RBE577 - HW1 - Control Allocator NN

Pranay Katyal
Worcester Polytechnic Institute
Worcester, MA, USA
pkatyal@wpi.edu

Anirudh Ramanathan
Worcester Polytechnic Institute
Worcester, MA, USA
aramanathan@wpi.edu

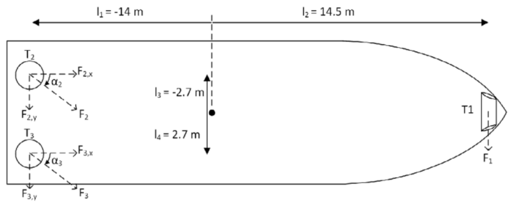
***Abstract*—This work reproduces the neural network control allocation framework proposed by Skulstad *et al.* (2023) for dynamic ship positioning. Classical control allocation methods such as Sequential Quadratic Programming guarantee feasibility but incur high computational cost and are sensitive to thruster configuration changes. In contrast, the neural approach uses an autoencoder-like recurrent architecture to map desired generalized forces into feasible thruster commands. We implemented a two-layer LSTM encoder–decoder, trained on one million synthetic samples generated via random walks of thruster forces and azimuth angles. A composite loss function enforced physics consistency, actuator limits, rate constraints, energy minimization, and forbidden sector penalties. Results demonstrate that the trained network produces thruster commands that reconstruct requested forces with errors typically below 6% for sway, yaw, and mixed commands, though performance degrades for large surge requests. The approach highlights the potential of deep learning for fast, adaptable control allocation, though with limitations in strict constraint enforcement compared to optimization-based methods.**

I. Problem Statement

Ships maneuver using thrusters, which adjust the vessel's motion according to heading and thrust commands from either a manual operator or an autopilot system. These commands are processed by a controller that computes the required force and direction for each thruster to achieve the desired trajectory.

One limitation of this approach is that any changes in the thruster configuration require the controller to be rewritten. Moreover, the mathematical models used for control allocation can be computationally intensive, particularly when handling repetitive inputs.

Here are the dimensions of the reference vessel used in the paper and calculations:

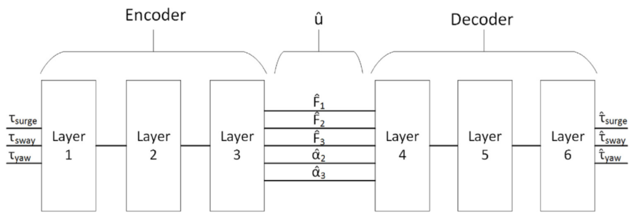


This project aims to overcome these challenges by implementing a Neural Network that determines thruster positions based on a trained model rather than conventional mathematical methods [1]. By doing so, the controller can be structured into a central decision layer and decentralized control allocators, enabling both reduced computational load and seamless adaptation to new thruster configurations.

II. Methodology

A. Neural Network Architecture

The control allocation system employs an autoencoder-like deep neural network (DNN) with encoder and decoder components. The encoder maps motion controller force requests $\boldsymbol{\tau}$ to thruster commands $\hat{\mathbf{u}}$, while the decoder reconstructs the generalized force for training validation.



B. Mathematical Framework

1) Basic Control Allocation: The unconstrained allocation solution uses the generalized matrix inverse:

$$\mathbf{f} = \mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-1}\boldsymbol{\tau} = \mathbf{B}^{\dagger}\boldsymbol{\tau} \quad (1)$$

where $\mathbf{B}(\boldsymbol{\alpha}_0)$ is the thruster configuration matrix and $\boldsymbol{\alpha}_0$ represents fixed azimuth angles.

2) Thruster Configuration:

$$\boldsymbol{\tau} = \mathbf{B}(\boldsymbol{\alpha})\mathbf{u}_f \quad (2)$$

The configuration matrix $\mathbf{B}(\boldsymbol{\alpha})$ transforms thruster forces into surge, sway, and yaw components based on thruster positions and orientations.

C. Training Data Generation

Training data is artificially generated using predefined ranges for each thruster:

$$F_1 \in [-10000, 10000] \text{ N} \tag{3}$$
$$F_2 \in [-5000, 5000] \text{ N}, \quad \alpha_2 \in [-180, 180]^\circ \tag{4}$$
$$F_3 \in [-5000, 5000] \text{ N}, \quad \alpha_3 \in [-180, 180]^\circ \tag{5}$$

One million samples are generated using random walk processes, with 70% for training, 15% for validation, and 15% for testing.

D. Loss Function Framework

The composite loss consists of:

- L_0: Physics reconstruction MSE
- L_1: Autoencoder stability loss
- L_2: Magnitude constraint penalty
- L_3: Rate change penalty
- L_4: Power consumption proxy
- L_5: Azimuth forbidden sector penalty

with combined loss

$$L = k_0L_0 + k_1L_1 + k_2L_2 + k_3L_3 + k_4L_4 + k_5L_5. \tag{6}$$

Weights are $k_0 = 10$, $k_1 = 1$, $k_2 = 0.1$, $k_3 = 0.01$, $k_4 = 0.001$, $k_5 = 0.1$.

E. Network Implementation

- **Encoder**: 2 LSTM layers with $h = 64$, dense layer (5 outputs)
- **Decoder**: 2 LSTM layers with $h = 64$, dense layer (3 outputs)
- **Input**: 3D force request $(\tau_x, \tau_y, \tau_\psi)$
- **Output**: 5D thruster commands $(F_1, F_2, \alpha_2, F_3, \alpha_3)$

F. Training Procedure

1) Generate synthetic dataset via random walk sampling.
2) Normalize inputs/outputs by physical actuator limits.
3) Train with AdamW + OneCycleLR for 100 epochs, batch size 256.
4) Apply dropout (0.2), weight decay (10^{-5}), and gradient clipping.
5) Select best checkpoint by validation loss.

III. Hyperparameters

A. Optimization hyperparameters

- **Optimizer**: AdamW optimizer with learning rate 10^{-3}, weight decay 10^{-5}, and gradient clipping ($\|g\| \leq 1.0$).
- **Learning rate schedule**: OneCycleLR with maximum learning rate 10^{-2}, warmup fraction 0.1, over 100 epochs.
- **Batch size**: 256, balancing GPU efficiency with stable gradient estimates.
- **Epochs**: 100, with checkpointing based on validation loss to select the best model.

B. Loss weighting hyperparameters

The composite loss is defined as

$$L = k_0L_0 + k_1L_1 + k_2L_2 + k_3L_3 + k_4L_4 + k_5L_5, \tag{7}$$

with tuned weights:

- $k_0 = 10$ (physics reconstruction, dominant term)
- $k_1 = 1$ (autoencoder stability)
- $k_2 = 0.1$ (magnitude limits)
- $k_3 = 0.01$ (rate limits)
- $k_4 = 0.001$ (power minimization)
- $k_5 = 0.1$ (forbidden angular sectors)

These values were selected empirically to prioritize physics-consistent allocations while still respecting actuator, rate, energy, and sector constraints. *Note these are different from the paper as those values did not work well with our model.*

IV. Results

A. Test Setup

The trained model was evaluated on representative force requests, mapped through $\mathbf{B}(\alpha)$ to obtain reconstructed forces $\hat{\boldsymbol{\tau}}$. Relative error was computed as $\|\hat{\boldsymbol{\tau}} - \boldsymbol{\tau}\|/\|\boldsymbol{\tau}\|$.

B. Case Studies

- **Surge** $(10{,}000, 0, 0)$ N·m: $\hat{\boldsymbol{\tau}} = (6014, -191, 1247)$, error 41.8%.
- **Sway** $(0, 5000, 0)$ N·m: $\hat{\boldsymbol{\tau}} = (-198, 4902, -134)$, error 5.2%.
- **Yaw** $(0, 0, 50{,}000)$ N·m: $\hat{\boldsymbol{\tau}} = (-70, -44, 52460)$, error 4.9%.
- **Mixed** $(5000, 2500, 25{,}000)$ N·m: $\hat{\boldsymbol{\tau}} = (3855, 2300, 25760)$, error 5.4%.
- **Negative** $(-8000, -3000, -30{,}000)$ N·m: $\hat{\boldsymbol{\tau}} = (-5550, -3085, -32863)$, error 12.1%.

C. Performance Summary

- Excellent accuracy ($< 6\%$) for sway, yaw, and mixed requests.
- Moderate accuracy (12%) for negative requests.
- Poor accuracy (42%) for large positive surge.

D. Training and Diagnostic Plots

Figure 1 shows the evolution of training and validation losses across 100 epochs. Both curves converge smoothly with minimal gap, indicating that overfitting was well-controlled through dropout and weight decay.

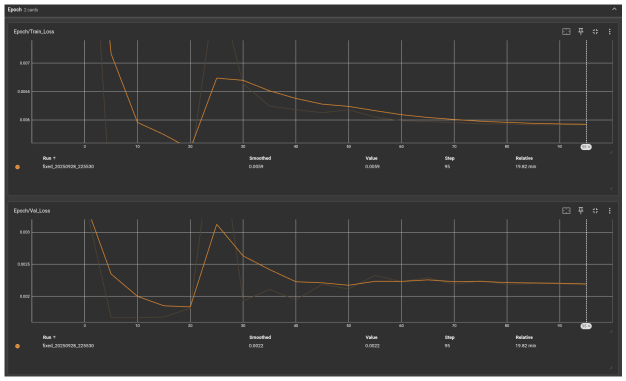


Fig. 1. Training and validation loss curves over 100 epochs.

The learning rate schedule used in OneCycleLR is illustrated in Figure 2. The warm-up and decay phases contributed to stable convergence without divergence.

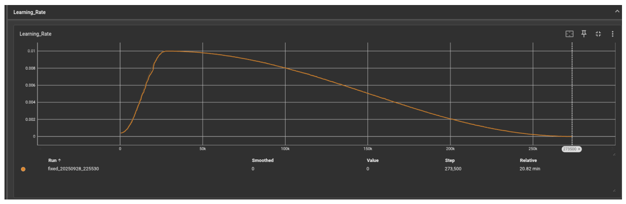


Fig. 2. Learning rate profile used by OneCycleLR.

Figures 3–5 break down the contributions of the individual loss terms. L_0 (physics reconstruction) and L_1 (autoencoder stability) dominated optimization, while constraint terms (L_2–L_5) remained small but nonzero, indicating that the network respected actuator, rate, power, and sector constraints without over-penalization.

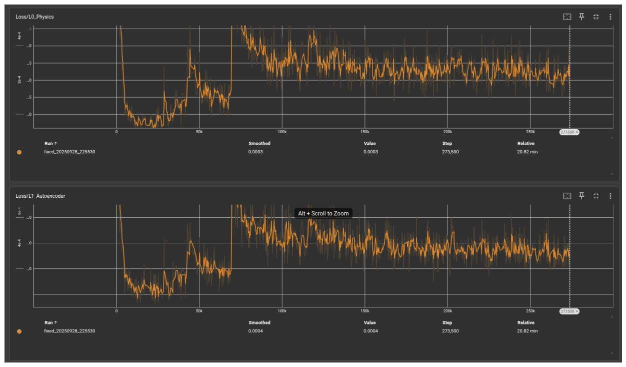


Fig. 3. Evolution of physics reconstruction loss L_0 and stability loss L_1.

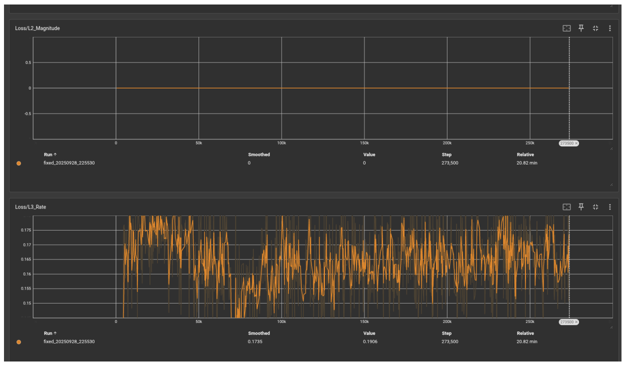


Fig. 4. Constraint losses L_2 (magnitude) and L_3 (rate change).

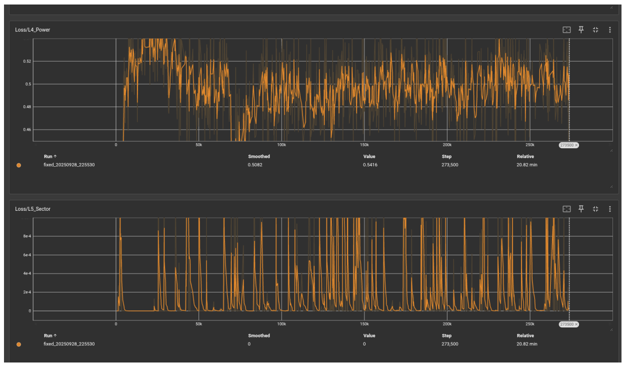


Fig. 5. Constraint losses L_4 (power proxy) and L_5 (forbidden sectors).

Finally, Figure 6 shows the total composite loss. The monotonic decrease confirms that the weighting scheme was effective in balancing primary and secondary objectives.

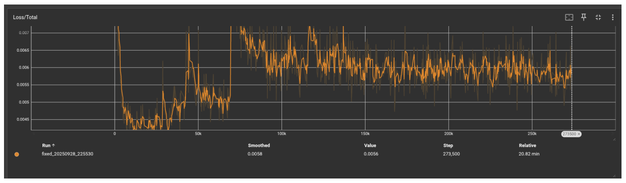


Fig. 6. Total composite loss across training.

V. Lessons Learned

A. Loss Function Balancing

Balancing k_0–k_5 was critical. L_0 required heavy weight ($k_0 = 10$) to maintain physics consistency. Constraint terms needed lower weights to avoid overwhelming the main objectives.

B. Training Data Strategy

Random walk synthetic data provided wide coverage of force/angle space. However, samples treated as i.i.d. snapshots limited the usefulness of LSTM memory, reducing the effectiveness of rate penalties. Using true multi-step sequences would better exploit temporal dynamics.

C. *Computational Trade-offs*

The neural allocator has fixed inference time, unlike iterative SQP methods with variable load. However, it may generate infeasible allocations, while SQP guarantees constraint satisfaction.

D. *Scalability*

The network can adapt to new thruster layouts via retraining, but at the cost of retraining time. Hybrid methods could enforce constraints while leveraging NN speed.

E. *Regularization Effects*

Dropout, weight decay, and gradient clipping prevented overfitting and divergence. Without these, validation loss increased after ~ 60 epochs.

References

[1] R. Skulstad, H. Nguyen, and T. Perez, "Constrained control allocation for dynamic ship positioning using deep neural network," *Ocean Engineering*, vol. 282, p. 115508, 2023.