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

Technical Note	LIGO-T2200196-v1	2022/07/28

Developing Deep Learning Solutions for Lock Acquisition Report 2

Peter Ma — Mentor: Gabriele Vajente

Distribution of this document:

LIGO Scientific Collaboration

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 NW17-161
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/

1 Introduction

This project's primary goal is to investigate and develop deep learning techniques to approach the problem of LIGO's lock acquisition. Specifically, we will look into leveraging modern techniques in attention-based learning to help estimate the state of the mirrors given optical signals from the Power Recycled Michelson configuration (PRMI). We will also look into the usage of deep reinforcement techniques in controlling the mirrors directly.

In this interim paper, we report the first $\frac{1}{3}$ of the project in progress. During the first 3 weeks of active research, we first replicated previous deep learning approaches using gated recurrent units (GRU). Secondly, we developed an attention-based state estimator using a transformer architecture. Thirdly, we conceptualized a velocity-position uncertainty estimator in an attempt to use sensor fusion techniques to reconstruct the state of the mirrors. Furthermore, we ruled that the Encoder-Decoder model should be explored only when a working model has been established. Lastly, we implemented a Deep Deterministic Gradient Policy (DDPG) model for a reinforcement learning approach. We report that all three approaches have yet to be thoroughly tested but have now been built. We remain inconclusive but hopeful as to the success of the three techniques.

2 Problem Statement

There are two approaches to acquiring the lock for a given set of mirrors at LIGO. The safe approach is to learn an estimate of the state of the mirrors and the ambitious approach is to learn the controls directly to drive the motions close to 0. The first approach is considered sufficient since if we know the state of the mirrors, it is relatively easy to build these controllers. This is considered safer because the model only suggests additional information to the controllers, furthermore, it is more interpretable than the latter approach. The second approach removes any trace of interpretability and requires one to have faith in the model that it can do what it is set out to do and not break in production.

2.1 The State-Estimator Problem

We revisit the central issue in the state-estimator problem. What we want is, given a set of optical signals, we want to get out the relative positions of the mirrors. The problem is that there exists an infinite number of solutions where these positions satisfy the optical signals we inputted to the model. This is generally not a problem, any solution would work. The issue is, once a solution is chosen, we need to follow that solution over time to preserve the continuity of these trajectories. To simplify the nonuniqueness of the problem, we can "wrap" the data to force the labels to have unique solutions during training.

To perform this wrapping technique we first need to understand the system we're working with. Recall that the PRCL and MICH are *sets of mirrors* and together form a system highlighted in

$$
\begin{aligned}
\phi_{MICH} &= k\delta L_{MICH} \pm \frac{\Omega}{c} L_{MICH} \\
\phi_{PRC} &= k\delta L_{PRC} \pm \frac{\Omega}{c} L_{PRC}
\end{aligned}
\qquad
\begin{aligned}
r_{MICH} &= r_X t_{BS}^2 e^{i\phi_{MICH}} + r_Y r_{BS}^2 e^{-i\phi_{MICH}} \\
t_{MICH} &= t_{BS} r_{BS} \left(r_X e^{i\phi_{MICH}} - r_Y e^{-i\phi_{MICH}} \right) \\
\Psi_{PRC} &= \frac{t_{PR}}{1 - r_{PR} r_{MICH} e^{2i\phi_{PRC}}} \Psi_{IN} \\
\Psi_{REF} &= \frac{i r_{PR} - i t_{PR}^2 r_{MICH} e^{2i\phi_{PRC}}}{1 - r_{PR} r_{MICH} e^{2i\phi_{PRC}}} \Psi_{IN} \\
\Psi_{AP} &= \frac{i t_{MICH} t_{PR} e^{i\phi_{PRC}}}{1 - r_{PR} r_{MICH} e^{2i\phi_{PRC}}} \Psi_{IN}
\end{aligned}
$$

Figure 1: Ψ are the signals we get, however note that there are complex exponentials thus are periodic and is related to L_{MICH} and L_{PRCL}

figure 2. We can numerically recover the signals produced from the length L_{MICH} and L_{PRC} using a set of equations shown in 1. following equations, where k is the wave number and Ω is a constant related to the laser, c is the speed of light:

$$\phi_{MICH} = k\delta L_{MICH} \pm \Omega/c L_{MICH}$$
$$\phi_{PRC} = k\delta L_{PRC} \pm \Omega/c L_{PRC}$$

With enough elbow grease one can reduce the signals Ψ's such that they are related to L_{MICH} and L_{PRCL}. The issue is that leaving it in such a form it is difficult to retrieve the period of the signals analytically since there is a mix of terms with L_{PRC} and L_{MICH}. Thus to simplify this we can apply a linear transformation namely

$$Z_1 = 2(L_{PRCL} + \frac{L_{MICH}}{2})$$
$$Z_2 = 2(L_{PRCL} - \frac{L_{MICH}}{2})$$

If we plug this transformation into the equations we retrieve a period of $/2$. With this transformations we can now wrap the data.

Firstly we will take the PRCL and MICH positions and follow the transformations described for Z1 and Z2. After that is done, we iterate through the data and linearly shift up or shift down by integer multiples of the period such that all positions are restricted between $\lambda/4 \rightarrow -\lambda/4$. Then we transform back returning the PRCL and MICH positions successfully wrapped. This helps guarantee that for every signal the corresponding wrapped position is unique!

This technique is a problem in production. When computing these we have to undo this preprocessing step and unwrap the data in a manner that retains continuity such that the reconstructions

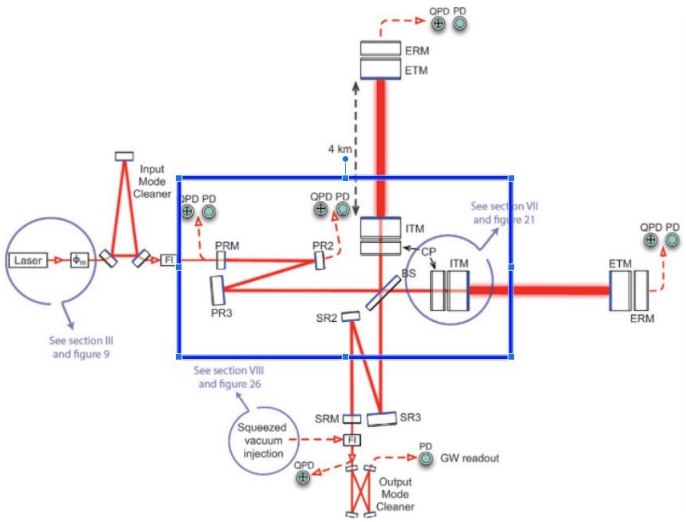

Figure 2: The rectangle boxed section is the PRCL MICH system that we're considering.

are accurate. We need to unwrap the data inorder to recover the true positions. This is what we call the wrapping problem.

3 Baseline Model

Previous attempts in state estimation used gated recurrent units called GRUs [1] [2] to predict the wrapped positions. GRUs are a class of recurrent neural networks. Recurrent neural networks (RNN) are simply neural networks that work with sequential data. They are often written with a recursive definition. A more concrete implementation requires one to pass information from previous time steps to the next. At the first time step, we input data into a classical neural network, then during the next time step, we input both the new data observed and the hidden state from the previous time step and continue until we reach the end of the sequence. Each set of feed forward networks for input data is called a cell or unit. This way we can pass information from previous observations to inform the model in the decision-making process. The classical RNN described suffers from a host of memory issues. With each passing of hidden states, data from the beginning may not be represented well further down the sequence, GRUs take the RNN idea further by developing specific gates within each cell that manipulate the hidden states that are passed down called the reset gate, and update gate[1]. The reset gate can selectively choose to nullify certain data that are passed from before, and the update gate decides what's important to pass on to the next state[2].

This technique was chosen because the data is time series and secondly the input data is relatively long on the order of 1000's input samples. This was the reasoning for this approach back in 2017. To that end, our first task was to replicate these results which we successfully did.

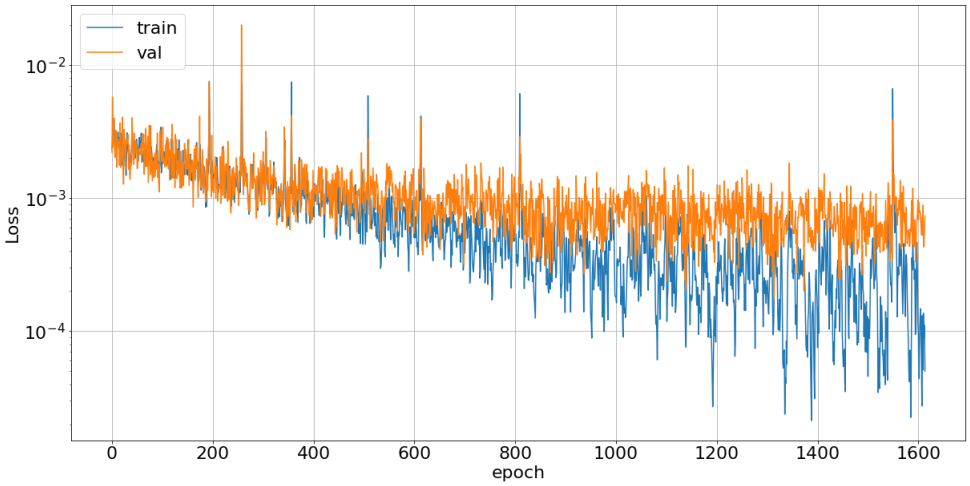

Figure 3: This is the loss curve for the model training.

3.1 Methods

We implemented a GRU in KERAS framework[3]. The data used for training was simulated PRCL and MICH with approximately 40,000 samples each 0.5 seconds long sampled at 2048Hz. Given these optical signals, we then try to recreate a single-time step of position data.

The model architecture had 2 GRU layers, with 128 units each, followed by 4 fully connected layers of 4092, 512, 32, and 2 units respectively. Each layer used a ReLU activation except for the final layer which is given a linear activation. We used the Adaptive moment (AdaM) optimizer with a learning rate of 1e-3. The reconstructions are as follows in the following diagrams [1].

This approach is insufficient because we cannot leave the data wrapped. This is because in reality the mirrors are free to move beyond the range which we restricted the solutions to. Thus to retrieve the true relative motions we must stitch back the reconstructions. Furthermore, these sharp positions are unphysical and is entirely the result of training the model on a training set where we linearly shifted the data down into the range of $\lambda/2 \rightarrow -\lambda/2$ where we know unique solutions exist. However, piecing together these solutions is a surprisingly formidable challenge.

To appreciate the problem, how one would typically approach unwrapping the data is to write a program that iterates through the reconstructions and detect sharp changes in position. Once the discontinuity is found we transform the data into the Z1/Z2 space and linearly adjust by integer multiples of $\lambda/2$ such that the discontinuity is minimized. The way in which we can detect these discontinuities is to look at the derivative as finite differences. The big problem is that neural networks are smooth mappings from input to outputs and in this case the outputs are also relatively smooth thus it makes recovering the original discontinuities difficult see, figure 4. Although it appears to be easy to detect by eye, to detect these sharp changes numerically is harder because numerically because they appear smoother and can be confused with inperfect reconstructions. Thus we're met with the wrapping problem.

[1] Code link

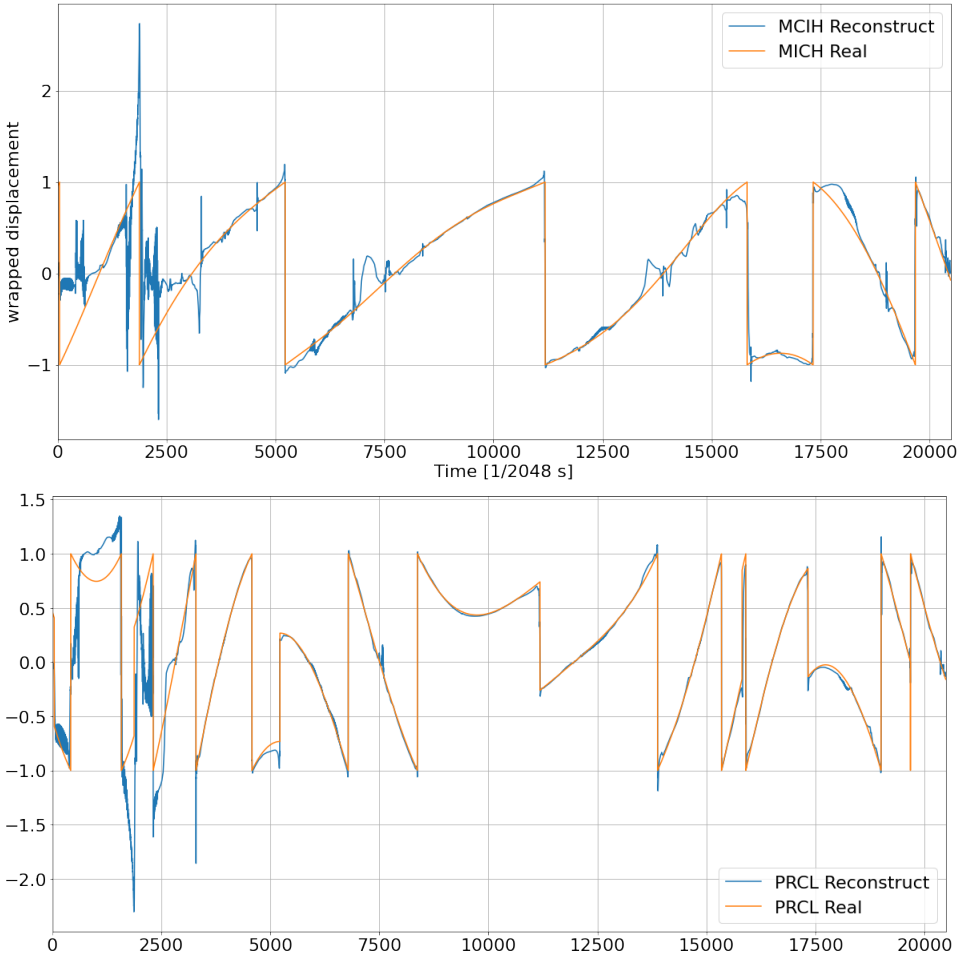

Figure 4: These are the reconstructions for PRCL and MICH.

3.2 Control Problem

4 Attention Based Estimators

Perhaps we can defeat the wrapping problem, by ignoring wrapping problem? We approach the problem with *brute force*. We assume that given sufficient history of signals, we can hope to extract unique solutions for the input signal. The major challenge here is that even the GRU's used in the original implementation still fail to generalize to *very* long sequences of data. The problem of retaining memory of various parts of a long sequence is a well-known shortcoming of recurrent neural networks. Luckily in 2017, a novel approach in the domain of natural language processing sought the advancement of transformers and the use of self-attention which effectively solved this problem[4].

4.1 Self Attention

In principle, attention is an intuitive concept[2]. Traditional attention helps assigns a weight to certain inputs of data as to how important they are to the problem they are trying to solve. In technical terms, we call this a query. However there is an issue with this, we lack context! For example, in the sentence, "I picked up some cash from the bank and went for a swim at the bank of a river" the word bank has 2 different meanings. Humans interpret this by looking at the context. To get machines to do the same is called self-attention. We look at the relationship of the word "bank" with the word "cash" and then we look at the relative attention of the words "bank" and "river" to infer these meanings. We look at the attention relative to words in the sentence itself, hence self-attention. This attention gives context to the data. This effectively defeats the memory problem because the model has no recurrent element! We do not need to engineer a means of holding a single "memory" unit when the model has completely freedom to attend to any input information freely at any point.

Armed with intuition, we build such an algorithm [3][4]. We have three main elements, the KEY, the QUERY, and the VALUE each will be three separate linear layers 5. This is motivated by how retrieval systems work. We have a query, and we find the matching key and return the value. Attention is this search process. We only pay attention to the most similar items that we look for and disregard the rest. This suggests that we want to find the highest amount of similarity with itself as this is self-attention. Hench the KEY and the QUERY will have the same inputs to each linear layer. Concretely one finds similarity by looking at the cosine similarity. This is easily computed via the dot product. To do this with the matrices of the KEY and the QUERY, we simply matrix multiply the outputs of the linear layers. This encodes the ATTENTION FILTER. We then scale this layer and compute the softmax of this filter to arrive at numbers that sum to 1. Then we multiply this with the VALUE, which up until now is just the input passed through a linear

[2]Good video on general attention here
[3]tutorial on transformers KERAS here
[4]Very good Youtube video on self attention here

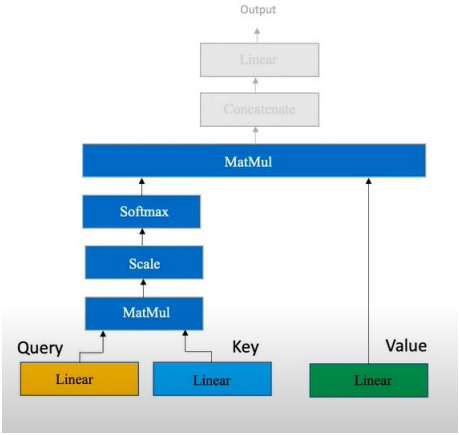

Figure 5: This is a concrete diagram showing the flow of data for self attention.[4]

layer. This matrix of numbers then helps us weigh and select the values from the original input and thus gives us the FILTERED VALUES. This process is called self-attention with a single head. If we want multiheaded self-attention, we repeat the entire process but with multiple layers and concatenate the final results together!

Attention works well, however, there is a key problem, these models don't consider ordering of the data. One of the transformer's great power is that it processes data in parallel instead of sequentially. This means that in the layer for self attention the aggregation is done on its own. More concretely, the attention values will always be the same no matter the order in which they appear. If we think about it, all the attention mechanism is doing is computing a weighted scalar between any pair of data in the sequence this scalar is created with 0 information about positions. However this would be a problem because in reality we know position affects how much attention is given to each data point. Thus before the transformer step we need to encode the positions, footnote Good video on positional embeddings here.

To fix this, we apply a positional embedding of the input, whereby we take the embedded data (in NLP terms means converting text to vectors) we learn an embedding by adding vectors to the embedded word vector. This small offset helps encode information about the positional information of the vector WITHOUT changing the contextual representation of the input. For example, the appearance of the word "Queen" as the second word in the sentence should appear nearby in the vector space from other appearances of the word "Queen" but they all should nonetheless be slightly different from each other depending on where in the sentence they appear. However, they can't be so far apart such that the word "QUEEN" appears to be an entirely different world altogether! These "offsets" are what is learned by positional embeddings.

Here we arrive at the second major issue, positional embeddings work well for language tasks, however, general time series data are not words. NLP tasks involve discrete chunks of inputs, that are finite, for generic time series problems, we don't have that luxury as data is continuous and infinite. Hence positional embeddings become a problem. The solution involves representation learning with an algorithm called the TIME2VEC embedding technique[5] [5].

[5]Good Blog post on implementation of time2vec here

4.2 TIME2VEC

Time to vector is an attempt to embed temporal information in time-series data. Firstly there are 2 major requirements for this embedding[5].

1. Embedded data should be invariant to time rescaling. In other words, if the data were taken at increments of 1 day or 2 months, or 3 picoseconds it shouldn't affect how the data appears in the end. 2. Representations of time should both capture periodic patterns and nonperiodic patterns [5].

To satisfy these constraints we (of course very naturally) arrive at the following formulations[5]. The i is the index of the number of kernels, τ is the timeseries data itself. The w_i, φ_i are the learned parameters in which the

$$\mathbf{t2v}(\tau)[i] = \begin{cases} w_i\tau + \varphi_i, \text{if} i = 0, \\ \mathcal{F}(w_i\tau + \varphi_i), \text{if} 1 \leq i \leq k \end{cases} \tag{1}$$

I have yet to truly appreciate the mathematics of why this works but, concretely the first element represents the linear behaviors of the data and the second represents the periodic behaviors of the data where $\mathcal{F}$ is described as the a periodic activation function. Read more here [5].

Armed with these new techniques, we approach defeating the wrapping problem.

4.3 Results Stage 1

In stage 1, we want to first and foremost, ensure that we can at least replicate the performance of the GRU on wrapped data with the transformers. We implemented the following model:

Time2Vec embedding Layer [no hyperparameters]

6 X Transformer Blocks [embed dim = 12, fully connected encoder =1024 , num heads=8]

5x Fully Connected [RELU, dims= 4096,800, 128, 32, 1]

Transformers can be difficult to train, as such we need a custom "warm-up" period whereby the learning rate is low but grows exponentially to the initialized learning rate. Then the learning rate decays for the remaining period until its reached a base learning rate. The warmup learning rate is$1 \times 10^{-6}$and the peak learning rate is 1×10^{-3} and the base learning rate is 1×10^{-6} the warmup time is 500 epochs[6].

Note: we were only able to achieve comparable performance with the GRU models if we trained separate neural networks for the PRCL and the MICH data.

[6]code here

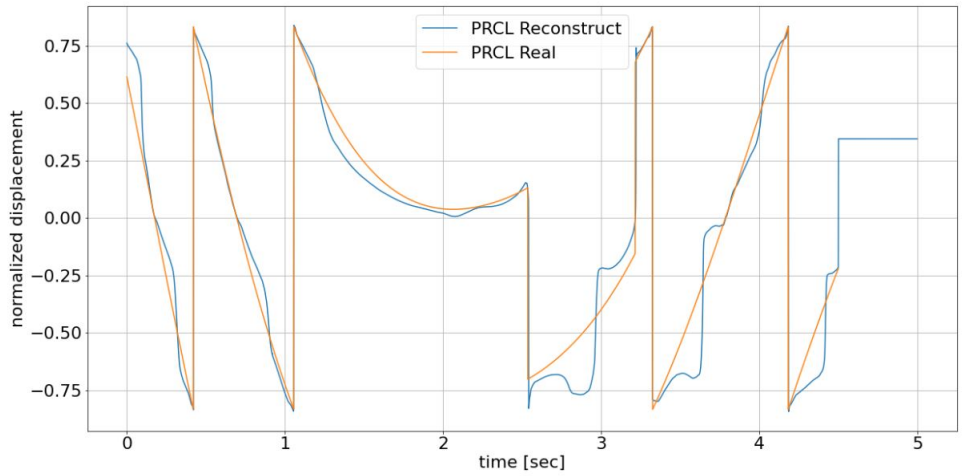

Figure 6: We see that the result is relatively similar and on par with the GRU model.

Here are the reconstructions after training in 6

We note that the training loss on the same data had a factor of 2 mean squared error more than the traditional recurrent model.

4.4 Stage 2

In stage 2 we now try and and produce a transformer model that can reconstruct the positions of unwrapped data. However before we attempt such we first train and evaluate a baseline model using the GRU model on unwrapped data as a benchmark

4.4.1 Baseline GRU Model on Unwrapped Data

To be true to the problem at heart we use the same model design as with the wrapped data except we extend the number of timesteps the GRU sees during training. In this case we extend it to 5 seconds of data. This was chosen due to memory constraints on the GPU when comparing this with the transformer model later on. Transformer has more memory intensive operations and was thus a bottleneck.

We have made the assumption that given sufficient amount of data we can accurate reconstruct the unique positions despite the non-uniqueness of the signals corresponding to any given position.

Let us see how a GRU might perform against such a setup consider figure. 7

The model was trained with early stopping with 300 epochs worth of patience. The model training only lasted 300 epochs since the loss does not chnage and stays at a constant 0.07 MSE.

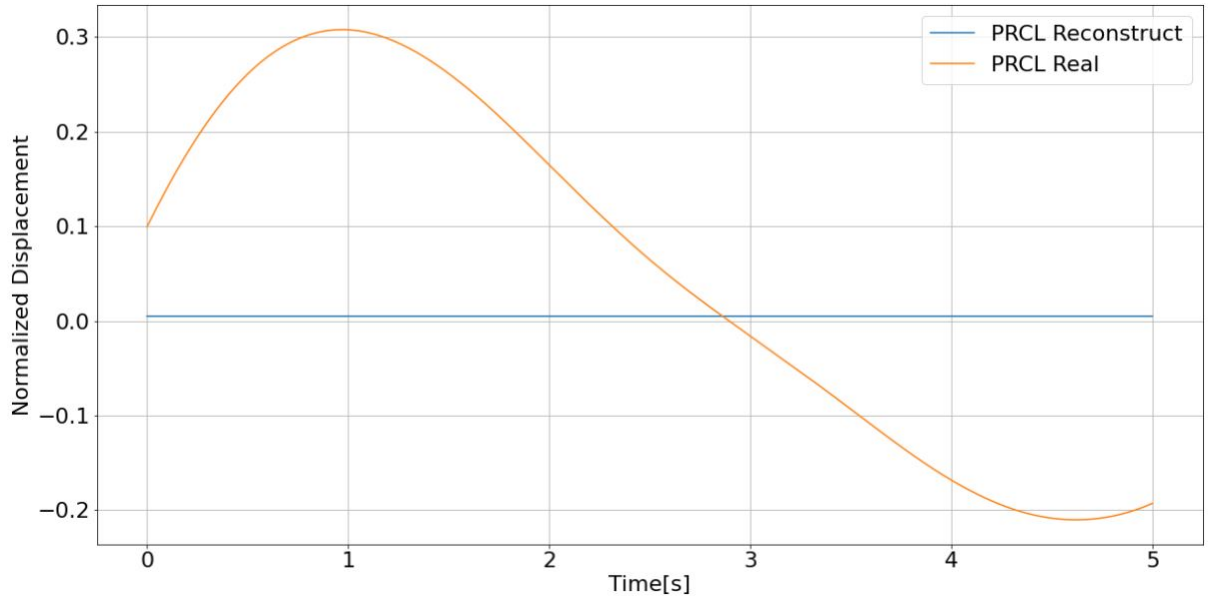

Figure 7: We see the predictions from the GRU is just a straight line.

We can directly conclude that the GRU is unable to produce the real unwrapped positions. There are two potential reasons: 1) the GRU suffers memory issues due to long sequences or 2) There isn't enough data to inform the model which solution to pick and thus can not solve the wrapping problem. We will investigate the issue by investigating the attention based model.

4.4.2 Attention Based Transformer Encoder Model

We now take the same attention based model used for the wrapped data but instead we are now training it on unwrapped positions. Should our assumption be correct, that this is indeed a memory issue with the GRU, then an attention based Transformer should alleviate such a problem. Below we see a reconstruction after training it for 300 epochs with early stopping with a patience of 300 epochs. We once again note that the model produces a constant error of 0.06 MSE during training. Consider the reconstructions below in figure 8

We see that the model clearly fails at reconstructing the positions as it predicts a constant function which happens to be the average of the positions. We note that this is most likely not a failure with the algorithm but an issue with the data. Should the data provided by solve-able with our technique it could fail however, it should not fail as spectacularly as it did here. If it was a matter of number of parameters or the number of layers etc then it should hint at at least possible variations. When a model is a constant it could hint that the problem might not be well defined for the approach or there is a fault in implementation. As the baseline test with wrapped data we ruled out the possibility of the latter.

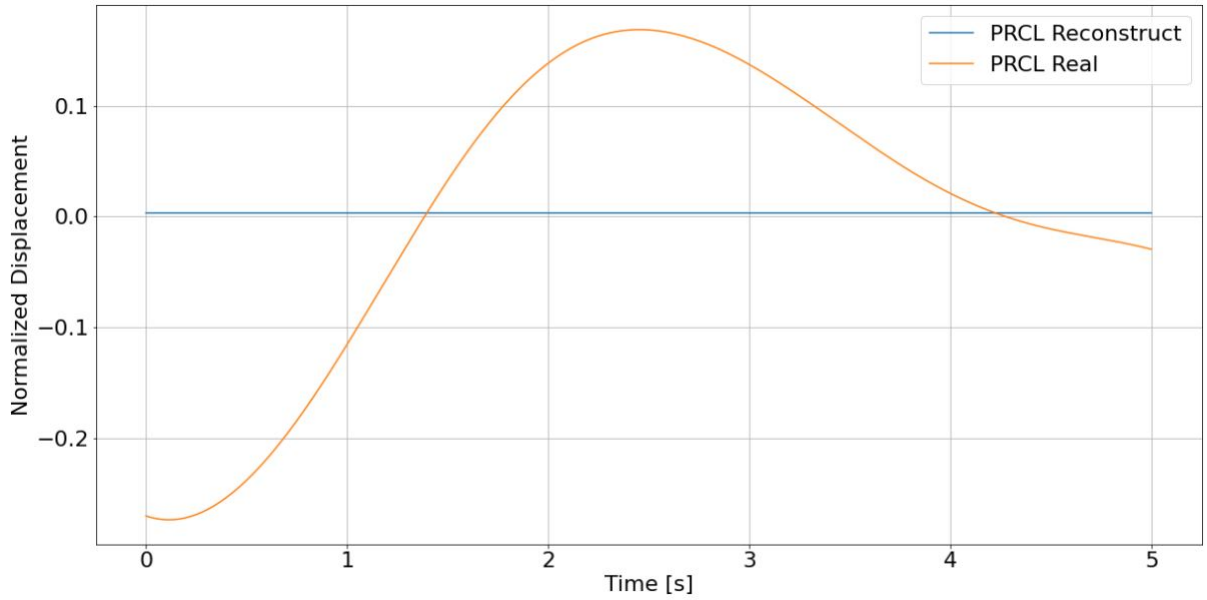

Figure 8: We are getting just a constant function once again.

4.4.3 Attention Based Transformer Encoder Model Part 2

Despite our unsuccessful attempt, we further pursue this approach with a slight twist. During the data generation process we restrict the initial start to be within the $\lambda/4 \rightarrow -\lambda/4$ range. The idea is that if we restrict the model to see data that start from the same interval range, the model can learn to reconstruct positions starting from the same initial state. This could potentially remove some of the ”confusion” during training.

We note the results below. First note the training loss curve is jagged this time rather than a constant, 9. This is in part due to the cycles of 10 epochs where we regenerate data. However we note that during each cycle the training loss remains relatively constant indicating its not learning anything.

We once again are faced with the same situation, the model fails to learn meaningful representations of the data see the reconstruction in figure 10. This let us conclude that the attention based methods appear unfruitful from our investigation.

Note: this does not mean we've *proven* that attention based models will always fail for our problem, we just concluded that for this project this direction appears unfruitful and due to time constraints we decided to allocate time in other promising approaches.

4.5 Discussion On Attention Based Estimators

There are a number of reasons we decided to conclude this direction of investigation.

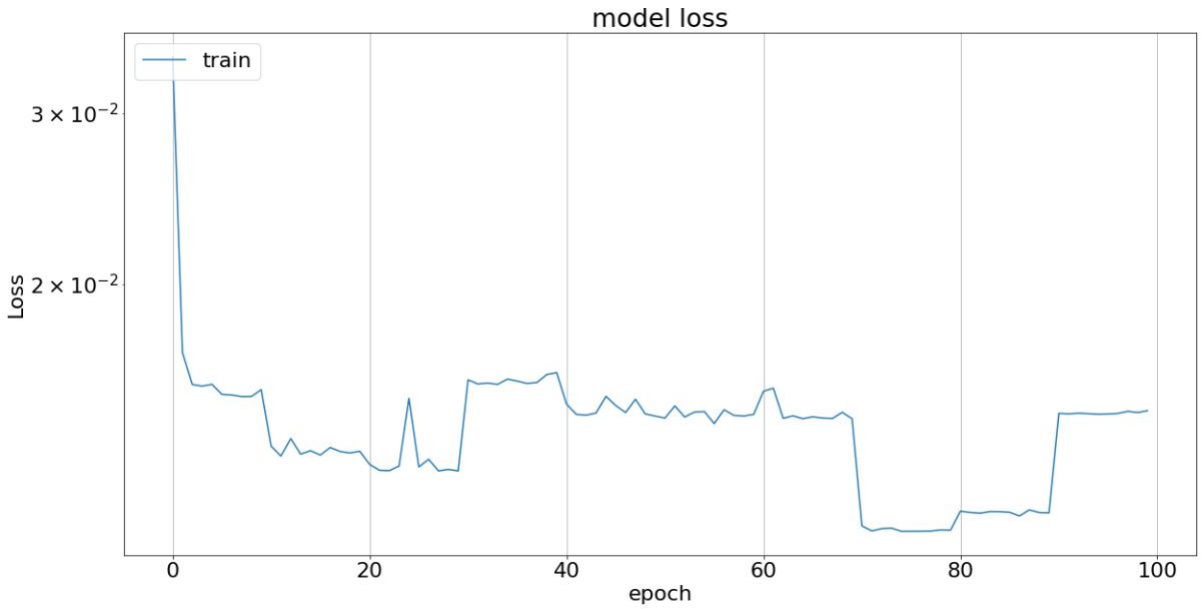

Figure 9: The loss curve for second transformer approach.

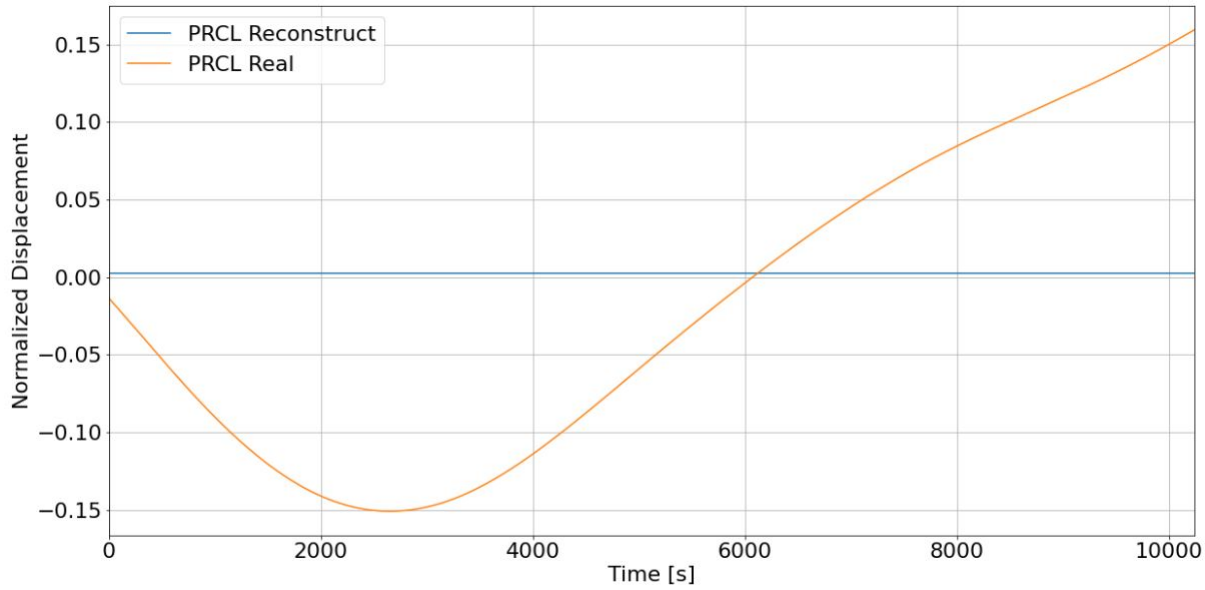

Figure 10: Reconstructions from the second transformer approach.

1. 5 - 10 seconds of hsitorical signals appear insufficent in uniquely reconstructing the unwrapped position.

2. Computing constraints and production implementation challenges.

Firstly, we see from our investigation with 5-10 seconds of historical data we fail to reconstruct the positions in a satisfactory way. From our experiments we conclude that more historical data is needed to have any hope of solving the wrapping problem and even then there is no guarantee of success. This brings us to the computing constraints. In production we cannot expect to save all the historical data and recompute them on demand with every sequential update in timestep. Unlike the GRU where we can simply take the last hidden state and store it in memory and then recompute a single forward time step given the hidden state which is computationally cheap compared to running 6 transformer layers each new timestep. Beyond the implementation issues, even just training the model on the GPU is an issue as 5-10 seconds is the maximal limit of data we can compute. Note that 10 seconds of data is approximately 20,000 time steps. If this were a language task even for large language models this is rather unorthodox. In a language setting this is like getting a large language model to read a prompt that the size of 1/3 of Harry Potter's Philosopher's Stone and be able to produce meaningful results from it. Not surprised this approach wasn't fruitful yet.

5 Velocity Position Uncertainty Estimate

Having approached the problem with brute force as with attention, we concurrently investigate more elegant means of reconstructing the positions. In this approach, we accept our doomed fate and wrap the positions of the estimator. The trick is, that we know that the velocity of the mirrors is unique. Given this information, we attempt to recreate both the velocity and the position of the data. Together we can attempt to fuse these two data points together to reconstruct the accurate positions.

To build intuition, for a given position, the next forward position will have an infinite number of solutions. With the knowledge of the velocity estimate, we can use that to select a position (out of the infinite solutions) such that the new velocity has the least uncertainty. To accomplish this task we can use the Kalman Filter [6] to do such magic. Thus we reduced the problem to: "can we build neural networks that can predict the inherent uncertainty in the prediction".

5.1 Probabilistic Models

We take inspiration from Variational Autoencoders[7]. Consider a classical feed-forward neural network for a regression problem. Now consider that the last layer splits into two layers which we denote as the mean and the other as the log variance. Then when optimizing the model, we compute the probability that the target appears in our predicted probability distribution. Intuitively this makes sense in capturing the error of the estimator. If the model was inherently poor at making

predictions, the variance should be high, this way there is a higher random chance that the model predicts the correct solution. However, if the model is highly accurate, then the variance should drop, or else there is a higher random chance it predicts data points far away from the target. Thus in theory as the model improves, the variance should drop and the mean should approach the target value. The PDF we chose is a Guassian given by:

$$P(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{1}{2\sigma^2}(-(y-\mu)^2)} \tag{2}$$

5.1.1 Position and Velocity Model

Let us take this probabilistic model and try to reproduce both the position and the velocity and their associated uncerainties. To do so we construct two models one for position and its uncertainty and similarly another for velocity. These models are identical in architecture.

The design is the exact same as the GRU baseline model except for the last layer where it splits into a mean and a standard deviation which is the error. The standard deviation uses ELU activation instead of RELU or linear.

We then construct a loss function which computes the negative log of the probability density function of a Gaussian given the mean and standard deviation produced by the model and the value x which is the actual position or velocity. For given position or velocity y, and where μ, σ are the associated mean and standard deviation, we want to maximize the likelihood.

$$\max_{\theta} P(y|\mu,\sigma^2)) \tag{3}$$

This is an equivalent optimization problem of maximizing the log likelihood which is the same as minimize the negative log - likelihood.

$$\min_{\theta}[-\ln(P(y|\mu,\sigma^2)))] \tag{4}$$

We implement this in the code as a VNET as this takes inspiration from variational autoencoders.[7]

5.1.2 Position and Velocity Training Results

TBD we have to rerun the training results because of wrapping problems.

[7] code: here

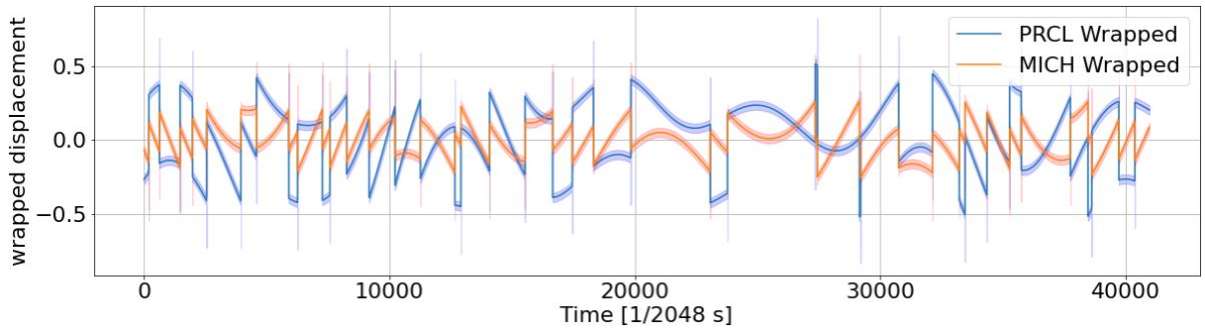

Figure 11: Wrapped positions with shaded area as the error.

5.1.3 Kalman Filter

The last part of the puzzle is to implement the Kalman filter for our specific use case. As usual we begin with intuition. Firstly we have both velocity and position data and their associated uncertainties.

Now for every given new time step we can predict the wrapped position and we can retrieve its velocity. Now theres two things we can do with this information, first we can take the previous position and take the velocity and use simple kinematics estimate the next position. Secondly we can also just get the next wrapped position, however to get back the unwrapped position we know there is an infinite number of these each adjusted by some multiple of the wavelength. As the usual wrapping problem states, which of these infinite solutions do we pick? Since we can use kinematics to estimate a forward position we can use this information to help us "select" one of the infinite solutions.

What we do is we can take the uncertainties to propagate forward the errors when constructing our kinematic solution for the next time step, this will be our prior distribution. Then we lay out all the possible solutions forming a Gaussian mixture from our next predicted wrapped position. Each of the solutions will be a candidate likelihood distribution. Now for each possible solution to pick from, we can obtain the corresponding posterior distribution by using Bayes Theorem. We simply multiplying the two distributions together. What we want to pick is the likelihood distribution that gives us the maximum posterior probability. This posterior distribution with the highest peak will be the best fitted position estimate!

Let us formalize this in an algorithm, please see pseudo code 1 for the structure of the algorithm. Let us define the model $p(\theta_p, x_t)$ for the position and $v(\theta_v, x_t)$ for velocity with mean and covariance μ_{p_t}, Σ_{p_t} and μ_{v_t}, Σ_{v_t} where the subscript denotes either position or velocity at times t. x_t is the optical signals received at time step t. θ is the weights. The total time is T. We denote P^* as the best prediction of position. $\hat{p}_t$ is the position predicted by the dynamics. dt is the change in time.

Algorithm 1 Kalman Filter

Initialize: we set m as the number of solutions we search for.
Initial estimate
$\mu_{p0}, \Sigma_{p0} \leftarrow p(\theta_p, x_0)$
$\mu_{v0}, \Sigma_{v0} \leftarrow v(\theta_v, x_0)$
Best estimate initially comes directly from the model.
$P_0 \leftarrow \mu_p$
while $t \leq T$ **do**
We first use the dynamics and update
$\mu_{p_{\hat{t+1}}} \leftarrow \mu_{p_t} + \mu_{v_t} \cdot dt$
$\Sigma_{p_{\hat{t+1}}} \leftarrow \Sigma_{p_t} + \Sigma_{v_t} \cdot dt^2$
Now we construct a Gaussian mixture of various solutions.
Get Wrapped position looking at one step ahead
$\mu_{p_{t+1}}, \Sigma_{p_{t+1}} \leftarrow p(\theta_p, x_{t+1})$
$\mu_{v_{t+1}}, \Sigma_{v_{t+1}} \leftarrow v(\theta_v, x_{t+1})$
We construct m possible solutions which have the same covariance but have shifted means we denote $\mu^i_{p_{t+1}}, \forall i \in (0, 1, \cdots m)$
Compute products of Gaussians
while $i < m$ **do**
$\mu^i_{t+1} \leftarrow (\Sigma^{-1}_{p_{t+1}} + \Sigma^{-1}_{p_{\hat{t+1}}})^{-1}(\Sigma^{-1}_{p_{t+1}}\mu^i_{p_{t+1}} + \Sigma^{-1}_{p_{\hat{t+1}}}\mu_{p_{\hat{t+1}}})$
$\Sigma^i_{t+1} \leftarrow (\Sigma^{-1}_{p_{t+1}} + \Sigma^{-1}_{p_{\hat{t+1}}})^{-1}$
Normalization Constant
$C^i \leftarrow \frac{1}{\sqrt{\det(2\pi(\Sigma_{p_{t+1}} + \Sigma_{p_{\hat{t+1}}}))}} \exp[\frac{-1}{2}(\mu^i_{p_{t+1}} - \mu_{p_{\hat{t+1}}})^T(\Sigma_{p_{t+1}} + \Sigma_{p_{\hat{t+1}}})^{-1}(\mu^i_{p_{t+1}} - \mu_{p_{\hat{t+1}}})]$
end while
Sort for the i that maximizes C^i. Denote the best value as i^*
$P^*_{t+1} \leftarrow \mu^{i^*}_{p_{t+1}}$
Now we update the positions
$\Sigma_{p_{t+1}} \leftarrow \mu^{i^*}_{p_{t+1}}$
Update the velocity with the predictions from before.
end while

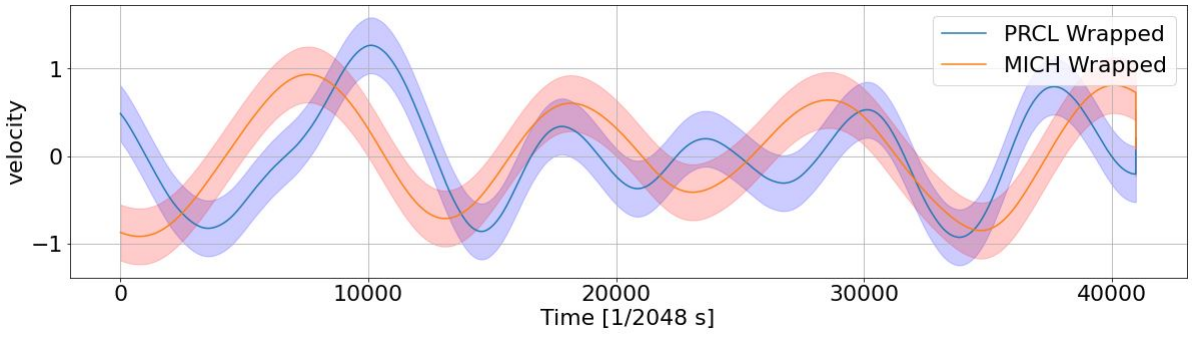

Figure 12: The velocity where errors are the shaded.

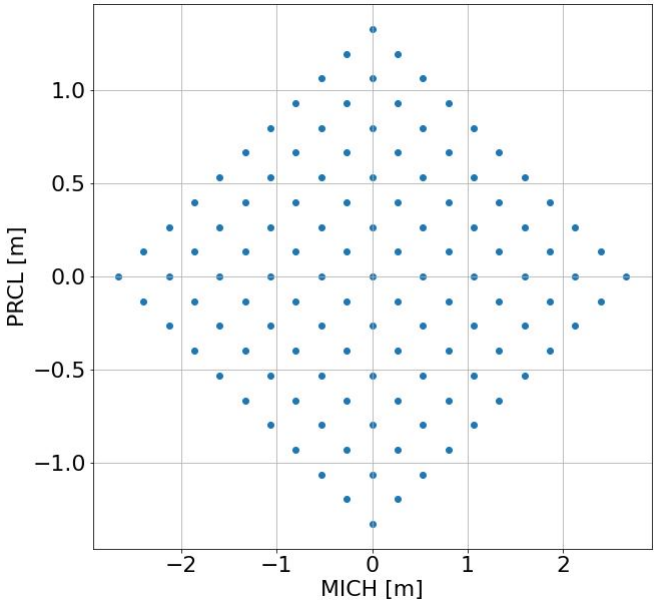

Figure 13: This is a grid of valid adjustments to the positions that would result in the same signals.

5.1.4 Kalman Filter Results

We then construct a grid of possible solutions, each with a Gaussian centered around them. This can be seen in figure 13. Thus we can finally implement the algorithm to stitch back our solutions. We can see the results in figure 14

6 Deep Deterministic Policy Gradient

6.1 Motivation

Generally speaking, some of the greatest advances in machine learning come from increasingly abstracting away tasks and simply letting the machine do everything. This has loosely been the trend in ML research for a number of years. Back in the 'prehistoric eras', ML models used to work only with highly hand-crafted features for input, that was until the advent of deep learning when we saw these features learned implicitly by our algorithms. We increasingly try to move the human further away from the picture and surprisingly found huge improvements from computer vision tasks to all the way to NLP. This similar philosophy is adopted by some of the most successful ML researchers tackling various kinds of hard problem. Quoting Deepmind's CEO Demis Hassabis, "let the gradients run through everything". We approach hard problems initially with an intermediate solution that often requires lots of hand-tuning, to which then practitioners try to generalize the approach to the bigger problem. We saw this with the invention of the CNN, we saw this with the invention of Alphafold1 $\rightarrow$ Alphafold2[8], we saw this with the invention of transformers, and we will probably continue to see this trend until either we create the Terminator

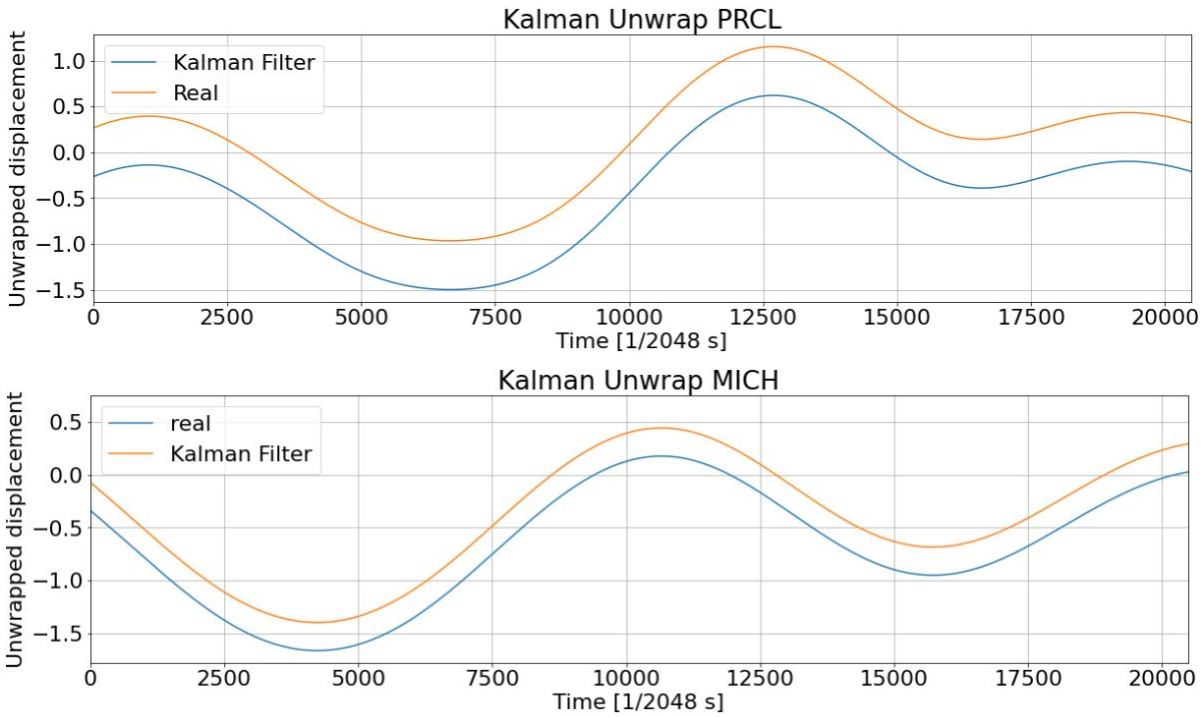

Figure 14: This shows that we've successfully unwrapped the positions with our approach!

and kneel before our AI overlords, or figure out some other way to solve intelligence. This is scary because we lack agency, but time and time again we've been proven wrong, in a good way. This possible upside in "superhuman" abilities alone is enough to warrant at least an investigation into its potential. Because of this, despite my supervisor's skepticism in letting me build an AI to control LIGO mirrors directly, I wanted to approach this problem in the most general way possible which is directly learning the controls of these mirrors.

More concretely, RL could be interesting because we know that giving only the information about the signals might not be enough to recreate the positions. What might hint that RL can solve the problem is that we can provide models with additional information, specifically the reward signal at certain time steps. The optical signals received plus the expected reward for certain actions could help us gauge what kinds of controls engage. This could provide unique solutions to the control problem. However, this is not trivially clear that this is the case and further investigation is needed. I briefly diverted some time on this problem.

6.2 DDPG Method

In classical approaches to reinforcement learning, we typically use tabular methods. These algorithms work by searching a space of actions and storing these optimal actions at a given state in Q tables. These tables help inform how to perform tasks given the state you're at. Typically these solutions could involve dynamic programming, temporal difference learning, or Monte Carlo tree searches. These methods are only tractable when the state space is finite. However, in most prac-

tical cases these state spaces are near-infinite. For this, we need approximate solution methods. A well-known means of approximation is to use neural networks such as Deep Q learning [9]. Given some arbitrary continuous state, we can predict the expected reward value for a particular action, this is the q function in RL terms. However, neural networks have a finite number of predictions, thus this method also relies on the fact that the action space itself is finite. What happens when the action space is also infinite? Just like how we used a neural network to predict the action's value we use a second neural network to then evaluate the network's performance in an ACTOR and CRITIC setup. Similar to a generator and a discriminator in GANs for computer vision, the actor, given the input produces actions that are continuous values. Then we have a critic that takes in the state, and the predicted actions and tries to predict the expected discounted reward on that action taken. This allows an agent to maneuver and learn from a continuous state space and action space which is important in our setup.

To use this technique we have numerical simulations that provide actions based on the continuous state of the optical signals. We also want to predict continuous actions and direct how much force each of the servos should engage to drive down the motions. Thus the first RL approach we should take would be DDPG methods[8] [10].

6.3 RDPG Method

Not only do we need to implement a DDPG method, we also need to implement a recurrent version of DDPG called RDPG. This is in part because we know in order to predict the dynamics we need to get the velocity somehow and this requires us to use a recurrent model. Now this RDPG will use the exact same set up with the original DDPG method except the replay buffer now indexes a history of observations and actions to which we then feed the actor and the critic. The training step is identical as a traditional DDPG except both models will get both the state/observation history and the corresponding history of previous actions. We implement the the algorithm from the following paper [11] section 3.1.2

6.4 First Iteration RDPG

For the first iteration of the RDPG we wanted to iterate quickly and thus we have resorted to down sampling the simulation from a 2048 Hz to 20 Hz. This is meant as demonstration purposes to iterate on various ideas before we tackle the bigger problem.

We implemented a Critic with the following architectures. Signal history is fed into a GRU with 128 units followed by another GRU of same size. Then we feed the action history into another set of GRU with 128 units followed by a second GRU of the same size. The outputs of the two are then concatenated and fed into a feedforward neural network of size 512 followed by hidden layers of 256, 32 and 1 respectively. The model uses RELU activations. The Actor is symmetrical meaning design wise it is the same except it has 2 outputs and the outputs are activated using tanh

[8] code: here

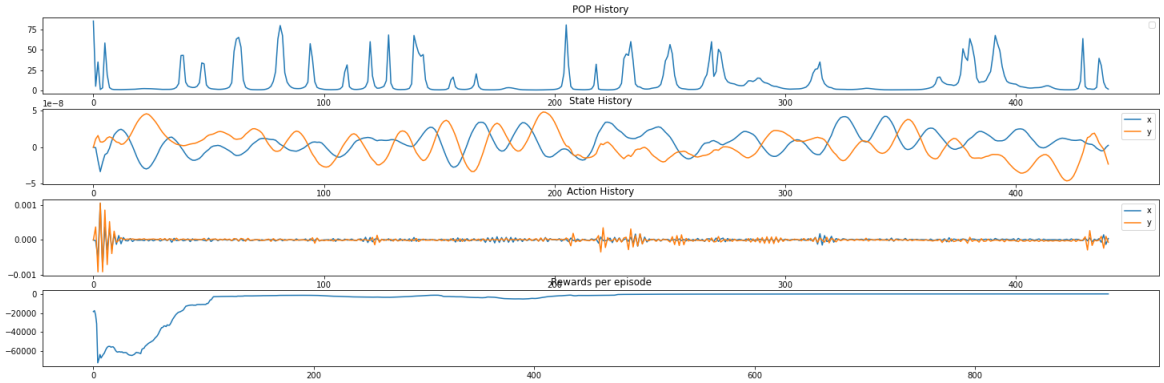

Figure 15: This shows promising results! Note that the movements are on the order of 1/100 of the wavelength and at this point the model starts producing signals that are high and constant for a short duration of time.

since the actions need to be bounded.

We also have various hyperparameters. We restricted actions to a force of 10^{-3} and we gave the initalization of the last layer of the actor $\pm 4 \times 10^{-4}$ so that values initially stay close to 0. For exploration we added noise to the predictions in the policy with a variance of 10^{-4} with mean 0. We also have soft updates and gave $\tau = 10^{-6}$

Another caveat to this iteration is that we are using the real positions as inputs for the reward function. The reward function takes the negative MSE of the position, thus the smaller the movements the greater the reward the model receives. The reason why we have not used only the signals for the rewards is because crafting such a reward function is difficult and we want to first test if this idea could even work. Hence the first iteration.

6.4.1 First Iteration Results

We see from first glance, the results appear promising! We see that the model has begun to learn and is improving over time. Now the actual performance is still quite poor, we haven't spent time tuning or investigating this just yet. We believe this preliminary result is enough see figure 15

6.5 Reward Function

Reward function or the reward signal is the objective of the problem. On each timestep the environment sends the agent a single number and the goal is to maximize this reward over a long term period. We know our goal is to push the mirrors into a linear regime. We can detect this linear regime by looking at the signal responses specifically POP. When the power of the optical signals shown by POP is high and constant, then we will know we can easily acquire the lock.

To mathematically formulate this, we craft a reward function which is the inverse square of the derivative of the power. This way, when the change is constant we get maximal reward whereas any large change won't receive any. We square it to make the reward invariant to the sign of the derivative. We add an ε term to make this numerically stable. We also have a step function for the reward. Given high power, we reward the model for achieving such. S_t is the state received from the environment i.e the signals at time t. We set some *const* as the threshold for when we consider it to be high enough power. We set α as a constant hyperparameter in scaling the rewards.

$$R_t = \begin{cases} [(\frac{dS_t}{dt})^2 + \varepsilon]^{-1} + \alpha & \text{if } S_t > const \\ -\alpha, & \text{else} \end{cases} \tag{5}$$

6.6 Results

TBD

7 Forward

In the first 3 weeks of the project, we have finished implementing nearly all the algorithms we hope to investigate. We have implemented baseline models, attention-based estimators, velocity-position uncertainty estimators, and DDPG models. We currently plan to dedicate the rest of the time to tuning and investigating if these implementations work.

References

[1] Cho, K., van Merrienboer, B., Bahdanau, D. & Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches (2014). URL https://arxiv.org/abs/1409.1259.

[2] Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate (2014). URL https://arxiv.org/abs/1409.0473.

[3] Chollet, F. *et al.* Keras. https://keras.io (2015).

[4] Vaswani, A. *et al.* Attention is all you need (2017). URL https://arxiv.org/abs/1706.03762.

[5] Kazemi, S. M. *et al.* Time2vec: Learning a vector representation of time (2019). URL https://arxiv.org/abs/1907.05321.

[6] Kalman, R. E. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering* **82**, 35–45 (1960).

[7] Kingma, D. P. & Welling, M. Auto-encoding variational bayes (2013). URL https://arxiv.org/abs/1312.6114.

[8] Jumper, J. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature* **596**, 583–589 (2021). URL https://doi.org/10.1038%2Fs41586-021-03819-2.

[9] Mnih, V. *et al.* Playing atari with deep reinforcement learning (2013). URL https://arxiv.org/abs/1312.5602.

[10] Quillen, D. *et al.* Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods (2018). URL https://arxiv.org/abs/1802.10264.

[11] Heess, N., Hunt, J. J., Lillicrap, T. P. & Silver, D. Memory-based control with recurrent neural networks (2015). URL https://arxiv.org/abs/1512.04455.