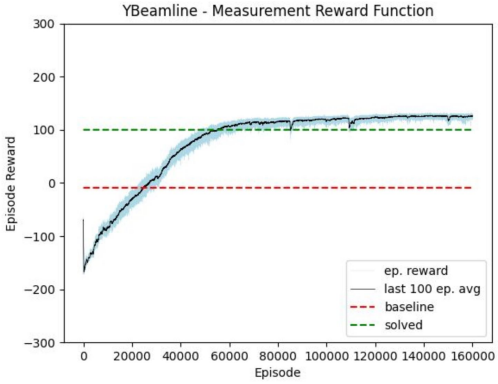

Figure 9: Average over 14 seeds of reward vs steps during training for YBeam with Measurement reward function

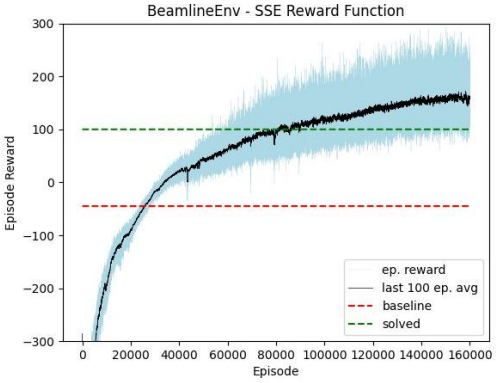

Figure 10: Average over 14 seeds of reward vs steps during training for Beam with SSE reward function

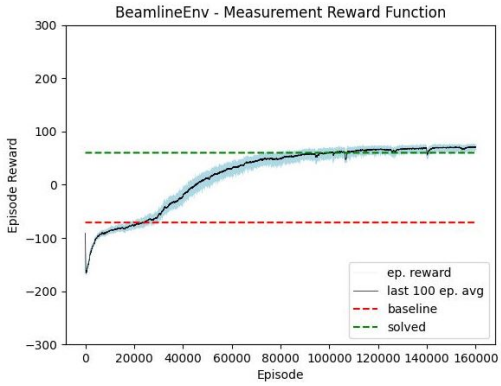

Figure 11: Average over 14 seeds of reward vs steps during training for Beam with Measurement reward function

Training times required around 16 hours for YBeamEnv and around 20 hours for BeamEnv. Up to 8 agents can be trained on a single GeForce RTX 2080 GPU, and each agent utilizes around 2.7 Intel(R) Xeon(R) Gold 6130 (2.10GHz) CPUs during training.

A.7 Demo

Figure 12 shows a demo of a trained agent on BeamlineEnv with the Measurement reward function, predicting on a random episode. Transmission to FC3 and FC6, measurements at the RPM in x and y, step reward, and episode reward are included in each figure. Initial transmission to FC6 was 0% and the agent achieved a tune with 82% after 5 steps. In the remaining steps, the agent maintained this level of transmission while slightly improving in alignment, resulting in a total episode reward of 69.

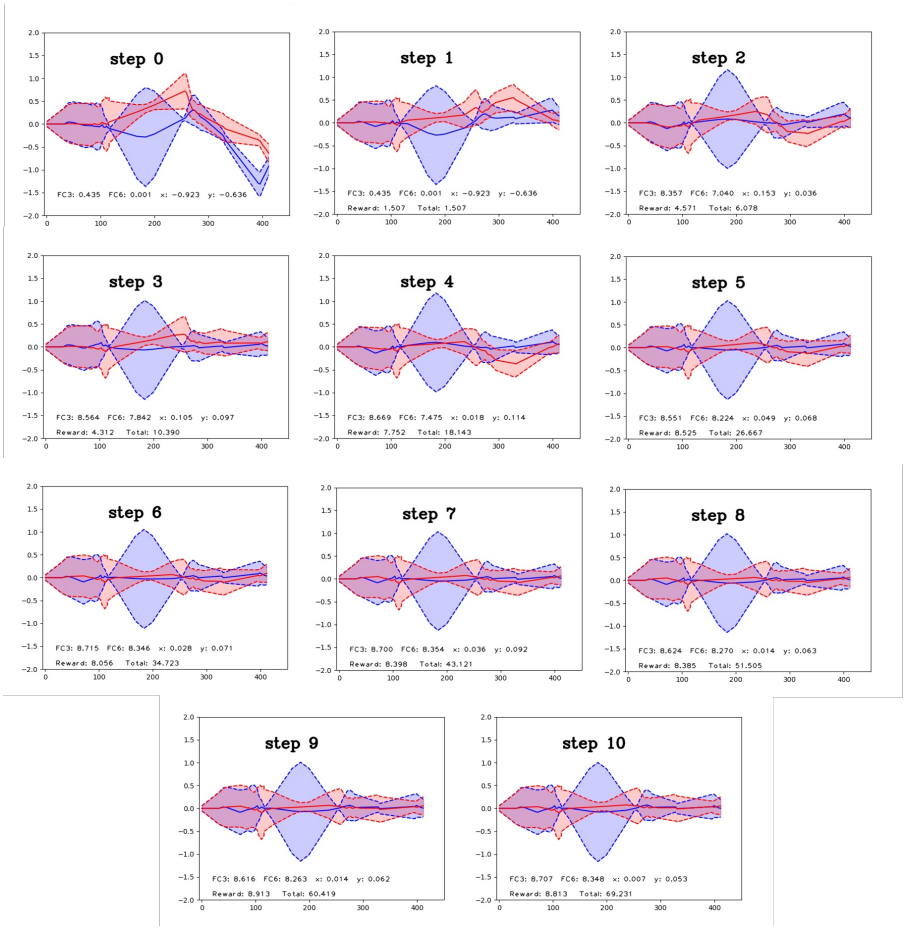

Figure 12: A demo episode of a trained agent predicting on a newly generated instance of BeamEnv.