Implementation of ANN on Microcontrollers in Line Follower Robots

Sumantri Kurniawan Risandriya[1], Alex Sandro Wijaya[2], Alex Sonang Nababan[3]

[1,2,3]Jurusan Teknik Elektro, Prodi Teknik Rekayasa Elektronika, Politeknik Negeri Batam, Batam, Indonesia

[1]sumantri@polibatam.ac.id , [2]Alexwijaya1605@gmail.com, [3]Alexnababan4646@gmail.com

***Abstract*— Line follower robots commonly use conventional control methods such as if–else logic and PID control, which suffer from limited flexibility, poor adaptability to changing track conditions, and require complex parameter tuning. This study proposes a machine learning–based control system using an Artificial Neural Network (ANN) with a feedforward architecture and two hidden layers. The ANN model is trained using supervised learning with sensor data and motor output obtained from initial simulations, then implemented on an Arduino microcontroller for real-time control. Experimental results under both bright and completely dark lighting conditions show that the ANN-based system can control the robot stably and responsively, achieving a 90% success rate in completing the test track. These results demonstrate that ANN improves navigation accuracy, motion consistency, and adaptability, while also being effectively deployable on microcontrollers with limited computational resources.**

Keywords*— *Arduino, Artificial Neural Network (ANN), Control System, Line follower robot, Machine Learning.

I. INTRODUCTION

The development of artificial intelligence (AI) technology, particularly in machine learning, has significantly improved the capabilities of automation and robotics systems. One application of this technology is in line-following robots, which can follow a specific path based on real-time sensor data. Conventional methods, such as if–else logic and PID, are still widely used but have limitations in terms of flexibility and adaptation to path changes, and they require manual tuning[13]. The Artificial Neural Network (ANN) approach, particularly the feedforward architecture, offers a more adaptive control system because it can recognize complex patterns from sensor data without explicit rules and remains efficient to implement on microcontrollers such as the Arduino.

Previous studies have demonstrated the effectiveness of ANN approaches in robot navigation[1]. Study [1] reported that Q-learning-based ANNs can improve the navigation accuracy of line follower robots. Research [2] showed that multilayer ANNs with infrared sensors can follow complex paths stably under various lighting conditions. In addition, study [3] proved that an ANN can be optimally implemented on microcontrollers with limited computing resources. Based on this, this study focuses on developing a feedforward ANN-based line follower robot control system to achieve more stable, responsive, and adaptive performance than conventional methods.

II. METHOD

A. Line Follower

To support the development of more adaptive line follower robots, several relevant studies were used as references. Research by [4] shows that PID control on STM32 can be optimized, although its performance decreases at high speeds.

[5] applies LSTM-based ANNs and CNNs that can follow complex trajectories with high accuracy. [6] proves that a multilayer ANN provides a faster, more adaptive response than a PID controller.

Additionally, [2] implemented an ANN with four infrared sensors and two hidden layers, demonstrating high accuracy under various path conditions. Conversely, a study by [7] emphasizes that the conventional if–else-based approach on the ATmega32A is optimal only at low speeds and is less adaptive to path changes. From these studies, the ANN approach has been proven to be more flexible, accurate, and adaptive, making it highly promising for use in microcontroller-based line-following robots[15].

B. Artificial Neural Network

An Artificial Neural Network (ANN) is a computing system inspired by the human brain that learns from data and makes decisions[8][9]. It consists of interconnected artificial neurons arranged in layers with weighted connections. In line-following robot control systems, ANNs enable the robot to interpret sensor data and generate appropriate actions, even under varying or imperfect path conditions[8], as shown in Fig. 1.

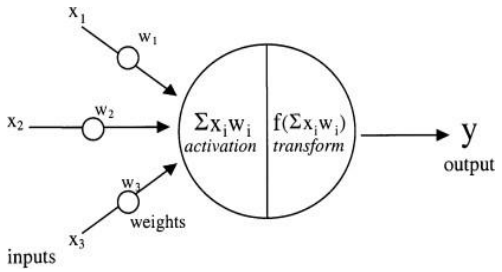

Figure 1 Model Artificial Neuron

C. Machine Learning and FeedForward

An Artificial Neural Network (ANN) is a Machine Learning algorithm that mimics the workings of biological neurons. An

ANN consists of input, hidden, and output layers, where each neuron performs weighted calculations and passes the results through an activation function, such as the sigmoid function. The training process uses backpropagation to reduce errors between predictions and targets.

The most common ANN architecture in embedded systems is the Feedforward Neural Network (FNN), where data flows in one direction from input to output without feedback[10]. Its simple, efficient design makes FNN suitable for microcontrollers such as the Arduino[11]. In line-following robots, IR sensor signals (0/1) are processed by a feedforward network to generate motor control decisions.

The ANN model is first trained on a computer, then its weights are applied to the microcontroller for real-time inference. With this approach, the robot can recognize path patterns more accurately, respond faster, and adapt more effectively, as shown in Fig. 2.

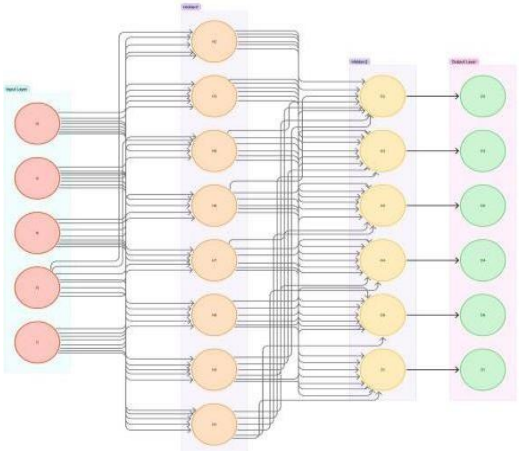

Figure 2 Arsitektur Neural Network

D. Multi Layer Perceptron (MLP) & Algorithm BackPropagation

A Multilayer Perceptron (MLP) is a type of ANN that consists of one input layer, one or more hidden layers, and one output layer. MLP is a feedforward network, meaning that signals flow from the input to the output without feedback. Each neuron in an MLP calculates Z = ∫j w x + b. The result is then passed to the activation function $a_j = f(z_j)$.

Backpropagation is the core algorithm for training MLPs. This algorithm works in two phases:

1) Forward Pass: Calculates the output based on the current weights.
2) The Backward Pass calculates the Error and adjusts the weights to reduce it.

Backpropagation minimizes the error function, for example:

$$E = (\frac{1}{2}\, y_{target} - y)^2$$

Backpropagation:

1. Initialization of Weights and Parameters. Initial weights are randomly initialized within a small range (e.g., −0.5 to 0.5). Other parameters, such as the learning rate "α", are specified.
2. Forward pass
   - Calculate Hidden Layer Output $z_j = \sum_i w_{ij} x_i + b_j$
     $y_j = f(z_j)$.
   - Calculate Output Layer $z_j = \sum_j w_{kj}\, y_j + b_k$
     $y_j = f(z_k)$
   - Calculate Error: This Error shows how far the prediction is from the target. $e_k = y_{dk} - y_k$
   - Calculate gradient error :
     - Delta on the output layer $\delta_k = e_k . y_k\ (1-y_k)$
     - Delta in the hidden layer $\delta_j = y_j(1- y_j) \sum_k w_{jk}\delta_k$
   - Update Weight: The weight is updated according to the error gradient. $\Delta w_{ij} = \propto . x_i . \delta_j$ and $w_{ij}\,(baru) = w_{ij}\,(lama) + \Delta w_{ij}$.
   - Calculate the total Error using loss functions such as MSE or entropy.

E. Activation Function

Activation functions introduce nonlinearity to the network. Without activation functions, ANNs would be simple linear models incapable of learning complex patterns. Some of the activation functions we use are:

1) ReLU (Rectified Linear Unit), commonly used in hidden layers to accelerate convergence.
   $$f(x) = \max(0, x)$$
2) SoftMax is used for multi-class classification.
   $$softmax(z_k) = \frac{e^{zk}}{\int e^{zi}}$$

In this study, the hidden layer uses ReLU, and the output uses softmax.

F. The Learning Process In ANN

ANNs can learn by adjusting their weights to make their predictions closer to the target. Learning is done by providing appropriate input and target examples (supervised learning). The learning process involves:

3) Calculating output through feedforward
4) Measuring Error
5) Returning Error backward
6) Updating weights based on Error

This learning process is repeated many times in epochs until the model achieves a small error. Training is done on a computer using TensorFlow/Keras, which automatically implements the backpropagation algorithm.

G. Implementasi FeedForward di Microcontroller

Feedforward is the process of calculating network output based on existing inputs and weights. This process is used in microcontrollers because it is lightweight and does not require learning (training). FeedForward does not have a connection back from the output to the neuron input and therefore does not store records of previous output values[10]. Feedforward stages:

1) Read input (IR sensor).
2) Multiply the input by the weights in the hidden layer.
3) Add bias.

4) Pass values to the activation function.
5) Multiply the hidden layer results by the output layer weights.
6) Generate output in the form of the class with the highest probability (softmax + argmax).

The feedforward process is very fast, making it suitable for running on an Arduino to control robot movements in real time[11][12], as shown in Fig. 3.

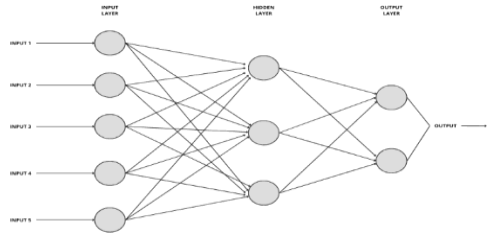

Figure 3 FeedForward Network

- Input Representation and Network Parameters. In a microcontroller, the network input consists of five infrared sensor signals that represent the condition of the line:

$$X=[x1,x2,x3,x4,x5]$$

- Hidden Layer Calculation (Weighted Sum + ReLU), Each neuron in the hidden layer calculates a linear operation:

$$h1_j = \sum_{i=1} (x_i \,.\, W1_{ij}) + b1_j$$

Then, it was activated using the ReLU (Rectified Linear Unit) function:

$$h_j^{(1)} = \max(0, \quad z_j^{(1)})$$

- Output Layer and Softmax Calculations, the output layer does not use the ReLU function, but rather produces logits, which are linear values before being converted to probabilities:

$$o_m = \sum (\mathrm{h2}_k \,.\, W3_{km}) + b3_m \;, \mathrm{K} = 1$$

Since the output is intended to represent the robot's action choices (classification), the Softmax function is used:

$$y_{m=\frac{e^{om}}{\sum_{i=1}^{6} e^{oi}}}$$

III. RESULTS AND DISCUSSION

A. Research Design

Figure 4 presents the machine learning workflow for developing an ANN-based control system for a line follower robot. The process includes weight and bias initialization, data acquisition, forward propagation, and error calculation. If the Error exceeds the predefined threshold, backpropagation is applied iteratively to update the parameters until an optimal model is achieved, which is then stored as the final trained model, as shown in Fig. 4

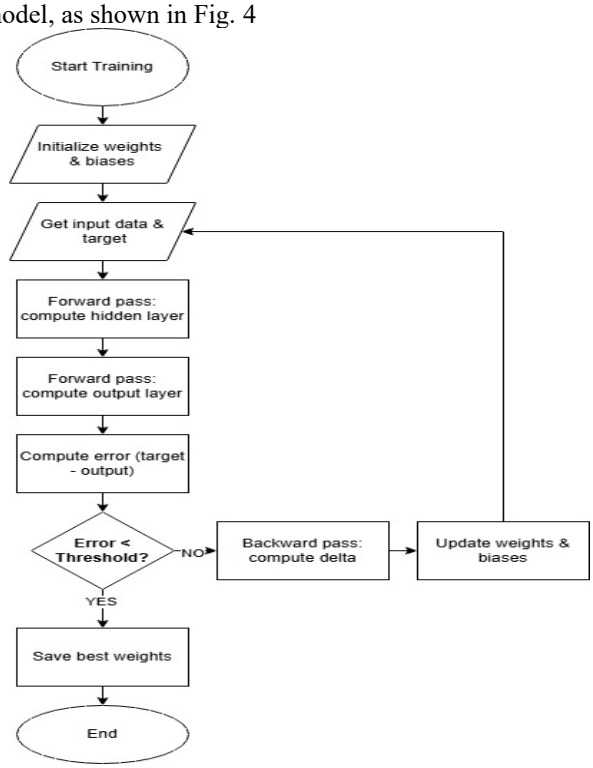

Figure 4 Flowchart Training learning

Figure 5 shows the feedforward flowchart implemented on the microcontroller. IR sensor data are processed through the hidden and output layers using weighted sums and activation functions, followed by softmax to obtain class probabilities. The robot's motion is determined by selecting the highest-probability class, and the corresponding command is sent to the motors in real time as shown in Fig. 5.

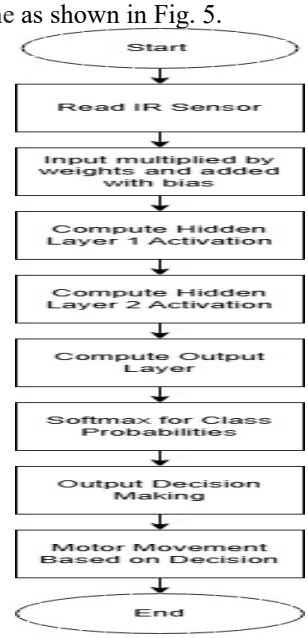

Figure 5 Flowchart FeedForward diMicrocontroller

B. Work System Design

The line follower robot system consists of two main stages: ANN training using supervised learning and feedforward implementation on a microcontroller. Sensor data and movement labels are used to train the model offline, and the resulting weights and biases are deployed on the Arduino for real-time inference. During operation, the ANN processes IR sensor inputs to directly control motor direction and speed without relying on if-else logic or PID control, as shown in Fig. 6.

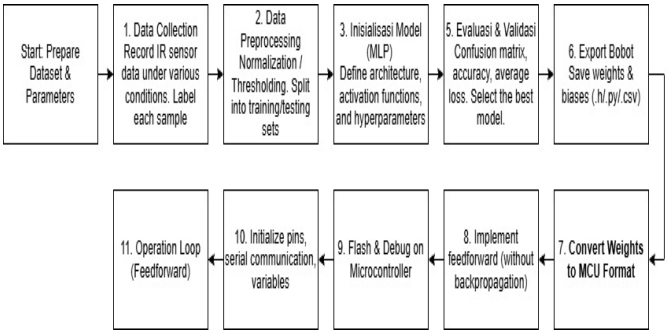

Figure 6 Block Diagram of Working System

C. Mechanical Design

The robot uses a lightweight acrylic chassis with a three-wheel configuration, consisting of two rear drive wheels and one front free wheel for balance. Sensors, microcontroller, and battery are mounted symmetrically for stable weight distribution, while IR sensors are placed at the front underside to ensure accurate path detection. This design enables smooth, stable, and responsive motion during path tracking, as shown in Fig. 7.

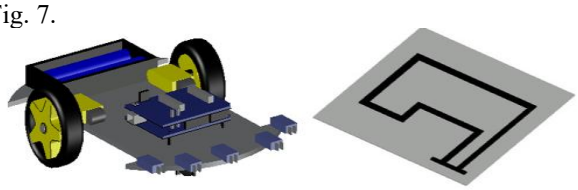

Figure 7 Design Mechanical & Design Track

D. Electrical Design

The Arduino Uno functions as the main controller, processing track data from TCRT5000 sensors using an ANN algorithm. The ANN outputs are converted into PWM signals to drive DC motors via an L298N motor driver. A 9V Li-ion battery powers the system, ensuring stable, reliable operation, as shown in Fig.8.

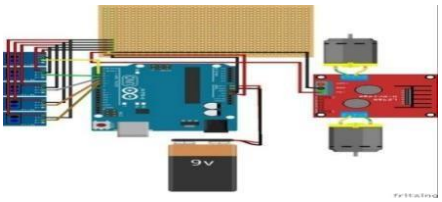

Figure 8 Design Electrical

E. Sensor Testing

At this stage, each TCRT5000 infrared sensor is tested on black and white surfaces to ensure accurate HIGH/LOW logic readings through the Arduino serial monitor before being used as inputs for the ANN, as shown in Table I.

TABLE I
SENSOR TESTING

Sensor	Sensor Status	Output (Logic)	Description
S1 (Right)	The sensor is aimed at the line	1	The sensor detects a line on the right side
S1 (Right)	The sensor is pointed at the background	0	The sensor is not detecting the line
S2	The sensor is aimed at the line	1	The sensor detects the line
S2	The sensor is pointed at the background	0	The sensor is not detecting the line
S3 (Center)	The sensor is aimed at the line	1	The sensor detects the line in the center
S3 (Center)	The sensor is pointed at the background	0	The sensor is not detecting the line
S4	The sensor is aimed at the line	1	The sensor detects the line
S4	The sensor is pointed at the background	0	The sensor is not detecting the line
S5 (Left)	The sensor is aimed at the line	1	The sensor detects the line on the left side
S5 (Left)	The sensor is pointed at the background	0	The sensor is not detecting the line

F. Test Track Testing

This test evaluated ANN performance after deploying the Python-trained model on an Arduino microcontroller. The trained weights and biases were used for feedforward processing of sensor inputs, with outputs observed in the Serial Monitor. The results confirm that the Arduino implementation produces outputs consistent with the Python environment, as shown in Table II.

TABLE I
TEST TRACK TESTING

NO	Action Robot	Input Sensor	Notes
1	FORWARD	0, 0, 1, 0, 0 0, 1, 1, 1, 0 0, 1, 1, 1, 0 0, 0, 1, 0, 0	The robot moves forward
2	LEFT MAX	1, 0, 0, 0, 0 1, 1, 0, 0, 0 1, 1, 0, 1, 0	The robot moved to the left
3	LEFT LOW	0, 1, 0, 0, 0 0, 1, 1, 0, 0 1, 1, 1, 0, 0	The robot moved slowly to the left
4	RIGHT MAX	0, 0, 0, 0, 1 0, 0, 0, 1, 1 0, 1, 0, 1, 1	The robot moved to the right max
5		0, 0, 0, 1, 0	The robot moves to the right low

	RIGHT LOW	0, 0, 1, 1, 0 0, 0, 1, 1, 1 1, 0, 1, 1, 1	
6	TURN AROUND	1, 1, 1, 1, 1 1, 0, 0, 0, 1 1, 0, 0, 1, 1 1, 0, 1, 0, 1	Rotating Robot

G. Testing against Variations in Lighting Conditions

This test aims to evaluate the consistency of the ANN feedforward outputs on the microcontroller under varying lighting conditions. The results show that for the same sensor inputs, the ANN produces consistent class predictions in both bright and dark conditions, demonstrating that the system is stable against changes in light intensity, t, as shown in Table III.

TABLE III
TESTING AGAINST VARIATIONS IN LIGHTING CONDITIONS

Light Conditions	Input Sensor (S1–S5)	*Feedforward Results (Prediction)*
Bright	0, 1, 1, 1, 0	*FORWARD*
	1, 1, 0, 0, 0	*LEFT_MAX*
	1, 1, 1, 0, 0	*LEFT_LOW*
	0, 0, 0, 1, 1	*RIGHT_MAX*
	1, 0, 1, 1, 1	*RIGHT_LOW*
	1, 0, 0, 1, 1	*TURN AROUND*
Dark/Dim	0, 0, 1, 0, 0	*FORWARD*
	1, 0, 0, 0, 0	*LEFT_MAX*
	0, 1, 1, 0, 0	*LEFT_LOW*
	0, 0, 0, 0, 1	*RIGHT_MAX*
	1, 0, 1, 1, 1	*RIGHT_LOW*
	1, 1, 1, 1, 1	*TURN AROUND*

H. Epoch Testing of Training Performance and Results

This test evaluates the deployment of Python trained ANN weights on a microcontroller using different epoch values. The feedforward outputs on the microcontroller remain consistent with those from Python inference, t, as shown in Table IV.

TABLE IV
EPOCH TESTING TABLE FOR TRAINING PERFORMANCE AND RESULTS

Epoch	**Training Accuracy**	**Loss Training**	**Sensor Input**	**ANN Output (Python)**	**ANN Output (Microcontroller)**
10	10,000	48,280	1, 0, 0, 1, 1	TURN AROUND	TURN AROUND
			0, 0, 1, 0, 0	FORWARD	FORWARD
			1, 0, 0, 0, 0	LEFT_MAX	LEFT_MAX
			0, 1, 0, 0, 0	LEFT_LOW	LEFT_LOW
			0, 0, 0, 1, 0	RIGHT_LOW	RIGHT_LOW
			0, 0, 0, 1, 1	RIGHT_MAX	RIGHT_MAX
30	10,000	19,868	1, 0, 0, 1, 1	TURN AROUND	TURN AROUND
			0, 0, 1, 0, 0	FORWARD	FORWARD
			1, 0, 0, 0, 0	LEFT_MAX	LEFT_MAX
			0, 1, 0, 0, 0	LEFT_LOW	LEFT_LOW
			0, 0, 0, 1, 0	RIGHT_LOW	RIGHT_LOW
			0, 0, 0, 1, 1	RIGHT_MAX	RIGHT_MAX
50	10,000	13,245	1, 0, 0, 1, 1	TURN AROUND	TURN AROUND
			0, 0, 1, 0, 0	FORWARD	FORWARD
			1, 0, 0, 0, 0	LEFT_MAX	LEFT_MAX
			0, 1, 0, 0, 0	LEFT_LOW	LEFT_LOW
			0, 0, 0, 1, 0	RIGHT_LOW	RIGHT_LOW
			0, 0, 0, 1, 1	RIGHT_MAX	RIGHT_MAX
100	10,000	0,0043	1, 0, 0, 1, 1	TURN AROUND	TURN AROUND
			0, 0, 1, 0, 0	FORWARD	FORWARD
			1, 0, 0, 0, 0	LEFT_MAX	LEFT_MAX
			0, 1, 0, 0, 0	LEFT_LOW	LEFT_LOW
			0, 0, 0, 1, 0	RIGHT_LOW	RIGHT_LOW
			0, 0, 0, 1, 1	RIGHT_MAX	RIGHT_MAX
200	10,000	0,0011	1, 0, 0, 1, 1	TURN AROUND	TURN AROUND
			0, 0, 1, 0, 0	FORWARD	FORWARD
			1, 0, 0, 0, 0	LEFT_MAX	LEFT_MAX
			0, 1, 0, 0, 0	LEFT_LOW	LEFT_LOW
			0, 0, 0, 1, 0	RIGHT_LOW	RIGHT_LOW
			0, 0, 0, 1, 1	RIGHT_MAX	RIGHT_MAX

IV. RESULT AND ANALYSIS

A. Testing Overview

The results cover ANN training in Python, performance evaluation, weight deployment on the microcontroller, and direct robot testing. The feedforward ANN was trained using IR sensor data (5 inputs) with six action classes, and the trained weights and biases were implemented on the microcontroller for real-time robot control, as shown in TableV.

TABLE V
NETWORK ARCHITECTURE

Layer	**Number of Neurons**	***Activation***
Input	5	–
Hidden Layer 1	8	*ReLU*
Hidden Layer 2	6	*ReLU*
Output Layer	*6*	*Softmax*

B. Learning Model Results in Python

The training was conducted using a line-following sensor dataset generated through simulation and several manual data collections. The training process used 80% of the data for training and 20% for validation. The training parameters were set as follows: epoch = 200, optimizer = Adam, learning rate =

0.01, loss function = Categorical Cross-entropy, and batch size = 16 t, as shown in Fig 9

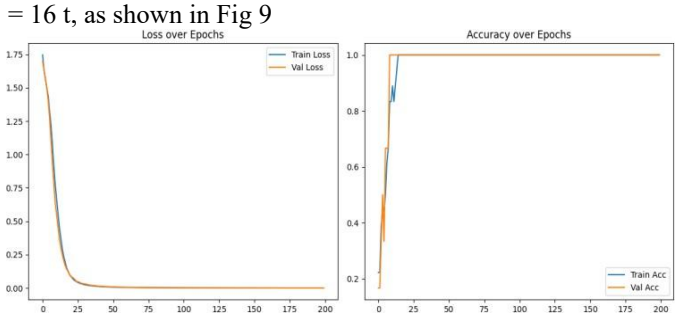

Figure 9 Train epochs 200

The graph shows that training and validation losses decrease and stabilize within 200 epochs, while both accuracies reach 1.0. This indicates effective learning and optimal model performance, with minimal overfitting, as shown in Fig 10.

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 8)	48
dense_31 (Dense)	(None, 6)	54
dense_32 (Dense)	(None, 6)	42

Figure 10 Parameter table for each layer of the ANN model

This process describes parameter calculations for three Dense layers. The first layer has 8 neurons with 48 parameters; the second has 6 neurons with 54 parameters; and the output layer has 6 neurons with 42 parameters, using softmax activation. These parameters represent weights and biases optimized during training to minimize Error and improve accuracy.

C. *Implementation of FeedForward ANN on Microcontroller*

The trained model is exported as a weight file (ANN_WEIGHTS.h) and then embedded in the Arduino Uno microcontroller. The feedforward process runs using the same structure as used during training. The following is the feedforward process on the microcontroller:

$$\text{Hidden Layer 1 } h1_j = RELU\left(\sum_{i=1}^{} (x_i \cdot W1_{ij}) + b1_j\right)$$
$$\text{Hidden Layer 2 } h2_j = RELU\left(\sum_{i=1}^{} (h1_i \cdot W2_{ij}) + b2_j\right)$$
$$\text{Output Layer } o_k = \sum_{j=1}^{} (h2_j \cdot W3_{jk}) + b3_k$$

Figure 11 Formulas for Calculating the Hidden Layer and Output Layer

The output results are still in the form of logits (raw ANN values). The softmax function $P_k = \frac{e^{ok}}{je^{ok}}$.

D. *Classification Result Confusion Matrix*

The confusion matrix in the figure shows the performance of the Artificial Neural Network (ANN) in classifying six robot action classes. Each row corresponds to the true labels, and each column to the predicted labels. All values on the main diagonal are positive, and all off-diagonal values are zero, indicating that all test data were classified correctly with no misclassification errors , as shown in Fig 12

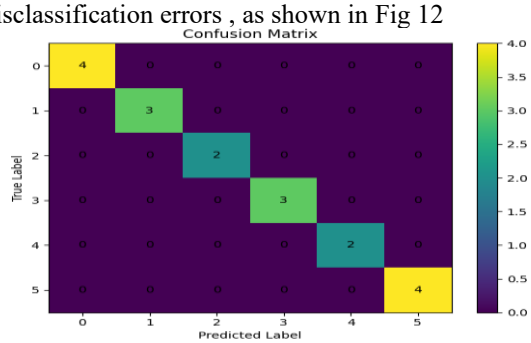

Figure 12 Confusion Matrix

E. *Discussion of Error and Accuracy of the ANN System*

The Mean Absolute Error (MAE) values obtained in Python tests and in manual tests on microcontrollers were relatively small and close to zero. Larger error values occurred only in conditions where sensor input was ambiguous or close to the boundary between classes. Yet, the ANN still accurately determined the robot's movement based on the maximum probability. Furthermore, although the ANN achieves high accuracy and low error, several limitations remain. The model depends on the quality of training data, so unseen patterns may reduce reliability. In addition, Arduino constraints such as limited memory and processing power restrict model complexity. The use of binary sensor input (0/1) also reduces sensitivity to gradual changes, which may affect performance at high speeds. Therefore, further optimization is needed to improve robustness in more dynamic conditions, as shown in Table VI.

TABLE VI
MANUAL INPUT IN PHYTON

No	Manual Input	Action	ANN Output	MAE Value
1	0, 0, 1, 0 ,0	FORWARD	0,9996	0,1999
2	0, 1, 1, 1, 0	FORWARD	0,9994	0,3998
3	1, 0, 0, 0, 0	LEFT_MAX	0,9987	0,1999
4	1, 1, 0, 1, 0	LEFT_LOW	0,9619	0,1076
5	0, 1, 0, 0, 0	LEFT_LOW	0,9972	0,000552
6	0, 1, 0, 1, 1	RIGHT_MAX	0,9983	0,200339
7	0, 0, 0, 1, 1	RIGHT_MAX	0,9976	0,1000011
8	0, 0, 1, 1, 0	RIGHT_LOW	0,9990	0,1000019
9	1, 1, 1, 1, 1	TURN_AROUND	0,9997	0,5000000
10	1, 1, 0, 0, 0	LEFT_MAX	0,9988	0,299776

TABLE VII
MANUAL INPUT IN MICRO

No	Manual Input	Action	ANN Output	MAE Value
1	0, 0, 1, 0 ,0	FORWARD	0,9998	0,0002
2	0, 1, 1, 1, 0	FORWARD	0,9999	0,0001
3	1, 0, 0, 0, 0	LEFT_MAX	0,9974	0,0026
4	1, 1, 0, 1, 0	LEFT_MAX	0,8991	0,1009
5	0, 1, 0, 0, 0	LEFT_LOW	0,9995	0,0005
6	0, 1, 0, 1, 1	RIGHT_MAX	0,9529	0,0471

7	0, 0, 0, 1, 1	RIGHT_MAX	0,9997	0,0003
8	0, 0, 1, 1, 0	RIGHT_LOW	10,000	0
9	1, 1, 1, 1, 1	TURN_AROUND	10,000	0
10	1, 1, 0, 0, 0	LEFT_MAX	0,9998	0,0002

F. Overall System Test Result

$$\%\text{success} = \frac{\text{Total success}}{\text{Total Experiment}} \text{ x } 100\%$$

- Lights On Condition

  Experiment 1 $\%Success = \frac{9}{10}\ x\ 100\% = 90\%$

  Experiment 2 $\%Success = \frac{10}{10}\ x\ 100\% = 100\%$

- Lights Completely Out

  Experiment 1 $\%Succes = \frac{8}{10}\ x\ 100\% = 80\%$

  Experiment 2 $\%Succes = \frac{90}{10}\ x\ 100\% = 90\%$

$$\%\text{Succes} = \frac{90\%+100\%+80\%+90\%}{4}\ \% = 90\%$$

Thus, the overall success rate of the Artificial Neural Network (ANN)-based line follower robot is 90%.

V. CONCLUSION

This study shows that a feedforward ANN provides more adaptive control for a line follower robot than rule-based methods. The trained ANN reliably recognizes path patterns and runs effectively on a limited Arduino microcontroller, enabling real-time execution of six action classes. Overall, ANN integration improves robot accuracy and adaptability without complex hardware.

REFERENCE

[1] S. Saadatmand, S. Azizi, M. Kavousi, and D. C. Wunsch, "Autonomous Control of a Line Follower Robot Using a Q-Learning Controller," 2020.

[2] S. K. Risandriya and J. A. Tarigan, "Optimasi Sensor Pada Robot Line Follower Dengan Jenis Lapangan Berbeda Dengan Metode Neural Network".

[3] M. Education, "Microcontroller based line follower robot 1," vol. 11, no. 3, pp. 2730–2735, 2020.

[4] O. Erbay, "Line Follower Robot with PID Control," no. January, 2024.

[5] H. M. Leal, R. S. Barbosa, and I. S. Jesus, "Control of a Mobile Line-Following Robot Using Neural Networks," pp. 1–26, 2025.

[6] C. Minaya, R. Rosero, M. Zambrano, and P. Catota, "Application Of Multilayer Neural Networks For Controlling A Line - Following Robot In Robotic Competitions Abstract :," vol. 18, no. September 2023, 2024, doi: 10.14313/JAMRIS/1.

[7] A. Latif, H. A. Widodo, R. Rahim, and K. Kunal, "Implementation of Line Follower Robot based Microcontroller ATMega32A," no. February, pp. 15–20, 2020, doi: 10.18196/jrc.1316.

[8] R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," vol. 22, pp. 717–727, 2000.

[9] T. M. Mitchell, *Machine Learning*.

[10] Trivusi, "Algoritma Feedforward Neural Network: Pengertian dan Cara Kerjanya." [Online]. Available: https://www.trivusi.web.id/2022/07/algoritma-feedforward-neural-network.html#

[11] W. Pav and J. Campa, "Intelligent Control System for Line Follower Robot : From Theory to implementation on the mechatronics field," no. November 2023, 2024, doi: 10.1109/DASC/PiCom/CBDCom/Cy59711.2023.10361359.

[12] R. A. Sarhan and M. S. Hassan, "Arduino-based implementation of kinematics for a 4 DOF robot manipulator using artificial neural network Arduino-Based Implementation Of Kinematics For A 4 Dof Robot," no. February, 2024, doi: 10.29354/diag/184235.

[13] A. Salim and W. S. Pambudi, "Implementasi Metode Hybrid Artificial Neural Network (Ann) – Pid Untuk Perbaikan Proses Berjalan Pada Prototype Robot Material Handling," vol. 1, no. 3, pp. 155–164, 2015.

[14] I. Pramana and D. Futra, "Implementasi Algoritma Q Learning Pada Robot Line Follower," pp. 1–6, 2021.

[15] A. E. Muñoz-zavala, J. E. Macías-díaz, and D. Alba-cuéllar, "A Literature Review on Some Trends in Artificial Neural Networks for Modeling and Simulation with Time Series," pp. 1–45, 2024.