DEVELOPMENT OF DYNAMICAL MODELS FOR GUIDANCE AND CONTROL OF OCEAN VEHICLES

Nikolaos I. Xiros[1], Miguel Trejos[1], and Erdem Aktosun[2]

[1] Bollinger School of Naval Architecture & Marine Engineering, University of New Orleans, New Orleans, LA, 70148, USA
nxiros@uno.edu, matrejos@uno.edu
[2] Department of Shipbuilding and Ocean Engineering, İzmir Kâtip Çelebi University, Havaalanı Şosesi Cd, 35620, Izmir, Türkiye
erdem.aktosun@ikcu.edu.tr

Abstract. An autonomous boat with electric propulsion using direct-current motors as prime movers and screw propellers is investigated. Dynamical modeling capable to adequately describe the seakeeping of the craft in various conditions is employed to fuse fundamental principles and data sets obtained from an experimental campaign. A standard nonlinear state-space model is formulated using neural networks that are trained using supervised training machine learning methods. The dynamical model forms the basis for applying physicomimetic approaches to control and navigation of standalone boats or swarm thereof.

Keywords: Ocean Vehicle Dynamics · System Identification · Neural Networks.

1 Introduction

With the growing success of data-driven approaches in various fields particularly machine learning (ML) and its subset deep learning, people are increasingly investigating their applicability to complex engineering problems. Among these, one important application is the modeling and control of dynamic systems, which are often nonlinear, high-dimensional and difficult to represent analytically. Machine learning models and especially neural networks offer a compelling alternative due to their ability to learn complex input-output relationships directly from data without requiring explicit equations [3, 7].

In particular, deep neural networks have shown promise in system identification tasks where the goal is to estimate the mathematical model of a system based on observed data. This is especially relevant for dynamic systems where accurate modeling of time-dependent behavior is essential. Time-domain modeling approaches that aim to capture the relationship between a system's state, its input and the time derivatives of its state, are vital for understanding and

predicting system dynamics [12]. Neural network serve as universal function approximators [4, 3], making them suitable tools for this purpose, as they can approximate the significant functional relationships governing a system's evolution over time.

This research focuses on applying such methods to model the dynamic of a marine vehicle specifically a surface watercraft. Marine vehicles pose unique challenges for modeling due to their interaction with a complex and often unpredictable marine environment. A typical surface watercraft has six degrees of freedom (6-DOF) surge, sway, heave (translations), and roll, pitch, yaw (rotations), which together define its motion and orientation in three-dimensional space [1]. Accurate modeling of these DOFs is critical for control, navigation, and simulation purposes, yet obtaining high-accuracy models using traditional physics-based approaches can be time-consuming and limited by simplifying assumptions.

Recent studies have also explored how including boat dynamics into the control of autonomous surface swarms impacts performance. In scenarios comparing dynamic versus nondynamic conditions, boat dynamics significantly affect trajectory smoothness, velocity profiles, yaw rates, and convergence behavior. Boats operating with full dynamics tend to converge more smoothly and avoid collisions more effectively despite slightly slower response times due to increased complexity. These results emphasize the importance of including physical dynamics for more realistic, robust, and energy-efficient swarm control in marine environments [11].

In our work, we concentrate on two primary dynamic components: surge (forward acceleration) and yaw (rotational acceleration about the vertical axis), as they are especially relevant for maneuvering and path-following control. To collect the necessary training data, a physical prototype of the surface watercraft was constructed and equipped with onboard sensors capable of measuring motion and input commands. A series of controlled experiments were conducted in a test environment, and the resulting data were used to train feedforward neural networks to model the input-output dynamics of the craft. Once trained, the models were included into a simulation framework designed to predict the trajectory of the watercraft based only on motor inputs. This enables testing and validation of control strategies in a virtual environment before deployment. The details of model training, validation, and the resulting performance in simulation are well analyzed and presented with given results.

2 Boat Construction and Trials

2.1 Vehicle Design

The vessel used was a repurposed New Bright's Sea Ray Sundancer 29" remote control (RC) speed boat which came with oppositely rotating twin screw propellers of unknown origin. Though they most closely match the Blomiky's H102 RC Boat Propellers. For propulsion, we used two of Injora's 550 watt, 29 turn,

brushed, waterproof RC motors which were connected to the shaft via universal couplings. After inspection, it was found that the motors did not operate at the same rpm, which propagated to the model.

To control the motors, two of GoupRC's 2 - 3S, Lithium Polymer (LiPo), Waterproof, Brushed Electronic Speed Controllers (ESC) were implemented which received inputs wirelessly from anr Elegoo UNO R3 board onshore. In order to accomplish this, we used Radiolink's R8EF RX receiver and T8S, 8 Channel, 2.4 GHz transmitter. The sensors implemented on the vessel were a 9-axis inertial measurement unit (IMU) and a compass magnetometer. To collect and send the data we installed a Elegoo's UNO R3 ATmega328P, Arduino-compatible board. Power for the entire vessel came from a Exceed RC's 8.4-volt, 3000 milliamp hour, NiMh battery. Figure 1 shows the electronics diagram of the vessel. For a more in depth description of the vessel design view [2].

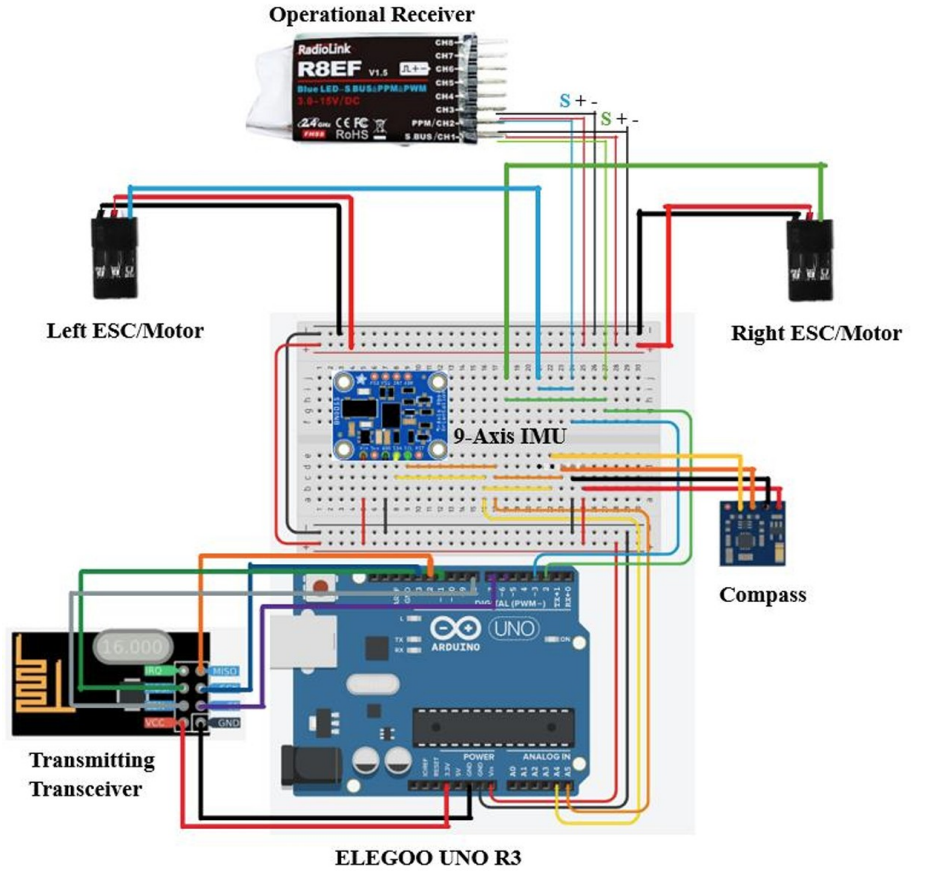

Fig. 1. Diagram of surface watercraft electronics

2.2 Dynamic Tests and Data Collection

A set of experimental tests were performed for calibration and to gather system data for the purposes of training neural networks to approximate the state equations of the system. The data batch was collected on Monday March 15th, 2024, at 29.9997° N and 90.0857° W. Three different dynamic runs were performed to collect data. First, a run where the vessel made counterclockwise circles of about 3-4 meters in diameter for 5-8 complete laps. Second, a run where the vessel made clockwise circles following the same instructions as the previous run. Third, a run where the vessel drove back and forth in a straight line of 10 meters at full thrust 5-8 times.

As stated previously, both an IMU and a compass where used to collect data. The IMU module implemented in the design was a Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout – BNO055 that consists of a Magnetometer, Accelerometer, Gyroscope, and MCU (Mirco-Controller Unit). This module measures linear accelerations and angular velocities using 3-axis capacitors, and measures magnetic strength vector using a magneto-resistive magnetometer. To interface with the IMU using the micro-controller, an open source Adafruit library, Adafruit BNO055, was used. The compass module implemented was a GY-271 QMC5883L compass magnetometer from WWZMDiB. This module was used as an extra sensor to check the data from the IMU. The sensor modules are not perfect in their operation so small waves, mostly caused by the vessel turning back on its' own wake, wind, and short malfunctions translate a fair amount of noise. The computer filters this noise as far as it can without losing credibility, but it still surfaces.

To transmit the data from the boat to the shore, two Deegoo-FPV's nRF24L01 2.4 GHz, wireless RF Transceiver modules are implemented. The modules are initialized for maximum range and for a data rate of 2 Mbps. Every loop, eight values are sent in a single string. Those values being: forward acceleration, yaw velocity, pitch, roll, yaw, compass readings, and inputs to both motors. Unfortunately, the transceivers communicate in irregular time steps and these small variations were not noticed until after network training and simulation modeling.

3 System Identification

3.1 Data Pre-processing

To ensure adequate performance, the raw data was pre-processed before being used. Firstly, the accelerometer vector was normalized to the acceleration due to gravity. Using trigonometry, the values from the accelerometer were used to calculate the pitch (θ) and roll (ϕ) of the model. The roll and pitch from the accelerometer alone went through a 90% low-pass filter to stabilize the data. These calculated values and the values from the gyroscope were sent through 98% filters preferring the readings from the gyroscope and previous accelerometer reading to the new calculated one. This allowed quick measurements that were corrected by the accelerometer in the long run.

$$\theta_a^\circ = \tan^{-1}(\frac{\ddot{x_a}}{\ddot{z_a}}) * \frac{180^\circ}{\pi} \tag{1}$$

$$\phi_a^\circ = \tan^{-1}(\frac{\ddot{y_a}}{\ddot{z_a}}) * \frac{180^\circ}{\pi} \tag{2}$$

$$\theta^\circ = 0.98(\theta - \dot{y_g}dt) + 0.02(\theta_a) \tag{3}$$

$$\phi^\circ = 0.98(\phi - \dot{x_g}dt) + 0.02(\phi_a) \tag{4}$$

Where variables with script 'a' come from the accelerometer vector, script 'g' come from the gyroscope vector, 'm' from the magnetometer, and 'dt' is the change in time. Calculating the cardinal direction the vessel is facing regardless of pitch and roll, using Euler angle transformations, takes a different form:

$$x = x_m \cos(\theta) + y_m \sin(\phi) \sin(\theta) - z_m \cos(\phi) \cos(\theta) \tag{5}$$

$$y = y_m \cos(\phi) - z_m \sin(\phi) \tag{6}$$

$$\psi^\circ = \tan^{-1}(\frac{y}{x}) * \frac{180^\circ}{\pi} \tag{7}$$

This completely rids the yaw angle (ψ°) of any vertical noise and ensures an accurate reading no matter the vessel orientation. With the intention of using the extra compass magnetometer sensor to check the data from the IMU, we use a filter that feeds back 90% of the IMU yaw with 10% of the compass azimuth. This value is then sent through a 70% low-pass filter preferring the older value for stabilization. Finally, the data is rescaled to a new range [0,1] before being input to the neural networks.

3.2 Nonlinear State-Space Model

To approximate the non-linear state equations for forward acceleration and yaw acceleration we employed the use of neural networks which are a supervised learning model. Neural networks were originally created as an imitation of the neuroscience model of human brain neurons, and are a form of biologically inspired model. The model architecture consists of layers of "neurons" that are fully connected to the layer in front. These layers are split into three distinct categories: input layer, hidden layer(s), and output layer. Each neuron takes in all the outputs of the previous layer and performs a linear transformation of the form:

$$y = Wx + b \tag{8}$$

Where 'y' is the output, 'x' is the input to the neuron, 'W' is the weight associated with the connection between the output neuron from the previous

layer and the input neuron, and 'b' is the bias term. The outputs can be seen as derived features. To be able to approximate non-linear functions, a non-linear activation function is applied to these derived features. Typical activation functions that are used include sigmoid, hyperbolic tangent, and rectified linear unit (ReLU).The modern recommendation and the most widely used activation function is ReLU [3], [8]. Among the top reasons are ReLU typically performs better than other activation functions, solves the vanishing gradient problem that other activation functions face, and is computationally inexpensive [5], [8], [10].

As with other parametric, supervised learning models, neural networks approximate a function by optimizing their parameters to reduce an error/cost function. For regression, the typical error function used and the one used in this paper is mean squared error (MSE):

$$MSE = \frac{1}{n} \sum (y - \hat{y})^2 \tag{9}$$

Where 'y' is the given target value, '$\hat{y}$' is the predicted value from the model, and 'n' is the number of outputs. To optimize the parameters, a gradient descent algorithm is typically used where the gradient of the loss function is calculated with respect to each weight and bias. The weights and biases are then updated as followed:

$$W_{(r+1)} = W_r - \alpha \Delta_W MSE \tag{10}$$

where W_r is the current weight value, $W_{(r+1)}$ is the updated weight value, $\Delta_W MSE$ is the gradient of the loss function with respect to the weight, and α is a user defined hyperparameter called the learning rate. This method requires manual tuning of the learning rate, however, there are optimization algorithms that can adapt their learning rates during training. For our network, we decided on Adam which was introduced in [6]. Although there is no clear best algorithm, adaptive methods generally perform better and are computationally efficient [9].

Neural networks, just like all other machine learning techniques, need to be able to perform well on data that it hasn't seen or been trained on. The two main ways that the model can fail is by underfitting or overfitting. Underfitting happens when the model is unable to approximate the function resulting in high training error and test error. Overfitting occurs when the model follows too close to the training data and is not able to generalize; resulting in very low training error and higher test error [3]. Typically, neural networks face overfitting problems, so regularization/ weight decay is used to correct it. This method for regularization, also referred to as L^2 regularization, adds a penalty term to the error function which forces the weights to smaller values:

$$L = MSE - \lambda \sum W \tag{11}$$

where L is the total cost function and λ is a tunable parameter.

For both of our neural networks, we used one input layer, one hidden layer, and one output layer. The model was designed with 4 input neurons, 9 hidden neurons, and 1 output neuron as shown in Figure 2. We chose ReLU for the

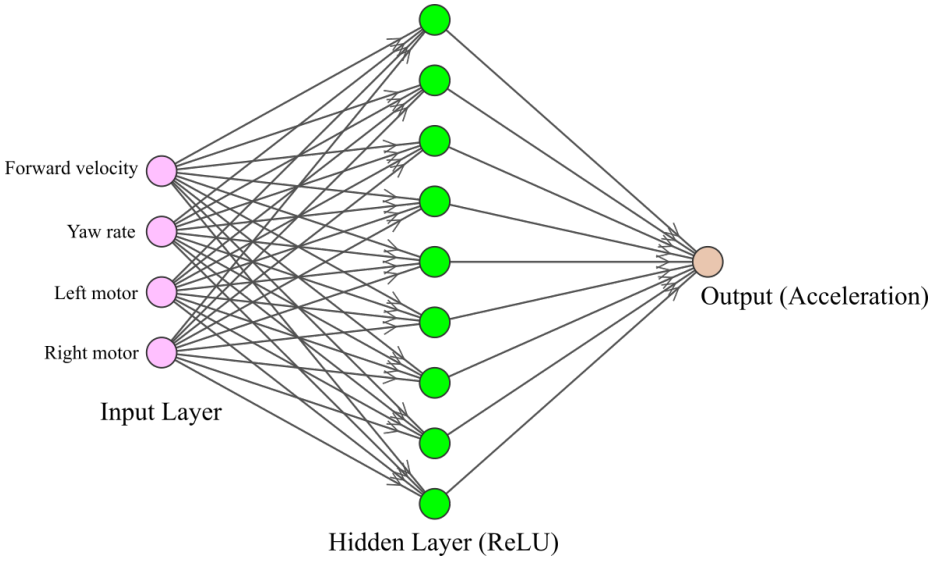

Fig. 2. Neural network architecture

hidden neuron activation functions and a linear function for the output. The 4 inputs to the network were forward velocity, yaw rate, left motor, and right motor. The output for one network is forward acceleration and the other is yaw acceleration. Both networks are built and trained in MATLAB.

Once trained,the neural networks were implemented into the simulation shown in Fig. 3. The system used to create the simulation was Mathworks' Simulink. Since the networks approximate the state equations, they were fed the inputs which are the motor values, and their current state variables which were the forward velocity and yaw rate. Initial conditions for both yaw rate and forward velocity were assumed to be zero. All inputs to the networks are normalized the same way as in training. The networks output forward acceleration and yaw acceleration. These values are then integrated to obtain forward velocity and yaw rate. Yaw rate is integrated once more and then sent to a Matlab function block along with the forward velocity. This block calculates the x and y components of the velocity and the heading in radians. Both velocity components are integrated to obtain the ship coordinates and sent to output along with the heading. The simulation is run and the ship path is plotted from the output.

4 Results

The neural networks were trained for 10000 epochs, with an initial learning rate 0.001, weight decay of 0.0001, and the weights were initialized using the default Glorot/Xavier initialization. Network training was done on a GeForce RTX 4080 Ti. After being fully trained, both networks were sent the full test

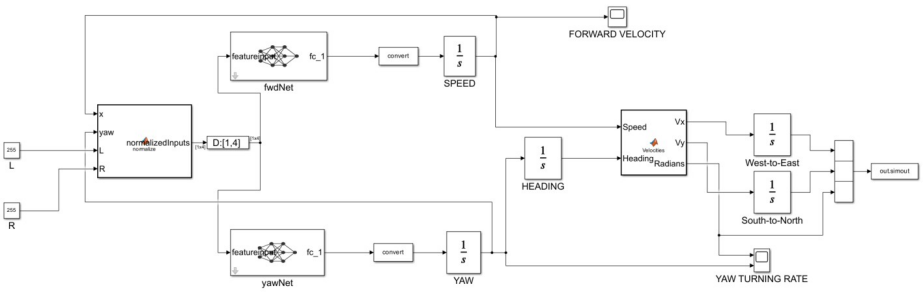

Fig. 3. Block diagram flow of simulation model

data to compare their outputs to the ground truth seen in Figure 4(a) and Figure 4(b).

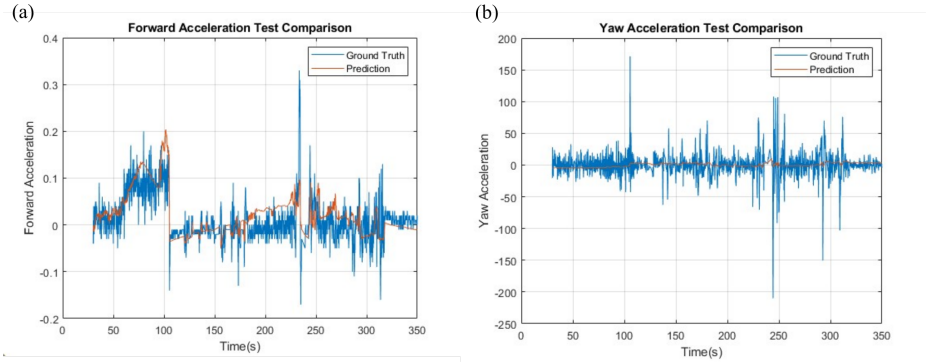

Fig. 4. (a) Forward acceleration comparison on test data, (b) Yaw acceleration comparison on test data

Figure 4(a) presents the predicted surge (forward) acceleration generated by the neural network model in comparison with the ground truth obtained from experimental measurements. The model effectively captures the fundamental dynamics over time that is showing strong time-related coherence despite the sensor noise. This alignment indicates successful learning and generalization of the nonlinear surge response from the input-output data. Similarly, Figure 4(b) shows the predicted yaw acceleration compare with the corresponding measured values. The close agreement across a range of input profiles confirms the neural network's capability to approximate complex, nonlinear yaw dynamics with high resolution that is validating its potential for real-time guidance and control applications. Note that the neural network signals are much smoother than the experimental measurement time series. This means that the neural networks act also as frequency-selective filter. Also, note that the yaw acceleration signal is

more noisy that the forward acceleration one; this observation has been made in other similar studies, too.

Now, one may see a series of trial runs, including linear forward trajectories, clockwise circular maneuvers, and counter-clockwise circular maneuvers with additional dynamic tests to assess model generalizability under varied operational conditions.

4.1 Forward run

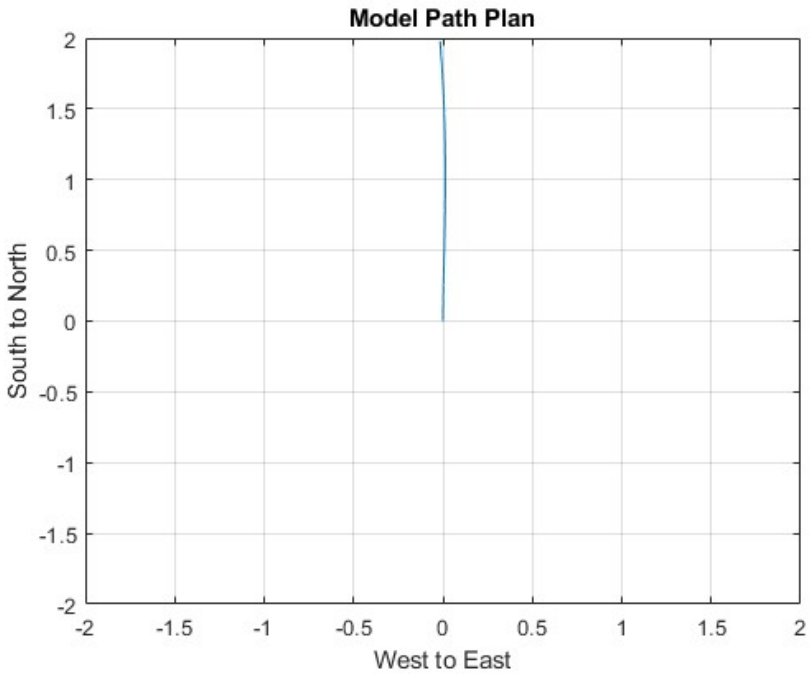

Fig. 5. Forward run at 22.5 and 255

Figure 5 shows the trajectory of the vessel during a forward thrust test. Motor inputs were at 22.5 and 255, producing a slightly curved trajectory. The plot shows the vessel paths for validation. The plot looks like a map and the trajectory like one that would appear on an ECDIS (Electronic Chart Display System) or radar.

4.2 Port circle

Figure 6(a) shows the simulation and test results for a counterclockwise circle maneuver where the right (starboard) motor was at full thrust (255) and the

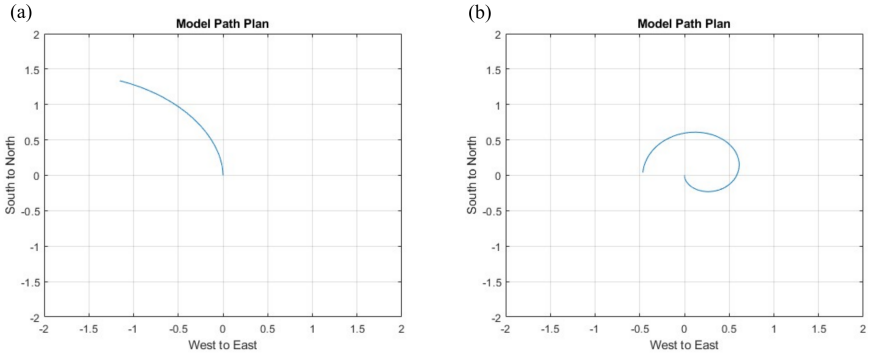

Fig. 6. (a) Port circle run at 0 and 255, (b) Port circle run at -255 and 0

left (port) motor was inactive (0). The vessel's circular path confirms differential thrust-induced yaw. Another counterclockwise circle test is shown in Figure 6(b), with full reverse thrust on the port motor (-255) and zero thrust on the starboard one. This plot highlights the nonlinear behavior of the vessel under reverse input and validates the model's response accuracy. Note a standard test like that is typically performed during sea trials of full-scale vessels. The important parameters to note are the time to complete the maneuver and the radius of the circle.

4.3 Starboard circle

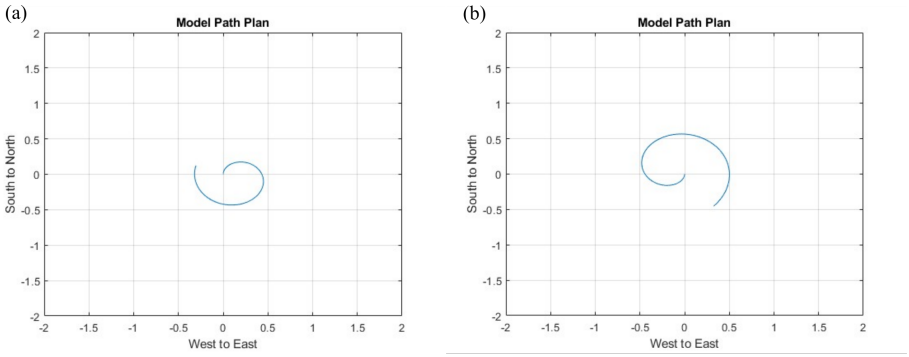

Fig. 7. (a) Starboard circle run at 255 and 0, (b) Starboard circle run at 0 and -255

Figure 7(a) shows a clockwise turning maneuver with full thrust applied to the left (port) motor and none to the right (starboard) one. The predicted trajectory aligns well with the experimental path, showing the model's capability

to simulate yaw-induced motion. In this test, clockwise turning was induced by full reverse thrust on the starboard motor, with no input on the port motor as shown in Figure 7(b). The comparison of simulated and actual trajectories supports the model's robustness under reverse input scenarios.

4.4 Additional Tests

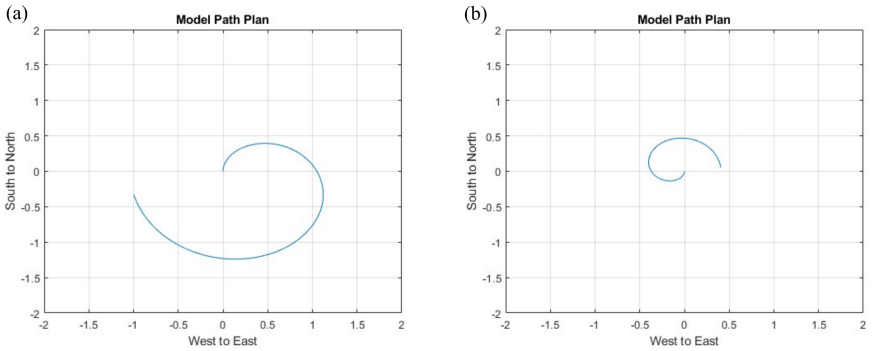

Fig. 8. (a) Both motors full (255/255), (b) Both motors neutral (0/0)

In Figure 8(a), the vessel was operated with both motors at full forward thrust. The resulting trajectory is implemented according to expectations, and the model's predictions closely match the test data. This result strengthens model fidelity particularly for straight-line forward navigation. Also, Figure 8(b) shows the vessel behavior when both motors were idle. The plot shows minimal movement, validating the model's handling of stationary conditions and confirming minimal drift in the absence of exogenous forces.

The results show that a data-driven nonlinear model, particularly one based on neural networks, can accurately replicate complex physical behaviors of a watercraft such as surge and yaw dynamics. The fusion and coupling of IMU, compass, and filtered data pre-processing adds robustness to the training data where it allows the model to generalize across different maneuvers. Some limitations persist such as small lags in yaw estimation during sharp turns which is possibly due to noisy sensor input or under-representation of aggressive maneuvers in the training data. However, the framework is a promising step toward real-time control and navigation of autonomous surface watercraft.

5 Conclusions

This study presented a novel approach to developing dynamical models for surface ocean vehicles using machine learning, specifically neural networks trained on filtered sensor data. The boat at hand was modeled with an empirical,

reduced-order state-space model to depict planar kinetics of the boat by means of her forward and yaw turning dynamics in a body-fixed frame of reference. Particularly, the system identification method accurately modeled surge and yaw behaviors, validated through real-world experiments and simulations. The proposed model effectively maps motor inputs to vessel states, laying the groundwork for further applications in guidance, control, and cooperative swarming. Future work will focus on expanding the model to additional degrees of freedom and integrating real-time feedback for autonomous path planning.

Acknowledgments. The authors wish to acknowledge the continuing support provided in the H2theFuture framework funded by Greater New Orleans Development Foundation (08-79-05681 – 03)/U.S. Economic Development Administration (08-79-05681). We would also like to acknowledge Jonathan Freels for constructing the watercraft and running the tests to gather the data used within the paper.

Disclosure of Interests. The authors have no competing interests.

References

1. Fossen, T.: Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons (2011)
2. Freels, J.P., Xiros, N., Trejos, M.A.: Simulation model development and calibration of surface watercraft dynamics using neural networks and machine learning. In: ASME International Mechanical Engineering Congress and Exposition. vol. Volume 5: Dynamics, Vibration, and Control (2024). https://doi.org/10.1115/IMECE2024-147065
3. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
4. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural networks **2**(5), 359–366 (1989)
5. Jagtap, A.D., Karniadakis, G.E.: How important are activation functions in regression and classification? a survey, performance comparison, and future directions. Journal of Machine Learning for Modeling and Computing **4**(1), 21–75 (2023)
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014), https://api.semanticscholar.org/CorpusID:6628106
7. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
8. Sharma, S., Sharma, S., Athaiya, A.: Activation functions in neural networks. International Journal of Engineering Applied Sciences and Technology **4**(12), 310–316 (2020). https://doi.org/10.33564/IJEAST.2020.v04i12.055
9. Soydaner, D.: A comparison of optimization algorithms for deep learning. International Journal of Pattern Recognition and Artificial Intelligence **34**(13) (2020). https://doi.org/https://doi.org/10.1142/S0218001420520138
10. Szandała, T.: Review and comparison of commonly used activation functions for deep neural networks. In: Bhoi, A.K., Mallick, P.K., Liu, C.M., Balas, V.E. (eds.) Bio-inspired Neurocomputing, pp. 203–224. Springer Singapore (2020)
11. Xiros, N.I., Aktosun, E., Loghis, E.C.: Distributed control of autonomous watercraft dynamics using physicomimetics and robust synthesis for disturbance rejection. Franklin Open **7**, 100099 (2024). https://doi.org/https://doi.org/10.1016/j.fraope.2024.100099

12. Zaknich, A.: Principles of adaptive filters and self-learning systems. Springer Science & Business Media (2005)