PROJECT C048: NEXT GENERATION E-DRIVETRAIN FOR AUTOMATED MOBILE ROBOT PART I : CONDITION MONITORING

PARV PATODIA

U2023093L

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

A final year project report presented to

Nanyang Technological University

in partial fulfilment of

the requirements for the

Degree of Bachelor of Engineering (Mechanical Engineering)

Nanyang Technological University

1. INTRODUCTION

1.1 Background

Since the beginning of human progress towards network and technology, Robotics has been right at the forefront of some of the most astonishing and life-changing inventions. The sole purpose behind research and development into robotics was to help humans in doing tasks that either required monotony or precision continuously to create an effective process. These machines have made, rising labor costs and a shrinking number of skilled workers over the past decade, irrelevant by providing the growth and opportunity for all industries that could implement this technology to scale even bigger.

Automated Mobile Robots (AMR) are a type of robot that have been specifically designed for working in the warehouses and factories for industrial purposes. These robots can navigate in a workplace without the need of any extra infrastructure and can perform multiple functions such as navigation, autonomy, detection, collaboration, and connectivity accustomed to specific situations and user requirements. Their major job is to handle non-value-added operations such as transportation, pickup, and product delivery. This enables human workers to focus on tasks that provide value to the overall product or business, such as order picking, quality control, or packaging.

This AMR adoption by industries has been observed in every region of the world with every continent experiencing robust increase in market value of this technology. For any of these techniques and results to work, the robot must have a foolproof workflow of how it receives and sends data for the command center to react accordingly and initiate the required action for user. For instance, a collaborative robot gripper that weighs and detects small unknown objects relies on a force feedback function to run all algorithms.

Figure 1: Types of AMR in factory

Hence, it is vital to have compatibility between the communication ports (channels that are used for data transfer) and communication protocol (medium which allows communication between devices) as well as the graphical user interface (application that is receiving this data). However, all these components have different basic architecture that needs to be analyzed to achieve a common ground for all the data to get processed and monitored in a useful manner. Hence, this research study focuses on the concepts of condition monitoring for an AMR by examining all components involved in the data transfer between a computer system and the robot.

1.2 Project Overview

The motivation behind this project was to find a method of conducting condition monitoring for the AMR to address the issue of data transfer so that a detailed analysis of several variables could be performed.

The primary purpose to incorporate MATLAB graphical user interface was to provide a seamless transition between the raw data extracted from AMR and a suitable platform to provide visual data representation for evaluation. The challenge, though, is to figure out a way to plot the received data from the AMR in a way that it keeps updating based on the received data from the robot. The reason for developing a GUI from MATLAB script is to ultimately perform condition monitoring for an entity of interest for analysis in a real-world workplace. Based on the difficulty and significance of all the factors that the AMR could monitor, torque has been selected as the variable because it's one of the most vital components in ARM applications. This leads us to the research question for this project:

How to create a MATLAB GUI capable of dynamic plotting of the data received from AMR to predict and then use that concept for torque estimation of the robot?

This research is supported by the Schaeffler Hub for Advanced Research at

Nanyang Technological University.

```
isHexSend = getappdata(handles.figure1, 'isHexSend');
for j = 1:frames_num
    frame = str((1+(j-1)*24):(23+(j-1)*24));
    if ~isHexSend % if not in hex, then send
        val(j,:) = double(frame);
    else  % if in hex
        n = find(frame == ' ');   % find space
        n =[0 n length(frame)+1]; % index of space
        %% convert strings(hex) between two spaces into strings(dec)
        for i = 1 : length(n)-1
            temp = frame(n(i)+1 : n(i+1)-1);    % get length of each string
            if ~rem(length(temp), 2)          % if string length is 2  rem -- reminder
                b{i} = reshape(temp, 2, [])'; % reshape each string into a cell array
            else
                break;
            end
        end
        val(j,:) = hex2dec(b)';                    % hex to dec -- 2-row vector
```

Figure 19: Data Processing and Conversion Function based on string length

Finally, the transmitting text area's 'UserData' attribute is updated with the processed data (val). Thus, the 'sends_Callback' function makes it easier to analyse and process user-inputted data before it is communicated to the serial port, allowing for greater flexibility in dealing with various data formats.

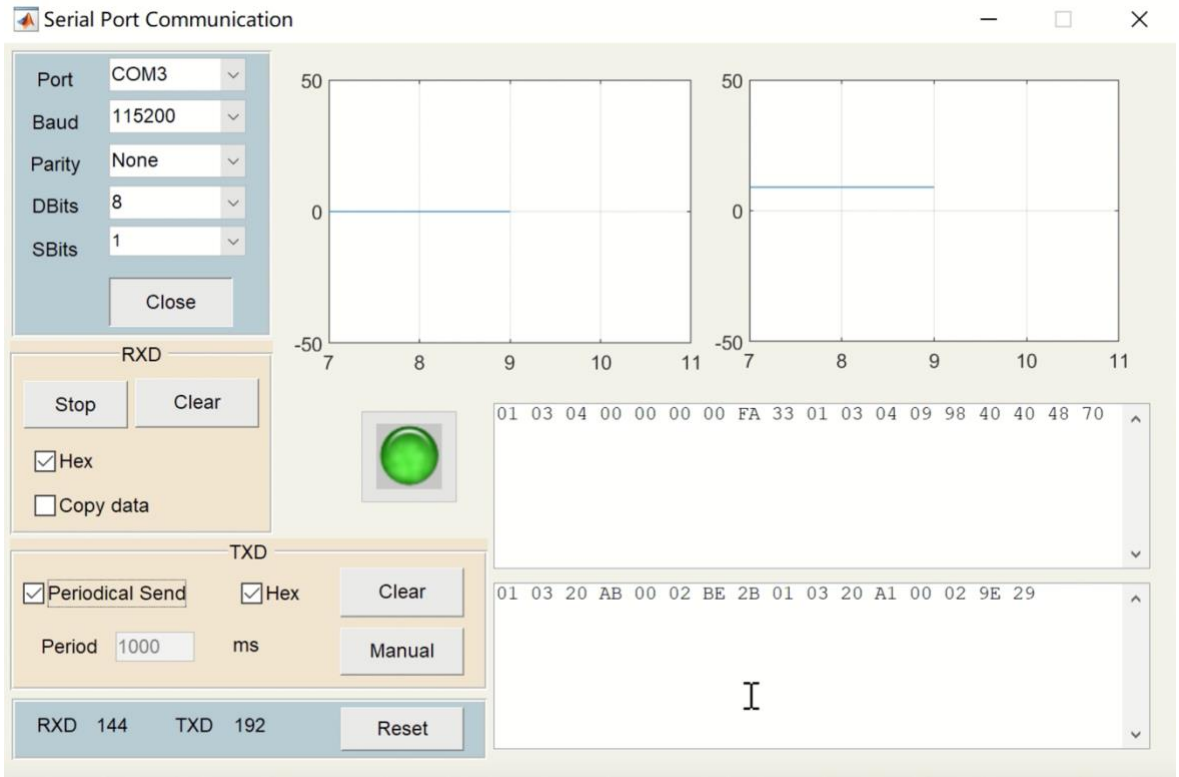

Figure 20: Snapshot of MATLAB GUI showing all buttons, sliders, I/O windows

This is snapshot of the GUI that has been developed by the script analyzed in this section. The buttons visible here are the serial port, baud rate, parity bit, Dbits, Sbits in the top left corner of the screen. The serial port is just the port that is being used to transfer data between the system and the robot. The baud rate controls the data transfer rate. Parity bit is used for error detection in data bits by using either even or odd parity depending on the chosen parity scheme. The 'Dbits' are basically data bits (8 bits) and 'Sbits' are stop bits which are required in serial communication which has a start bit followed by the data bits and the stop bits to indicate the transfer being complete. Periodical send button allows the user to configure the message for periodic transmission over the serial port. In the window above, the period is set to 1000 ms which determines the transmission frequency, and the input message is visible in the white window at the bottom of the screen. Lastly, the stop button halts the data transmission after the last input. As mentioned earlier, the next step in this investigation will be to look at ways to monitor torque for condition monitoring analysis.

4. TORQUE ESTIMATION IMPLEMENTATION

After reading several research papers and articles, it was decided that selecting a physics based neural network would be the best way to estimate torque in this application. A Physics-informed Neural Network (PINN) for torque estimate combines the flexibility of neural networks with fundamental scientific laws that control mechanical systems. PINNs use neural network architecture and add physics equations or constraints to describe the link between torque and system parameters into their structure. During training, these networks learn from labelled data while enforcing physics-based restrictions, ensuring that predictions are consistent with physical principles. By minimizing a composite loss function that considers both data-driven mistakes and physics-based deviations, PINNs successfully balance data fitting and constraint satisfaction. The trained PINN can then reliably predict torque values

Less Reliance on Data: PINNs require less labelled data than solely data-driven techniques since they train using physics-based limitations. This can be useful in situations when gathering huge volumes of labelled data is impractical or costly.

4.1 Selecting Torque Estimation Cases

By addressing two scenarios (current timestamp data, multiple timestamp data) for torque estimate using the PINN technique, we can thoroughly evaluate the model's performance under different intervals of time, resulting in a more robust and dependable torque estimation framework for mechanical systems.

In the case of noised data, noise can come from a variety of sources, including sensor errors, environmental disruptions, or measurement mistakes. By training the PINN on noisy data, we can examine its robustness and capacity to generalize to noisy environments. The inclusion of noise in the training data forces the PINN to learn and adapt to noisy input signals, resulting in improved torque estimates in real applications. This instance allows us to assess how effectively the PINN can manage and mitigate the impacts of noise in input data, offering insights into its performance in real-world scenarios with poor data quality.

The second case of using 3 previous timestamps of torque data was to allow for more stabilized dataset while removing outliers. Additionally, a longer time span increases statistical power, facilitating the detection of relationships, trends, and anomalies with greater reliability.

4.2 Torque Estimation Method

A Permanent Magnet Synchronous Motor (PMSM) is a type of electric motor that is often used in a variety of applications, including electric cars, industrial automation, and renewable

energy systems. Torque estimate in PMSMs is critical for assessing motor performance, optimizing control techniques, and guaranteeing smooth operation. In the context of PMSM torque estimation, a PINN incorporates physics equations that describe the link between motor parameters (such as rotor speed, stator current, and magnetic flux) and torque output into the neural network design. The PINN architecture is made up of several linked layers of artificial neurons, including input, hidden, and output layers. Input characteristics may include motor operating circumstances (e.g., rotor speed, stator current) as well as motor geometrical parameters. The PINN design incorporates physics equations that control PMSM functioning, such as the torque-speed relationship obtained from electromagnetic theory. These physics constraints provide extra assistance during training by verifying that the model's predictions follow fundamental physical laws. The dataset used to train the PINN is derived from MATLAB/Simulink simulations. During training, the PINN adjusts the weights and biases of the neural network layers to learn how to translate input parameters (such as rotor speed and applied voltage) to torque values.

Noise in signals is an important factor to consider when estimating torque using a Physics-informed Neural Network (PINN). Noise is defined as random changes or disruptions that influence the precision of readings acquired from sensors. In real-world circumstances, sensor flaws, ambient variables, and electromagnetic interference can all inject noise into the data. Torque estimate often involves many signals, such as speed, angular acceleration, lateral force, and labelled torque signals. These signals are critical inputs for the PINN model because they give information on the dynamic behavior of the system being studied. However, due to noise, the measurements acquired from these signals may be inaccurate.

To account for the impacts of noise, the PINN approach combines noise modelling techniques into its prediction model. This includes determining the statistical features of the noise in the signals, such as its magnitude, frequency distribution, and correlation structure. By explicitly

modelling noise in the training data, the PINN may learn to discriminate between signal and noise components, increasing its capacity to generate correct predictions in the face of noisy input data. The physics informed loss equation adjusted to the noised signal variables is shown below:

$$T_m(k) \ - [I_w \dot{\omega}_N(k) + F_{xN}(k)r + T_b(k)]$$

Furthermore, the PINN framework uses its physics-informed structure to improve the resilience of torque prediction in noisy environments. By adding physical principles and limitations into the neural network design, the model is able to capture underlying patterns and correlations in data even when there is noise. This physics-informed method not only increases torque estimate accuracy, but it also improves the model's generalization capabilities across a wide range of operating situations and configurations. As a consequence, the PINN can reliably predict torque values for the PMSM while requiring little training time and processing resources.

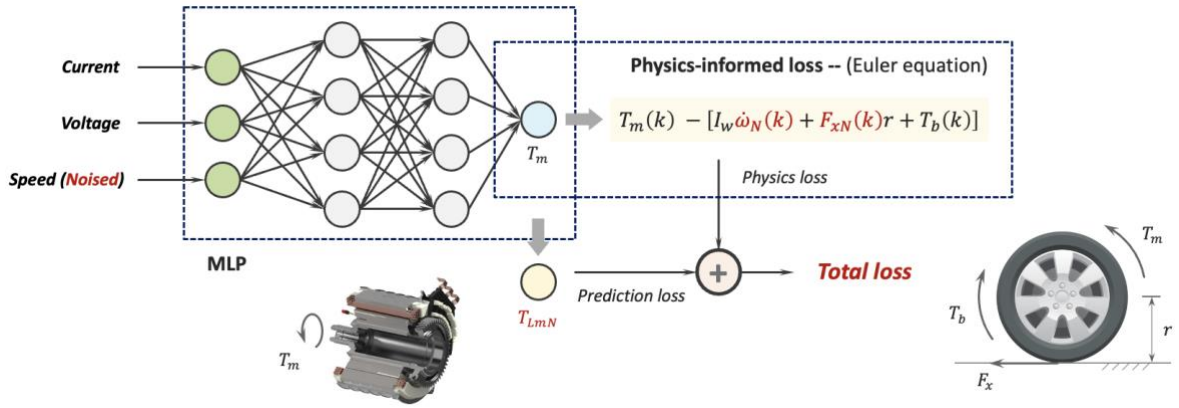

Figure 21: Architecture of PINN for Torque Estimation

The above architecture includes a multi-layer perception model which is a form of feedforward neural network with several interconnected layers of neurons. Each layer performs a distinct job, allowing the network to learn complicated patterns from incoming data. The phrase "multi-layer" alludes to the presence of hidden layers sandwiched between

the input and output layers, which do the majority of the computational processing. These hidden layers are made up of many neurons, each of which is responsible for processing incoming data via weighted connections and an activation function. Every input layer of this model contains input data which is received and forwarded through the network. Each and every input node corresponds to a unique feature that is vital for the model's functionality. Meanwhile, the output layer produces the model's predictions, with the number of output nodes altering according to the issue type. Regression tasks typically have only one output node, but classification tasks require one for each class. Proper hyperparameter selection, such as the number of layers, neurons, and activation functions, is critical and frequently requires careful tweaking. Weight initialization is also important for convergence, and regularization techniques like dropout and weight decay can assist to reduce overfitting.

4.3 Net Architecture of the Torque Estimation Code

Neural Network & Forward Propagation

Before delving deeper into the breakdown of the code, it is pertinent to discuss the net architecture being used throughout different pieces of code. Neural networks are strong machine learning models made up of linked layers of nodes, each with weights and biases that control the network's behavior. In this sense, "neural network parameters" are the weights and biases associated with the connections between nodes in each layer of the neural network. These parameters are learnt during the neural network's training phase, which involves exposing the network to a large dataset and adjusting its parameters to reduce prediction errors.

Weights describe the strength of connections between nodes in neighboring layers, which influences how information flows across the network. Biases are extra parameters given to

each node that help the network learn more complicated correlations between inputs and outputs. The code uses MATLAB's load function to load these parameters from external files. Each file includes the settings for a certain neural network layer, such as the input layer's weights and biases, as well as the hidden and output layers. By loading these parameters, the code provides the neural network model with pre-trained values, allowing it to make predictions on fresh data without the need for retraining. For forward propagation, the input layer receives the features of the inputs that have been given denoted as $X = [x_1, x_2, \ldots, x_n]$ where n is basically the total number of features that will be used by the model for processing through its layers. A weighted sum z is calculated for each neuron in the hidden layers by multiplying the input values by their respective weights, summing the results, and adding the bias term:

$$z = \sum_{i=1}^{n} (w_i \cdot x_i) + b$$

This expression is then generally expressed in the form of:

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

where Z is the matrix of weighted sums of layer l, W is the weight matrix for layer l, A is the activation matrix from the previous layer and the bias vector for layer l. The weighted sums z is passed through an activation function f(z) to introduce non-linearity and compute the activations A of the neurons:

$$A = f(z)$$

The procedure is continued for each additional concealed layer until the output layer is reached. The output layer's activations $A^{[L]}$ represent the network's projected output. The final

activations $A^{[L]}$ of the output layer are used to compute the network's output $\hat{y}$, which is generally achieved by adding another activation function or a linear transformation:

$$\hat{y} = g(z^{[L]})$$

A loss function is used to compute the difference between the expected and actual outputs (y). There are several loss functions that compute the difference between predicted and true values and is used to train the network by modifying weights and biases to minimize loss. The first one is MSE is commonly used for regression tasks, where $\hat{y}$ represents the predicted value and y_i represents the true value for each data point.

$$L(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y_i} - y_i)^2$$

Binary cross-entropy loss is commonly used in binary classification problems. It shows the expected probability of the positive class and the true class label (either 0 or 1).

$$L(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - y_i)]$$

Categorical cross-entropy loss is mostly used for multiclassification projects which basically means having various classes to segregate the data into. The equation for this loss includes the predicted probability of class j for data point i, and C is the number of classes.

$$L(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y_{ij} \log(\widehat{y_{ij}})$$

Activation Function

Next concept used is an activation function which is a critical component of a neural network that adds nonlinearity to the network's calculations. It uses the weighted total of inputs plus a bias at each neuron to determine if and to what extent the neuron should be engaged, impacting the neuron's output. The MATLAB code given uses the ReLU (Rectified Linear

Unit) activation function. ReLU is one of the most popular activation functions owing to its simplicity and efficacy. It is defined as f(x) = max (0, x), which means it returns zero for negative inputs and the input value for positive inputs. This function also enables sparse activation i.e. only some neurons are kept active while others are maintained at zero. The purpose behind this implementation was to inhibit overfitting of variables by increasing the model's ability to learn all diverse aspects and decrease redundancy among neuron activations. ReLU allows for more efficient gradient propagation during backpropagation than other activation functions. This is because ReLU generates a constant gradient (1) for positive inputs, making gradients flow more smoothly across the network and mitigating the vanishing gradient issue. Another function that is essential to the foundations of the hidden layers in the neural network is the sigmoid function (below) as it brings nonlinearity.

$$f(x) = \frac{1}{1 + e^{-x}}$$

By transforming real-valued integers to the range [0, 1], the network can represent complicated patterns and perform binary classification tasks. During forward propagation, the sigmoid function takes the weighted sum of inputs, adds a bias term, and uses the sigmoid transformation to generate the neuron's output. This output, which represents the activation level, is subsequently forwarded to the following layer for further processing.

Tanh functions, like sigmoid functions, add nonlinearity to neural networks and are frequently utilized in hidden layers. They map real-valued integers to the range [-1, 1], with the added benefit of being zero-centered, which helps ease the vanishing gradient problem. During forward propagation, the tanh function works similarly to the sigmoid. It takes the weighted sum of the inputs, adds a bias factor, then performs the tanh transformation to produce outputs ranging from -1 to 1. These results are subsequently forwarded to the subsequent layer for further processing. The tanh function can be expressed like this:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

The neural network model built for torque estimation uses both sigmoid and tanh functions as activation functions. These functions introduce nonlinearity, allowing the network to record complicated correlations between input characteristics and torque output. During MATLAB forward propagation, the sigmoid and tanh functions are applied element by element to the weighted inputs before transferring the altered values to the next layer. The selection and tweaking of appropriate activation functions considerably improves the network's capacity to properly predict torque, allowing the autonomous mobile robot to operate more efficiently.

4.4 Torque Estimation MATLAB Code

Data Pre-processing of test results

The purpose of the MATLAB code is to provide a visual representation of the torque estimation that the testing data which has been collected from the AMR during experimental run. Both of these cases use two different kinds of datasets (one uses normalized and not normalized while the other uses the test dataset) for implementing the prediction neural network to compare the different results. In the code, ‘udata’ is created which is a subset of few columns from the normalized dataset, trained on pre-trained parameters and undergoes further processing. The data processing is done by selecting portions of data from different columns based on ‘tstart’ and ‘tend’ indices. It then calculates the RMSE, maximum error and displays the results. As mentioned before, all the physics-informed and prediction losses will also be weighed in before plotting a graph. Initially, the code just reads the test result file and does the preprocessing of the data by splitting it into 3 different sets (x_con, ydata, udata).

4.6 Results and Discussion

After the breakdown of the code for torque estimation case, it is vital to understand the implications and behavior of the graphical representations for both cases. In the results section, torque estimating techniques are used to assess the performance of the next-generation e-drivetrain for autonomous mobile robots. Two separate scenarios are evaluated to determine the system's accuracy and dependability. The first scenario concerns torque estimate based on a single timestamp, which provides information about the e-drivetrain's instantaneous performance under various operating situations. In comparison, the second scenario takes a more complex strategy, combining data from three earlier timestamps. The graph from the first case i.e. single timestamp torque data is shown below.

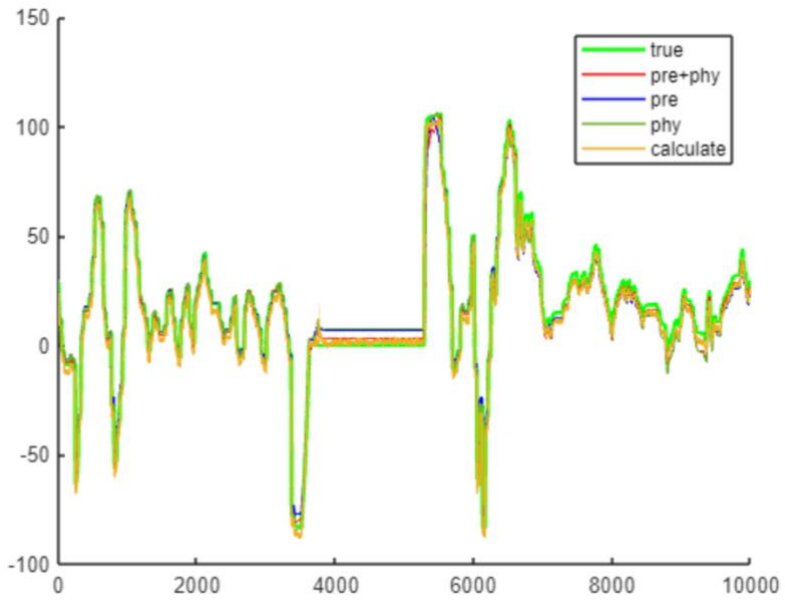

Figure 27: RMSE Results for single timestamp Torque Estimation

The graph has five lines that reflect different sets of data. The green line reflects the genuine data collected from the dataset for a specific time period. The red line indicates values computed using the Pre+Phy model, which combines prediction and physical properties. The blue line indicates the prediction loss, which is the error introduced by the prediction model alone, whereas the light blue line represents the physical loss, which is the error introduced

by the physical model just. In addition, a yellow line shows computed values acquired using another method, presumably for comparative reasons. Each line's behaviour over time provides information on the accuracy and performance of the various modelling methodologies, which are measured using root mean square error (RMSE) and maximum error calculations. The model's predictions are well matched with the actual values, as seen in the graph by the tight overlap of the real, pre+phy, and compute lines. The dark regions may represent confidence intervals or error margins, which appear to be fairly small, implying that the model has a high level of assurance in its predictions.

The graph for the second case (3 previous timestamps) can be seen below as well.

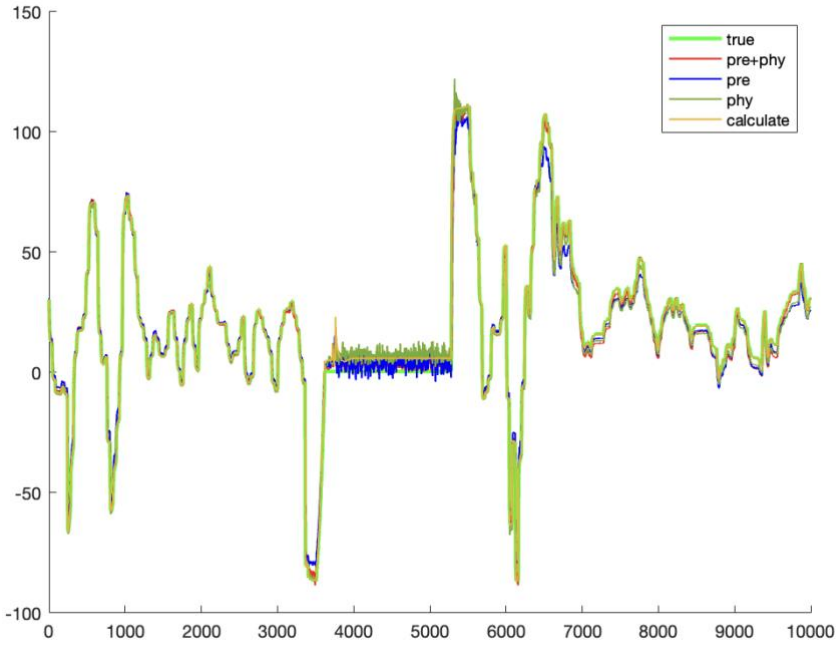

Figure 28: RMSE Results for multiple previous timestamps Torque Estimation

The lines are consistent with the first graph, however you may note that the pre+phy line is smoother and more steady. This may imply that by adding prior information, the model may produce more robust predictions while filtering out noise or variations in the data. The legend for both the graphs is the same for ease in comparison analysis with the same set of losses

and variables. Overall, the introduction of extra temporal information (as seen in the second graph) appears to result in a model that can more effectively smooth out predictions, potentially providing a more stable and dependable torque estimation under changing conditions. This shows that incorporating previous data points can improve the model's prediction capabilities, which is especially beneficial in real-time systems where the near past might provide insight into the current condition.

4.7 Comparative Graphical Analysis

Upon closer observation of both graphs, it is possible to delve deeper into two separate instances where the RMSE yield is the lowest with the combination of prediction and physics informed loss. From the two instances highlighted, it is visible that the all the three losses follow the true line very closely when it is in the range from 1650 to 2000 but they are constantly predicting a little less than true value from 7500 to 9000 range. This behaviour indicates that prediction in both these cases is minimal as compared to other data points but accuracy is higher when dealing with a lower range. The comparison graphs can be seen below with the RMSE values for reference as well. It can been seen that there are 2 intervals of graph between the absolute peak observed where the accuracy rates are higher as indicated by the closeness of losses to the true line.

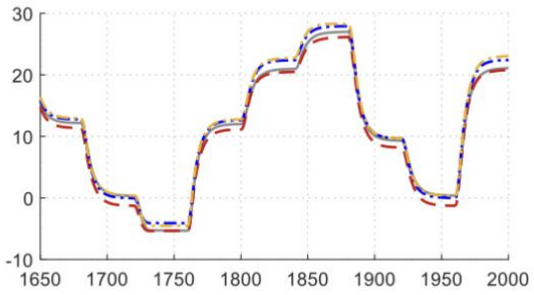

Figure 29: RMSE values for all losses for instance 1 (left arrow)

This graph compares the performance of three different models: "Pre," "Phy," and "Pre+Phy." The x-axis depicts data points ranging from 1650 to 2000, while the y-axis indicates the projected values. The "True" line shows the real ground truth values. The "Pre" model, shown in blue, indicates predictive loss, whereas the "Phy" model, shown in red, reflects physical loss. The combined "Pre+Phy" model (shown in purple) closely matches the real data, suggesting improved performance. This shows that combining prediction- and physics-based techniques produces more accurate outcomes.

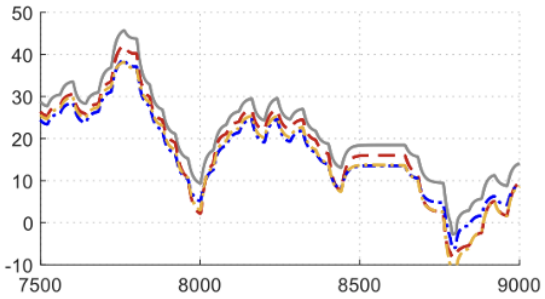

Figure 30: RMSE values for all losses for instance 2 (right arrow)

This graph is similar to the first graph as it shows consistency in terms of the Pre+Phy line which is very close to the True line with the least deviation compared to the singular loss functions.

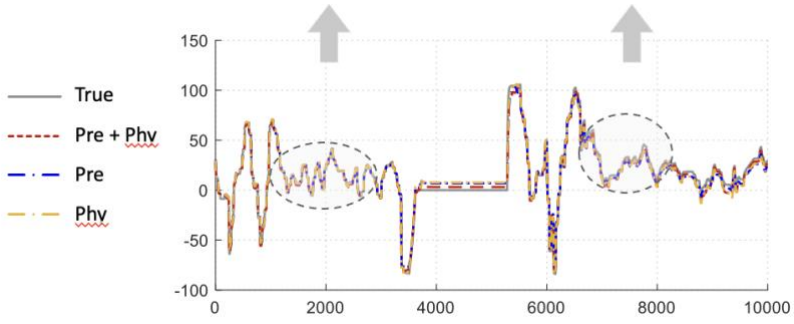

Figure 31: RMSE values graph with focus on minimal loss sections

The third graph here represents the entire range of data points for which the loss functions have been plotted. Upon closer observation, it is very obvious to see that a combination of

prediction and physics informed loss function is much more accurate to the true value. This suggests a successful neural network model implementation as it helps the theoretical function to minimize its losses as compared to the true function. This inference can also be validated by looking at the table below which indicates a big difference between combined loss function to the singular ones.

Table 1: RMSE values for the 3 different losses

	Pre + Phy	Pre	Phy
RMSE	3.7484	4.7387	4.8975

5. CONCLUSION

In conclusion, the concept of performing condition monitoring for an AMR is still a vastly challenging task to successfully implement due to the nature applications that the robot can face in outdoor working environments. For the course of this experiment, using a next generation e-drivetrain robot for condition monitoring explores a relatively new field of robotic applications. The selection of software and machine learning concepts used for creating a reliable graphical user interface was the backbone of this research because using MATLAB for developing it turned out to be a good choice for dealing with serial communication ports and single data frame per second transfers. Although, the use of certain machine learning concepts like leveraging cloud-based inputs through PyTorch and TensorFlow, Convolutional neural networks or hyperparameter tuning could further enhance the learning model used for torque estimation.

Theoretically, condition monitoring complexities depend on the type of system architecture used like DAQ centralized, edge node or DAQ distributed systems because the circuit response time varies. Hence, selecting a system with suitable architecture for the existing

environment is the biggest hurdle due to compatibility with relevant machine learning models for predictive modelling of different entities.

Furthermore, instead of building a separate predictive model using a platform, it is possible to use an edge computing framework like NVIDIA Jetson or perhaps AUTOSAR which could deploy machine learning models directly through the processing unit of the AMR and enable the robot to perform dynamic plotting with much more ease. This could also eliminate potential issues faced with the RS485 protocol during the formulation of graphical plots using data bits received from the robot.

The scope of this experimental investigation is limited to monitoring a single quantity, but this application only serves its purpose if it's able to estimate multiple quantities with the ability to extract data from the robot and simultaneously represent it into a visual presentation useful to the user. For instance, in a heavy machining production unit, it is imperative to monitor factors like current, voltage, speed, frequency in order to achieve the industry 4.0 standards which state criteria that deems an IoT or automated workplace. In order to achieve data transfer of multiple variables per second, it is vital to establish serial ports for several objects (baud rate, parity, stop, data bits for this case) and implement successful data transfer functions for each using the GUI.

Finally, this research piece acts just as a beginning of the potential opportunities that lie ahead in the field of AMR management applications with multiple entity data collection and real-time estimations of all concerned variables processed and calculated every second for automated environment through predictive learning models running machines.