RBE 577: Homework 1 Report

Azzam Shaikh
Sept. 29, 2024

Introduction

The purpose of this assignment is to implement the work proposed by R. Skulstad et. al called "Constrained control allocation for dynamic ship positioning using deep neural network." The paper introduces a novel approach for control allocation using deep neural networks, specifically deep auto-encoders.

Control allocation refers to the method of distributing the desired forces and moments required by the vehicle to the control efforts in a system with redundant actuators.

The focus for this assignment will be to implement the neural network proposed and evaluate the capabilities of the model.

Methods

The implementation of the paper includes three major aspects: (1) artificial data generation, (2) a custom loss function, and (3) model development, training, and testing. Each of these sections and their implementation will be discussed in this section

(1) Data Generation

In the paper, in Section 3.1, it was described that the model was generated artificially. The dataset was generated from a range of force and angle command ranges for a ship, shown in Figure 1 below.

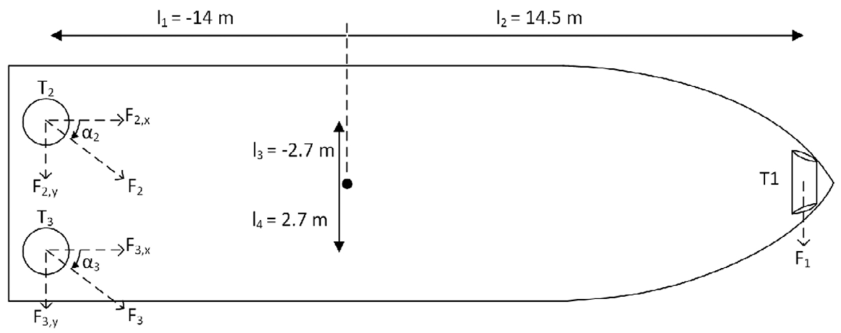


Figure 1: Boat model from the paper

The boat dynamics are controls by three thrusters, T_1, T_2, and T_3, For each thruster, the commanding inputs are F_1, F_2 and control angle α_2, and F_3 and control angle α_3. These inputs can be mapped to input vector u, as shown below:

$$u = [F_1, F_2, \alpha_2, F_3, \alpha_3]^T$$

The ranges described in the paper (Eq. 7) for the 5 inputs are shown below:

$F_1 \in [-10000, 10000]\ N$

$F_2 \in [-5000, 5000]\ N$

$\alpha_2 \ \in [-180, 180]\ N$

$F_3 \in [-5000, 5000]\ N$

$\alpha_3 \ \in [-180, 180]\ N$

Using these ranges, one million samples are drawn using a randomly initialized random walk process for each thruster command per mini-batch. The random walk model can be described as follows (reference):

1. Start with a random number of either -1 or 1.
2. Randomly select a -1 or 1 and add it to the observation from the previous time step.
3. Repeat step 2 for as long as you like.

Thus, for the case of u, a random force value from the defined constraints will be selected. From this random value, the next time step will add or subtract a defined angle or force step from the value, while being constrained within the prescribed ranges. This procedure will continue until one million samples are drawn. Since the paper states that the number of samples in a single batch is 1024, the actual number of samples generated for u is 999,424 samples. Thus, u is a $[999424,\ \ 5]$ vector. The u vector generated can be seen below in Figure 2.

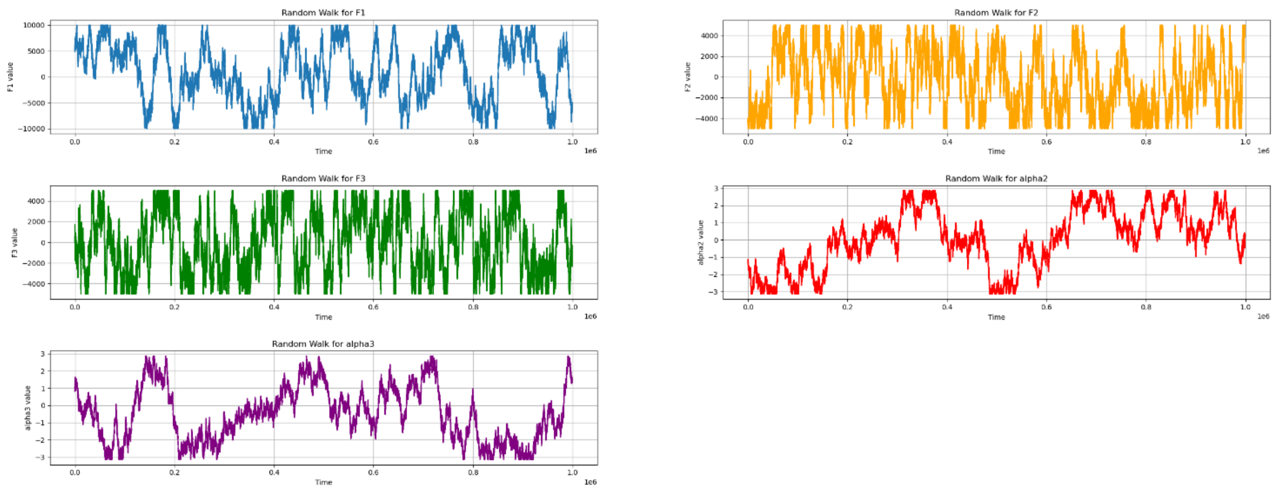


Figure 2: Generated u data via a random walk

After generating the u data, these force and angle vectors need to be converted to thruster commands, τ. This conversion is completed based on the following equation:

$$\tau = B(\alpha)u_f$$

Where u_f is a 3x1 force vector extracted from u, α is a 2x1 angle vector extracted from u, and B is the following:

$$B = \begin{bmatrix} 0 & \cos(\alpha_2) & \cos(\alpha_3) \\ 1 & \sin(\alpha_2) & \sin(\alpha_3) \\ l_2 & l_1 \sin(\alpha_2) - l_3 \cos(\alpha_2) & l_1 \sin(\alpha_3) - l_4 \cos(\alpha_3) \end{bmatrix}$$

Where l_1, l_2, l_3 and l_4 are defined in Figure 1.

Thus, the final equation for $\tau = B(\alpha)u_f$ is:

$$\begin{bmatrix} \tau_{surge} \\ \tau_{sway} \\ \tau_{yaw} \end{bmatrix} = \begin{bmatrix} 0 & \cos(\alpha_2) & \cos(\alpha_3) \\ 1 & \sin(\alpha_2) & \sin(\alpha_3) \\ l_2 & l_1 \sin(\alpha_2) - l_3 \cos(\alpha_2) & l_1 \sin(\alpha_3) - l_4 \cos(\alpha_3) \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$$

After completing this operation for the u dataset, the τ dataset is generated and is a $[999424,\ 3]$ vector. This is the dataset that will be fed into the model. The τ vector generated can be seen below in Figure 3.

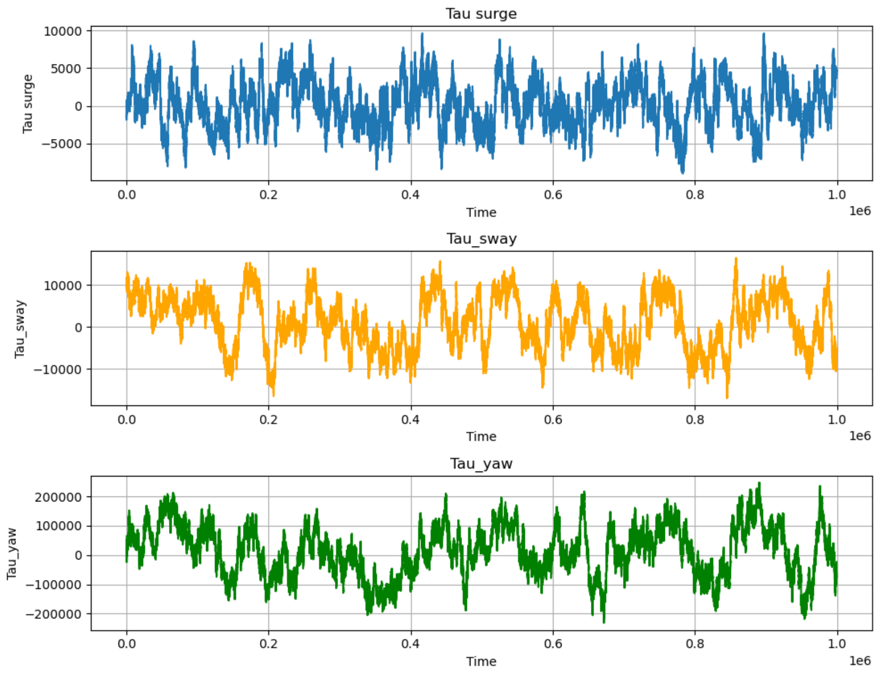


Figure 3: Generated τ dataset

Based on Figure 3, it can be seen that the data for the three variables have very different scales. Thus, this will impact the optimizers ability to tune the model. Thus, input normalization is implemented to ensure all three features of the dataset have 0 mean and a standard deviation of one. This will improve the gradient descent optimization as all parameters will have similar scales. After applying the normalization, the normalized τ vector generated can be seen below in Figure 4. It can be seen that all the features are now on a similar scale.

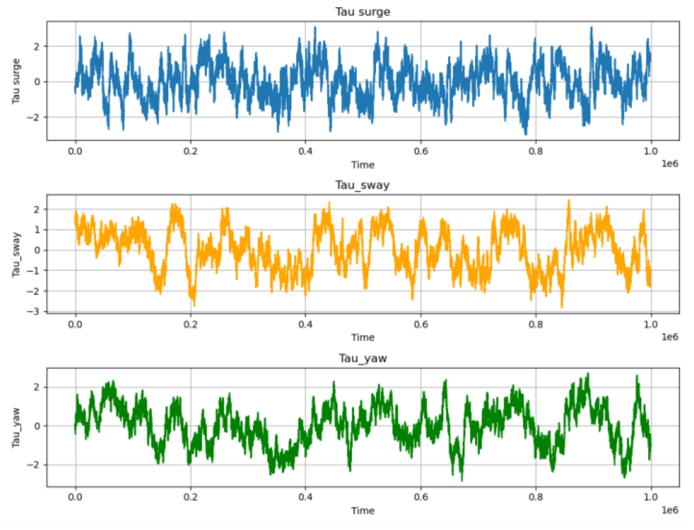


Figure 4: Normalized τ dataset

As mentioned previously, each batch of data will contain 1024 samples. Thus, the one million samples of τ was split into 976 batches. Thus, the τ dataset will be a $[976, 1024, 3]$ vector. A batch of τ can be seen in Figure 5.

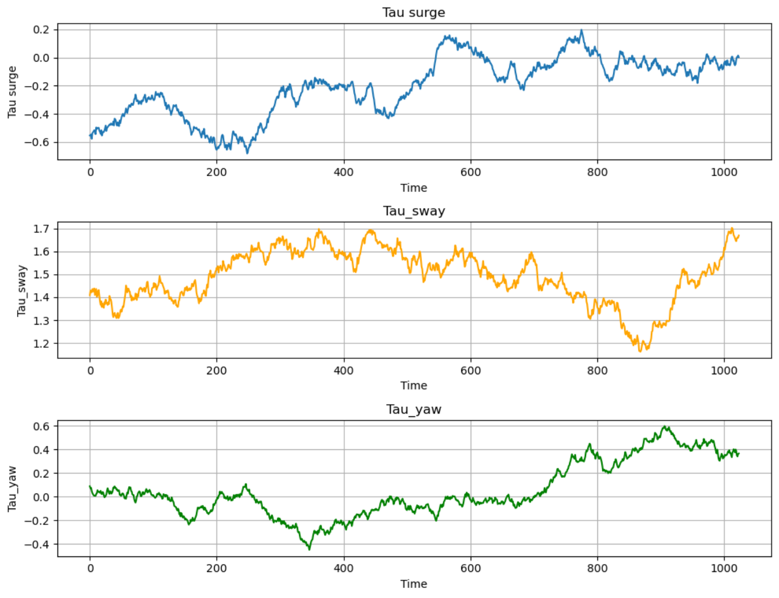


Figure 5: A batch (1024 samples) of the normalized τ dataset

From these 976 batches (million samples) of τ, 80% of the dataset be used to create the training dataset, and the remaining 20% will be used for testing. Using sklearns test_train_split() function, the 976 batches were split into a training set with 780 batches and a testing set of 196 batches. Since the function by default shuffles the dataset, a batch of the training set can be seen in Figure 6, which is different from the batch shown above.

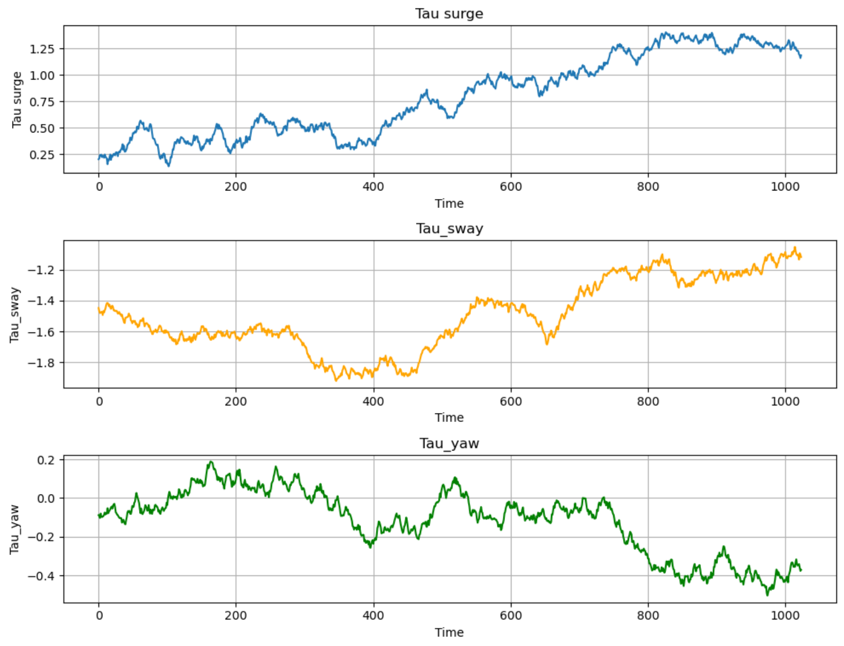


Figure 6: A batch (1024 samples) of the training set

Using this τ data, the training and testing dataset can now be passed to the model. The next section will discuss the custom loss functions.

(2) Custom Loss Function

The paper mentions that custom loss functions were implemented to keep the control allocation in the constraints of the ships magnitude and rate of the control vector within certain bounds and facilitating power minimization. This implementation of the loss functions mimics what an optimization-based allocation would do.

For the first loss function, L_0, the $\hat{u}$ that is output from the encoder is used to recalculate the τ values that it would generate. This will be referred to as the command input, τ_{cmd}. Using this value, the MSE loss between the ground truth input, τ, and the command input, τ_{cmd}, will be computed. This quantifies how well the encoder is mapping the desired values to its own $\hat{u}$.

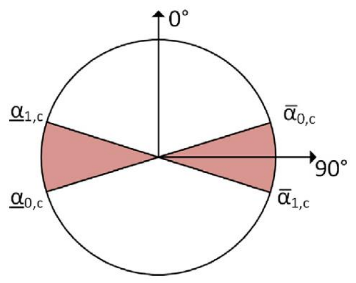
For the second loss function, L_1, the MSE loss is computed between the ground truth, τ and the output from the decoder, $\hat{\tau}$, will be computed. This quantifies how well the model regenerates the input.

For the third loss function, L_2, the function penalizes commands that exceed the max force and angle threshold set for each command variable. Thus, for the generated input variables created by the encoder, $\hat{u}$, each variable is compared against the max limit of the respective control input. Based on Table 1 in the paper, the max force value for F_1, F_2 and F_3 is 30000 N, 60000 N, and 60000 N, respectfully, and the max angle for α_2 and α_3 are both 180°. If the force or angle generated by $\hat{u}$ is bigger than the max force or angle, then a penalty will be applied.

For the fourth loss function, L_3, the function penalizes commands that exceed the max rate of change that the thrusters are capable of. Thus, for the generated input variables crated by the encoder, $\hat{u}$, each variable is compared against the max rate of change limit of the respective control input. Based on Table 2 in the paper, the max rate of change of the force are all 1000 N/s and the max rate of change of the azimuth angles are both 10°.

For the fifth loss function, L_4, losses will be applied based on the amount of power consumed by the system. The power consumption is equated based on the amount of force generated and raising it to the power of 3/2. This mimics the relation between the quadratic function relating thrust to engine RPM and a cubic function relating consumed power to engine RPM. The power to rotate the azimuth thrusters is not considered.

For the sixth loss function, L_5, losses will be applied whenever the angle generated by $\hat{u}$ is within the sectors shown below in Figure 7. This loss constraints the azimuth angles to avoid decreasing thruster efficiency due to disturbance of inflow velocities by neighboring thrusters.



0°
$\underline{\alpha}_{1,c}$
$\overline{\alpha}_{0,c}$
90°
$\underline{\alpha}_{0,c}$
$\overline{\alpha}_{1,c}$

Figure 7: Unallowed azimuth sectors as shown in the paper

All these losses are combined and applied with specific weighting, as described from Table 2 of the paper. The specific weighting of all the six loss functions are shown below.

$$k_0 = 10^0$$

$$k_1 = 10^0$$

$$k_2 = 10^{-1}$$

$$k_3 = 10^{-7}$$

$$k_4 = 10^{-7}$$

$$k_5 = 10^{-1}$$

The final loss value can be equated to the following:

$$L = k_0 L_0 + k_1 L_1 + k_2 L_2 + k_3 L_3 + k_4 L_4 + k_5 L_5$$

All the loss functions are implemented in the ConstrainedControlLoss() class in the script.

The final aspect is the development of the model.

(3) Model Development, Training, and Testing

The paper describes that the model of the deep neural network is an autoencoder, with the architecture shown in Figure 8.

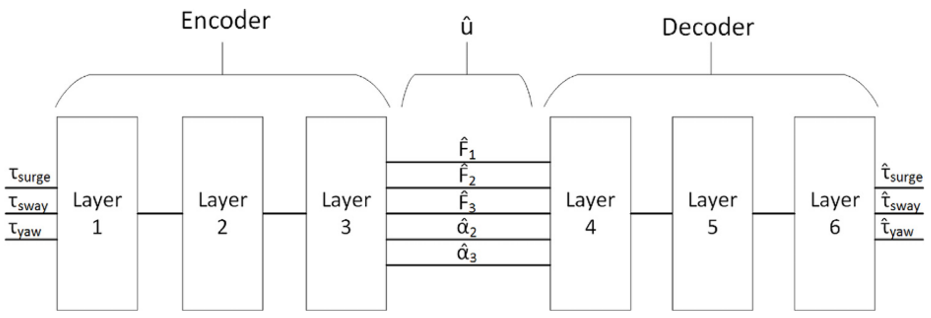


Figure 8: AutoEncoder model architecture presented in the paper

The encoder and decoder portion of the model each consists of three layers. The input to the decoder has three features, as does the output of the decoder. The output of the encoder and input to the decoder each have 5 features. The paper states that layers 1, 2, 4, and 5 consisted of LSTM nodes while the 3 and 6 consisted of a fully connected layer. In the case of this assignment, MLPs were implemented in place of the LSTMs while the fully connected layer was kept. Each layers inputs and outputs dimensions are described below:

	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6
Input dims.	3	4	5	5	5	4
Output dims.	4	5	5	5	4	3

The models input and output will have dimensions of 1024x3 while the encoder will have an output of 1024x5.

The activation function for the hidden layers of 1, 2, 4, and 5, are each LeakyReLUs while the activation function for the output layers of 3 and 6 are Linears. LeakyReLUs are selected as the it prevents creating dead neurons that can be created via negative numbers.

The model is contained in the AutoEncoder() class in the script.

After creating the model and the loss function, an optimizer can be selected. For this application, an Adam optimizer is used as it uses an adaptive learning rate. The learning rate is set to 0.001 and L2-norm is applied with a weight decay of 0.01.

With the necessary components defined, the model can be trained and tested. A training loop is implemented and is ran for 200 epochs. The paper does not specify the number of epochs, thus, based on trial and error, this provided the best results. During every epoch, the dataset is shuffled to prevent overfitting. The results of the training and testing is detailed below.

Results

After running the training and testing for 100 epochs, the training loss per batch sample and per epoch can be seen below in Figure 9 and 10, respectively.

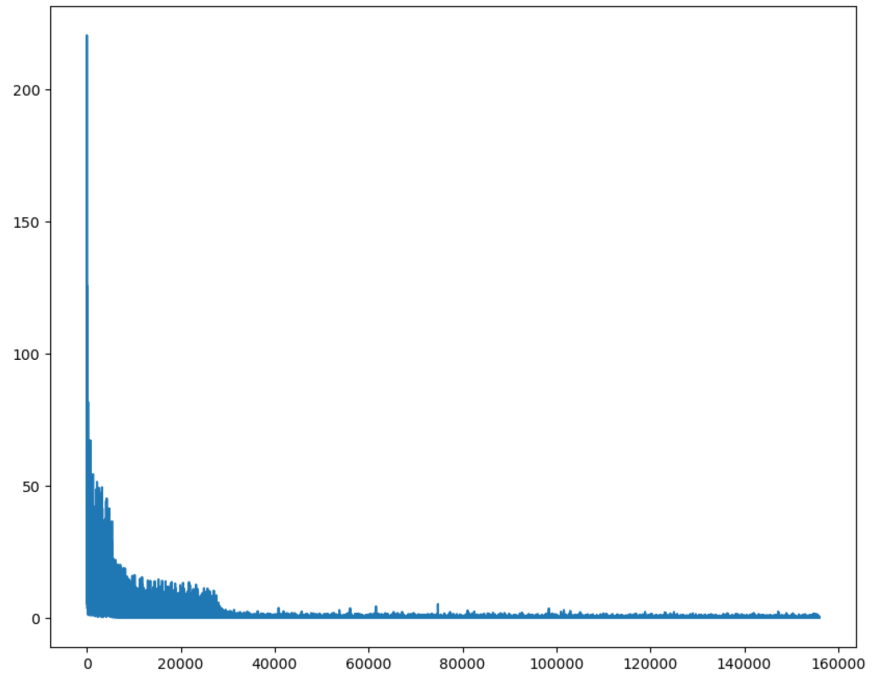


Figure 9: Training loss per batch sample

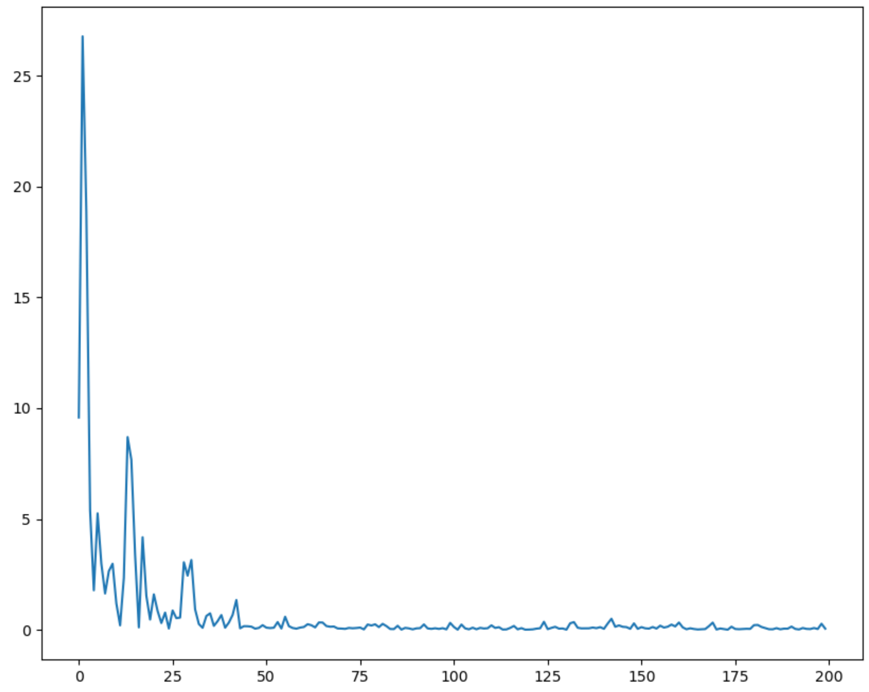


Figure 10: Training loss per epoch

The testing loss per batch sample and per epoch can be seen below in Figure 11 and 12, respectively.

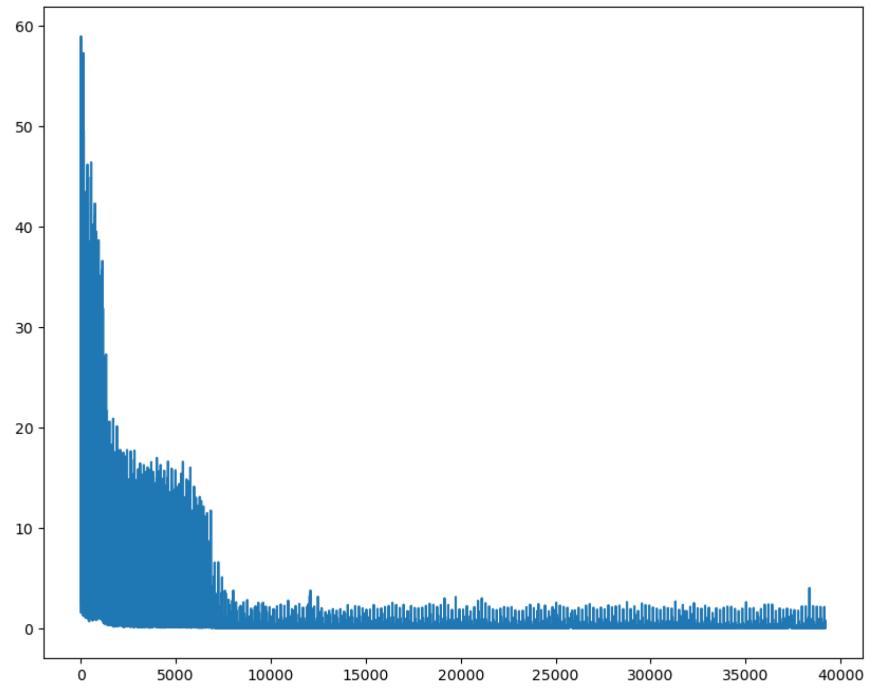


Figure 11: Testing loss per batch sample

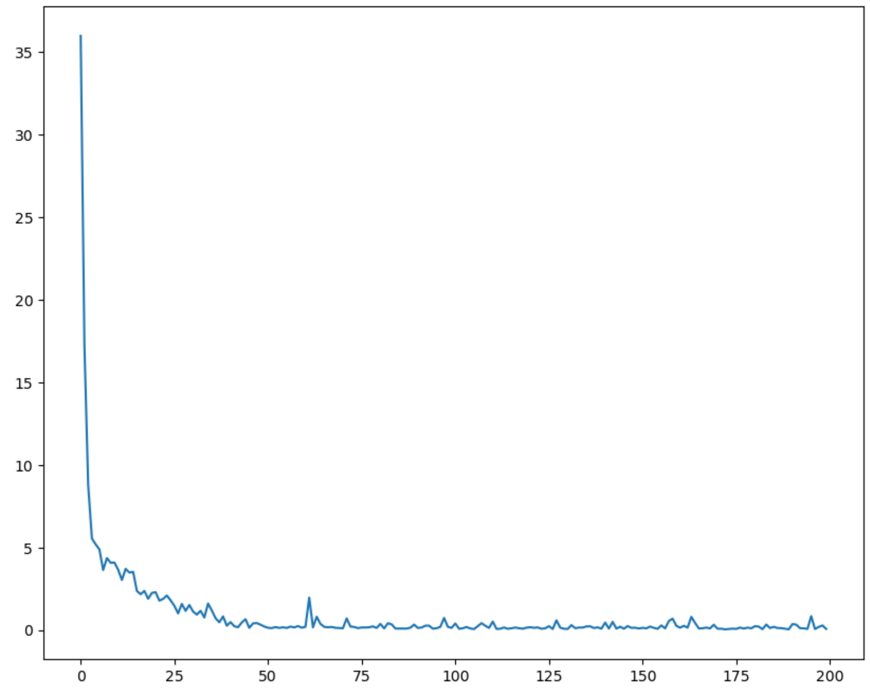


Figure 12: Testing loss per epoch

It can be seen that the losses for both the training and testing both decrease and head toward lower loss values at similar rates. This shows that the model is not being overtrained during training. While there is some choppiness throughout the training process, this can be refined by further adjustments of the hyperparameters of the model.

The individual losses for the training can be seen in Figure 13.

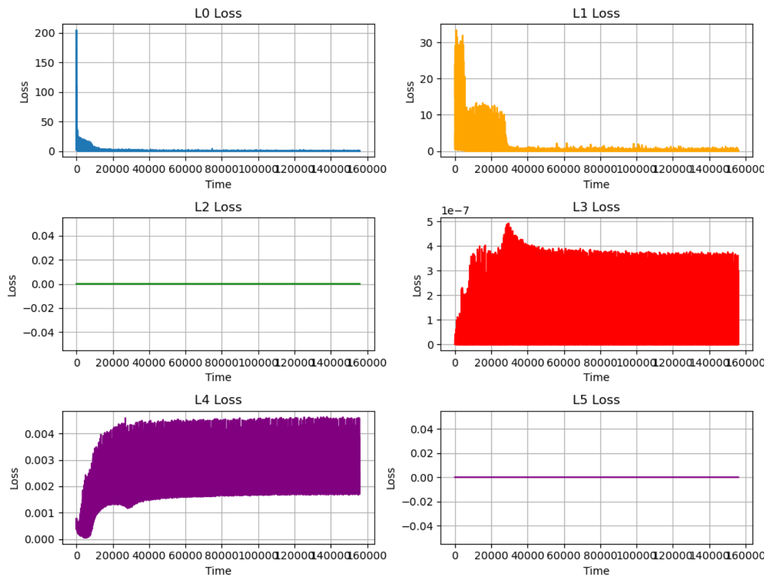


Figure 13: Individual loss during training

It can be seen that the MSE loss from L0 and L1 help drive the loss down toward 0. The contribution from the L3 and L4 loss are minimal due to the weighting of 10^{-7} applied to those loss values. For L2, since the range of values selected during the random walk cycle are from 10000 N for F_1 and 5000 N for F_2 and F_3, these will always be lower than the 30,000 N maximum value that can be generated. Thus, the loss will always be 0. For L5, interestingly, there seems to be no loss accumulated. This is indicating that the there were no instances where the azimuth angles were in the disallowed sectors. This is an interesting observation as the u parameters were all randomly selected. The model might could be potentially selecting angles that are not in disallowed sectors or there is an issue with this loss function.

The individual losses for the testing can be seen in Figure 14.

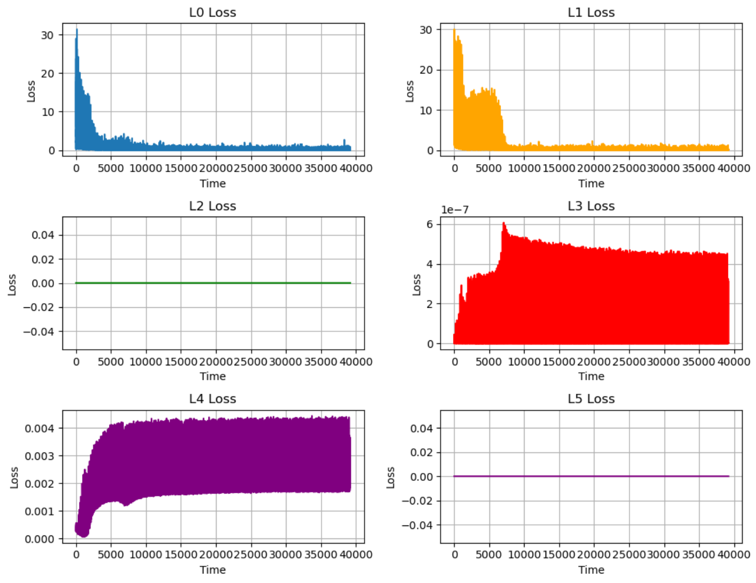


Figure 14: Individual loss during testing

The same trend from Figure 13 is seen here for Figure 14.

As a final validation comparison, the outputs from the encoder and decoder will be compared to the ground truth data.

For the training data, Figure 15 compares the ground truth input, τ, on the left against the output of the decoder, $\hat{\tau}$, on the right for a single batch.

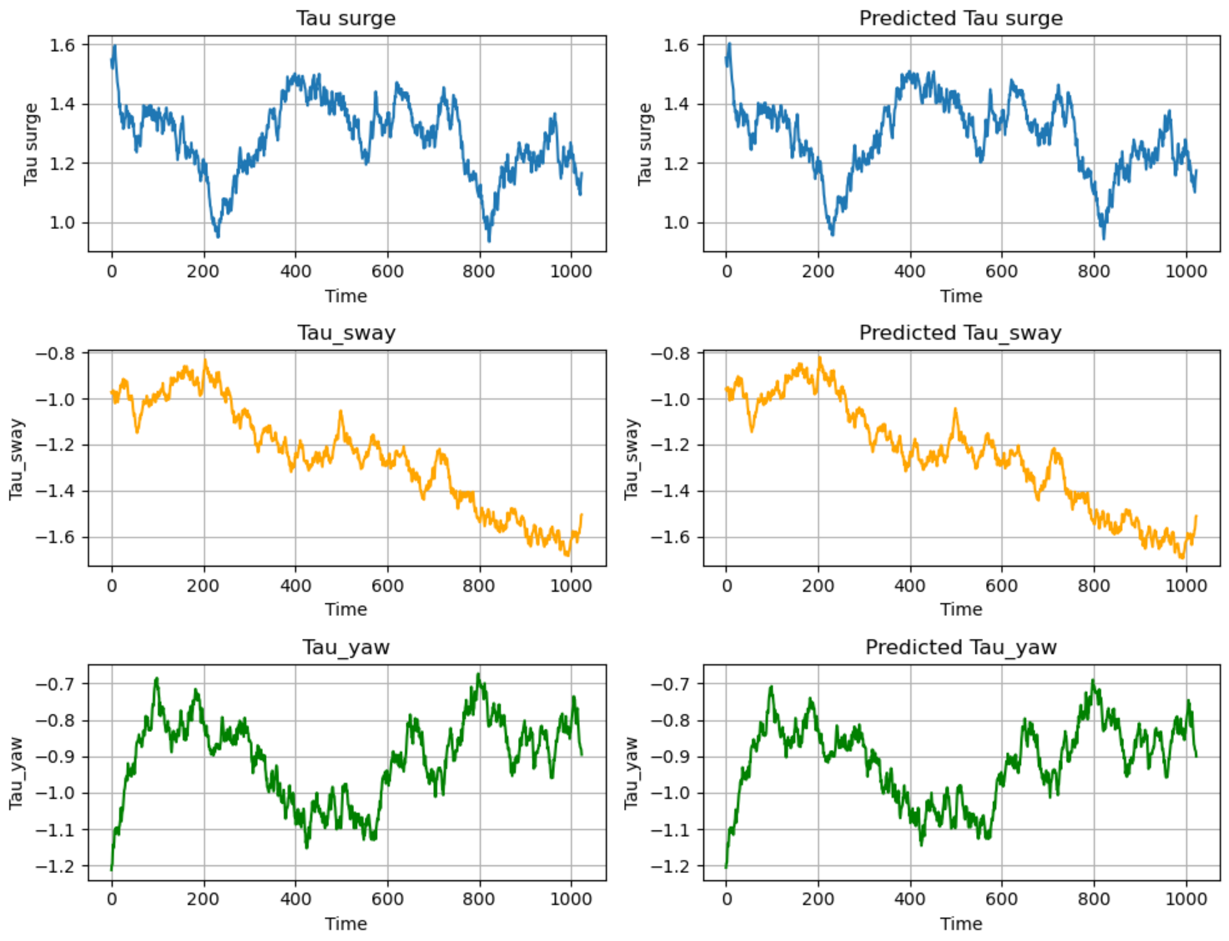


Figure 15: Comparison between the training τ and $\hat{\tau}$

Based on Figure 15, overall, it can be seen that the plots are virtually identical. The MSE loss between the two batches is 7.741e-05.

Additionally, Figure 16 compares the ground truth input, τ, on the left against the $\hat{u}$ data output from the encoder and recalculated to create, τ_{cmd} for a single batch.

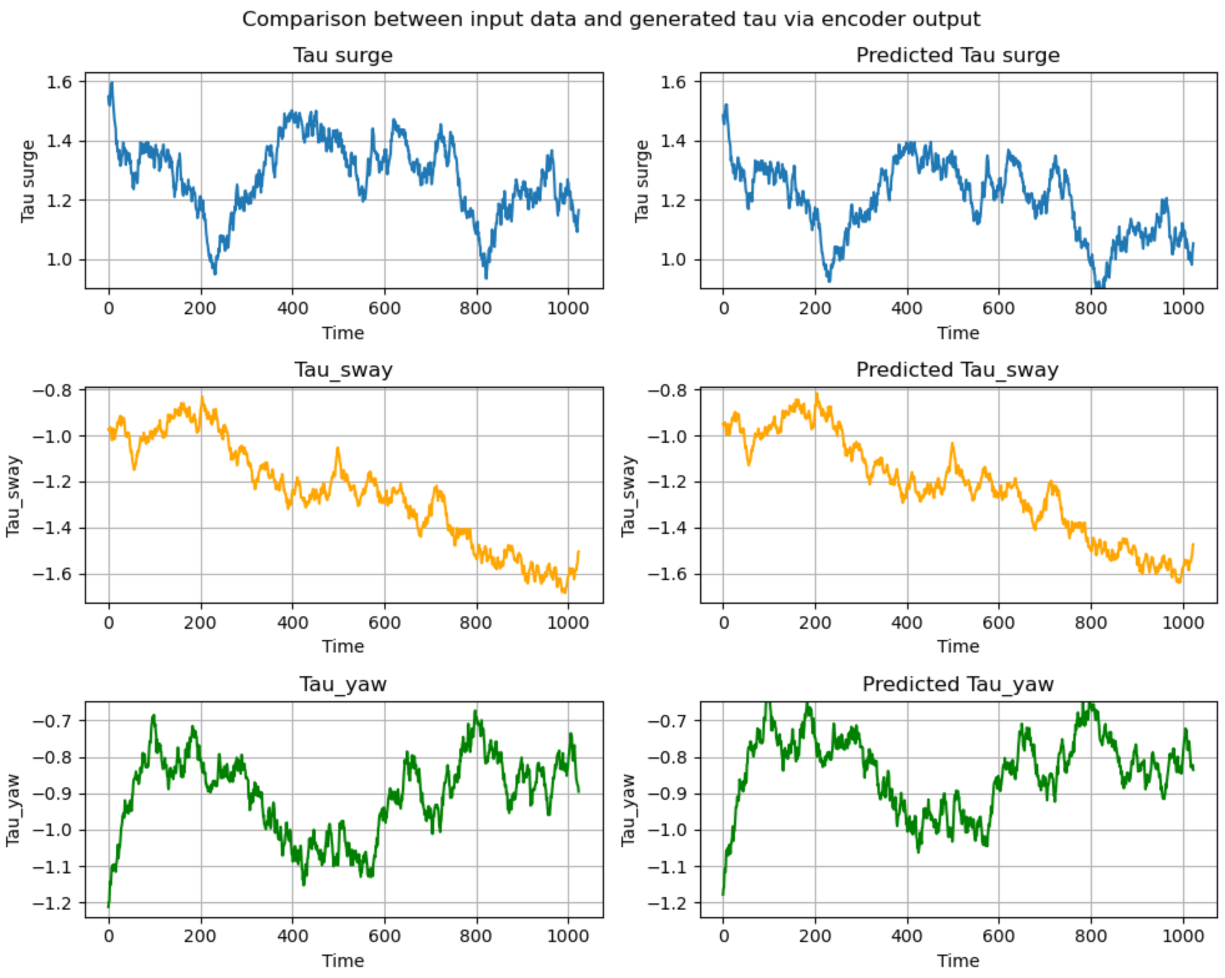


Figure 16: Comparison between the training τ and τ_{cmd}

For this case, there some higher variation between the τ_{yaw} and τ_{surge} values. This is acceptable as the redundant control present on the ship allows different combinations of τ values to achieve the same overall net control. The MSE loss between the two batches is 0.004803.

For the testing data, Figure 17 compares the ground truth input, τ, on the left against the output of the decoder, $\hat{\tau}$, on the right for a single batch.

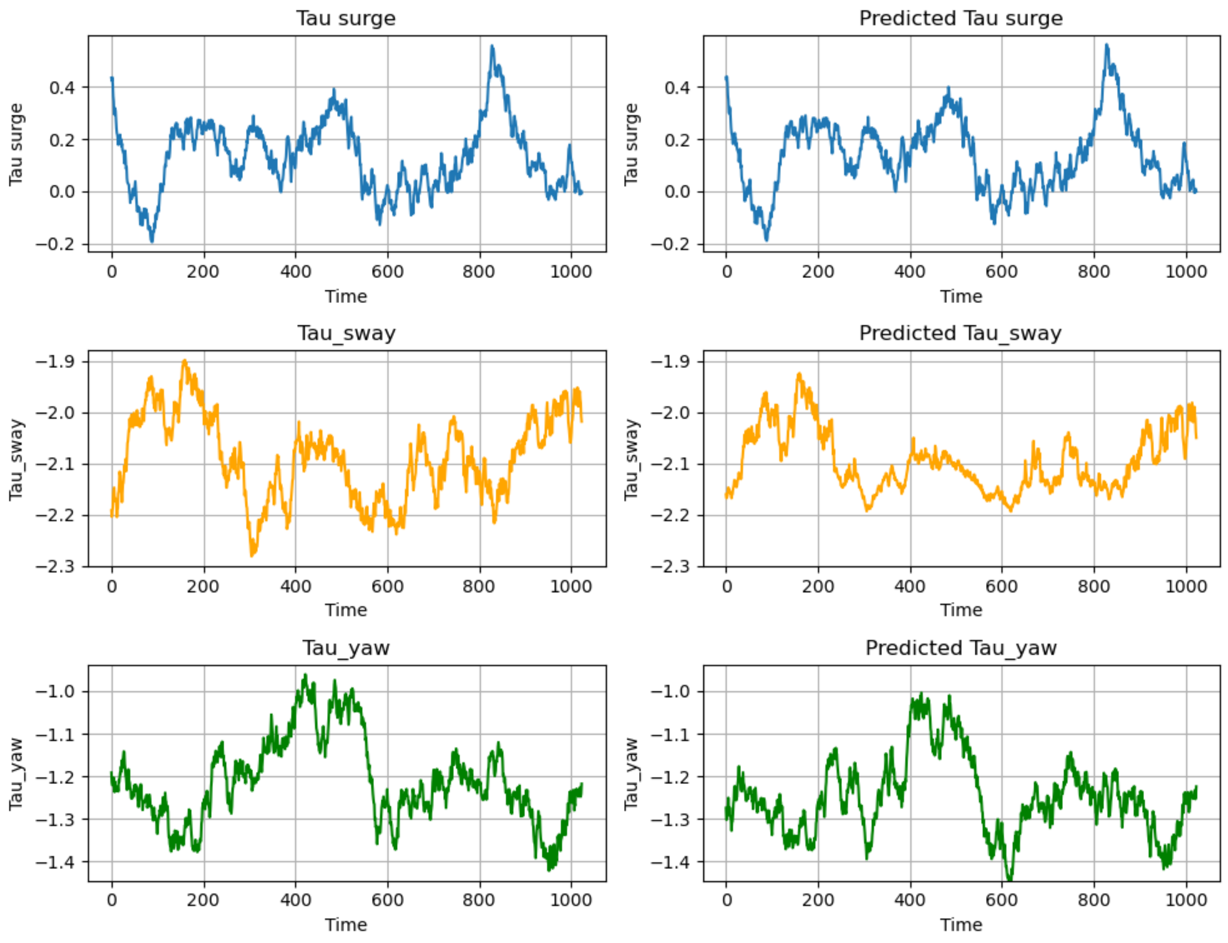


Figure 17: Comparison between the testing τ and $\hat{\tau}$

For the testing data, it can be seen that there is slight variation in the τ_{yaw} and τ_{sway} values. This is most likely due to some error present in the model. The MSE loss for between the two batches is 0.001435.

Additionally, Figure 18 compares the ground truth input, τ, on the left against the $\hat{u}$ data output from the encoder and recalculated to create, τ_{cmd} for a single batch.

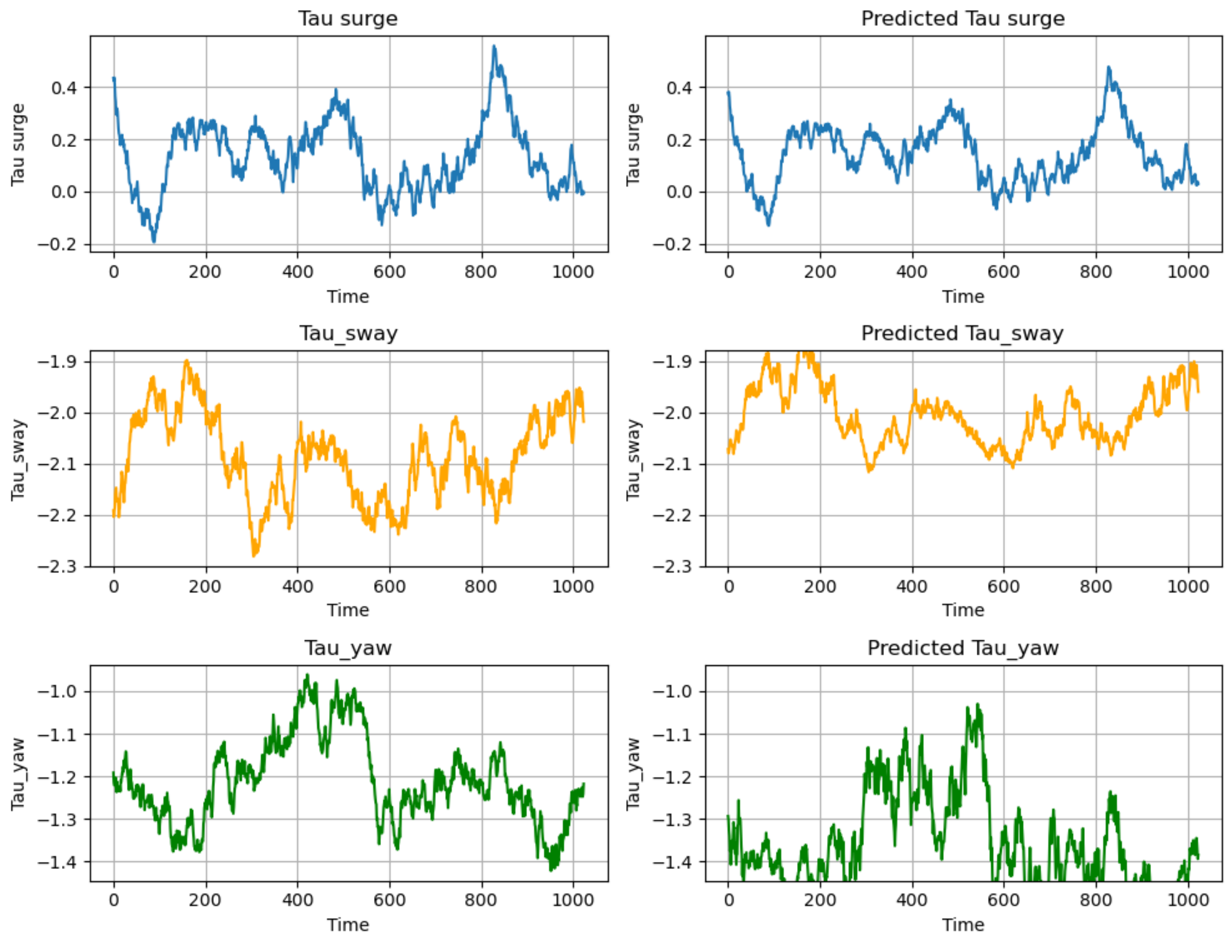


Figure 15: Comparison between the testing τ and τ_{cmd}

For this case, there some higher variation between the τ_{yaw} and τ_{sway} values. This is acceptable as the redundant control present on the ship allows different combinations of τ values to achieve the same overall net control. The MSE loss between the two batches is 0.011257. This variation is most likely what is impacting the output of the decoder, as the MSE loss is slightly higher relative to the training MSE loss between the τ and τ_{cmd}.

Discussion

For this project, the objective was to recreate the autoencoder model developed for control allocation as presented by R. Skulstad et. al. In this report, the methodologies that were presented in the paper were reimplemented and evaluated.

For the data generation portion, using a random walk, an appropriate u vector of one million datapoints was generated and presented. The u input data was then converted to τ values to be used as part of the training. The τ values were then normalized and split into training and testing datasets.

For the custom loss function, the various loss functions were reiterated and implemented within the model.

For the model, the model presented in the paper was reimplemented and modified as per the project requirements (i.e. usage of MLP versus LSTM).

After training and evaluating the dataset, it can be seen that the training and testing error approach zero over 200 epochs and discussion of the individual loss functions was presented. Additionally, comparison between the τ input data and τ_{cmd} and $\hat{\tau}$ was visualized and presented. Overall, the decoder was successfully able to generate its own $\hat{\tau}$, that regenerated the τ values from the input, with its own $\hat{u}$ input vector developed by the encoder. The model did this without overtraining and implementation of regularization schemes presented in the course material.

During the development of the models, various challenges were encountered. The paper was not entirely clear on the overall development process for the model nor about the overall training scheme. Thus, there were gaps in the paper that had to be filled to ensure an accurate result of the work.

For the development of the loss functions, translating the equations and wordings described in the paper to functions was a challenging process to get right. Even currently, within the individual loss functions, there is interesting behavior for the azimuth sector loss need to be further investigated. In addition, the paper describes a loss in L_3 that will always remain 0, posing the question on whether the paper had a typo in it.

For the development of the models, the paper had used LSTM nodes. In our case, we were tasked to use MLPs in our models. Thus, changing the architecture of the model required some tuning to ensure the model could learn the data appropriately. A variety of hidden layer node sizes were evaluated to see which model best learned the data. Having too many or too little nodes impacted how effectively the model was being trained. Various activation functions were utilized as well. Initially a ReLU function was utilized but the performance was not ideal. Due to the presence of negative values with, a LeakyReLU was utilized. The usage of dropout was also tested but the performance of the model was significantly different, as shown below in Figure 16. It can be seen that the outputs were very choppy as compared to the input data.

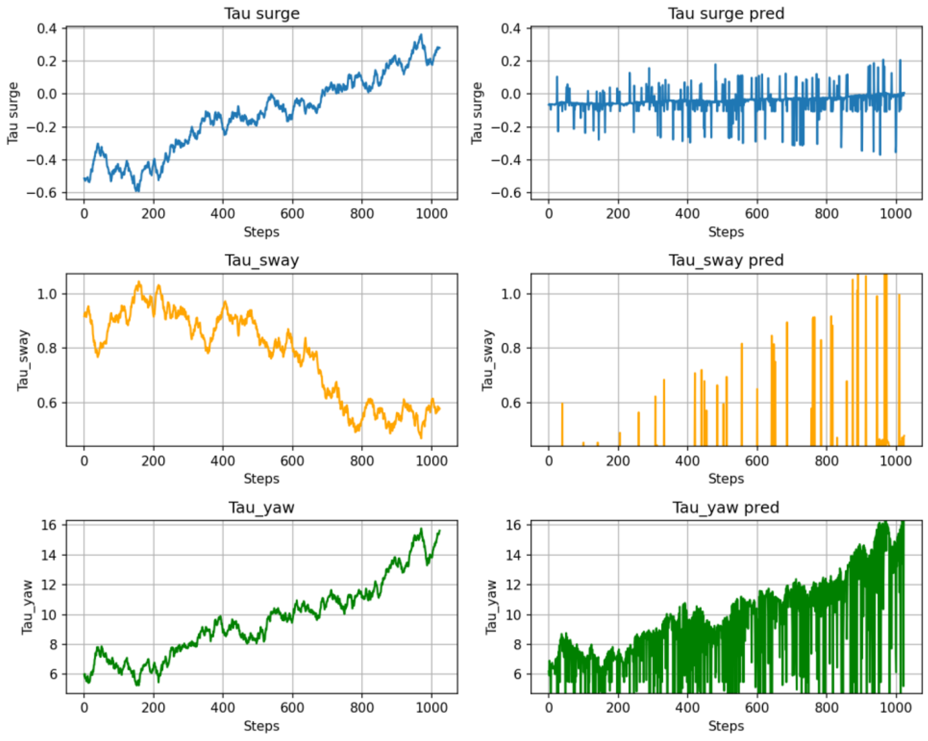


Figure 16: Output of the model with dropout

Ultimately, the final, presented design resulted in the best success. Furthermore, various optimizers, such as SGD and RMSProp were used along with various learning rates and decay rates to improve model performance and reduce overfitting. Ultimately, the Adam optimizer with L2 normalization was best suited for this model.

Overall, the model and work presented in this report and accompanying code and suitable replicate the efforts by in the paper written by R. Skulstad et. al.