LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note	LIGO-T2400258–	2024/09/27

Final Report: Replicating Finesse simulation using neural networks to train RL agent for lock-acquisition

Anubhav Prakash

California Institute of Technology
LIGO Project, MS 18-34
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project, Room NW22-295
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
Route 10, Mile Marker 2
Richland, WA 99352
Phone (509) 372-8106
Fax (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (225) 686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

http://www.ligo.caltech.edu/

Contents

1 Overview

This report presents a summary of my SURF project at 40m laboratory, Caltech conducted during summer of 2024. The goal of the project was to assist in decreasing the training time of an RL agent by providing a faster way of generating 40m IFO using neural networks instead of Finesse which is ≈ 30 times slower that the neural networks with first order laser mode.

My very initial steps involved basic simulations of a Fabry-Perot cavity, followed by transitioning from kat script-based programming to using modern model constructor functions. This phase laid the groundwork for understanding the simulation environment and the necessary coding practices.

Subsequent tasks included performing mode scans, analyzing transmitted power variations, and simulating a high-finesse Fabry-Perot cavity with RF modulation, to accommodate myself with finesse and with the physics of optical cavities. I also explored using astigmatism to identify the peaks and understand the degeneracies when multiple modes resonate at the same cavity length. This led to the identification of peaks corresponding to specific mode orders and their sidebands.

I further found thermally robust and isolated carrier peaks by simulating thermal effects, basically by changing radii of curvature of both mirrors, and identifying cavity lengths that maintain isolated peaks across the range of radii of curvature. A significant portion of the time was spend doing supervised learning aimed at training a neural network that copies the output of finesse and generates 31 signals corresponding to total power and demodulated powers at 3 ports of the IFO. A lot of time was spent on further improving the training as well the the result's accuracy and providing a scientific way to quantify the accuracy while ascertaining whether the trained model will be succesful to help the RL agent properly lock the PRMI. I used Neptune AI and Optuna for hyperparameter optimization in neural network training. The integration facilitated efficient hyperparameter tuning and experiment tracking, streamlining the model development process. The later part of the report explains the idea behind the approach taken to quantify the accuracy of the neural network compared to finesse and I have also shown some plots for real life data to show how far we are from the realistic scenario. Future work includes extending the supervised learning approach to replicate Finesse simulation outputs, particularly for higher-order modes.

2 Learning Finesse and Higher-Order Mode Sweep Analysis

2.1 Finesse: A Python Package for Optical Interferometer Simulation

Finesse is a Python package used for simulating optical interferometers and systems with optical components. According to the Finesse Documentation, "It employs frequency-domain optical modeling to create accurate quasi-static simulations of arbitrary interferometer configurations." It is based on an object-oriented structure and provides a wide range of utility functions to simplify complex simulation tasks, such as simulating a Fabry-Perot cavity

with a Gaussian laser beam containing multiple Higher-Order Hermite-Gauss modes up to order 12. The learning process involved reviewing the API documentation: https://finesse.ifosim.org/docs/latest/api/index.html and studying the physics behind Finesse: https://finesse.ifosim.org/docs/latest/physics/index.html

2.2 Short summary of my initial work for learning finesse and simulating various optical cavities with specific conditions

I started by exploring Finesse with a basic Fabry-Perot cavity simulation, focusing on transitioning from the traditional kat script to modern model constructor functions. I reviewed the API documentation to identify functions for adding optical components, linking them, and defining parameters.

Next, I conducted a mode sweep simulation up to maxtem 12, where different Hermite-Gauss modes resonate at distinct cavity lengths. This produced peaks in transmitted power as the cavity length was varied.

I then simulated a high-finesse Fabry-Perot cavity with RF modulation, plotting the transmission power against the frequency offset. The plot revealed multiple peaks within the free spectral range (FSR), corresponding to various maxtem values, each having a sideband due to RF modulation. I further analyzed astigmatism to understand degeneracies, finding that certain mode combinations (e.g., (0,7), (1,6)) resonate at the same cavity length.

Subsequently, I aimed to calculate robust L_{cav} values where carrier peaks would be isolated from other peaks. This involved calculating peak positions and checking for isolated peaks in a specific frequency range, considering thermal effects on the mirror's curvature. The final results showed a safe range of L_{cav} values (1.32–1.37 m) that remained robust against curvature variations.

Finally, I plotted these values as an animation.
Link of Animation 1
Link for animation 2

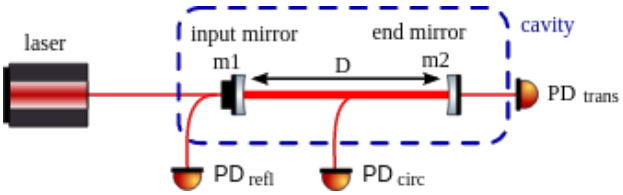

Figure 1: Image Source: Finesse documentation

2.3 Interesting Thing to Notice

In the second animation is that the peak value of the peaks oscillates. The output power expression for the CR_0 peak is given as (for critical coupling):

$$P_{\text{trans}} = \frac{T}{1 + R^2 - 2R\cos(2kL)} \tag{1}$$

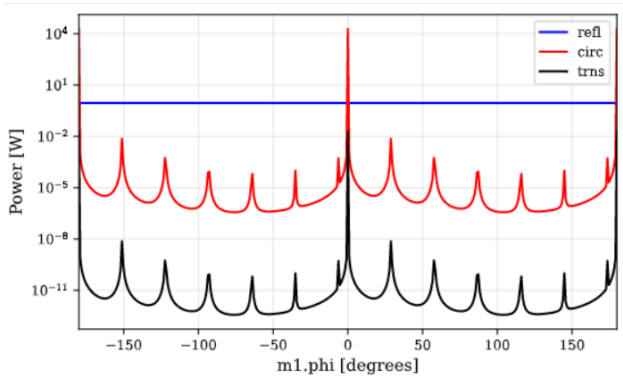

Figure 2: Mode sweep Scan

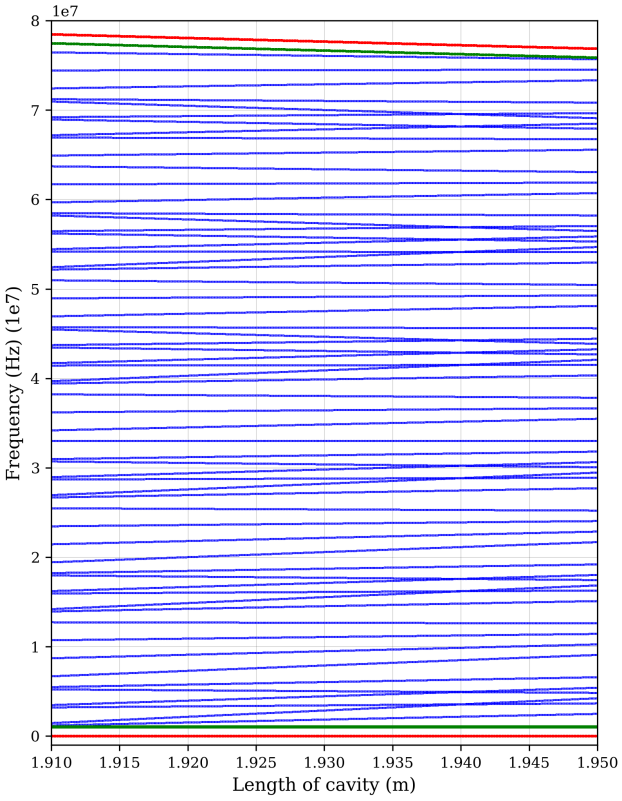

Figure 3: The space between the green and red line (both top and bottom) is where no blue line should reach

Similar expressions give the power transmitted when higher modes are resonant. The length at which the modes oscillate is changing with a change in RoC, and the change rate is faster if the mode order is higher. Thus, for a faster change in L, the power expression will oscillate faster due to the presence of $\cos(2kL)$ in the denominator, where L corresponds to the resonating cavity length for the mode.

3 Supervised Learning for copying finesse 40m simulation output

We use the FINESSE 40m package to model the interferometer, which is configured with parameters specific to the 40-meter interferometer. This package provides the interferometer's response to changes in its degrees of freedom and serves as the foundation of our environment. However, since FINESSE is inherently static, it does not simulate how these degrees of freedom evolve over time. Therefore, we must develop additional code to account for these changes.

A key challenge is the large number of samples needed for the RL agent's training, which is limited by FINESSE's extensive call times. To address this, we created a neural network model that mimics FINESSE's output, drastically reducing computational overhead. For higher-order mode simulations (HG eigenmodes), we expect up to a three-order reduction in runtime, and for lower-order simulations, the neural network approach is already about 30 times faster than FINESSE.

NOTE: Before jumping into training the neural network for the 40m finesse model, I started with a supervised learning exercise on MNIST data set to gain some idea of the training process while incorporating neptune and optuna integration. The code can be found in the github code base.

3.1 Neptune-Optuna Integration

Optuna is an open-source hyperparameter optimization framework that scans over all the hyperparameter values available while training a neural network model and helps us automatically find the best combination of hyperparameters for the model.

On the other hand, Neptune AI is a cloud-based platform that provides tools and infrastructure to build, deploy, and manage machine learning models and experiments.

Integrating Optuna with Neptune AI allows us to leverage Optuna's efficient hyperparameter optimization capabilities directly from within the Neptune AI platform. This integration streamlines the model development process by enabling users to configure and launch Optuna optimization studies seamlessly within Neptune AI's collaborative workspace. The integration also ensures that all hyperparameter values, metrics, and model artifacts generated during the optimization process are automatically tracked and logged in Neptune AI, facilitating analysis, reproducibility, and collaboration among team members.

3.2 What did I do?

3.2.1 Signals generation for 40m PRMI

I have generated 31 signals which are as follows:

- POP power (reflected from Power recycling mirror)
- REFL power (Power from Anti-reflective surface for the beam splitter)
- AS power (Power passing through Signal recycling mirror)
- POP demodulated at f1 frequency (Both I and Q phase signal)
- POP demodulated at 2f1 frequency (Both I and Q phase signal)
- POP demodulated at f2 frequency (Both I and Q phase signal)
- POP demodulated at 2f2 frequency (Both I and Q phase signal)
- AS demodulated at f1 frequency (Both I and Q phase signal)
- AS demodulated at 2f1 frequency (Both I and Q phase signal)
- AS demodulated at f2 frequency (Both I and Q phase signal)
- AS demodulated at 2f2 frequency (Both I and Q phase signal)
- REFL demodulated at f1 frequency (Both I and Q phase signal)
- REFL demodulated at 2f1 frequency (Both I and Q phase signal)
- REFL demodulated at f2 frequency (Both I and Q phase signal)
- REFL demodulated at 2f2 frequency (Both I and Q phase signal)
- REFL demodulated at 3f1 frequency (Both I and Q phase signal)
- REFL demodulated at 3f2 frequency (Both I and Q phase signal)

The code used for data generation is in this ipynb file

In the provided code, I have generated the required signals using finesse_40m library.

- I import all the necessary libraries and then initialise the neptune run at the beginning to start logging my constants values.
- I modify the "Forty_Meter_Factory" function imported from finesse_40m package to turn off the default DOFs and turn off all the default detectors to that the finesse model does not do unnecessary computations to generate signals which I won't be using.

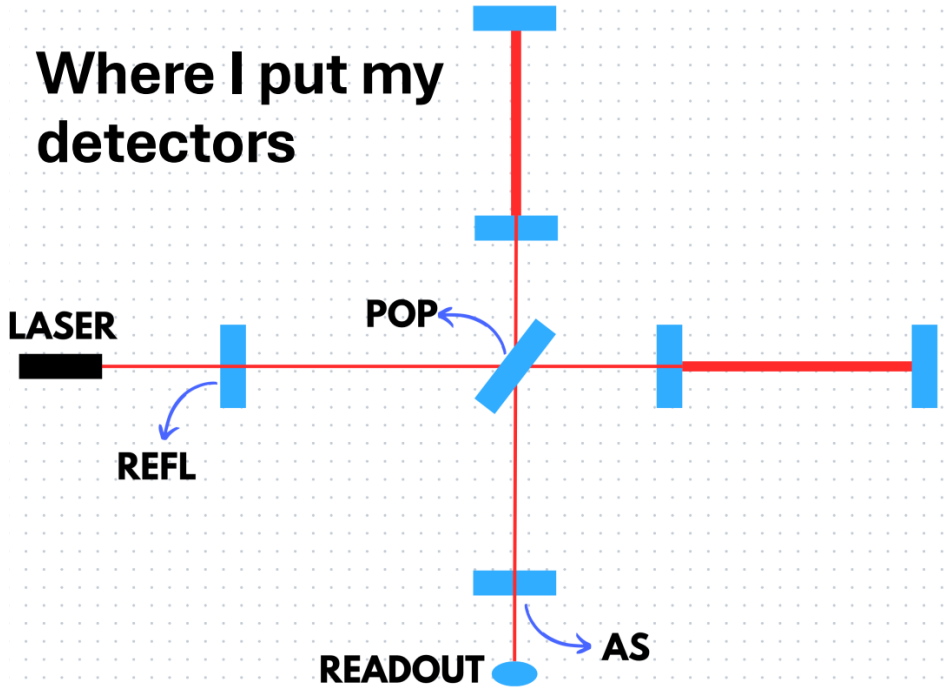

Figure 4: Image giving a rough idea of where the signals are picked from

- I start with adding my required DOFs for the PRMI which is the MICH length and the PRCL length.
- I then add all my 17 detectors as follows:

```
model.add([
    PowerDetector('P0', model.BS.p3.o),
    PowerDetector('REFL', model.PRM.p1.o),
    PowerDetector('AS', model.SRM.p1.o),
    PowerDetectorDemod1('P0demodf1', model.BS.p3.o, freq1),
    PowerDetectorDemod1('P0demod2f1', model.BS.p3.o, 2 * freq1),   # Demodulating it parallel, but note that the demodulation is done in series
    PowerDetectorDemod1('P0demodf2', model.BS.p3.o, freq2),
    PowerDetectorDemod1('P0demod2f2', model.BS.p3.o, 2 * freq2),
    PowerDetectorDemod1('REFLdemodf1', model.PRM.p1.o, freq1),
    PowerDetectorDemod1('REFLdemod2f1', model.PRM.p1.o, 2 * freq1),
    PowerDetectorDemod1('REFLdemodf2', model.PRM.p1.o, freq2),
    PowerDetectorDemod1('REFLdemod2f2', model.PRM.p1.o, 2 * freq2),
    PowerDetectorDemod1('REFLdemod3f1', model.PRM.p1.o, 3 * freq1),
    PowerDetectorDemod1('REFLdemod3f2', model.PRM.p1.o, 3 * freq2),
    PowerDetectorDemod1('ASdemodf1', model.SRM.p1.o, freq1),
    PowerDetectorDemod1('ASdemod2f1', model.SRM.p1.o, 2 * freq1),
    PowerDetectorDemod1('ASdemodf2', model.SRM.p1.o, freq2),
    PowerDetectorDemod1('ASdemod2f2', model.SRM.p1.o, 2 * freq2),
])
```

The last 14 are the demodulated signals which are then split into its real and imaginary part to give I and Q phase signals and thus giving my 31 signals.

- I generate 1000x1000 grid in the MICH-PRCL phase plane and then do a 2D scan to generate data for 10^6 different pair of values for MICH and PRCL.
- I then finally split the demodulated signals into their real and imaginary parts and then save the whole data for preservation in case of crashes like "OUT_OF_MEMORY ERROR".
- I can either move to another code file for the sake of training my NN or I can do it in the same file, but it is much better to do it in another code file otherwise the chances of crashes are very high and then you will need to restart the training a lot many time before it runs successfully.

Now my data is generated and I will use it to train my neural network. The code is given here: GitHub Repository.

3.2.2 Normalising and denormalising the data

Note that for efficient training I cannot just input the data straight away. I need to prepare the data, normalise it and then define functions to denormalise it back to its usual scale on both input and output axis.

I do not use the scikit scalers to do the job because my signals have sharp peaks which might not be properly resolved due to not so high input resolution(which I chose to limit the runtime to normal timescales) hence I first shift the data between 0 to 1 range and then take a log to make sure that the small peaks are also enhanced and the width of the peaks are also broadened. The max and min value of the array and the scale factor is all saved to ensure that the output of the neural network can be properly denormalised to bring the data back to its usual scale.

Hence I define such normalising and denormalising functions which also returns the min and max values and the scale factors which wil lbe used as input in denormalising function.

3.2.3 Training the neural network

This provides an overview of the Python script I used that utilizes Optuna for hyperparameter optimization and Neptune for experiment tracking in training a neural network.

- The script begins by importing necessary libraries, including TensorFlow, NumPy, and the Optuna and Neptune integrations. The API token for Neptune is read from a file for secure access, and a new run is initiated to track parameters and metrics.
- Data is loaded and normalized using my custon normalisation functions. This preprocessing ensures the training data is in an appropriate range, enhancing model performance.
- Optuna is employed to optimize the neural network architecture through the `create_model(trial)` function, where hyperparameters such as the number of units in each layer and the learning rate are suggested. An objective function is defined to train the model and return the loss, allowing Optuna to evaluate various configurations.

- Additionally, a custom callback, `LogMetricsCallback`, is implemented to log training metrics (loss and accuracy) to Neptune at the end of each epoch. This integration provides real-time monitoring and analysis of the model's performance.
- After finding the optimal hyperparameters, the model is trained, and the training history is logged into Neptune. Once training is complete, the model is saved, and predictions are made, with results stored in a file.

Throughout the process, Neptune AI tracks and logs all the hyperparameters, trial results, training metrics, and final model performance, allowing us to visualize and analyze the optimization process and model performance within the Neptune AI platform.

Plotting Data Using Matplotlib

The script generates visualizations of simulated and predicted data using Matplotlib, focusing on several signals derived from the loaded datasets. The plotting section is organized into two main loops, one for the first three signals and another for the remaining signals.

1. Setup for Plotting

A figure is created with four vertical subplots using `plt.subplots(4, 1, figsize=(10, 30))`, ensuring that each subplot can display different visual aspects of the data.

2. Data Preparation

For each signal, data is reshaped into two dimensions (N by M) to facilitate plotting.

- **Error Calculations:**
  - The relative error is computed using the logarithm of the absolute difference between the actual (z_{fin}) and predicted (z_{pred}) values divided by the actual values.
  - The absolute error is similarly calculated.

3. Creating Plots

- **Subplot 1:** Displays the actual simulated data (z_{fin}) using a grayscale colormap. A colorbar is added to indicate the values of the finesse simulated.
- **Subplot 2:** Shows the neural network predictions (z_{pred}) in the same format as the actual data, allowing for visual comparison.
- **Subplot 3:** Visualizes the relative error as a percentage using a custom colormap (`cmap_residual`) that highlights errors, with a colorbar indicating the error ranges.
- **Subplot 4:** Presents the absolute error using a *coolwarm* colormap, facilitating identification of areas where the model's predictions significantly deviate from actual values.

4. Colorbars

Colorbars for each subplot help interpret the values. The colorbar ticks are customized to display values in scientific notation where necessary.

5. Uploading to Neptune

Each generated figure is uploaded to Neptune for tracking and visualization of results over different runs.

6. Saving Figures

Plots are saved as JPEG files in a specified directory, facilitating later analysis or presentation.

7. Finalizing the Run

After all plots are generated, the Neptune run is stopped, finalizing the logging of the experiment.

3.3 Plots

The plots can be found on the neptune website here.
Figure 5 is an example plot to showcase here in the report.

4 Judging the accuracy of the prediction

We need to scientifically quantify if the prediction result is good enough.

To judge that the model can be successfully used to help the RL agent lock the PRMI, I plot the distance between corresponding pair values of PRCl and MICH.

This means, that first I take a pair of PRCL and MICH value, then the signal (say S, which is a 31 elements long array) which is generated by Finesse at that point and find out the corresponding pair of PRCl and MICh value for which the NN prediction is the closest to the signal S.

Then I find out the distance between the two pairs of MICH and PRCL. The plot generated is shown in figure 6.

5 How the real data looks like

It looks quite miserably far from what we have generated because we are currently working in a 0 noise, 1st order HG mode zone which is highly idealized and our work needs to be made more realistic by addition of more and more complexity.

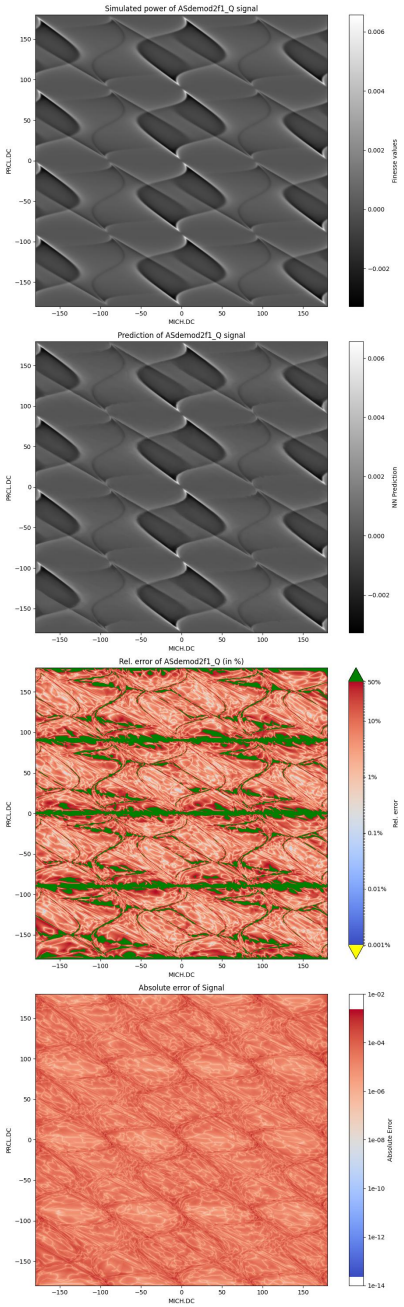

Figure 5: Plot for AS signal demodulated at 2f1 frequency in Q phase

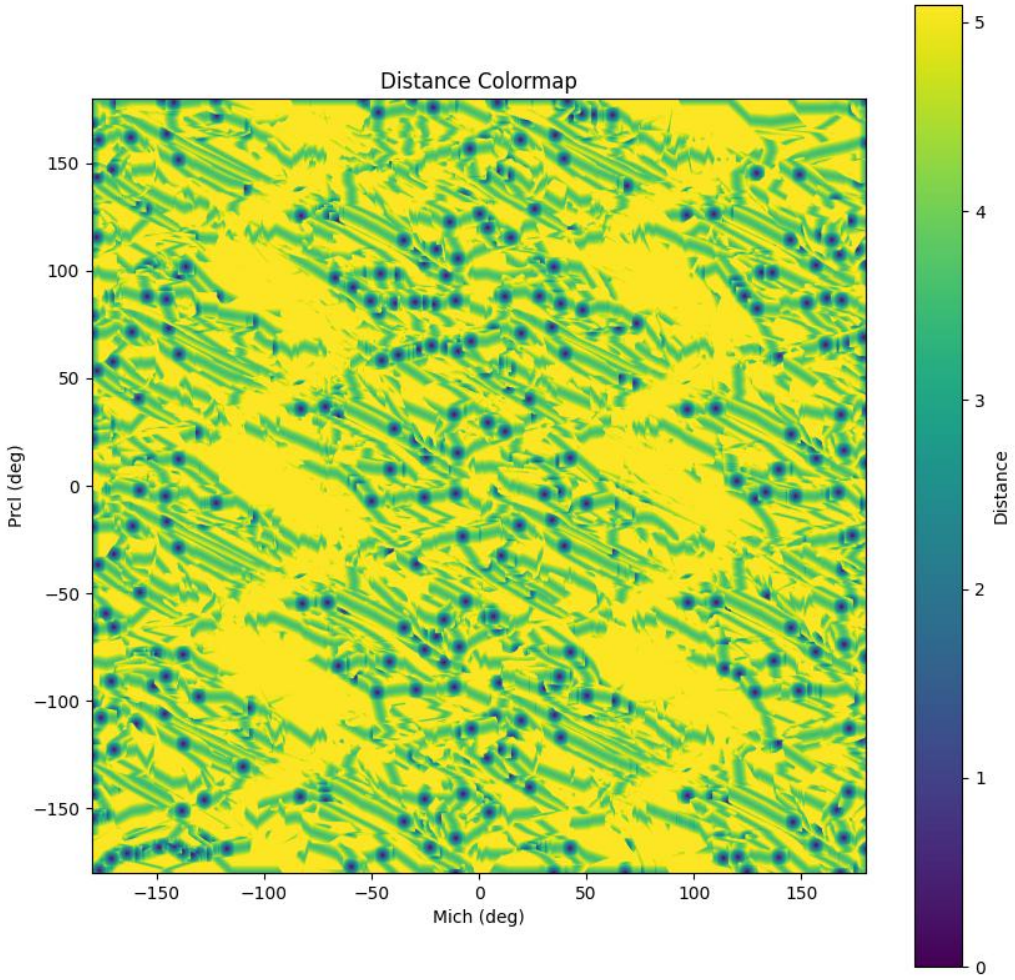

Figure 6: Accuracy Plot

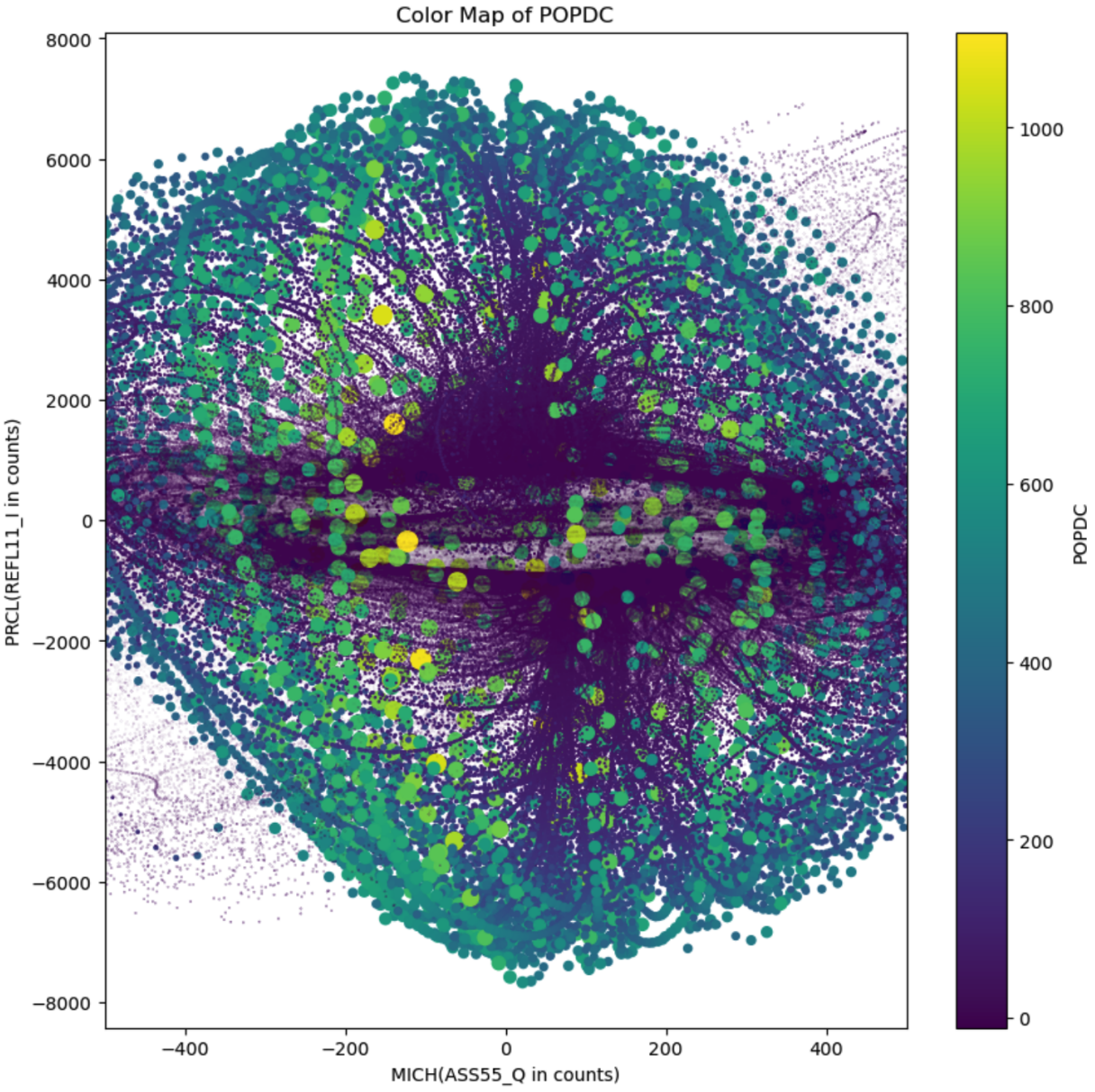

Figure 7: Plot of POP DC signal

6 Results

1. We achieved the duplication of finesse data to a decent level in first order laser mode, i.e., with maxtem 0.
2. The RL agent training process was seen to increase by 30 times when the model was used to train.

7 Future goals

1. We need to increase the maxtem and make the data more realistic
2. We need to incorporate the issues which the real-life noise may bring in. This is being done parallely to make the RL agent robust.
3. We might need to add better resolution around the region where we intend to lock the PRMI so that once locked the lock is sustained by active control using the RL agent
4. Finally the goal is to shift to DRMI to fully control the 40m IFO, except the arms where ALS works very nicely.

8 References

1. Finesse Python API Documentation
2. Finesse Documentation
3. Bond, C., Brown, D., Freise, A. et al. Interferometer techniques for gravitational-wave detection. Living Rev Relativ 19, 3 (2016).
4. CaltechExperimentalGravity/RL-Cavity-LockAcquisition
5. A. Effler et. al. 2014. Performance Characterization of the Dual-Recycled Michelson Subsystem in Ad- vanced LIGO
6. A. E. Siegman (1986) LASERS