Predicting Fault Behaviors of Networked Control Systems Using Deep Learning for Mobile Robots

Conor Wallace, Patrick Benavidez, and Mo Jamshidi
ACE Laboratories
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249
Email: conorw8@gmail.com, patrick.benavidez@utsa.edu, mojamshidi4@gmail.com

***Abstract*—The field of robotics research is continuously expanding at an ever-increasing rate. So much so, that as a systems' complexity grows, so too does the amount of possible points of failure. In recent years, these systems have been integrated together to create systems of systems, dramatically increasing the fragility of these networked systems, also known as a swarm. This paper presents a method for abstracting the fault of a networked control system, namely a system of mobile robots, into general feature sets and producing the capability of predicting the present fault as well as the compensation thereof.**

***Index Terms*—Deep Learning, Fuzzy Logic, Long Short-Term Memory Units, Recurrent Neural Networks, Robot Operating System, Unmanned Ground Vehicles**

I. Introduction

A. Introduction to Mobile Robot Fault Diagnostics

Mobile robot systems have an adept capacity for system of systems engineering and research. This is because of their potential for cooperative behavior in real world situations, whereas previously such roles have been enlisted by humans. Suppose a battalion of Unmanned Ground Vehicles (UGV) approaching a target destination, all of which are following the lead of an independent Unmanned Aerial Vehicle (UAV). Each child system in this networked formation is both individually complex and cooperative within their parent system. This individual complexity leads to fascinating accomplishments in the field of Networked Control Systems (NCS), however this also presents a multitude of possible faults including but not limited to: wheel slippage, time synchronization, time delay, data packet dropout, and cybersecurity threats. Using the hypothetical above, consider that the UAV has a tolerance for a percentage of data packets to be dropped before control of the UGVs. If an unknown threat hacks the on-board computer in order to take over the parent system, this system may not be intelligent enough to predict that an outside threat has taken over and corrupted this system. This is an example of just how detrimental these faults can be. In order to address this problem, a system is needed that autonomously detects the presence of this fault which causes this anomaly, whether internal or external, and compensates for such a fault. This can be done by perpetuating the fault to a test system, recording the anomaly data, and developing a set of rules for compensation given the particular fault driving the anomaly.

*This work was supported by Grant number FA8750-15-2-0116 from Air Force Research Laboratory and OSD through a contract with North Carolina Agricultural and Technical State University.

This problem has given way to an expanse of research into possible solutions for fault diagnostics as well as the compensation of such faults. For example the research completed in [1], the author discusses the effects of time delay and data packet dropout as a hindrance for a given NCS and proposes a novel approach for taking advantage of the packet-based transmission whereas a typical control system processes a single control variable at a time, this new method could process an entire packet at a time. In [2], the author discusses the effects of data corruption and proposes a solution whereby data is encoded before transmission so as to design an optimal controller for a distributed control system. Random data loss is discussed in [3] where an observer-based feedback controller is developed to handle such losses where the data loss is assumed to follow Bernoulli random distribution. The authors in [4] have developed a recurrent fuzzy network for the control of fluctuating temperatures. Further neural-fuzzy research was conducted in [5] where the authors have developed a neural-fuzzy controller to predict the behavior of chaotic systems.

All of these works have been sources of inspiration for the design introduced in this paper, however, none of these introduce methods directly related to the concept of real-time fault prediction of autonomous vehicles. In [6], the authors have used a multi-agent diagnosis system where each agent reports the diagnosis of a particular fault to a parent agent. In an attempt to predict and compensate for faults of an autonomous underwater vehicle (AUV), the authors in [7] have developed a support vector machine (SVM) to be trained offline and validated online. The concept of using machine learning techniques to intelligently predict faults corresponding to fault data is the driving force in the scope of this paper. Artificial intelligence combined with the robustness of a fuzzy logic controller produce the intelligent controller that this research is attempting to develop.

978-1-7281-0457-7/2019/$31.00

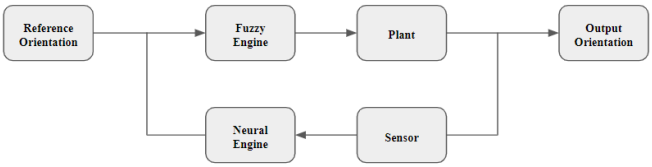

Fig. 1: Architecture of the Neuro-Fuzzy Controller

II. Neuro-Fuzzy Controller

This paper presents a novel method for using a neuro-fuzzy control system. Previous work has been done related to neuro-fuzzy controllers, but none have been used in fault diagnostic implementations.

A. Neuro-Fuzzy Controller

In order for an autonomous system to become self-sufficient, it will have to have the intelligence to diagnose its own faults in order to retain stability and avoid failure. For the purposes of the research in this paper, a system with the intelligence to predict its own fault is required. This intelligence is acquired using the block diagram of the control loop designed in this paper which can be seen in Figure 1. This model uses two classic soft computing technologies; a neural network and a fuzzy logic controller. Typically, neural networks are used for the purpose of identification, whether it be of simple objects or complex system dynamics. Whereas fuzzy logic control is implemented in various applications where human expertise is desired for making complex decisions where multiple parameters must be taken into account. Using these two technologies together permits a system to make state identifications as well as a decision based upon the current state. In the context of fault diagnostics research, the current state refers to the performance of the subsection of the system being analyzed and whether or not this performance is ideal or not. The performance of these subsystems can drastically decrease the stability of the total system and in the event that performance is not ideal, accommodations must be made to push the system towards stability. This is done by performing control strategies based upon fuzzy inferences.

III. Fault Diagnostic Neural Network

In order to endow the control architecture with predictive capabilities, a machine learning algorithm has been developed to predict the upcoming fault given the error data.

A. Recurrent Neural Network

The breadth of machine learning techniques that have been developed and honed in the last decade have given birth to astounding accomplishments in the field of artificial intelligence. Artificial neural networks (ANNs) posses the ability of reducing complex dynamic mathematical models to a simple relationship between an input and an output by creating a general line of best fit to a set of data. This is done by iteratively adjusting a set of weights and biases, by way of a gradient descent algorithm, which classify which perceptrons help identify where along this line of best fit this data lies. However, generic ANNs can only analyze data from a single time step and often dynamic models require predictions based off of more than just a single time step of data. This is the thought process that drives Recurrent Neural Networks (RNNs). Analyzing data from a sequence of time steps allows a model to predict an outcome based off of how the system changes over time. The architecture of an RNN is very similar to that of a traditional ANN except that the output of every perceptron is recurred as an input to the next perceptron. This behavior is represented mathematically equation (1) where *t* is the number time steps for the network to backpropogate, *x(t)* is the input data of the current time step, omega is the weight matrix, beta is the biases matrix and *F(t+1)* is the prediction for the next time step:

$$F(t+1) = \sum_{n=0}^{t} [(x(t) * \vec{\omega}) + \vec{\beta}] \tag{1}$$

B. Long Short-Term Memory

RNNs provide the temporal analysis desired for fault detection over time, however, the use of basic perceptron cells all but render this architecture inert by way of the phenomenon known as vanishing and exploding gradients. Each weight is optimized via gradient descent calculations based on the change of data given these adjustments. Exploding gradients occur when a drastic change is given to a certain weight which causes the output of the gradient to go towards infinity, however this problem is fairly simple to address by simply squashing these gradients into a certain range. Vanishing gradients occur any time a gradient close to zero is computed which does not allow the model to continue learning. This problem is the reasoning for the Long Short-Term Memory (LSTM) cell's creation. A model of a LSTM cell can be seen in Figure 2 and it can be seen that each of these cells are made up of an input gate, an output gate, and a forget gate. These gates allow the cell to determine if information is important, thereby allowing the data to pass to the output gate, or irrelevant, thus being deleted in the forget gate. Each gate contains a sigmoid function which squashes the data to a range of 0 to 1. This restriction prevents gradients from being reduced to zero, thus allowing the model to continue learning.

C. Architecture of the Model

The test parameters can vary with each new fault being tested on. The fault and the error data it instigates can easily be formatted to function in this design. All that is needed are the classifications of the fault and a data set containing attributes that are affected by said fault. Sense the model simply generalizes to a set of data, it is only required for the data to be in the format that the model expects which are referred to as tensors. The two frameworks used to develop the model were tensorflow [8], and keras [9]. Because of the selective nature of the LSTM unit, any data can be fed into the model and the LSTM units will be able to determine its

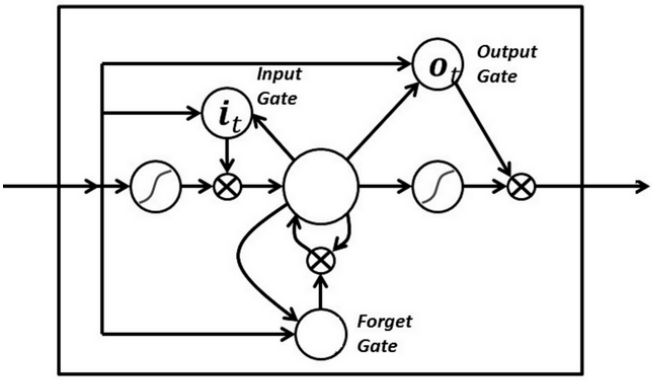

Fig. 2: Model of a LSTM Cell. Reprinted from Recurrent Neural Networks and LSTM, by Niklas Donges, 2018, Copyright 2008 by the Towards Data Science 2018.

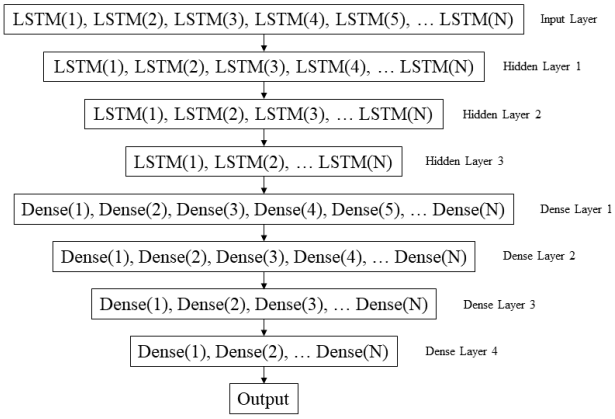

Fig. 3: Fault Diagnostic Model with LSTM and Dense Units, Notice N is Diminishing as the Model Approaches the Output Node

relevance. The model is comprised of an input LSTM layer, three hidden LSTM layers, four dense layers, and an output layer. The architecture of this model can be seen in Figure 3. The number of LSTM units for each layer affects the fitting of the model. If the ratio of the number of units to the batch size is too high, the model could overfit. Similarly, if the ratio is too low, the model could underfit. A model is overfitting if the training set converges to a certain accuracy, but the testing set doesn't increase in accuracy. In other words, the model has memorized the training set and consequently can't make predictions on the new testing set. On the other hand, a model is underfitting when the training and testing sets fail to increase in accuracy, i.e. the model is not learning. To help address this concern, the number of units decreases as the model approaches the output layer so as to help the model converge more effectively.

IV. Fuzzy Logic Controller

Due to the situational prediction of the fault diagnostic RNN, the linguistic properties of a fuzzy control system

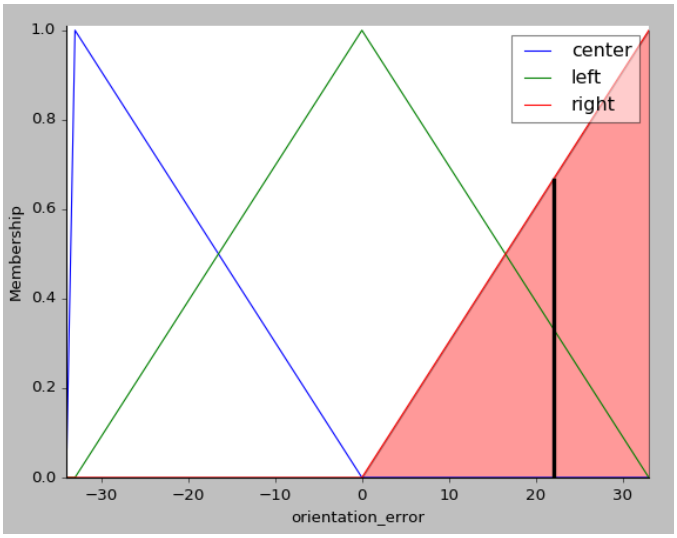

Fig. 4: Fuzzy Logic Membership Functions

lends itself well to making meaningful interpretations on the predicted output. Using fuzzy logic, a fault and the induced error, data can be intelligently correlated with each other.

A. Fuzzy Logic Engine

Rather than using a classic controller such as a *PID* controller, this system uses a fuzzy logic controller to make compensations to the plant. Fuzzy control systems determine compensation values with an expansion of knowledge about a particular system. This allows the designer to create an accurate control system even when the mathematical model of the system is either unknown or too complex. A fuzzy controller is composed of a set of control rules based on the control parameters of the system and a set of membership functions for assigning these control parameters and thus computing an output compensation. In the experiment discussed in this paper, the rule set was based on tire inflation faults for a four-wheeled UGV which can be seen below where F_n is the predicted fault in the system, T_n is the tire pressure combination, C_n is the control variable for the system, and X_n is the computed compensation:

$$\textbf{IF } F_n \; is \; T_n, \; \textbf{THEN } C_n \; is \; X_n$$

The fuzzy membership functions used for this model were trim functions which create simple triangle membership fields based on the range of a given input. Scikit Fuzzy contains a useful Application Programming Interface (API), for developing fuzzy control models in python, its documentation can be found in [10]. The output of these membership functions the model depicted above can be seen in Figure 4, where either all the tires were inflated, the left two tires were deflated, or the right two tires were deflated. The results of this controller will be discussed in section VI. The same approach can be applied to a much more complicated rule set once a more detailed data set is collected.

V. Technical Issues

The computational complexity of the blocks of this feedback control loop introduce a latency issue. Due to the transmission of data in ROS, this issue could have a negative effect on the real-time performance of the controller.

A. Real-Time Data Distribution

The Robot Operating System (ROS) has become one of the most popular frameworks for networked robot development. ROS is essentially a data distribution service (DDS) that allows multiple systems and subsystems to make data easily accessible across the network. However, ROS publishes data packets using the notorious Nagle's algorithm, a method of increasing data transfer efficiency by filling a data packet buffer so to only publish data when there is a sufficient collection to transfer. With lower latency control schemes, this is never an issue and is negligible. However due to the high computation requirement of the proposed neuro-fuzzy control architecture, this presents a significant design challenge where the delay between data packets will cause the performance of the system to become unstable. To address this, the ROS argument *tcp_nodelay* is set to true to allow the constant flow of data as soon as it is available. The manner of collecting data for the RNN also presents a challenge due to the difference between capturing live data as compared to static data. In order to compensate this, a queue was implemented so that each element of the queue is a time series of input data collected as soon as it is made available and once the sequence has been analyzed, that sequence is popped off the queue. Using this approach, the RNN can efficiently process each time series without skipping data and creating instability in the system.

B. Feedback Control Loop

Figure 5 will be used in order to illustrate the control flow of the intelligent loop designed in this paper. The control framework entirely functions inside the Robot Operating System (ROS) master node. The specifics of ROS nodes and methodology can be found in [11]. It begins by publishing a 1 to *cmd_motor_state* which enables the robot's motors. Next a linear velocity is published to the robot's *cmd_velocity* topic to drive at a constant velocity. The system continues by retrieving a *sensor_msgs/IMU* topic reading from an Inertial Measurement Unit (IMU) sensor node, then saves this data to a rosbag which is then formatted into a file that is readable by the RNN model. Next the data is passed through the model which then makes a prediction based off the data. Then the prediction is passed to the fuzzy logic controller in the form of a *std_msgs/String* topic where the rule sets determine what the expected compensation should be and pass this compensation value to the plant in the form of a *std_msgs/Float32* topic. Finally the plant will make these compensations and continue to take readings, thereby closing the loop. In order to achieve such computationally complex tasks in real time, a queue was used to keep a backlog of fault data, on which predictions are run. This way as the system progresses, the neural network can function independently to make predictions on its current status. Following this design, while a prediction is not ready, the fuzzy controller will use the last known prediction so that at any time the system is aware of its fault. This intermediate state will cause the system to oscillate about an axis until the system eventually converges and becomes stable.

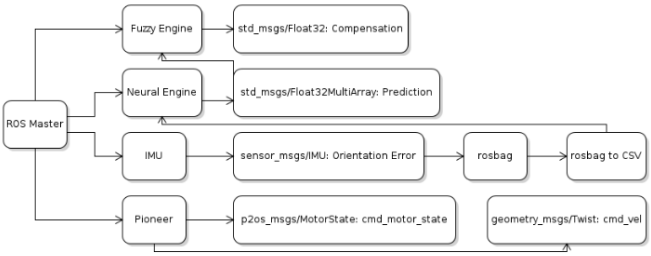

Fig. 5: ROS Neuro-Fuzzy Control Flow Diagram

VI. Experimental Setup and Results

A. Pioneer UGV and Orientation Fault Data

In order to validate the viability of the control paradigm developed in this paper, a four-wheeled robot was introduced to the fault of deflated tires. The tire deflation drastically reduces the stability of the mobile robot when approaching a given pose. The robot used was the Pioneer 2 four-wheeled UGV in cooperation with an Adafruit BNO055 IMU as can be seen in Figure 6. The goal in this experiment is to gather a large data set relating the parameters of the system to the fault at a given time step. The system parameters recorded for this experiment were the timestamp, x-axis orientation, y-axis orientation, velocity, and the tire inflation levels. Rosbags, a commonly used API of ROS, allows for the rapid collection of all relevant system parameters. Rosbags allow the user to collect data at any given frequency. In the scope of this experiment, the IMU sensor publishes data at a rate of thirty-four data packets per second. Given this frequency, a course with a length of 5 meters and a velocity of 0.2 meters per second, one test run produces roughly 400 data samples. Due to the learning methods involved with neural networks, a sufficiently large data set is desired. Therefore these tests must be carried out many times to increase the accuracy of the model. In order to make valuable predictions about the fault presented, a sufficient amount of test classes must be performed. For this experiment, the test cases used were the sixteen possible combinations of the Pioneers tires set to intervals of 25 percent, giving a total of 64 test cases. The process of collecting this data consisted of deflating the Pioneers tire to the level of one of these test cases, calibrating the IMU using an Arduino Mega, and finally driving the Pioneer along a straight course. It is important to note that the robot was driven along a straight path without the compensation of a controller in order to test the system in an open loop environment and to prove that the system grows unstable as the fault grows. This experiment is entirely running in the ROS environment making for both efficient data transfer as well as data capturing. The devices involved in this experiment include

sensors, motors, drivers, and micro-controllers, all of which utilize their own ROS topics for publishing and/or subscribing. The ROS environment architecture is comprised of an omniscient master node, a Pioneer robot node, and an IMU/Arduino node. Each node may be a collection of several data topics. For example, the Pioneer makes use of the *cmd_motor_state* and *cmd_velocity* topics where *cmd_motor_state* is a flag that activates the Pioneer's motor drivers and *cmd_velocity* controls both the linear and angular velocity of the robot. The IMU makes use of the rosserial topic to publish all relevant data topics to the IMU through a serial USB connection via the topic *sensor_msgs/IMU*. This configuration of the ROS environment allows the Pioneer to complete its course, independent of the IMU's readings, which results in legitimate data. Using this experimental procedure, an expansive data set was collected.

B. Intelligent Control of a UGV

Using this data set, the fault diagnostic RNN could be properly trained and tested. In order to have an accurate representation of the model's accuracy, the data set was partitioned into a training set and a testing set with partition sizes of 70% and 30% respectively. However, before this occurs, the data set must be randomized so as to avoid over-fitting the model. To do this the data set was split into sequences of 10 data points and then the set of these sequences were shuffled. The data was then fed through the model over a course of 50 epochs so that the model could generalize to the sequence of data being fed through. Once the model is trained, the test set would attempt to introduce new data in order to verify the model's accuracy. With the model validated, an accurate prediction of the present fault could be made, which would be the input of the fuzzy engine. The fuzzy engine's membership function was developed with this experiment's fault cases in mind. This was done by becoming familiar with the expected error in orientation associated with each tire pressure case. For example, given that both of the Pioneer's left tires are fully deflated, the Pioneer is expected to have a drift of $-22°$. With this knowledge, a set of fuzzy rules were implemented corresponding with triangular membership functions. Given these experiment parameters, the control loop transaction begins without a prediction until a full data sequence is collected (10 data points / 34 data points per second = 294 milliseconds until the first prediction). While a prediction is being processed, the fuzzy engine assumes no fault and the system performs as an open loop system. After the first prediction is made, the fuzzy engine computes the corresponding compensation to the plant. The latency of the fuzzy engine is much lower than that of the neural engine and thus there is a delay between predictions. During this delay the system performs once again as an open loop system. This behavior continues to oscillate until the system converges.

C. Results

At this point in time each of the control blocks have been fully developed and tested and the results are very promising.

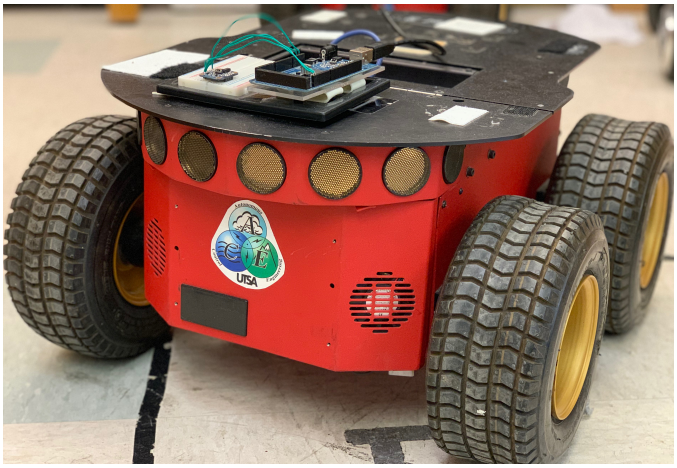

Fig. 6: Pioneer UGV with BNO055 IMU Sensor

The data set collected contains 160,000 samples comprised of the sixteen possible tire combinations. In the scope of machine learning models, this is still a relatively small training sample, however, due to the design of the experiment, a much larger data set can easily be captured. Nevertheless, the training set provided remarkable results. After training on the entire training set for 50 epochs, the model reached a training accuracy of 94%. Next the remaining 30% of the data set was used for testing. Once the model completed training, the test set acquired an accuracy of 93%. The plots of the accuracy and cost function for the model can be seen Figures 7 and 8 respectively. Given a larger and more detailed data set, these numbers can increase, however these results are very promising given the testing conditions. With the fault prediction network providing accurate results, the fuzzy logic engine could be tested. The input of the fuzzy engine is the output of the fault prediction model. In other words, the tire fault that is predicted is the input of the fuzzy engine. Using this approach, the fuzzy engine is able to make an accurate compensation base on the expected error corresponding to the predicted fault. With all control blocks fully developed and tested, the closed loop system was tested. Using ROS, the output of each block was published and subscribed to as an input to the next block, thus it can be seen that the performance of each portion of the control system is a sufficient proof of concept.

VII. Future Developments and Conclusion

The next stage of development is to apply this approach to a legitimate System of Systems (SoS) in the form of a UGV/UAV relationship. The concept for this new experiment is to use a tracker with both the UGV and UAV and a lighthouse system where the lighthouse is a device that emits beams of light for which the tracker sensors on the unmanned vehicles can use for absolute localization. The UAV will be given a camera system which will be used to visually track the UGV and control its navigation. There are several fault tests to be done given this relationship including data packet dropout,

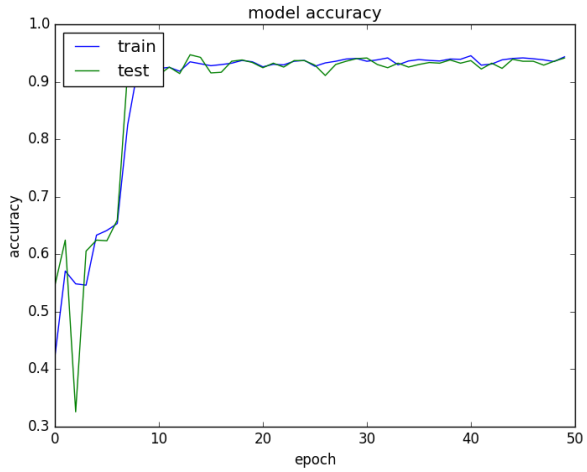

Fig. 7: Plot of Neural Network Prediction Accuracy Given 4 LSTM Layers and 4 Dense Layers

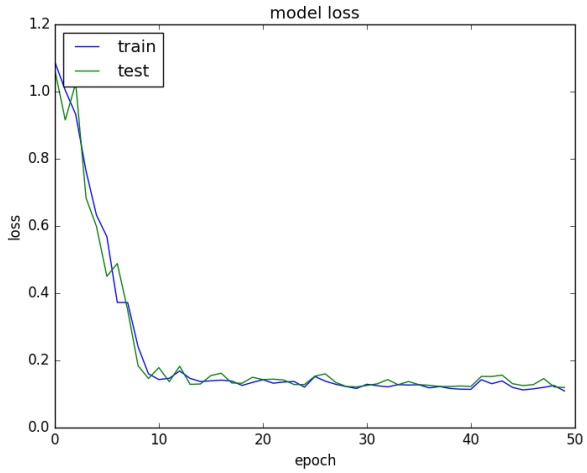

Fig. 8: Plot of Neural Network Prediction Loss Given 4 LSTM Layers and 4 Dense Layers

time delay, data corruption, single entity malfunction and more. The approach developed in this paper can be applied to this experiment as well. First by collecting data corresponding to the fault detected and then by developing a set of fuzzy rules based on the new data set. This approach can be applied to any quantifiable, non-stochastic system.

This paper has introduced an approach for a real-time neural-fuzzy control system for fault detection. Related work was discussed as a means of defining the desired approach and the literature supporting such a system has been discussed. The architecture of the system was described in detail and the corresponding functionality of the proposed system was discussed. A block diagram for the real-time neural-fuzzy controller was used to visualize both the workings of the system and how the neural engine works independently of the system to avoid latency issues. The neural network used was described in detail and the method for extracting and formatting data for the model was discussed. The fuzzy engine that produces compensation for the system was examined as well as its rule sets and membership function functionality. Finally, the experiment used to develop and test such a designed system was described as were the experiment's results. The future improvements of the design were discussed and so was the next step of development using a system of systems with a UAV/UGV relationship. The research discussed in this paper clearly portrays the real-world functionality of such a system and eludes to further research in such categories for years to come.

REFERENCES

[1] G.-P. L. Yun-Bo Zhao and D. Rees, "Design of a packet-based control framework for networked control systems," *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 2009.

[2] V. Gupta, "Distributed estimation and control in networked systems," 2006.

[3] M. S. Mahmoud, "Intelligent approach to uncertain networked control systems with random packet losses," *SAI Intelligent Systems Conference*, 2015.

[4] J.-S. C. Chia-Feng Juang, "A recurrent neural fuzzy network controller for a temperature control system," *Proceedings of the 12th IEEE International Conference on Fuzzy Systems*, 2003.

[5] H.-H. C. Chaio-Shiung Chen, "Robust adaptive neural-fuzzy-network control for the synchronization of uncertain chaotic systems," *Nonlinear Analysis: Real World Applications*, 2009.

[6] Z. C. Yi Lu Murphey, "A multi-agent system for complex vehicle fault diagnostics and health monitoring," *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*, 2010.

[7] C. S. L. V. G. Antonelli, F. Caccavale, "Fault diagnosis for auvs using support vector machines," *IEEE International Conference on Robotics and Automation*, 2004.

[8] tensorflow.org. Tensorflow python documentation. [Online]. Available: https://www.tensorflow.org/versions/r1.13/api_docs/python/tf

[9] keras.io. Keras documentation. [Online]. Available: https://keras.io/

[10] pythonhosted.org. Scikit learn fuzzy documentation. [Online]. Available: https://pythonhosted.org/scikit-fuzzy/overview.html

[11] ROS.org. Ros technical overview. [Online]. Available: http://wiki.ros.org/ROS/Technical Overview