TuneAI: Documentation & Handover

Achim Andres, David Wang, Harpriya Bagri

November 17, 2022

Contents

1 Overview

This document provides information about the current status of the project with detailed information about the set up and functionalities of the code. Chapter 2 shows how to setup your account. In chapter 3, detailed information about TRANSOPTR, the beam simulation and the python wrapper for the reinforcement learning agent is given. Chapter 4 gives actual information about the agent that is used to tune the beamlines, while chapter 5 provides details about the setup of the project on Compute Canada. Having an agent trained, an explanation how to deploy and test an agent on a real beamline is given in chapter 6, while chapter 7 shows you how to validate trained agents on the simulation. The documentation ends with chapter 8 with some ideas and tasks for the future.
Some parts of this document are copied from:

- Harpriya Bagri. She wrote a similar document for the previous co-op student which is attached to this document.
- Olivier Shelbaya's A Quick TRANSOPTR Primer
- David Wang's Accelerator Tuning with Deep Reinforcement Learning

Write me an email, if you want to edit and contribute the documentation: Achim Andres

2 Setup

2.1 triumf-ml1.phas.ubc.ca Cluster

To get access to the local TRIUMF ml1 cluster, send Wojtek Fedorko a ssh key (RSA, 4096 bits long in openssh format) and preferred user id. On linux, WSL and OSX, a key can be generated via

```
ssh-keygen -t rsa -b 4096
```

Two keys are generated and saved (if not specified) in your local .ssh/ directory. The key you need to send starts most likely with ssh-rsa. If you need further information, visit: `https://www.ssh.com/academy/ssh/keygen`. The most convenient way to get access to ml1 and work on the project is via x2go `https://wiki.x2go.org/doku.php`. After installation, you need to put the settings as specified in Figure 1 to get access. Further information can be found in Table 1.
ML1 provides eight GeForce RTX 2080 GPUs. Each GPU can train up to 8 agents and each agent utilizes around 2.7 Intel(R) Xeon(R) Gold 6130 (2.10GHz) CPUs during training. To display the GPU usage run

```
nvidia-smi
```

in your terminal. In case you have finished a training but you see under processes still an orphan process running, kill it by running

```
kill PID_number
```

to free up the space for other users on ml1.

Table 1: x2go settings for ml1.

Name	Description
Sitzungsname	Name of your choice for the session
Host	host adress: triumf-ml1.phas.ubc.ca
Login	User id you sent to Wojtek
SSH-Port	22
RSA-/DSA-Schlüssel	path to your ssh key
Sitzungsart	KDE was not working, but XFCE worked (you might need to try it out).

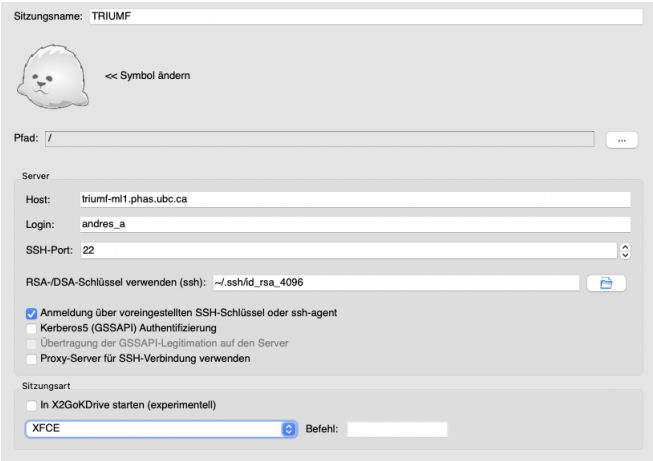


Figure 1: x2go session settings for ml1.

2.2 TuneAI Git Repository

The git repository link is: `https://gitlab.triumf.ca/beamrl/tuneai`. It contains all code that is needed to train, test and validate (off- and online) agents. Create your own development branch and only merge your working code to master.

2.3 The Container

The code runs in a container. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. To start the container, start_container_beams.sh needs to be sourced.

```
source path/to/tuneai/start_container_beams.sh
```

It is convenient to add this line to your .bashrc in your home directory, so that the container is automatically started whenever you open a terminal. Within the container all software (mainly python packages) is available to run the code. If you need to add more software, you need to get in touch with Wojtek Fedorko. The container itself is a large file, saved in /fast_scratch1/triumfutils/containers. Note: If you want to run your code on ComputeCanada, you need to copy the container to the ComputeCanada cluster of your choice (see chapter 5). The container also allows using VirtualStudioCode in your x2go session, to modify the code. VSCode can be opened by running

```
code .
```

in the terminal. However, for some students VirtualStudio had problems recognizing the computer's keyboard via x2go. Instead, a sshfs connection to ml1 is used to mount the repository to the computer to use a remote code editor. More information about sshfs can be found here. An alias can be created in your bashrc to mount and unmount data. The bash code below also shows how to mount and unmount from ComputeCanada (see chapter 5).

```
alias mount-triumf='sshfs -o allow_other triumf:/home/andres_a/repos/
    tuneai/ /Users/achim/mnt/triumf/'
alias umount-triumf='umount  /Users/achim/mnt/triumf/'

alias mount-coca-narval='sshfs -o allow_other aandres@narval.
    computecanada.ca:/home/aandres/ /Users/achim/mnt/coca-narval/'
alias umount-coca-narval='umount  /Users/achim/mnt/coca-narval/'

alias mount-coca-cedar='sshfs -o allow_other aandres@cedar.computecanada.
    ca:/home/aandres/ /Users/achim/mnt/coca-cedar/'
alias umount-coca-cedar='umount  /Users/achim/mnt/coca-cedar/'
```

By running mount-triumf, the git repository is loaded on your remote machine (in the above case into /Users/achim/mnt/triumf/). You can open your favorite code editor in this directory to modify the code. Another benefit of running sshfs is, that you don't need to worry about displaying plots for monitoring the training on ml1 or ComputeCanada, because the plots are also mounted on your local machine and can be opened with your favorite program.

2.4 TRIUMF Data Science slack space

The TRIUMF Data Science slack space is the heart of communication inside TRIUMF among Machine Learning scientists. Wojtek is responsible for inviting you. **IMPORTANT:** Always announce on which GPU id you are training as shown in Figure 2. People will get angry at you, if you don't.
A lot of internal communication (especially with beam physicist people) is done via Microsoft Teams. You get an account as soon as you are enrolled with TRIUMF.

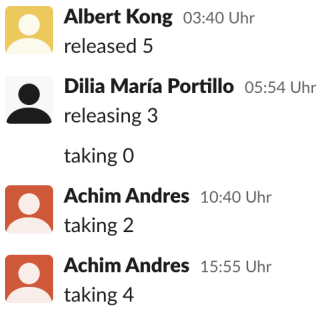


Figure 2: Always specify the GPU id you are training on.

3 Transoptr

To simulate the beamlines at OLIS and ALTIS, TRANSOPTR is used. TRANSOPTR was originally written in 1984, in Fortran. It is now maintained by the Beam Physics group at TRIUMF. It tracks moments of beam distribution and has parameters like length, aperture radius, quadrupole, etc. TRANSOPTR assumes the beam is on axis and does not include misalignments of elements.

3.1 xml2optr & acc database

Both xml2optr and the acc database are essential to create the files which are used by the environment to evaluate the beamline. Both should be already available inside the container. Further information can be found here

- Acc: `https://gitlab.triumf.ca/hla/acc`
- xml2optr: `https://gitlab.triumf.ca/hla/acc-utilities/xml2optr/-/tree/master/`.

In the acc database, all kinds of information about elements in a beamline is stored. Before pulling information, make sure the acc repository is up to date by running

```
update-acc
```

Your local acc database gets automatically updated as soon as you start training or validating agents. The following script shows how to pull the "inc_steers" variable from the acc database. The final output of the script is "pos_x". Figure 3 shows a screenshot of the acc/ariel/sequence/ALTS_2.xml file. In the repository, this is heavily used to avoid hard coding in configuration files.

```
from accpy import acxml
path = 'altis-altn-agts-ayield'

elem_id = 'ALTS:XCB5'
```

```
my_element = acxml.get_element_by_id(elem_id, path=path, json=True)
print(my_element['epics']['setpoint']['@inc_steers'])
```

```
<element id="ALTS:XCB5" l="0*mm" s="186.199*mm" type="ecb">
  <epics>
    <setpoint pv="ALTS:XCB5:VOL" unit="V" min="-1000.0" max="1000.0" inc_steers="pos_x"/>
  </epics>
</element>
```

Figure 3: Screenshot of the acc database.

To be able to simulate a beamline, a data.dat and an sy.f file need to be produced. Create a folder with a file called your_custom_tune.xml.

```
<root xmlns:xi='http://www.w3.org/2001/XInclude'>
    <optr s11="0.2373*cm" s22="10*mrad" s33="0.2373*cm" s44="10*mrad" s55
        ="0*cm" s66="0.000017*rad" r12="0.99099" r34="0.99099" bunchcharge
        ='0' end="ALTS:FC19"/>
    <tune chargestate="1.0" energy="30.0*keV" mass="132.90545*u"/>
    <xi:include href='ariel/tune/altis-altn-agts-ayield_ref.xml' xpointer
        ="xpointer(//root/tune)" parse="xml"/>
</root>
```

After running

```
run-xml2optr altis-altn-agts-ayield -tf your_custom_tune.xml
```

the data.dat and sy.f file are produced. The above code produces the simulation input files for the ALTIS source to ALTS:FC19 at ARIEL. The file contains the beam starting phase space s11 to s66, charge, until where you want to have the simulation, starting energy and isotope. If you need further guidance, Spencer Kiy is the first person to approach.

3.2 sy.f

This file is an input file. It is written in Fortran and describes each element of the beamline. For a detailed description of each function call, please see the source code. A few changes need to be made to prepare the file for training. Before learning how to modify the sy.f file, a few rules about fortran:

- Fortran only allows 72 characters per line. The transoptr executable won't be produced if this number is exceeded.
- A number like 60 without a decimal point will be interpreted as an integer
- All variables beginning with character i,j,k,l,m,n are assumed to be integers, unless there is an implicit statement overruling it.

The following list gives information about how to modify the sy.f file.

- **Steering Shifts:** The file does not include the steering shifts. They need to be added as shown in Figure 4. The genshift function as shown in Figure 4 adds an angle to the beam in x direction. In case a y steerer needs to be added, the variable needs to be placed in 4^{th} position. The positions of the steerers are automatically marked in the sy.f file.

```
58      call drift(4.4705,".")
59      ! ALTS:XCB5
60      call drift(8.2,".")
61      ! ALTS:Q5:match-point
62      ! FIT at the start of a periodic section
```

```
56      call drift(4.4705,".")
57      ! ALTS:XCB5
58      call genshift(0., XCB5, 0., 0.)
59      call drift(8.2,".")
60      ! ALTS:Q5:match-point
```

Figure 4: **Left:** Original sy.f file. **Right:** Processed sy.f file with genshift function.

- **Misalignment of quadrupoles:** Transoptr does not include the misalignment of quadrupoles. To simulate this effect, again the genshift function is used in a fortran subroutine as follows:

```
subroutine detshequad(VX,AAu,ELu,wt,x_sh,y_sh,QUA)
character (*) :: qua
common/misal/smis
call genshift(x_sh,0.,y_sh,0.)
call equad(VX,AAu,ELu,wt,QUA)
call genshift(-x_sh,0.,-y_sh,0.)
return
end
```

The subroutine adds an offset to the beam in x and y direction, propagates it to the (electrostatic) quad and shifts the beam back. Instead of misaligning the quadrupole, the beam is misaligned. All *Equad* functions in the sy.f file need to be replaced by the subroutine.
For example:

```
call Equad(0.001*QE1,1.27,3.441,wo,'ALTIS:Q1')
```

gets replaced by

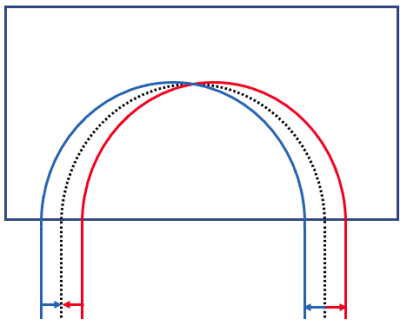
```
call detshequad(0.001*QE1,1.27,3.441,wo,qx1,qy1,'ALTIS:Q1')
```

where qx1 and qx2 denote the misplacement of the quadrupole in x and y direction, respectively.

- **Misalignment of dipoles:** Also the misalignments of dipoles are not foreseen in TRANSOPTR. In contrast to quadrupoles, the misalignment of dipoles is not trivial. Benders have focusing properties both in the bend plane and in the non-bend plane. Like the OLIS benders, that bend in horizontal direction. They have focusing properties in the vertical direction too. In that sense, they are just like quadrupoles. **Vertically**, you'd shift it upward at entry and downward at exit. (see *by* in genshift in y direction line(79) and (86) in figure 5(b)) But **horizontally**, that is not the case. For a misaligned bender in that direction, you shift the beam on entry, but the amount you shift on exit depends on the total bend angle. You can work this out with a compass. For example, if the bend is 90 degrees, you shift at entry and do not shift at exit. If it's 180 degrees, you shift at entry and shift in opposite direction at exit. So the shift at exit is cosine of the bend angle times the entry shift:

$$x_{\text{After}} = \cos(\alpha_{\text{Bend}}) x_{\text{Before}}. \tag{1}$$

In line 83 is an additional genshift function. The XMB variable is the interaction of the agent with the bending power. By changing the current, the bending radius changes which is simulated by adding a kick in the magnet in x direction. The variable *dby* is used to simulate a **roll** of the magnet by adding a kick to the beam in vertical direction. A rotation of the magnet around the vertical axis is not yet implemented (and also not straight forward).



(a) Schematic of Dipole

```
79      call genshift(bx,0.,by,0.) ! This function takes the shift
80      call edge(0.0,29.9887,60.0,0.0,0.317,2.0,5.08,0.0,wo)
81      call bend(29.9887,30.0,0.0,'IOS:MB')
82      call marker('IOS:MB')
83      call genshift(0.,XMB,0.,dby)! Additional kick in y directio
84      call bend(29.9887,30.0,0.0,'IOS:MB')
85      call edge(0.0,29.9887,60.0,0.0,0.317,2.0,5.08,0.0,wo)
86      call genshift(-bx*cos(60*pi/180)+bz*cos(60*pi/180),0,-by,0)
```

(b) Implementation in fortran

Figure 5: Implementation a dipole in sy.f

- **Misalignment of the source:** Beam coming out of the source is most probably not aligned to the beamline axis. Therefore an additional genshift function is applied before starting the beamline with offsets and additional starting angles both in x and y.

```
call genshift(dx,dxp,dy,dyp)
```

- **File header:** After finishing the editing of the sy.f source code, all variables need to be declared in the file header. By default, only the quadrupole strength are already there. The misalignment of the source, the individual quadrupole misplacements and steering angles still need to be declared as shown in Figure 6. The syntax in Fortran is harsh. To avoid the exceeding of numbers of characters per line, a line break can be done by using +. The position of + has to be at the place as shown in Figure 6.

```
COMMON/BLOC1/QE1,QE2,QE3,QE4,QE5,QE6,QE7,QE8,QE9,QE10,QE11
```

```
COMMON/BLOC1/QE1,QE2,QE3,QE4,QE5,
+QE6,QE7,QE8,QE9,QE10,QE11,
+dx,dy,dxp,dyp,qx1,qy1,qx2,qy2,qx3,qy3,qx4,
+qy4,qx5,qy5,qx6,qy6,qx7,qy7,qx8,qy8,
+qx9,qy9,qx10,qy10,qx11,qy11,qx12,qy12,
+qx13,qy13,qx14,qy14,qx15,qy15,qx16,qy16,
+qx17,qy17,qx18,qy18,qx19,qy19,qx20,qy20,
+XCB5,YCB5,XCB7,YCB7,
+XCB15,YCB15,XCB17,YCB17
```

Figure 6: **Left:** Original sy.f file. (Only the quadrupole strengths are declared) **Right:** Processed sy.f file with all variables declared.

3.3 data.dat

The data.dat file is an input file for TRANSOPTR to generate the executable which contains additional information about beam properties and variables, declared in sy.f. It is automatically generated after running xml2optr, but needs modification. An example of the modified data.dat file is shown in Figure 7.

```
0.03 0.0 0.0 1.23801e+05 1.0 0.0 ! En[MeV], mom., brho, mass[MeV], charge, beam current or bunch charge
1 5 0.01 0.0001 ! iprint (-1=slim output for envelope app, >0 for legacy optr outputs), IVOPT (4: 4-D space-charge, 5: 6-D
0 0.0 1.0 0.0 ! for external Bs field: nb. lines in data file (0=disable), s offset, unit of s (1=cm), unit of Bs (1=kG)
0.2373 0.01 0.2373 0.01 0.0 1.7e-05 ! bunch dim: x,x',y,y',z(bun. len.),dp/p
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ! 1 means x,y,z in cm, x',y',dp/p in rad (dimensionless)
2
1 2 0.99099 3 4 0.99099
63
805.0      0.0        5000.0     0 ! ALTIS:Q1:VOL QE1 V
1550.0     0.0        5000.0     0 ! ALTIS:Q2:VOL QE2 V
1431.0     0.0        5000.0     0 ! ALTIS:Q3:VOL QE3 V
760.6      0.0        5000.0     0 ! ALTIS:Q4:VOL QE4 V
1691.0     0.0        2500.0     0 ! ALTIS:Q5:VOL QE5 V
1029.0     0.0        2500.0     0 ! ALTE:Q26:VOL QE6 V
1029.0     0.0        2500.0     0 ! ALTS:Q6:VOL QE7 V
1343.0     0.0        5000.0     0 ! ALTS:Q8:VOL QE8 V
815.0      0.0        2500.0     0 ! ALTS:Q9:VOL QE9 V
1029.0     0.0        2500.0     0 ! ALTS:Q12:VOL QE10 V
1029.0     0.0        2500.0     0 ! ALTS:Q16:VOL QE11 V
0.00     -1.0  1.0  0 ! xmis dx cm
0.00     -1.0  1.0  0 ! ymis dy cm
0.00     -1.0  1.0  0 ! xpmis dxp rad
0.00     -1.0  1.0  0 ! ypmis dyp rad
0.00     -1.0  1.0  0 ! quad1xmis q1x cm
0.00     -1.0  1.0  0 ! quad1ymis q1y cm
0.00     -1.0  1.0  0 ! quad2xmis q2x cm
0.00     -1.0  1.0  0 ! quad2ymis q2y cm
```

Figure 7: Modified data.dat file.

- **Line 1** contains energy, momentum, brho, mass, charge and beam current. The energy is important when running tests on the real beamline, as the particles energy should be set to the particles energy in the simulation (see chapter 6.2). The values are adapted from the xml configuration file before creating the sy.f and data.dat file.
- **Line 2** contains the iprint variable, which needs to be changed to 1 for successfull compilation of the transoptr executable. The deals with beam calculation parameters , which do not need to be changed.
- **Line 3** The initial boolean allows for the activation of an external s-oriented magnetic field, where s is the Frenet-Serret accumulated arclength of the reference particle. In general, no numbers need to be changed.

- **Line 4** specifies the starting bunch parameters, in terms of the 2rms size of the distribution in the six canonical phase space coordinates. These are the elements ii of the beam matrix. The transverse beam dimensions are in units of length and the canonical momenta in angles. Units are specified in line 5. Spencer will provide the correct values.

- **Line 5** controls unit definitions. Default dimensions for x,y and z are cm, meaning if the entry in line 5 is set to 1.0. Dimensions for Px,Py,Pz are radians. For instance, if one wishes to use inches for the x-dimension, the first entry in line 5 would read 0.3937, which is the factor (1"/2.54cm). Regarding the longitudinal coordinates z, Pz, we clarify that in TRANSOPTR they are, by definition

$$z = \beta c \Delta t \tag{2}$$

$$Pz = \frac{\Delta E}{\beta c} \tag{3}$$

  The base units for Pz, when set to 1.0 produce units in radians. In general, no numbers need to be changed.

- **Line 6:** Number of correlation parameters. In general, no numbers need to be changed.

- **Line 7:** The correlation coefficients for the I,J elements of the σ-matrix. Note, in the present example, line 7 only contains r12 and r34, the correlation coefficients for x,Px and y,Py, though other I J correlations can be provided, meaning line 6 needs to be updated as well. These parameters are important as they define the orientation of the phase space ellipse containing the rms distribution of the beam. In general, no numbers need to be changed.

- **Line 8:** Number of tuneable element parameters.

Before executing transoptr on data.dat and sy.f you need to add the tuneable parameters to the data.dat file. The structure for adding a tuneable parameter is the following:

```
[set point], [min value], [max value], [bool: optimize? (int)] !
          [configfile var name] [sy.f-file var name] [unit]
```

- **Quadrupoles**

set point: Theoretical Value for the Quad in V. Should be already correct after running xml2optr. They can also be found here for OLIS and here for ARIEL. The extraction voltage from the data.dat and the correct charge need to be set to see the correct values for the quads. As all optics are electrostatic, the isotope doesn't really matter.

min value: minimum voltage of quadrupole

max value: maximum voltage of quadrupole

optimize: always at 0

configfile var name: quad name as specified in environment config file (see chapter 3.8)

sy.f var name: quad name as specified in sy.f

unit: V

- **Source Misplacement**

set point: 0

min value: -1.0

max value: 1.0

optimize: always at 0

configfile var name: variable name as specified in environment config file (see chapter 3.8)

sy.f var name: variable name as specified in sy.f

unit: for displacements: cm / for angles rad

– **Quadrupole Misplacement**

set point: 0
min value: -1.0
max value: 1.0
optimize: always at 0
configfile var name: variable name as specified in environment config file (see chapter 3.8)
sy.f var name: variable name as specified in sy.f
unit: cm

– **Steerers**

set point: 0
min value: -0.1
max value: 0.1
optimize: always at 0
configfile var name: variable name as specified in environment config file (see chapter 3.8)
sy.f var name: variable name as specified in sy.f
unit: rad

Important: The sequences of variables in data.dat need to be the same as in the header of sy.f! If you need further information about TRANSOPTR, you can refer to Olivier Shelbaya's A Quick TRANSOPTR Primer.

3.4 Running TRANSOPTR

After the sy.f and data.dat files are produced, edited and moved into a new directory in transoptr/ in your repository directory, you can create the transoptr executable. The source files for transoptr are in /workspace/transoptr/. However, for some reason I don't have read permission for this folder (You should complain to root, if you also don't have read permission) anymore. Therefore I copied the transoptr source files into my home directory (/home/andres_a/repos/). To create the executable, run

```
/directory/to/transoptr/source/files/runoptr.sh
```

This will create a lot of files. In fort.xml file you can find a lot of details about the beamline. Especially the positions of elements are saved in this file, which you will need when creating the configuration file for the beamline.

3.5 loop_transoptr.sh

Two steps need to be done for the code to read in the transoptr executable. The first task is to create a *loop_transoptr.sh* file (this chapter) and setup PyOptr (see chapter3.6). Due to inefficiencies in running FORTRAN directly from python, the currently fastest implementation makes use of an external bash script to run the simulation. The code for this execution is called loop_transoptr.sh. An instance of the simulation will generate a folder containing the data.dat file and the optr executable and initialize a sim_condition.txt file. The loop_transoptr.sh bash script will be started which continuously loops the execution of the executable based on states that are written to sim_condition.txt. The environment wrapper executes in 2 folders, one for mode 2 and one for mode 3 and parses the generated fort files to compile the simulation. loop_transoptr.sh will save the PID of the process which started it and terminates once the PID no longer exists. The contents of the file need to be the following. The file needs to be saved in the tuneAI repository in transoptr/beamline_you_are_using/.

```
SIMULATION_FILE='sim_condition.txt'
sim_done_flag='sim_finished'
start_sim_flag='generate_sim'
```

```
init_sim_flag='init_sim'
termination_flag='terminated'
PROCESS_ID=$(cat processid.txt)
OPTR_EXECUTABLE=$(cat executable_path.txt)

echo $init_sim_flag > $SIMULATION_FILE

while :
do
    if [ -d "/proc/$PROCESS_ID" ]; then

        if grep $start_sim_flag $SIMULATION_FILE
        then
            $OPTR_EXECUTABLE
            echo $sim_done_flag > $SIMULATION_FILE
        fi
    else
        break
    fi
done
echo $termination_flag > $SIMULATION_FILE
```

3.6 PyOptr

The PyOptr is written by Paul Jung and used as a python wrapper for transoptr and can get accessed by

```
https://gitlab.triumf.ca/beamphys/pyoptr.git#egg=pyoptr .
```

To make pyoptr run, you need to add the following two lines to your bashrc file

```
export PYTHONPATH=$PYTHONPATH:directory/to/transoptr/pyoptr/
export JUPYTER_PATH=$JUPYTER_PATH:directory/to/transoptr/pyoptr/
```

PyOptr allows overwriting the initial values in the data.dat files and rerun transoptr with the new settings. From pyoptr you load the DataDat class, which has the update_element method to overwrite steerer strength and misalignments. By calling the to_string() method, you can display the actual version of data.dat, to make sure the correct parameters are overwritten.

3.7 Troubleshooting

It is very easy to miss things while setting up a new beamline to train a new agent. The main problem is, that no debugging is possible. Sometimes key dictionary errors in python can give a hint that something is going wrong..

Here is a list to double-check when running into errors:

- Is the total number of variables in data.dat line 8 correct?
- Is the sequence of variables listed in data.dat the same as in the sy.f header?
- Is the syntax in sy.f correct? Are you exceeding the character limitations? Are the + signs for line breaks in the correct positions? Do you finish each line in the sy.f header with a comma?

3.8 Setup of the beamline configuration file

The code runs with the hydra framework. Hydra is an open-source Python framework that simplifies the development of research and other complex applications. The code reads in a configuration file in yaml file format. The main idea of these configurations file is to not have any hard-coded variables in the code and get a convenient overview of beamline and agent parameters. You can find more information about hydra here.
Before being able to test the new beamline, a new beamline configuration file needs to be created and saved into config/env. More information about the configuration files is given in chapter 4.1. In general each element needs to be implemented with its name given in the acc database (chapter 3.1).

3.8.1 General Settings

Figure 8 shows the general settings for the environment configuration file.

```yaml
name: ALTIS_to_FC19
description: 'screenshot_for_the_documentation'
type: beamline

acc_path: altis-altn-agts-ayield

optr_dir: /home/USERNAME/repos/tuneai/transoptr/ALTIS_to_FC19
datadat_path: /home/USERNAME/repos/tuneai/transoptr/ALTIS_to_FC19/data.dat
backup_path: /home/USERNAME/backup/ #mainly used for saving backups on compute canada

nominal_current: 1.0 #nA
current_fluctuation: 0.0

# random offset factor to apply to each steering angle each episode 0.05 = 5%
steering_descrepancy: 0.05

# randomly shfits each action within some ratio of the action bounds
steering_shift: 0.0

# linear increase of the misalignment during the training process
dynamic_misalignment: True

extraction_voltage_pv: ALTIS:BIAS
extraction_voltage_conversion: 1e6 #for data.dat

beamline_wall_width: 2.0 # cm
```

Figure 8: Environment Configuration File: General Settings

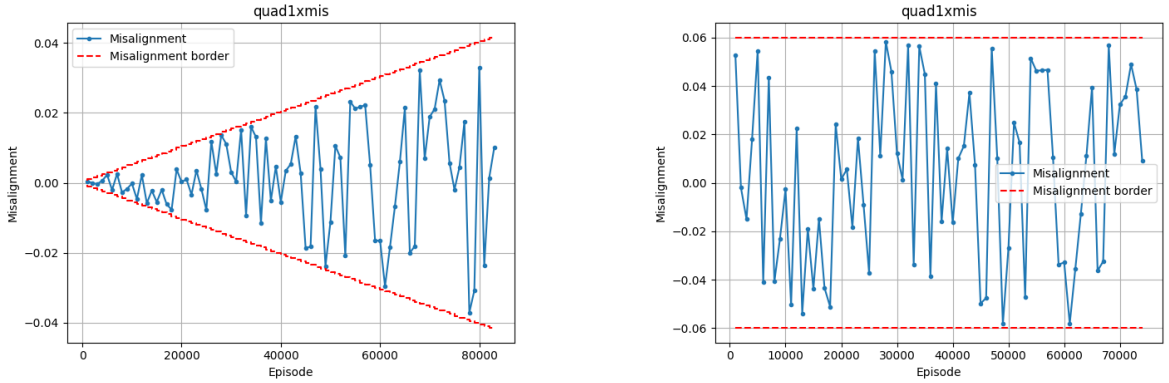
- **name** Name of the beamline. The agent will be saved under runs/name in your repository directory.
- **description** The description is added to the filename under which the agent is saved. I can only encourage you to document there, what you changed or what you intended to try during this training. This helps to keep an overview of your trained agents.
- **type** Needs to be set to beamline.
- **acc_path** The file that needs to be read in by acc to download the data from the database. The filenames for the different beamlines are listed in Table 2. If you implement a new beamline, asking Spencer is the best idea to find out which file to use.
- **optr_dir** Path to the transoptr files generated for this beamline. If you have the same file structure (repos directory in your home folder with the repository saved under tuneai) you can keep the path like this. USERNAME is overwritten in the code with your username on ml1.
- **datapath_path** path to the data.dat file for the correct beamline.
- **nominal_current** Start current at the source. The absolute value doesn't really matter, as only the percentages of transmission at the FC cups are passed to the agent as an input. This

Table 2: acc_path names for different beamlines

Beamline	Acc_path Name
OLIS	ios-mws-hebt1-prague
OLIS to RFQ	ios-mws-hebt1-prague
OLIS to FC5	ios-mws-ile1b-griffin
ALTIS to FC19	altis-altn-agts-ayield

is actually not the best way to do it, as the current from the source is in general unknown. For the tests on the real beamline, "a guess" of the real current needs to be specified to calculate the transmission from the source to the first FC cup. A better approach would be to work with absolute values at FC cups.

- **current_fluctuation** Fluctuation of the current at the source. Not really important, as only relative values are calculated and passed to the agent.
- **steering_shift** Moves the action space of the steerers.
- **dynamic_misalignment** When set to *True*, the misalignment of all elements (Quadrupoles, slits, fringe field at source, source offset and source angle) increases linearly during the training of an agent (see Figure 9. The agent is reaching higher scores when training on small misalignments at the beginning.



(a) Linear increase

(b) No linear increase

Figure 9: Misalignment of quadrupole 1 in x (cm) direction as a function of training episodes. These plots are saved for each element in runs/beamline_you_are_training_on/name_of_the_agent/misalignments

- **extraction_voltage_pv** pv name to download the current extraction voltage of the source. The pv name can be found in the acc database. Ask Spencer if you need help.
- **extraction_voltage_conversion** Conversion from the energy unit of the beam energy specified in data.dat to eV. There is for sure a more elegant way to do the calculation.
- **beamline_wall_width** The name is a bit confusing. The distance from the 0 line to the beam wall is meant, see Figure 10.

3.8.2 Misalignments

The misalignment of the following elements is implemented: Quadrupole, Source, Slits, Electric- and Magnetic Bender and Fringe Fields at the source. They are controlled within the configuration file. The implementation is shown in Figure 11. In bounds, the maximum bounds of individual misalignment sources can be set. Note that the actual misalignment during an episode is a number between min and max. These values are passed to variables. **IMPORTANT:** The variables need to be named, as named in data.dat. Figure 11 shows the misalignment of the

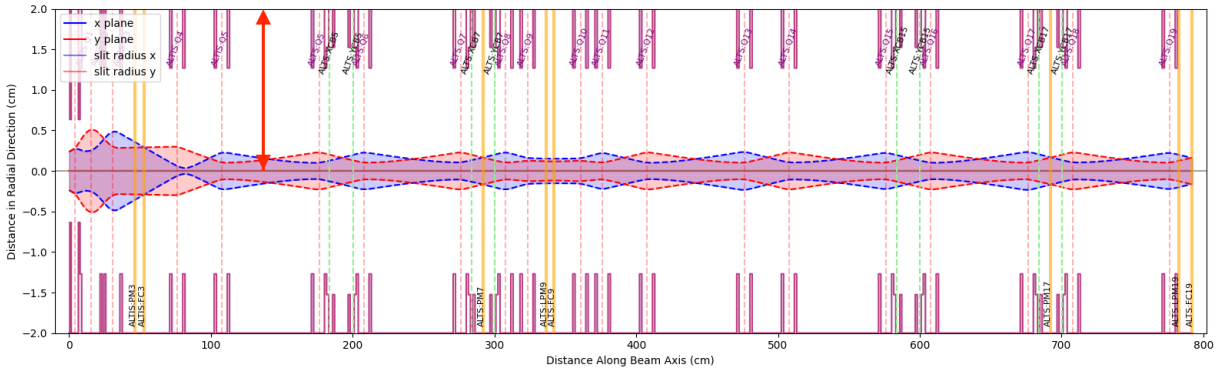


Figure 10: Environment Configuration File: beam_wall_width

source (offsets and angles). Within beamlines, quadrupoles are mounted in groups which lead to a correlation of misalignments among different quadrupole. A group of quadrupoles can be specified as shown in Figure 12. They all get a correlated misalignment (group_quad_mis). It is still possible to add an individual misalignment within a group by in- or decreasing single_quad_mis.

```
misalignments:
  bounds:
    src_position:
      max: 0.15 #cm
      min: 0.0  #cm
    src_angle:
      max: 0.005 #rad
      min: 0.0   #rad
    single_quad_mis:
      max: 0.05 #cm
      min: 0.0  #cm
    group_quad_mis:
      max: 0.01 #cm
      min: 0.0  #cm
  variables:
    xmis:
      max: ${misalignments.bounds.src_position.max}
      min: ${misalignments.bounds.src_position.min}
    ymis:
      max: ${misalignments.bounds.src_position.max}
      min: ${misalignments.bounds.src_position.min}
    xpmis:
      max: ${misalignments.bounds.src_angle.max}
      min: ${misalignments.bounds.src_angle.min}
    ypmis:
      max: ${misalignments.bounds.src_angle.max}
      min: ${misalignments.bounds.src_angle.min}
    quad1xmis:
      max: ${misalignments.bounds.single_quad_mis.max}
      min: ${misalignments.bounds.single_quad_mis.min}
    quad1ymis:
      max: ${misalignments.bounds.single_quad_mis.max}
      min: ${misalignments.bounds.single_quad_mis.min}
```

Figure 11: Environment Configuration File: beam_wall_width

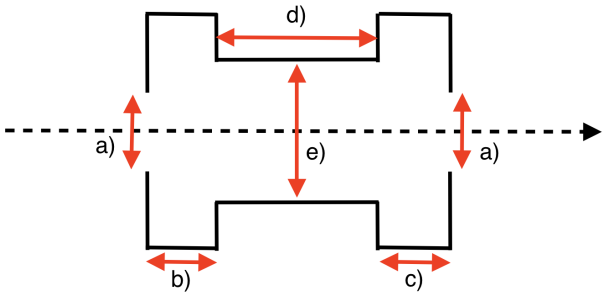
3.8.3 Steerers

The implementation of steerers is shown in Figure 13. Note that the name of each steerer needs to be the same as in the acc database.

– **loc**: Location of the steerer. The location of the steerers is more complicated to determine, as they do not show up in fort.xml, when running transoptr. The location of steerer is saved

```
quad_misalignment_groups:
  group1x: ['quad1xmis', 'quad2xmis','quad3xmis']
  group1y: ['quad1ymis', 'quad2ymis','quad3ymis']
  group2x: ['quad4xmis', 'quad5xmis','quad6xmis']
  group2y: ['quad4ymis', 'quad5ymis','quad6ymis']
```

Figure 12: Environment Configuration File: Quadrupole Groups



(a) Schematic of a steerer

```
steering_elements:
  ALTS:XCB5:
    loc: 183.8609 #cm
    plate_length: 2.54 #cm
    plate_separation: 3.81 # cm
    skimmer_diameter: 3.05 #cm
    skimmer_distance_before_steerer: 0.95 #cm
    skimmer_distance_after_steerer: 0.95 #cm
```

(b) Variables in environment config

Figure 13: Implementation of Steerers into the config file.

in the acc database (xml files in sequence directory). However, the location is only stored with respect of the beginning of a section. There is a workaround by looking for the location of a quadrupole and calculate the offset with respect of that quadrupole in fort.xml. For example:
The location of the quadrupole ALTS:Q6 in the ALTS section file ALTS_2 (`https://gitlab.triumf.ca/hla/acc/-/blob/master/ariel/sequence/ALTS_2.xml`) is 42.5199 cm with respect to the beginning of section ALTS_2. When looking at the fort.xml file in your repository directory for ARIEL to FC19, the location of ALTS:Q6 is 207.7603 cm with respect to the location of the source (0 cm). The offset of the ALTS_2 section with respect to the source is therefore

$$\text{ALTS\_2}_{\text{offset}} = 207.7603\,\text{cm} - 42.5199\,\text{cm} = 156.2404\,\text{cm} \tag{4}$$

To determine the location of steerers in ALTS_2, this offset needs to be added to the location of steerers in the ALTS_2 file. If you are bored you can implement an automatic way to determine this offset and add it to the location of steerers from the acc database :).

- **plate_length**: Plate length of steerer plates. This is necessary for the calculation of the maximum angle the steerer can kick the beam. **d)**
- **plate_separation**: Separation distance of the plates. This is necessary for the calculation of the maximum angle the steerer can kick the beam. **e)**
- **skimmer_diameter**: Diameter of the skimmer plate. **a)**
- **skimmer_distance_before_steerer**: distance of the skimmer to the steerer electrode in opposite beam direction. **b)**
- **skimmer_distance_after_steerer**: distance of the skimmer to the steerer electrode in beam direction **c)**

Note that the steerer plates should in principle also be included into the aperture calculation, however, as the skimmer plates are already taken into account, the steerer plates are neglected. It is maybe a good exercise to include them to get familiar with the code.

3.8.4 Slits

The implementation of slits is shown in Figure 14. Note that the name of each slit needs to be the same as in the acc database. The (misaligned) slits can be seen in Figure 15. In principle

these slits are motorized. The positions of the slits can be controlled by EPICS. Therefore the optimal positions of the slits could also be implemented as an output parameter for the agent. However, the motors are for the time being broken. Additionally, the positions of the slits are not known. They are therefore also included in the misalignment calculation of the beamline.

```
slits:
  MCOL3A:
    loc: 122.779007
    x_lower: 0.05
    y_lower: 0.5
    x_upper: 0.05
    y_upper: 0.5
    misalignment_x: 0
    misalignment_y: 0
```

Figure 14: Environment Configuration File: slits

- **loc** The location needs to be determined as for the steerers.
- **x_lower** Half length width of the slit in x direction.
- **y_lower** Half length width of the slit in y direction.
- **x_upper** Half length width of the slit in x direction.
- **y_upper** Half length width of the slit in y direction.
- **misalignment_x** Misalignment in x direction. When changed, x_lower and x_upper are automatically overwritten.
- **misalignment_y** Misalignment in y direction. When changed, y_lower and y_upper are automatically overwritten.

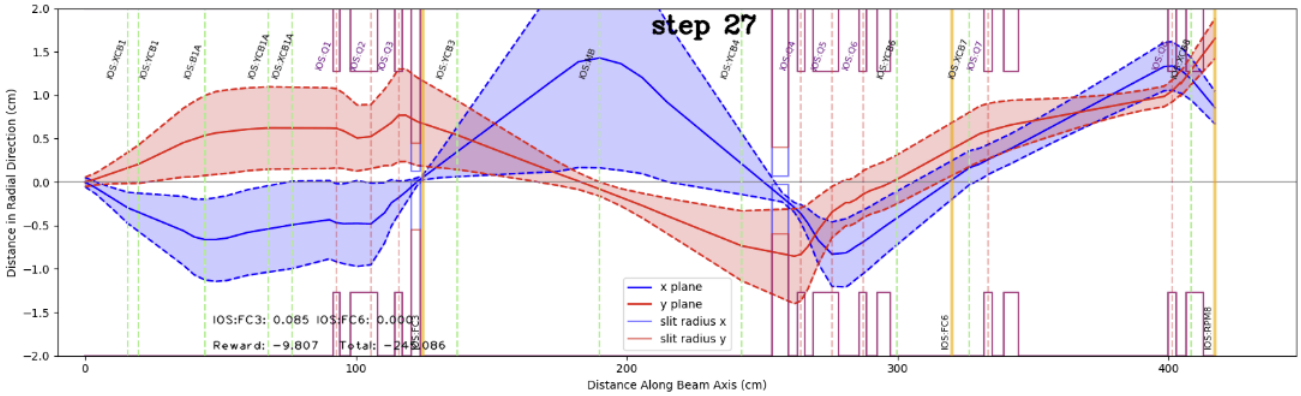


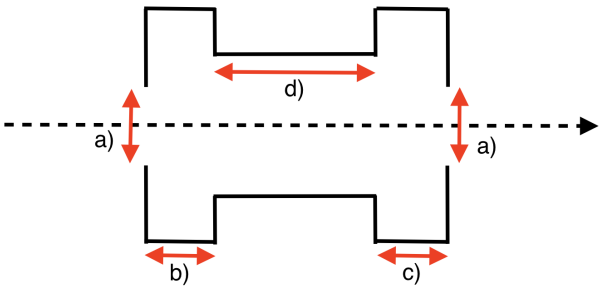
Figure 15: Slits in OLIS (after Q3 and before Q4)

3.8.5 Measurement Devices (PMs and FCs)

3.8.6 Quadrupoles

The implementation of quadrupoles is shown in Figure 16. To protect the quadrupole electrodes from the beam, skimmer plates are installed before and after each quadrupole as shown in Figure 16(a).

- **loc**: Location of the quadrupole. The locations can be found in the fort.xml file, which is produced after running transoptr (chapter 3.4)
- **skimmer_distance_before_quad**: distance of the skimmer to the quadrupole electrode in opposite beam direction **b)**



(a) Schematic of Quadrupoles

```
quads:
  ALTIS:Q1:
    loc: 4.3200
    skimmer_distance_before_quad: 1.84
    skimmer_distance_after_quad: 1.84
    quad_length: 2.54
```

(b) Variables in environment config

Figure 16: Implementation of Quadrupoles into the config file.

- **skimmer_distance_after_quad**: distance of the skimmer to the quadrupole electrode in beam direction **c)**
- **quad_length**: length of the quadrupole electrodes. Note that in the acc database only the effective length is saved. You need to ask for the real length. **d)**
- the variable **a)** in Figure 16 denotes the skimmer diameter. This information is saved in the acc database and automatically pulled.

Note that the quadrupole plates should in principle also be included into the aperture calculation, however, as the skimmer plates are already taken into account, the quadrupole plates are neglected. It is maybe a good exercise to include them to get familiar with the code.

3.9 OLIS Beamlines

Several agents are already trained on different beamlines at ISAC and ready to be tested. The main focus is on beamlines coming from OLIS. Unfortunately, the OLIS beamline is an old beamline, which means that the optical elements are highly misaligned and a lot of steering is needed for transmission. The current coming from the source is low, which means that a beam hitting components of the beamline is not destroying, but it needs beam destructive measurement devices. Taking measurements of the beam current and position takes time. However, due to its high availability OLIS is a good starting for the project to test and learn about trained agents. Figure 17 shows a technical drawing of the beginning of the beamline. In total 3 sources are available (Microwave Source (MS), Surface Source (SS) and Multi Charge Ion Source (MCIS)). The MCIS produces a large B-field, which requires a lot of steering at the beginning of the beamline. The fringe field of the source steers the beam down. For beam coming from SS, it is the opposite. Agents have been only trained for beam coming from MWS, however, for online tests, the more reliable SS has been used. As the beamlines are symmetrical, only polarization of the switching magnet B1A need to be opposite. The first measurement device on the beamline is Faraday Cup (FC) 3, which only measures current. This is challenging for an agent (and operator) as 5 elements need to be adjusted before getting beam on FC 3. There are discussions going on, to add an additional position measurement device (RPM) at the beginning of the beamline. We should push for this idea. MB1 is a magnetic 60° bender. Before and after the magnet, a collimator slit is installed to shape the beam. More information about the slits can be found in chapter 3.8.4. The main OLIS beamline ends with another FC (FC6), several steerers and a position measurement device (RPM 8).

After RPM8, the beam arrives at a crossing (see Figure 18). The OLIS beamline ends south. Going east, the beam continues to RFQ (see chapter 3.9.2). Going north, the beam continues to FC5 (chapter 3.9.3). For both ways, agents have been trained and tested on the real beamline.

You should ask around to get a tour to the beamlines. :)

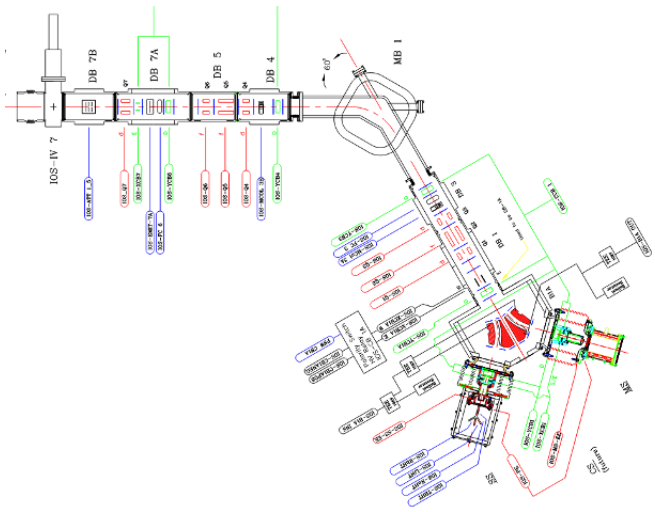


Figure 17: Olis Technical Drawing

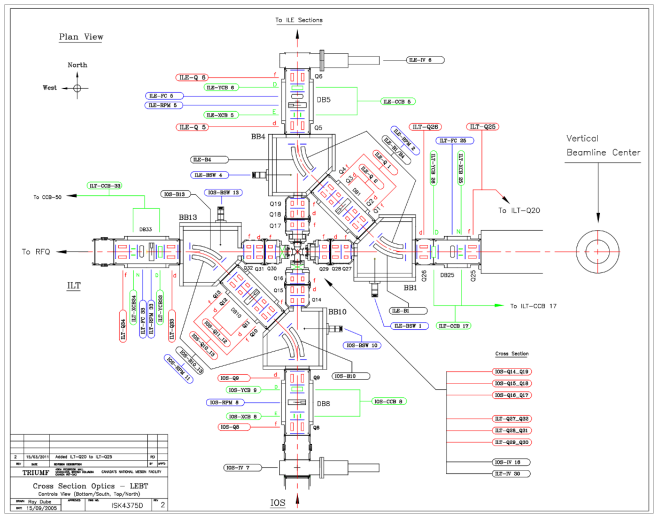


Figure 18: OLIS Crossing Technical Drawing

3.9.1 OLIS

The transoptr simulation for the OLIS beamline can be seen in Figure 19. The position of the beam with beam envelopes (multi particle tracking) is shown as a function of the beamline distance. The green dotted vertical lines show the position of tuneable beam parameters, they are optimized by the agent (Steerers and Magnets). The vertical red lines show the position of quadrupoles. On the edge of the beamline, the skimmer plates of steerers and quadrupoles are marked. Before and after the magnet MB1, a collimator slit is installed in the beamline. Measurement devices like Faraday Cups to measure the beam current and RPMs to measure the beam position are marked in yellow. During the training process, fringe fields, source misalignment, dipole misalignment and the misplacement of all quadrupoles are taken into account.
The OLIS beamline is the start beamline for the project. However, the OLIS beamline is not easy to tune. Due to large magnetic fringe fields at the beginning of the beamline, the beam needs a lot of steering. Additionally, the amount of diagnostics on the beamline is limited. Before reaching the first Faraday Cup (FC3), 5 steering components need to be optimized. A few results of the online tests at OLIS are summarized in chapter . Due to the lack of diagnostics, the simulation of OLIS is extended to RFQ (chapter 3.9.2) and FC5 (chapter 3.9.3) for further tests. Layout plans for all OLIS beamlines can be found here under Beamline Diagrams.

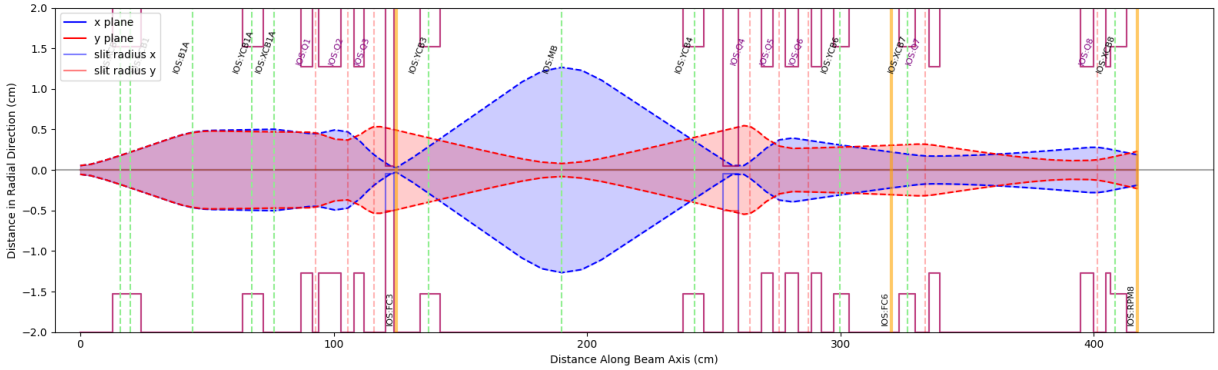


Figure 19: Olis Beamline

3.9.2 OLIS to RFQ

Due to the lack of diagnostics on OLIS, the OLIS simulation is extended to the RFQ. The transoptr simulation is shown in Figure 20. Instead of only having three measurement devices in the beamline, the amount of diagnostics is increased to 11 (+7 RPMs and +1 FC). The section after OLIS is a new beamline, which doesn't need a lot of steering. Therefore, only misalignments of quadrupoles are taken into account to Quadrupole IOS:Q8. The agent also only controls the steerers to IOS:YCB9. All quadrupoles and dipoles after OLIS are assumed to be on axes. I tried to implement the entire beamline (misalignment on all quads/dipoles + all steerers). However, TRANSOPTR can only handle up to 100 parameters (Line 8 in data.dat). For further tests, you should get in contact with Thomas Planche to change this. Another problem with this beamline is, that it is often in operation and therefore not accessible for the project.

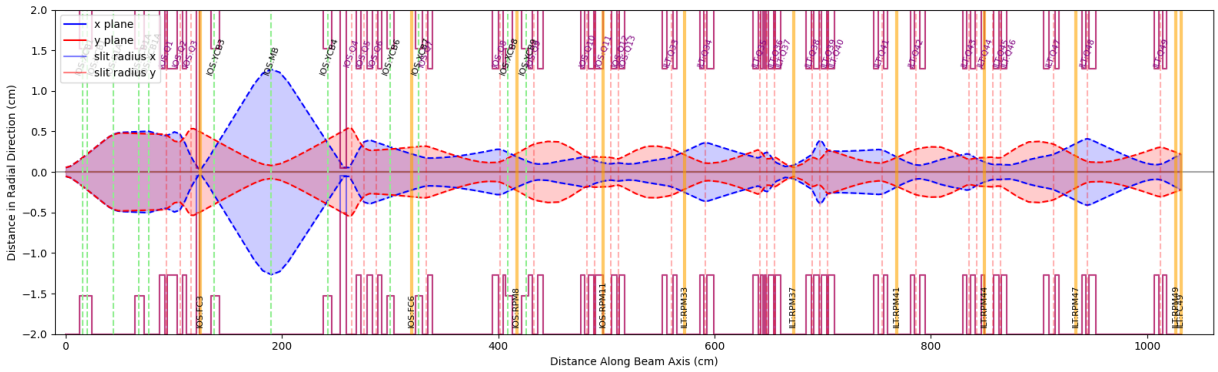


Figure 20: OLIS to RFQ Beamline

3.9.3 OLIS to FC5

Due to the difficulties on OLIS and OLIS to RFQ, the OLIS beamline is extended towards FC5. The main advantage is, that this beamline is also available, when OLIS to RFQ is not available. Additionally, the extension is a straight beamline adding two more measurement devices. The transoptr simulation is shown in Figure 21.

3.10 ARIEL Beamlines

As tests on the OLIS beamline have not really been successful until now, another idea is to move the project to an ARIEL beamline. The beamlines at ARIEL are fairly new and do not need a

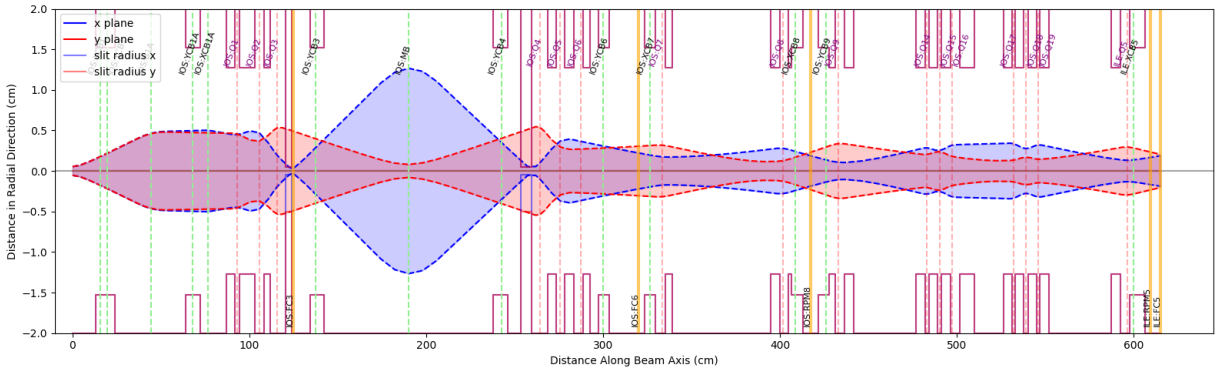


Figure 21: Olis to FC5 Beamline

lot of steering when being tuned. A floorplan of the ARIEL beamlines is shown in Figure 22. The source is called ALTIS and the TRANSOPTR simulation is set up from the source to FC19.

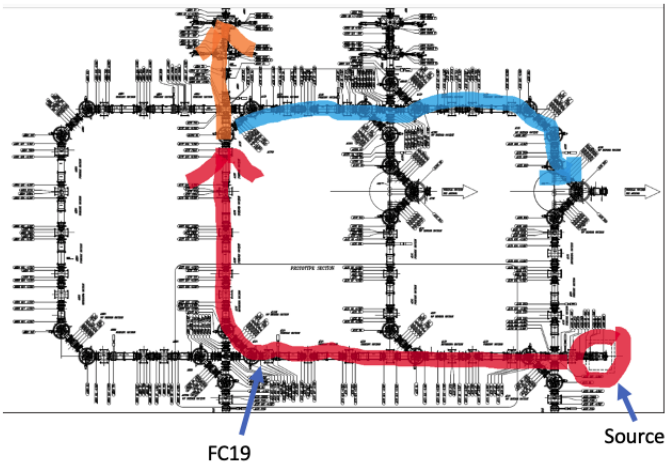


Figure 22: ARIEL beamlines overview.

3.10.1 ALTIS to FC19

The TRANSOPTR simulation for ALTIS to FC19 is shown in Figure 23. Five position measurement devices and 3 current measurement devices are installed in the ring. The position measurement devices are no longer called RPMs, but LPMs and PMs. For the simulation nothing changes, but the profiles of the position monitors need to be processed differently (see section and 6.5 6.5.2). Unfortunately tests were not done due to a lack of time, but in principle everything is set up to run the first tests. We hope to achieve better results than on the OLIS beamlines.

3.11 Test Simulation

To test if everything is done correctly, the test_simulation.py program can be used. The simulation you want to display needs to be specified in your_repository_directory/config/test_sim_config.yaml. By changing the defaults.env variable, the beamlines can be switched. Note, that the name of defaults.env needs to be the same as the environment name in your_repository_directory/config/env/. If everything is done correctly, the beamline is displayed along with buttons to modify steerers

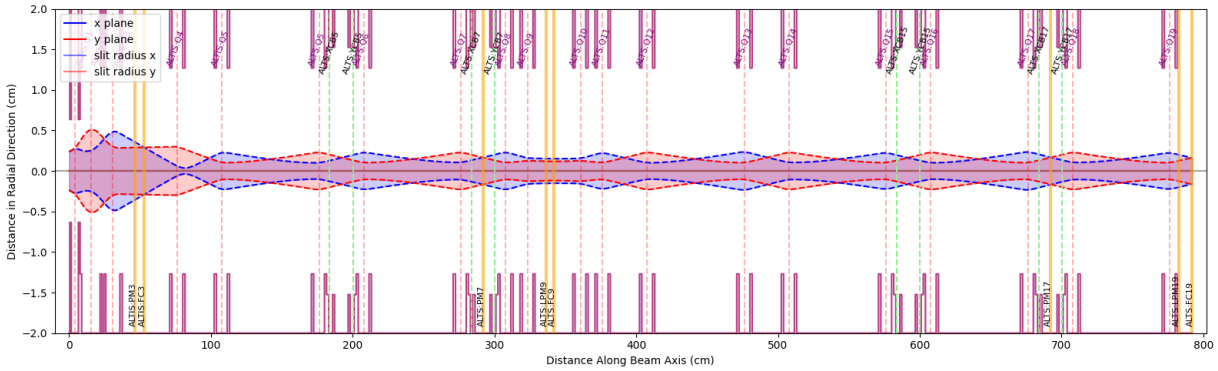


Figure 23: ARIEL beamlines overview.

and the misalignments of elements in the beamline. It's a lot of fun to misalign the beamline and try to steer the beam back on axis :). Every time you click on a misalign a random number is generated between 0 and the set value for this misalignment. Reset sets all steering values back to 0.
This software is useful when trying to determine the maximal bounds for misalignments in the environment configuration file (see chapter 3.8.2)

3.11.1 qdelete_this_folder

When testing the simulation the TRANSOPTR executable is saved for each test in a new folder in qdelete_this_folder. This folder can get quite busy after running a lot of tests. To avoid unnecessary data space usage, you can delete this folder from time to time. Also the results from the validation of agents (see chapter 7) are saved temporally in qdelete_this_folder.

4 Reinforcement Learning on the simulation

This chapter deals with an overview of reinforcement learning algorithms that are used to tune the beamlines. In case you don't have any experience with Reinforcement Learning Algorithms, participating in the following UDEMY lectures is highly recommended.

- Introduction to reinforcement learning: `https://www.udemy.com/course/beginner-master-rl-1/`
- Advanced Algorithms `https://www.udemy.com/course/actor-critic-methods-from-paper-to-code-with-pytorch/`

Don't pay 120$ for each of them! Both lectures are usually on discount for 20$ (or even less) each. Try to open the links in incognito mode and/or different browsers. Basic knowledge in python and Neural Networks is required to follow both lectures. Especially the Advanced Algorithms online class covers state of the art algorithms, that are used for the project. Therefore only a small overview of the available agents is given. The basic principle of reinforcement learning is shown in Figure 24. The algorithm is an interplay of the so called environment (beamline) and the agent. The agent predicts an action A_r which is applied to the environment and the environment sends its state back to the agent along with a reward. Based on that state and the reward, the agent predicts new values which are again applied to the environment. For the project, the overall structure of reinforcement learning is shown in Figure 25. The learning process happens in so-called steps. At the beginning of each step, the beamline is randomly misaligned and random small action to the steerer is applied. After that the beamline gets measured for the first time (Obs_0). The agent gets a certain amount of steps to optimize beam transmission. Each step steerer actions are predicted based on the observation in the step before.

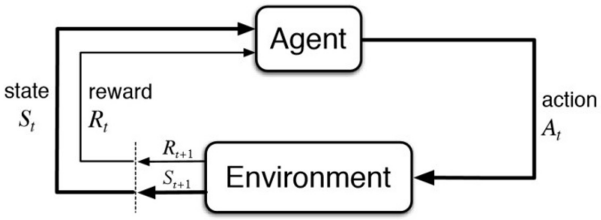


Figure 24: Reinforcement Learning Strategy

For exploration, noise is added to those actions (see chapter 4.4. After that, a new observation is measured based on which a reward is calculated (see chapter 4.3). All steerer actions, rewards and beamline observations are saved and used at the final steps to optimize the agent.

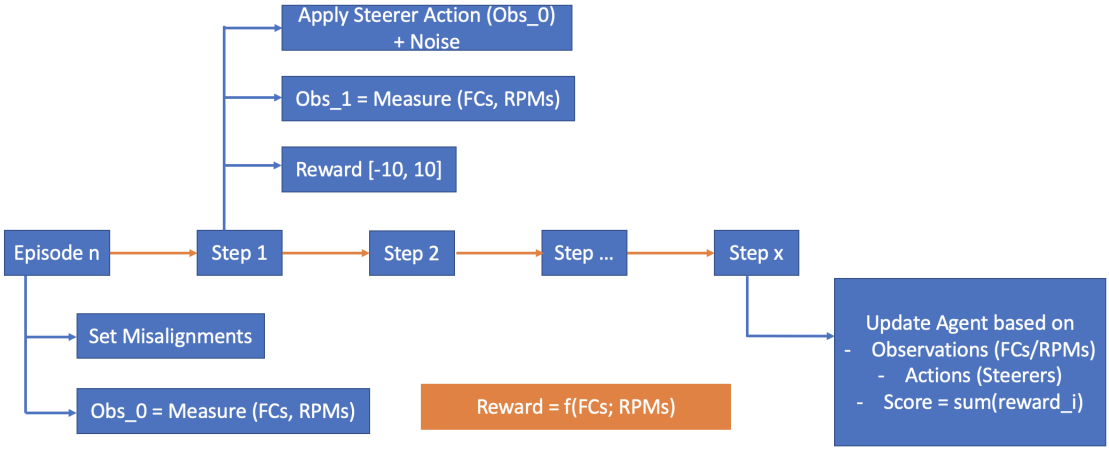


Figure 25: Reinforcement Learning Strategy

4.1 Agent Configuration File

For each agent a configuration needs to be written and saved under config/agent/. The agent you want to use needs to be selected in config/config.yaml by changing the defaults.agent variable. The name of the variable needs to be set to the name of the yaml file.

4.2 Environment

The environment, the interaction of the agent with the environment and TRANSOPTR is done in simulation/Beamline.py. The Beamline class provides all functionalities to misalign the beam, update parameters (misalignments + steerer values) in data.dat, calculate the aperture of the beamline and measure virtually the beamline (RPMs and FCs).

4.2.1 Beam Position (RPM, LPM and PM)

Within the training on the simulation, no differentiation is made along PMs, RPMs or LPMs. The position of the beam after applying misalignments and steerer strength is saved in the self.s variable. The position of the beam is evaluated at the position of the position measurement device.

4.2.2 Beam Current (FC)

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 and ARIEL. 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:

$$\Theta_{i,q} = \frac{1}{2}\left[\mathrm{Erf}\left(\frac{w_i - \mu_i}{\sqrt{2}\sigma_i}\right) + \mathrm{Erf}\left(\frac{w_i + \mu_i}{\sqrt{2}\sigma_i}\right)\right] \tag{5}$$

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:

$$T_{k,q} = \prod_{i=0}^{k} \Theta_{i,q} \quad \text{(1D)} \tag{6}$$

A nominal current I_0 of 1.0 + 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:

$$I_k = I_0 \cdot T_k \quad \text{(Beam)}. \tag{7}$$

The class function *get_erf_losses* has an attribute *show*. The flag is set by default to False. However, if you want to see, where the beam losses appear in the beamline, you can set this flag to *True* as shown in Figure 26.

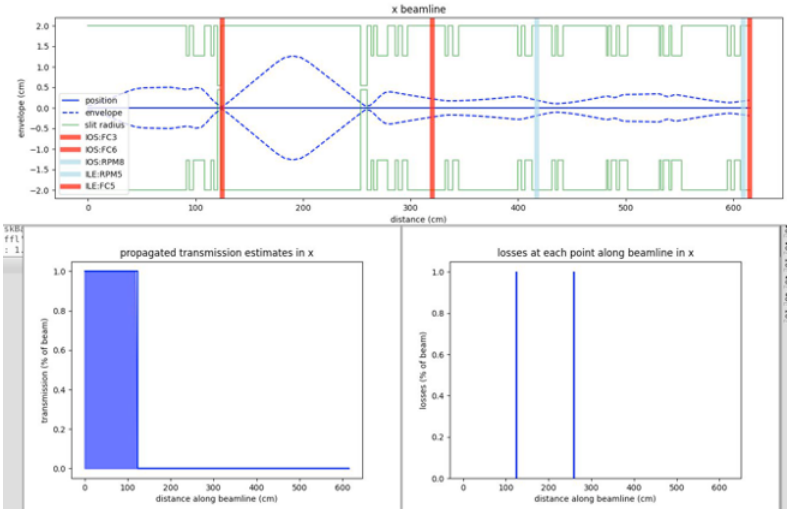


Figure 26: Transmission losses in the simulation.

4.3 Reward

The files for the reward calculation can be found in env/reward. To choose which reward function you want to use, change the reward_fxn variable in the agent configuration file to the name of the reward function specified in env/reward/.

4.3.1 MeasurementReward

MeasurementReward uses only measurable information from the FCs and RPMs. Separate terms are constructed to incentivize high transmission at FCs and to penalize large offsets at the RPM.

Additionally, it is possible to weight the reward measured by the FCs and RPMs with the *fc_reward_weight* $\in [0., 1.]$ variable in the agent config file.

The transmission ratios at the FCs are weighted along the FCs and scaled

$$R_{\text{FCs}} = 2 \cdot \text{MAX\_REWARD} \cdot \text{fc\_reward\_weight} \cdot \sum^{\#\text{FCs}} \frac{\text{ratio}_i}{\#\text{FCs}} \tag{8}$$

where MAX_REWARD is the maximum reward return (10) per step. The reward contribution for the RPMs is given by

$$R_{\text{RPMs}} = 2 \cdot \text{MAX\_REWARD} \cdot (1 - \text{fc\_reward\_weight}) \cdot \text{rpm\_ratio}, \tag{9}$$

where rpm_ratio ratio is given by

$$\text{rpm\_ratio} = \frac{1}{2} \sum_i^{x,y} \left(\sum_j^{\#\text{RPMs}} \min(\text{pos}_{i,j}^2 \cdot \beta, 1) \right) / \#\text{RPMs} \tag{10}$$

The scaling constant is used to appropriately scale episodic rewards: $\beta = 25$ such that a RPM measurement of magnitude 0.2 cm or more will give $P_q^{\text{RPM}} = -10$ The final reward is given by

$$R_{\text{Final}} = \text{MAX\_REWARD} \cdot (1 - 2 \cdot \text{fc\_reward\_weight}) + R_{\text{FCs}} - R_{\text{RPMs}} \tag{11}$$

4.3.2 MSEReward

The MSE reward is very similar to Measurement Reward. Instead of only taking into account the position of the beam at the RPMs, it uses the entire position measurement from TRANSOPTR to calculate rpm_ratio.

$$\text{rpm\_ratio} = \frac{1}{2} \sum_i^{x,y} \left(\sum_j^{N} \min(\text{pos}_{i,j}^2 \cdot \beta, 1) \right) / N \tag{12}$$

where N is the length of the position array. The idea behind this was, that it shouldn't matter, if the agent sees directly the entire position array or only the positions at the RPMs as the reward is not used when predicting the values on the real beamline. However, the performance of both reward functions is comparable.

4.3.3 new_reward_2022

The new_reward_2022 function uses an easier approach to calculate the reward. Starting on a reward of -10, each measurement device contributes equally to the reward, e.g if 5 devices are installed in the beamline each device can add up a reward of $20/5 = 4$. The current at each FC cup, is always divided by the current measured at the FC before (for the first FC, it's the source current), which gives a number between 0 and 1. Each contribution of a FC cup is given by

$$\text{FC}_{\text{reward}} = \text{max\_reward\_per\_device} \cdot \text{percentage\_of\_transmission}. \tag{13}$$

The contribution of a position measurement to the final reward is 0 if

$$\left| \frac{1}{2} \left(\text{pos}_x + \text{pos}_y \right) \right| > 1. \tag{14}$$

where pos_x and pos_y denote the measured position of the beam in x and y, respectively. Else the reward is calculated as

$$\text{RPM}_{\text{reward}} = -\text{max\_reward\_per\_device} \cdot \left| \frac{1}{2} \left(\text{pos}_x + \text{pos}_y \right) \right| + \text{max\_reward\_per\_device}. \tag{15}$$

which leads to a maximum reward when the beam is centered $\left|\frac{1}{2}\left(\text{pos}_x + \text{pos}_y\right)\right| = 0$. During training, it was noticed that sometimes the agent predicts for a steerer x a maximum negative angle, while the following steerer in x predicts a maximum positive angle. This behaviour we are penalizing by subtracting 5 from the final reward, if the difference between two consecutive steerer angle predictions is higher than a certain threshold. (This threshold is still hard coded, this should be moved to the agent config file). A second idea is to introduce a weight factor for the FC, to see if this changes the training performance.

4.4 Exploration Noise

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

$$\sigma_m = \begin{cases} \sigma_0 \cdot (\frac{\sigma_0}{\sigma_f})^{-n/N} & \text{if } m \leq T \\ \sigma_f & \text{if } m > T \end{cases} \tag{16}$$

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

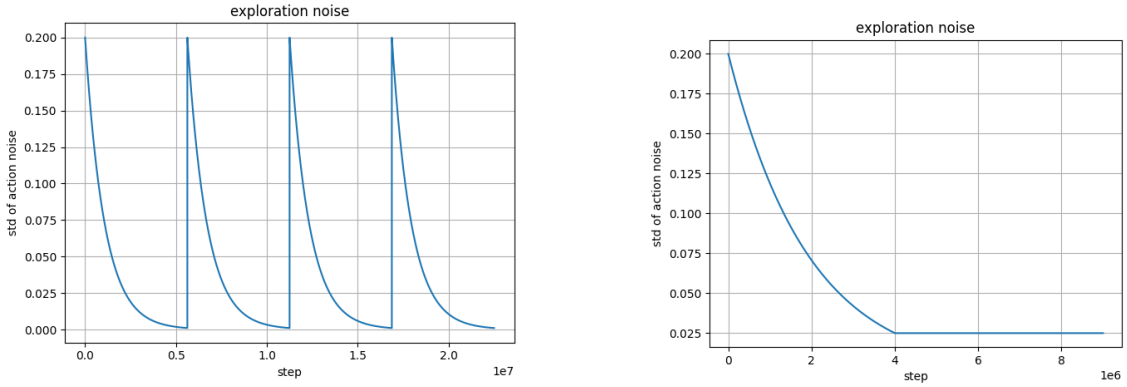
Each selected action a_i is added with some noise $a_i \leftarrow a_i + \nu(\sigma_t) * a_{max}$ where a_{max} is the action bound and $\nu(\sigma_m)$ samples a random normal noise with standard deviation σ_m. the exploration noise strategy can be changed within the RDPG.yaml configuration file. Figure 27 shows the parameters for the noise strategy.

```
25  strategy:
26    start_std: 1.2
27    end_std: 0.001
28    repetition: 2
29    reset_noise: True
```

Figure 27: Noise Strategy Settings in RDPG.yaml

- **start_std** Starting Standard Deviation at episode 0 and step 0.
- **end_std** End value for the standard deviation. If repetition > 1 and reset_noise == True, the standard deviation is reset to start_std
- **repetition** Noise repetition cycles during training.
- **reset_noise** Noise resets if σ = end_std.

In Figure 28, two different exploration noise strategies are shown. Both Figures show the standard deviation as a function of training steps. In Figure 28(a), the noise is reset after reaching end_std. In Figure 28(b) the noise stays constant after the exponential decay.

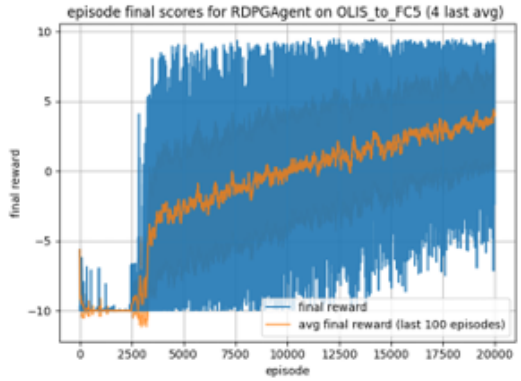


(a) Noise repetition = True

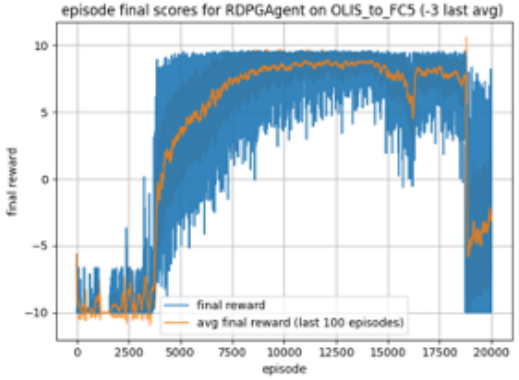
(b) Noise repetition = False

Figure 28: Exploration Noise Strategies for two different trained agents.

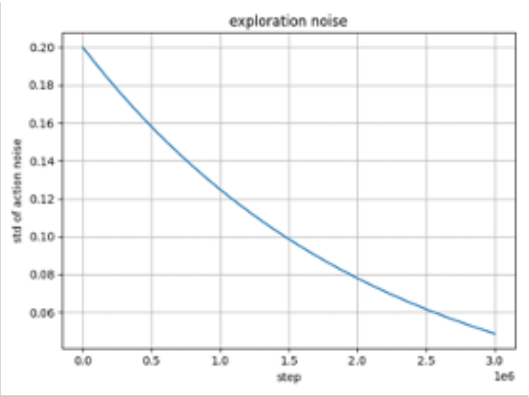
The noise exploration is crucial for the training performance of agents. The training performance is inverse proportional to the decaying standard deviation of the noise as shown in Figure 29. Both agents are trained on the same settings, only the decay of the standard deviation is slower in the left column. The score at episode 20000 for the slow decay is 4, while this value is already reached at episode 5000 for the right column. The best results have been achieved by resetting the noise three times, e.g four decay cycles (see chapter 7.2).



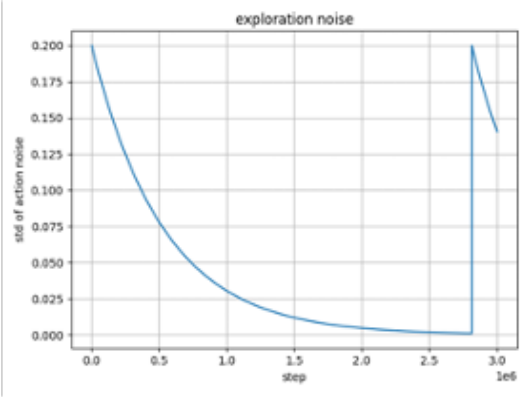
(a) Noise repetition = True



(b) Noise repetition = False



(c) Noise repetition = True



(d) Noise repetition = False

Figure 29: Influence of the noise on the agent training performance.

4.5 Agents

4.5.1 RDPG (Recurrent Deep Deterministic Policy Gradients) Agent

Deep deterministic policy gradients (DDPG) implements an actor-critic architecture which has been shown to be effective for high dimensional and continuous action spaces. The addition of a long short-term memory (LSTM) network to the actor and critic networks forms the architecture for recurrent deep deterministic policy gradients (RDPG). 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. We utilize the architectures from

- `https://arxiv.org/pdf/1512.04455.pdf`
- `https://arxiv.org/pdf/1509.02971.pdf`

and develop a RDPG agent using PyTorch. 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. 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.

4.6 cleanup_runs.py

When running tests with agents, a lot of files are created. To automatically delete them, run cleanup_runs.py. You need to specify the beamline name *beamline_name*. All runs will be deleted which do not contain the misalignment directory (e.g which trained less than 100 episodes).

5 Compute Canada

Compute Canada is a great tool to drastically decrease training time.

5.1 Account Setup

To create an account on Compute Canada, register at `https://ccdb.computecanada.ca/`. You need to register as a sponsored user with Richard Baartman CCRI: haw-882-01.
Multiple Clusters are available. However, until now, only the Narval and the Cedar cluster have been used to train agents. In the following two chapters, the procedure of training agents on the cluster is explained.
To modify the code on ComputeCanada clusters, use again a sshfs connection to ComputeCanada. This is also useful, for monitoring the progress of training. Both clusters, Narval and Cedar use SLURM for job scheduling. This means, that you cannot start right away your training, but the training gets queued and starts, when resources are reserved for your job. However, on both clusters you have the option to start an interactive session for 1 hour, to test and run the code directly. This session automatically gets killed after 1 hour. Once your code starts running, you have no option to monitor the progress of your training. Therefore, data gets additionally saved in your backup_directory which you specified in the environment configuration file. The tuneAI contains bash scripts which help you to start a job on both clusters.

5.2 Narval

Before being able to start the training, you need to copy the container to the Narval cluster. You can copy the container by running

```
scp /fast_scratch_1/triumfmlutils/containers/container_beamrl_v2
    .0.xx.sif your_coca_user_id@narval.computecanada.ca:/home/
    your_coca_user_id/projects/def-baartman/your_coca_user_id/
```

After that, copy the PyOptr files to compute canada by running

```
scp -r /path/to/pyoptr/ your_coca_user_id@narval.computecanada.ca
    :/home/your_coca_user_id/path/where/you/want/to/save/pyoptr/
```

Don't forget to add the pyoptr paths to the bashrc on compute canada:

```
export PYTHONPATH=$PYTHONPATH:/path/to/pyoptr/
export JUPYTER_PATH=$JUPYTER_PATH:/path/to/pyoptr/
```

The computing nodes on narval do not have access to the internet, which means that the code needs to be copied from your home directory to the computing node. By running

```
git clone your_gitlab_id@gitlab.triumf.ca/beamrl/tuneai.git
    tuneAI
```

you clone the tuneAI repository in tuneAI directory. For easier access to gitlab you can create a gitlab token. Then you can copy the repository by running

```
git clone https://gitlab-ci-token:your_token_id@gitlab.triumf.ca
    /beamrl/tuneai.git tuneAI
```

without using a username or password. Now it is time to edit and recheck the configuration files except for the hyperparameter you want to change. The bash script to run a training on ComputeCanada is called run.sh and is saved in the tuneAI repository. A few variables need to be changed

- **submitdir** The directory in which you saved the tuneAI repository
- **hyperparameter_variable** Overwrite the variable you want to perform hyperparameter tuning on. If the variable is saved in the agent config file load the variable with agent.variable_name = ... If the variable is saved in the environment config file with env.variable_name = ...
- **beamline_name** Name of the beamline you want to train the agent on. Needs to be the same as the config environment file.

The next thing you need to know is how long your code will approximately run. In tuneAI repository is a bash script called start_interactive_shell_narval.sh. Call the script by running

```
bash start_interactive_shell_narval.sh
```

which opens an interactive session on the narval cluster. Sometimes you need to wait until the session is started, it depends how busy the cluster is. An interactive session is only active for one hour but allows one to immediately start a job. By running

```
bash run.sh
```

the training should start. After the replay buffer is filled, the program tells you when it is approximately done. Once you have a rough estimate of the run time you can exit the interactive session. Change the *#SBATCH –time=hh:mm:ss* variable in run.sh to the estimated run time of the code (+ a buffer). Additionally, change the mail-user variable to your email address. You get an email, when your job starts, ends or fails. By running

```
sbatch run.sh
```

your job gets scheduled. Your job gets a unique id. When your job starts, a file called slurm-your_job_id.out appears in your home directory. The file contains the terminal output of the running job. However, you don't have direct access to the node where the job is running. Therefore a backup of the files is regularly saved in the backup directory in your home directory. When the training is done, all contents are saved automatically in the multirun directory in your home directory. To get an overview of your submitted jobs, you can run

```
sq
```

To cancel a job, run

```
scancel job_id
```

By running

```
scancel -u your_coca_user_id
```

all jobs are canceled. To get an estimate of when your jobs are scheduled to start, run

```
sq --start
```

Sometimes the waiting times for Narval can take some days, especially if you already trained a lot of agents. Switching to Cedar is then a good idea.

5.3 Cedar

Running jobs on Cedar is similar to running jobs on Narval. However, Cedar does not allow to submit jobs from your home directory. Follow all steps from Narval instructions. To start an interactive session run

```
bash start_interactive_session_cedar.sh
```

Once your code is setup and you determined the run time you need to copy the repository and run.sh into the scratch directory

```
scp -r tuneAI run.sh /scratch/coca_user_id/
```

Change directory to /scratch/coca_user_id/ and run

```
sbatch run.sh
```

to start the training.

6 Tests on the real beamline

6.1 TuneAI Web Application

With the TuneAI Web application you can collect data from the beamline and send voltages for steerers and magnets to their power supplies. The web pages can be accessed:

- Tune AI Web Application for OLIS: `https://beta.hla.triumf.ca/degaussing/tuneAI`
- Tune AI Web Application for ARIEL: `https://beta.hla.triumf.ca/degaussing/tuneAI/altis`

You won't be able to collect data and send voltages to the beamline from your own computer. To perform tests on the real beamline you need to go into the ISAC control room (room 250). Ask Spencer Kiy, Tiffany Angus or the operator in the control room for advice. You need to specifically ask for

- Extraction voltage of the beam at the source as set in data.dat (1 eV $\widehat{=}$ 1 V) This is extremely important for the angle to voltage conversion for the steerers.
- all quadrupoles at theory values (as specified in data.dat)
- Same source as on which you trained on
- Same isotope as on which you trained on (this is only important as some isotopes cannot be cleanly produced)
- Ask for a pre tuned beamline and how to restore the steerer values in case you want to go back to a good tune
- The olis beamline uses grids to reduce beam current on the beamline. If they are in the beamline, the RPM profiles show an interference pattern. Ask the operator how to remove the grid, you don't want to have them in.

A screenshot of the webpage for the ARIEL beamline is shown in Figure 30. The predicted values of the agent and sent by the predictor interface (see chapter 6.2) are displayed in the table. By clicking *Load steerers above to EPICS* those voltages will be applied to the beamline. To collect data from the beamline click first on ALTIS to ALTW which will open a dropdown menu. You can only collect data from the Faraday Cups (which is faster) or data from all measurement devices in the beamline. I recommend to only collect data from FCs, when transmission is really bad and you want to save time, as the agent expects input from all devices if transmission is higher than a certain threshold.
When the jaya server is restarted, the variables below *Beam Properties* turn to "*INITALIZED*" as shown in Figure 30. For the ARIEL beamline, the "*INITALIZED*" values need to be replaced

by the values as given in Table 3. All values need to be inside quotes. Note, that the beam properties field is a normal text field. You can replace all initialized by their proper values.
After the measurement is done, the data gets saved to disk for 24 hours. The agent automatically reads the latest measurement. However, on the website, there is an option to store the collected data from the measurement devices to disk. After clicking the button below the webpage, a measurement ID is displayed that you need to write down. In chapter 6.2, it is explained how to load the measurement. Until now this option was mainly used, to debug the predictor interface. However, in future training on real data might be an option. If you have questions about the TuneAI Web Application, contact Spencer Kiy.

Table 3: Beam properties for the ARIEL beamline.

PV Name	Value
ALTIS:BIAS	30.0
ALTIS:CHARGE	1
ALTIS:EXPERIMENT	DEVELOPMENT
ALTIS:ISOTOPE	85Rb

Table 4: Beam properties for the OLIS beamline. (Ask Spender about the values)

PV Name	Value
HEBT:CHARGE	????
HEBT:ENERGY	????
IOS:BIAS	????
IOS:CHARGE	????
IOS:EXPERIMENT	????
IOS:ISOTOPE	????
MEBT:CHARGE	????

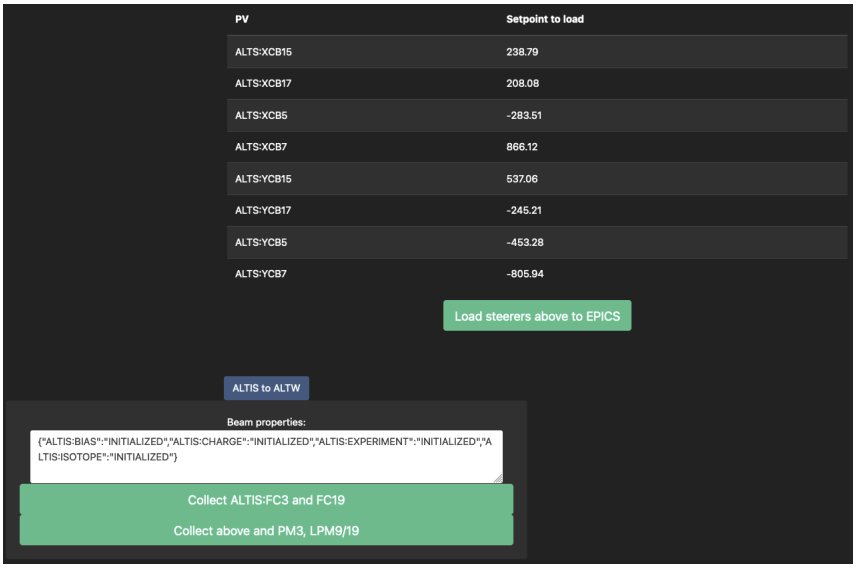


Figure 30: Screenshot of the ALTIS Web Application. The application for OLIS looks similar.

6.2 Predictor Interface

The predictor interface is started by calling *predictor_interface.py* and used to test and validate trained agents on the real beamline. It automatically reads the latest measurement saved in EPICS and processes the data (RPMs, PMs, LPMs, FCs and Steerer). To load an agent, you

need to set the agent_path variable in predictor_interface to the agent you want to use to predict steering values. Figure 31 shows the graphical interface for applying the agent to the real beamline. In the predictor menu, the first thing you are asked for is to process a measurement, which appears in the measurement menu. The data from the steerers, RPMs and FCs is displayed. Sometimes the measurement from the RPM is strangely processed as shown in Figure 32. The determination of the x peak in RPM8 is for example obviously wrong. By selecting the measurement in Measurements, you can edit the value in the Value Box and update it by hitting Update Record. You should get in touch with Thomas Planche to fix the processing of RPM profiles, he is responsible for the software. By clicking on IOS:RPM8 you can undo a measurement of an RPM, if no reliable spectra can be seen in the Beam Simulation interface window. Another important value to keep track of is the ratios measured to the FCs. In the config file for the predictor interface you need to specify a rough value for the current which comes out of the source. This is actually a bad method to process the data from the FCs, as the current from the source is unknown. A more realistic approach would be to train the agent with absolute currents. However, starting from a good tune, I always added a few nA to the measured current at the first FC as a starting current from the source. Make sure, that the ratio measured at the first cup is a number between 0 and 1 (not as in Figure 31. Once the data is correctly processed you can predict the values for the steerers in the Predictor window. the predicted voltages appear in the Predicted Steering window. After applying steering, the voltages are sent and displayed on the tuneAI webpage. To apply the voltages to the beamline, click on *Load steerers above to EPICS*.

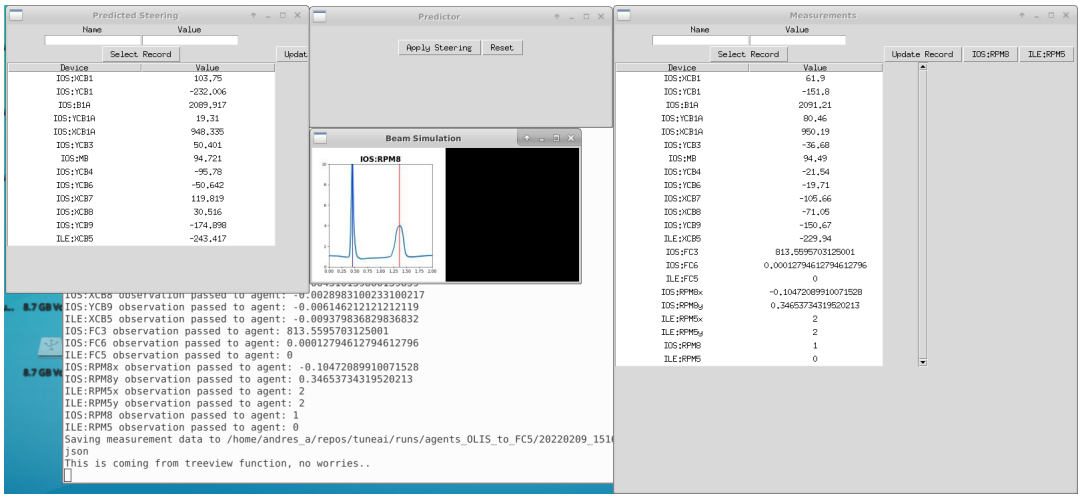


Figure 31: Graphical Interface to predict and apply steerer voltages on the real beamline.

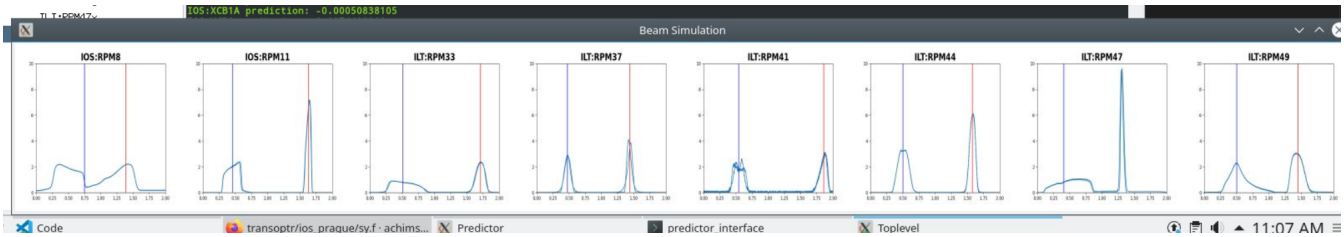


Figure 32: Failed processing of RPM profiles.

The predictor interface automatically generates plots in agent_directory/measurements/ and saves a lot of data for post analysis. It is convenient to keep track of predicted angles, position measurements and measured currents at the FCs.

6.3 Steerer: Angle - Voltage Conversion

The script for angle to voltage conversion can be found in utils/AngleConverter.py. The deflection angle $\Delta\theta$ of an electrostatic steerer is given by

$$\Delta\theta = \frac{\Delta V}{2V_E} \cdot \frac{L + \frac{d}{2}}{d} \tag{17}$$

where ΔV denotes the voltage difference between the steerer plates, V_E the extraction voltage at the source, L the steerer plate length and d the distance between the steerer plates. A full derivation of the formula is given here.

6.3.1 The TRANSOPTR Coordinate System

The coordinate system, that the TRANSOPTR simulation uses is shown in Figure 33 and Figure 34. The convention is

- Positive z is pointing along the beamline axis, from source
- Positive x is left, when facing direction of $+z$
- Positive y is up (out of the page in Figure 33)

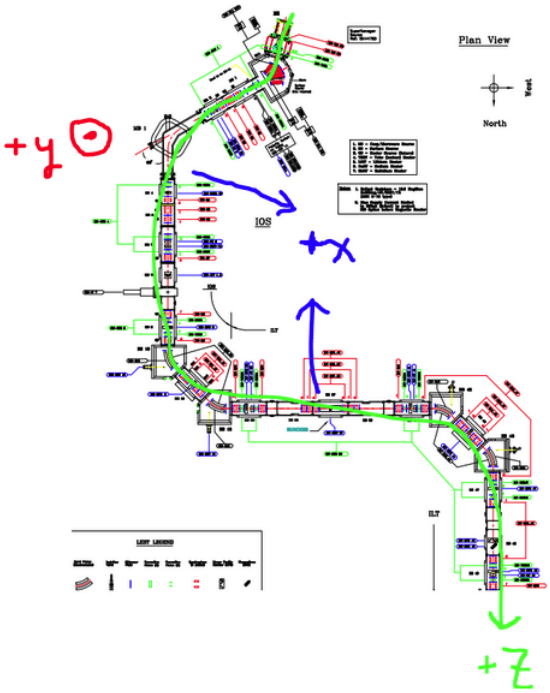


Figure 33: OLIS coordinate system.

6.3.2 Steerer in ARIEL

Each steerer at ARIEL is connected to a bipolar power supply, e.g. multiple steerer can be adjusted individually without adjusting common plates. When 1000 V is predicted, you have one electrode at 500 V and the opposite at −500 V. The direction in which the steerer steers the beam is defined in the inc_steerers variable in the acc_database. inc_steerers = neg_x/y means positive voltage on the steerers steers the beam to -x (east) and -y (down) and pos_x/y vice versa. Pos_x/y coincides with TRANSOPTR coordinate system described in chapter 6.3.1.

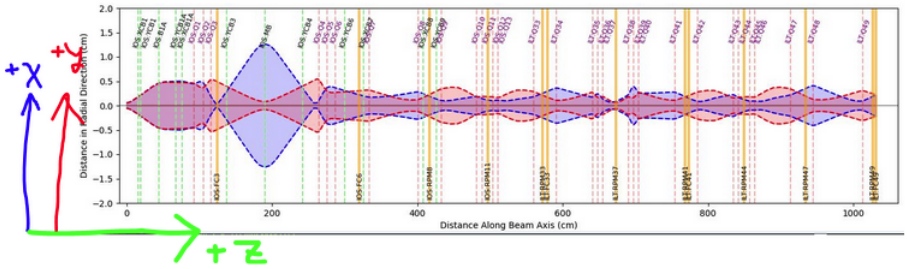


Figure 34: OLIS coordinate system.

6.3.3 Steerer in OLIS

The steerers in OLIS are not connected to individual powersupplies, but are connected to so-called common-plates which are connected to multiple steerers. The common plate CCB1 is connected to IOS:XCB1, IOS:YCB1, IOS:YCB1A and IOS:YCB3. Only a single plate for these steerers can be individually adjusted and a compromise for the common plate needs to be found. The convention for steering components in OLIS is defined by

- increasing voltage of AngleIncrease plate OR decreasing voltage of AngleDecrease plate steers the beam Left/Up (+x/+y)
- For electrostatic steerers, decreasing voltage of steerer plate will steer the beam to +x or +y
- Positive angle in TRANSOPTR == steers to +x/y direction == decrease voltage of steerer plate
- For electrostatic benders, dipoles and deflectors that bend in the +x direction, increasing the field strength (any knob: increasing either of the plates or the MB current is an increase of the field strength) steers beam in the +x direction
- For electrostatic benders, dipoles and deflectors that bend in the -x direction, increasing the field strength steers beam in the -x direction

The information about AngleIncrease and AngleDecrease Plate is saved in the acc database as shown in Figure 35. The inc_steers variable is neg_x which makes it an AngleDecrease Plate. The common plate is therefore automatically an AngleIncreasePlate. For practical reasons the common plates are all set to 500 V. When predicting the voltages of the steerers

```
280   <element id="IOS:XCB7" type="ecb" s="2427.19*mm" l="0.0*mm">
281     <layout x="1156.8052*mm" y="68226.387*mm" z="7906.868*mm"/>
282     <epics>
283       <setpoint pv="IOS:XCB7:VOL" unit="V" min="0.0" max="1000.0" inc_steers="neg_x" type="steerer"/>
284       <setpoint pv="IOS:CCB4:VOL" max="1000.0" min="0.0" unit="V" type="common"/>
285     </epics>
286   </element>
```

Figure 35: Steering implementation in acc.

6.4 Magnetic & Magnetic Bender: Angle - Voltage/Current Conversion

The script for angle to voltage conversion can be found in utils/AngleConverter.py. The benders are driven by current (for magnetic benders) or voltage across the plates (for electrostatic). In each case, the bend angle is proportional to the bender strength. Thus, if bend is 60° and the electric field is changed by .1%, the deflection is 45/1000 degrees.

The electric bend depends on the electric field, not the voltage. So if you change the voltage of only one of the two electrodes, the effect is half. Typically, the electrodes are at +V and -V.

Then the electric field is the voltage difference, +V-(-V)=2V, divided by the distance s between them.
For electrostatic benders the change of deflection angle is given by

$$\Delta\theta = \frac{\Delta V}{V_0} \cdot \alpha_{\text{bend}} \tag{18}$$

where ΔV denotes the change of voltage with respect to the initial voltage V_0. For magnetic benders, the change of deflection angle is given by

$$\Delta\theta = \frac{\Delta I}{I_0} \cdot \alpha_{\text{bend}} \tag{19}$$

where ΔI denotes the change of current with respect to the initial current I_0. More information can be found here here.

6.5 Beam Position Measurements

6.5.1 RPMs

To monitor the position of the beam in OLIS, RPMs are installed in the beamline. The code repository to analyse the RPM profiles is already installed inside the container. The repository can be found here. An example of an rpm profile is shown in Figure 36.

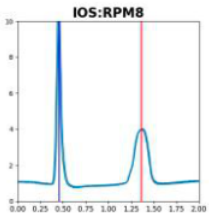


Figure 36: Processed RPM profile.

The left peak refers to the position in x, the right peak in y direction. The units on the x axis are in inches. The unit in y direction scales with the beam current, however, it is not possible to calculate the beam current with the data. The peak of the profile in x direction is shifted by 0.5 inch, in y direction by 1.5 inch. The profiles of the RPMs have a polarity in x and y direction.

Polarity of 1: $+x/y$ centroid measured on TRANSOPTR = $+x/y$ centroid measured on beam as shown in Figure 37.

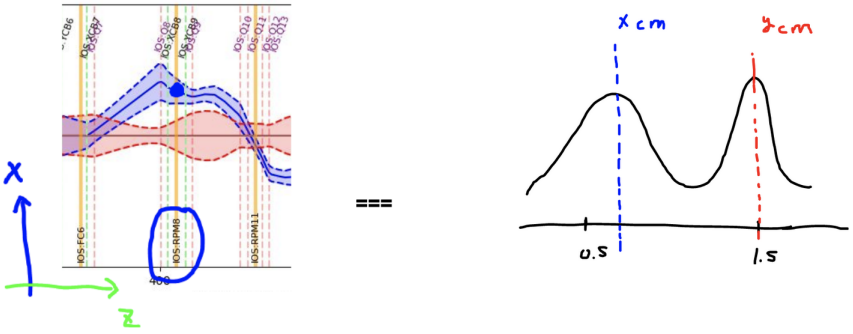


Figure 37: RPM profiles with positive polarity.

Polarity of -1: $+x/y$ centroid measured on TRANSOPTR $= -x/y$ centroid measured on beam as shown in Figure 38. Note, that the peak moved to positive direction, while the position is negative.

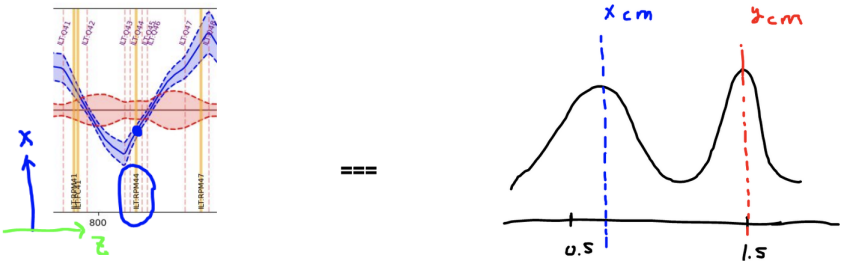


Figure 38: RPM profiles with negative polarity.

The polarity is automatically determined by the code. To determine polarity is determined by the profile variables shown in Figure 39. If neg_x/y $<$ pos_x/y the polarity is positive, else it is negative.

```
<element id="ILT:RPM44" type="rpm" s="591.321*mm" l="0.00*mm">
  <layout x="-2339.8805*mm" y="70606.27*mm" z="7872.4318*mm"/>
  <profile neg_x="0.98" pos_x="0.02" pos_y="1.02" neg_y="1.98" scaleToCm="2.54"/>
  <epics>
    <setpoint pv="ILT:RPM44:SUBRPM" type="scan"/>
    <setpoint pv="ILT:RPM44:GAIN" type="gainset"/>
    <readback pv="ILT:RPM44:CURARR" type="profile"/>
    <readback pv="ILT:RPM44:POSARR" type="position"/>
    <readback pv="ILT:RPM44:RDPRK" type="status"/>
  </epics>
</element>
```

(a) RPM44: Polarity $= -1$

```
<element id="ILT:RPM33" type="rpm" s="167.8952*mm" l="0.00*mm">
  <layout x="320.19523*mm" y="70314.819*mm" z="7872.4327*mm"/>
  <profile neg_x="0.02" pos_x="0.98" pos_y="1.02" neg_y="1.98" scaleToCm="2.54"/>
  <epics>
    <setpoint pv="ILT:RPM33:SUBRPM" type="scan"/>
    <setpoint pv="ILT:RPM33:GAIN" type="gainset"/>
    <readback pv="ILT:RPM33:CURARR" type="profile"/>
    <readback pv="ILT:RPM33:POSARR" type="position"/>
    <readback pv="ILT:RPM33:ENB" type="status"/>
  </epics>
</element>
```

(b) RPM33: Polarity $= 1$

Figure 39: RPM implementation in acc

6.5.2 LPM & PM

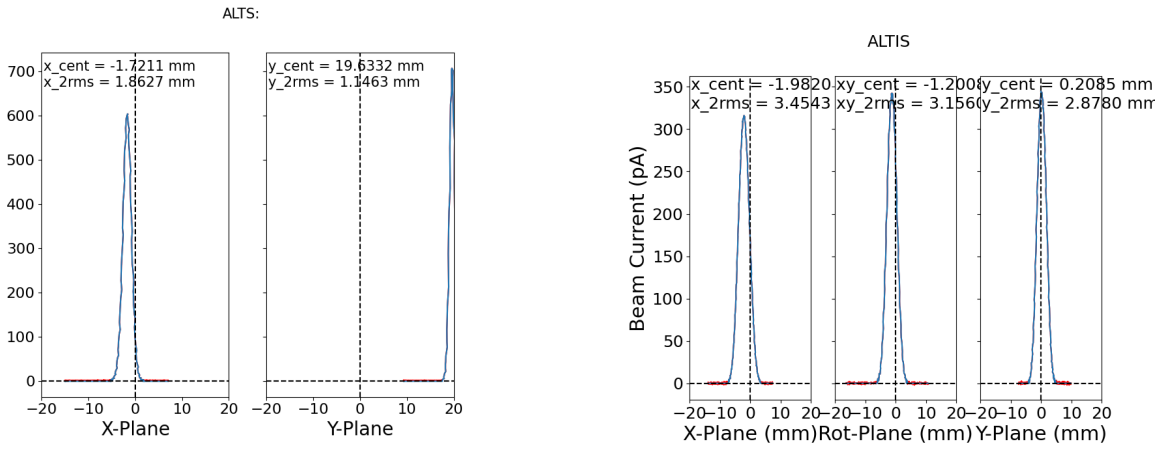
To monitor the position of the beam in ARIEL, PMs and LPMs are installed in the ring. The software to process the profiles measured by both types of position measurement devices is maintained by Suresh Saminathan. The software is saved in interface/pm_lpm_scripts/. This direction contains a configuration file, which is read in by the software to analyse the profiles. The equation to convert the position axis (which is actually a potentiometer voltage reading) to a distance in mm is :

$$D = A \cdot ((B - C)/10) + C \tag{20}$$

where d is the travel distance in mm of the monitor, A is the potentiometer reading in volts, B and C are constants in mm from the config file. They come from measurements on a test bench prior to installation. This distance D is then the distance along the direction of motion which is at a 45 degree angle to the horizontal plane. So to really convert the voltages into a distance in X or Y plane you need to tack on an additional $1/\sqrt{2}$:

$$X = \left[1/\sqrt{2}\right] \cdot [A \cdot ((B - C)/10) + C] \tag{21}$$

The beamline center in both X and Y (as a pot reading in volts) is given in the same config file. The information in which direction is +/-x and y is saved under quadrant. An example of the processed profiles is shown in figure 40.



(a) LPM Profile

(b) PM Profile

Figure 40: Position measurement profiles on the ARIEL beamline.

7 Validation of Trained Agents

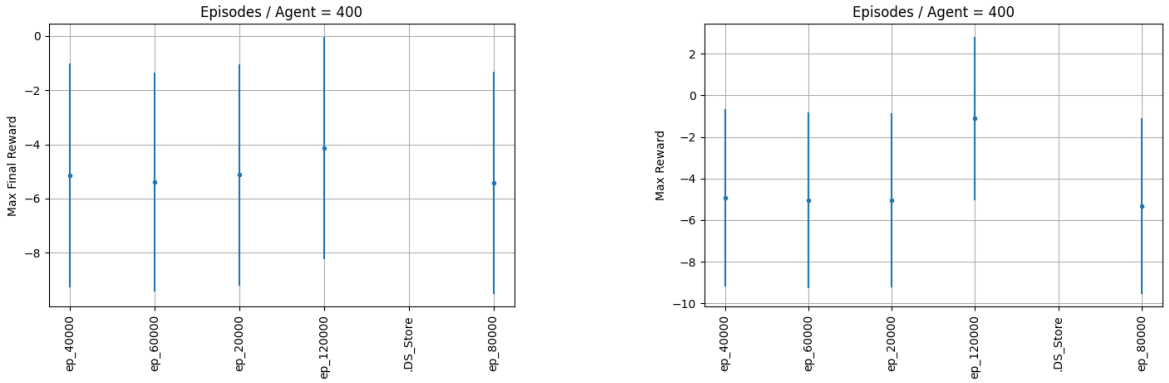
To get an inside, what an agent is actually doing and to compare with other agents, analyze_agent.py is developed. The *root* directory is the path, where your agents you want to analyze and compare are stored. *runs* contains the names of agents you want to analyze and compare. This program is useful after hyperparameter tuning. You can load several agents and the program tells you which agent performed best. Having many agents trained, mixing settings in configuration files happens very fast! To be sure you are comparing and validating the correct agents with the correct settings, the configuration files of all agents are loaded. The program will give you warnings, if settings are different.
At the beginning of the program you have some more options to validate the agents performance

- **show_measurement** If set to True, you can compare the predicted angles from a measurement with the predicted angles from the simulation
- **date** and **time** is the directory where the measurement is saved for the agent
- **load_data** Validating takes some time! The raw data is saved into the agent directory. If you want to rerun the code but not generate the raw data again you can simply load it, as the raw data will be the same if validated on the same seed. If you want to load the data, set this flag to True.
- **episodes** Number of episodes you want to validate the agent
- **analyze_all_models** When training an agent, the agent parameters are saved for several milestones (each 20000 episodes, best score,..). If set to *True*, all agents in the models directory of an agent are validated. This takes a lot of time, but it is also interesting so see how different agents behave.
- **custom_models** If you don't want to validate all models, you can specify model names in the list and only the specified models are validated. *analyze_all_models* needs to be set to False.

Furthermore you have the option to overwrite the configurations file within the code. This needs to be done, to evaluate the agents on the same seed for example.
IMPORTANT: Before I left the project I did a lot of changes to the configuration files, e.g merged the configuration files for interface and training. While training, the configuration file is automatically copied into the runs/ directory. From this directory, the validation program loads the configuration file. With all changes done lately, loading in old configuration files will lead to problems. However, debugging is straight forward. Change and add the variables, the program tells you to change. This is not a problem for new trained agents, as the correct configuration files is saved.

7.1 Output of the Validation

All results of the validation of an agent are saved into the agent directory under demo/. The folder contains a subfolder for each model analyzed as well as a comparison of the models as shown in Figure 41. For each validation episode, the maximum reward as well as the final reward and saved. At the end of the validation, the average value as well as the standard deviation is computed and plotted. Looking into the different subfolders, all data of a few episodes is



(a) Maximum final reward

(b) Maximum reward in an episode

Figure 41: agent performances for different agents.

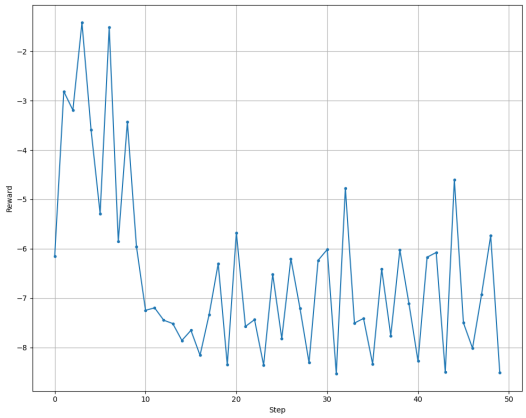
saved, e.g all actions taken in that episode, the reward as a function of steps (Figure **??** (a), a gif showing the beam envelopes during the training, the position measurements at the position measurement devices and the fractional current measured at the RPMs.

7.1.1 Analysis of rewards during a validation episode

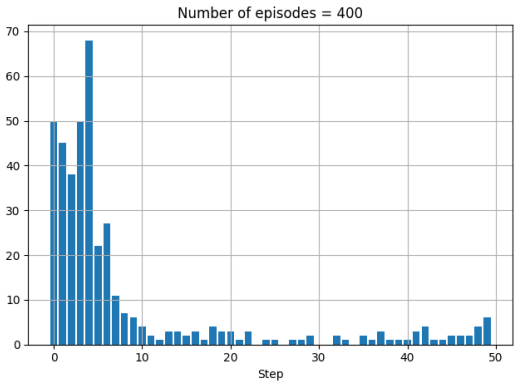
Doing this validation of the agents, we noticed that the reward is behaving not as we expected. Figure 42(a) shows the reward as a function of steps during a validation episode. The reward at the beginning is much higher (and so is transmission), than towards the end of an episode which is not intuitive, as you would expect the agent to either improve the transmission or a saturation of transmission. The reason, why the agent chooses actions towards the end of an episode, which decrease the performance, is unknown. Figure 42(b) shows the distribution of steps in which the maximum reward is achieved among the 400 episodes used for validation.It can be clearly seen, that only in a very few episodes the maximum reward is achieved in the last episode. Most episode behave like shown in Figure 42(a).

7.1.2 Analysis of steerer actions during a validation episode

In this section, the predicted angles from agents is discussed. For each step in an episode, the predicted angle for a steerer is saved. When the validation is done, the steerer angles are histogrammed as shown in Figure 43(a). We were surprised about the results, as the histograms have a Gaussian shape. The mean and standard deviation values of the steerer distributions are plotted as a function of steerer in Figure 43(b). The predicted values are in general always centered around zero with a small standard deviation. This behaviour has been seen in all trained agents. This is a problem, as the agent is not exploring and using the entire action space, but limited to a narrow range of possible actions. The overall performance of these agents is bad when training on large misalignments, which is a problem as the misalignments on the OLIS beamline are fairly large and unknown. The problem becomes even clearer, when comparing the predicted values on the simulation with the predicted steering angles on the real beamline as shown in figure 44. Figure 44(a) shows the predicted values on the simulation (blue) and the predicted values on the real beamline (red, scaled for better visibility). The predicted values for both scenarios cover



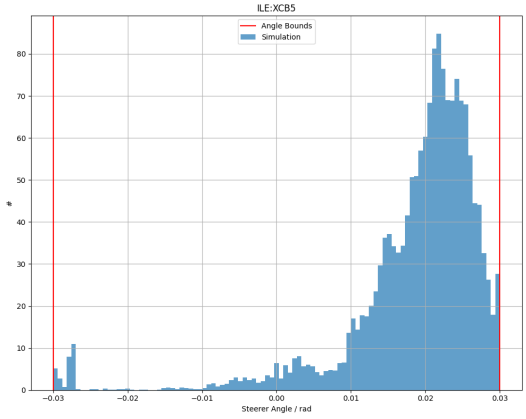
(a) Maximum final reward



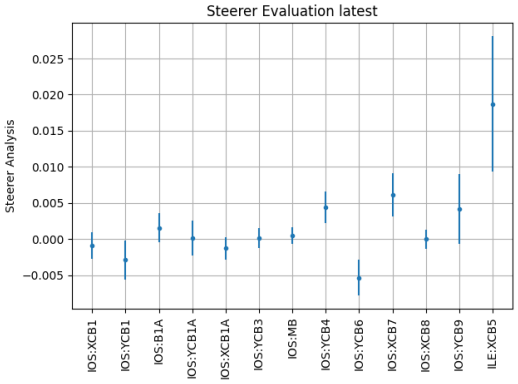
(b) Maximum reward in an episode

Figure 42: Reward during an episode.

the same range. In other words, the agent seems to have "favorite" values, when predicting new values, independent from the agent inputs. The limited action space range becomes problematic, because the OLIS beamline needs a lot of steering as shown in Figure 44(b). The blue data points show the settings for good beam transmission through OLIS. In the simulation, the agent has never even touched the region of large steering angles, e.g the agent will never predict such a large value, the width of the predicted steering angles distribution is too small. Figure 45 shows the evolution of the distribution of predicted steering angles. During the training process, the exploration is increasing but not enough to explore the full action space. This effect is still under investigation. Recently a bug was found in the exploration noise. We started monitoring the actions predicted by the agent (greedy actions) and the noisy actions which are passed to the environment and as an input for the next step. Apparently, the exploration noise function was not acting as we expected, there is no real 'exploration' happening (46). Increasing the amount of randomization makes the agent use the entire action space 47.



(a) Maximum final reward

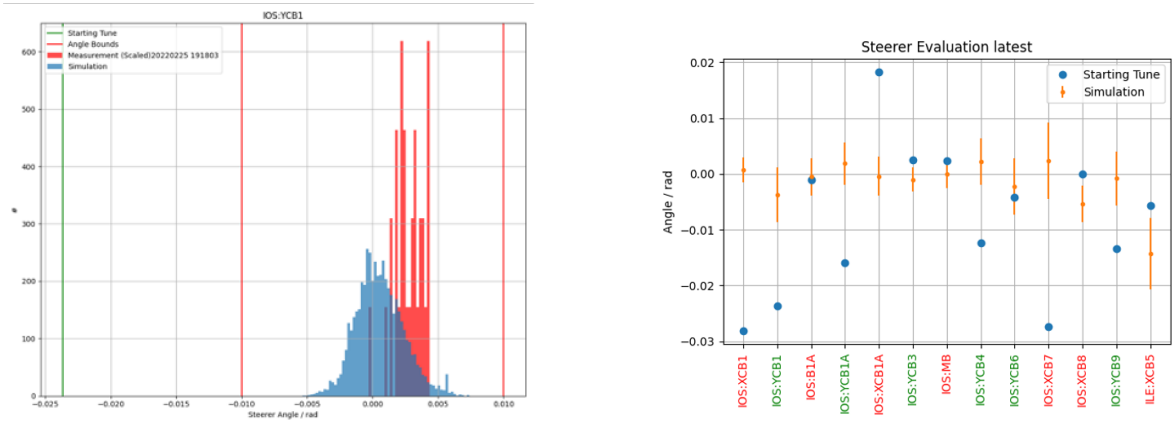


(b) Maximum reward in an episode

Figure 43: Predicted steering value distributions on the simulation.

7.2 Hyperparameter Tuning

Training an agent involves a lot of fine tuning of parameters. Especially RDPG agents are very sensitive when it comes to the tuning of their hyperparameters. Hyperparameter tuning should be done on Compute Canada, as it involves a lot of computing resources and takes a long time!



(a) Maximum final reward

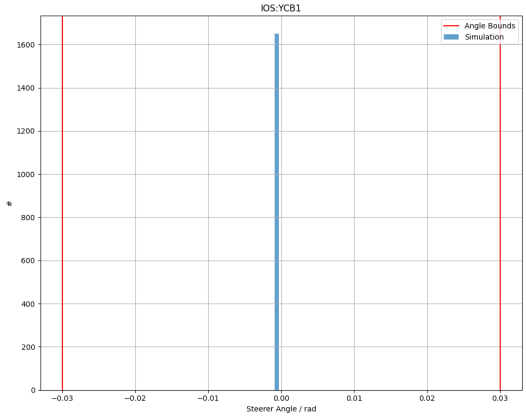
(b) Maximum reward in an episode

Figure 44: Predicted steering angles comparison (Simulation and real beamline)

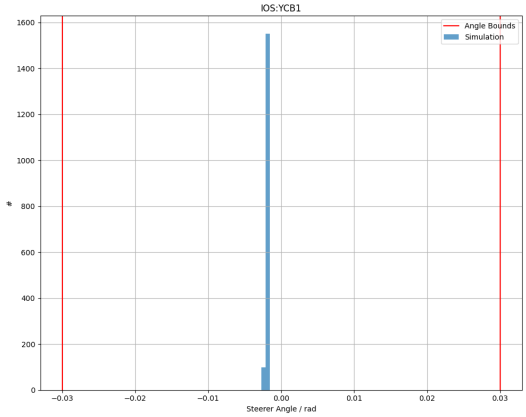
In Figure 48 a few agents with different hyperparameters are shown. In general, correlations between hyperparameters cannot be excluded. One of your tasks is to bring the hyperparameter tuning to the next level and explore a big range of numbers! It is also very important to train the same settings on different seeds to make sure you are actually seeing an effect. Random_seed can be set in config/config.yaml.

8 What to do next?

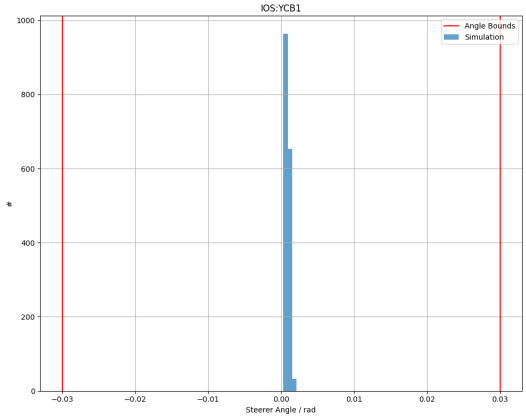
- Run OLIS without slits
- Once the motors on the slits on OLIS are fixed and accessible to EPICS, make the slit position a tuneable parameter of the agent.
- Don't add noise all the time
- Apertures of magnets
- rerun IOSPrageMWS and OLISMWS data dat files
- Think of a way to include absolute values of currents at FCs
- Make things work on ARIEL
- Current measurement at the edge of the beamline
- Studies of the misalignment of the OLIS source
- fc weighting in reward function
- x deflector XCB1A in OLIS is basically a steerer, it might be a better idea to treat it like this in the predictor interface
- the algorithm for common - plates



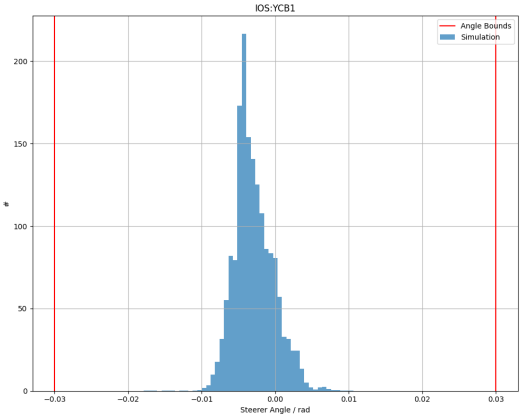
(a) Episode 20000



(b) Episode 40000



(c) Episode 60000



(d) Episode 120000

Figure 45: Predicted steering angles for models saved at 20000, 40000, 60000 and 120000 episodes. The width of the distribution slowly increases, which means that the agent is increasing its exploration of the beamline.

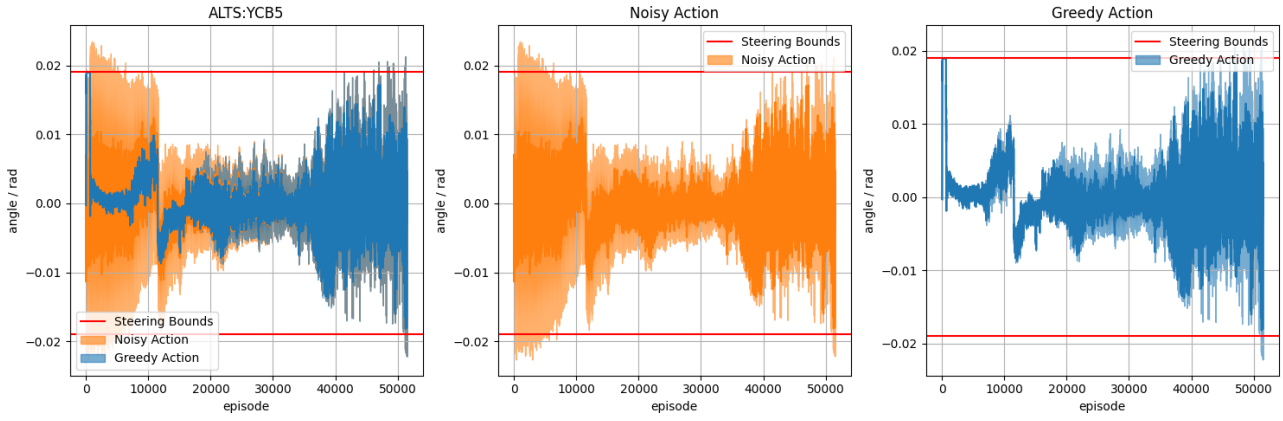


Figure 46: Noisy Actions and Greedy actions before finding the bug.

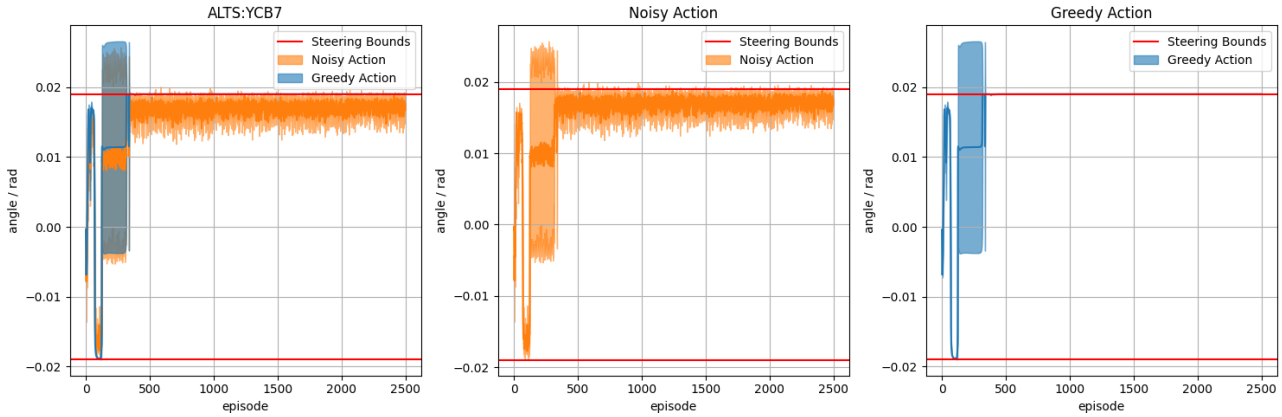
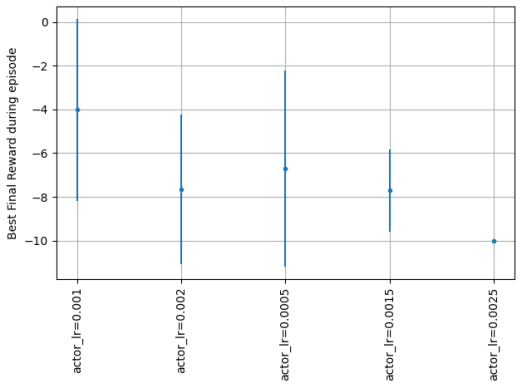
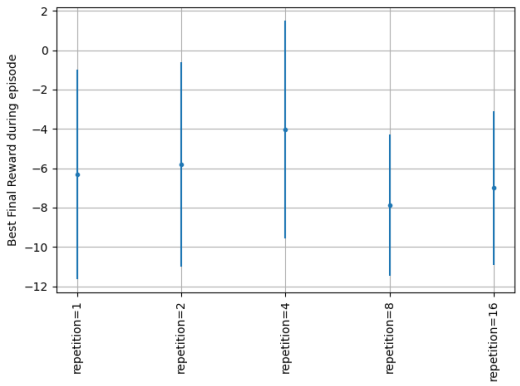


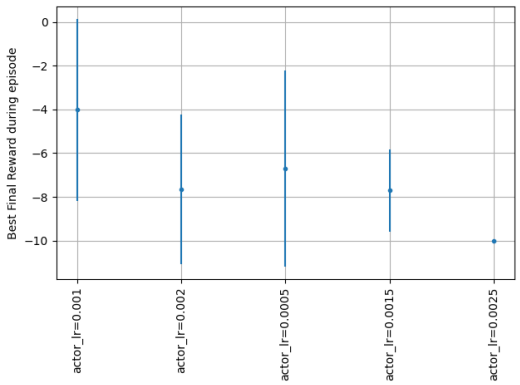
Figure 47: Noisy Actions and Greedy actions after finding the bug.



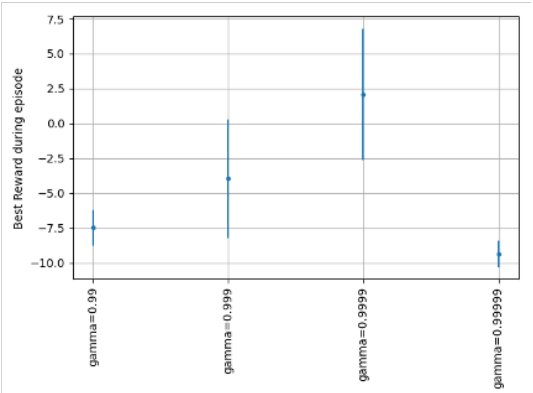
(a) Maximum reward in an episode



(b) Maximum reward in an episode



(c) Maximum reward in an episode



(d) Maximum reward in an episode

Figure 48: Exploration Noise Strategies for two different trained agents.