THRESHOLD-FEEDFORWARD NEURAL NETWORKS WITH APPLICATION TO TRACTOR AUTOMATED GROUND LEVELING

by

TIEN-CHUONG LIM

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTORATE OF ELECTRICAL AND COMPUTER ENGINEERING

2025

Oakland University
Rochester, Michigan

Doctoral Advisory Committee:

Ka C. Cheok, Ph.D., Chair
Subramaniam Ganesan, Ph.D. Co-Chair
Osamah Rawashdeh, Ph.D.
László Lipták, Ph.D.

To my wife, Nguk-King Diong, who has supported me throughout the years in everything I do, including my academic endeavors. Words cannot express my gratitude to her.

ACKNOWLEDGMENTS

With tremendous gratitude, I thank my academic advisors, Chair Dr. Cheok and Co-Chair Dr. Ganesan, for their tireless efforts and investment in me. This dissertation would not have been possible without their persistent guidance and support. They gave me hope and reasons to continue during my moments of self-doubt and darkness. In the early stages of my research, Dr. Ganesan's constant encouragement and prompt feedback on all my questions helped me navigate the complexities of PhD studies. Dr. Cheok's deep knowledge of the subject matter in my research area was crucial to my final push to complete this laborious yet worthwhile journey. It has undoubtedly been one of the most challenging tasks I have ever undertaken. I will forever cherish this now that I have finally achieved this milestone I have dreamed of for a long time.

Secondly, my dissertation committee members, Dr. Rawashdeh and Dr. Lipták, were more than supportive throughout my studies. They have tolerated my occasional absences during active research phases, which are sometimes dormant due to COVID or other reasons, only to re-appear for various gate reviews required for this degree.

Tien-Chuong Lim

ABSTRACT

THRESHOLD-FEEDFORWARD NEURAL NETWORKS WITH APPLICATION TO TRACTOR AUTOMATED GROUND LEVELING

by

TIEN-CHUONG LIM

Adviser: Ka C. Cheok, Ph.D.

This dissertation presents an enhanced supervised machine learning method, referred to as the Threshold-Feedforward Neural Network (TFNN). The TFNN operates on continuous inputs, generates discrete outputs, and effectively produces superior classifications of outputs in a noisy environment. The TFNN is successfully applied to a tractor ground leveling system, increasing operator comfort and improving ground leveling with consistent quality. Training of the TFNN and Tractor Automated Ground Leveling (TAGL) simulations was formulated, applied, and verified using a virtual profile that detailed the terrain.

A tractor was equipped with a GPS receiver to verify the simulation results partially. Data were collected to show the receiver's ability to locate the tractor's position, altitude, and pitch angle. The rear implement arm angle was detected using motor position feedback sensing. The data gathered showed all necessary inputs and output information to feed into the simulation model to realize the theoretical TFNN results. Further work will equip the tractor with a display to show the tractor operator

real-time leveling error input, enabling the completion of TAGL training data gathering and implementation.

TABLE OF CONTENTS

TABLE OF CONTENTS – Continued

TABLE OF CONTENTS – Continued

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES—Continued

LIST OF FIGURES—Continued

LIST OF FIGURES—Continued

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ALVINN	An Autonomous Land Vehicle in a Neural Network
AGL	Automated Ground Leveling
ANN	Artificial Neural Network
BCI	Brain Computer Interface
CAN	Controller Area Network
CPU	Central Processing Unit
EEG	Electroencephalogram
FCM	Fuzzy C-Mean
FNN	Feedforward Neural Network
GPS	Global Positioning Systems
GPU	Graphic Processing Unit
HOG	Histogram of Oriented Gradients
k-NN	k-Nearest Neighbor
LMA	Levenberg-Marquardt Algorithm
LVQNN	Learning Vector Quantization Neural Network
ML	Machine Learning
MSE	Mean Square Error
NB	Naïve Bayes
RMS	Root Mean Square
RNN	Recurrent Neural Network

RTK	Real Time Kinematic
SVM	Support Vector Machine
TAGL	Tractor Automated Ground Leveling
TFNN	Threshold-Feedforward Neural Network

CHAPTER ONE

INTRODUCTION

1.1 Machine Learning Overview

Machine learning is a subset of artificial intelligence (AI), broadly defined as a machine's capability to imitate intelligent human behavior. Artificial intelligence systems are used to perform complex tasks, much like humans solve problems. Arthur Samuel, a pioneer in machine learning, did some important early work by applying machine learning algorithms to checkers' playing. His early papers and statements on machine learning can be summarized to define machine learning as a “field of study that gives computers the ability to learn without being explicitly programmed.” [1][2][3] A principal research scientist, Boris Katz, said, “The goal of AI is to create computer models that exhibit 'intelligent behaviors' like humans.” [4] The “learning” part of machine learning attempts to minimize error or maximize the likelihood of their prediction being true. Therefore, we can ask which error function one tries to minimize when performing machine learning. The initial guesses from the machine learning framework that multiplies inputs to arrive at the outputs or guesses can be quite wrong. Using neural networks or other methods, the algorithm measures the error and modifies the parameters in the algorithms until specific error criteria have been achieved. In the case of neural networks, the parameters are called weights and biases. [5] The following sections will discuss many other parameters that various machine learning types employ.

1.2 Machine Learning Types

There are three main types of machine learning: supervised, unsupervised, and reinforcement learning. My research interest is concentrated on Supervised Learning. Supervised learning (inductive learning) is the Machine Learning that uses training data that includes desired outputs. This is when we are given examples of a function in the form of data (x) and the output of the function $f(x)$. The goal is to learn the function for the new data (x). The learning is called "Classification" when the learned function is discrete. It is called "Regression" when the function being learned is continuous, and "Probability Estimation" when the function's output is a probability.

Supervised Machine Learning algorithms include SVM (Support Vector Machines), linear regression, logistic regression, Naïve Bayes, linear discriminant analysis, decision trees, k-nearest neighbor (k-NN) algorithm, and Neural Networks.

1.2.1 Supervised Machine Learning - Classification

Supervised machine learning is called classification when the targeted output variable is discrete, a particular category or class. [13] A simple classification example would be to classify higher learning institutions or colleges. The training data must have sufficient inputs and outputs, called data labels, to perform machine learning training successfully. The model is fully trained using the training data and then evaluated on test data before being used to predict new or unseen data. [5] A simple training data using colleges as output labels can be seen in the table below. The inputs are colors, the location of the college, and the school mascot. The outputs or data labels are the school

names that match the input description. In real life, ideally, there will be more input attributes other than the three inputs below. This example nonetheless serves as a simple illustration.

No	Inputs (Colors, State, Mascot)	Outputs (School)
1	Black, Gold, Michigan, Golden Grizzlies	Oakland
2	Scarlet, Gray, Ohio, Buckeyes	Ohio State
3	Maize, Blue, Michigan, Wolverines	Michigan
4	White, Green, Michigan, Spartans	Michigan State

Table 1.1: Machine Learning example inputs and outputs

The inputs and outputs above are fed into the training program or block diagram below, where the classification machine learning method is chosen to perform the machine learning training.

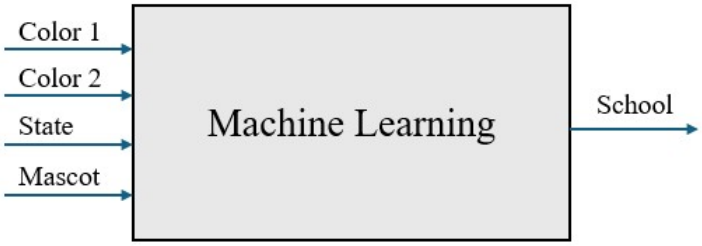

Figure 1.1: Machine Learning Example

After the program above is fully trained, we can feed an unknown school with Scarlet, Gray, Ohio, and Buckeyes inputs into the program. If properly trained, the output would be “Ohio State University.”

A second example is fruit classification research conducted by Nguyen et al. using computer vision pictures as the inputs to help with identification and classification to increase the efficiency of product packaging, arranging fruit, and detecting mismatches on corresponding shelves in supermarkets. Additionally, a well-trained fruit classification algorithm using pictures can help cashiers quickly identify the prices of fruits using images instead of a barcode scanner. The results show that the machine learning trained model has the best accuracy of 96%. [14] A similar study was conducted by Winda Astuti et al., which classified fruits based on their shapes. Fast Fourier Transform (FFT) was extracted from the images and fed into machine learning algorithms to produce 100% accurate results. [15]

1.2.2 Supervised Machine Learning – Regression

As indicated in section 1.2, the supervised machine learning method is called “Regression” or “Function Approximation” when the function being learned is continuous and “Probability Estimation” when the output of the function is a probability. The regression machine learning model has a wide range of real-life applications. It is helpful for any machine learning problem that involves continuous numbers. An example is the energy consumption prediction for a smart building by Mel Keytingan M. Shapi et al. [16] k-NN, SVM, and FNN were used to generate a predictive model. The result showed that the predictive model is highly data-dependent. No one regression above had

a significant advantage, though SVM performed slightly better than the other two. In another paper by A. Behera et al., regression machine learning was used for stock price prediction. [17] The data used for training included two years' worth of information, such as starting price, closing price, neighboring close price, and volume of trade, to generate a predictive model. In another study, Z. Liu et al. utilized an SVM algorithm to create a prediction model stock price using experiment data from the Yahoo Finance website with ten (10) representative companies, including Google, IBM, Amazon, etc., from 2000 to 2016. The data used included seven essential attributes: time, highest price, lowest price, opening price, closing price, trading volume, and adjusted closing price. Although a total prediction in stock price is impossible, the model demonstrated results with 61%-65% accuracy. [18]

1.2.3 Unsupervised Machine Learning

In an unsupervised machine learning algorithm, the data used for training is not labeled. In other words, no given output variable tells the learning algorithm the result to aim for or to achieve. An unsupervised learning algorithm is adept at discovering the unlabeled data's hidden structure and inferring the output pattern based on the density groupings of the inputs. One example is shown in Figure 1.2 below. The K-mean clustering algorithm identifies and groups the data points that are alike. In this case, the goal of machine learning is not to predict things; instead, it is about finding structures and putting them into appropriate buckets or groupings. One example is to group all US news categorized as such in the same grouping using unsupervised machine learning. Figure

1.2 gives a visual representation of unsupervised machine learning by detecting three clusters of dots and placing them into three groups.

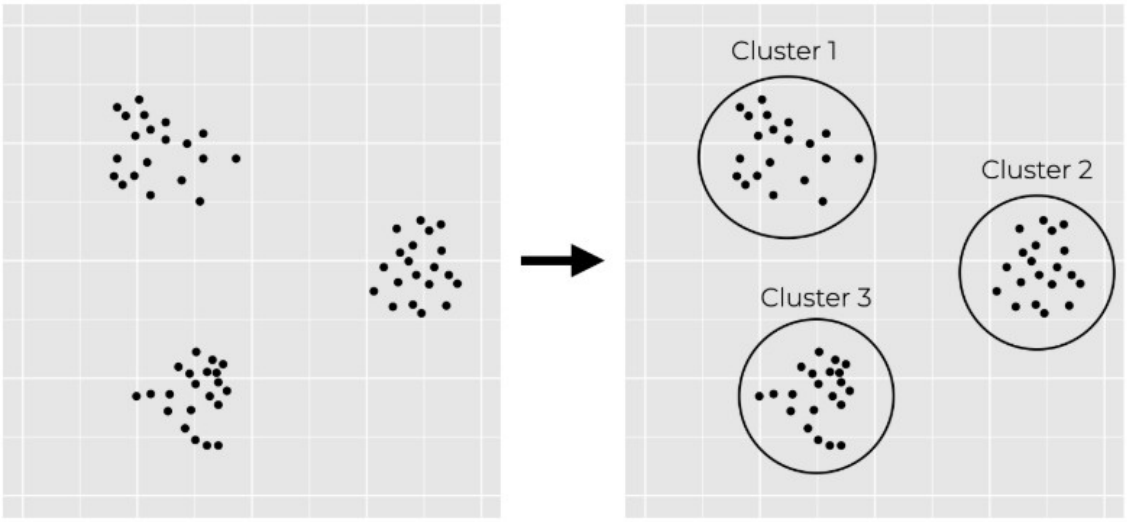

Figure 1.2: Unsupervised clustering of data [19]

The common unsupervised learning method is K-mean clustering, in which the algorithm partitions the N observations in space into k-clusters. The algorithm is iterative, with k-clusters split into Voronoi cells. It starts with a random selection of the K-means group spacing and continues to update with each iteration until the cluster mean of points ceases to change. [20] [46] The variant of K-means is called K-medoids, which have the centrally located data point as a reference point to the corresponding clusters. [21]

1.2.4 Reinforced Machine Learning

Reinforced Machine Learning is a trial-and-error mechanism that learns through constant action from an agent to the environment, giving constant feedback on whether

the action reaps any reward or produces errors. The interactions between an agent and environment are shown below:

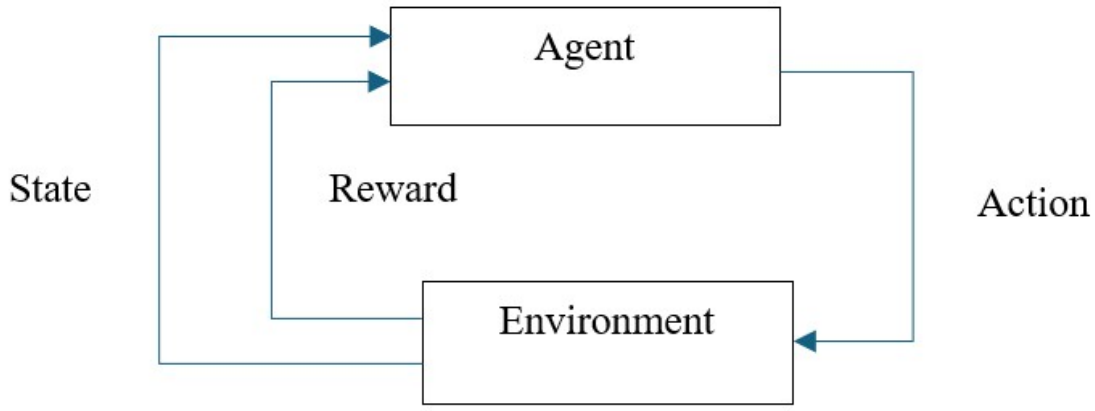

Figure 1.3: Reinforcement Learning Model

In the reinforcement learning model, the agent is not told what to do or not to do, such as the labeled output data. Instead, it tries to discover what action can produce the maximum reward. [22] This algorithm was first proposed by Samuel et al. and was first applied to the game of checkers by learning the value function expressed by a linear function to guide its action choices. [23] Many developments in the game-playing algorithms were very complex and uniquely suited for reinforcement learning. In 2016, AlphaGo, developed by Google's DeepMind team, caught the world's attention by defeating Go master Li Shishi, a significant breakthrough. Since then, brilliant minds in the field have made many improvements. Outside of game playing, reinforcement learning was also instrumental in robot obstacle avoidance models. In 1999, Zhang Rubo et al. realized robot obstacle avoidance by learning skillfully using reinforcement

learning. [24] With further advancement in machine learning that introduces Deep Learning, which will be briefly discussed in the next section, reinforcement learning can be combined with Deep Learning to form Deep Reinforcement Learning. This unique learning algorithm can be utilized to overcome challenges that cannot be tackled by either Reinforcement Learning or Deep Learning alone. [25]

1.2.5 Deep Learning

Deep Learning is a subset of machine learning. Deep is a technical term that refers to the number of layers in a neural network. The multiple hidden layers allow deep neural networks to learn features of the data in a feature hierarchy. [26] Simple features combine from one layer to the next to form more complex features. Therefore, Deep Learning generally works exceptionally well on tasks that involve recognizing unstructured data, such as pictures or text, without explicitly telling it the features list.

The standard machine learning breaks problems down into parts and solves them individually. Deep learning solves the problem from end to end. In a straightforward example, [27] Machine learning can distinguish between a square and a triangle based on human information, such as that a square has four edges/points and a triangle has three edges/points. With deep learning, the program does not start with the given feature information. The algorithm determines how many edges/lines the shapes have and specifies the requirements for a square and triangle.

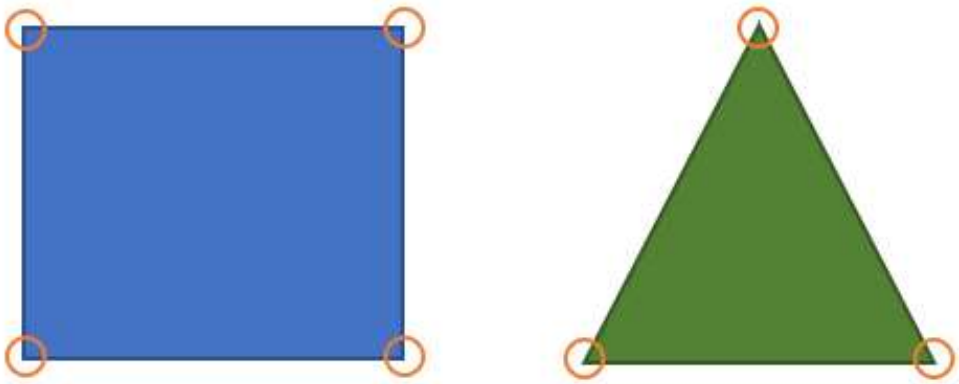

Figure 1.4: Square and triangle shape recognition

The main difference between deep learning and other machine learning methods is that the deep learning algorithm does not perform well when data is small. A deep learning algorithm needs a lot of data to understand it perfectly. Deep learning performance increases with the amount of data and does not plateau as early as other machine learning algorithms. The relationships between the amount of data and performance are shown in Figure 1.5. [28]

Deep learning algorithms depend on high-end machines that can perform billions of instructions per second, while machine learning algorithms can run on relatively simple machines with CPUs (Central Processing Units). High-end machines include GPUs (Graphics Processing Units) specifically built for deep learning. CPUs usually consist of four (4) to eight (8) cores, while the GPUs have 100s or 1000s of cores using parallel processing and break jobs into separate tasks to process simultaneously. Therefore, GPUs can have much higher throughput. [29]

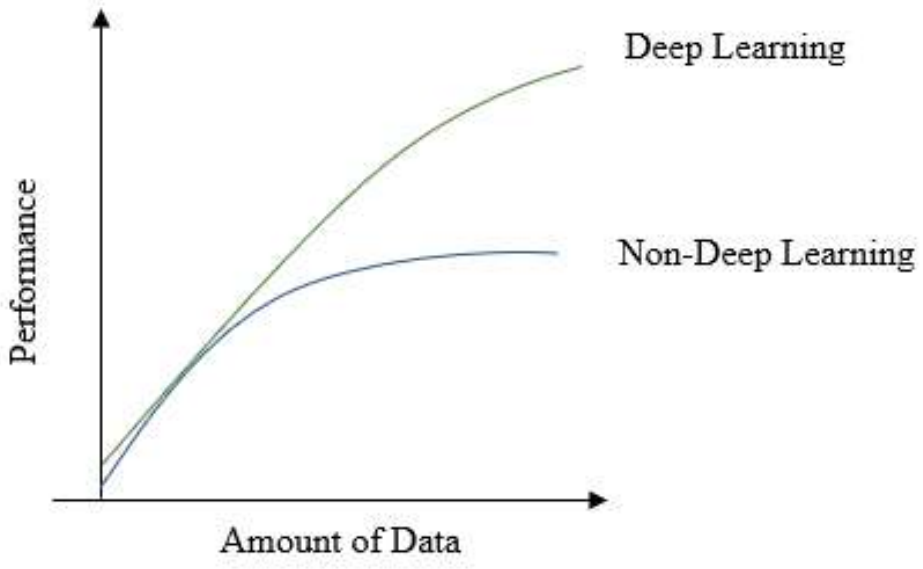

Figure 1.5: Deep vs Traditional Machine Learning

A typical machine learning approach to identifying an object in an image breaks the problem down into object detection and recognition. It first uses an object recognition algorithm like GrabCut to find all the possible objects. Then it uses another algorithm, for example, SVM with HOG (Histogram of Oriented Gradients), to recognize the relevant object. [30]

The deep learning approach applies to the end-to-end process. For example, YOLO net processes the image to detect and recognize the object. Deep learning takes a long time to train due to its size and usually has many layers. Some deep learning takes two weeks to train, while some machine learning takes a few seconds to a few hours. Deep learning is much less interpretable due to its black box nature. Not many clues were

given due to its end-to-end training nature. On the other hand, machine learning algorithms like decision trees and linear/logistic regression can trace the decision back to the feature definition and how it concludes. The difference can be summarized in the table below: [31]

	Machine Learning (standard)	Deep Learning
Training dataset	Small	Large
Defined features	Yes	No
Classifiers availability	Many	Few
Time to Train	Short	Long

Table 1.2: Machine Learning and Deep Learning

One example of deep learning is AlexNet, one of the first to make a breakthrough in deep learning with convolutional neural networks (CNN). AlexNet is a Deep CNN for image classification that won the ILSVRC-2012 competition and achieved a winning test error rate of 15.3%. The next best result in the competition was 26.2%. AlexNet has eight layers; the first five are convolutional, and the last three are fully connected. [32]

Deep Learning was also used in end-to-end learning for self-driving cars, [33] a paper published by NVIDIA in 2016 aimed to demonstrate work done to enhance self-sufficient autonomous land vehicles. The paper shows success in 72 hours of driving data using three cameras and steering angle as inputs and CNN (Convolution Neural Network)

as the training algorithm. The trained network generated accurate steering commands from a single front-facing center camera.

In another example, RNNs (Recurrent Neural Networks) were used for the joint estimation and tracking of facial features. This is a departure from traditional tracker engineering, which uses Bayesian Filtering. Instead, RNNs learn directly from a large amount of training data. This end-to-end method performs best in head pose recognition and landmark localization. [34]

CHAPTER TWO

PROBLEM DESCRIPTION

2.1 Motivation for Research

Machine learning in the automotive industry has made vast progress since Dean A. Pomerleau's 1989 paper on ALVINN (An Autonomous Land Vehicle in a Neural Network). [6] Since then, many companies, such as Google, Tesla, and Baidu, have attempted to research and implement autonomous cars. Tesla cars are the most recognized for being able to implement up to level 2 of automation in cars. This means the driver is still responsible for the vehicle's actions and must be alert and ready to take over at any time. To further improve safety, transportation efficiency, and avoid traffic accidents, level 3 – eyes off, level 4 – mind off, and ultimately level 5 – steering wheel optional are required to achieve full autonomy. [7]

Machine learning has played an increasing role in larger agricultural machines with the emerging concept of smart farming that makes agriculture more efficient and effective with the help of high-precision algorithms. [8] The trend is expected to propagate to the smaller agricultural machines that target part-time farmers or hobby tractor users. This is most expected, especially in the automation and autonomy space in research and development. In a paper published in 2015 by M. Song et al., path planning for an autonomous lawn mower tractor was proposed. In this paper, a lawn tractor was fitted with a GPS unit, an inertia motion unit, a gyro compass, a wheel angle sensor, and a mower on/off sensor, and all data were collected at a 10-Hz rate. Although no machine learning was applied in the research, a decrease of about 13% in travel distance and 30%

of working time was achieved. [9] Since then, many advances in agricultural machinery navigation technology have been made with the advent and more widespread use of machine learning. A survey paper published by Z. Yao et al. in Feb 2024 reviewed, compared, and analyzed many previous studies on automatic navigation, path planning, control systems, and communication modules. This paper proposed twenty-two (22) available algorithms according to different application scenarios and summarizes the challenges and difficulty in using machine learning, as well as where other types of machine learning could be more suited to be employed. [10] In another research on agricultural machines by Wei Lu et al. in 2020, brain EEG signals were collected from a brain-computer interface (BCI), followed by denoising and input into Recurrent Neural Network (RNN) training. After the training, the systems allowed the tractor to go straight, brake, turn left, and turn right by reading the brain's EEG signal. This research paves ways for operators with disability to operate the machine comfortably. Field operations such as tillage, harvesting, weeding, and land preparation for farming and construction could be the targeted uses. All vehicle operators and drivers with disability account for about one-fifth or 20% of the total US population. [11] With proper focus, research, and development, this could evolve into a lucrative business for companies and meaningful solutions for many people who are in need.

Applications such as tractor features that usually contain inputs and labeled outputs are most suitable for using supervised machine learning, such as artificial neural networks (ANN), as a training algorithm. This dissertation will focus on one of the ANN

algorithms, the Feedforward Neural Network (FNN), with an improvement detailed in Chapter Three to train the application.

The output of an FNN is continuous, so it is categorized as a regression or function approximation model. For many agricultural applications, a discrete or digitized output is required to activate a particular function, such as raising and lowering a rear implement in a tractor ground leveling application. The main contribution of this research is to discover that in a noisy system generated by human delay and errors, a Threshold-Feedforward Neural Network (TFNN) effectively generates classification outputs that are superior to other commonly used classification models.

2.2 Supervised Machine Learning's Limitation

The supervised machine learning algorithm is suitable for training data with known inputs and clear separation outputs. In many applications where no human operators are involved and limited system noises are present, traditional supervised machine learning methods such as FNN, SVM, NB, and LVQNN can effectively predict the outputs given known inputs and outputs training data. In addition to system noise, human-induced noise due to error or response delay could negatively impact the training data used for supervised machine learning. When the labeled output data are often mixed in with systems and human errors, supervised machine learning can be problematic in filtering out the noise, producing undesirable outputs.

Various classification-type supervised machine learning methods are also explored and found to have room for improvement. Therefore, regression-supervised machine learning, such as FNN, is explored to take advantage of the fuzzy nature of

continuous output. My research taught me that optimally thresholding the regression supervised learning outputs can produce optimal classification outputs. This is my main theoretical contribution. TFNN is a combination of Function Approximation/Regression and Classification. Thresholds are applied to convert the FNN continuous outputs into discrete outputs for the discrete nature of applications such as On-Off, Up-Hold-Down, Forward-Neutral-Reverse, etc.

2.3 Automated Tractor Ground Leveling - Application

Traditional ground leveling with no electronic assistance requires constant control and adjustment of leveler implement height and simultaneously looking behind the tractor to gauge the quality of leveling work. The operation leads to neck, hand, and arm fatigue. According to some research, in the U.S. market, about 50 percent of the tractor industry is under 20 hp, and 70 percent of those belong to first-time tractor owners. [12] Therefore, automated features such as tractor ground leveling are great applications for inexperienced users who want to do a good job without spending extended time acclimating to the operation.

Figure 2.1: Tractor Ground Leveling Illustration [35]

Figure 2.2: Tractor Ground Leveling Illustration [36]

To address the shortcoming mentioned above, an FNN-trained electronics system learns the parameters required to level ground by being exposed to training data. The trained system automates leveler height control to improve user comfort and quality of work.

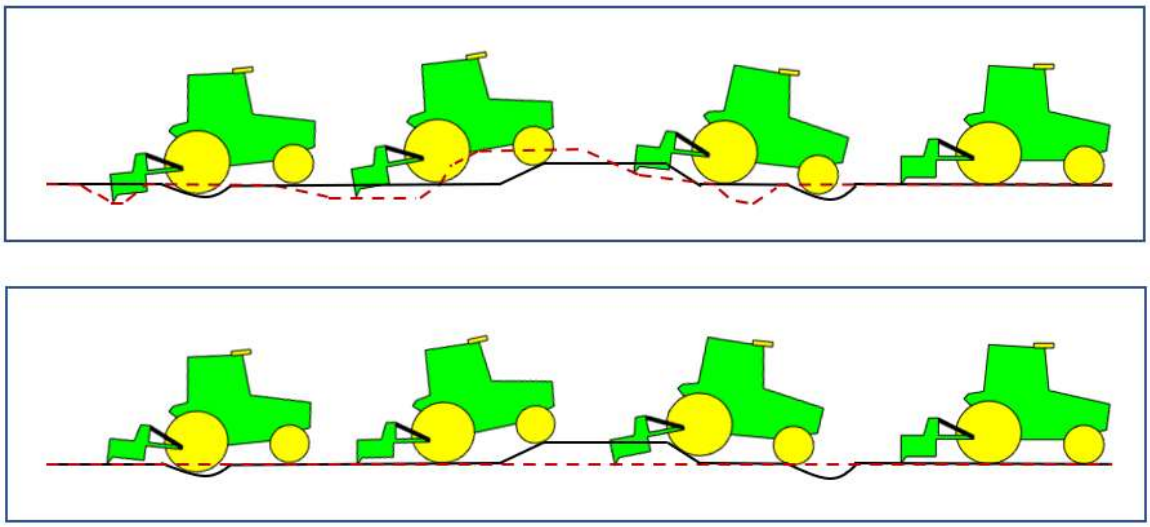

Figure 2.3: Top Picture - no AGL, Bottom Picture - with AGL

Tractor automated ground Leveling (TAGL) aims to increase operator comfort and improve the consistent quality of ground leveling. This dissertation details an improved machine learning approach termed the Threshold-Feedforward Neural Network (TFNN). Traditional FNN is a continuous function output resulting from a continuous input. The TFNN effectively produces classification outputs that include the tractor leveler implement raise, neutral, and lower commands. The proposed TFNN used tractor inclined angle and leveling error as inputs.

2.4 Other applications

The applications for TFNN need not be limited to TAGL. All applications with specific internal and external system disturbances or human errors can benefit from TFNN. Further information, examples, and theory can be found in Chapter Three. In a high level, all outputs that have overlapped data due to noise, non-preciseness, systems, or human errors or mistakes can potentially benefit from TFNN. The key to getting good outputs or results is to apply the optimized thresholds to arrive at the best solution. Therefore, one can use the proposed TFNN theory to find the best solution that contains slightly different optimization parameters depending on the specific applications and situation.

2.5 Literature review

There are other algorithms like TFNN, i.e., the logistic regression model. The logistic regression model converts the regression model into a classification model via the probability of an event occurring or not occurring. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied to the odds. For binary classification, a probability less than .5 will predict 0, while a probability greater than 0.5 will predict 1. [37]

The TFNN proposed in this dissertation converts a regression FNN into a TFNN by setting output thresholds and converting the continuous output into classes or levels. An exhaustive binary search is performed to find the optimum threshold that maximizes performance and minimizes errors. The search algorithm could also involve other

parameters that impact the overall performance, such as the electronics' durability and functionality. The number of output thresholds is exactly $q-1$, with q being the number of output classes. Because the thresholds are real numbers, finding the optimal threshold values is essential to achieving the optimum results. Another search algorithm, such as gradient descent or the Levenberg-Marquardt Algorithm (LMA), can also be used to find the optimum thresholds. Chapter three will discuss each of the three methods in more detail.

CHAPTER THREE

FORMULATION AND TECHNIQUE OF TFNN

3.1 Motivation

Data collected in the real world is not always clean and straightforward. It often contains internal and external disturbances, delay, and human error. Therefore, machine learning that uses flawed data can yield erroneous results. This section details the basis and theory for TFNN, which improves the accuracy of training results using otherwise good data that contain noise. This is the main contribution of this dissertation.

3.2 Observation, Decision Function, Action, and Output

A typical electromechanical machinery system containing inputs, outputs, and feedback control loops generally has operating parameters that include observation, decision function, action, and outputs. The observation(**e**)–decision(f)–action(u)–output(h) and their relationships will be mathematically represented and explained in the following few paragraphs. The practical manual operation or real-life system block diagram is shown below:

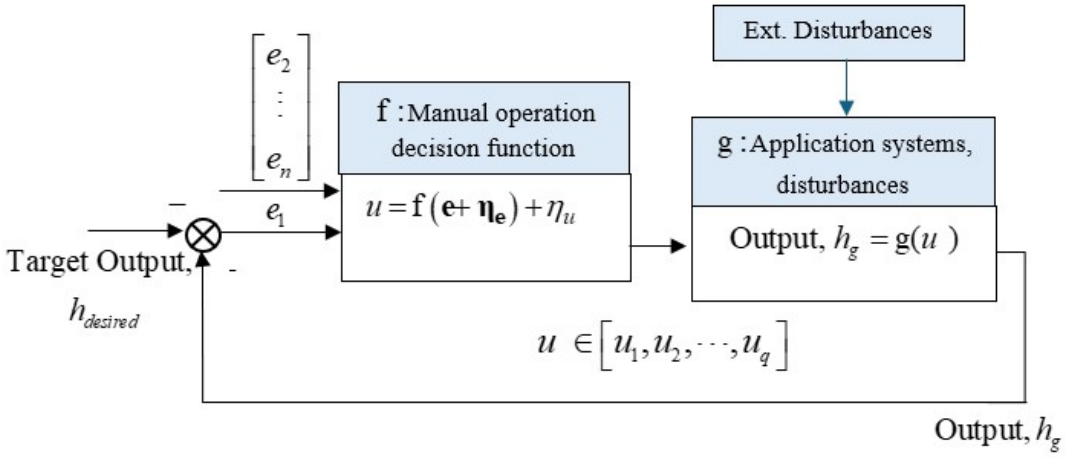

Figure 3.1: Manual Operation System Block Diagram

The observation values, **e** can be a vector of real numbers, at a minimum with a single observation variable, $e_1 = h_g - h_{desired} = E$, where h_g is the output and $h_{desired}$ is the target or desired output of the systems. For the remainder of this dissertation, e_1 will often be referred to as $e_1 = E$ the error of the systems, which is the first parameter of observation vector **e**.

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \in \mathbb{R}^n \tag{3. 1}$$

Secondly, the decision function f is a general mapping represented by:

$$\mathrm{f}:\mathbb{R}^n \to \mathbb{Q}^1 \subset \mathbb{R}^1 \qquad \mathbb{Q}^1 = \left\{u_1, u_2, \cdots, u_q\right\} \tag{3.2}$$

In this case, decision f to determine the action u is assumed to be unknown and nonlinear; action u is observable, estimable, or constructible from observation $\mathbf{e}$. The action variable u is denoted below:

$$u = \mathrm{f}\left(\mathbf{e}+\boldsymbol{\eta}_{\mathbf{e}}\right) + \eta_u \tag{3.3}$$

For the case considered in this dissertation, the action value, u will be a scalar in the form of quantized real numbers, $u_1,\ u_2, \cdots, u_q$. That is

$$u = u_1 \text{ or } u_2 \text{ or } \cdots \text{ or } u_q \tag{3.4}$$

$$u \in \mathbb{Q}^1 = \left\{u_1, u_2, \cdots, u_q\right\} \tag{3.5}$$

In practical applications, the action variable u is expected to be influenced by disturbances, non-preciseness, systems, or human errors or mistakes. The action(u) variable is mainly affected by the decision variable, $\mathrm{f}\left(\mathbf{e}+\boldsymbol{\eta}_{\mathbf{e}}\right)$. Where $\mathbf{e}$ is the observation variable, and $\boldsymbol{\eta}_{\mathbf{e}}$, the systems-generated error is non-human related. The observation variable $\mathbf{e}$ can be multi-variable inputs to the decision function f. Apart

from the decision function f, there are also human errors denoted by η_u, i.e., all errors that are attributed to human deficiencies and fallacies.

Finally, h_g the system output variable is the output of $\mathrm{g}(u)$, or represented by $h_g = \mathrm{g}(u)$, where g is the output plant model or the real-world application is designed to process the action u to achieve or get as close as possible to the desirable output $h_{desired}$. The result h_g is compared to the target value $h_{desired}$ to generate $e_1 = E$.

The formulation of the TFNN for reconstructing u from $\mathbf{e}$ is described in the following sections by first training an FNN and then threshold optimization to arrive at TFNN.

3.3 Main Theory Contribution

The contribution of this thesis can be summarized in the system block diagram below. The forward transfer loop consists of a feedforward neural network, a threshold activation function, and an application system with external and internal disturbances. The system block diagram below summarizes the system flow and interactions, including the machine learning training FNN, which produces an approximation function output that acts as the decision function, and TFNN improvements, which come with the optimized thresholds.

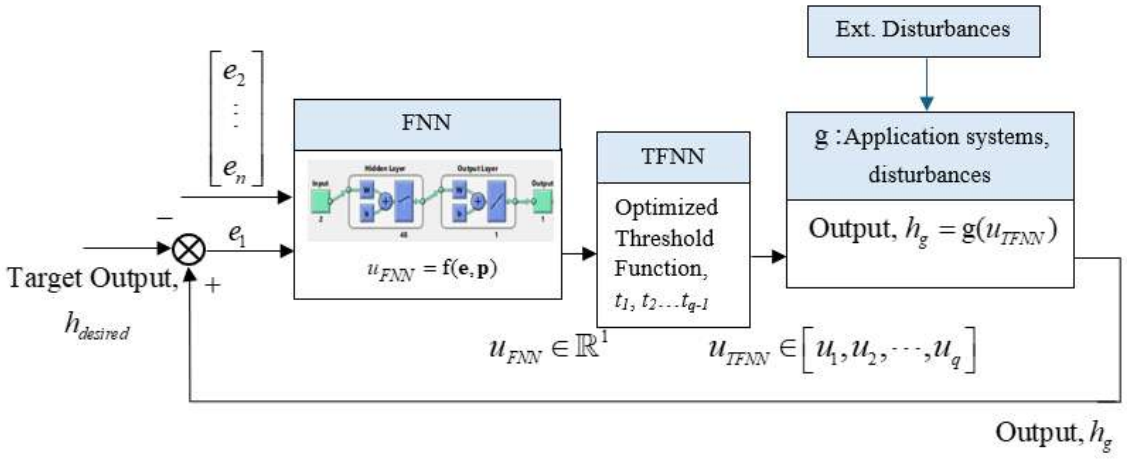

Figure 3.2: TFNN System Block Diagram

3.4 Feedforward Neural Network (FNN).

First, we consider the basis for a two-layer FNN represented by:

$$\begin{aligned} \mathbf{s_1} &= \mathbf{w_1 e} + \mathbf{b}_1 \\ \mathbf{y_1} &= \mathrm{f}_1(\mathbf{s}_1) \\ u_{FNN} &= \mathbf{w_2 y_1} + b2 \end{aligned} \tag{3.6}$$

where

$\mathbf{e} \in \mathbb{R}^{n\times 1}$		is the network input
$\mathbf{w_1} \in \mathbb{R}^{n_1 \times n}$	$\mathbf{b_1} \in \mathbb{R}^{n_1 \times 1}$	are the 1st layer weights and biases
$\mathbf{w_2} \in \mathbb{R}^{1 \times n_1}$	$b_2 \in \mathbb{R}^1$	are the 2nd layer weights and biases
$\mathbf{s_1}, \mathbf{y_1} \in \mathbb{R}^{n_1}$	$\mathrm{f}_1 : \mathbb{R}^{n_1} \to \mathbb{R}^{n_1}$	are the 1st layer variables and activation function
$u_{FNN} \in \mathbb{R}^1$		continuous output of FNN
$u_{TFNN} \in \mathbb{R}^1$		threshold quantized class output of the FNN

We can abbreviate the FNN output as

$$u_{FNN} = u_{FNN}(\mathbf{e}, \mathbf{p}) \tag{3.7}$$

where the parameters are lumped as

$$\mathbf{p} = [\mathbf{w_1}, \mathbf{b_1}, \mathrm{f}_1, \mathbf{w_2}, b_2] \tag{3.8}$$

The FNN outputs are dependent on the parameters shown above, i.e., $u_{FNN} = u_{FNN}(\mathbf{e}, \mathbf{p})$. FNN is to be trained in such a way that u_{FNN} approximates u.

To assess the performance of the u_{FNN}, mean square errors of trained output were calculated per the formula below:

$$MSE_{FNN} = \frac{1}{n}\sum_{i=1}^{n}\left(u(\mathbf{e}) - u(\mathbf{e}, \mathbf{p})\right)^2 \tag{3.9}$$

The MATLAB function newff in the Deep Learning Toolbox yields optimal parameters $\boldsymbol{p}^*$ that minimizes the MSE_{FNN} shown below:

$$\min_{\mathbf{p}} \; MSE_{FNN} \tag{3.10}$$

The optimally trained FNN is denoted as

$$u^{*}_{FNN}(\mathbf{e}) = u_{FNN}(\mathbf{e}, \mathbf{p}^{*}) \tag{3.11}$$

With FNN optimally trained, the following section explains the basis for thresholding the FNN output, thereby improving the training performance in terms of Root Mean Square (RMS) Error.

3.5 Threshold Feedforward Neural Network (TFNN)

For the case herein, we consider $t_1,\ t_2, \cdots, t_{q-1}$, arrange the TFNN to yield a quantized level action such as

$$u_{TFNN}\left(\mathbf{e}, t_1, t_2, \cdots, t_{q-1}\right) = \begin{cases} u_1 & \text{if } u^{*}_{FNN}(\mathbf{e}) > t_1 \\ u_2 & \text{if } u^{*}_{FNN}(\mathbf{e}) \le t_1 \ \ \& \ \ u^{*}_{FNN}(e) \ge t_2 \\ \vdots & \vdots \\ u_q & \text{if } u^{*}_{FNN}(\mathbf{e}) < t_{q-1} \end{cases} \tag{3.12}$$

Note that $u_{TFNN} \in \left[u_1, u_2, \cdots, u_q\right] \subset \mathbb{R}^1$. The unique number of thresholds, *t*, will always be one less than the sum of action, u, which we define as *q*. To optimize the performance of TFNN, we define the root mean square error measurement of $e_1 = E = h_g - h_{desired}$ shown below, where $h_{desired}$ is the target value and h_g is the output value.

$$\min_{T}\ RMS_{TFNN} = \min_{T} \sqrt{\frac{1}{m}\sum_{i=1}^{m} E_i^2} \tag{3.13}$$

We want to find the optimum thresholds, $t_1^*, t_2^*, \cdots, t_{q-1}^*$, that minimizes

$$\min_{t_1, t_2, \cdots, t_{q-1}} RMS_{TFNN} \tag{3.14}$$

The TFNN consists of the FNN* and $t_1^*, t_2^*, \cdots, t_{q-1}^*$ gives rise to the equation:

$$u_{TFNN}\left(e, t_1^*, t_2^*, \cdots, t_{q-1}^*\right) = \begin{cases} u_1 & \text{if } u_{FNN}^*(e) > t_1^* \\ u_2 & \text{if } u_{FNN}^*(e) \le t_1^* \ \ \& \ \ y_{FNN}^*(e) \ge t_2^* \\ \vdots & \vdots \\ u_q & \text{if } u_{FNN}^*(e) < t_{q-1}^* \end{cases} \tag{3.15}$$

3.6 Optimization Methods

Optimization can be done for the thresholds using exhaustive search, gradient search, or Levenberg–Marquardt algorithm (LMA) methods. Their brief descriptions are in the following few sections. For this dissertation, we employed an exhaustive binary search and some systems-level considerations for electronics durability and feature stability, which will be discussed in the following chapters.

3.6.1 Exhaustive Search

An exhaustive search is the most straightforward search algorithm for the best solution. It starts with creating all possible subsets of available inputs and the outputs

evaluated to arrive at the best one. If the response surface has multiple peaks, an exhaustive search will evaluate all of them and choose the best one. [38] It is also a brute-force approach to evaluate every possible combination and permutation. [39] This brute-force algorithm search is simple to implement and will always find a solution if it exists. It is typically used when the problem size is limited. Its main shortcoming is that the number of combinations to be evaluated can become very large. Therefore, it is only used when simplicity of implementation is more important than speed. [40]

3.6.2 Gradient Descent

Gradient descent is a first-order iterative algorithm, a linear optimization that works by minimizing the cost function of a model, which is quantified by the difference between the predicted and actual outputs. This optimization algorithm is widely used and known for its ability to handle models with many parameters, such as machine learning tasks. Especially when dealing with complex non-linear data. A well-trained model with a low-cost function will have predicted output values close to the actual output values. [41] This idea takes repeated or iterative steps in the opposite direction of the function's gradient at the current point in search of a local minimum. Conversely, stepping in the direction of the gradient will lead to a local maximum, which is known as Gradient ascent. The challenges for Gradient descent are selecting an appropriate learning rate so that the computation does not take too long because the learning rate is too small. If the learning rate is too large, the algorithm may conclude sooner but run the risk of overshooting the intended target or having a large cost function. This algorithm may not

work well if the cost function has many local minima, for this algorithm does not guarantee the result will be the global minimum. [42]

3.6.3 Levenberg-Marquardt Algorithm (LMA)

The Levenberg-Marquardt algorithm is a nonlinear optimization method that applies an iterative process to converge to a local minimum. This method blends the steepest descent method and the Gauss-Newton algorithm. When the current solution is far from the correct one, it behaves like a steepest descent, which is slow but guaranteed to converge. When the current solution is close to the correct solution, it becomes a Gauss-Newton method. [43][44] Similar to gradient descent, this method, like other iterative optimization algorithms, finds only a local minimum, which is not necessarily the global minimum. [45]

3.7 Practical Application Illustration

3.7.1 Tractor Automated Ground Leveling (TAGL)

In the case of TAGL, the manual operation systems block diagram and interaction are presented below:

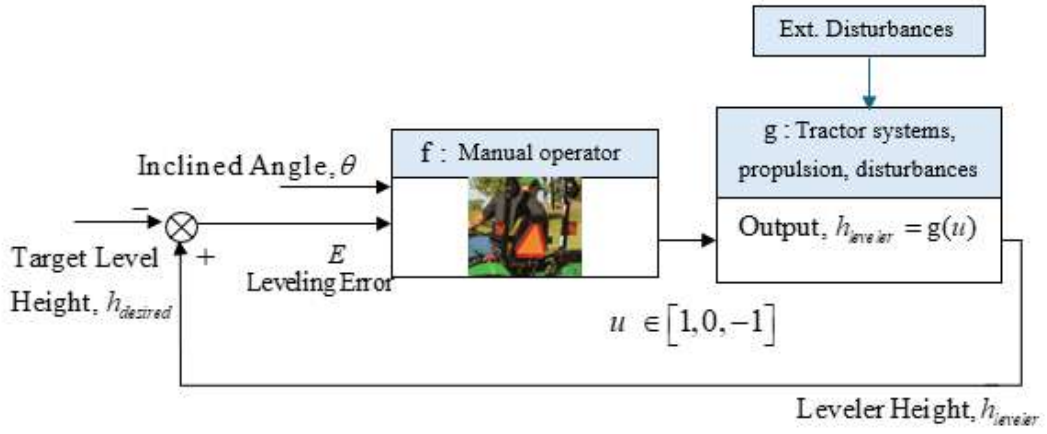

Figure 3.3: TAGL Manual Operation System Block Diagram

A single variable $E = h_{leveler} - h_{desired}$ is a calculated error value, which is the difference between the desired level height $h_{desired}$, and the output $h_g = h_{leveler} = g(u)$, where g in this case is the tractor systems that include the controller, switches, and electro-hydraulic systems that process the action u. The tractor system also inherently has propulsion and disturbances that affect the leveler's height and accuracy. The observation vector $\mathbf{e}$ for this illustration is:

$$\mathbf{e} = \begin{bmatrix} E \\ \theta \end{bmatrix} \in \mathbb{R}^2 \qquad (3.16)$$

$u \in [1, 0, -1]$,where, 1: Leveler Raise, 0: Neutral, -1: Leveler Lower

For the case herein, we consider t_1 & t_2, $T = \{t_1, t_2\}$, the upper and lower thresholds, and arrange the TFNN to yield a three-quantized level action such as

$$u_{TFNN}(e,t_1,t_2)=\begin{cases}1 & \text{if } u^*_{FNN}(e) > t_1 \\ 0 & \text{if } u^*_{FNN}(e) \le t_1 \ \ \& \ \ u^*_{FNN}(e) \ge t_2 \\ -1 & \text{if } u^*_{FNN}(e) < t_2\end{cases} \tag{3.17}$$

The data flow diagram for TFNN is formulated for TAGL application. The simulated or real-life collected data are fed into the FNN training MATLAB program to generate FNN, and exhaustive search and systems considerations are performed to arrive at the optimum threshold shown below:

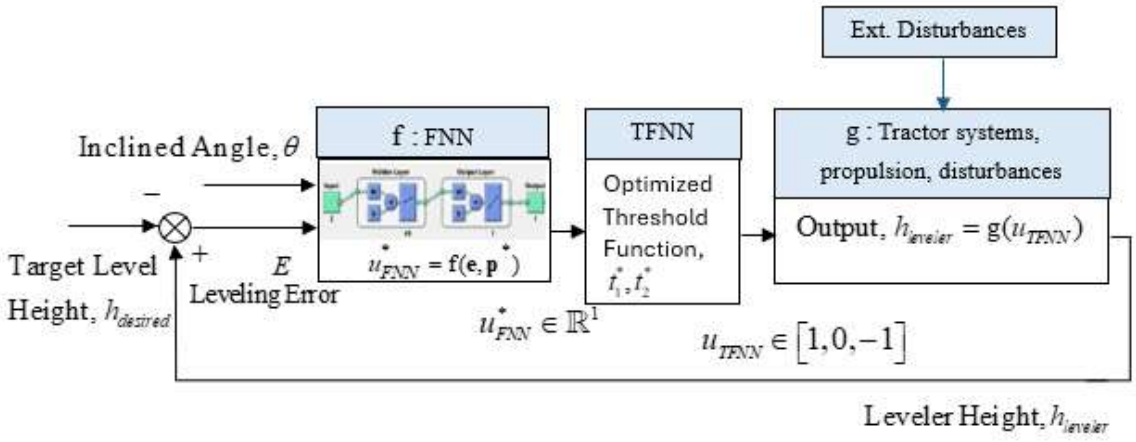

Figure 3.4: TAGL Data Flow Diagram for TFNN

Note that $u_{TFNN} \in [1, 0, -1]$, where 1: Leveler Raise, 0: Neutral, -1: Leveler Lower. The output of the TAGL application is $h_g = h_{leveler} = g(u)$. The performance measurement for TAGL is calculated using the difference between the leveler height and the target

level height of the leveling scraper, which is labeled E in Figure 3.5 below. Root mean square (RMS) error of E is tracked and minimized to maximize the performance.

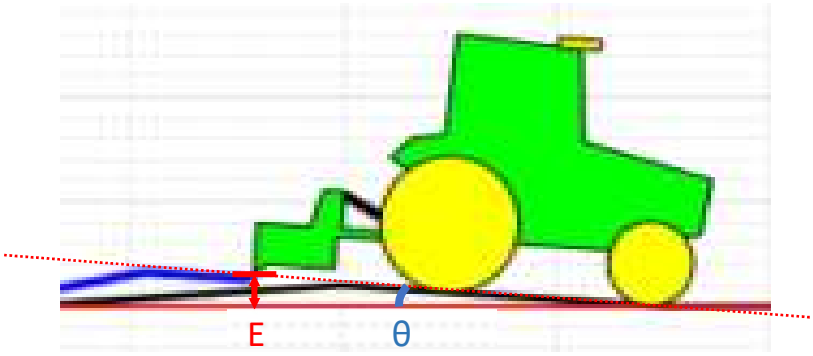

Figure 3.5: Tractor Ground Leveling Error

Define the RMS error measurement for TFNN:

$$RMS_{TFNN} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} E_i^2} \tag{3.18}$$

We want to find the optimum thresholds, t_1^* & t_2^*, that minimizes RMS_{TFNN}. In mathematical form, the equations are defined below:

$$\min_T RMS_{TFNN} = \min_T \sqrt{\frac{1}{m}\sum_{i=1}^{m} E_i^2} \tag{3.19}$$

$$u_{TFNN}^* = u_{TFNN}\left(e, T^*\right), \qquad T^* = \text{optimized parameters} \tag{3.20}$$

In this case u_{TFNN} is optimized by optimizing t_1 & t_2, where $T = \{t_1, t_2\}$, and $T^* = \{t_1^*, t_2^*\}$.

The plant model above is realized in the next section to collect $\mathbf{e}$ & u. The raw data are collected as $N \times 1$ vectors, $\mathbf{e}_{\mathbf{data}}$ & u_{data} .

CHAPTER FOUR

TRACTOR GROUND LEVELING APPLICATION

4.1 Tractor Ground Leveling – Virtual data gathering

Traditional ground leveling with no electronic assistance requires constant rear implement height control adjustment and simultaneous looking behind the tractor to gauge the quality of leveling work. This operation leads to neck, hand, and arm fatigue. To address the shortcoming mentioned above, an FNN-trained electronics system learns the parameters required to level ground by being exposed to training data. The trained system automates leveler height control to improve user comfort and quality of work.

4.1.1 Simulation Block Diagram

A two-input and one-output system was defined with ground leveling error and inclined angle as the inputs and the rear implement raise/neutral/lower as the output. The block diagram can be seen in Figure 4.1.

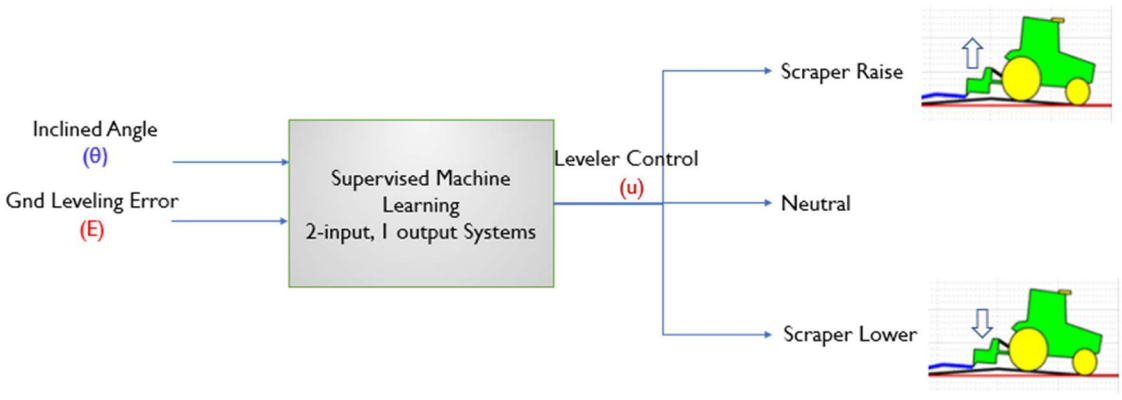

Figure 4.1: AGL Simulation Block Diagram

Because the training data outlined above has fully defined labels of inputs and outputs, supervised machine learning is the most suitable algorithm to analyze the training data and generate a complete function that can be utilized to process inputs that contain ground-level error and inclined angle. [46]

4.1.2 Simulation equation of AGL

The data-gathering simulation is set up with a tractor vehicle figure with points **A**, **B**, **C**, **D**, and **P** shown in Figure 4.2. Point **P** with coordinate $\mathbf{P} = [P_x, P_y]$ and the tractor pitch inclined angle, θ, can be obtained with a GPS receiver device readily available in Agricultural OEM aftermarket, e.g., John Deere StarFire 6000 model used in this research.

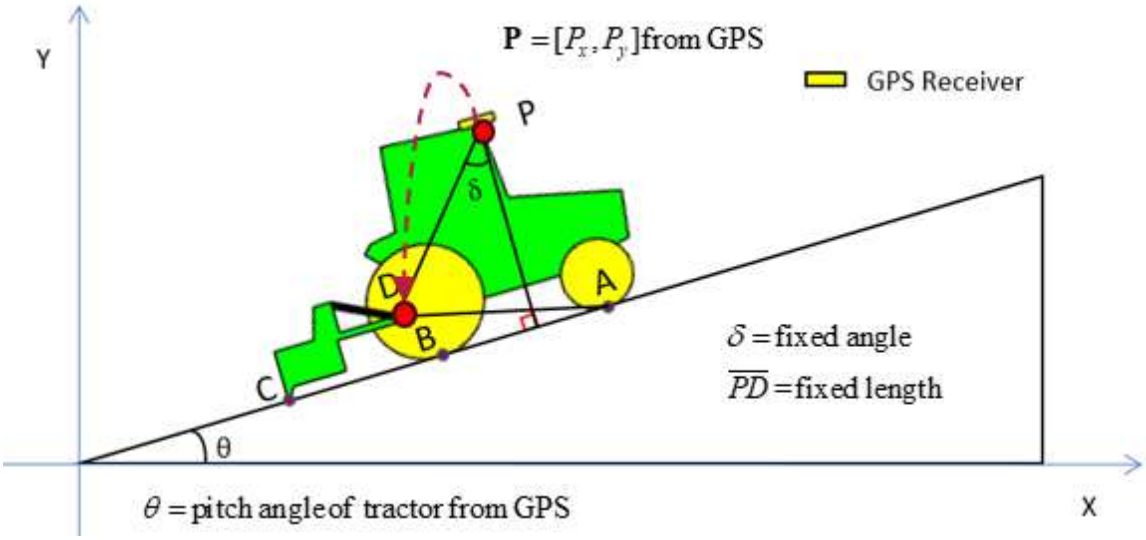

Figure 4.2: Tractor and Leveler Coordinate References

The rear leveler implement is pivoted at point **D**, representing the fixed point in the tractor's three-point hitch that moves point **C**'s tip up or down. Point coordinate **D** can be derived from the following equations using the length $\overline{PD}$, and angles δ and θ shown in Figure 4.2. δ is the angle measured from point **P** to between point **D** and the point perpendicular to the tractor's instantaneous ground plane. θ is the pitch angle in a real-life tractor implementation, which the information can be obtained from the GPS receiver. The mathematical equation for obtaining **D** is shown in the equation below:

$$\mathbf{D} = [D_x, D_y] = [(P_x - \overline{PD}.\sin(\delta - \theta)), (P_y - \overline{PD}.\cos(\delta - \theta))] \tag{4.21}$$

The variable position of the leveler tip is represented by **C** in Figure 4.3 below and is affected by the tractor's inclined angle θ and angle β. β is the angle of the leveler relative to the tractor controlled by the operator or computer, with the raise or lower command measured by a position sensor. When no command is given, the leveler stays in a neutral position with β value unchanged.

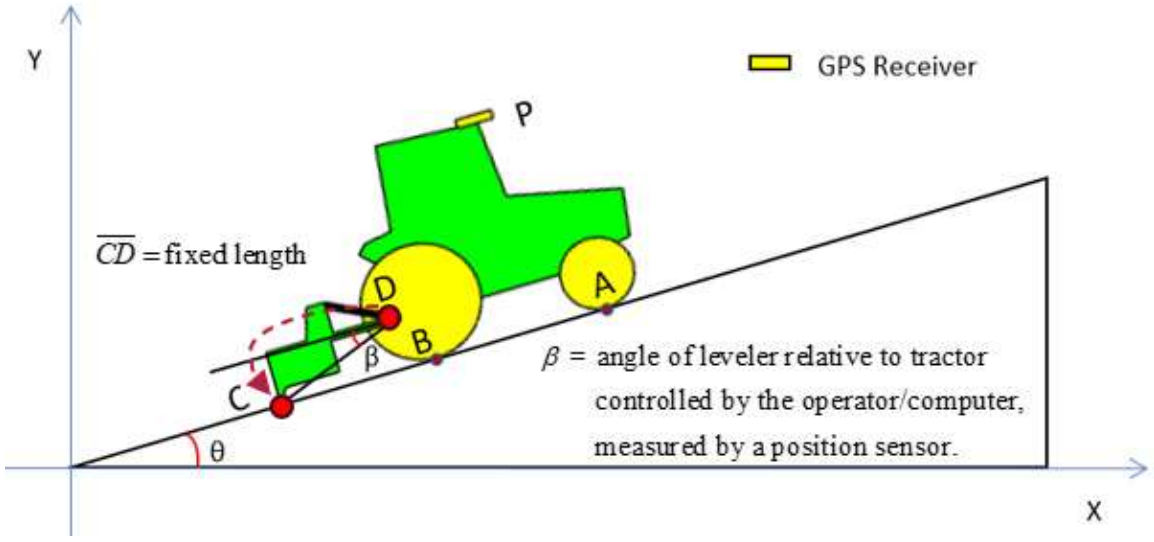

Figure 4.3: Leveler Tip **C** Pivoted at Point **D**

In the AGL, output *u* will be motorized, electro-hydraulic controlled by the TFNN outputs. The equation for **C** is shown below:

$$\mathbf{C} = [C_x, C_y] = \mathbf{D} - [(\overline{CD}.\cos(\theta + \beta)), (\overline{CD}.\sin(\theta + \beta))] \tag{4.22}$$

The ground leveling error input *E* is derived from comparing the y component of point **C** i.e., C_y with the y component of ground reference line point $\mathbf{R} = [R_x, R_y]$ i.e. R_y shown equation below.

$$\text{Leveling Error, } E = C_y - R_y = h_{leveler} - h_{desired}$$
$$C_y = h_{leveler} \tag{4.23}$$
$$R_y = h_{desired}$$

4.1.3 MATLAB simulation and calibration procedure

A MATLAB program and Simulink routine were written to realize the TFNN theory in a Tractor Automated Ground Leveling (TAGL) application. The program block diagram is shown on the next page in Figure 4.4. During the data collection process, a human-in-the-loop method is employed. A person can move the leveler up by pressing the keyboard up button or down by pressing the keyboard down button. If no leveler movement is desired, the keyboard button is left untouched or in a neutral position.

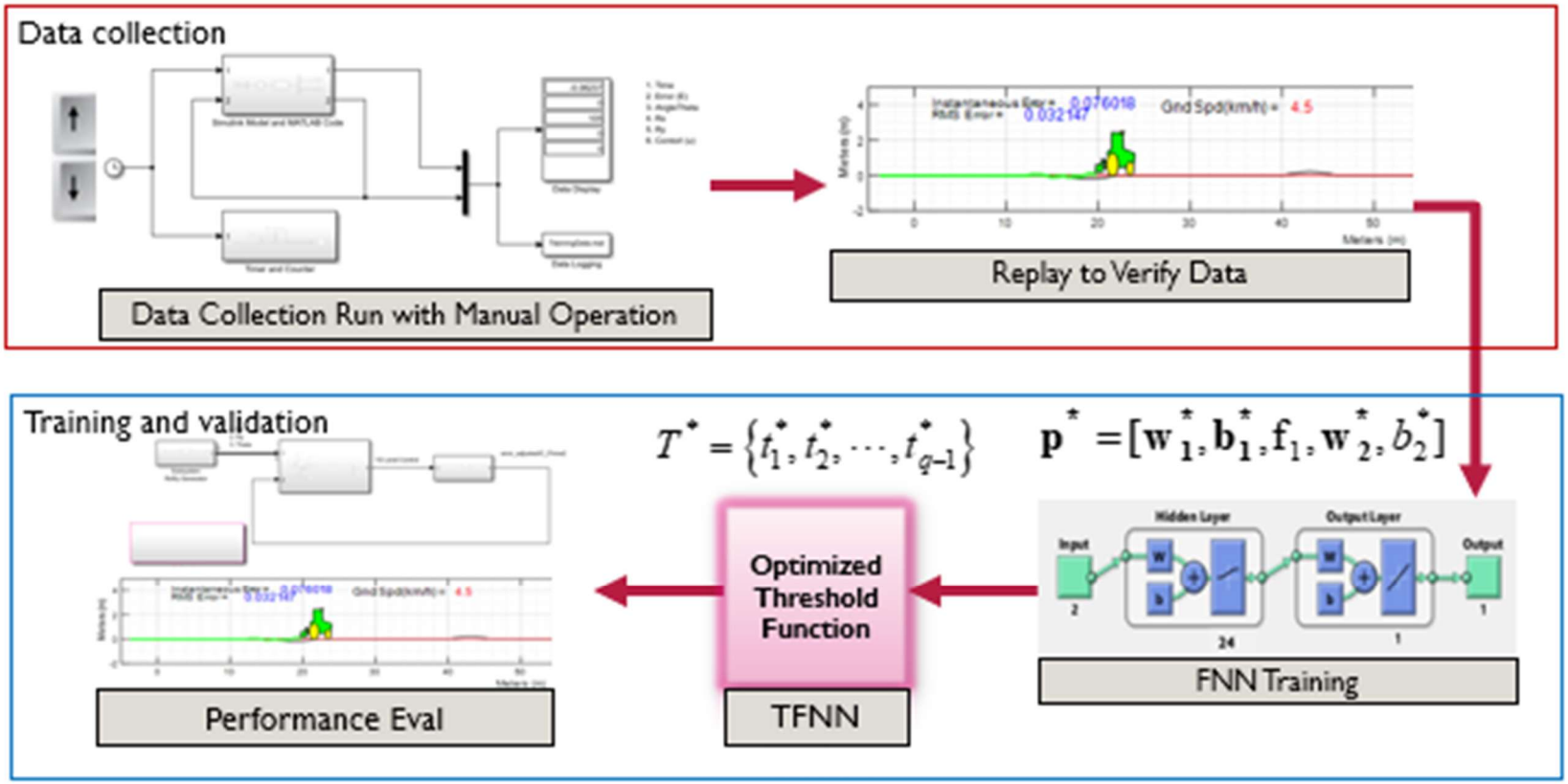

Figure 4.4: TAGL Modeling and Simulation Block Diagram

The running terrain was set up with a length that totals 105 meters per Figure 4.6. The instantaneous leveling error and RMS error are tracked during the simulation. Throughout this dissertation, the vehicle's inclined angle, represented by theta, θ, can be acquired by the John Deere GPS receiver StarFire 6000 unit on a tractor in practical application. The ground reference line, R, can be obtained using the calibration techniques outlined below and shown in Figure 4.5. To generate the ground reference line, R, two points each at the starting and ending points of the intended work field are needed. John Deere GPS receiver StarFire 6000 can be mounted on the cabin. The two points then can be used along with the tractor's inclined pitch angle θ and the leveler angle β to generate the ground reference line (highlighted in red in Figure 4.5).

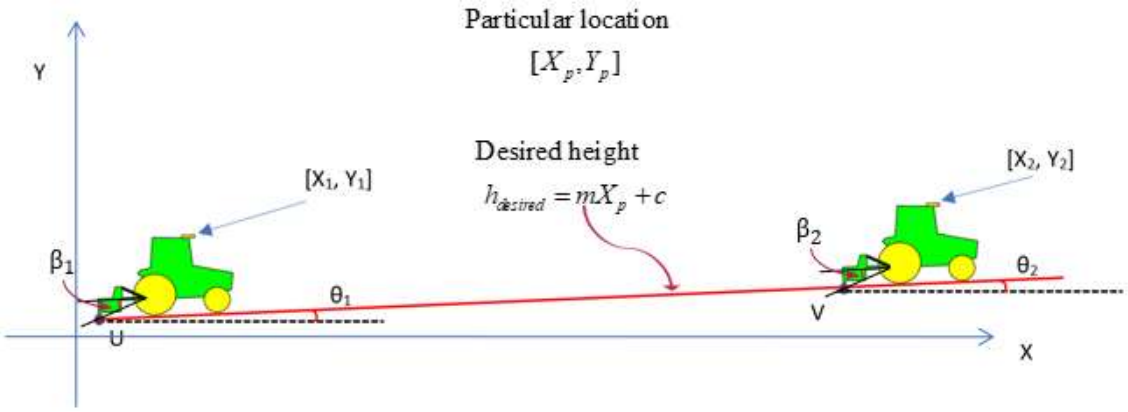

Figure 4.5: Ground Reference Line Calibration

Ground Reference Line Calibration Procedure:

1. Position the tractor at the beginning of the path to be leveled. Read GPS point $[X_1, Y_1]$, θ_1, and leveler angle β_1 using a John Deere GPS receiver and angle sensor fitted on the tractor.
2. Position the tractor at the end of the path. Read point $[X_2, Y_2]$, θ_2, and β_2.
3. Determine coordinates of **U** =$[U_x, U_y]$ and **V** =$[V_x, V_y]$ from points $[X_1, Y_1]$, $[X_2, Y_2]$, and angles θ_1, θ_2, β_1, and β_2 obtained, using equations formulated in the previous page.
4. Compute a ground reference line, $Y = mX + c, \quad m = \dfrac{V_y - U_y}{V_x - U_x}, \quad c = V_y - mV_x$
5. When the tractor is at a particular location $[X_p, Y_p]$, the desired leveler height is

   $h_{desired} = mX_p + c$

As the simulation progresses, the tractor figure moves along the pre-designed terrain shown in Figure 4.6, and the operator presses the raise or lower arrow key on a computer keyboard to move point **C** the leveler tip on the rear implement or the leveler pivoted on point **D** up or down.

The training run data is logged in a .mat file for a training routine written to train the FNN. To ensure data integrity, a replay code was written to replay the data captured during the training run. A training data .mat file is then used for FNN training. A set of optimized weights and biases is then generated and logged in a .mat file. Finally, the .mat file that contains the weights and biases is fed back into the MATLAB program code to evaluate the effectiveness of the FNN training.

The RMS error of the training run is compared with the FNN-trained RMS error data. A successful training would result in an FNN-trained output RMS error that either comes close to or outperforms the training data.

In this simulation, it was assumed that the beginning and the end points have the same altitude, which is the ground reference line, $\mathbf{R} = [R_x, R_y] = [R_x, 0]$.

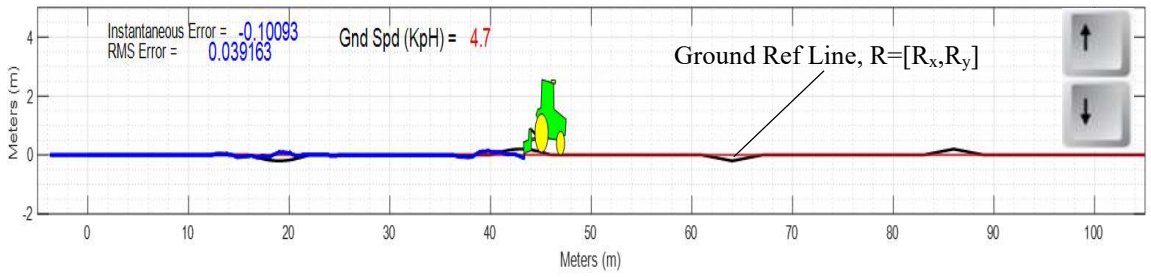

Figure 4.6: Simulation Environment with Human-in-the-loop Keyboard Buttons

The RMS error was calculated per the equation below:

$$\text{RMS Error} = \sqrt{\frac{\sum_{i=1}^{m} (E_i)^2}{m}} \tag{4.24}$$

The training data .mat file was replayed using a playback routine written to ensure the integrity of logged data before being used for the training routine in the next section. Figure 4.7 shows a complete tractor data-gathering run. The top graph shows the tractor at 20 meters, the middle graph shows the tractor at about 60 meters, and the bottom graph shows the tractor at the finish point.

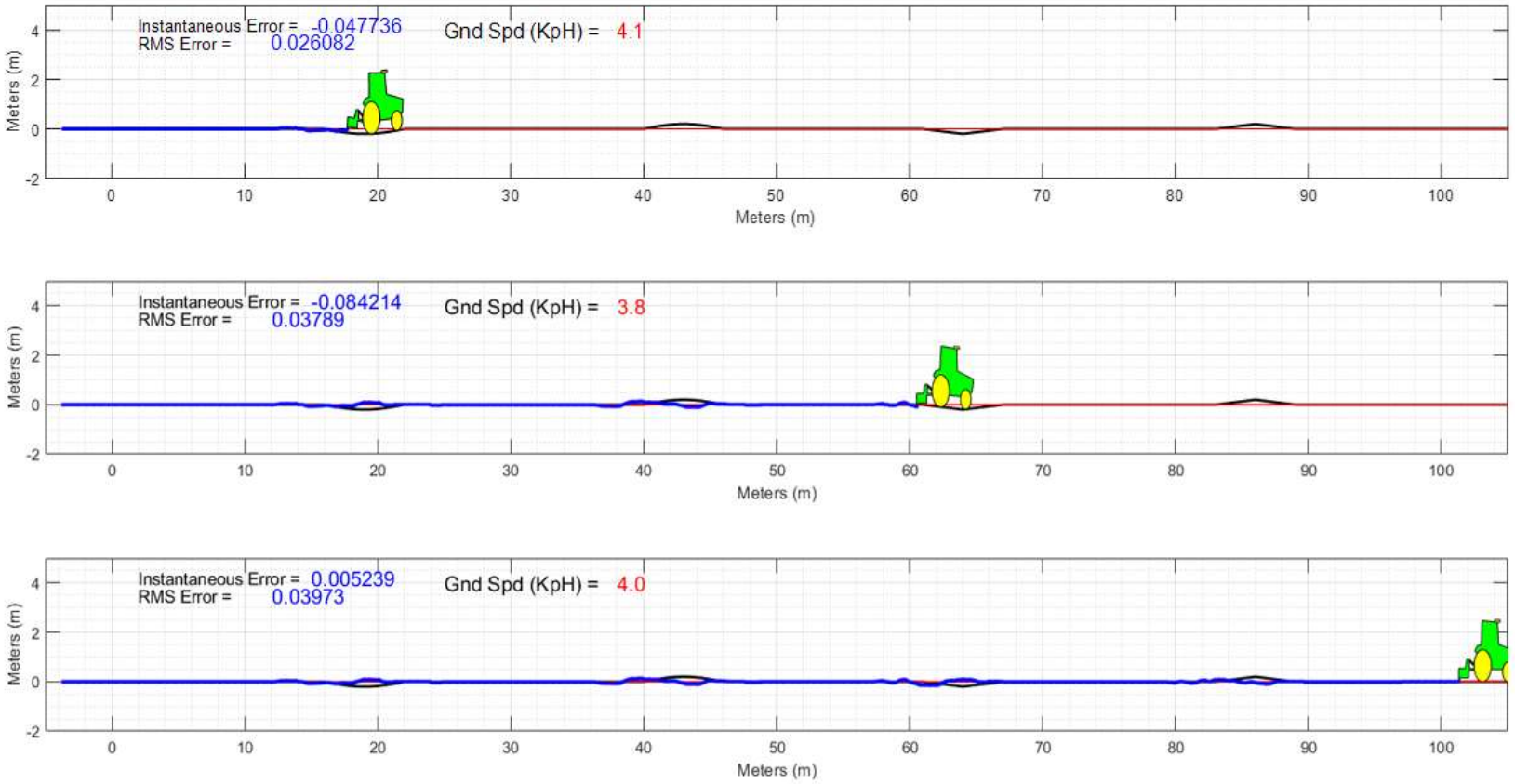

Figure 4.7: Data collection snapshots at various points during simulation

4.2 Train with SVM, LVQNN, NB

As stated in the previous chapter, because the training data outlined in Figure 4.1 has fully defined labels of inputs and output, supervised machine learning is the most suitable algorithm to generate a complete function that can be utilized to process inputs that contain ground-level error and inclined angle.

4.2.1 Why SVM, LVQNN, NB, and FNN?

After examining the various types of supervised machine learning algorithms, the most appropriate ones that were considered are the Support Vector Machine (SVM), Learning Vector Quantization Neural Network (LVQNN), Naïve Bayes (NB), and Feedforward Neural Network (FNN). This dissertation modified the traditional FNN and established TFNN as a digitized version of the continuous FNN function to allow three distinct output levels. All the above are Linear Classifiers, except for NB, a Probabilistic Classifier. [47]

4.2.2 Datasets used for training

Two separate training data sets were used for supervised machine learning algorithm training in sections 4.2.5, 4.2.6, 4.2.7, and 4.2.8.

The first set of data was manually edited to have a clear distinct separation among the three user inputs, which are 1 (raise), 0 (no action), or -1 (lower). The training data set is presented in section 4.2.3.

The second data set was unedited and had “flawed” or “noisy” human-in-the-loop overlapped region when the user should have commanded raise/neutral/lower due to delayed reaction time. This training data set is presented in section 4.2.4.

The two training data sets were used to show which machine learning algorithms would be effective in ideal “clean” data versus real-life “flawed” or “noisy” data.

4.2.3 Data Set 1 – Edited clean data with clear separation

This training data set was collected and edited to distinguish between Raise, Neutral, and Lower output decisions. The plotted data points in 3D and 2D views can be seen in Figure 4.8 and Figure 4.9. The rest of the dissertation will refer to this as the Training Data Set 1.

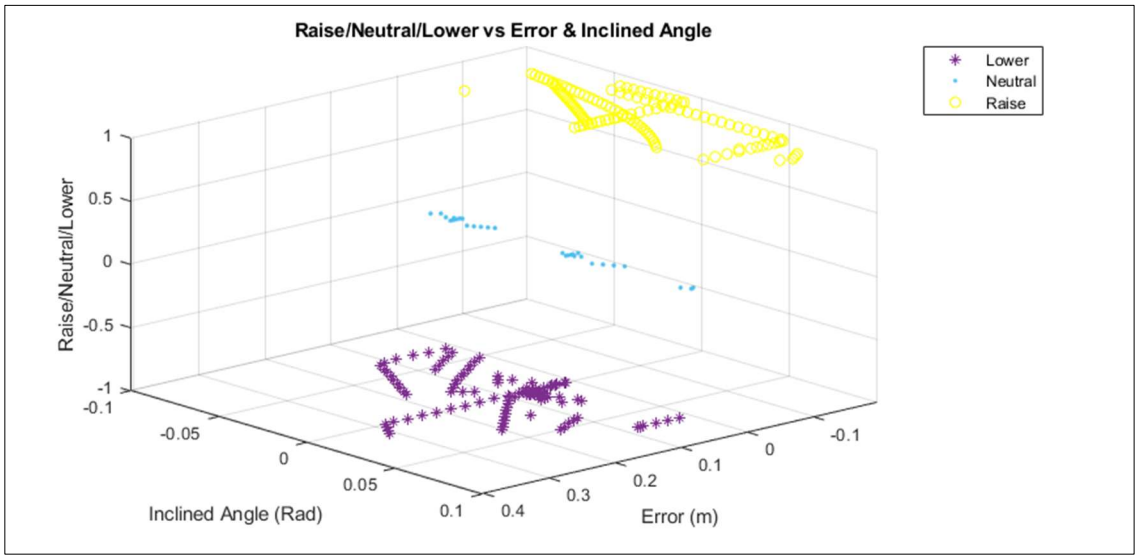

Figure 4.8: Raise/Neutral/Lower output points plot in 3D view

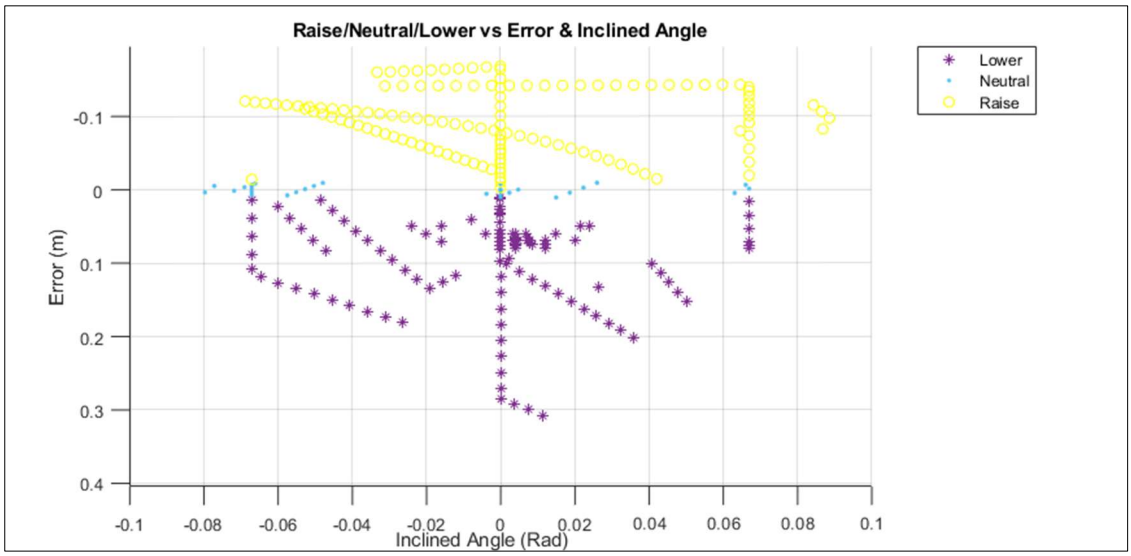

Figure 4.9: Raise/Neutral/Lower output points in 2D top view

4.2.4 Data Set 2 – Unedited noisy data with no clear separation

This training data set shows the unedited data with the original flawed user-controlled raise/neutral/lower commands. When the rear leveling scraper was seen dipping below the ground leveling surface, it often takes some human reaction time to raise the implement by pressing the raise keyboard button. Therefore, it was inevitable that the neutral command would spill into either over-correction or under-correction. Hence, this data set closely resembles what the actual tractor operator would do during the ground leveling operation on a real tractor. The plotted data points in 3D and 2D top views of the training data can be seen in Figure 4.10 and Figure 4.11. The rest of the dissertation will refer to this as Training Data Set 2.

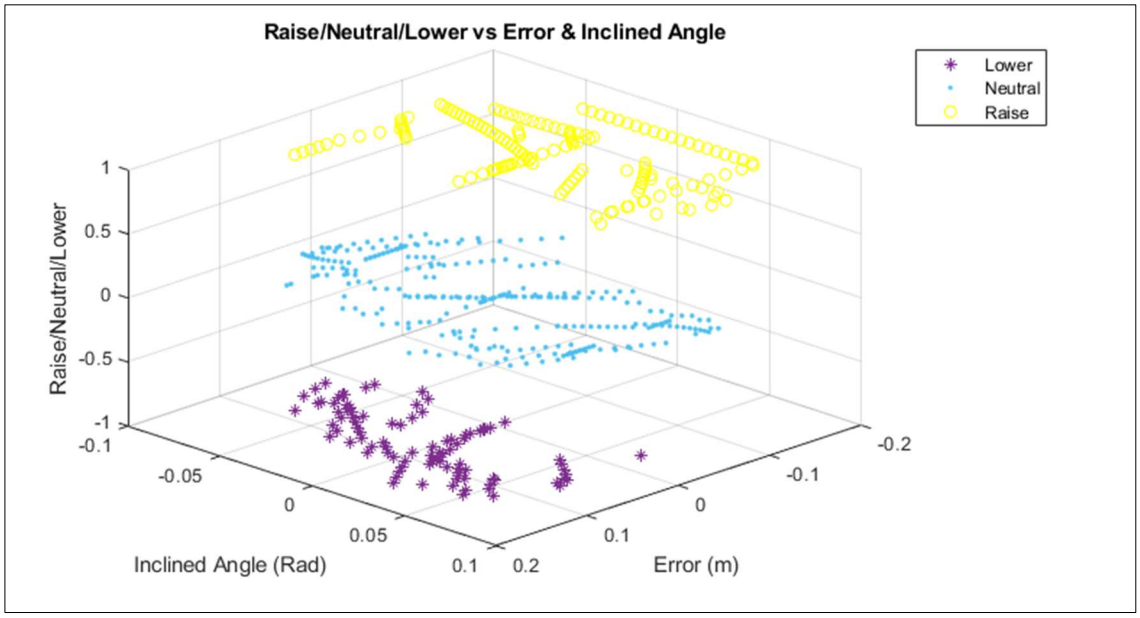

Figure 4.10: Raise/Neutral/Lower output points plot in 3D view

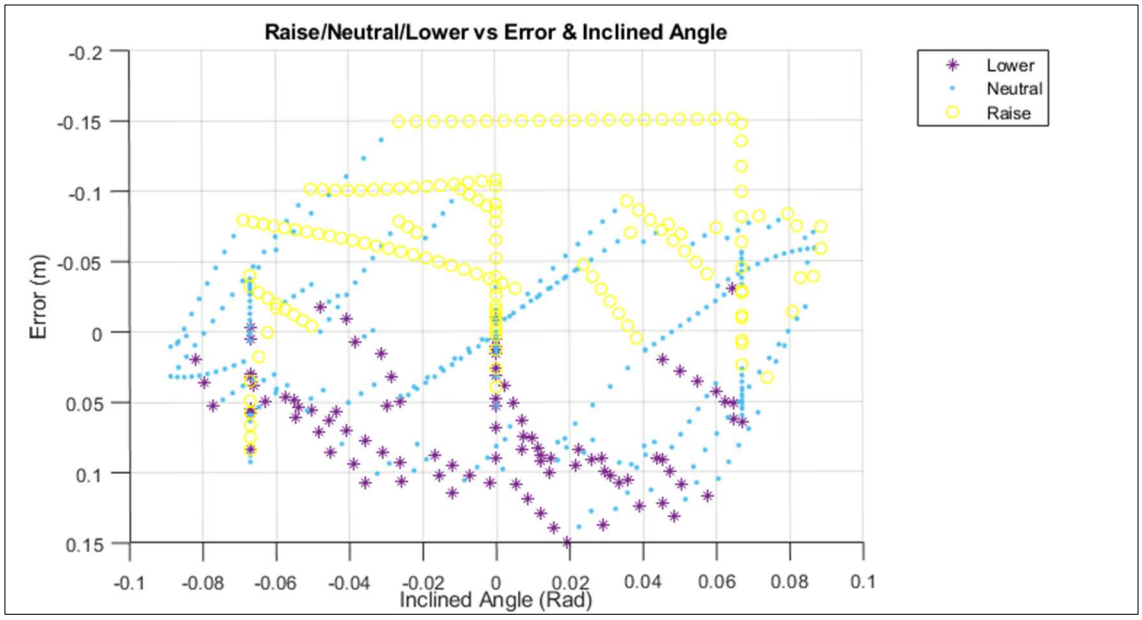

Figure 4.11: Raise/Neutral/Lower output points plot in 2D top view

In the following four (4) sections (4.2.5, 4.2.6, 4.2.7, and 4.2.8), both training data sets above will be used for SVM, LVQNN, NB, and FNN/TFNN training algorithms.

4.2.5 Support Vector Machine (SVM)

A. SVM Intro

Support Vector Machine, also called Support Vector Classification, is a commonly used and effective supervised Machine Learning algorithm. This method helps sort the data into two or more categories with clear boundaries to differentiate among them. [48]

In 1995, Cortes and Vapnik developed a support-vector network utilizing polynomial input transformation for Optical Character Recognition and compared the performance with various classical learning algorithms. [49]

B. Simulation and Result

Both training data sets 1 and 2 are shown in sections 4.2.3 and 4.2.4 were used for SVM algorithm training. The training results are shown below.

a) SVM training result with data set 1 (clear separation):

SVM training algorithm was applied to the edited data taken from the Human-in-the-loop data gathering simulation program shown on the top in Figure 4.13. The training output on the bottom shows distinct raise (yellow), neutral (cyan), and lower (purple) levels that match the training data patterns. Therefore, it is effective in learning the training data.

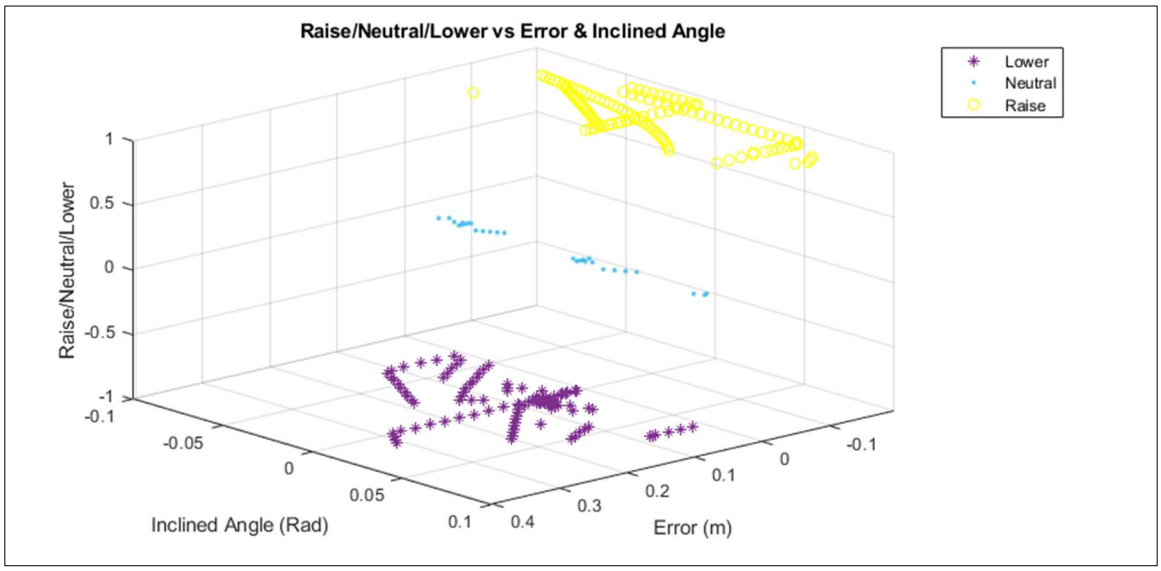

Figure 4.12: Training data set 1

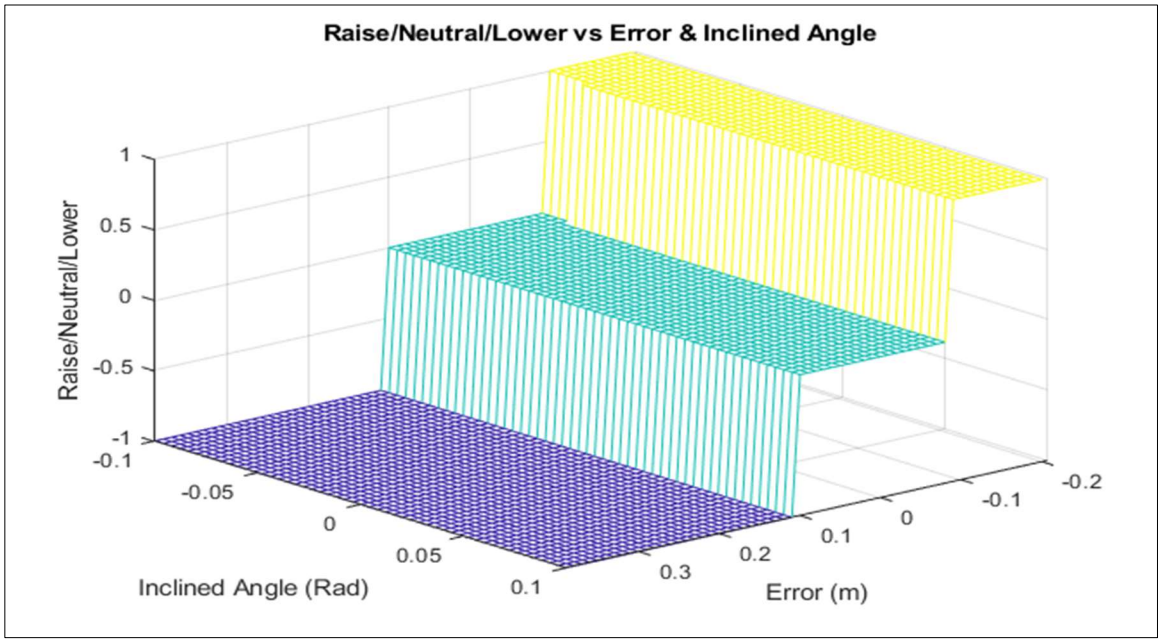

Figure 4.13: SVM training result

The trained outputs were used to process the inputs, which are the terrain parameters to assess the effectiveness of the training algorithms. The results can be observed in Figure 4.14. The algorithm performs well with an RMS error of 0.07961.

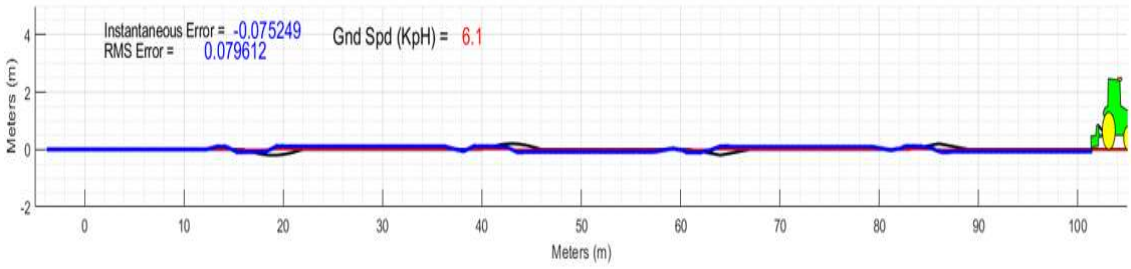

Figure 4.14: Dataset 1 simulation result using SVM algorithm

b) SVM training result with data set 2 (no clear separation):

The SVM algorithm was then applied to the training dataset 2 (no clear separation). Due to the overlapped nature of the data resulting from typical human reaction time that impacted the preferred clear separation between raise/neutral/lower, the SVM algorithm was ineffective in producing a sound output compared to the ones trained with training data 1 (clear separation). As can be seen on the bottom of Figure 4.16 the training output was only neutral. Therefore, it did not produce a learned output to adjust the rear scraper or leveler to raise/neutral/lower accordingly.

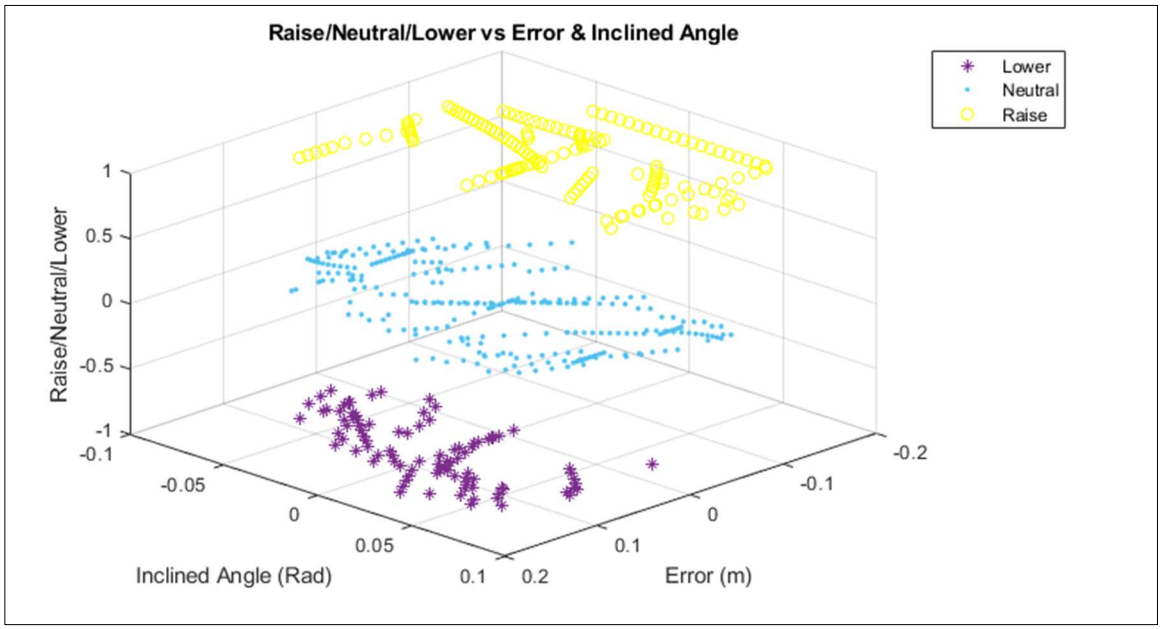

Figure 4.15: Training dataset 2

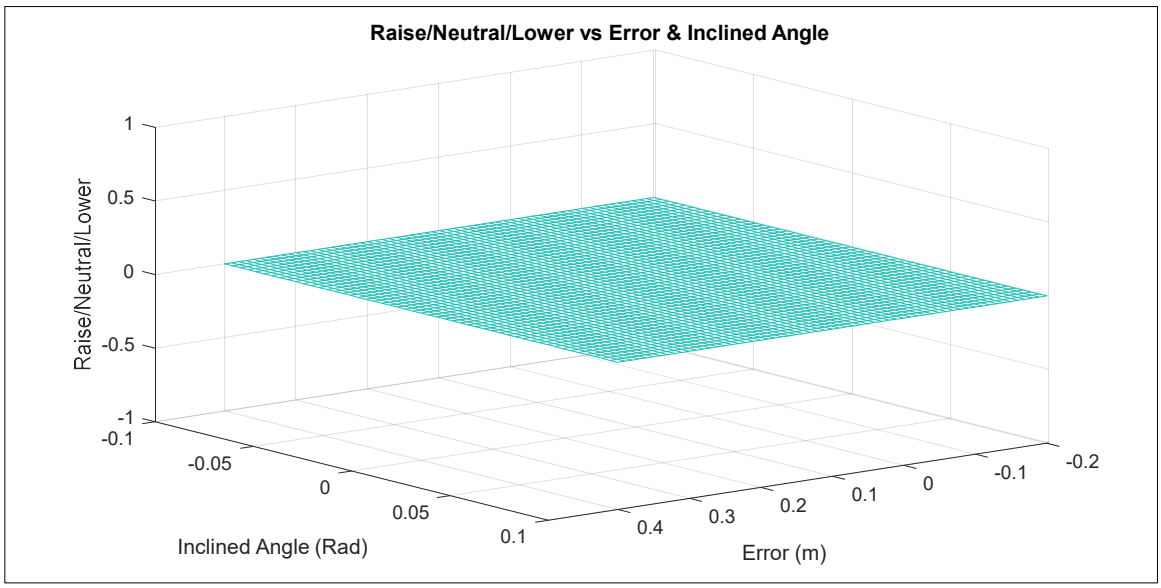

Figure 4.16: SVM training result

The training output above evaluated tractor leveling performance using the same terrain for data gathering. The simulation yielded an RMS error of 0.09906, shown in Figure 4.17 below, which is not a good result as predicted from the training output data.

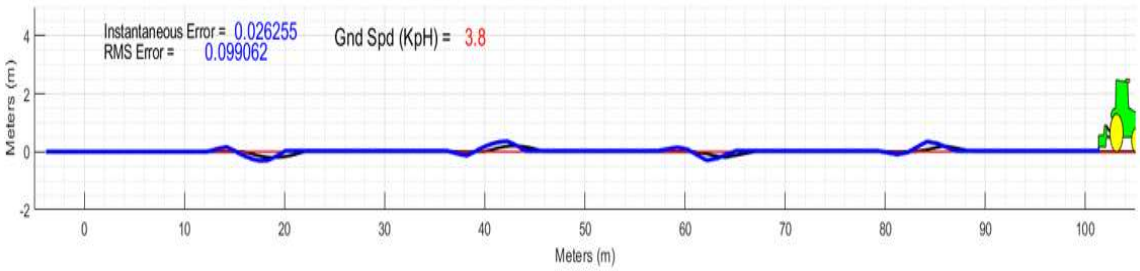

Figure 4.17: Dataset 2 simulation result using SVM algorithm

4.2.6 Learning Vector Quantization Neural Network (LVQNN)

A. LVQNN Intro

Learning Vector Quantization Neural Network (LVQNN) is a competitive learning system introduced by Teuvo Kohonen. [50] It is the supervised learning counterpart of SOM (Self-Organizing Map). Therefore, a labelled training data set shall be provided to train LVQNN.

LVQNN was successfully applied in many applications, including automatic text classification, specifically text categorization and word sense disambiguation. [51]

B. Simulation Result

Both data sets 1 and 2 were used for LVQNN algorithm training. The training results are shown below.

a) LVQNN training result with data set 1 (clear separation):

The LVQNN training algorithm was applied to training data set 1 (clear separation) shown on the right of Figure 4.19. The training output on the bottom graph shows distinct raise/neutral/lower levels that match the training data patterns. It shows that LVQNN worked well in learning the three different levels of user operation.

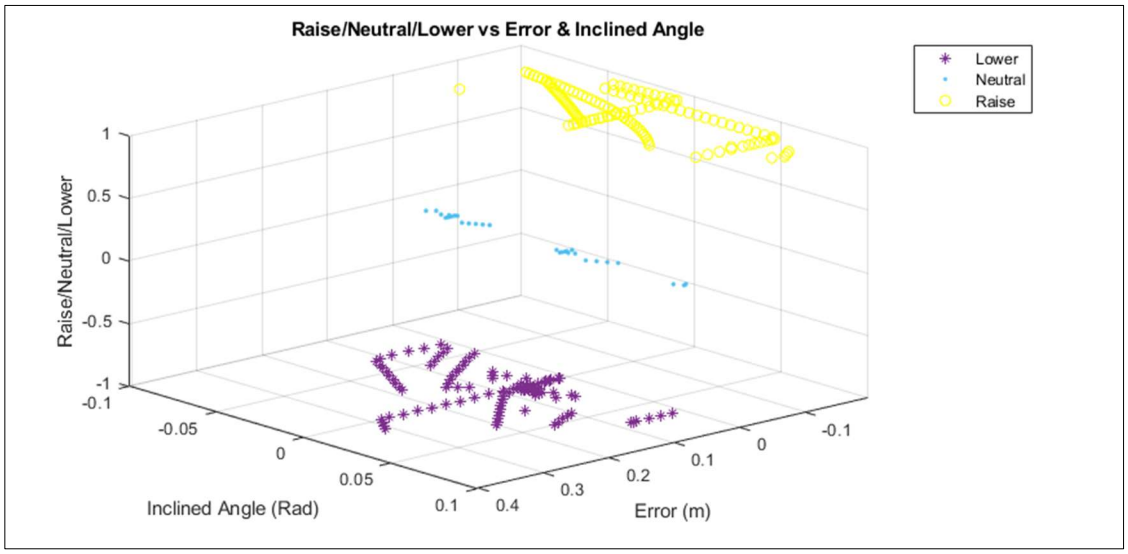

Figure 4.18: Training data set 1

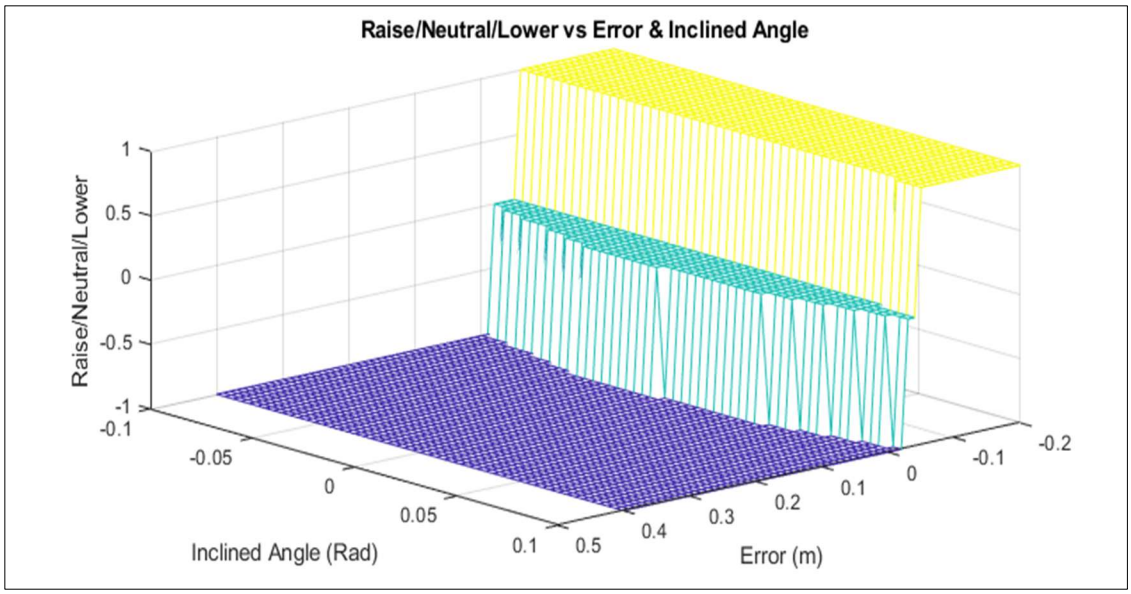

Figure 4.19: LVQNN training result

The trained outputs were applied to the terrain used for training. The results can be observed in Figure 4.20. The algorithm performs well with an RMS error of 0.0345.

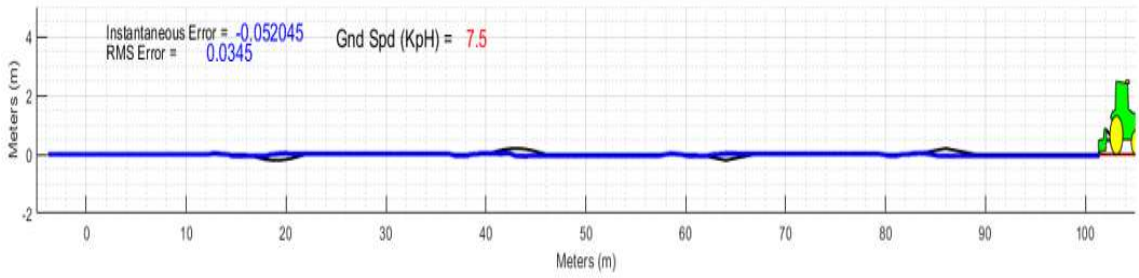

Figure 4.20: Dataset 2 simulation result using LVQNN algorithm

b) LVQNN training result data set 2 (no clear separation):

The LVQNN training algorithm was applied to training data set 2 (no clear separation). The result showed that it was ineffective in producing a good simulation output

compared to those trained with data set 1. As can be seen on the bottom of Figure 4.22, the training output does not have any raise command with the value of 1.

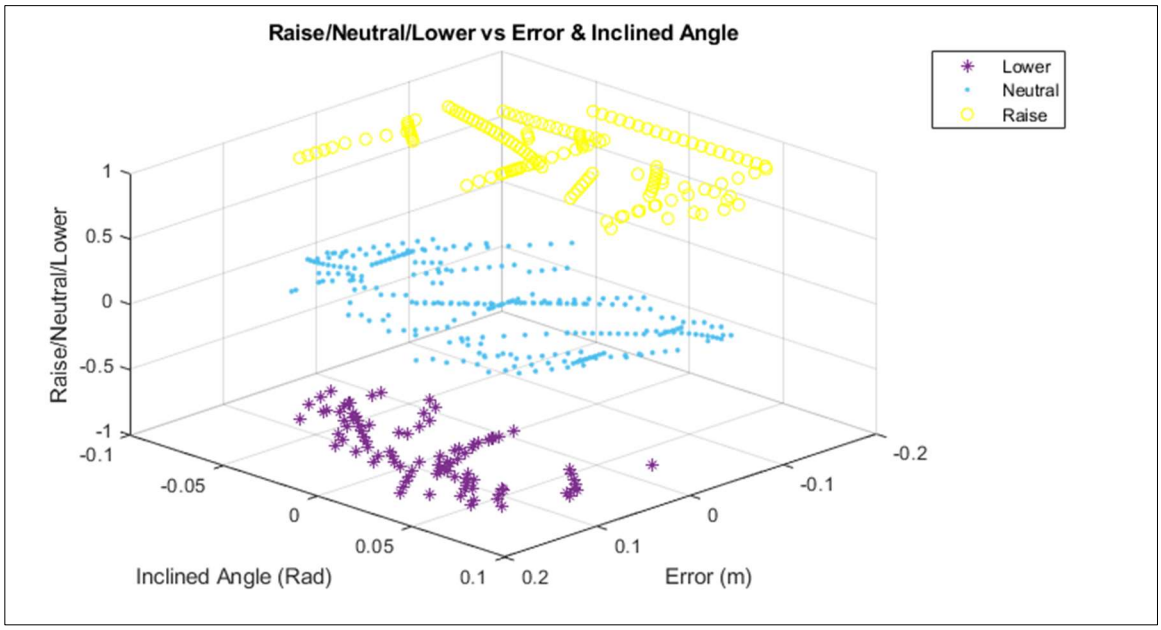

Figure 4.21: Training data set 2

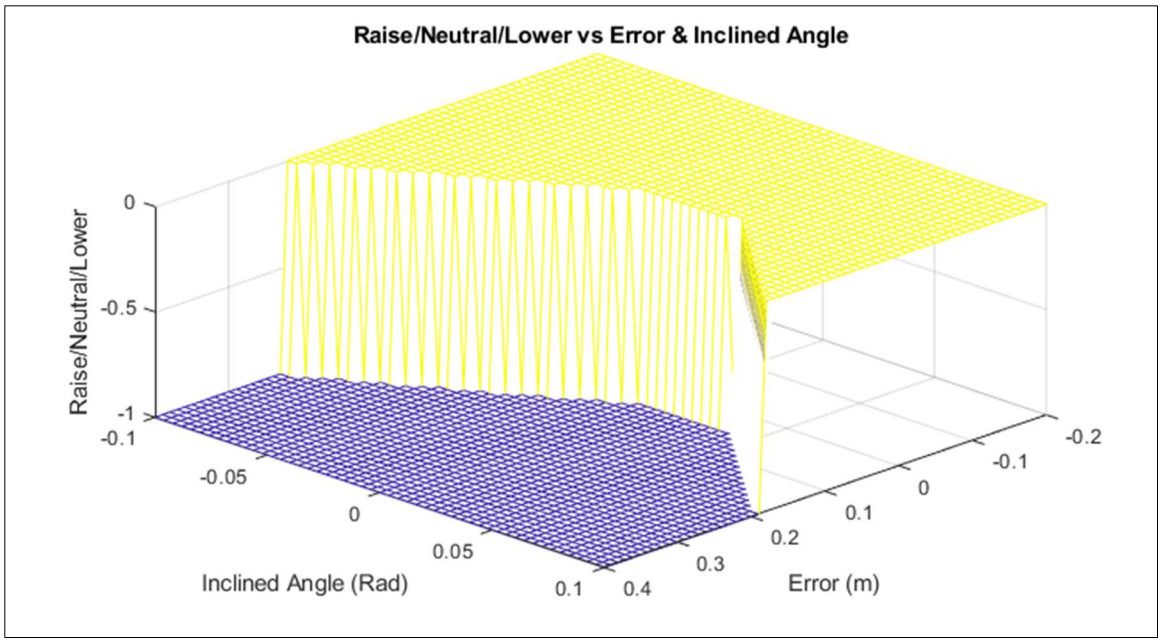

Figure 4.22: LVQNN training result

The trained outputs were applied to the terrain used for training. The results can be observed in Figure 4.23. The algorithm performs the worst among all four supervised machine learning methods with an RMS error of 0.14737.

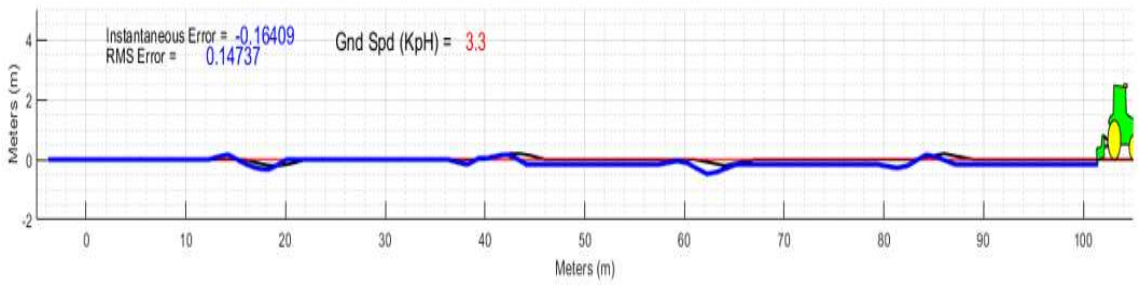

Figure 4.23: Dataset 2 simulation result using LVQNN algorithm

4.2.7 Naïve Bayes (NB)

A. Naïve Bayes Intro

Naïve Bayes is a family of algorithms based on a common principle that each feature is independent of other features. It was also demonstrated that Naïve Bayes works well for nearly functional feature dependencies. [52] Naïve Bayes' classification has been applied in many medical areas with good accuracy and results. [53]

B. Simulation Result

Both data sets 1 and 2 were used for Naïve Bayes algorithm training. The training results are shown below.

a) NB training result with data set 1 (clear separation):

NB training algorithm was applied to training data set 1 (clear separation). The training output on the bottom graph below shows three distinct levels that match the training data patterns. It showed that NB worked well in learning the training data.

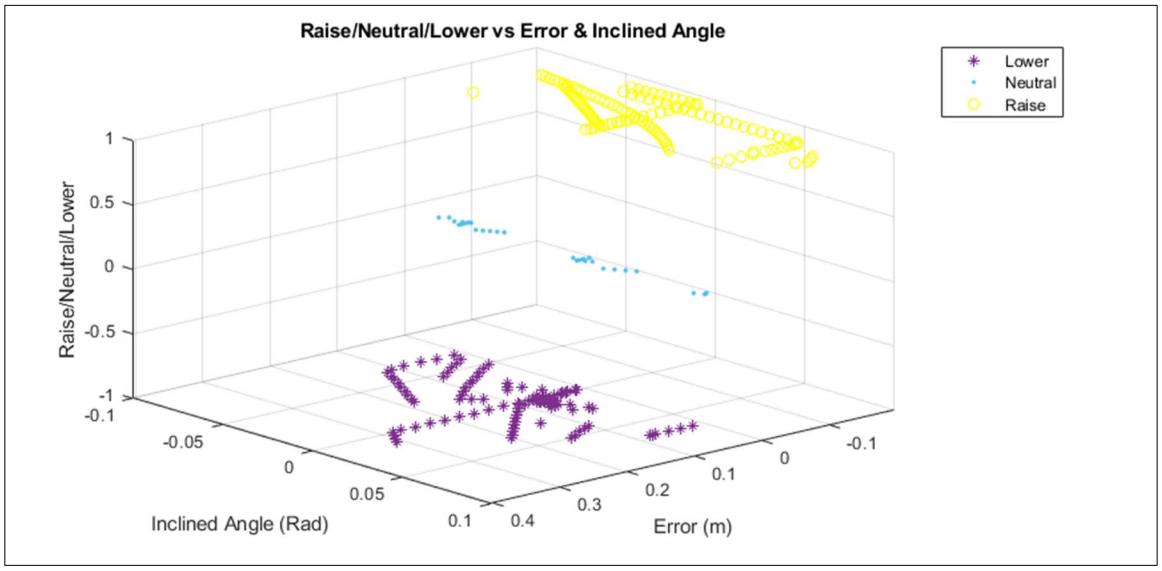

Figure 4.24: Training data set 1

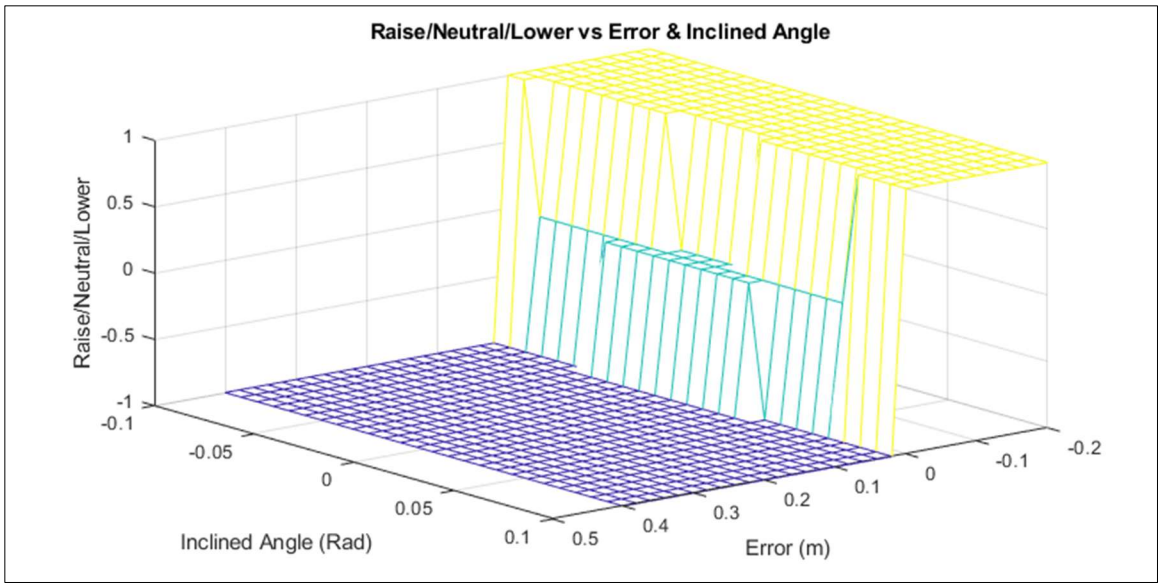

Figure 4.25: NB training result

The trained outputs were applied to the terrain used for training. The results can be observed in Figure 4.26. The algorithm produced a respectable RMS error of 0.013018.

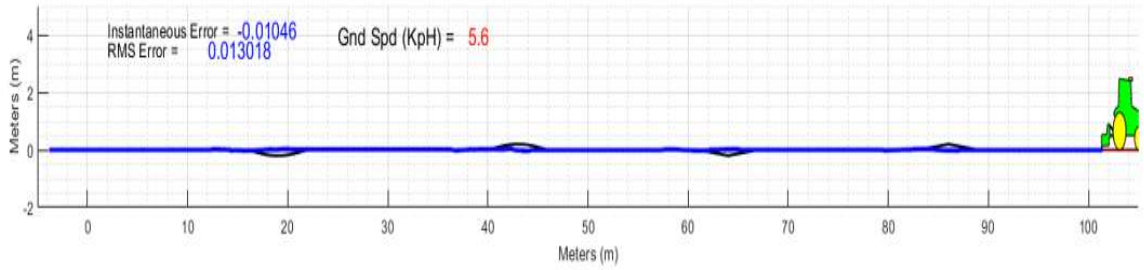

Figure 4.26: Dataset 1 simulation result using output trained by NB algorithm

b) NB training result with data set 2 (no clear separation):

The NB training algorithm was applied to training data set 2 (no clear separation). The result showed it was still somewhat effective in producing a good output, although not as good as training data set 1.

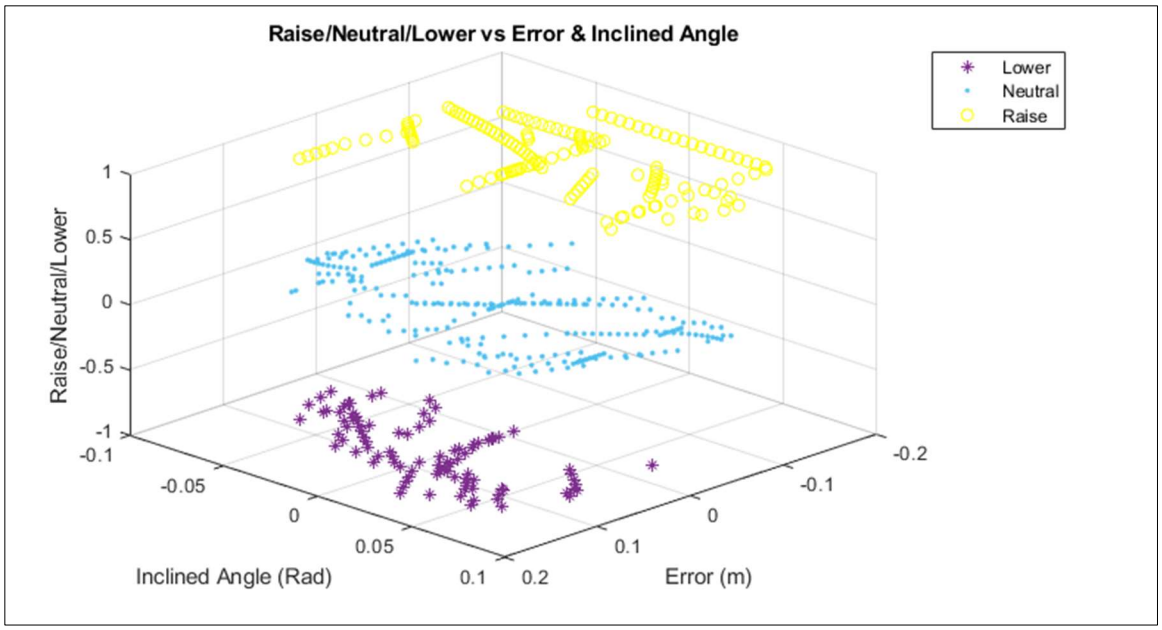

Figure 4.27: Training data set 2

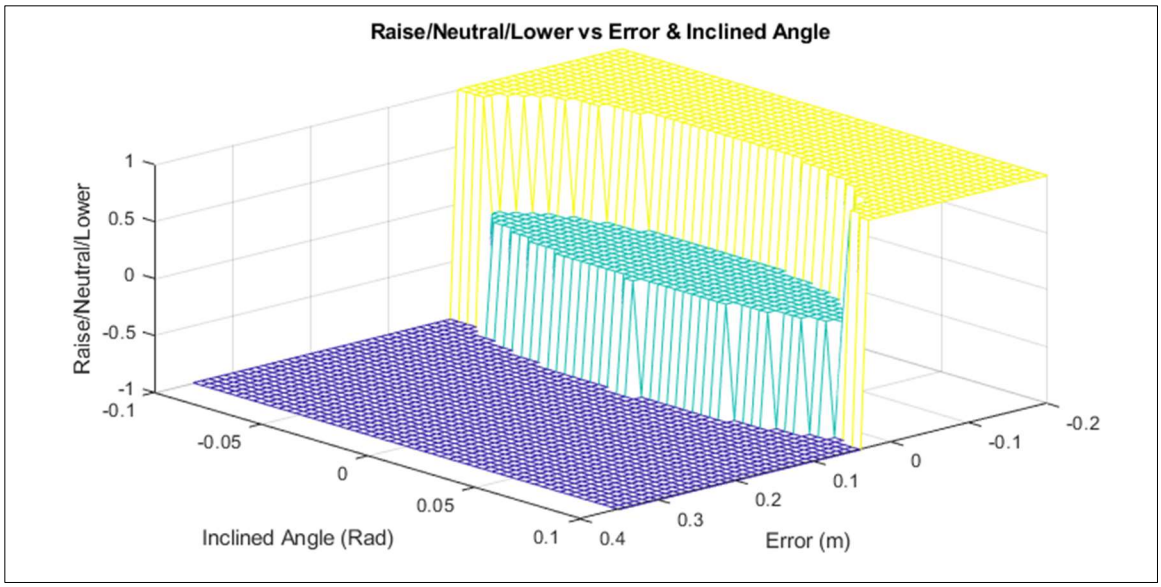

Figure 4.28: NB training result

The trained outputs were applied to the terrain used for training. The results can be observed in Figure 4.29. The algorithm produced an RMS error of 0.05079, which is only second to TFNN. The TFNN training result will be shown next in section 4.2.8.

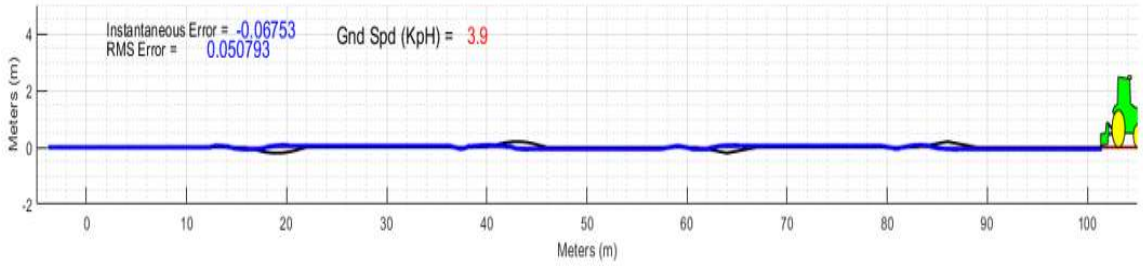

Figure 4.29: Dataset 2 simulation result using output trained by NB algorithm.

4.2.8 Threshold-Feedforward Neural Network (TFNN)

A. FNN Intro

FNN is a type of ANN. The simplest FNN is a one-layered perceptron proposed by Frank Rosenblatt in 1957. [54] The implementation can be seen below:

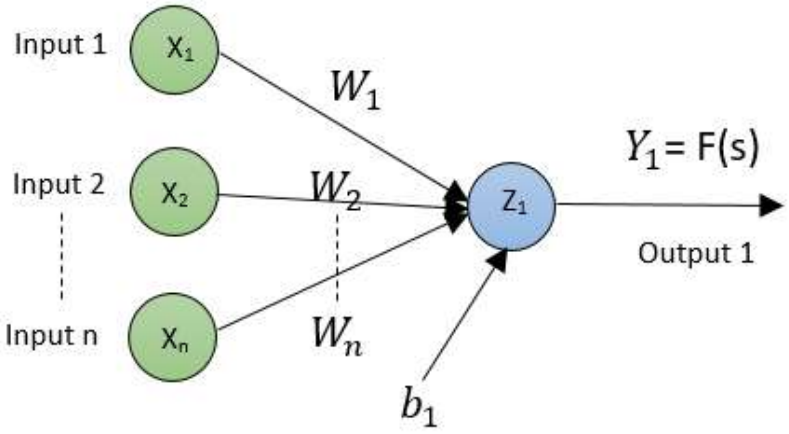

Figure 4.30: Perceptron Block Diagram

The inputs are X_1 to X_n, the weights are W_1 to W_n, and the bias is b_1. The single neuron takes the summation and yields $\mathbf{s} = \mathbf{W}\mathbf{x} + b$, where $\mathbf{W} = [W_1, \cdots, W_n]$, $\mathbf{x} = [x_1, \cdots, x_n]'$, $b = [b_1]$. The result **s** is fed into an activation function $F()$. The activation function can be chosen to suit the application needs. The matrix operation is shown below:

$$\mathbf{s} = [W_1, \cdots, W_n] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + b_1 = \mathbf{W}\mathbf{x} + b \tag{4.25}$$

$$Y_1 = F(s) \tag{4.26}$$

Data $x_1 \ldots x_n$ are multiplied by training weights $w_1 \ldots w_n$ and added to bias b. The output Y_1 is compared with the desired output. If Y_1 is not equal to the desired output, the

weights and bias can be adjusted with each iteration until the error converges to 0. At this point, the weights and bias cease to change.

A multiple-layered, multiple-neuron perceptron is also called an FNN. A two-input, two-layered network with two neurons in the hidden layer (middle layer) and one neuron output layer (2-2-1) can be seen below:

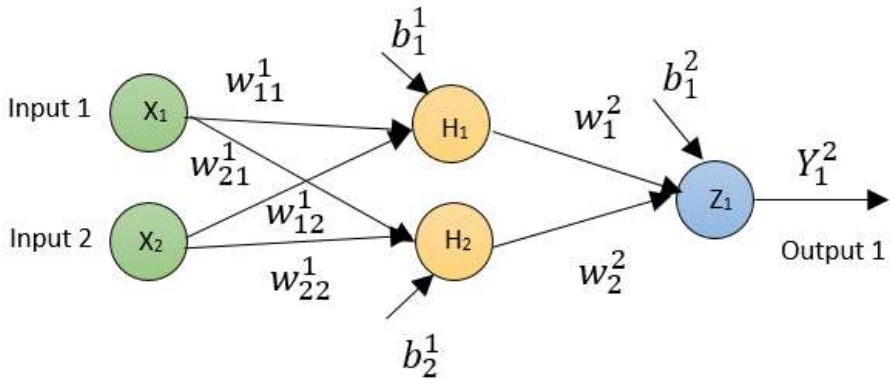

Figure 4.31: FNN 2-2-1 Block Diagram

There are three (3) artificial neurons represented in Figure 4.31 above, which are H_1, H_2, and Z_1. The multilayer artificial neural network teaches itself using a backpropagation learning algorithm. The basic idea is to calculate and minimize the error by comparing the desired output with the actual output Y_1^2. Weights and biases, **W** and **b,** are adjusted and backward-propagated from the output layer to the input layer. After iterations, when the error reaches zero, meaning the desired output is close to the actual output, the weights and bias values would converge and cease to change. [55]

The two-layered FNN used for the simulation can be seen in Figure 4.32:

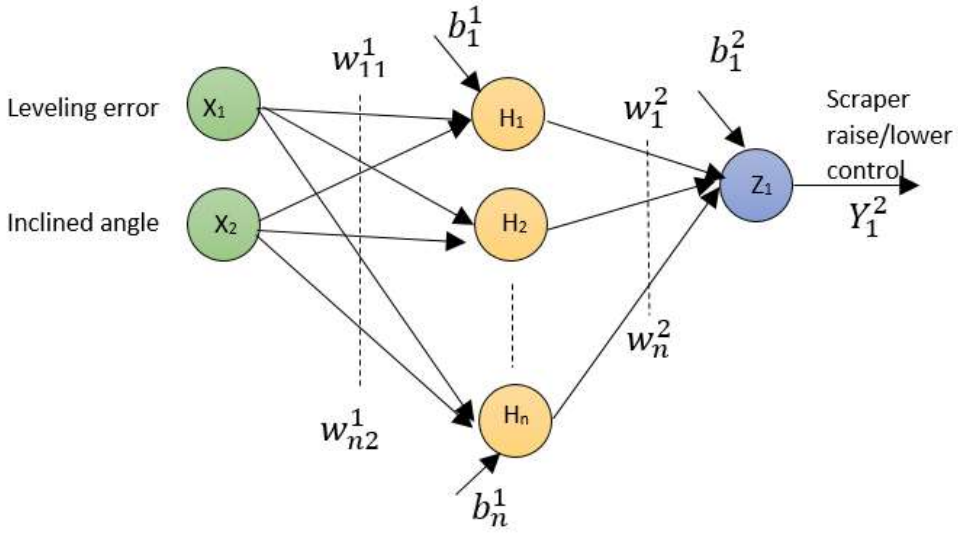

Figure 4.32: 2-n-1 FNN Block Diagram

The input layer includes $x_1 = E = \text{Leveling error}$, and $x_2 = \theta = \text{Inclined angle}$. The hidden layer consists of n-number of neurons, which can be adjusted to achieve the best output. $Y_1^2 = u_{FNN}$ is the lone action output.

The feedforward 2-n-1 FNN can be represented in vectors below:

$$\begin{bmatrix} s_1^1 \\ \vdots \\ s_n^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ \vdots & \vdots \\ w_{n1}^1 & w_{n4}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ \vdots \\ b_n^1 \end{bmatrix} \tag{4.27}$$

$$\begin{bmatrix} Y_1^1 \\ \vdots \\ Y_n^1 \end{bmatrix} = \begin{bmatrix} F^1(S_1^1) \\ \vdots \\ F^1(S_n^1) \end{bmatrix} \tag{4.28}$$

$$[S_1^2] = [w_{11}^2, \cdots, w_{1n}^2] \begin{bmatrix} Y_1^1 \\ \vdots \\ Y_n^1 \end{bmatrix} + [b_1^2] \tag{4.29}$$

$$[Y_1^2] = [F^2(S_1^2)] \tag{4.30}$$

where $Y_1^2 = u_{FNN}$ = action output

The first or the hidden layer uses logarithmic sigmoid function (logsig), $F^1(s) = \dfrac{1}{1+e^{-s}}$ as the activation function. The second, or the output layer, uses a linear function (purelin), $F^2(s) = s$ as the activation function.

The FNN outputs were digitized and optimized with a lever control threshold of t_1, and t_2, to arrive at the Threshold-Feedforward Neural Network (TFNN) using equations formulated in Chapter Three, i.e.

$$u_{TFNN}(e, t_1, t_2) = \begin{cases} 1 & \text{if } u_{FNN}^*(e) > t_1 \\ 0 & \text{if } u_{FNN}^*(e) \le t_1 \ \ \& \ \ u_{FNN}^*(e) \ge t_2 \\ -1 & \text{if } u_{FNN}^*(e) < t_2 \end{cases}$$

$$u^*_{FNN}(\mathbf{e}) = u_{FNN}(\mathbf{e}, \mathbf{p}^*)$$

In this case $t_1 = 0.1$, and $t_2 = -0.1$ are the two optimized thresholds.

B. Simulation Result

Both data sets 1 and 2 were used for TFNN algorithm training. The training results are shown below.

a. FNN and TFNN training result with data set 1 (clear separation):

FNN training algorithm was applied to the edited data. The training output on the left shows three distinguishable levels that match the training data patterns. It indicates that FNN works well in learning the training data. The graph on the right shows the digitized output of TFNN with the lever control threshold level for the user output set at $t_1 = 0.1$, and $t_2 = -0.1$.

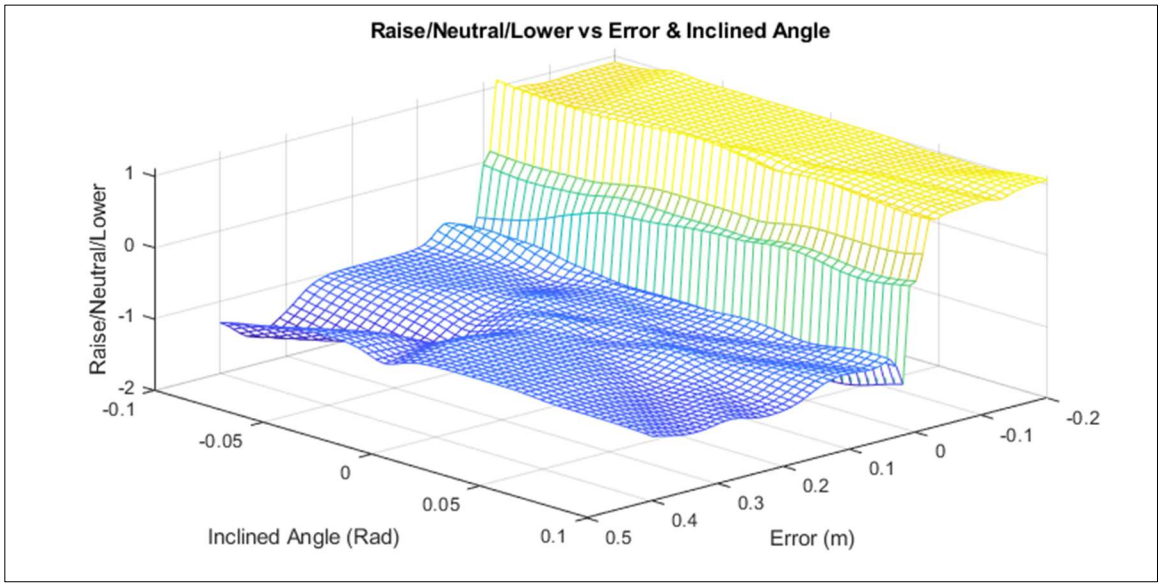

Figure 4.33: FNN training result

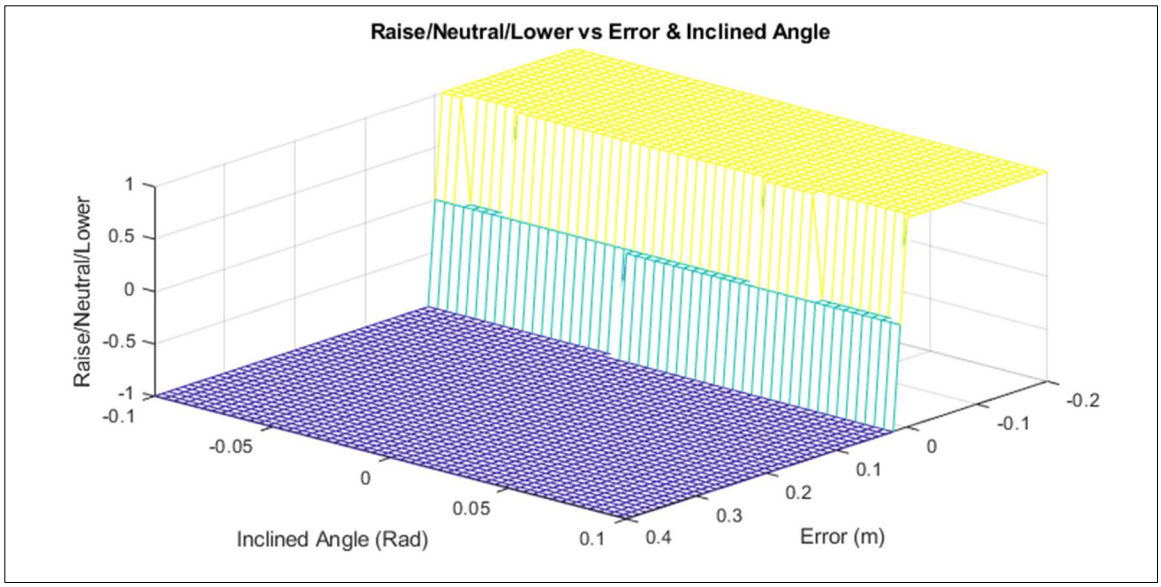

Figure 4.34: TFNN threshold at ± 0.1

The trained outputs were applied to the terrain used for training. The results can be observed in Figure 4.35. The TFNN algorithm performed best among all algorithms with an RMS error of 0.00870.

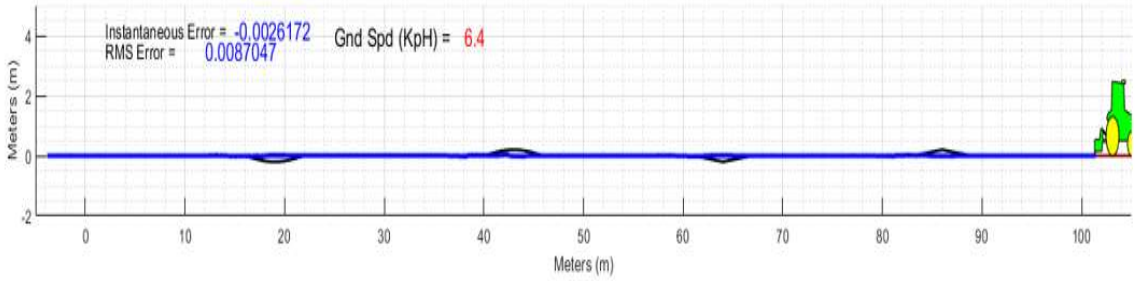

Figure 4.35: Dataset 1 simulation result using FNN algorithm

b. FNN and TFNN training result with data set 2 (no clear separation):

The FNN training algorithm was applied to the non-edited data set 2 without clear separation. The result showed it was still effective in producing the best simulation result. The graph below shows the training result for the TFNN.

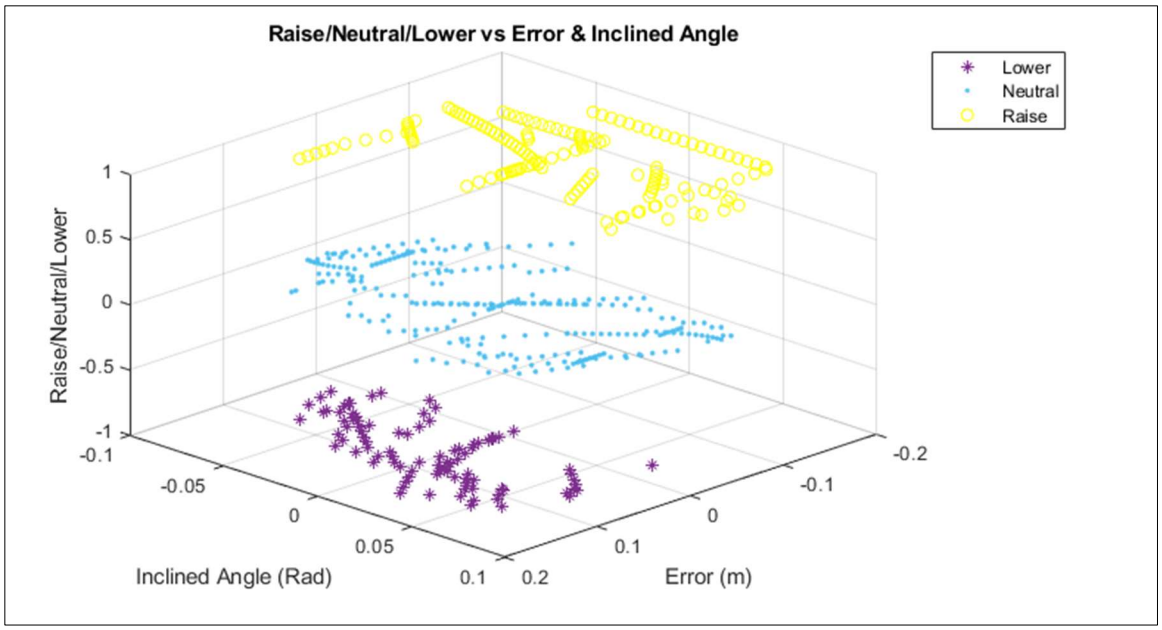

Figure 4.36: Training data set 2

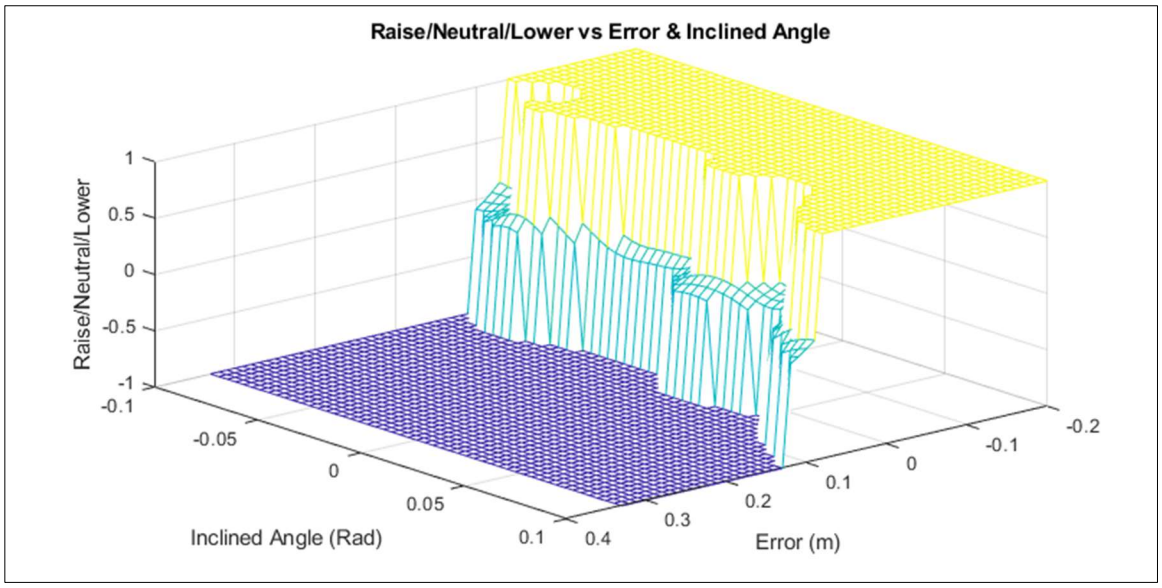

Figure 4.37: TFNN training result threshold ± 0.1

The trained outputs were applied to the terrain used for training. The results can be observed in Figure 4.38. The TFNN algorithm produced an RMS error of 0.01954, the best among all four algorithms.

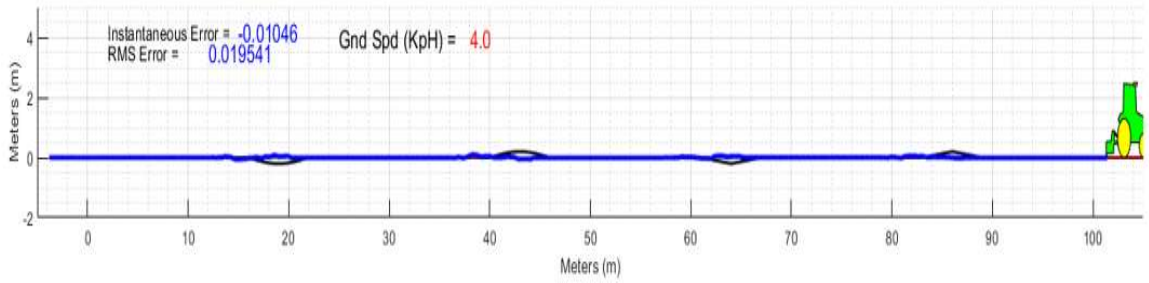

Figure 4.38: Dataset 2 simulation result using FNN algorithm

Different thresholds were evaluated against TFNN training results with Training Data Set 2, and the simulation results like Figure 4.38 were gathered with RMS errors, and Raise/Lower Cycle Counts were compiled below.

Lever Control Threshold (±)	0.025	0.05	0.1	0.2	0.4
RMS Errors	0.0150	0.0161	0.0195	0.0281	0.0510
Raise/Lower Activation Counts	70	29	14	6	5

Table 4.1: RMS errors with different thresholds for TFNN training outputs

Plotting the data in Table 4.1 resulted in the graph in Figure 4.39 that shows diminishing return in RMS error reduction below the ±0.1 lever control threshold value. Furthermore, the rear implement raise/lower motor actuation cycle counts increase at an accelerated rate below the same threshold. Increased frequency of rear implement activation counts of controlling motor actuators could result in long-term durability and

stability issues. Therefore, a Lever Control threshold of ±0.1 was chosen for optimum accuracy and control stability.

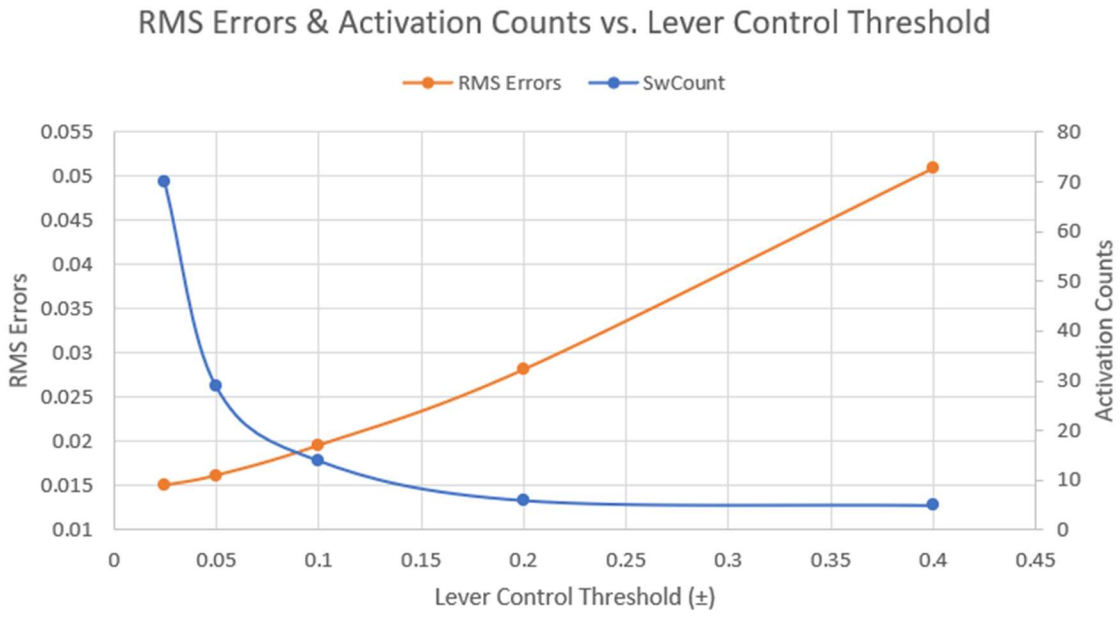

Figure 4.39: RMS Errors & Activation Counts vs Lever Control Threshold

C. SVM, LVQNN, NB, and TFNN Comparison Results

From the qualitative final simulation data based on the four training algorithms above, TFNN proved to be the most effective method compared with SVM, LVQNN, and NB. The results can be seen in Table 4.2 below. For cleaned dataset 1, all four supervised machine learning methods can recognize the patterns of the three-leveled classification outputs, although TFNN had the best performance by far. For noisy Dataset 2, SVM and LVQNN failed to recognize the three-level outputs when applied to the training data, and there was no clear separation. NB performed reasonably well but was not as effective as the TFNN method.

	SVM	LVQNN	NB	TFNN
Dataset 1	0.0796	0.0345	0.0130	0.0087
Dataset 2	0.0991	0.1474	0.0508	0.0195

Table 4.2: RMS errors with various training algorithms

4.2.9 Further and Future Research– K-Mean Clustering

K-Mean Clustering, an unsupervised Machine Learning algorithm, was explored and applied to both the edited and unedited data. As introduced in section 1.2.3, K-Mean Clustering recognizes the pattern and locates the center of grouped data. In the edited data training, the center of each decision output, i.e., raise, neutral, and lower, are located using the Fuzzy C-mean MATLAB function. The centers are plotted with the bigger dots highlighted with the red rounded edges in the graphs on the right in Figure 4.40.

To formulate the threshold of decisions for raise, neutral, and lower, a Voronoi diagram partitions are constructed to separate the three central points of each decision. Voronoi diagrams can be used to cover a variety of phenomena by considering their functions, properties, and characteristics.

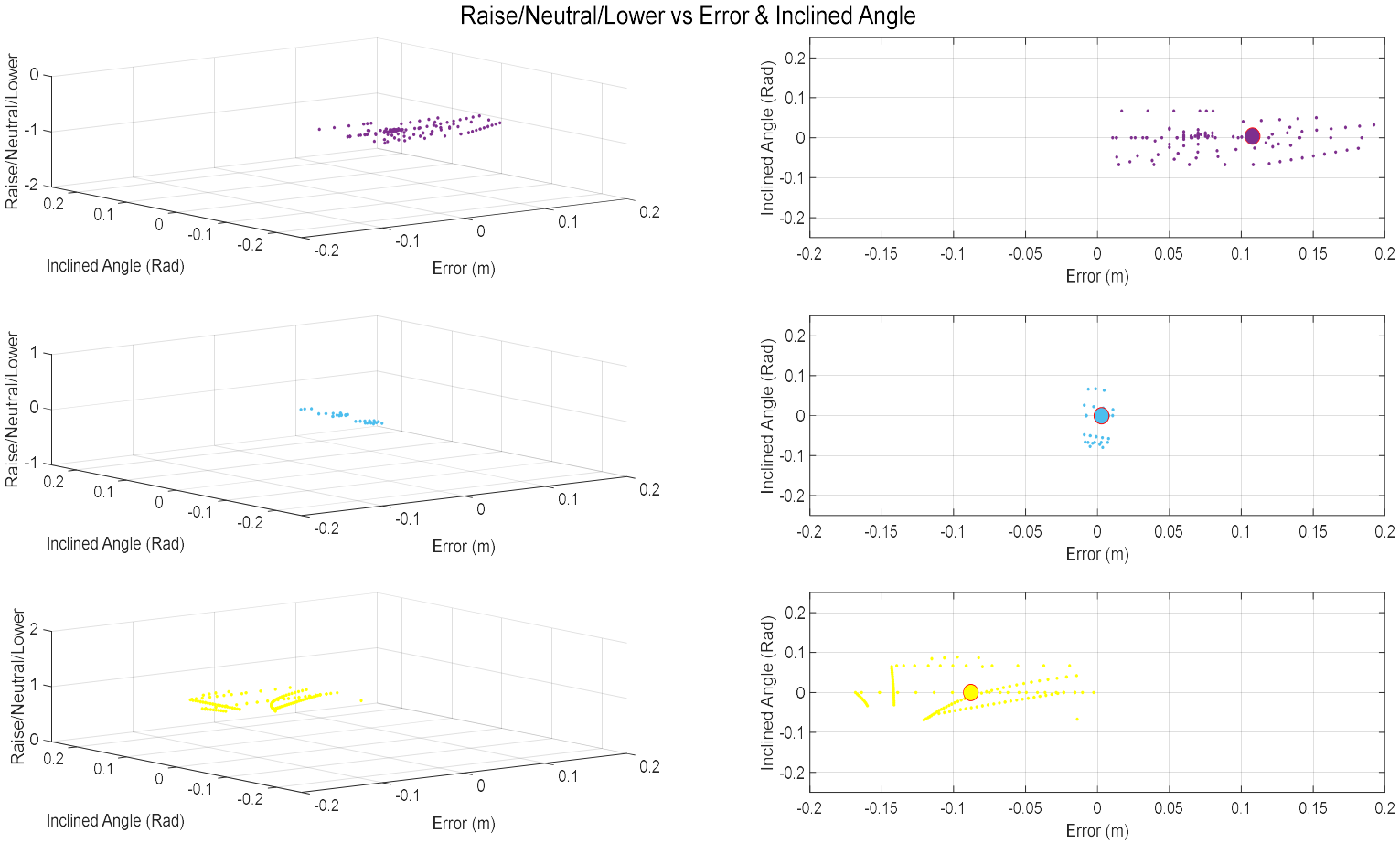

Figure 4.40: Fuzzy C-Mean Centers for the Clean Data

The types of information Voronoi diagrams can help analyze include the shapes of the dominant regions of the data, the neighbor relations among the data, and the structures of empty space between the data groups. [56] In this application, Voronoi is used to draw the equal distributions among the centers of the data groups, namely the raise, neutral, and lower. These would be used as the raise, neutral, and lower thresholds.

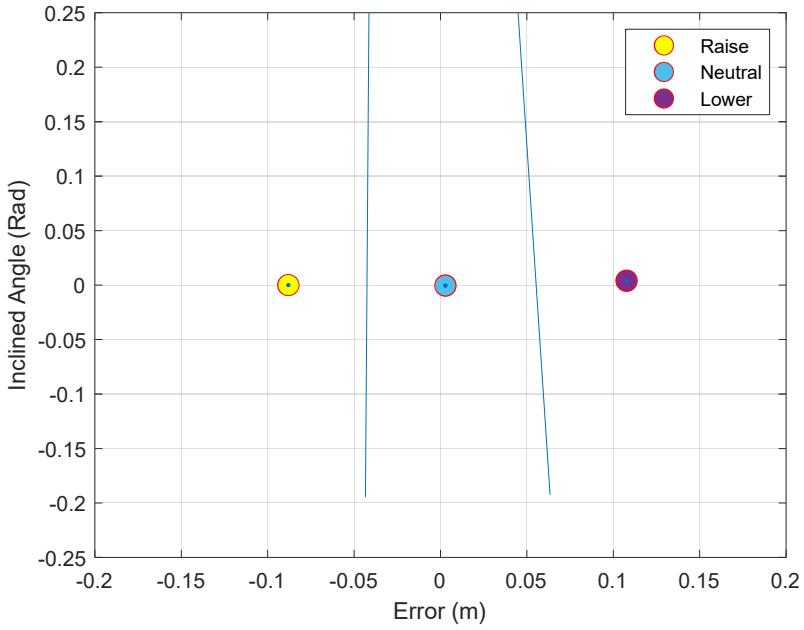

Figure 4.41: Voronoi of FCM Centers of the Clean Data

To compare with the clean data training. The unedited noisy data was also trained with MATLAB fcm K-mean clustering function. The function was able to locate the center of the data group like the clean data group shown in Figure 4.42 in the graphs to the right.

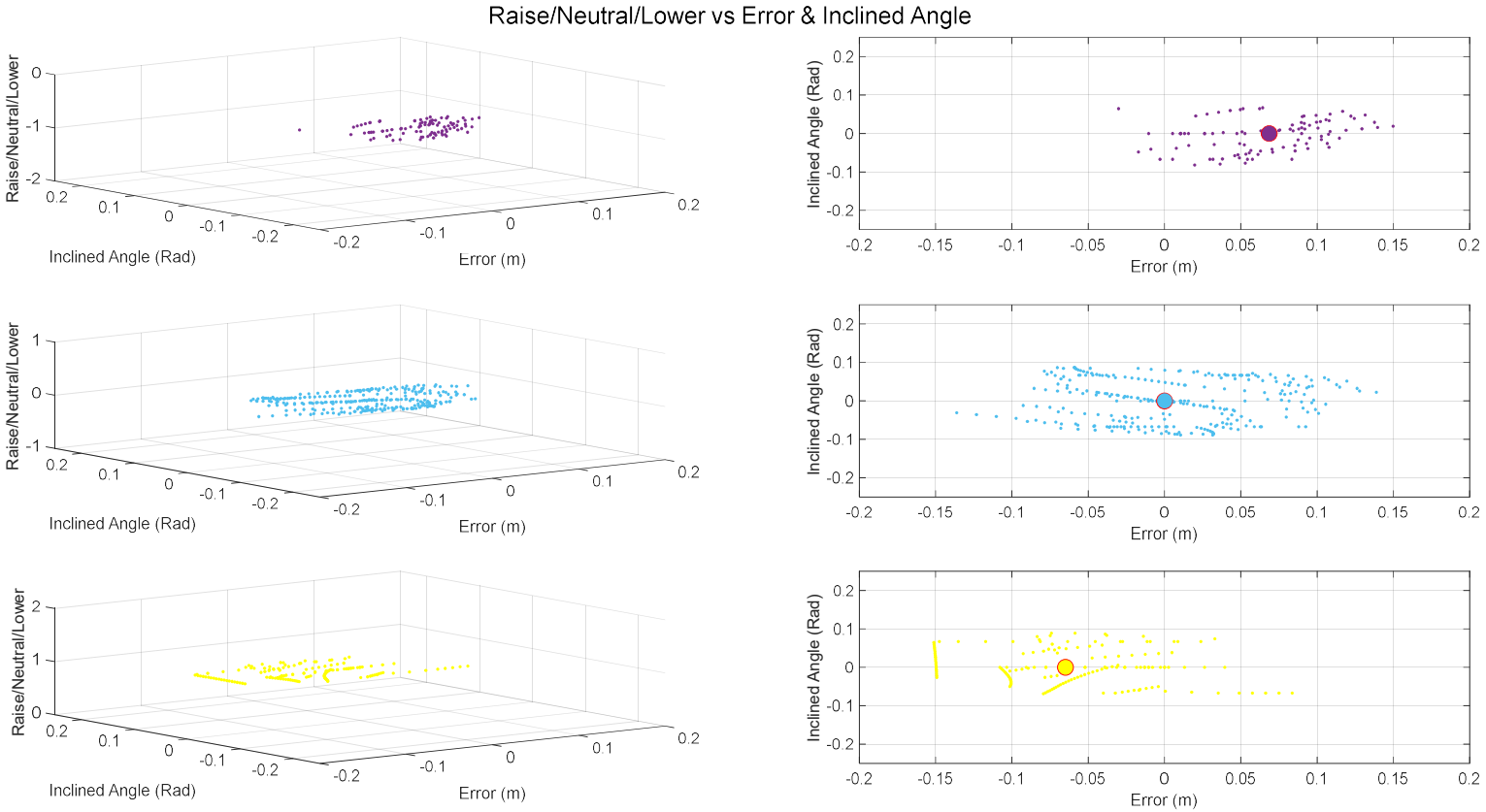

Figure 4.42: Fuzzy C-Mean Centers for the Unedited Noisy Data

The centers for raise, neutral, and lower data groups are merged into one 2-dimensional graph in Figure 4.43. Note that the Voronoi separation threshold lines are like the edited clean data.

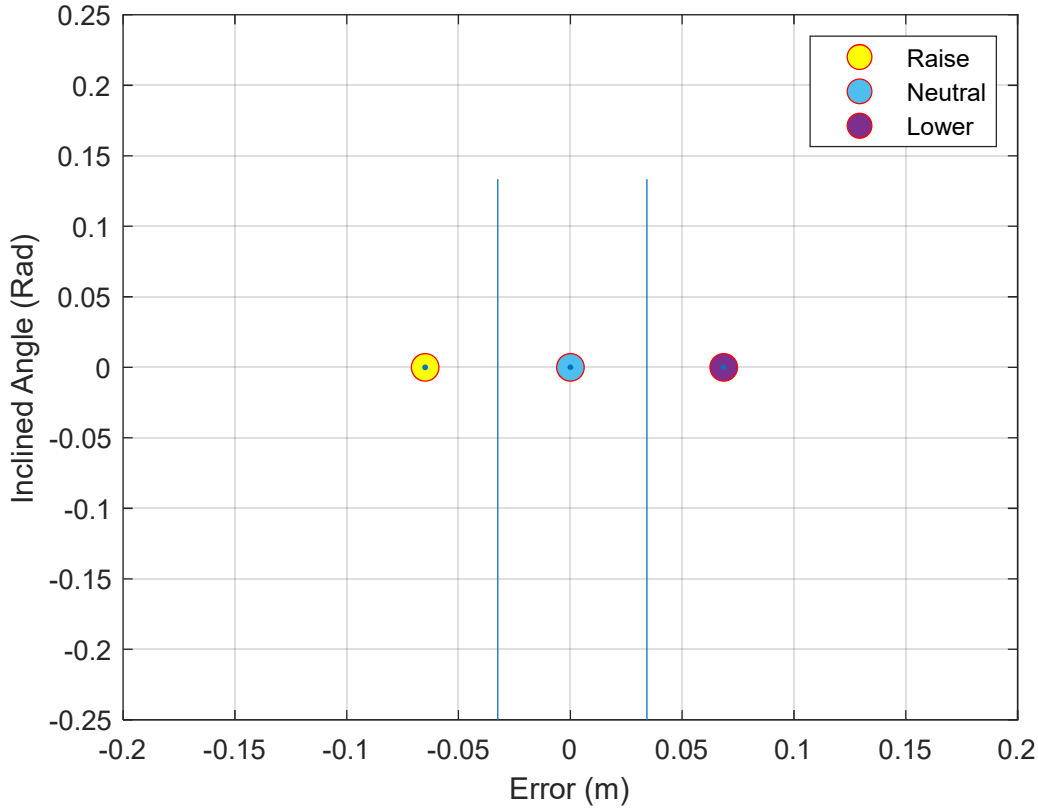

Figure 4.43: Voronoi of FCM Centers of the Unedited Noisy Data

Both FCM and Voronoi outputs of clean and unedited noisy data suggest that this method can also be used to process noisy data such as the tractor ground leveling application; However, without having to feed these two outputs back to the simulation model, it can be deduced that K-mean clustering will not yield as accurate of result compare

to TFNN due to the threshold line being further away from the neutral point. Further research can be done to investigate having multiple centers in both data to move the threshold line closer to the neutral point to improve the result.

CHAPTER FIVE

TFNN AND REAL-LIFE TRACTOR LEVELING

To partially realize the simulation done in the previous sections on a tractor, a tractor was set up to collect tractor Controller Area Network (CAN) data. The CAN data contain inputs and outputs shown in Figure 5.1 below. Engine load and ground speed data can be additional system inputs to take advantage of the various signals available on the vehicle CAN network.

Longitude, latitude, and altitude data are needed to calculate the ground reference line/position highlighted in yellow below. Longitude, Latitude, Altitude, Inclined angle (θ), and implement angle (β) data are needed to calculate the Leveler Tip Position highlighted in green. The Leveling Error can be derived from the difference between the Leveler Tip and Ground Reference positions. The leveler output can be obtained from the CAN message that shows the valve stepper position, whether it commands Raise, Neutral, or Lower.

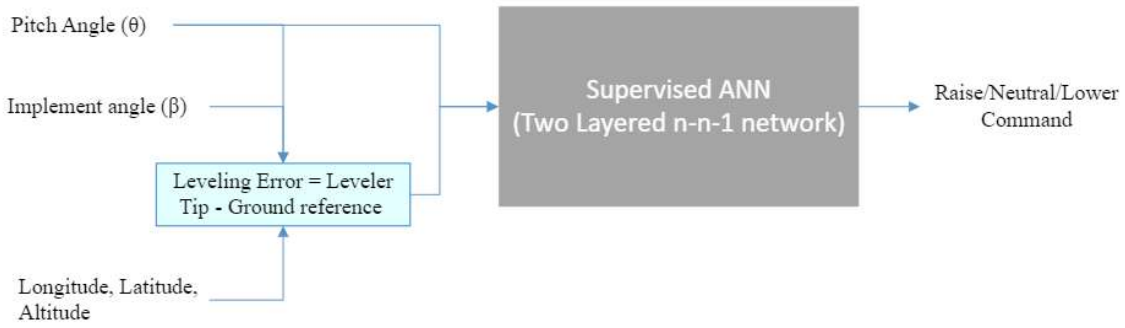

Figure 5.1: Block Diagram of Data Gathering Tractor

5.1 Tractor Ground Leveling – Tractor Set-up

A tractor set-up fitted with a GPS device, a John Deere StarFire 6000 model on the front center of the roof, and CAN data collection while driving around a test facility.

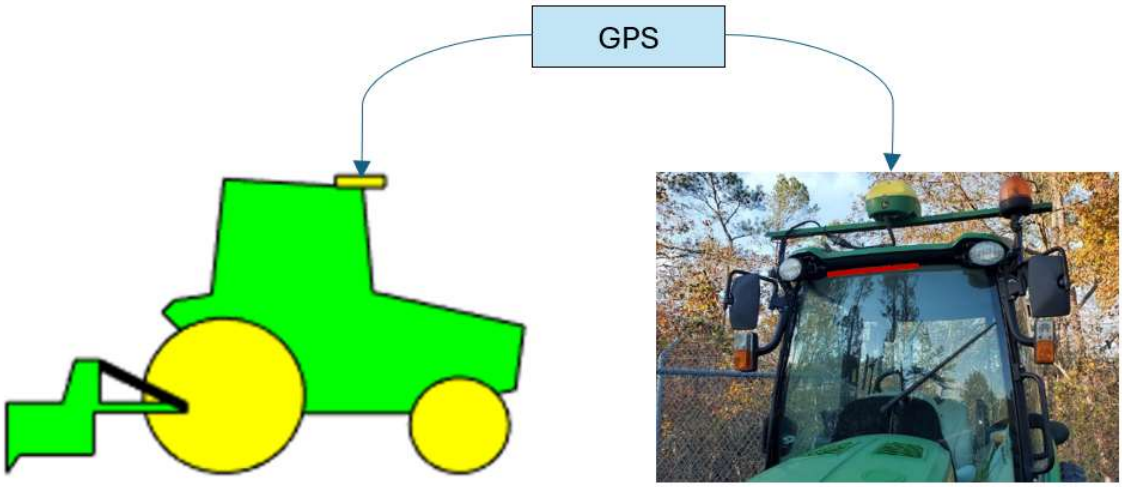

Figure 5.2: Tractor Fitted with John Deere StarFire 6000 GPS

A box blade scraper was also fitted at the rear of the tractor for regular leveling applications. The picture in Figure 5.3 below shows the scraper blade before it is mounted on the tractor. In a compact utility tractor application, the OEM implementation can typically be obtained via a partnership with a third-party supplier. Many aftermarket solutions can also be purchased at relatively low prices. However, the product quality can also be suspect when not as much testing or validation was put into the product to ensure consistent quality and durability.

Figure 5.3: Box Blade Scraper Attachment

The picture in Figure 5.4 below is the tractor configuration once the box blade attachment is installed. The three-point hitch mount is an industry standard that various off-road manufacturers adopt to ensure consistency in attachment mounting and design. The three-point hitch consists of a top link and two lower lift arms, one on each side. The hydraulic system powers the two lower lift arms. They provide lifting, lowering, and even tilting to the arms. The upper link arm can be moved, but the tractor's hydraulic system does not usually power it.

Figure 5.4: Tractor View After Rear Box Blade Attachment is Installed

Once the rear box blade scraper was installed, the tractor was driven around the test facility to collect data. A data collection laptop was connected to the data collection connector via a CAN bus port. The driving route is highlighted with the red trace around the loop, which started and ended at the same point shown in Figure 5.5. The two hills within the route taken were highlighted in the figure to enable the comparison to the data taken for verification purposes.

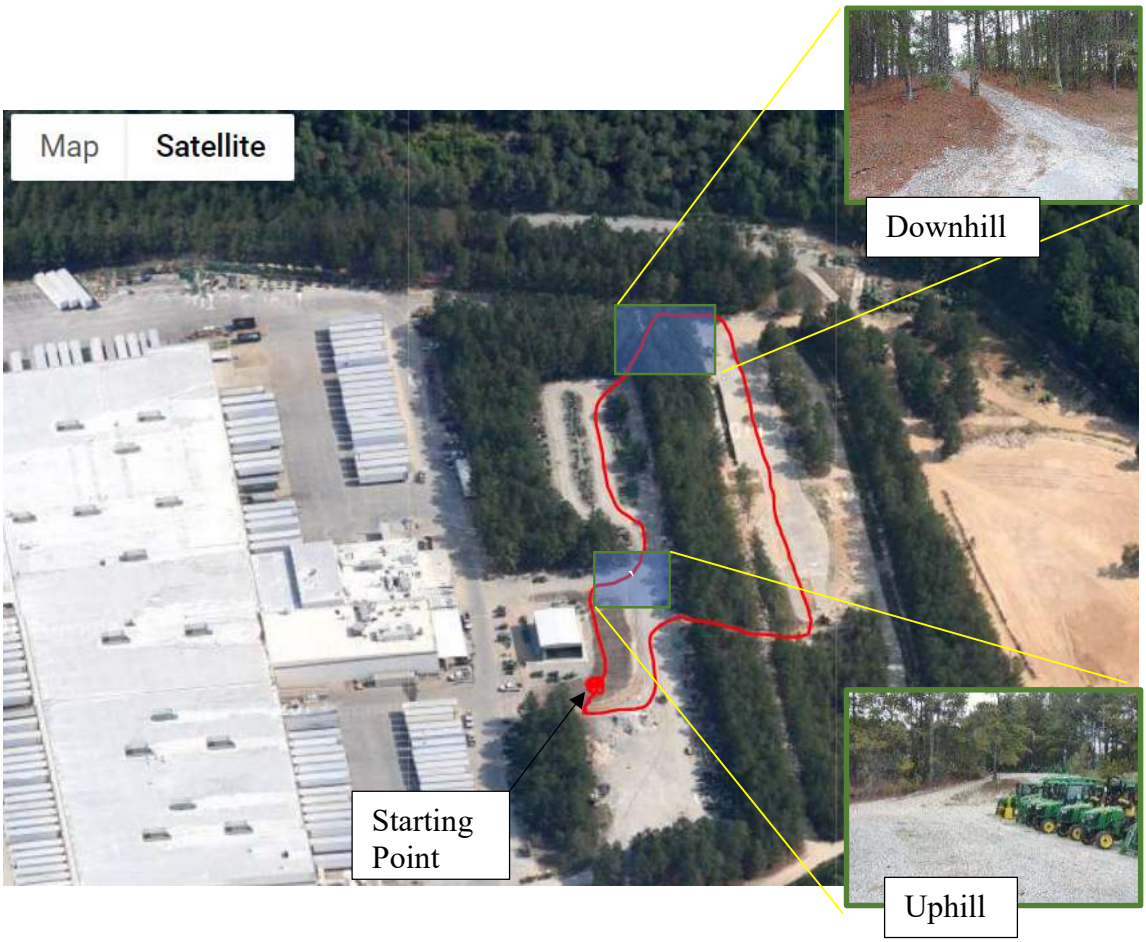

Figure 5.5: Tractor GPS Function Verification Test Site

CAN data was collected and analyzed using Vector CANalyzer software. Data analyzed is graphed in Figure 5.6. The GPS coordinate data was put into https://maps.google.com to cross-check the accuracy of the location and was confirmed to be good. The confirmed position is shown in Figure 5.7 with a red highlighted dot on the map.

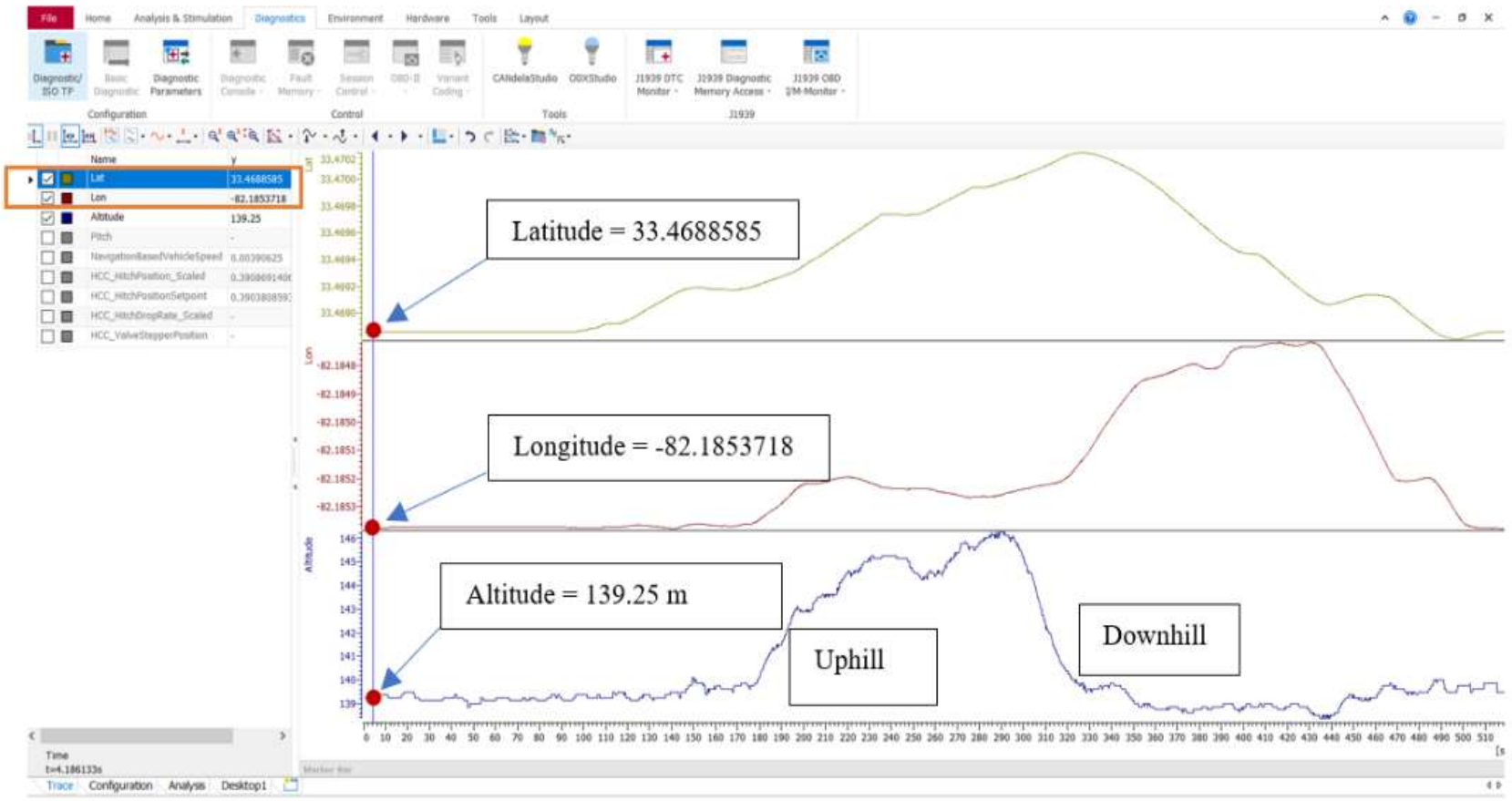

Figure 5.6: CAN Data Trace Analysis using Vector CANalyzer

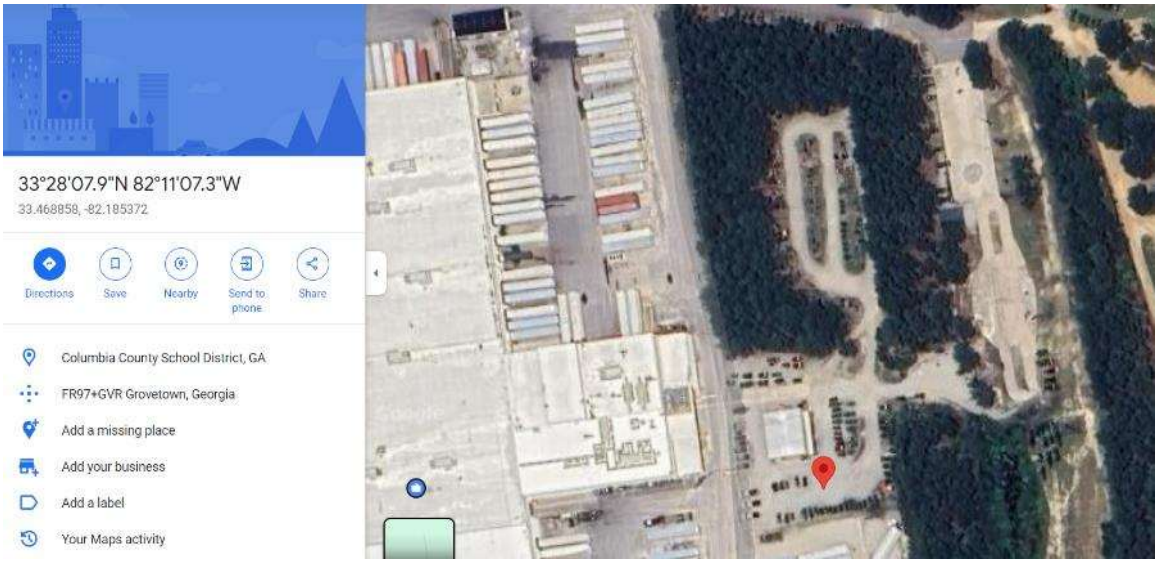

Figure 5.7: GPS Data Confirmation Highlighted in Orange Dot on Map

Decimal Places	Degrees	Distance
0	1.0	111km
1	0.1	11.1km
2	0.01	1.11km
3	0.001	111m
4	0.0001	11.1m
5	0.00001	1.11m
6	0.000001	111mm
7	0.0000001	11.1mm
8	0.00000001	1.11mm

Table 5.1: Latitude and Longitude Resolutions at Equator

The distance between two points can be calculated using the conversion website: https://www.omnicalculator.com/other/latitude-longitude-distance. The resolution of Latitude and Longitude at the equator with decimal places are provided in the Table 5.*1*. [57]

The accuracy of John Deere Starfire 6000 with an SF3 subscription is ±3 cm (1.2 in). With RTK (Real Time Kinematic), it is improved to ±2.5 cm (1-in), a 17 percent improvement. [58]

The StarFire 6000 unit also allows the tractor to gather tractor rotational tilt about the three-perpendicular axis (roll, pitch, yaw). The pitch data used for machine learning is gathered and sent out in CAN data format. The picture below shows the tractor pitching forward and backward:

Figure 5.8: Tractor Pitches Backward (left), Tractor Pitches Forward (right)

The definitions of pitch, roll, and yaw are consistent with the orientations specified in SAE J670, which is also shown in the figure below. Pitch is the rotation of a vehicle about the transverse axis. Roll is the rotation of a vehicle about the longitudinal axis. Yaw

is the rotation of a vehicle about the vertical axis. [59] The depiction of pitch, roll, and yaw can be seen in Figure 5.9 below:

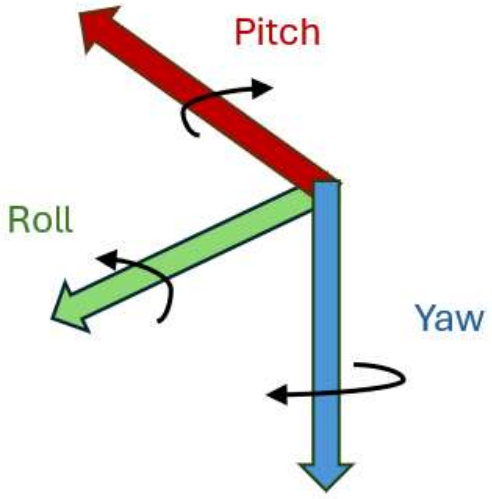

Figure 5.9: Pitch, Roll, and Yaw Illustration

5.2 Tractor Ground Leveling – Real Data Gathering

After the tractor set-up is completed, CAN message set-ups are needed and logged while running the tractor in intended operation for Neural Network training. The CAN message log file .asc format must be converted to MATLAB recognizable .MAT file is used as input to the Neural Network training routine. First, the .asc format CAN log file is converted to the .CSV file using CANalyzer's Tool/Log File conversion feature. The .CSV is then converted to the MATLAB readable .MAT file using a MATLAB unique conversion code.

The CAN log .asc format is shown in Figure 5.10 below that conforms to SAE J1939 protocol. The J1939 protocol is a set of standards created by the Society of

Automotive Engineers (SAE) to define how Electronics Control Unites (ECUs) transmit data over the CAN bus communication line. J1939 protocol applies to heavy-duty vehicles such as trucks, buses, tractors, and industrial machinery. [61] Figure 5.10 shows the data format for the CAN log, with the first column as the time stamp in seconds down to six decimal places of accuracy. The second column is the CAN bus number. In each off-road vehicle, there could be more than one CAN bus forming communication clusters within the controllers' ecologies. Typically, there would be at least a vehicle CAN bus. If there are smart implement devices such as a smart sprayer, spreader, or loader, there would be an implement CAN bus that communicates within the cluster of implement controllers.

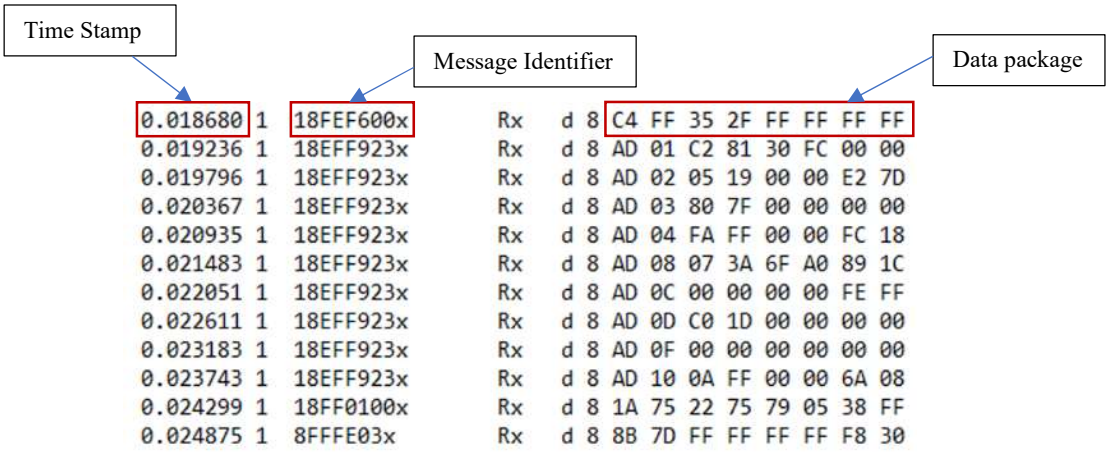

Figure 5.10: CANLog in .asc Format

The message identifier consists of message priority, one reserved bit, and one data page bit, followed by a 16-bit Parameter Group Number (PGN) and the source address shown below.

Priority	Reserved	Data Page	PDU Format	PDU Specific	Source Address
3 bits	1 bit	1 bit	8 bits	8 bits	8 bits

Table 5.2: CAN Identifier Layout

The final .MAT extension files are in the binary data container format that the MATLAB program uses. MathWorks developed the .MAT files, which are categorized as data files that include variables, functions, arrays, and other information. [62] The data gathering session was conducted in a field set-up to hold Longitude steady while changing the Latitude and Altitude only. The data gathering field can be seen below in Figure 5.11.

Figure 5.11: Tractor Field Data Gathering

Figure 5.12: Inside the Data Gathering Tractor CAB

5.3 Real tractor data analysis for Machine Learning Training

To perform machine learning with the theory established in Chapter Three, we need data that contains Tractor Pitch Angle (θ) and Leveling Error (E) as inputs, and Tractor Hitch or Leveler, i.e., Raise/Neutral/Lower information as the output (u). Data gathered and described in the previous sections were analyzed and converted into a CSV file, and additional columns were added based on real-life measurements below:

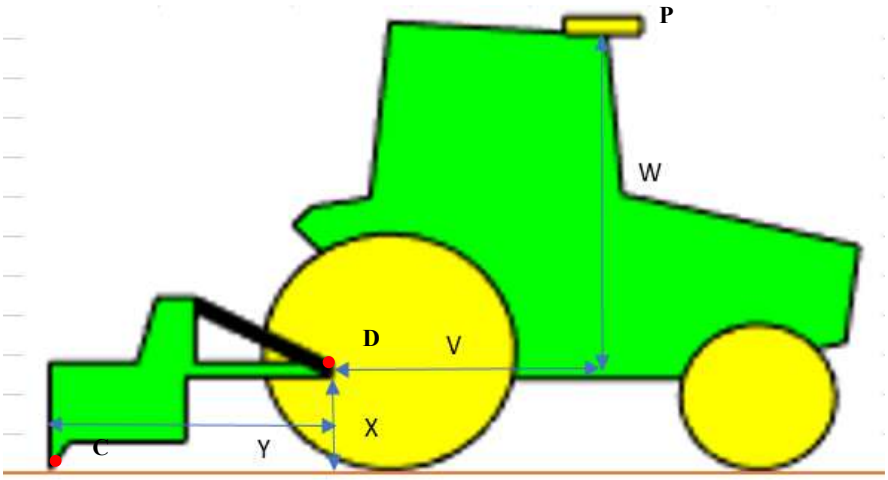

Figure 5.13: Rear Tractor Dimensions

Using the equations $\mathbf{D} = [(P_x - \overline{PD}.\sin(\delta - \theta)), (P_y - \overline{PD}.\cos(\delta - \theta))]$ and $\mathbf{C} = \mathbf{D} - [(\overline{CD}.\cos(\theta + \beta)), (\overline{CD}.\sin(\theta + \beta))]$ developed in Chapter Four, points **D** and **C** can be found. Tractor pitch data is available through the GPS unit, and the leveling error can be obtained through the difference between **C** and the reference line. The reference line can be obtained using the calibration process detailed in Figure 4.1.3 by taking coordinate and altitude snapshots at each field's start and finish points and then drawing a line. The Leveler output information can be obtained through the CAN bus information shown in Figure 5.14, which highlights Leveler Raise/Neutral/Lower.

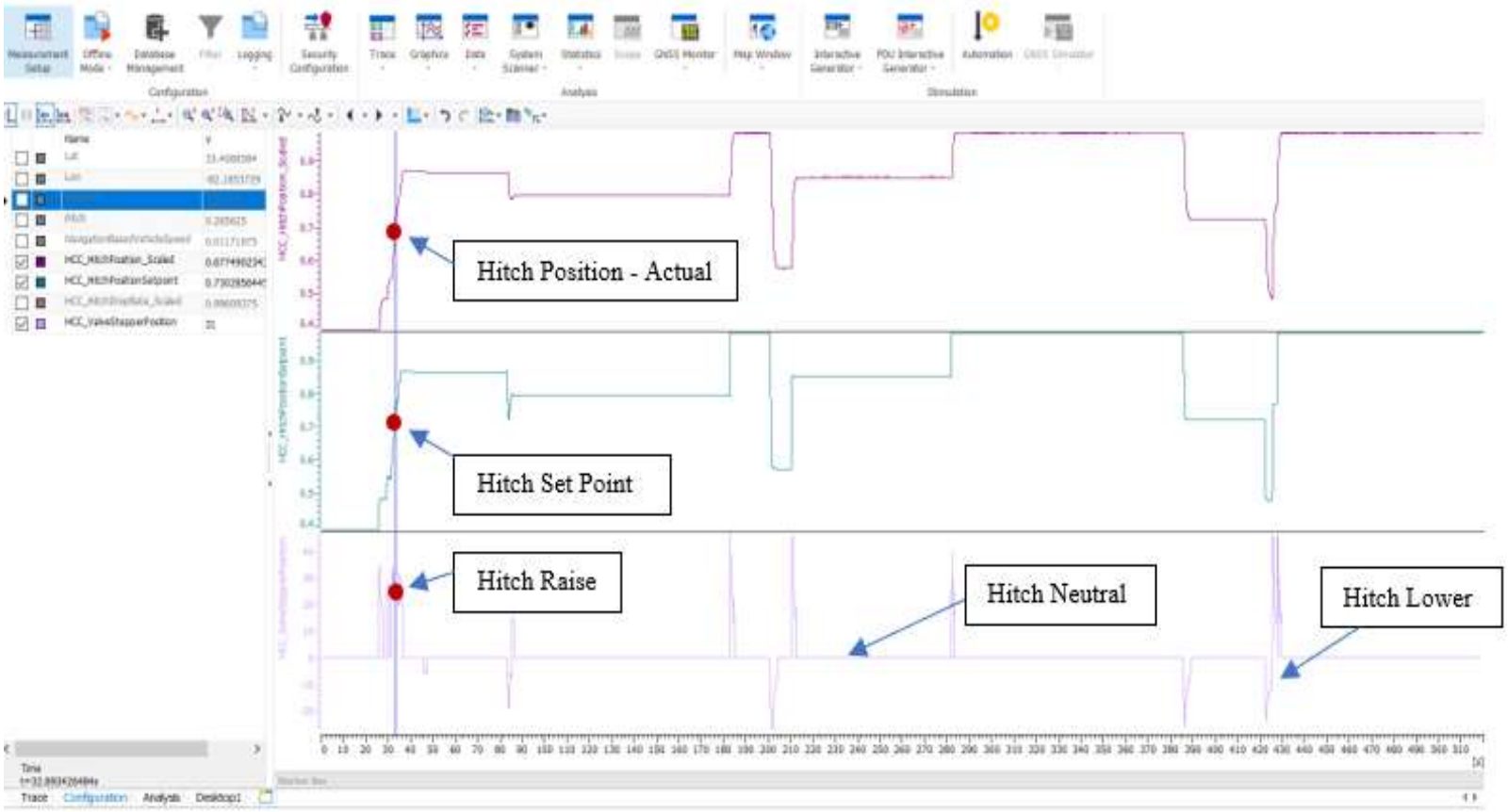

Figure 5.14: Hitch Actual Position, Hitch Set Point, and Hitch Output

The information outlined above completes the necessary inputs and output requirements for the machine learning training. The data are tabulated and can be seen in Figure 5.15. The required inputs and output are in red, and columns are highlighted in green.

Hitch Output	Altitude[m] E_y	Pitch[Deg] θ	Lat[Deg] E_x	Lon[Deg]	Angle Beta β (Degree)	Ground Reference	C'_y	Leveling Error
1	144.875	0.11319	33.469203	-82.183578	20.388	0.561	0.473	-0.089
1	144.875	0.101471	33.469203	-82.183578	20.394	0.561	0.473	-0.088
1	144.875	0.099081	33.469203	-82.183578	20.396	0.561	0.473	-0.088
1	144.875	0.114706	33.469202	-82.183578	20.399	0.564	0.472	-0.092
1	144.875	0.13033	33.469202	-82.183578	20.399	0.564	0.471	-0.093
1	144.875	0.145955	33.469202	-82.183578	20.399	0.564	0.471	-0.093
1	144.875	0.194225	33.469201	-82.183578	20.397	0.567	0.468	-0.099
1	144.875	0.305542	33.469201	-82.183578	20.394	0.567	0.463	-0.104
1	144.875	0.416859	33.469201	-82.183578	20.392	0.567	0.458	-0.109
1	144.875	0.528176	33.4692	-82.183578	20.376	0.570	0.453	-0.117
1	144.875	0.570944	33.4692	-82.183577	20.356	0.570	0.451	-0.119
1	144.875	0.481084	33.4692	-82.183577	20.342	0.570	0.456	-0.114
0	144.875	0.391225	33.4692	-82.183577	20.328	0.570	0.461	-0.109
0	144.875	0.301365	33.469199	-82.183577	20.319	0.573	0.465	-0.107
0	144.875	0.22485	33.469199	-82.183577	20.306	0.573	0.469	-0.103
0	144.875	0.174071	33.469199	-82.183577	20.299	0.573	0.472	-0.101
0	144.875	0.123292	33.469199	-82.183577	20.296	0.573	0.474	-0.098
0	144.875	0.072513	33.469198	-82.183577	20.296	0.575	0.477	-0.098
0	144.885664	0.033064	33.469198	-82.183577	20.297	0.575	0.489	-0.086
0	144.916914	0.015486	33.469198	-82.183577	20.296	0.575	0.522	-0.054
0	144.948164	-0.002092	33.469198	-82.183577	20.292	0.575	0.554	-0.021
0	144.979413	-0.01967	33.469197	-82.183577	20.294	0.578	0.586	0.008
0	145	0.020738	33.469197	-82.183577	20.295	0.578	0.604	0.026
0	145	0.173099	33.469197	-82.183577	20.289	0.578	0.597	0.019
0	145	0.325459	33.469196	-82.183577	20.220	0.581	0.592	0.010
0	145	0.47782	33.469196	-82.183577	20.099	0.581	0.587	0.006
-1	145	0.582128	33.469196	-82.183577	20.005	0.581	0.585	0.003
-1	145	0.593845	33.469196	-82.183577	19.959	0.581	0.585	0.004
-1	145	0.605562	33.469195	-82.183577	19.927	0.584	0.586	0.002
-1	145	0.617278	33.469195	-82.183577	19.850	0.584	0.587	0.003
-1	145	0.648981	33.469195	-82.183577	19.737	0.584	0.589	0.004
-1	145	0.719311	33.469194	-82.183577	19.636	0.587	0.588	0.001
-1	145	0.789641	33.469194	-82.183577	19.548	0.587	0.587	0.000
-1	145	0.859971	33.469194	-82.183577	19.502	0.587	0.584	-0.003

Figure 5.15: CAN Data Formatted to Facilitate TFNN Training

The tabulated information above is plotted into graphs that show the two inputs and one output in Figure 5.16. Because the leveling height was constantly influenced by the terrain soil that pushed against the implement during data collection, the actual leveling

height was often higher than the set point. This prompts the output to constantly command lower to counter the force that pushed the leveling implement up. It was also noticed that the data-gathering session was conducted by eyeballing the terrain and estimating the error rather than relying on accurate knowledge of actual errors. Therefore, the data will need further control and fine-tuning to be ready for TFNN training. Additional research will have to be done to have a real time error monitor as the data gathering is taking place to determine better when to command the output to raise, lower or leave at neutral; However, this data gathering does show that it is possible to gather tractor data that can be used to feed into the simulation model to realize the theoretical TFNN results. Namely, the two inputs are the leveling error and the pitch angle, and the hitch or leveler output is the sole output.

Further tractor research will involve a real-time monitor or indicator that prompts the user to raise, do nothing, or lower the rear hitch. This real-time operator prompt will be based on leveling error in Figure 5.14 to determine when to adjust the hitch outputs that determine the leveler height.

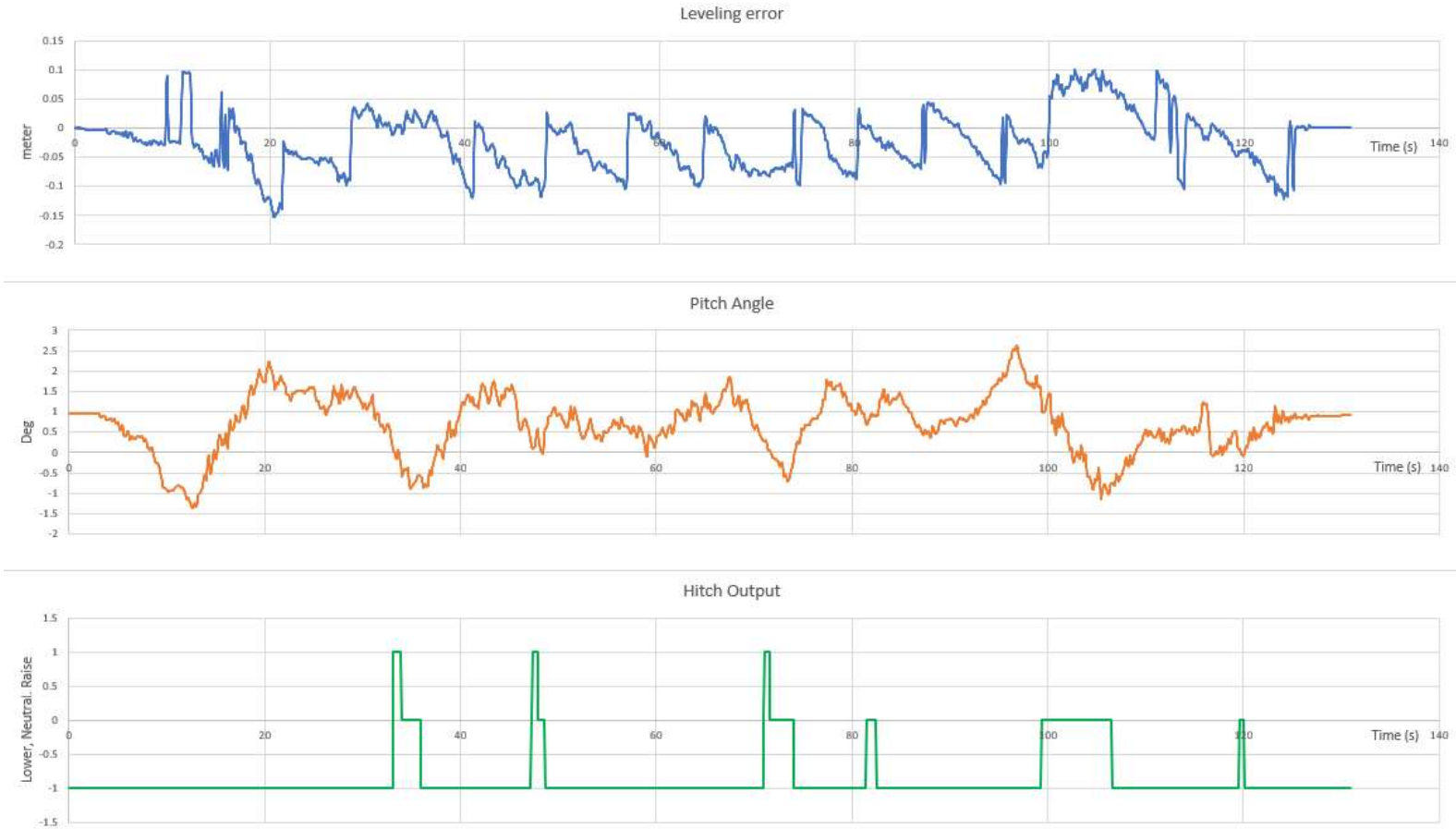

Figure 5.16: Real Life Tractor Data Collection Graph Plot

CHAPTER SIX

SUMMARY, CONTRIBUTION, AND CONCLUSION

The dissertation began by citing the advancement in machine learning in the automotive industry in a paper published by Dean A. Pomerleau's paper on ALVINN (An Autonomous Land Vehicle in a Neural Network) in 1989. Today, we have varying autonomy in automotive technology, including ADAS (Advanced Driver-Assistance Systems) and AV (Autonomous Vehicle). ADASs still require human input; they only seek to enhance driver safety rather than replace it. Features such as adaptive cruise control, lane keep assist, forward collision warning, automatic emergency braking, blind spot detection, etc., are all ADAS-enhancing features. The operator still needs to assume some driving responsibility. [63] A fully automated vehicle requires no input from the driver. It can handle every part of the driving from start to finish without the help of a driver. Drivers can use driving time to watch movies, read books, or take backseat naps. Furthermore, machine learning can use every mile driven to get smarter continuously. Examples of AV are robotaxis and robots. It could eliminate accidents related to human errors and traffic delays due to human drivers' inefficiency. [64]

The agricultural machine space's autonomy lags due to a lack of development focus. This dissertation aims to take advantage of the latest developments in machine learning to improve the automation aspect of tractor operation, namely automated ground leveling. This dissertation is based on the deficiency in supervised machine learning, that supervised machine learning tends to copy the exact pattern of the training data, the accuracy of which may be hampered by the person-in-the-loop or the noisy environment in which the data

was collected. This dissertation made contributions by improving FNN's performance via TFNN.

6.1 Summary

The accuracy of supervised machine learning training results is highly dependent on the training data fed into the training systems. Inaccurate training data would lead to erroneous training outcomes. Unfortunately, operator errors, environmental noise, and machine system disturbances contribute to the overall inaccuracy in many operator-dependent operations. Applications that include target tracing and tractor ground leveling are the two relevant examples. By applying thresholds to FNN to arrive at TFNN, it was proven that the algorithm can effectively improve the outcome of the training when compared to the currently established supervised machine learning methods.

Chapter Three lays out the theoretical illustration of the basic theory behind TFNN by establishing the mathematical framework. The generalized mathematical representation of the idea can be applied to many applications using *q-1* thresholds with *q* quantized output levels. The optimum thresholds can be found by optimizing the thresholds for feedforward neural networks and calculating the RMS errors and other systems' durability considerations. Optimizing methods can be exhaustive search, gradient search, or Lavenberg-Marquardt algorithm (LMA) methods.

Chapter Four applies the TFNN theory to Tractor Automated Ground Leveling (TAGL) application. The application uses ground leveling error and inclined angle as the two inputs, and raise, neutral, and lower as the three quantized output levels. A virtual simulation plant model was constructed to collect simulation data for TFNN training.

SVM, LVQNN, and NB supervised machine algorithms were also used to train the simulation data. Two data sets were used: one dataset was edited clean data with clean separation, and another unedited human-in-the-loop noisy data without clear separation. TFNN demonstrated superior results among all four supervised machine learning algorithms.

Chapter Five details the efforts to gather data from a real tractor in an actual field environment. Tractor was fitted with a John Deere 6000 GPS, and real tractor dimensions were considered. GPS signals are translated into the two inputs, i.e., the tractor pitch angle and the leveling error calculated. The tractor output level data of hitch or leveler raise, neutral, and lower commands were also gathered and summarized in Figure 5.15. Further elaborate tractor software needs to be constructed to show the actual error in real time, and data similar to the virtual data gathering needs to be gathered to complete the actual tractor simulation work. The real tractor data gathering proves that tractor automated ground leveling has the necessary inputs and output information for the TFNN framework proposed in this dissertation.

6.2 Contribution and Conclusion

This dissertation's first and primary contribution is a successfully developed TFNN theory that improves performance over several other methods by optimizing the threshold for feedforward neural networks to form classification outputs. This is particularly effective in applications that contain human, environmental, and system disturbance errors.

Secondly, this dissertation contributed by detailing a virtual simulation of a Tractor Automated Ground Leveling (TAGL) application and successfully applying the theoretical

equation to the MATLAB Simulink model to accomplish a TAGL framework that was achieved using TFNN.

Thirdly, this dissertation simulated a noisy and a clean data set to demonstrate the effect of noise on supervised machine learning. Qualitative simulation data using the TAGL application were generated and compared with SVM, LVQNN, and NB. The results were summarized and presented. The simulation results demonstrated superior performance with TFNN over other Supervised Machine Learning algorithms.

Fourth and finally, this dissertation collected real-life tractor data to pave the way for the real-life application of TFNN. The collected data shows that the correct data of achieving an automated tractor ground leveling was successfully acquired and can be realized with further work beyond this dissertation's scope.

Further work will be focused on tractor application software development that enables real-time ground leveling error indication on the operator display to instruct the ground leveling operator to raise, lower, or stay neutral on the hitch control command. This will result in meaningful data being collected to match simulation data. More applications with similar noisy environments outside of TAGL can be explored, and data can be gathered to take advantage of the TFNN theory. For example, a tractor application that requires additional inputs, such as engine loads, engine RPMs, etc. Tractor applications that can benefit from TFNN have discrete outputs such as loader work, grass cutting, snow blowing, and debris removal using a grapple that also operates in noisy environments.

REFERENCES

[1] G. Wiederhold and J. McCarthy, "Arthur Samuel: Pioneer in Machine Learning," in IBM Journal of Research and Development, vol. 36, no. 3, pp. 329-331, May 1992

[2] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress," in IBM Journal of Research and Development, vol. 11, no. 6, pp. 601-617, Nov. 1967

[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," in IBM Journal of Research and Development, vol. 44, no. 1.2, pp. 206-226, Jan. 2000

[4] "Machine learning, explained" [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[5] "Fundamentals of Neural Networks" [Online]. Available: https://www.wandb.com/articles/fundamentals-of-neural-networks

[6] Pomerleau, Dean A., "An autonomous land vehicle in neural network". In Advances in Neural Information Processing Systems (NIPS), 1989.

[7] S. Perla, N. N. K and S. Potta, "Implementation of Autonomous Cars using Machine Learning," 2022 International Conference on Edge Computing and Applications (ICECAA), Tamilnadu, India, 2022, pp. 1444-1451,

[8] "Machine Learning in Agriculture" [Online]. Available: https://medium.com/sciforce/machine-learning-in-agriculture-applications-and-techniques-6ab501f4d1b5

[9] M. Song, M. Kabir, S. Chung, Y. Kim, J. Ha, K. Lee, "Path planning for autonomous lawn mower tractor". CNU Journal of Agricultural Science, Vol. 42, No. 1, pp. 63-71, March 2015.

[10] Z. Yao, C. Zhao, T. Zhang, "Agriculture machinery automatic navigation technology". iScience 27, 108714, Feb. 16, 2024.

[11] W. Lu, Y. Wei, J. Yuan, Y. Deng, and A. Song, "Tractor assistant driving control method based on EEG combined with RNN-TL deep learning algorithm". IEEE Access, Vol. 8, pp. 163269 – 173279, 2020.

[12] "2024 NFMS snapshot: Mahindra's new line of small tractors" [Online]. Available: https://www.agdaily.com/news/2024-nfms-mahindra-new-line-small-tractors/

[13] “Classification in Machine Learning: An Introduction” [Online]. Available: https://www.datacamp.com/blog/classification-machine-learning

[14] N. V. Thinh, A. N. T. Y. Nhi, T. G. Huy, N. Hoang Khoa and N. T. Cam, "Fruits classification by using machine learning - An experiment using popular approaches on local data," 2021 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT), Soyapango, El Salvador, 2021, pp. 1-6

[15] W. Astuti, S. Dewanto, K. Soebandrija, and S. Tan, “Automatic fruit classification using support vector machines: a comparison with artificial neural network”, 2018 IOP Conf. Ser.: Earth Environ. Sci. Volume 195 012047

[16] Mel Keytingan M. Shapi, Nor Azuana Ramli, Lilik J. Awalin, "Energy consu mption prediction by using machine learning for smart building: Case study in Malaysia, Developments in the Built Environment", Volume 5, 2021, 100037, ISSN 2666-1659

[17] A. Behera and A. Chinmay, "Stock Price Prediction using Machine Learning," 2022 International Conference on Machine Learning, Computer Systems and Security (MLCSS), Bhubaneswar, India, 2022, pp. 3-5

[18] Z. Liu, Z. Dang and J. Yu, "Stock Price Prediction Model Based on RBF-SVM Al gorithm," 2020 International Conference on Computer Engineering and Intelligent Control (ICCEIC), Chongqing, China, 2020, pp. 124-127

[19] “Supervised vs Unsupervised Learning, Explained” [Online]. Available: https://www.sharpsightlabs.com/blog/supervised-vs-unsupervised-learning/

[20] J. B MacQueen, "Some Methods for classification and Analysis of Multivariate Observations", Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281-297, 1967.

[21] L. Kaufman and P.J Rousseeuw, "Clustering by means of Medoids" in Statistical Data Analysis Based on the Norm and Related Methods, North-Holland, pp. 405-416, 1987.

[22] J. Jia and W. Wang, "Review of reinforcement learning research," 2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC), Zhanjiang, China, 2020, pp. 186-191

[23] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers [J]. IBM Journal. 1967, pp.601-617

[24] Zhang Rubo, Zhou Ning, Gu Guochang, etc., ”Research on Intelligent Robot Collision Avoidance Method Based on Reinforcement Learning”, ”Robot”, 1999, pp.204-209.

[25] L. Lyu, Y. Shen and S. Zhang, "The Advance of Reinforcement Learning and Deep Reinforcement Learning," 2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA), Changchun, China, 2022, pp. 644-648

[26] "Artificial Intelligence (AI) vs. Machine Learning vs. Deep Learning" [Online]. Available: https://skymind.ai/wiki/ai-vs-machine-learning-vs-deep-learning

[27] "Deep learning vs. machine learning: What's the difference between the two?" [Online]. Available: https://www.digitaltrends.com/cool-tech/deep-learning-vs-machine-learning-explained/2/

[28] "Deep Learning vs Machine Learning – the essential differences you need to know!" [Online]. Available: https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/

[29] "GPUs vs CPUs (Threads, core, speed)" [Online]. Available: https://medium.com/@abhishekjainindore24/gpus-vs-cpus-threads-core-speed-75b5732ce389

[30] "Training SVM classifier with HOG features" [Online]. Available: https://www.kaggle.com/code/manikg/training-svm-classifier-with-hog-features

[31] "Introduction to Deep Learning: Machine Learning vs Deep Learning" [Online]. Available: https://www.youtube.com/watch?v=-SgkLEuhfbg

[32] "Understanding Alexnet" [Online]. Available: https://sushscience.wordpress.com/2016/12/04/understanding-alexnet/

[33] Marius Bjarski, et al., "End to End Learning for Self-Driving Cars," 2016 NVIDIA Corporation. [Online]. Available: https://devblogs.nvidia.com/deep-learning-self-driving-cars/

[34] Jinwei Gu, et.al, "AI Co-Pilot: RNNs for Dynamic Facial Analysis" 2017 NVIDIA Corpration. [Online]. Available: https://devblogs.nvidia.com/ai-co-pilot-rnn-dynamic-facial-analysis/

[35] Picture [Online]. Available: https://www.evergladesfarmequipment.com/blog/how-to-use-a-box-blade/

[36] Picture [Online]. Available: https://www.deere.com/en/attachments-accessories-and-implements/utility-tractors-attachments-accessories/landscape-equipment/bb50l-series/

[37] "What is logistic regression?" [Online]. Available: https://www.ibm.com/topics/logistic-regression

[38] "Search Algorithms" [Online]. Available: https://sawtoothsoftware.com/help/lighthouse-studio/manual/search-algorithms.html

[39] "What is an exhaustive search" [Online]. Available: https://codestandard.net/articles/exhaustive-search/

[40] "Brute-force search" [Online]. Available: https://en.wikipedia.org/wiki/Brute-force_search

[41] "Gradient Descent" [Online]. Available: https://www.codecademy.com/resources/docs/ai/search-algorithms/gradient-descent

[42] "Gradient Descent" [Online]. Available: https://en.wikipedia.org/wiki/Gradient_descent

[43] Hao Yu and B. M. Wilamowski, "Levenberg Marquardt Training," Industrial Electronics Handbook, vol. 5 Intelligent Systems, 2nd Edition, chapter 12, pp. 12-1 to 12-15, CRC Press 2011.

[44] Manolis I. A. Lourakis, 2005, "A Brief Description of the Levenberg-Marquardt Algorithm Implemented by Levmar," Institute of Computer Science, Foundation fo Research and Technology – Hallas (FORTH)

[45] "Levenberg-Marquardt algorithm" [Online]. Available: https://en.wikipedia.org/wiki/Levenberg-Marquardt_algorithm

[46] Thomas Rincy N; Roopam Gupta "A Survey on Machine Learning Approaches and its Techniques" 2020 IEEE International Steudents' Conference on Electrical, Electronics and Computer Science (SCEECS), pp. 1-6

[47] R. Saravanan; Pothula Sujatha "A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification" International Conference on Inteligent Computing and Control Systems (ICICCS 2018), pp. 945-949

[48] "The Mathematics Behind Support Vector Machine Algorithm" [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/#h-1-support-vector-machine

[49] Corinna Cortes; Vladimir Vapnik "Support-Vector Networks" 1995 Machine Learning, Volume 20, pp. 273-297

[50] T. Kohonen "Self-Organizing Maps" September 1990 Proceedings of the IEEE, Vol. 78, No. 9

[51] M.T. Martin-Valdivia; L.A. Urena-Lopez; M. Garcia-Vega "The Learning Vector Quantization Algorithm Applied to Automatic Text Classification Tasks" Neural Networks 20 (2007) pp. 748-756

[52] I. Rish “An Empirical Study of the Naïve Bayes Classifier” IJCAI 2001 workshop on empirical methods in artificial intelligence, vol. 3, no. 22, 2001

[53] D.J. Hand; K. Yu “Idiot’s Bayes: Not So Stupid After All?” May 2007 International Statistical Review Vol. 69, no. 3, pp. 385-398

[54] Rosenblatt, Frank (1957). "The Perceptron—a perceiving and recognizing auto maton". Report 85-460-1. Cornell Aeronautical Laboratory.

[55] R. E. Uhrig, “Introduction to artificial neural networks,” in Proceedings of the IEEE 21st International Conference on Industrial Electronics, Control, and Instrumentation (IECON1995), vol. 1, November 1995, pp. 33–37.

[56] K. Sugihara, "Why Are Voronoi Diagrams so Fruitful in Application?," 2011 Eighth International Symposium on Voronoi Diagrams in Science and Engineering, QingDao, China, 2011, pp. 14-14

[57] “Decimal degree” [Online]. Available: https://en.wikipedia.org/wiki/Decimal_degrees

[58] “Display and receiver” [Online] Available: https://deerequipment.com/precision-ag/displays-receivers/

[59] “Vehicle Pitch, Roll and Yaw” [Online]. Available: https://www.liskeforensics.com/blog/title/vehicle-pitch-roll-and-yaw/id/249/

[60] “StarFire™ RTK Radio 900” [Online]. Available: https://www.deere.co.nz/en/technology-products/precision-ag-technology/guidance/starfire-rtk-radio-900/

[61] “The J1939 Protocol, Explained” [Online]. Available: https://www.calamp.com/blog/j1939/

[62] “MAT-File Versions” [Online] Available: https://www.mathworks.com/help/matlab/import_export/mat-file-versions.html

[63] “DriverlessEd Chapter 2: The Difference Between ADAS and Avs” [Online] Available: https://motional.com/news/driverlessed-chapter-2-difference-between-adas-and-avs

[64] “Autonomous Vehicles vs. Advanced Driver Assistance Systems” [Online]. https://www.hesaitech.com/autonomous-vehicles-vs-advanced-driver-assistance-systems/