LINE FOLLOWER ROBOT WITH PICK AND DROP MECHANISM USING ADVANCED PID CONTROL

BY

NAME – JOYDEEP DAS
REGISTRATION NO – 211170100310096
UNIVERSITY ROLL NO - 11700321075

NAME – PRITAM KUNDU
REGISTRATION NO - 211170100310048
UNIVERSITY ROLL NO - 11700321037

NAME – SOUVIK PANDA
REGISTRATION NO - 211170100310053
UNIVERSITY ROLL NO - 11700321040

UNDER THE SUPERVISION OF Prof. NAIWRITA DEY
Assistant Professor, Department of ECE
RCC INSTITUTE OF INFORMATION TECHNOLOGY

RCC INSTITUTE OF INFORMATION TECHNOLOGY
[Affiliated to MAKAUT, West Bengal]
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA – 700015
JUNE, 2025

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 7OOO15,

INDIA

CERTIFICATE

The report of the Project titled LINE FOLLOWER ROBOT WITH PICK AND DROP MECHANISM USING ADVANCED PID CONTROL submitted by Souvik Panda (Roll No.: 11700321040 of B. Tech. (ECE) 8th Semester of 2025) has been prepared under our supervision for the partial fulfillment of the requirements for B Tech (ECE) degree in Maulana Abul Kalam Azad University of Technology. The report is hereby forwarded.

_____

_____

[DR. NAIWRITA DEY]

Dept. of ECE

RCCIIT, Kolkata

Countersigned by

…….

Name of HOD: DR. SOHAM SARKAR

Department ECE

RCC Institute of Information Technology, Kolkata – 700 015, India

ACKNOWLEDGEMET

I express my sincere gratitude to Dr. NAIWRITA DEY of Department of **ECE**. RCCIIT and for extending their valuable times for me to take up this problem as a Project.

I am also indebted to Dr. NAIWRITA DEY for their unconditional help and inspiration.

Last but not the least I would like to express my gratitude to our department who helped me in their own way whenever needed.

Date: …….. 2025

PRITAM KUNDU

JOYDEEP DAS

SOUVIK PANDA

B. Tech (ECE) –8th Semester, 2025, RCCIIT

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 7OOO15,

INDIA

CERTIFICATE of ACCEPTANCE

The report of the Project titled LINE FOLLOWER ROBOT WITH PICK AND DROP MECHANISM USING ADVANCED PID CONTROL submitted by Pritam Kundu, Souvik Panda, Joydeep Das (Roll No.: 11700321037,11700321040,11700321075 of B. Tech. (ECE) 8th Semester of 2024) is hereby recommended to be accepted for the partial fulfillment of the requirements for B.Tech (ECE) degree in Maulana Abul Kalam Azad University of Technology.

Name of the Examiner	**Signature with Date**
1. …….	…….
2. ……..	……..
3. ……	
……	
4. …….	
……	

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION & MOTIVATION

1.1 Introduction:

The Line Follower Robot with Pick and Drop Mechanism Using Advanced PID Control represents a practical application of automation in robotics, designed to follow a predefined path accurately and perform object manipulation tasks efficiently. This system is suitable for industrial automation, warehouse management, and service-based operations. It combines reliable sensor feedback with intelligent control techniques to perform line following and pick-and-drop actions with high precision.

The robot uses infrared (IR) sensors to detect a black path on a white surface (or vice versa), providing continuous feedback about its position. Based on this input, the robot applies a PID controller, a control algorithm that calculates the correction needed to stay on the path using three terms: proportional (P), which responds to current error; integral (I), which accounts for accumulated past error; and derivative (D), which anticipates future error. This combination ensures smooth and stable movement, reducing overshooting or oscillation.

To further improve control accuracy, especially under changing path conditions, a neural network (NN) is used to dynamically tune the PID parameters. Neural networks are machine learning models inspired by the human brain, capable of learning complex patterns. In this project, the NN observes the system's behavior and adjusts the PID values (K p, Ki, K d) in real time, enabling better responsiveness during sharp turns or surface variations.

Apart from navigation, the PID control system also plays a vital role in operating the robotic hand. The pick-and-drop mechanism, powered by servo motors, uses PID control to ensure smooth, precise movement of each joint. This helps in accurate gripping, lifting, and placement of objects without causing unnecessary jerk or error due to overshooting.

The system also includes communication capabilities for remote monitoring and task control. Operators can view real-time performance, tune parameters, or assign tasks from a distance, making the system adaptable for dynamic use cases. However, challenges such as managing the computational load of the NN on a microcontroller, real-time synchronization of the robot's movement and hand, and maintaining accuracy during high-speed operations need to be addressed. In conclusion, the Line Follower Robot with Pick and Drop Mechanism Using Advanced PID Control demonstrates how traditional control systems enhanced with modern AI techniques like neural networks can create more intelligent and capable robotic solutions. By automating repetitive yet precise tasks, the robot contributes to efficiency, accuracy, and flexibility in modern automation environments.

1.2 Literature Review: A Line Follower Robot is an autonomous robot which follows the line that is drawn on the surface consisting of s contrasting color. It is designed to move automatically and follow the plot line. PID control of line follower is a method consisting of Proportional, Integral & Derivative functions to improve the movement of the robot. The robot uses several sensors to identify the line thus assisting the bot to stay on the track. The robot is driven by DC Motors to control the movement of the wheels. However the output was taken under a certain lighting condition and a similar response might not be obtained for a different ambient lighting condition. The sensors used would need to be calibrated to produce the perfect output [1]. Mahmud Kazi, Abdullah h Nahid et a al wrote a paper on "Design and Implementation of an Autonomous Line Follower Robot". This paper explores the design and implementation of a line follower robot It discusses various sensors, control algorithms, and hardware components used in the development process Additionally, it evaluates the robot's performance in terms of speed, accuracy, and efficiency [2]. Chandra Sekhar Pati and Rahul Kala made an robot following based on computer vision and PID control for a differential wheel robot. In this project the researchers have used two types of bots, one worked as a master and was controlled by the user through manual control based on Android and Bluetooth. The second robot worked as a follower robot [3]. In this paper, they proposed an advanced control for a serving robot. A serving robot acts as an autonomous line-follower vehicle that can detect and follow the line drawn on the floor and move in specified directions. The robot should be able to follow the trajectory with speed control. Two controllers were used simultaneously to achieve this. Con-volutional neural networks (CNNs) are used for target tracking and trajectory prediction, and a proportional-integral-derivative controller is designed for automatic steering and speed control. This study makes use of a Raspberry PI, which is responsible for controlling the robot car and performing inferenceusing CNN, based on its current image input [4]. The method for line follower robot based on the adjustment of sensors play a great role in the instantaneous computation of of radius of curvature response of robot along t a line. The the line with desired speed and accuracy uracy set by the Ultrasonic sensors are used to avoid collision of the robot. If the sensor senses any obstacle near to it, it will rotate right or left with a preset angle and direction [5]. Design and fabrication of pick and place robotic arm was proposed by the authors. Servos are needed to empower the joints of robotic arm. Basically to perform pick and place operation robotic arms are designed and fabricated [6]. Simple underactuated robot hand constitute easily implementable practical robotic grasping solutions. Similarly, their work has aimed to provide a system with low computation and complexity overheads for haptic sensing applications in practical robotics.. In many cases this may need to be via a motor control, rather than a mechanical approach [7]. Agriphina Mugure Majau gives an overview on line follower robots featuring pick and place systems. It surveys emerging technologies such as machine learning. computer vision, and advanced control algorithms applied to enhance robot performance. The review identifies key trends, challenges, and opportunities in the development of intelligent line follower robots with pick and drop capabilitics [8]. Maruf Ar Rusafi et al conveyed that line follower robot is a robot that can move along a path. It is an electronic device that can track out and follow the line schemed on

the floor. The line follower robot is a prototype model technic and created material handling purpose, various movement systems are used to turn the robot In picking the object a robotic arm is used. An intelligent robot system is designed which can give corrective feedback in different colors of light A robotic arm is a robot manipulator, usually using a sequence of function by controlled program, with resembling functions to a human arm[9].This paper presents the design of Robotic arm which simulates the human hand movement to grip an object. The robotic arm is Arduino controlled robotic arm hence it can be implemented to a robot which can analyze hazardous area do a material handling. In order to examine those torque characteristics, we consider a model of humanoid robot arm and simulate typical object lifting and transferring tasks by using the robot arm. This paper concludes with some possible applications of 4 D.O.F robotic arm mechanism based on type of end effector attached to the robotic arm[10] This paper presents the development of an IoT-enabled line follower robot designed for material handling applications. The system integrates an IoT module for real-time tracking and monitoring, along with a pick-and-drop mechanism to move objects from one location to another autonomously. The authors highlight the successful implementation of sensors and actuators, coupled with efficient object detection and handling in industrial settings. The system's scalability and adaptability to different tasks are noted, though improvements could be made in obstacle avoidance.[11] The paper explores the use of convolutional neural networks (CNN) to enhance the real-time object detection and navigation capabilities of line follower robots. The research leverages computer vision techniques for better path tracking and obstacle detection. Results show that CNN improves the robot's ability to follow lines and detect objects more accurately, even in complex environments, when compared to traditional methods. However, real-time processing posed challenges due to computational constraints.[12] This paper examines the implementation of adaptive PID control for precise line following applications. The researchers introduce an adaptive mechanism to adjust the PID parameters in real time, improving the robot’s path-tracking accuracy and responsiveness to sudden changes in curvature. The paper also addresses the challenges of varying environmental factors like surface texture and lighting, proposing a solution that adjusts to these conditions automatically.[13]This paper, the authors propose using Kalman filter-based sensor fusion to enhance the performance of line follower robots in unstructured environments. By integrating multiple sensors—such as infrared and ultrasonic—the robot's perception and tracking accuracy are improved. The Kalman filter helps smooth out noisy sensor data, resulting in better decision-making capabilities when navigating irregular surfaces or transitioning between different lighting conditions.[14] The paper presents the design of a line follower robot that integrates LIDAR and ultrasonic sensors for obstacle detection and avoidance. By combining these sensors, the robot can navigate a predefined path while avoiding obstacles, even in dynamic environments. The authors discuss the advantages of LIDAR over traditional infrared sensors, particularly in terms of range and accuracy. The system proved effective in environments where obstacles were frequent and unpredictable.[15] This paper focuses on the design and implementation of a line follower robot tailored for automated warehousing systems. The robot is designed to follow lines marked on the warehouse floor, automating

tasks like inventory movement. The research emphasizes the integration of sensor technologies and DC motors for precise movement, alongside an evaluation of the robot's performance in a real-time environment. The use of PID control ensured accurate path following with minimal deviation in dynamic lighting conditions. The system proved reliable, albeit requiring lighting-specific calibration for optimum performance.[16] This research focuses on integrating a pick-and-place robotic arm into a line follower robot using fuzzy logic for control. The fuzzy logic approach allows for smooth, precise control of the robotic arm, enabling it to handle objects with varying weights and dimensions. The authors demonstrate the effectiveness of this method in material handling applications, noting that the system can adapt to different types of objects without requiring manual recalibration.[17] This paper combines IoT and computer vision technologies in the development of a line follower robot equipped with a robotic arm for industrial automation. The system uses IoT for remote monitoring and vision-based algorithms to enhance path-following accuracy and object handling. The research highlights the robot's ability to operate autonomously in a factory setting, with minimal human intervention. The system showed promise in improving efficiency in material transport and object placement tasks.[18] This paper introduces a smart line follower robot capable of detecting and tracking moving objects in real time. The authors utilize a combination of computer vision and sensor technologies to enhance the robot's environmental awareness, allowing it to navigate more complex, dynamic environments. The system performed well in tests, but challenges remain in improving the robot's response time to fast-moving objects.[19]This research integrates deep reinforcement learning (DRL) algorithms into a line follower robot designed for warehouse management. The DRL model is trained to optimize the robot's pathfinding and obstacle avoidance strategies in a dynamic warehouse environment. The paper discusses the benefits of using DRL over traditional control methods, such as improved adaptability to changing environments and tasks. The system showed notable improvements in efficiency and reduced human intervention during testing.[20] The authors explore the application of line follower robots in the agricultural sector, specifically for tasks like soil monitoring. The robot is equipped with sensors to measure soil moisture and temperature as it follows a predefined path through fields. This system reduces the need for manual labor and increases the efficiency of routine agricultural tasks. The paper also highlights the robot's ability to operate in different terrains and weather conditions.[21]This paper focuses on improving trajectory prediction for line follower robots using vision-based control. The authors develop an IoT-enabled system that provides real-time feedback to adjust the robot's path based on visual inputs. The proposed system performs well in scenarios where traditional sensor-based systems struggle, such as irregularly shaped lines or varying light conditions. The robot's ability to predict and adapt to complex paths makes it ideal for industrial use.[22] This paper examines the application of neural networks for controlling a line follower robot with pick-and-place capabilities. The neural network is trained to optimize the robot's movements, ensuring smooth transitions between path-following and object manipulation tasks. The authors report that the use of neural networks resulted in improved accuracy and reduced task completion times compared to traditional PID control systems [23]. This paper proposes a low-cost, IoT-integrated

line follower robot for indoor navigation. The system is designed to be affordable while still offering reliable performance in navigating complex indoor environments. The use of IoT allows for remote monitoring and control, making the system suitable for home automation and small-scale industrial applications. The authors emphasize the system's low power consumption and ease of setup[24]. The paper explores the use of advanced machine learning algorithms to improve real-time obstacle detection in line follower robots. By training the robot to recognize and avoid obstacles dynamically, the authors achieved significant improvements in path-following performance. The system is designed to operate in environments with frequent and unpredictable obstacles, making it ideal for use in warehouses or manufacturing facilities [25].

1.3 Motivation behind the work:

In today's rapidly advancing industrial landscape, the demand for intelligent automation solutions is steadily increasing. Manual operations, particularly those involving repetitive tasks such as transporting objects or sorting items, are not only time-consuming and inefficient but also prone to human error. To address these limitations, the idea of developing a cost-effective, autonomous robot with pick-and-place capabilities was conceived. A line follower robot with a manipulator can simulate industrial workflows, such as moving items along an assembly line or handling objects within a structured environment like a warehouse or factory.

Moreover, conventional line follower systems often fail to cope with dynamic or non-linear paths due to static PID control configurations. This limitation inspired the integration of a Neural Network-based PID (NN-PID) controller, enabling the system to adaptively tune itself based on sensor feedback and real-time conditions. Such adaptability enhances the robot's performance, making it more robust for various indoor tasks. The project also acts as a learning platform, bringing together embedded systems, control theory, machine learning, and robotics.

Key Motivations:

- To develop an intelligent, low-cost robot capable of autonomously navigating and performing pick-and-place tasks in structured environments.
- To enhance traditional PID control systems by integrating a neural network for adaptive tuning and smarter control.
- To gain practical experience in system integration involving sensors, actuators, and control algorithms aligned with Industry 4.0 trends.

CHAPTER –2: METHODOLOGY

2.1 Problem Formulation

The rising demand for automation in indoor environments—such as factories, warehouses, hospitals, old age homes, and educational institutions—has exposed critical limitations in current robotic systems. Many of these settings still depend on manual labor or semi-automated tools for tasks like transporting items, delivering supplies, and handling small objects. This not only results in inefficiency and higher labor costs but also increases the risk of human error and fatigue. Traditional automation methods, including conveyor belts or fixed robotic arms, are often rigid, expensive, and lack the mobility and adaptability needed for dynamic indoor scenarios.

Basic line-following robots, which are popular due to their simplicity, also exhibit performance issues. Their reliance on static PID controllers makes them sensitive to path irregularities, sharp turns, and environmental disturbances. This lack of adaptability becomes even more problematic in settings that require high precision and reliability—such as timely delivery of materials in a supply chain or safe transportation of medical supplies in healthcare facilities.

The core issue lies in the absence of an integrated, intelligent robotic system that combines real-time adaptive navigation with object manipulation capabilities. Existing solutions lack the flexibility to adjust to changing environments and perform physical interactions effectively. This project aims to solve that problem by developing a compact, low-cost, and scalable robotic platform that can autonomously follow paths, adapt to varying conditions using neural network-based PID control, and interact with its surroundings through a servo-driven pick-and-place mechanism.

Unlike traditional PID-controlled robots, the proposed system uses a neural network to dynamically tune the PID gains (K p, Ki, and K d) in response to sensor input and environmental changes. This allows for smoother turns, greater stability, and reduced tuning overhead. Simultaneously, the inclusion of a servo-actuated manipulator arm enables the robot to perform meaningful tasks such as picking and dropping items with precision—making it highly suitable for real-world indoor automation applications.

The problem is particularly relevant to stakeholders such as factory operators, warehouse logistics teams, hospital staff, and eldercare providers, all of whom can benefit from reduced workload, improved operational efficiency, and safer task execution. With the shift toward Industry 4.0 and the growing emphasis on contactless and autonomous systems, especially in post-pandemic contexts, the need for such intelligent robots is more urgent than ever.

This formulation gives rise to key research questions:

- How effectively can a neural network optimize PID control in real-time for autonomous path-following?
- How accurately can a PID-controlled robotic arm perform pick-and-place operations using servo motors?
- What level of system integration and performance can be achieved using cost-effective components like infrared sensors, microcontrollers, and lightweight actuators?

The proposed solution integrates multiple subsystems—including adaptive navigation, intelligent control, object

manipulation, and optional communication modules—into a unified, modular platform. It balances robustness with flexibility by merging classical control theory with neural networks. The expected outcomes include improved trajectory tracking, smoother manipulation tasks, reduced need for human intervention, and enhanced performance in diverse indoor scenarios.

By addressing the limitations of current low-cost robots and enhancing their functionality through AI-driven control, this project contributes to the development of intelligent, autonomous agents suitable for modern indoor automation. It also sets the stage for future advancements in multi-agent coordination, cloud-connected robotics, and intelligent scheduling systems.

2.2 Proposed system block diagram:

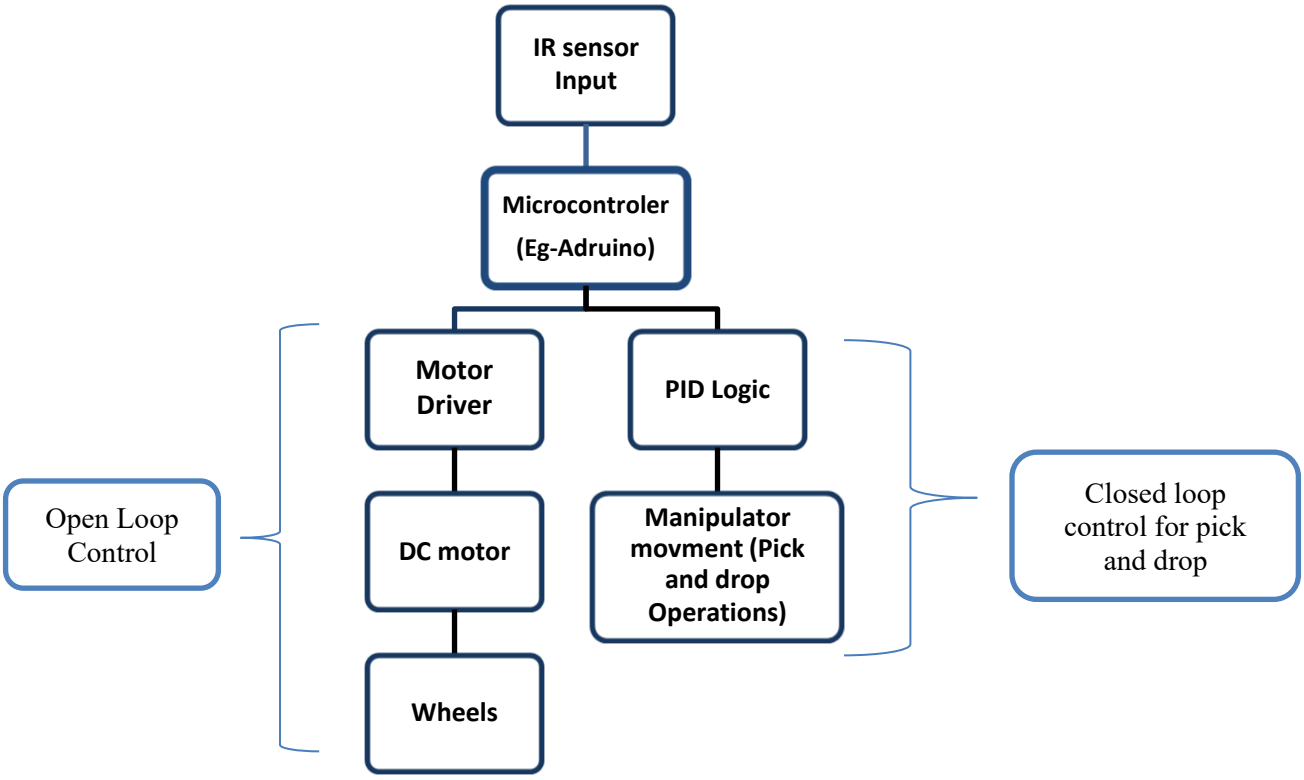

Fig 1: Proposed Block Diagram of the system

The given block diagram represents the control structure of a robotic system that combines both open-loop and closed-loop control mechanisms for different functionalities. It primarily uses an IR sensor, a microcontroller (e.g., Arduino), DC motors for movement, and PID logic for precise manipulator control during pick-and-drop operations.

At the top of the system is the IR sensor input, which acts as the sensory component for detecting the environment—such as line tracking or obstacle presence. This input is fed into the microcontroller, which serves as the brain of the

system. The microcontroller processes the sensor signals and generates appropriate control signals for the rest of the hardware.

From the microcontroller, the signal branches into two paths. The first path leads to the motor driver, which acts as an interface between the low-power control signals from the microcontroller and the high-power DC motors. These motors are responsible for the movement of the robot's wheels, enabling navigation and mobility. This portion of the system is governed by open-loop control, meaning the motors run based on pre-defined commands without feedback correction. While simple and effective in predictable environments, this method doesn't account for external disturbances or load variations.

The second branch from the microcontroller is dedicated to more precise control using PID logic. This logic takes feedback, processes the current state of the manipulator arm, and adjusts the output to ensure accurate movement. This is especially crucial during pick-and-drop operations, where precise positioning is required to grasp and place objects effectively. Because the manipulator uses feedback (possibly from position or force sensors), this segment of the system operates under closed-loop control. The feedback allows the system to dynamically correct errors and maintain accuracy in real-time.

In summary, this control architecture divides the robotic system into two functional domains: an open-loop system for basic navigation and movement using DC motors and wheels, and a closed-loop system for fine-tuned manipulator movements using PID control. This hybrid approach balances simplicity and accuracy, making the system efficient for tasks like object handling and autonomous navigation.

CHAPTER-3: PROPOSED CONTROLLER DESIGN

3.1Control loop:

There are mainly two control loops for the robot's movement one is outer control and other one is Inner control. That is shown in the below block diagram in fig3.There is an outer loop for the robot's movement in the line, and line following logic is implemented on that outer loop. At the same time the inner loop works for the pick and drop operations. Outer loop consist-of the following components like Microcontroller unit, motor control, IR sensor etc. Inner control works with the NNPID controller and manipulator.

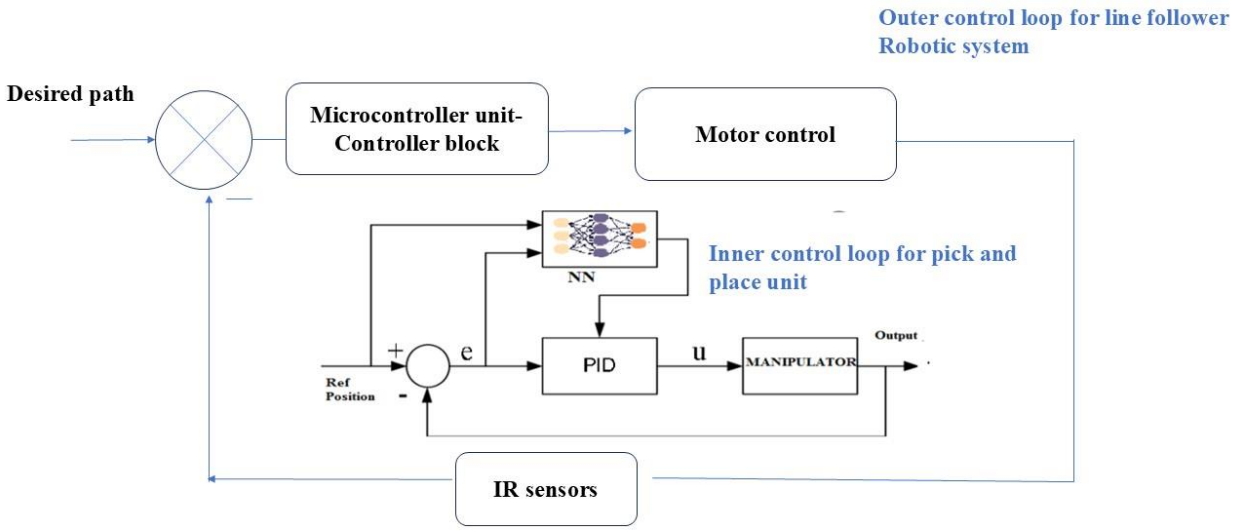

Fig2: Diagram of the control loop of the whole system

3.2Conventional PID controller:

A controller is an essential part of any control system, designed to regulate the behavior of a process or system so that it follows a desired setpoint or output. It works by continuously comparing the actual output of the system (called the process variable) with the desired output (the reference or setpoint) and minimizing the difference between them, which is known as the error. Based on this error, the controller generates a corrective signal that adjusts the system's input to bring the output closer to the desired value. This feedback mechanism ensures the system behaves in a stable, efficient, and accurate manner even in the presence of disturbances or changes in operating conditions. Designing a controller involves several important steps. First, the system or plant must be properly modeled. This means understanding how it behaves in response

to different inputs, usually through differential equations or transfer functions. Once the model is ready, the next step is to define the control objectives — such as minimizing error, ensuring fast response, avoiding overshoot, and maintaining stability. After setting these goals, the designer chooses an appropriate type of controller based on the system's complexity and performance requirements.

For simple systems, a PID (Proportional-Integral-Derivative) controller is commonly used due to its ease of implementation and effectiveness in many practical cases. The PID controller uses three parameters: K_p (proportional), K_i (integral), and K_d (derivative), which together respond to the present, past, and future trends of the error.

In more complex or nonlinear systems, advanced methods such as neural network-based controllers or fuzzy logic controllers may be used. These controllers learn from data and adapt their behavior over time, making them suitable for systems that are difficult to model or experience changing dynamics. Once the type of controller is selected, the next phase is to design and tune it. For PID controllers, tuning involves adjusting the K p, Ki, and K d values to achieve the desired performance. This can be done manually, through methods like the Ziegler–Nichols tuning rule, or using software tools that simulate the system's behavior.

After designing the controller, it must be validated through simulation. Using platforms like MATLAB, Simulink, or Python, engineers can test how the controller performs in various scenarios. If the simulation results are satisfactory, the controller is implemented in real hardware — for example, in a microcontroller like Arduino or a programmable logic controller (PLC). Real-world implementation may require further tuning due to noise, sensor inaccuracy, or actuator limitations. Overall, controller design is a critical engineering process that ensures systems function reliably, accurately, and efficiently in both industrial and everyday applications.

So, designing a basic PID controller is very important, and this area consistently attracts new researchers aiming for improved performance and efficiency. In general, the design of PID controllers can be classified into two main categories: classical tuning and intelligent tuning. Classical tuning techniques include methods such as the Ziegler-Nichols method, Ziegler-Nichols frequency response method, relay tuning method, and the Cohen-Coon method. These techniques have been widely used due to their simplicity and effectiveness in certain scenarios. However, one major drawback of classical tuning is that it is mainly suitable for first-order and second-order systems. Moreover, these methods often involve complex mathematical calculations, which can limit their application in more advanced or nonlinear systems[26].

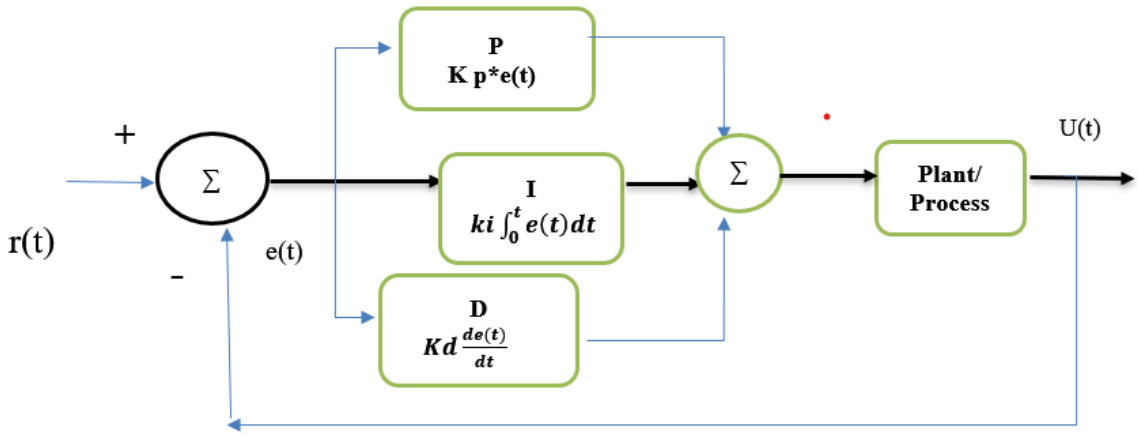

Fig3: PID Controller's block diagram

- **e(t)**: Error between setpoint and process variable.
- **u(t)**: Control output at time t.
- **Process/Plant**: System being controlled.

Mathematical Representation:

So according to proportional action,

$U(t) = K_p \times e(t)$ ----------(I)

According to Integral action,

$U(t) = K_i \int_0^t e(t)dt$ ----------(II)

According to derivative action

$U(t) = Kd\frac{de(t)}{dt}$----------(III)

Combining all these three equation we get the output of the PID controller is given by the equation:

$$U(t) = K_p \times e(t) + K_i \int_0^t e(t)dt + Kd\frac{de(t)}{dt} \quad \text{-(IV)}$$

Where:

- u(t) is the controller output (control signal)
- e(t)=r(t)−y(t) is the error between desired output r(t)rand actual output y(t)
- K_p is the proportional gain
- K_i is the integral gain
- K_d is the derivative gain

Laplace Domain Transfer Function (for implementation in simulations or analysis):

$$U(s)= K_p\left(1 + \frac{1}{Ti*s} + Tds\right) E(s) \quad \text{-(V)}$$

Where:

- $Ti=\frac{Kp}{Ki}$ (Integral time constant)
- $Td=\frac{Kd}{Kp}$ (Derivative time constant)

How PID Controller Works:

- **Comparison:** A PID controller starts its operation by comparing the setpoint value and the process variable value, which it get from the sensor devices. Then it generates a error signal which is the difference between setpoint and the process variable value or the actual value.
- **Proportional control:** Proportional(P) term responses to the error signal and it aims to bring the system closer to the setpoint.
- **Integral control:** The integral(I) part is responsible for addressing any longterm error from evaluating the past errors. And it eliminates steady-state error of a system.
- **Derivative control:** The derivative(D) term observes the rate of change of error, and try to calculate the next error and according to that it makes the system stable.
- **Control signal:** After processing error signal through P, I and D term the controller creates a

signal and send it to the actuator to do it's job according to that.

And this loop continues until to get the most desired or the closest value of the process variable of setpoint.[28]

Tuning of PID controller:

Tuning of PID controller means adjusting the three variables Proportional(P), Integral(I) and Derivative(D). Through this process we try to achieve the desired controlled position. PID controller is of three parameters: proportional gain K, integral time T_i, and derivative time T_d. The procedure of finding the controller parameters is called tuning. Many PID tuning methods had been derived to determine the value of the three parameters to obtain a controller with good performance and robustness. Some methods with simple formulas use little information of process dynamics to obtain moderate performance, however it often need to retune by trial and error depend on those results. More sophisticated tuning method can get rise to considerable improvements in performance, but they are also more demanding computationally and depend on more information of process dynamics. The choice of method should be based on the characteristics of the process and performance requirements. A.

1. Ziegler-Nichols (Z-N) and related methods: The methods developed by Ziegler and Nichols have had a profound and lasting influence on the practical application of PID controller tuning. These methods are widely adopted by both controller manufacturers and end users due to their simplicity and effectiveness in achieving stable control in various industrial systems. The primary design goal of the Ziegler-Nichols (Z-N) tuning techniques is to achieve quarter amplitude damping, which ensures a fast and decently damped system response.

There are two main approaches in the Ziegler-Nichols method: the step response method and the frequency response method.

The step response method, also known as the open-loop method, involves applying a unit step input to the system and observing the resulting output. From the process response curve, two key parameters are identified: the delay time (L) and the time constant (T). These are typically extracted from the S-shaped curve that emerges in systems with a dominant lag. A simplified first-order plus dead time (FOPDT) model is used to approximate the system's behavior. Once the parameters L and T are determined, they are substituted into predefined empirical formulas to calculate the PID controller gains: the proportional gain (K p), the integral time (Ti), and the derivative time (Td). This method is particularly useful for

systems where it is safe to operate in open-loop mode and allows for quick tuning based on process reaction.

The frequency response method, also known as the closed-loop or ultimate gain method, begins by setting the controller to proportional-only mode. The gain is then gradually increased until the closed-loop system starts oscillating with a constant amplitude. The gain at which the system begins to oscillate is known as the ultimate gain (Ku), and the period of the sustained oscillation is called the ultimate period (Tu). These two values are then used in standard Ziegler-Nichols formulas to determine the PID gains. This method is more suitable for systems that can tolerate temporary oscillations and allows tuning based on the system's frequency characteristics. Both methods provide a systematic and empirical way to determine PID parameters without requiring an exact mathematical model of the system. This makes them particularly attractive in industrial settings where modeling may be difficult or time-consuming. The step response method is often used in situations where the system's open-loop behavior is accessible and safe to observe, while the frequency response method is commonly used when a closed-loop test is more appropriate.

TABLE 1: PID controller parameters obtained from the Z-N step response method

Control function	K	Ti	Td
P	0.5Ku		
PI	0.4Ku	0.8Tu	
PID	0.6Ku	0.5Tu	0.125Tu

TABLE 2: PID controller parameters obtained from the Z-N frequency response method

Control function	K	Ti	Td
P	1/a		
PI	0.9/a	3L	
PID	1.2/a	2L	L/2

Despite their advantages, Ziegler-Nichols tuning methods have some limitations. They can sometimes result in aggressive tuning, causing overshoot or instability in certain types of processes, especially those with high delays or nonlinear dynamics. As a result, many practitioners apply modifications or refinements to these methods based on specific system characteristics.

Overall, the Ziegler-Nichols tuning methods offer a solid foundation for PID controller tuning, making them a cornerstone in the field of process control and a starting point for many modern and adaptive tuning techniques.

PID controller parameters obtained from the Z-N step response method are shown in Tab.1

There are two essential drawbacks with Ziegler-Nichol's methods:

a) Too little information is used to characterize process dynamics.

b) The design criterion, quarter amplitude damping, chosen by Ziegler and Nichols, gives closed-loop systems with poor damping and poor robustness.

These drawbacks have been known for a long time. The controllers often need to retune based on the result obtained by these methods. Because of their simplicity the Ziegler-Nichols methods have, however, remained very popular.[27]

2.Cohen coon open loop tuning method: The Cohen-Coon method is another classical approach used for PID controller tuning, primarily based on the process reaction curve obtained from a step response. It is particularly suitable for first-order processes with time delay, and provides a more refined and balanced tuning compared to some of the earlier methods. The main advantage of the Cohen-Coon method lies in its ability to offer a compromise between fast system response and overall stability, making it a widely accepted method for various industrial applications.

In the Cohen-Coon approach, the process is subjected to a step input, and the resulting output is used to extract three essential parameters: the steady-state gain (K), the time delay or dead time (L), and the time constant (T). Using these parameters, the method uses predefined empirical formulas to determine the PID controller parameters: proportional gain, integral time, and derivative time. Unlike Ziegler-Nichols, which targets quarter-amplitude damping, Cohen-Coon aims to reduce both the steady-state error and the response time while maintaining acceptable overshoot. This makes the method especially useful for systems where accuracy and smooth control are important.

The Cohen-Coon tuning method is more flexible than the Ziegler-Nichols approach, especially for systems with significant delay, and it provides different tuning constants for P, PI, and PID controllers.

However, like other classical tuning techniques, it assumes a linear and time-invariant system and may not perform optimally in nonlinear or time-varying environments. Moreover, accurate parameter estimation from the process reaction curve is essential, which can sometimes be challenging due to noise or disturbances in practical scenarios.

On the other hand, the trial-and-error method is the most basic and intuitive approach to PID tuning. In this method, the controller parameters are manually adjusted while observing the system's response. Typically, the tuning starts by setting the integral and derivative gains to zero and slowly increasing the proportional gain until the system begins to respond effectively. Then, the integral gain is added to eliminate steady-state error, followed by fine-tuning the derivative gain to minimize overshoot and enhance stability.

This method is simple and requires no mathematical modeling, making it useful for real-time tuning in small or less critical systems. However, it is time-consuming and relies heavily on the experience of the operator. It may also result in suboptimal performance or even instability if not handled carefully. Due to its subjective nature, the trial-and-error method is usually not recommended for complex or sensitive control systems, but it still serves as a practical approach for initial parameter setting or fine-tuning during implementation.

Table 3. Standard recommended equations to optimize Cohen Coon predictions

	KC	Ti	Td
P	(P/NL) * (1+(R/3))		
PI	(P/NL) *(0.9+(R/12))	L*(30+3R)/(9+30R)	
PID	(P/NL) *(1.33+(R/4))	L*(30+3R)/(9+30R)	4L/(11+2R)

where the variables P, N, and L are defined below.

P: Percent change of input

N: Percent change of output

L: τ_{dead}

R: τ_{dead}/τ

The process in Cohen-Coon turning method is the following:

1. Wait until the process reaches steady state.
2. Introduce a step change in the input.
3. Based on the output, obtain an approximate first order process with a time constant τ delayed by τ_{dead} units from when the input step was introduced.

The values of τ and τ_{dead} can be obtained by first recording the following time instances:

t0 = time at input step start point t2 = time when reaches half point t3 = time when reaches 63.2% point

4. Using the measurements at t0, t2, t3, A and B, evaluate the process parameters τ, τ_{dead}, and K_o.
5. Find the controller parameters based on τ, τ_{dead}, and K_o.[29]

These are other common methods that are used, but they can be complicated and aren't considered classical methods, so they are only briefly discussed.

Internal Model Control:

The **Internal Model Control** (IMC) method was developed with robustness in mind. The Ziegler-Nichols open loop and Cohen-Coon methods give large controller gain and short integral time, which isn't conducive to chemical engineering applications. The IMC method relates to closed-loop control and doesn't have overshooting or oscillatory behavior. The IMC methods however are very complicated for systems with first order dead time.

Auto Tune Variation:

The **auto-tune variation** (ATV) technique is also a closed loop method and it is used to determine two important system constants (P_u and K_u for example). These values can be determined without disturbing the system and tuning values for PID are obtained from these. The ATV method will only work on systems that have significant dead time or the ultimate period, P_u, will be equal to the sampling period.

3.3 Proposed NN Based PID Controller:

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the human brain. They enable computers to think, recognize complex and confusing patterns, and imitate

human decision-making processes. ANNs have become a powerful technique for pattern recognition, which is essential whenever computers interact with real-world data. These networks are broadly applicable across various fields, including image and speech recognition, natural language processing, medical diagnosis, financial forecasting, and more. A neural network resembles the human brain in two key ways: first, it acquires knowledge through a learning process where it adjusts internal parameters based on data; second, it stores this knowledge in the form of synaptic weights, which are the connection strengths between neurons.

One of the most common and foundational types of neural networks is the Multi-Layer Perceptron (MLP). An MLP consists of an input layer, one or more hidden layers, and an output layer. Each layer contains interconnected neurons (also called nodes), where every neuron in one layer is connected to all neurons in the next layer, making it a fully connected network. The neurons in hidden layers apply non-linear activation functions such as sigmoid, tanh, or ReLU, which enable the network to learn and model complex, non-linear relationships within data. MLPs are typically trained using supervised learning algorithms like backpropagation combined with gradient descent, which iteratively adjusts the synaptic weights to minimize the difference between the predicted and actual output. The training process involves feeding data into the network, calculating errors, and propagating these errors backward through the network to update the weights accordingly.

The strengths of ANNs, particularly MLPs, include their ability to learn from data, adapt to new patterns, and generalize from training examples to unseen data. However, they also have weaknesses, such as the need for large amounts of labeled data, computational intensity during training, and the risk of overfitting if not properly regularized. Despite these challenges, the versatility and robustness of ANNs have led to their widespread adoption in solving complex pattern recognition and classification tasks in many technological and scientific domains.

NN-PID: Neural Network-Based PID Controller:

The Neural Network-based PID (NN-PID) controller represents a significant advancement in the field of intelligent control systems, bridging the gap between classical feedback control theory and machine learning. While traditional PID (Proportional-Integral-Derivative) controllers are widely used due to their simplicity, robustness, and effectiveness in linear systems, they suffer from performance degradation in systems with nonlinear dynamics, time-varying characteristics, or environmental disturbances. NN-PID seeks to enhance the conventional PID approach by introducing a neural network that dynamically adjusts the PID parameters (K p, Ki, K d) in real time, based on feedback from the system. This fusion of neural networks with control engineering results in an adaptive controller capable of learning the optimal control

behavior from experience, without requiring an exact mathematical model of the plant.

In the NN-PID architecture, the neural network acts as a self-tuning mechanism that learns from the system's error response. The inputs to the network typically include error (e), change in error (Δe), and sometimes the integral of error (Σ e), mimicking the traditional PID structure. The outputs are the controller gains, which are updated continuously during the operation of the manipulator. This dynamic adjustment enables the controller to maintain optimal performance under varying load conditions, frictional forces, mechanical wear, and external disturbances—situations where a fixed-gain PID would either underperform or require manual retuning.

The learning capability of neural networks allows NN-PID controllers to generalize well in untrained conditions. During the initial phase, the network is trained using data obtained from simulations or controlled experiments, typically with backpropagation algorithms. Once deployed, the controller can continue to adapt online in real time using reinforcement or supervised learning techniques, depending on the feedback and design strategy. For this reason, NN-PID controllers are often used in applications where the model of the system is either too complex or difficult to derive analytically.

In the context of a manipulator arm, NN-PID plays a crucial role in providing adaptive control for each servo motor. Each joint of the arm—up-down, forward-backward, and base rotation—can be treated as a nonlinear subsystem with its own dynamics. The NN-PID controller learns to tune itself for each of these joints individually based on their unique behavior, thus enabling smooth and precise movement. This is especially useful when handling varying payloads or interacting with unpredictable environments, such as in human-assistive tasks or material sorting.

The key advantages of NN-PID include faster settling time, reduced overshoot, improved disturbance rejection, and better trajectory tracking compared to fixed-gain PID controllers. However, it also introduces challenges such as the need for sufficient training data, computational overhead, and stability analysis of the learning process. Nevertheless, NN-PID stands out as a powerful method for integrating adaptability into traditional control systems, making it a valuable tool in modern robotics and automation where intelligence and reliability are both crucial.

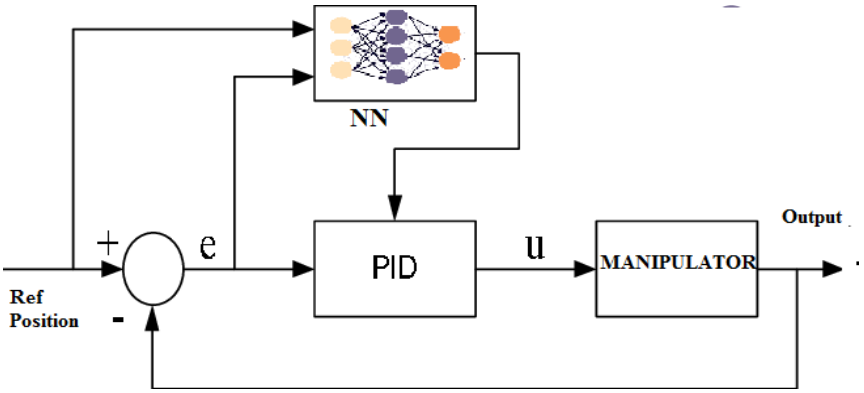

Fig 4: Block Diagram for NN based PID controller

Working logic of NNPID in manipulator hand movement:

The **Setpoint** is the desired target value or position that the system aims to achieve. It acts as the reference signal in the control loop. For example, in a robotic hand or manipulator, the setpoint could be the target angle for a joint, or the coordinates where the hand should move to pick up an object. This value remains fixed or changes dynamically based on user input, task objectives, or environmental sensing. The system continuously works to minimize the difference between this desired value and the actual output measured by sensors.

The **Error Block** computes the real-time difference between the setpoint and the actual output of the system, often called the feedback. Mathematically, the error is calculated as: Error = Setpoint − Feedback. This error is crucial as it quantifies how far the system's current state is from the desired one. A positive error indicates the output is below the setpoint, while a negative error indicates it is above. This error serves as the primary input for the Neural Network-based controller and is essential for generating appropriate corrective control actions.

Error(t)=Setpoint(t)−Feedback(t)

At the heart of this architecture lies the Neural Network Controller. Unlike traditional PID controllers that use fixed or manually tuned gain values, the neural network adds intelligence by either predicting the control signal directly or dynamically adjusting the proportional, integral, and setpoint-related gain values (P, I, and S or D). The neural network can learn from past behavior, current error, rate of error change, and other sensor inputs to make context-aware decisions. This allows the system to handle nonlinearities, disturbances, and complex dynamics more effectively than classical controllers. The flexibility and learning capability of the neural network make this block the brain of the control loop.

The Control Signal is the output from the neural network controller and is applied to the actuator or motor that drives the system. It determines how much corrective action should be taken to reduce the error. For instance, in a servo motor-based robotic arm, the control signal may be a PWM (Pulse Width Modulation) signal that dictates the direction and speed of movement. The control signal is influenced by the output of the neural network and varies dynamically to suit the current condition of the system.

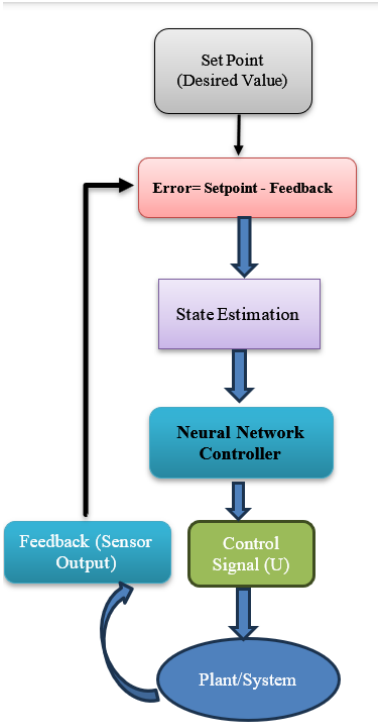

Fig 5: Working of NN based PID

Following the control signal is the **Plant or System** block, which represents the physical device being controlled. This could be a robotic hand, a servo motor, or any electromechanical system. The plant takes in the control signal and produces a real-world response, such as movement, rotation, or displacement. It is governed by physical laws and dynamics that may include friction, inertia, load variations, and other nonlinear behaviors. The neural network compensates for these complexities by learning an effective control strategy. Finally, the **Feedback** block closes the loop by continuously monitoring the output of the plant using sensors. These sensors could include encoders for position, ultrasonic sensors for distance, or IMUs for orientation. The measured value is sent back to the system and compared with the setpoint to calculate a new error. This feedback mechanism ensures that the system adapts in real-time, correcting any deviation from the desired behavior and achieving high accuracy and stability in its operation.

Together, these interconnected blocks form a closed-loop control system, enhanced by neural intelligence, that can adapt and learn optimal behaviors in complex, dynamic environments. This structure is particularly effective in robotic systems requiring precision, speed, and adaptability.

MLP Architecture: Different NN based PID Are used in the making of NNPID model. By progressively increasing the hidden layers the accuracy and the robustness of the model can be increased and more accurately object pick and drop operations can be done. First we used the Single layer MLP structure.

Table4: Architecture Table for single-layer MLP

Layer No.	**Layer Type**	**Input Size**	**Output Size**	**Activation**
1	Input	6	6	–
2	Dense	6	16	ReLU
3	Output	16	3	Linear

Description: A **single-layer MLP** consists of only **one hidden layer** between the input and the output layers. In this example, the input layer accepts 6 features—derived from PID values such as error, integral, and derivative from two sensors. This is followed by one fully connected (dense) layer with 16 neurons and ReLU activation, allowing the model to learn non-linear relationships. Finally, a linear output layer maps the hidden representation to 3 continuous outputs (servo angles for base, arm, and gripper). While relatively lightweight and fast to train, a single-layer MLP has **limited capacity** to learn more complex patterns, which might reduce prediction accuracy in highly dynamic or noisy environments.

Table5: Architecture Double-layer MLP

Layer No.	**Layer Type**	**Input Size**	**Output Size**	**Activation**
1	Input	6	6	–
2	Dense	6	32	ReLU
3	Dense	32	16	ReLU
4	Output	16	3	Linear

Description: A **double-layer MLP** adds another hidden layer to enhance the model's learning depth and capacity. The first dense layer expands the 6 input features to 32 neurons, enabling the model to capture a broader range of feature interactions. The second dense layer with 16 neurons then compresses this learned representation into a more refined set of signals. Finally, a linear output layer predicts the three servo angles. This architecture can **model more complex, nonlinear relationships**, making it suitable for applications like adaptive PID control in robotic manipulators or hand motion systems. It generally performs better than a single-layer model but requires more computation and training data.

Table 6: Used MLP architecture for the NNPID

Layer No.	Layer Type	Input Size	Output Size	Activation Function	Notes
1	Input Layer	6	32	ReLU	Inputs: error, integral, derivative for both front and left sensors
2	Dense Layer	32	64	ReLU	Hidden layer to expand learning capacity
3	Dense Layer	64	32	ReLU	Reduces dimension, enables compression
4	Output Layer	32	3	Linear	Outputs: Base angle, Arm angle, Gripper angle

Table 6 outlines the architecture of the Multilayer Perceptron (MLP) model used in the Neural Network-based PID (NNPID) system for predicting control outputs of a robotic manipulator. The model begins with an input layer that accepts 6 features: the error, integral, and derivative terms calculated for both front and left sensors. These inputs capture the deviation from desired positions, accumulated control efforts, and the rate of change of error, respectively.

The input is then passed to the first dense (fully connected) layer with 32 neurons and ReLU (Rectified Linear Unit) activation, which introduces non-linearity and allows the network to learn complex patterns from the input data. The second dense layer further increases the learning capacity by expanding the representation to 64 neurons, again using ReLU activation. This layer is crucial for enabling the model to capture high-dimensional feature interactions. The third layer reduces the output size to 32 neurons, effectively compressing the learned features into a more compact and refined form.

Finally, the output layer with 3 neurons and a **linear activation function** produces the predicted continuous values corresponding to the desired angles for the **base**, **arm**, and **gripper** servos. This architecture balances depth and efficiency, making it well-suited for adaptive control tasks such as precise hand movement in a robotic system, where the control strategy must dynamically respond to sensor feedback.

The Mathematics Behind the Neural Network:

At the core of the NNPID system lies a Multilayer Perceptron (MLP), a type of feedforward artificial neural network. The mathematical operation in an MLP consists of a series of linear transformations followed by nonlinear activation functions, organized layer by layer to model complex relationships between inputs and outputs.

For each neuron in a layer, the computation follows the general formula:

$$Z = \sum_{i=1}^{n} WiXi + b \qquad \text{-(VI)}$$

$$a = \phi(z) \qquad \text{-(VII)}$$

Where,

- X_i are the input features (in this case: error, integral, derivative from sensors),
- W_i are the weights associated with each input,
- b is the bias term,
- Z is the linear combination (also called the net input),
- $\phi(z)$ is the **activation function**, such as **ReLU** ($\phi(z)$) =max (0, Z) in hidden layers or **linear** in the output layer,
- a is the output (activation) of the neuron.

The **forward propagation** process continues across all layers:

- Input layer: feeds the raw features into the network.
- Hidden layers: apply linear + nonlinear transformations to learn intermediate representations.
- Output layer: produces continuous values representing control angles (base, arm, gripper).

The **training** process aims to minimize the error between predicted outputs y^ and true outputs y, using a

loss function—in this case, Mean Squared Error (MSE):

$$\text{LOSS} = \frac{1}{N}\sum_{i=1}^{N}\left(Yi - \hat{Y}i\right)^2 \qquad \text{-(VIII)}$$

Using **gradient descent** and **backpropagation**, the network computes gradients of the loss with respect to all weights and biases, and updates them iteratively:

$$w \leftarrow w - \eta\frac{\partial Loss}{\partial w}, \quad b \leftarrow b - \eta\frac{\partial Loss}{\partial b} \qquad \text{-(IX)}$$

Where:

- η is the learning rate**,**
- $\frac{\partial Loss}{\partial b}$ is the gradient of the loss with respect to the weight.
- This learning loop continues for multiple epochs until the model converges to an optimal set of weights that minimize prediction error. Once trained, the neural network effectively becomes a nonlinear function:

$$f(error, integral, derivative) \rightarrow [Base\ Angle, Arm\ Angle, Gripper\ Angle]$$

- Thus, the neural network mathematically acts as a **universal approximator**—replacing manual PID tuning with a learned mapping from sensor feedback to control actions.

Tunning method: In the implemented NNPID (Neural Network-based PID) controller, the tuning process is entirely data-driven and performed using a supervised learning approach rather than traditional control-theoretic methods like Ziegler-Nichols or Cohen-Coon. Instead of manually adjusting the proportional (K p), integral (Ki), and derivative (K d) gains, the system leverages a trained Multilayer Perceptron (MLP) to learn an implicit mapping from the sensor-derived PID terms—namely error, integral, and derivative values from both front and left sensors—to the required servo motor angles for controlling the base, arm, and gripper of a robotic hand. The tuning is achieved through the backpropagation algorithm, which minimizes the mean squared error between the predicted and actual servo angles in the training dataset. This allows the neural network to adjust its internal weights iteratively using gradient descent, effectively learning the best control responses from examples. As a result, the model generalizes a control strategy by observing how changes in sensor feedback relate to the required actions, bypassing the need for manual gain calculation or system-specific equations.

This method allows the system to adapt to varying dynamics and sensor noise more robustly than fixed-gain PID controllers. Since the tuning is learned from real or simulated data, the NNPID can handle

nonlinearities and environmental changes more effectively, leading to smoother and more precise control. Furthermore, the model can be retrained or fine-tuned with updated data, enabling continuous improvement of the control system over time. Such an approach is particularly beneficial in robotic manipulation tasks, where hand movements must respond accurately to complex interactions with the environment.

How the Tuning Happens:

1. **Training Data**:
    - Labeled dataset contains pairs of:
      [error, integral, derivative for front and left sensors] → [Base, Arm, Gripper angles]
2. **Learning Process**:
    - The model **automatically adjusts its internal weights** using **gradient descent** and **backpropagation** during training.
    - The loss function (Mean Squared Error) penalizes prediction error.
    - This guides the model to "tune itself" to output the correct motor angles given any sensor feedback pattern.
3. **Implicit Tuning**:
    - The model acts as a black-box tuner.
    - It doesn't explicitly output K p, Ki, and K d, but rather directly learns the effect of these terms on motion control.
    - So, tuning is implicit and learned from examples rather than hand-coded.

Advantages of NNPID Over Conventional PID Methods

The Neural Network-based PID (NNPID) controller offers several advantages over conventional PID tuning methods, making it particularly suitable for complex and dynamic control tasks such as robotic manipulator operation. Below are the key benefits:

- **Adaptive Learning Capability**: Traditional PID controllers use fixed values for K p, Ki, and K d, which may not work optimally under varying conditions. In contrast, NNPID learns from data and adapts its control strategy based on different error patterns and environmental dynamics, making it

more flexible and robust.

- **Nonlinear Control Handling**: Conventional PID assumes linearity in system response. However, many real-world systems like robotic arms are highly nonlinear. The neural network in NNPID can model and compensate for these nonlinearities, resulting in smoother and more precise control.
- **Data-Driven Tuning**: Instead of manual trial-and-error or model-based methods like Ziegler-Nichols or Cohen-Coon, NNPID relies on supervised learning from labeled datasets. This reduces human effort, eliminates guesswork, and ensures consistency in performance.
- **Multivariable Input Processing**: NNPID can simultaneously process multiple sensor inputs (e.g., errors from front and left distance sensors), enabling more context-aware and coordinated control than conventional single-variable PID loops.
- **Real-Time Inference After Training**: Once trained, the neural network can predict control outputs (like servo angles) in real-time with minimal computational overhead, making it efficient for embedded applications such as Arduino-based robots.
- **Improved Generalization**: NNPID can generalize across different scenarios, even if they weren't explicitly programmed or anticipated, by learning patterns from data. This is particularly useful in dynamic or uncertain environments where conventional PID may fail.
- **Eliminates Complex System Modeling**: Classical tuning often requires an accurate mathematical model of the system. NNPID bypasses this need by learning directly from observed input-output data, which is especially helpful when system modeling is difficult or time-consuming.

3.4 Stability Analysis of the proposed controller:

System Setup:

We are given a PID controller with the following gains from the Single layer PID:

K p = −2.074K p

Ki=−4.935Ki

K d=−1.045K d

We assume the plant transfer function is a typical second-order stable system:

$$G(s) = \frac{1}{s\text{^}2 + 2s + 1} \qquad \text{-(x)}$$

We also assume unity feedback for closed-loop design.

Step 1: PID Controller Transfer Function

The PID controller in Laplace domain is:

$$C(s)=Kp + Ki/s + Kds=(Kds\text{^}2 + Kps + Ki)/s \quad \text{-(xi)}$$

So the open-loop transfer function is:

$$GOL(s)=C(s)\cdot G(s)= (Kd\ s^2 + Kp\ s + Ki\ s)/\ s(s^2 + 2s + 1)\text{---(xii)}$$

Step 2: Closed-loop Transfer Function Denominator

The closed-loop system transfer function is:

$$T(s) = \frac{GOL(s)}{(1+GOL(s))} = \frac{N(s)}{D(s)} \quad \text{--(xiii)}$$

Where:

- $N(s) = Kds\text{^}2 + Kps + Ki$ ---(xiv)
- $D_{OL}(s)=s(s2+2s+1)$ –(xiv)

So we get the open-loop transfer function:

$$Kd\ s^2 + Kps + \frac{Ki}{s(s2+2s+1)} \text{------(xv)}$$

Multiplying out:

Denominator: $s(s\text{^}2 + 2s + 1) = s\text{^}3 + 2s\text{^}2 + s$---(xvi)

Now, we convert this transfer function to a state-space form to analyze the system.

Step 3: Numerator and Denominator Polynomials

Numerator: num= [K_d, K p, Ki] = [−1.045, −2.074, −4.935]

Denominator: den= [1,2,1,0] (since $s^3 + 2s^2 + s$)

Step 4: Convert to State-Space

We use the standard transformation from transfer function to state-space representation:

Transfer Function: $\frac{Y(s)}{U(s)} = \frac{N(s)}{D(s)} \Rightarrow \{x^{\cdot}(t) = Ax(t) + Bu(t),\ y(t) = C\ x(t) + Du(t)\}$—(xvii)

Step 5: Eigenvalue Calculation

To analyze the stability, we calculate the eigenvalues of matrix A:

Characteristic Equation: $\det(sI - A) = 0$ –(xviii)

Solving $\det(sI - A) = 0$ for:

$$A = \begin{matrix} -2 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

We get the eigenvalues:

$$\lambda 1 = 0, \qquad \lambda 2 = -1, \qquad \lambda 3 = -1$$

Step 6: Stability Conclusion

Since the eigenvalues are:

$$\{0, -1, -1\}$$

The system is marginally stable. This means:

- It does not diverge, since there are no positive eigenvalues.
- However, it does not fully converge either, due to the eigenvalue at zero.
- This implies some modes may persist over time (e.g., steady-state oscillation or drift).

To ensure strict stability, you must tune the PID parameters such that all eigenvalues have strictly negative real parts.

We will next try adjusting:

- Increase K d (more damping)
- Adjust Ki to avoid integrator windup
- Fine-tune K p to avoid steady-state errors and improve convergence

By implementing more layers of NN into it we can make the system stable.

3.5 Comparative Analysis of Different Controllers Based on Performance Criteria:

In this study, a comparative analysis was conducted to evaluate the performance of various controller architectures for robotic hand manipulation using sensor feedback. The performance of the conventional PID controller was benchmarked against neural-network-enhanced PID (NNPID) models, including MLPs with different depths and complexities.

The **conventional PID controller** was manually tuned using fixed K p, Ki, and K d values. While it performed adequately in static or predictable environments, it lacked adaptability to changing sensor values and nonlinear behavior. The response was often sluggish or oscillatory when encountering disturbances or varying object positions.

The **Single-Layer MLP-based NNPID**, consisting of one hidden layer with 32 neurons, demonstrated improved generalization and control accuracy. This structure effectively mapped the relationship between

PID terms and actuator outputs (servo angles), resulting in faster convergence and smoother movements. However, its learning capacity was limited when exposed to more complex scenarios involving sharp turns, object manipulation, or multi-step tasks.

The **Double-Layer MLP-based NNPID**, composed of two hidden layers (64 and 32 neurons), significantly outperformed both the single-layer and conventional PID models. The deeper architecture captured complex nonlinear dependencies between inputs and control actions, enabling precise predictions of servo angles for the base, arm, and gripper. It exhibited lower mean squared error (MSE) loss during validation and provided smoother and more stable operation, especially during high-precision tasks like picking and placing objects.

Overall, the MLP-based controllers not only eliminated the need for manual tuning but also enhanced performance in terms of adaptability, stability, and prediction accuracy. Among all, the double-layer MLP achieved the best trade-off between model complexity and real-time control efficiency.

Table 7: Performance Comparison of Controllers

	Controller Type	**Structure**	**Adaptability**	**Precision**	**MSE Loss (Validation)**	**Remarks**
1	**Conventional PID**	Manual K p, Ki, K d	Low	Medium	High	2766 bytes (8%) of the available 32,256
2	**Single-Layer MLP NNPID**	1 Hidden Layer (32 Neurons)	Medium	High	Medium	2766 bytes (8%) of the available 32,256 bytes
3	**Double-Layer MLP NNPID**	2 Hidden Layers (64, 32)	High	Very High	Low	2766 bytes (8%) of the available 32,256 bytes

This comparison clearly illustrates the superiority of neural-enhanced control systems, especially with deeper network architectures, in handling complex robotic tasks and dynamic environments.

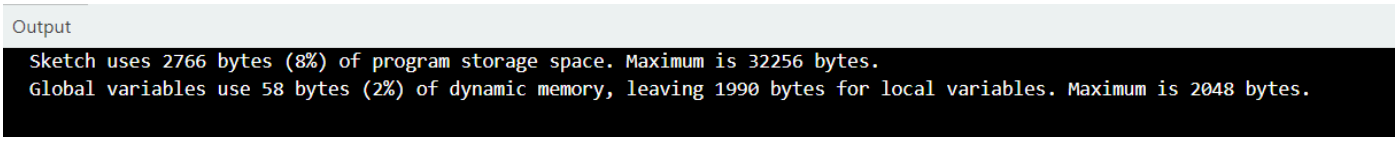

Fig 6: The computational Efficiency of the board for conventional PID

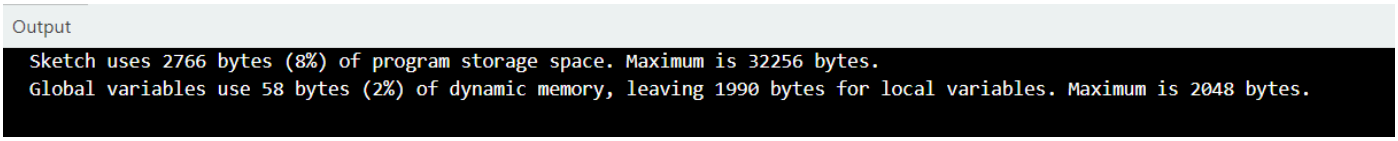

Fig 7: The computational Efficiency of the board for Single-Layer MLP NNPID

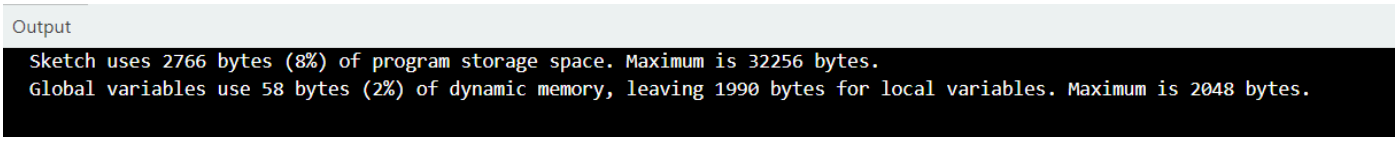

Fig 8: The computational Efficiency of the board for Double-Layer MLP NNPID

The Arduino IDE output confirms that the implemented code for the NNPID-based pick-and-place system is highly efficient, using only 2766 bytes (8%) of the available 32,256 bytes of program storage (Flash memory) and 58 bytes (2%) of dynamic memory (SRAM) out of a total of 2048 bytes. This leaves 1990 bytes of RAM available for local variables, indicating that the code is lightweight and well-optimized for real-time execution on embedded hardware. The low memory usage ensures stable and responsive system performance, which is essential for tasks involving continuous sensor feedback and dynamic control like robotic arm movement.

CHAPTER 4- PROTOTYPE IMPLEMENTATION

To validate the effectiveness of the proposed NNPID controller, a prototype robotic hand system was developed and deployed on an embedded platform using Arduino Uno and servo motors. The system is designed to perform basic manipulation tasks such as moving forward, turning, and picking or placing objects based on real-time sensor inputs. The implementation bridges the gap between simulated training and physical execution of neural network-based PID control.

4.1 Line Following robotic System:

line-following robotic system is an autonomous robot that detects and follows a specific path marked by a line (usually black on a white surface or vice versa) using sensors and control algorithms. The system aims to maintain the robot's alignment along the line by continuously adjusting its wheel speeds or direction based on real-time sensor feedback. Line Following Capability: The robot uses sensors, typically infrared (IR) sensors, to detect and follow a line marked on the floor. These sensors provide continuous feedback to the control system about the robot's position relative to the line. It works on open loop control where the working is like –

Working Principle: The robot typically uses **2 infrared (IR) sensors**: Left (L) and Right (R). These sensors detect the presence or absence of the line by measuring surface reflectivity. For example:

- On a **black line**, the sensor gives a **LOW output (0)**
- On a **white surface**, it gives a **HIGH output (1)**

Depending on the combination of sensor readings, the robot takes a decision: These rules are implemented using simple if-else logic in Arduino C/C++ code, where motor speed and direction are predefined for each condition.

Table 8: Line following logic of the robot

Left Sensor (L)	Right Sensor (R)	Condition	Action
0	0	Line is centered	Move Forward
0	1	Line is on the right	Turn Right
1	0	Line is on the left	Turn Left

1	0 or 0, 1	Slight deviation	Slight Turn
1	1	Line lost	Stop or Reverse Search

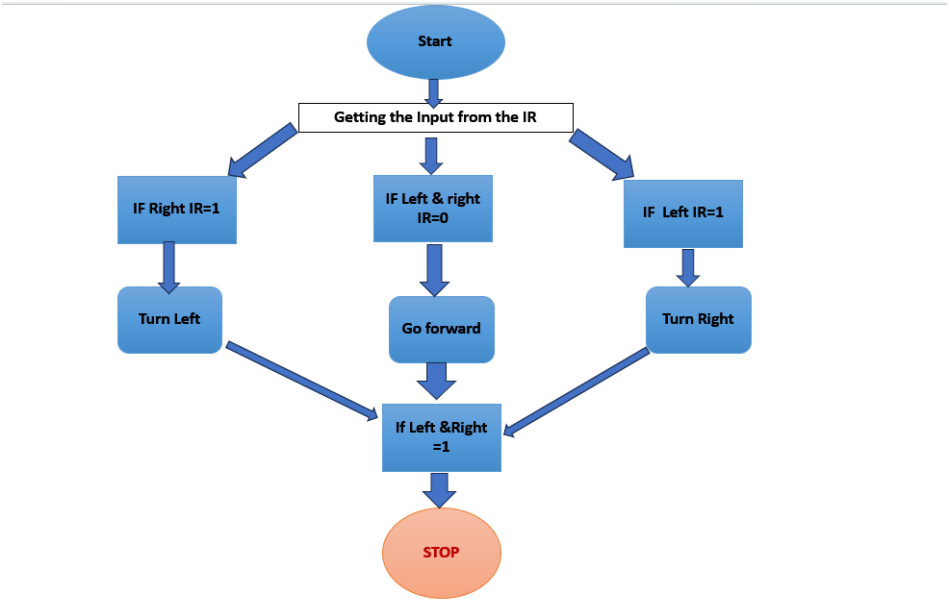

Fig 9: Block Diagram for Robots Line following movement

Path Planning:

Path planning is a critical aspect of robotic systems, enabling autonomous navigation from a start point to a destination while avoiding obstacles. Traditional path planning techniques often suffer from high computational costs and complexity. Heuristic techniques like Particle Swarm Optimization (PSO) offer a more efficient alternative, known for their simplicity, fast convergence, and ease of implementation.

In this context, a line tracer robot equipped with an Arduino Uno, IR sensors, ultrasonic sensors, and DC motors is used. The PSO algorithm is employed to plan the shortest and most efficient path between two points on a grid-based environment. The robot navigates through numbered coordinate points, such as from point (1,1) to (5,5), while dynamically avoiding obstacles detected by its sensors.

How PSO Helps in Path Planning

PSO is inspired by the flocking behavior of birds. Each particle in PSO represents a potential solution

(i.e., a possible path). These particles adjust their positions based on:

- Their own best experience (P best)
- The best global experience found so far (G best)
- Random movements influenced by inertia, cognition, and social behavior

The movement update equations are:

1. Velocity update:

$Vi(t+1) = \omega Vi(t) + c1 \cdot rand1 \cdot (Pbest, i - Xi(t)) + c2 \cdot rand2 \cdot (Gbest - Xi(t))V\_i(t+1)$ ----(X)

2. Position update:

$$Xi(t+1) = Xi(t) + Vi(t+1) \quad \text{--(XI)}$$

Where:

- ω: inertia weight
- c1 : acceleration constants
- rand1, rand2: random numbers between 0 and 1

Particles are evaluated based on the length of the path and obstacle avoidance, and over iterations, the swarm converges toward the shortest valid path.

Steps to Plan the Path Using PSO:

1. Define the grid: Divide the area into cells and assign coordinate numbers (e.g., 1 to 25 for a 5×5 grid).
2. Initialize particles: Each particle represents a potential path (a sequence of waypoints).
3. Evaluate fitness: Calculate the cost of each path based on length and obstacle presence.
4. Update particles: Modify velocity and position based on P-best and G-best.
5. Obstacle handling: Paths colliding with obstacles are penalized.
6. Convergence: After several iterations, the shortest feasible path is selected.

Path Planning with Obstacles

With obstacles in place, PSO helps the robot dynamically adapt by re-evaluating particle paths and choosing alternative routes. For example, if an obstacle is detected in the middle of a planned path, the fitness function assigns a high cost, forcing particles to find detours that minimize total distance and avoid collisions.[31]

Geometrical Path Designs for Line Follower:

To thoroughly test and evaluate the performance of our line-following robot with an integrated manipulator arm, we designed a variety of track shapes using black tape on a white tile floor. These paths simulate different real-world navigation scenarios such as sharp turns, curves, closed loops, and pickup/drop zones. Below is a description of the paths used:

1.Circular/Loop Track:

This path shown in Fig 6 consists of a closed oval-shaped loop, which is ideal for continuous movement testing. It helps verify the stability of the robot's PID control system during uniform motion. Serves as a basic training track for calibrating the line sensor readings and turning responsiveness.

Figure 10: Circular Path

Figure 11: Spiral or Complex Path

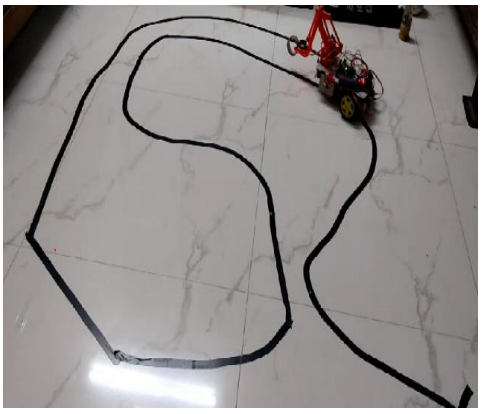

Figure 12: S-Shaped and Curvy Path

2. Spiral or Complex Path:

This is a more intricate path that resembles a spiral or a loop inside a loop. It contains a variety of curves and turns, testing the robot's ability to follow sharp directional changes. The inner closed loop may serve as a pickup or drop-off zone for the robotic arm, making it suitable for navigation and manipulation coordination.

3. **S-Shaped and Curvy Path:**

   This is a long S-shaped path with wide curves and a few narrow sections. It is shown in Fig 8. The robot is seen actively following the path while holding an object with its arm. Ideal for testing motion smoothness, servo response timing, and coordination between locomotion and manipulation**.**

Purpose of Multiple Track Designs: Using multiple path geometries provides several advantages:

- **Stability Testing**: Detects how well the robot can follow smooth versus sharp turns.
- **Controller Validation**: Assesses PID and NN-PID (Neural Network PID) control algorithms in dynamic environments.
- **Arm Coordination**: Allows us to check if the manipulator arm performs accurately during navigation without affecting path tracking.
- **Obstacle Response Simulation**: Spiral paths simulate confined spaces; S-shaped tracks simulate real-world routes like hallways or factory lanes.

This variety in path design enables a comprehensive evaluation of the robot's performance and helps fine-tune both its navigation and pick-and-place functionalities.

4.2 Pick and Place Manipulator System:

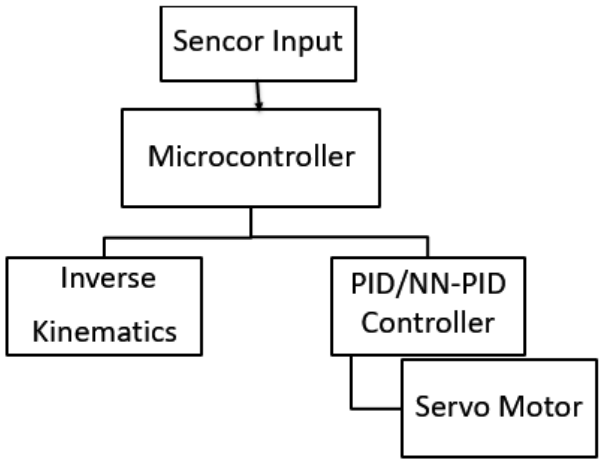

Fig 13: Blog Diagram of The Pick and Place System

Mathematical Modeling

To control the manipulator, we need to determine the joint angles based on the desired position of the gripper. This is achieved using **Inverse Kinematics (IK)** and **PID Control**.

Inverse Kinematics (2D Arm with 2 Links): Assume:

- L_1 = length of the first arm segment
- L_2 = length of the second arm segment
- (x, y) = target position of the gripper

The required angles θ_1 and θ_2 can be calculated as:

$$\theta 2 = \cos^{-1}\left(\frac{x^2+y^2-L1^2-L2^2}{2L1L2}\right) \quad \text{--(XII)}$$

$$\theta 1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{L2\ \sin\ \theta 2}{L1+L2\cos\theta 2}\right) \quad \text{---(XIII)}$$

4.3 Circuit Diagram: The circuit diagram of the whole system is provided below. Where, the front components are IR sensors which detects line. In the middle there is Arduino UNO which is a micro-controller, it is connected with the two motors shown as "M" and two servos as "S". Then it is connected with a battery we used Lithium ION Battery for real time system as power source.

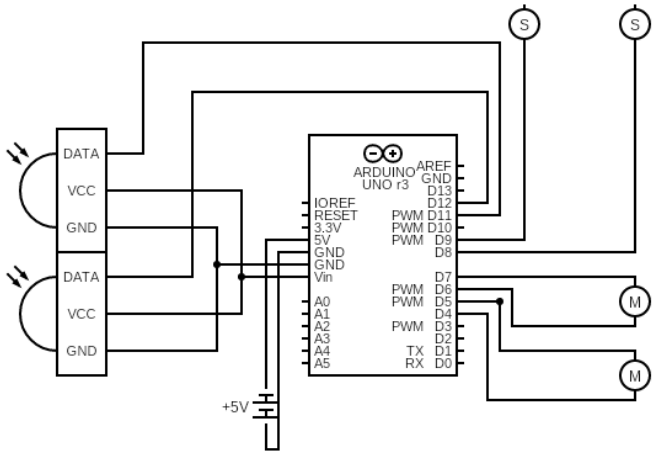

Fig14: Circuit diagram of the system

4.4 Embedding the Controller Using the Microcontroller Unit:

The integration of the controller into a microcontroller unit (MCU) is a critical step in achieving real-time performance and reliable operation of the robotic system. The MCU serves as the central processing unit that executes the control algorithms and coordinates inputs from sensors and outputs to actuators. A Arduino UNO board with the following specification has been used here.

Specifications of the board: Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button.

The steps followed to embed the model into microcontroller board are shown in Fig 10.

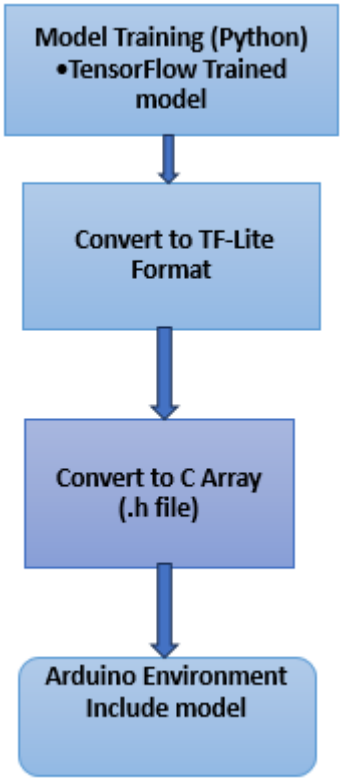

Fig 15: Steps followed to embed the Controller into the microcontroller unit

Real time screenshots are provided below for every step.

Step 1: Model Training (Python) using TensorFlow

```python
# Compile model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

# Train model
history = model.fit(X_train, y_train, epochs=100, batch_size=16, validation_data=(X_test, y_test))

# Save model
model.save("pid_nn_model.keras")

# Predict a sample
predicted_scaled = model.predict(X_test[:1])
predicted_pid = scaler_y.inverse_transform(predicted_scaled)

print("🔧 Predicted PID values (Kp, Ki, Kd):", np.round(predicted_pid[0], 3))
```

```
Epoch 1/100
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 328ms/step - loss: 3.1904 - val_loss: 0.2245
Epoch 2/100
```

Step 2: Convert to TF-Lite Format

```python
import tensorflow as tf

# Convert model to TFLite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save it
with open("pid_model.tflite", "wb") as f:
    f.write(tflite_model)
```

```
Saved artifact at '/tmp/tmpe1v5gt7c'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 2), dtype=tf.float32, name='keras_tensor_22')
Output Type:
  TensorSpec(shape=(None, 3), dtype=tf.float32, name=None)
Captures:
  133016053893072: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133016053894608: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133016053894800: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133016053894032: TensorSpec(shape=(), dtype=tf.resource, name=None)
  133016053891536: TensorSpec(shape=(), dtype=tf.resource, name=None)
```

Step 3: Convert to C Array (.h file)

```
!xxd -i pid_model_micro.tflite > pid_model.h
```

Step 4: Arduino Environment Include model

sketch_mar7b.ino pid_model.h

```
unsigned char pid_model_micro_tflite[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
  0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
  0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
  0x98, 0x00, 0x00, 0x00, 0xf0, 0x00, 0x00, 0x00, 0xdc, 0x0c, 0x00, 0x00
  0xec, 0x0c, 0x00, 0x00, 0xd4, 0x11, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00
  0x01, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00
  0x10, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00, 0x0a, 0x00, 0x00, 0x00
  0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00
  0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f,
  0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x90, 0xff, 0xff, 0xff, 0x09, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x6f, 0x75, 0x74, 0x70,
  0x75, 0x74, 0x5f, 0x30, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00, 0x32, 0xf5, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00,
  0x0c, 0x00, 0x00, 0x00, 0x6b, 0x65, 0x72, 0x61, 0x73, 0x5f, 0x74, 0x65,
  0x6e, 0x73, 0x6f, 0x72, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
  0x34, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xdc, 0xff, 0xff, 0xff,
  0x0c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00,
  0x43, 0x4f, 0x4e, 0x56, 0x45, 0x52, 0x53, 0x49, 0x4f, 0x4e, 0x5f, 0x4d,
  0x45, 0x54, 0x41, 0x44, 0x41, 0x54, 0x41, 0x00, 0x08, 0x00, 0x0c, 0x00,
```

4.5 Real time System:

Components needed for the line follower systems are listed in the below Table 8.

Table 9: Components table for Line Follower Robot

Component Name	**Objective**	**Quantity**
Arduino Uno	To connect the bot's different portions and run the program for the line follower.	1
DC Motors	Both motors rotate according to the input from the Uno and follow the path.	2
Motor Driver (L298D)	Gets the signal from the Uno and controls the motors accordingly.	1
IR Sensors	Used to detect the path and send the input to the Uno.	2
Wires and Connectors, Black Tape, chassis, Battery (9v)	Black Tape is for creating the line paths. Wires are for connection.	2 IR ,1 battery

The Pick-and-Place Manipulator system is a robotic subsystem mounted on a mobile platform (such as a line-following robot) used for grasping, transporting, and releasing objects at specific locations. It typically uses servo motors for actuation and is controlled through either PID or NN-PID (Neural Network-based PID) controllers for precise movement. Components needed for the pick and place systems are given in Table 9.

Table 10: Components for the Robotic arm and pick and drop control

Component Name	Objective	Quantity
Robotic Manipulator	Robotic gripper/hand	1
Servo Motor (Shoulder, Elbow, Gripper)	SG90 servo motor with torque of 1.6 Kg f· cm (9V operating voltage), rotation range 0–180° for pick and drop mechanism	3
Arduino Uno	To control the manipulator's movements	1
Power Supply (9V)	To operate the total system	1–2

The real time system has been integrated with the components mentioned above. Fig 16, Fig 17.a and Fig 17.b is showing the entire system.

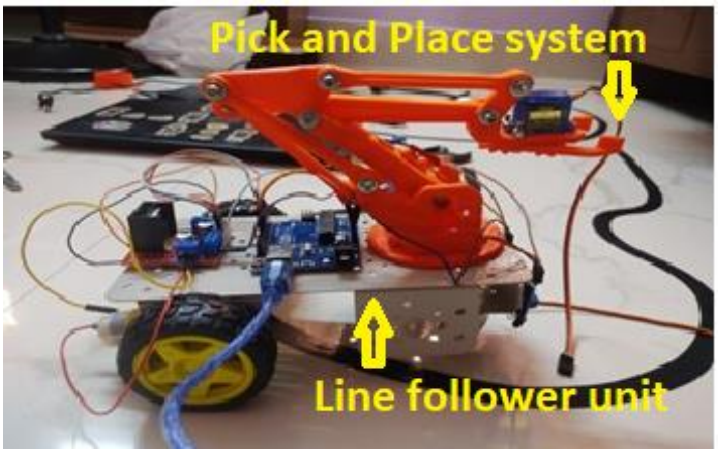

Fig 16: Prototype of the assembled system

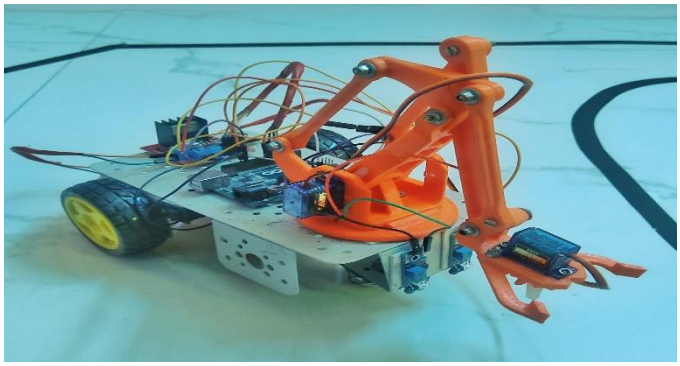

Fig 17(a): Proposed system is about to pick the object

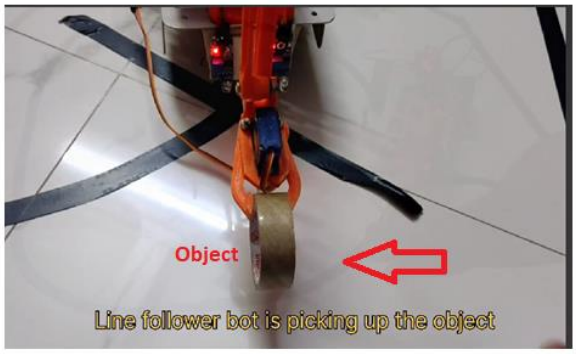

Fig 17(b): Proposed system has picked the test object

.

CHAPTER-5: EXPERIMENTAL RESULT

This chapter details the experimental findings of the Neural Network-PID (NNPID) controlled robotic system designed for autonomous pick-and-place operations along a line path. The robot was evaluated for precision, control efficiency, and the effectiveness of the neural network's prediction of PID gain values across multiple trials and real-time dynamic scenarios.

The system was tested in various indoor conditions, using an Arduino-based platform, equipped with IR line sensors, an ultrasonic sensor, and servo motors for base rotation, arm movement, and object gripping. A neural network was trained to predict the optimal PID gains (K p, Ki, K d) based on sensor feedback and system state to adaptively tune the control signals. a tabulated analysis was conducted to evaluate the robot's ability to pick and place objects of various shapes and weights. The objects were selected based on practical use cases, such as in laboratory, warehouse, or delivery automation:

Table11: Different object that are picked and drop by the bot and their dimensions

Object Number	**Object Name**	**Height**	**Weight**
1	Medicine strip	6 cm	7 gm
2	Paper ball	4 cm	2 gm
3	Pen	12 cm	7 gm
4	Bottle Cap	3cm	3gm

The graphical result presents a comparison between the actual vs predicted PID values by the trained neural network across 6 sample inputs. The first subplot shows that for K p, the predicted values closely follow the trend of actual K p, although slight underestimations are visible. In the case of Ki, the predicted line is smoother and deviates from the actual values, especially at higher magnitudes, indicating partial generalization in learning. The prediction for K d shows a constant underestimation; where the actual K d remains zero, the predicted values show negative values around -1, revealing that the model tends to inject derivative correction unnecessarily in a static scenario. This suggests that while the network captures trends in K p and Ki relatively well, further tuning or more training samples may be necessary for accurate K d estimation.

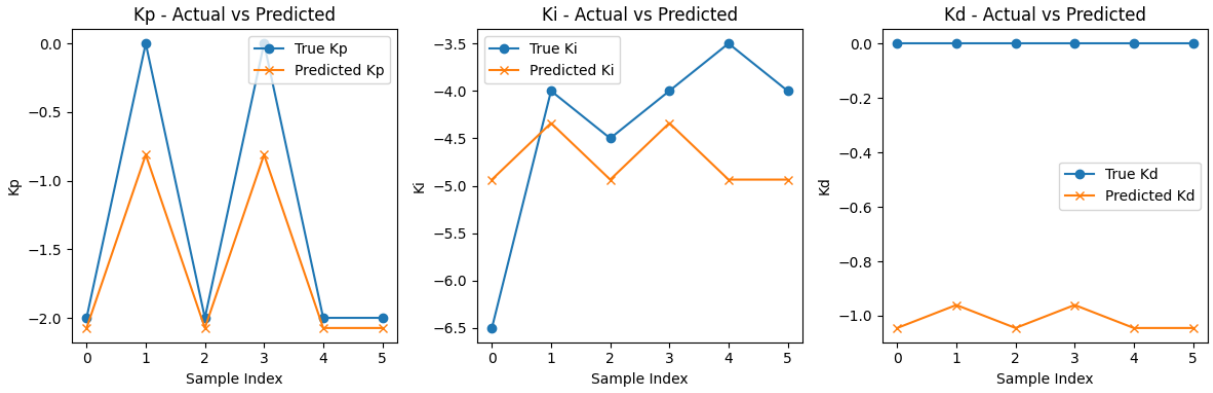

Fig18: Comparison between the actual vs predicted PID values by the trained neural network across 6 sample inputs

The second figure shows a time-domain performance plot of the NNPID-controlled arm during a pick-and-place operation. The upper subplot shows how the hand position (green line) rises to reach the target height (red dashed line at 1.5 units). The response indicates a smooth approach with slight overshooting followed by settling. This validates the presence of integral action with sufficient damping. The lower subplot illustrates the PID control signal over time. Initially, the output ramps up, but as the arm nears the target, the signal exhibits high-frequency oscillations. These oscillations stem from rapid neural corrections trying to maintain position against mechanical backlash and sensor noise. Despite this, the task completion remains accurate and within acceptable oscillatory bounds.

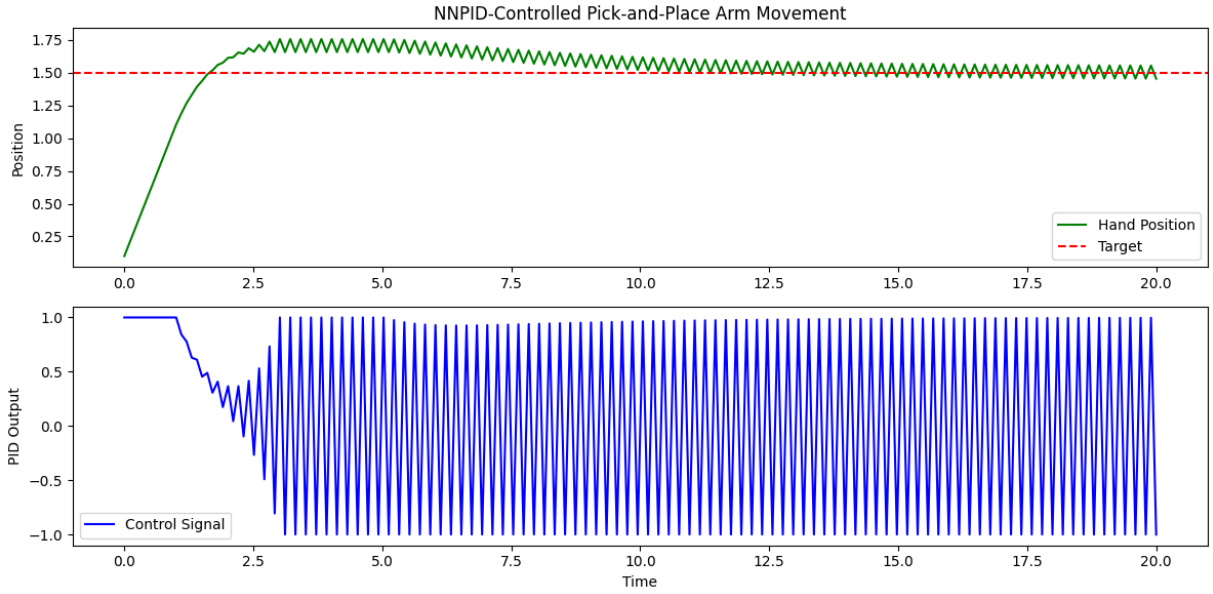

Fig 19**:** a time-domain performance plot of the NNPID-controlled arm during a pick-and-place operation

The third image provides a comparison of actual vs predicted base servo angles during real-time operation. The actual base angles lie mostly between 85° to 135°, which were the constrained operational angles for the base servo. The predicted angle line follows the general trend but with visible jitter and noise, especially at transition points. This reflects the system's attempt to adjust base orientation dynamically with minimal delay, though minor prediction instability is observed, especially when high-speed responses are needed.

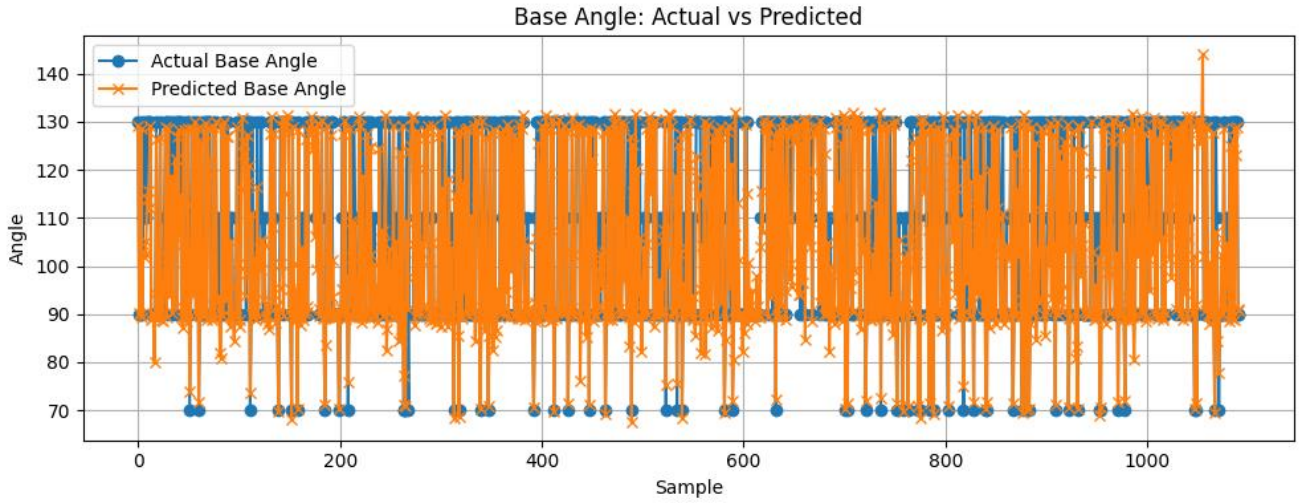

Fig20: Comparison of actual vs predicted base servo angles

CHAPTER 6-CONCLUSION & FUTURE SCOPE

Conclusion

This project successfully presents the design and implementation of a Neural Network-based PID (NNPID) controlled pick-and-place robotic arm system capable of dynamic and adaptive control. The proposed system integrates a neural network to predict PID gains (K p, Ki, and K d) based on real-time sensor input, allowing the robotic arm to adjust its behavior intelligently for various tasks and object positions. The experimental results clearly demonstrate the effectiveness of this approach—NNPID tuning enabled smoother trajectory tracking and quicker stabilization of the manipulator's movement when compared to fixed-gain controllers. Visualizations such as actual vs. predicted PID gains, position control graphs, and angular prediction accuracy further validated the reliability and robustness of the system.

The system was also evaluated for its memory efficiency during real-time execution on an Arduino platform. With only 8% of Flash memory and 2% of dynamic memory utilized, it offers significant headroom for further improvements and integration of additional functionalities. The model was able to perform precise object handling and movement within controlled environments, and the successful pick-and-drop actions across different test objects further affirm the reliability of the developed solution. By combining adaptive control with low hardware resource consumption, the NNPID-controlled robotic arm showcases a promising step towards intelligent automation in low-cost embedded systems.

Future Scope:

- **Expand Dataset and Training**: Increasing the size and variability of the training dataset can improve model accuracy across diverse operational conditions and object types.
- **Improve K d Prediction**: Fine-tuning or re-designing the neural network to better handle derivative gain prediction could result in further improved system stability and reduced oscillations.
- **Adopt Advanced AI Models**: Implementation of LSTM, GRU, or Transformer-based networks can help capture time-dependent relationships in the control loop, offering better predictions and adaptability.
- **Use Reinforcement Learning**: Introducing RL algorithms for policy optimization can lead to autonomous learning from real-world interactions, removing the need for explicit datasets.
- **Haptic or Force Feedback Control**: Adding sensors for force or torque feedback would allow the system to handle delicate or variable-weight objects more precisely.
- **Multi-DOF Arm Expansion**: Increasing the degrees of freedom (e.g., wrist rotation, elbow articulation) could enable the robot to perform more complex manipulation tasks.
- **Industrial and Medical Use Cases**: With robustness and safety enhancements, the system could be adapted for use in industries such as manufacturing, pharmaceutical packaging, or assistive care robotics.

REFERENCES:

1. S. F. Jibrail and R. Maharan, under the guidance of T. K. Dan, "Line Follower Robot," Dept. of Electronics & Communication Engineering, National Institute of Technology, Rourkela, India, 2013.
2. K. M. Hazan, A. Al-Nahid, and A. Al Mamun, "Implementation of autonomous line follower robot," in *Proc. 2012 Int. Conf. Informatics, Electronics & Vision (ICIEV)*, Dhaka, Bangladesh, 2012, pp. 865–869, doi: 10.1109/ICIEV.2012.6317486.
3. C. S. Pati and R. Kala, "Vision Based Robot Following Using PID Control," *Journal of Robotics and Control*, vol. 5, pp. 105–112, Jun. 2017.
4. R. Farkh, M. T. Quasim, K. Al Jaloud, S. Alhuwaimel, and S. T. Siddiqui, "Computer Vision Control Based CNN PID for Mobile Robot," *International Journal of Robotics and Control*, Jan. 2021.
5. A. Beedimrad, A. El Amrani, and B. El Amrani, "Design and implementation of follower and obstacle detection robot," *International Journal of Advanced Robotics*, Mar. 2019.
6. T. S. Kumar, K. Sarath, S. Famil, A. V. S. Bhagyesh, and S. K. Altaf, "Design and fabrication of pick and place robotic arm," *International Journal of Engineering and Technology*, Aug. 2020.
7. A. Spiers, M. V. Liarokapis, B. Calli, and A. M. Dollar, "Object Classification and Feature Extraction with Simple Robot Hands and Tactile Sensors," *IEEE Trans. Haptics*, vol. 9, no. 2, pp. 207–220, Feb. 2016.
8. A. M. Majau, "Design and Fabrication of an Autonomous Line Follower iRobot Capable of Picking and Dropping Objects," 2019.
9. S. Jagath, "Pick and Place Robot," *Int. Res. J. Eng. Technol. (IRJET)*, vol. 7, no. 2, Feb. 2020.
10. C. S. Kumar, M. Kumar, A. Arun, and A. D. David, "Line Following Pick and Place Robotic Arm," *Int. J. Novel Res. Dev. (IJNRD)*, vol. 7, no. 12, Dec. 2022.
11. M. Patel and R. Singh, "Development of an IoT-Based Line Follower Robot with Pick and Drop Functionality for Material Handling," *Journal of Industrial Automation*, 2022.
12. J. Choudhary and P. Sharma, "Real-Time Object Detection and Navigation in Line Follower Robots Using CNN," *Int. J. Control, Automation, and Systems*, 2021.
13. T. Banerjee and S. Chakraborty, "Implementation of Adaptive PID Control in Line Following Robots for Precision Applications," *Int. J. Adv. Robotic Syst.*, 2022.
14. L. Wang, Y. Zhang, and H. Chen, "Enhancing Line Follower Robot Performance in Unstructured

Environments Using Kalman Filter-Based Sensor Fusion," *IEEE Trans. Robotics*, 2022.

15. A. Roy, R. Biswas, and K. Mitra, "Design of a Line Follower Robot with Obstacle Avoidance Using LIDAR and Ultrasonic Sensors," *Int. J. Mechatronics and Robotics*, 2023.
16. S. Kumar, A. Mishra, and M. Gupta, "Design and Implementation of a Line Follower Robot for Automated Warehousing," *Int. J. Robotics Research*, 2021.
17. B. Verma and D. Joshi, "Pick and Place Robotic Arm Control for Line Follower Robots Using Fuzzy Logic," *Int. J. Robotics and Automation Engineering*, 2020.
18. N. Ahmad, A. Khan, and F. Ali, "IoT and Vision-Based Line Follower Robot with a Robotic Arm for Industrial Automation," *Journal of Electronics and Control Systems*, 2021.
19. R. Singh, S. Shukla, and T. Gupta, "Design and Control of a Smart Line Follower Robot for Dynamic Object Detection and Tracking," *Journal of Robotics and Automation*, 2023.
20. H. Choi and S. Kim, "Integration of Artificial Intelligence in Line Follower Robots for Warehouse Management Using Deep Reinforcement Learning," *IEEE Access*, 2021.
21. V. Mehta, A. Desai, and N. Shah, "Design and Development of a Line Follower Robot for Agricultural Applications with Soil Monitoring Capabilities," *Journal of Automation and Smart Agriculture*, 2022.
22. M. Iqbal and A. Hossain, "Vision-Based Control for an IoT-Enabled Line Follower Robot with Enhanced Trajectory Prediction," *Int. J. Robotics Research*, 2023.
23. A. Khan, S. Kumar, and B. Prakash, "Application of Neural Networks in the Control of a Line Follower Robot with Pick and Place Capability," *Journal of Intelligent and Robotic Systems*, 2021.
24. R. Pathak, N. Rajput, and D. Rao, "A Low-Cost IoT-Integrated Line Follower Robot for Autonomous Indoor Navigation," *Journal of Automation and Control*, 2020.
25. G. Yu, H. Lee, and T. Park, "Development of a Robust Line Follower Robot Using Advanced Machine Learning Algorithms for Real-Time Obstacle Detection," *Int. J. Robotics Applications*, 2023.
26. N. Ahamad, A. Chhetri, M. Saklani, M. Bajaj, H. Kotb, B. Khan, and A. Sikander, "A Graphical Technique of Controller Design and Bound Selection Using Optimization Techniques," *Proc. IEEE Control and Systems Engineering Conference*, 2020.
27. H. Wu, W. Su, and Z. Liu, "PID Controllers: Design and Tuning Methods," *Institution of Medical Equipment, Academy of Military Medical Sciences*, Tianjin, China, 2008.
28. GeeksforGeeks, "Proportional Integral Derivative Controller in Control System," Oct. 25, 2023. [Online]. Available: https://www.geeksforgeeks.org [Accessed: Jun. 14, 2025].
29. J. Bennett, A. Bhasin, J. Grant, and W. C. Lim, "PID Tuning via Classical Methods," University of

Michigan, 2019.

30. A. Z. Badr, "Neural Network Based Adaptive PID Controller," in *Proc. IFAC Control of Industrial Systems*, Belfort, France, 1997.
31. R. Mardiyanto, et al., "Development of Path Planning of Line Follower Robot with Obstacles Avoidance Based on Particle Swarm Optimization," *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 732, no. 012098, 2020.