A Project Report

On

DESIGN AND IMPLEMENTATION OF PID CONTROL FOR A DC MOTOR BELT CONVEYOR SYSTEM

by

OLUBOSEDE, OLUMIDE AUGUSTINE

EEE/18/6766

Submitted to:

The Department of Electrical and Electronics Engineering,
The Federal University of Technology, Akure (FUTA).

In Partial Fulfilment Of The Requirements For The Award Of A Bachelor Of Engineering (B.Eng.) Degree In Electrical And Electronics Engineering

Supervisor: **DR. K. B. Adedeji**

October 2024

CERTIFICATION

This document serves as confirmation that the technical report presented provides a comprehensive record of the final-year undergraduate project work completed by **OLUBOSEDE OLUMIDE AUGUSTINE**, with matriculation number EEE/18/6766, in the Department of Electrical and Electronics Engineering, the Federal University of Technology, Akure. The report has been prepared per the regulations governing the production of reports in the department and submitted in partial fulfilment of the requirement for the award of a Bachelor of Engineering (B Eng.) degree in Electrical and Electronics Engineering

Olubosede Olumide Augustine ……

(Student) Signature and Date

Dr. K. B. Adedeji ……

(Supervisor) Signature and Date

Dr. M. R. Adu ……

(Head of Department) Signature and Date

DEDICATION

I dedicate this project to Almighty God, whose grace and guidance have been my strength throughout this journey. His wisdom and unwavering love have been my source of inspiration, providing me with the courage and perseverance needed to see this work through. To Him be all the glory.

ACKNOWLEDGEMENT

First and foremost, I am deeply grateful to Almighty God for His endless grace, wisdom, and strength throughout this project. His guidance has been my foundation, making all things possible. I would like to express my heartfelt appreciation to my dear parents, Dr. and Dr. (Mrs) Olubosede, for their constant love, support, and encouragement. Their prayers and unwavering belief in me have been a pillar of strength.

I would like to express my sincere appreciation to my Head of Department, Dr. M. R. Adu, for their unwavering support throughout my academic journey. A special thanks to my supervisor, Dr. K. B. Adedeji, for his invaluable guidance, constructive feedback, and patience. His mentorship has been instrumental in shaping this work, and I am truly grateful for the time and effort dedicated to my success.

ABSTRACT

This project focuses on the design and implementation of a Proportional-Integral-Derivative (PID) control system for a DC motor belt conveyor system. The aim is to optimise the system's performance by comparing the effects of various control strategies, including open-loop, Proportional (P), Proportional-Integral (PI), and PID control. The DC motor's speed control is critical to ensuring the conveyor's stability and efficiency. The performance indices evaluated are the steady-state error, maximum overshoot, peak time, and rise time. In the open-loop configuration, the system exhibits significant steady-state errors, high overshoots, and slow responses, making it unsuitable for precise speed control. Proportional control reduces the peak time and rise time but results in a higher steady-state error and overshoot. PI control greatly reduces the steady-state error and has the fastest rise time, but suffers from significant overshoot. PID control, however, provides the best performance, eliminating steady-state error with moderate overshoot, faster responses, and reasonable peak and rise times. The comparative analysis demonstrates that PID control is the most effective strategy for optimising the DC motor belt conveyor system, offering a balanced improvement across all performance metrics.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1 Background

In the modern world, the design and development of machinery and robotic systems has reached a very advanced level of precision and automation. Robotics and automated systems have the potential to revolutionise and provide many advantages to the construction industry and to other labour-intensive fields (Davila *et al.*, 2019). Advanced electronic and mechanical systems have found applications in various industries such as assembly lines, healthcare, and consumer electronics. Every day, engineers work to develop new devices and apparatuses, or improve the current ones to obtain better outcomes (Taghavi *et al.*, 2020). The rapid advancement in automated systems is accompanied by higher demands for efficiency, precision, and automatic error correction. As these systems become more complex, there is an increasing need for effective control mechanisms.

The control system is the means by which any quantity of interest in a machine, mechanism, or other equipment is maintained or altered in accordance with the desired manner (Nagrath and Gopal, 2007). A control system is a set of mechanical or electronic devices that regulates other systems through open or closed control loops. Control systems are essentially the brains behind automated processes. They manage the behaviour of other devices or systems using control loops. Control systems are crucial in maintaining the desired output of a machine or process by constantly adjusting inputs based on feedback. Control systems are applied in robotics, industrial machinery, temperature control systems, consumer electronics, and other various fields. A control system can be divided into the controller and the machine (Ellis, 2012). The machine can also be divided into two parts: the plant and the feedback device. The plant receives two types of signals: a controller output from the power converter and one or more disturbances or loads. Simply put, the goal of the control system is to drive the plant in response to the command while overcoming disturbances (Ellis, 2012). The machine in this project is the conveyor belt.

The controller in software is known as a control algorithm, and it is the brain of an automatic control system. Control system design theory is the special theory that studies how control

algorithms are systematically designed and how to ensure their stable and accurate performance (Somefun *et al.*, 2021). In industrial settings, control systems enhance productivity by optimising the performance of machinery, minimising errors and ensuring continuous functioning. Automatic control systems are also used to implement safety features and automatic failsafes. Robust control systems are essential for preventing accidents and ensuring the safety of workers in applications where machines operate under extreme conditions or handle hazardous materials. They can detect anomalies and trigger automatic shutdowns or alerts, preventing potential accidents.

The control system that is widely used in industry is PID. Almost 90% of industries still use PID control systems because of its simplicity, applicability, and reliability (Maghfiroh *et al.*, 2020). PID control uses a feedback control mechanism to regulate processes and systems. It compares the output of the controlled variable (plant or machine) with the desired output and the difference is called the error or offset. The controller then adjusts the plant inputs (controller outputs) according to the PID algorithm to reduce the error. PID controllers are the most widely used type of control mechanism, due to its simplicity, effectiveness, and ease of implementation.

Most electromechanical systems include several sensors, actuators, embedded controllers, such as computers, microcontrollers or programmable chips, and a power supply system (Taghavi *et al.*, 2020). A belt conveyor system is an electromechanical system that moves materials from one place to another. It is composed of a closed loop of flexible belt stretched over rollers that are actuated by electric motors. Belt conveyors are the most commonly used powered conveyors because they are the most versatile and the least expensive. The performance of a conveyor belt system is often measured by its ability to maintain a consistent speed and accurately position items at specific points along the belt. Belt conveyor systems enable the continuous movement of goods and materials in many industries. Industries where belt conveyor systems are applied include mining, food processing, bottling and canning, electronics, and assembly lines. Ensuring the smooth and accurate operation of conveyor belts requires robust control mechanisms. PID control is a simple and effective control mechanism that can be applied in belt conveyor systems.

This project focuses on designing and simulating PID control for a conveyor belt system using a microcontroller. A microcontroller is a small portable computer designed for embedded applications. It contains one or more processors with memory (Taghavi *et al.*, 2020). Microcontrollers are embedded systems which when in a device control the actions and features of

that device. Most often, they control one dedicated task in the device and not all of the device's functions. Microcontrollers are applied in implantable medical devices, process control systems, automobile engine control systems, remote control systems, industrial instrumentation devices, voltmeter, office equipment, electronic appliances (Kondaveeti *et al.*, 2021). This project aims to enhance the performance of the conveyor belt by maintaining precise speed control. This project will explore the design, simulation, and testing of a PID control system for a belt conveyor system.

1.2 Problem Statement

The speed and torque of a motor depends on the nature of the load coupled to the axis of the machine. In the case of a light load the motor develops a relatively high speed and a low torque because it is the load requirement while if the load is heavy, the motor will move at a lower speed and deliver more torque as a higher load demands it (Hammoodi *et al.*, 2020). In industrial conveyor belt systems, it is necessary to control the speed and position of the belt especially in applications like assembly lines and food processing where precision is important. There is also the need to maintain the same speed under different loading conditions. Different applications of belt conveyor systems have different requirements, for example, the conveyor at an airport baggage claim may not require as much precision as the heavy duty conveyor at an assembly plant. Traditional methods of control often struggle to meet the requirements of precision, constancy, and varying operating conditions, leading to inefficiencies such as inconsistent speed and misalignment. The challenge is designing a PID control strategy that can effectively regulate the speed of the conveyor belt, keeping it constant under different loading conditions. Also, generalisation of the controller to control automation tasks at all times is very important (Somefun *et al*, 2021). It also involves tuning of the PID controller and ensuring the stability of the system. The goal is to achieve improved speed regulation, precise positioning, and enhanced overall system reliability, contributing to increased productivity and operational efficiency.

1.3 Aim

The aim of this project is to design and implement PID control for a DC motor belt conveyor system.

1.4 Objectives

The objectives of this project are to:

a. design and construct a DC motor belt conveyor system;

b. design and implement a PID controller on the belt conveyor system using Arduino Uno;

c. evaluate the performance of the system with and without PID control.

1.5 Scope

This project covers the design and construction of a belt conveyor system powered by DC motors, the implementation of PID control, and the system's wiring and programming. Performance evaluation of the system is also conducted. Certain factors, such as slip and the physical characteristics of the conveyor belt, are not considered in this project. Additionally, motor loading is assumed to be ideal.

CHAPTER TWO

LITERATURE REVIEW

2.1 History of PID Control Systems

Control systems and PID controllers have a long history. In 1769, James Watt developed the Watt steam engine and it was accepted as the first negative feedback device (Borase *et al.*, 2021). Elmer Sperry developed the first PID controller in 1911 for the US Navy. In 1939, the Taylor Instrument Companies introduced a completely redesigned version of its 'Fulscope' pneumatic controller which provided a ‘pre-act’ control action in addition to its proportional and reset (integral) control actions. In the same year, the Foxboro Instrument Company added 'hyper-reset' to the proportional and reset control actions provided by their Stabilog pneumatic controller. Pre-act and Hyper-reset control actions are based on the derivative of the error signal while the reset (also known as ”floating”) offers action that is commensurate to the integral of the error signal, and hence both controllers offered PID control (Borase *et al.*, 2021).

The proportional controller was first developed but had the problem of steady-state error. To address this issue, the integral (or reset) control action was introduced, creating the Proportional-Integral (PI) controller, which helped eliminate steady-state error. However, the PI controller still faced issues with large overshoot and sluggish response. To mitigate these problems, the derivative control action (or rate control) was introduced. This led to the development of the Proportional-Integral-Derivative (PID) controller, which combines all three control actions to provide a balanced approach to control system performance, minimising steady-state error, overshoot, and sluggishness. The PID controller is also called the three action controller. PID control systems are the fundamental building blocks of classical and modern control systems. The key reasons for the continuing applications of PID controllers are simplicity in its design, simplicity in its implementation, and the majority of physical systems in various engineering fields can be decomposed in terms of components of first order or second order systems. For these first order and second order systems, the PID controller is a natural candidate because of its simplicity (Wang, 2020).

PID (Proportional-Integral-Derivative) control is a classical control method. There are other schools or approaches to control theory, each with its own methodologies and applications. Some of the prominent ones include modern control theory, adaptive control, nonlinear control, intelligent control, and predictive control. Transfer functions, the basis of classical control theory, require linear, time-invariant (LTI) systems, but no real-world system is completely LTI. For most control systems, the primary solution is to design components close enough to being LTI that the non-LTI behaviour can be ignored or avoided. Nonlinear behaviour can usually be ignored if the changes in parameter values affect the loop gain by no more than a couple of decibels, or, equivalently, about 25%. A variation this small will be tolerated by systems with reasonable margins of stability. If there are larger parameter variations, there are at least three courses of action: modify the plant, tune for the worst-case conditions, or use gain scheduling (Ellis, 2012).

2.2 Review of PID control theory

There are two types of control systems: open loop (non-feedback) control systems and closed loop (feedback) control systems (Hassan, 2008). A PID control system is a type of closed loop control system. In PID control, the output of the plant (controlled variable) is continuously measured and compared with the set point (desired output). The difference between them is the error (or offset). PID control employs 3 control actions, i.e., proportional, integral and derivative. The proportional term applies an actuating signal that is proportional to the control output. This means that the actuating signal is directly related to the magnitude of the error; the larger the error, the stronger the control action. The integral term applies a control action based on the cumulative sum of past errors, which helps to eliminate residual steady-state errors by adjusting the controller output until the error is zero. The derivative term applies a control action that is proportional to the rate of change of the error, providing a predictive action that helps to dampen the system's response and reduce overshoot and oscillations.

Together, these three control actions allow the PID controller to provide a balanced approach to control, addressing both present errors and trends in error changes to achieve precise and stable control of the system. The block diagram representation of a PID controller is shown in Figure 2.1.

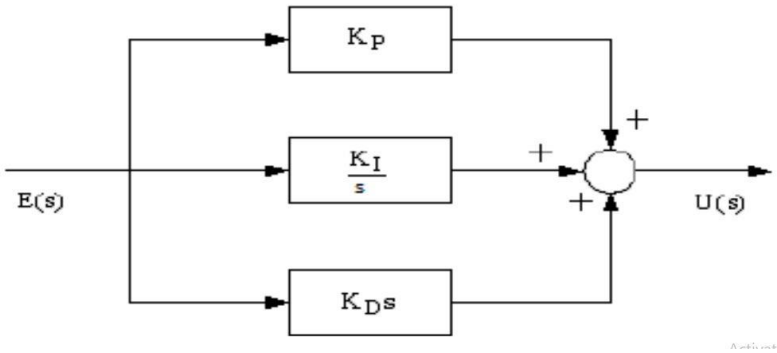

Figure 2.1: Block diagram of a PID controller (Kumar *et al.*, 2020).

Equation (2.1) (Borase *et al.*, 2021) shows the mathematical representation of the PID controller.

$$U(t) = K_C \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de(t)}{dt} \right) \tag{2.1}$$

where the proportional gain is $K_p = K_c$, integral time is T_i and $K_i = \frac{K_c}{T_i}$, and derivative time is T_d and $K_d = K_c . T_d$. K_p, K_i, and K_d denote the coefficients for the proportional, integral, and derivative terms respectively (sometimes denoted *P*, *I*, and *D*) and *e(t)* is the error signal which is the difference between the reference signal (set point) and the actual output. Figure 2.2 shows the block diagram of process control using PID.

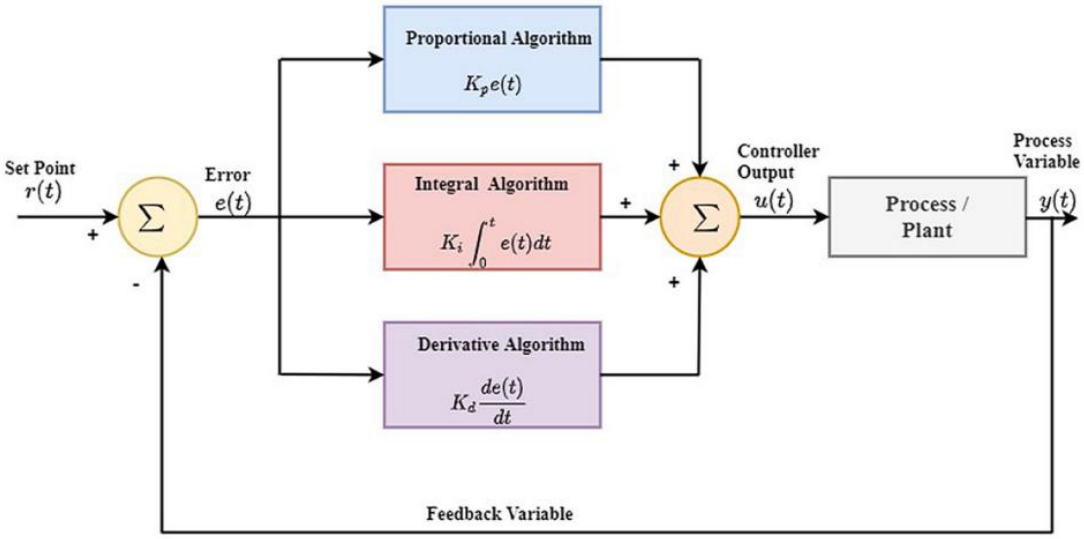

Figure 2.2: Block diagram of process control using PID (Borase *et al.*, 2021)

2.3 Controller Tuning

One weakness of PID is that it takes a long time to tune (Maghfiroh *et al.*, 2020). The process of selecting the controller parameters to meet given performance specifications is known as controller tuning (Ogata, 2010). PID controller tuning is the adjustment of the PID controller's parameters i.e proportional gain (K_p), integral gain (K_i), and derivative gain (K_d), to the optimum values to achieve the desired performance from the system. Tuning a controller means adjusting its parameters until stability is achieved. There are different methods of tuning a PID controller including the manual trial and error method, Ziegler-Nichols method, adaptive tuning, and Cohen-Coon method.

2.3.1 Ziegler-Nichols Method

Ziegler and Nichols suggested rules for tuning PID controllers based on experimental step responses or based on the value of K_p that results in marginal stability when only proportional control action is used. Ziegler–Nichols rules, which are briefly presented in the following, are useful when mathematical models of plants are not known (Ogata, 2010). Ziegler and Nichols proposed rules for determining values of the proportional gain integral time and derivative time based on the transient response characteristics of a given plant. There are two methods called Ziegler–Nichols tuning rules: the first method and the second method (Ogata, 2010). The first method is based on

the step response of the plant. The second method is based on critical gain (or ultimate gain), K_{cr}, and critical period, P_{cr}.

In Ziegler-Nichols second method, we first set the integral time (T_i) to infinity or its largest value and the derivative time (T_d) to zero. Note that this is equivalent to setting the integral and the derivative gains to zero since the integral time, $K_i = K_c/T_i$, and the derivative time, $K_d = K_cT_d$. Using the proportional control action only, increase the proportional gain (K_p) from 0 to a critical value K_{cr} at which the output first exhibits sustained oscillations of constant amplitude. (If the output does not exhibit sustained oscillations for whatever value *Kp* may take, then this method does not apply) (Ogata, 2010).

Thus, the critical gain K_{cr} and the corresponding period, P_{cr}, are experimentally determined. Ziegler and Nichols suggested that we set the values of the parameters K_p, T_i, and T_d as shown in Table 2.1.

Table 2.1: Ziegler–Nichols tuning rule based on critical gain (Ogata, 2010)

Type of Controller	**Kp**	**Ti**	**Td**
P	$0.5 \times K_{cr}$	∞	0
PI	$0.45 \times K_{cr}$	$P_{cr} / 1.2$	0
PID	$0.6 \times K_{cr}$	$0.5 \times P_{cr}$	$0.125 \times P_{cr}$

The Ziegler-Nichols method is a simple and effective PID controller tuning method and it is widely used especially when the plant transfer function is not known. However, it requires a time-consuming experiment and the initial results may need to be fine-tuned for optimal performance. Also, more advanced algorithms have been developed such as the fuzzy logic PID tuning and neural network PID tuning.

2.3.2 Tradeoffs in Controller Tuning

Controller tuning involves making tradeoffs between desired performance indices. When tuning a PID control system, adjusting the proportional, integral, and derivative gains inevitably involves balancing different performance criteria. Each parameter influences multiple aspects of the system's behaviour, such as stability, responsiveness, and accuracy, but improving one performance index often comes at the expense of another. This creates the need for tradeoffs.

A. **Tradeoff between Speed and Stability:** Increasing the proportional gain (P) makes the system more responsive, reducing the rise time and peak time, but it can also increase overshoot and lead to oscillations or instability. Tuning to reduce the rise time may cause the system to become less stable, which could lead to undesirable oscillations.

B. **Tradeoff between Overshoot and Settling Time:** While derivative action (D) can reduce overshoot and improve stability, it tends to increase the settling time. A highly damped system with minimal overshoot might take longer to reach steady state. On the other hand, a system tuned to have minimal settling time could experience higher overshoot.

C. **Tradeoff between Eliminating Steady-State Error and Response Speed:** Adding integral action (I) eliminates or reduces the steady-state error by continuously adjusting the control signal based on accumulated error. However, this can slow down the system's response and lead to integral windup, which increases overshoot and settling time. Therefore, reducing steady-state error typically comes at the cost of slower transient response and possible overshoot.

D. **Sensitivity to Noise and Robustness:** Controllers with high derivative gain are more sensitive to measurement noise because derivative action amplifies high-frequency components. Thus, tuning for aggressive derivative action to improve stability and reduce overshoot must be balanced against the risk of making the system too sensitive to noise, potentially destabilising the control process.

E. **Practicality and System Constraints:** Tuning for optimal theoretical performance may also conflict with practical considerations such as actuator limitations, sensor noise, or

physical constraints of the system. For instance, a high controller gain might result in faster actuator wear, requiring a balance between performance and system longevity.

Controller tuning is about finding the right balance between competing objectives—such as speed, stability, accuracy, and robustness—to meet the performance requirements. For this project, the most important performance characteristic of the conveyor belt system is the ability to maintain a constant movement speed. Thus the steady-state error and response time are the most important parameters and minimising them takes priority.

2.4 DC motors

DC drives are less complex with a single power conversion from AC to DC. Again the speed torque characteristics of DC motors are much more superior to That of AC motors. DC motors provide excellent control of speed for acceleration and deceleration. DC drives are normally less expensive for most horsepower ratings. DC motors have a long tradition of use as adjustable speed machines and a wide range of options have evolved for this purpose (Bansal and Narvey, 2013). DC motors are widely applied in many areas like industrial machines and consumer electronics. Although DC motors are designed in various ways, they all contain the following basic parts:

a. Rotor (or armature): This is the part of the machine that rotates.
b. Stator (field windings): This is the stationary part of the motor.
c. Commutator: This can be brushed or brushless, depending on the motor type.
d. Field magnets: These provide the magnetic field that rotates the motor.

According to Hammoodi *et al.* (2020), there are five types of general purpose DC motors. These are:

a. Independent excitation direct current motor.
b. Direct current motor in derivation.
c. Permanent magnet direct current motor.
d. Direct current motor in series.
e. Composite direct current motor

DC motor control can be implemented with a variety of methods including voltage control, pulse-width modulation (PWM), and current control. This project is based on a belt conveyor system actuated by DC motors. Therefore, it is necessary to consider the mathematical model of a DC motor. A motor presents its own dynamics, therefore, a motor model should be adopted and implemented appropriately to complete the accurate modelling of the system. A typical DC motor is described by the equation (2.2) (both for electrical and mechanical parts) (Katsioula *et al.*, 2018).

$$V_a = E_a + I_a R_a + L_a \left(\frac{dI_a}{dt}\right), \tag{2.2a}$$

$$T_{em} = T_L + T_f + T_{out} = k_t \phi I_a, \tag{2.2b}$$

$$T_f = B_m \omega_m, \tag{2.2c}$$

$$E_a = k_t \phi \omega_m, \tag{2.2d}$$

$$\omega_m = \frac{d\theta}{dt}, \tag{2.2e}$$

$$T_{out} = J_m (d\omega_m / dt) \tag{2.2f}$$

where V_a is the armature voltage, E_a is the back-EMF, I_a is the armature current, R_a is the armature resistance and L_a is armature inductance. Also, T_{em}, T_f , T_{out} and T_L are the electromagnetic torque, friction torque, output torque, and load torque respectively. Finally, B_m, Φ and ω_m are the viscous friction coefficient, the magnetic flux, and the rotor's mechanical speed (Katsioula *et al*, 2018). K_t is torque constant.

A problem with applying a conventional PID control algorithm in a speed controller are the effects of non-linearity in a DC motor. Generally, an accurate nonlinear model of an actual DC motor is difficult to find and parameters obtained from systems identification may be only approximated values. Fuzzy Logic Control (FLC) can provide non-linear controllers that are capable of performing different complex nonlinear control actions, even for uncertain nonlinear systems. Unlike conventional control, designing a FLC does not require precise knowledge of the system model (Bansal and Narvey, 2013).

2.5 Belt Conveyor System

A belt conveyor is a system designed to transport materials, goods, even people from one point to another. Unlike other conveying means that employ chains, spirals, hydraulics, etc., belt conveyors will move the items using a belt. It involves a loop of a flexible material stretched between rollers that are actuated by an electrical motor. A standard belt conveyor system has a head pulley, tail pulley, idler rollers, belt, and frame. There are different types of belt conveyors including Roller Bed Belt Conveyor, Flat Belt Conveyor, Modular Belt Conveyor, Cleated Belt Conveyor, Curved Belt Conveyor, Incline/Decline Belt Conveyor, Sanitary Washdown Conveyor, Troughed Conveyors, and Magnetic Belt Conveyor (IQS Directory, 2024). Figure 2.3 shows the components of a simple conveyor.

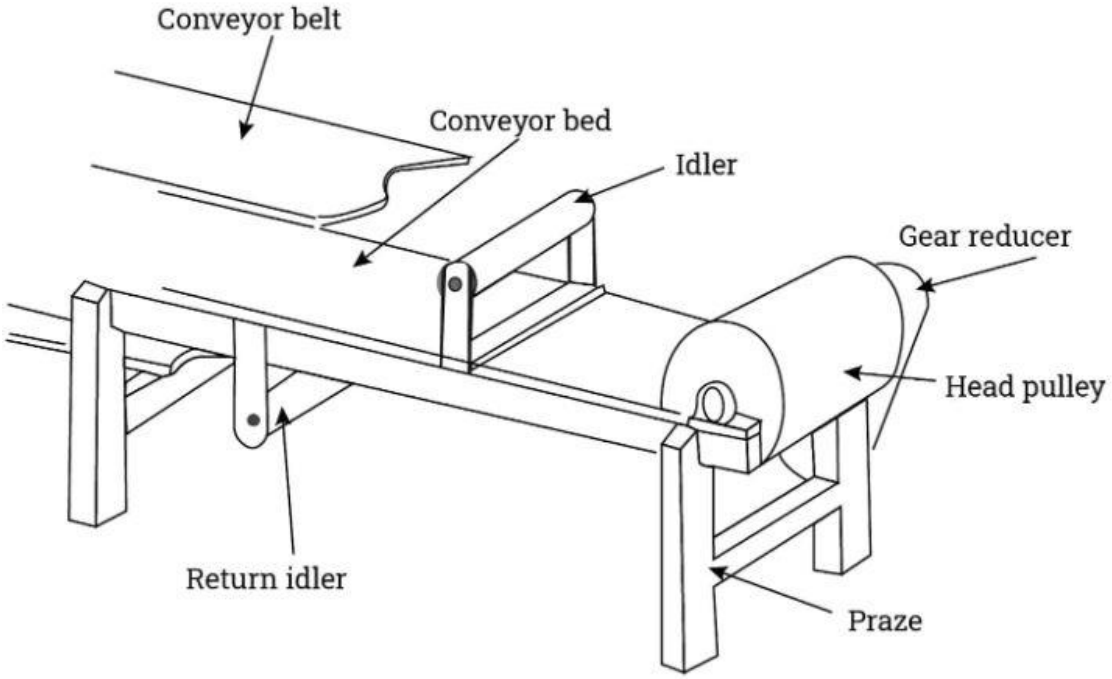

Figure 2.3: Simple conveyor components (IQS Directory, 2024)

When designing a conveyor belt, the major parameters to be considered are the motor and gearbox selection, the speed of the belt, the tension and take-up of the belt, the material to be conveyed, the distance over which to be transported, and the working environment e.g. temperature, humidity, etc.

2.5.1 Motor and Gearbox Selection

To aid the selection of the motor, one must first know what the effective pulling force required for the conveyor is. For a simple horizontal conveyor, the effective pulling force is given in equation (2.3) (IQS Directory, 2024).

$$F_U = \mu_R \times g(m + m_b + m_g) \tag{2.3}$$

where F_u is the effective pulling force, μ_R is the friction coefficient when running over roller, g is the acceleration due to gravity, m is the mass of goods conveyed on the whole length of the conveyor, m_b is the mass of the belt, and m_R is the mass of all rotating rollers minus mass of drive roller.

For a system on an incline, the effective pulling force is given in equation (2.4) (IQS Directory, 2024).

$$F_U = \mu_g \times g \times (m + m_b + m_g) + (g \times m \times sin(a)) \tag{2.4}$$

where F_u is the effective pulling force, μ_R is the friction coefficient when running over roller, g is the acceleration due to gravity, m is the mass of goods conveyed on the whole length of the conveyor, m_b is the mass of the belt, m_R is the mass of all rotating rollers minus mass of drive roller, and a is the Angle of inclination.

2.5.2 Speed of Conveyor

The speed of a conveyor will be the circumference of the drive pulley multiplied by the revolutions per unit time. The equation for the speed of a conveyor is given in equation (2.5) (IQS Directory, 2024).

$$V_c = D \times F \tag{2.5}$$

where V_c is the speed of the conveyor belt in m/s, D is the diameter of the drive pulley in metres, and F is the revolutions of the drive pulley per second.

2.6 Review of Related Studies

The design and simulation of Proportional-Integral-Derivative (PID) controllers for DC motor-driven belt conveyor systems have been the subject of extensive research. Kumar *et al.*. (2020) developed a PID controller for controlling the speed of a DC motor using LabVIEW 2011. They calculated the DC motor transfer function, tuned the PID controller, and developed the software for both software development and hardware implementation. Hammoodi *et al.*. (2020) designed and simulated a closed-loop speed control system for a brushed DC motor to maintain a constant speed despite load variations (disturbances). They used PID control to achieve this in MATLAB/Simulink. Todkar *et al.*. (2018) designed a heavy duty belt conveyor system for use in the coal processing industry. They created an exact 3D Computer Aided Design (CAD) model of the system using CAD software. They also carried out tension calculations and stress analysis and examined its effects on the belt conveyor components.

CHAPTER THREE

SYSTEM DESIGN AND METHODOLOGY

3.1 Activity Flow for the Project

Figure 3.1 shows the activity diagram for the project. The first step is to design and construct the physical conveyor belt system. The Arduino Uno microcontroller was used. The Arduino Uno is perhaps the most popular Arduino board in the market and the industry (Kondaveeti *et al*, 2021). The second step is to obtain the open loop response of the motor. The next step is to design and implement a Low Pass Filter (LPF) to remove the high frequency oscillations in the response. Then the proportional, proportional-integral, and proportional-integral-derivative control mechanism is implemented and the responses are obtained. Finally, the performance metrics for the various control mechanisms are compared.

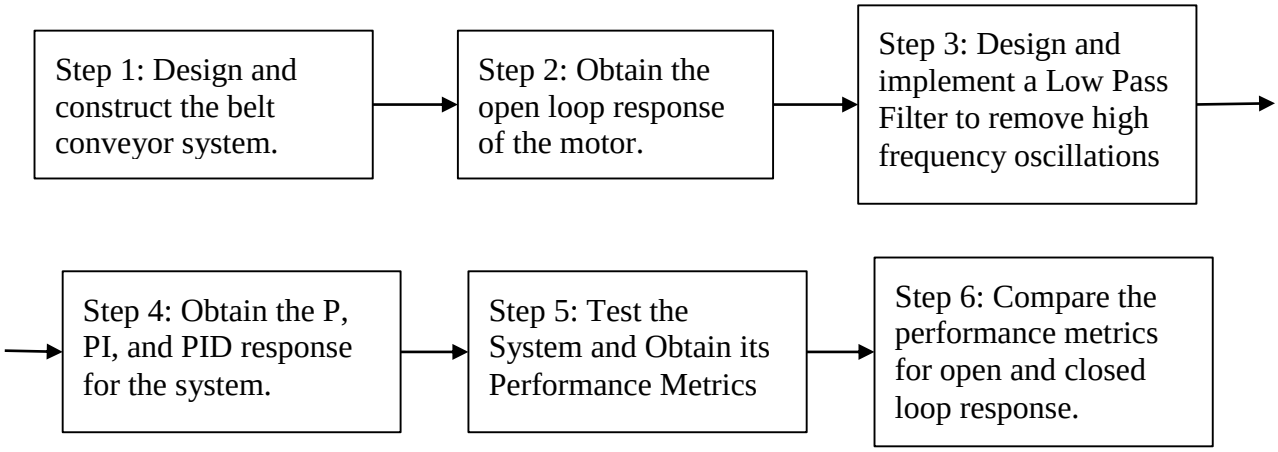

Figure 3.1: Activity flow diagram for the project

3.2 Components Used and System Schematic

The components used for the project are itemised in Table 3.1.

Table 3.1: Components used and their specifications

S/N	Component	Description	Specification
1	Arduino Uno	Microcontroller board based on the ATmega328P (Arduino, n.d.)	Operating voltage: 5V Digital I/O pins: 14 (6 PWM pins) 6 analog input pins 16 MHz ceramic resonator 1kb EEPROM
2	DC Motor	Electric motor that converts electrical energy into mechanical energy	Voltage: 12V RPM: 100-1500 Torque: variable
3	L298N Motor Driver	High power dual H-bridge motor driver for controlling DC and Stepper Motors (Components101, n.d.)	Max supply voltage: 5V-35V Max output current: 2A per bridge Logic Voltage: 5V Logical Current:0-36mA Maximum Power (W): 25W
4	Breadboard	Solderless board for prototyping circuits	Standard ½ size: 420 tie-points
5	Adaptable box	Protective casing for electronics	Adjustable size Material: plastic/metal
6	Leather	Material for the conveyor belt	Thickness: 1.5mm-3mm Flexible and durable
7	Rigid Motor Coupler	Connects motor shaft to another rotating element	Inner diameter: 8mm
8	1602 LCD	Display module with 16x2 character layout	Voltage: 5V Backlight Interface: parallel
9	I2C LCD interface	Adapter module to simplify LCD connection to Arduino	I2C address: 0x27, 4-pin connection

10	12V, 4A DC Adapter + DC socket	Power supply for electronic components	Voltage: 12V Current: 4A Connector: DC jack
11	Optical slot motor speed sensor	Measures the speed of rotating objects	Operating voltage: 3.3V-5V Slot width: 5mm
12	Jumper Wires (Set)	Wires used to connect components on the breadboard	Length: 10cm, Male to male, female to male
13	Ball bearings	Reduces friction between moving parts	Inner diameter: 8mm
14	20 grid speed counting disk	Disk with slots used for speed measurement	Diameter: 25mm, 20 equally spaced grids
15	Pipe (for rollers)	Used to create rollers for the conveyor system	Diameter: 50mm Material: Plastic
16	Wood	Material for the frame and support of the conveyor	Type: plywood or hardwood, Thickness: 15mm

Figure 3.2 shows the circuit diagram for the system and Figure 3.3 is a picture of the completed project.

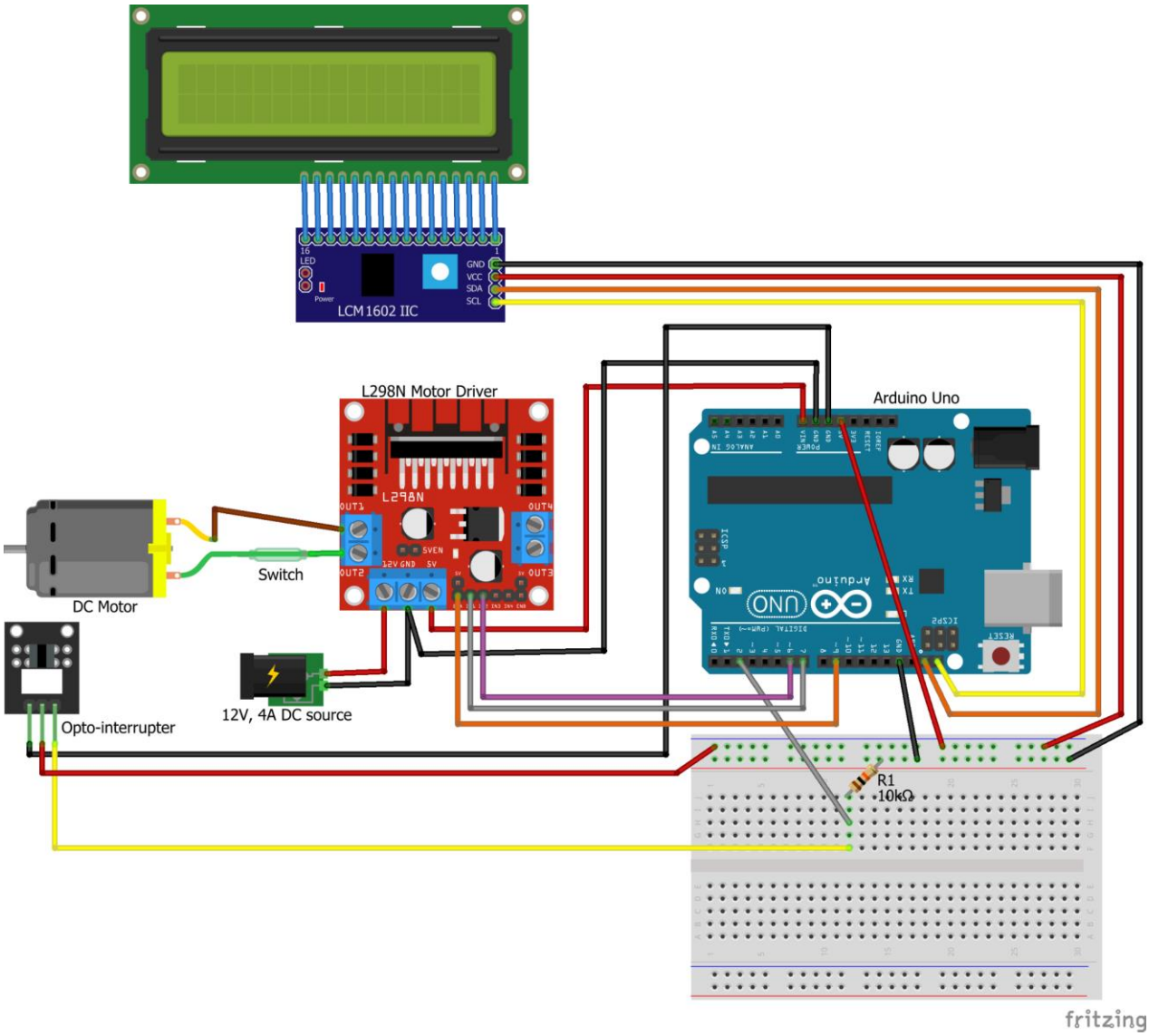

Figure 3.2: Circuit diagram for the hardware project

In this project, the Arduino Uno acts as the brain, controlling everything. It's connected to an L298N motor driver, which powers a DC motor. The motor gets its power from a 12V DC source, and the Arduino sends control signals to the motor driver to adjust speed and direction. An optical slot sensor, or opto-interrupter, sits near the motor, measuring its speed. It does this by detecting interruptions in light as a part of the motor spins. The sensor sends data back to the Arduino, which then displays the speed on a 16x2 LCD. The LCD uses an I2C interface to reduce

the number of connections, making the wiring cleaner. Every piece works together, with the Arduino reading from the sensor and adjusting the motor's operation, all while displaying real-time feedback on the screen.

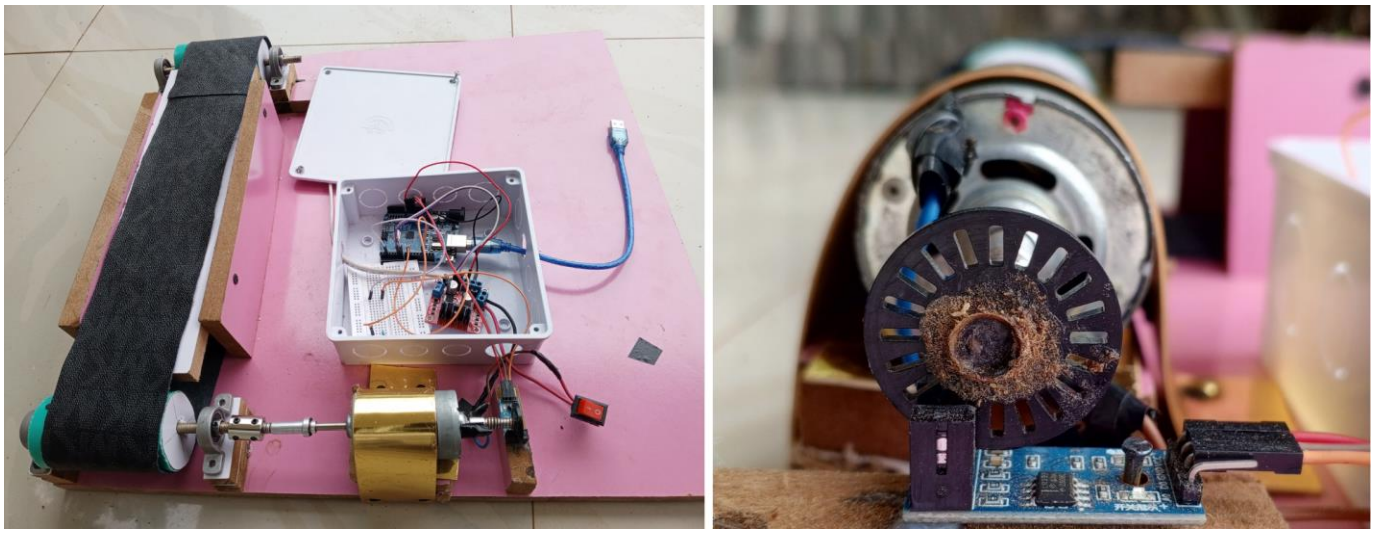

Figure 3.3: Conveyor belt system (left) and close-up of the speed measurement mechanism (right)

3.3 Bill of Engineering Measurement and Evaluation

The components used, along with the bill of engineering measurements and evaluation for the project, are shown in Table 3.2. The components were sourced in different ways: some were ordered from online vendors like AliExpress and Hub360.com, while others were purchased locally. A total of ₦108,400 was spent.

Table 3.2: Bill of Engineering Measurements and Evaluation for the Project

S/N	Item	Quantity	Rate (₦)	Amount (₦)
1	Arduino Uno	1	10000	10,000
2	L298N Motor Driver	1	3700	3,700
3	DC Motor	1	17000	17,000
4	Jumper Wires (Set)	2	1240	2,480
5	Breadboard	1	1300	1,300
6	Adaptable box	1	1500	1,500
7	Leather	1	1500	1,500
8	Wood + carpentry			16,000
9	Rigid Motor Coupler + delivery	1	9000	9,000
10	1602 LCD	1	3700	3,700
11	I2C LCD interface	1	1600	1,600
12	12V, 4A DC Adapter + DC socket	1	4700	4,700
13	Pipe (for rollers)			1,700
14	Optical slot motor speed sensor	1	1100	1,100
15	20 grid speed counting disk	1	120	120
16	Ball bearings	4	2000	8,000
17	Delivery Fees			10,000
18	Transportation			5,000
19	Miscellaneous			10,000
	Total			**108,400**

3.4 Algorithm for Calculating the Speed

After obtaining the components and wiring them according to the circuit diagram, the Arduino Uno is programmed with Arduino IDE. The speed is calculated using the optical slot sensor and the 20 grid speed counting disk. Since the speed counting disk doesn't physically touch the sensor, there is minimal wear and tear, ensuring longevity and reliability.

In Arduino, an interrupt is attached to the sensor via the attachInterrupt() function, which triggers every time the optical sensor detects a rising edge (when an object like a hole or slot passes through the sensor). Each time the sensor detects this event, it increments a counter variable. This counter stores the number of interruptions that have occurred. The 20 grid speed counting disk is attached to the shaft of the motor and it is positioned such that it fits into the slot of the optical slot sensor. In this arrangement, one complete revolution of the motor shaft will produce 20 pulses from the sensor pin. To measure the speed of rotation using the arduino, the time in microseconds (for very high precision) is logged on every loop pass. The velocity is calculated with the formula shown in equation (3.1).

$$V = (\frac{C}{20}) \div \Delta T \tag{3.1}$$

Where *V* is the velocity, *C* is the number of pulses detected since the last reading, 20 is the number of pulses per revolution (determined by the number of slots in the grid speed counting disk), and ΔT is the time interval in seconds between the current and previous speed calculation. The resulting speed is in rotations per second (RPS), which is then multiplied by 60 to convert it to rotations per minute (RPM). The exact code implementation can be seen in Appendix A.

3.5 Procedure for Data Capture and Visualization

The responses are printed in the Arduino terminal using Serial from the Arduino library. However, it is impossible to copy the data of that size from the Arduino terminal due to the scroll limits, limited buffer size, and copying limits. To resolve this problem, CoolTerm, an external software, is used.

CoolTerm is a freeware serial port terminal app and networking program, developed by Roger Meier for Windows. It is a software for communicating with serial devices via USB and Bluetooth, often used by engineers and hobbyists. CoolTerm was developed as a useful and user-

friendly software and acts as a serial port terminal application (CoolTerm, n.d.). In this project, CoolTerm is used to directly log the data to a text file.

3.6 Project Procedure

3.6.1 Obtain the Open Loop Motor Response

First, the circuit is connected according to the circuit diagram shown in Figure 3.2. Then, the Arduino Uno is programmed with the code to obtain the open loop motor response. The relevant code can be found in Appendix A. The motor pins are configured, and an interrupt is set up to increment a pulse counter whenever the sensor detects a rotation (indicated by a rising edge). It initialises the motor using PWM for speed control. The Arduino is programmed to log the readings for 10 seconds. It reads the pulse count in an atomic block to prevent misreads. An atomic block refers to a section of code that is executed without interruption. This means that once the program enters this block, it will complete all the instructions within it before allowing any other interrupts to occur. This concept is particularly important in embedded systems where timing and data integrity are critical. Then, it calculates the motor's speed in RPM based on the number of detected pulses and the time elapsed since the last reading. The current time and calculated speed are logged to the Serial Monitor in CSV format for further analysis.

3.6.2 Design the Low Pass Filter (LPF)

The main function of filters is to suppress or filter out components from mixed frequency signals. Essentially, they allow a certain range of frequencies to pass, which is known as “pass band,” rejecting (or suppressing) all other frequencies that are called “stop band.” The cut-off frequency is the parameter that separates these two bends. Depending on the pass and stop bends, there are four types of filters (low-pass, high-pass, band-pass, and band-reject filters). LPFs allow all of the frequencies that are lower than its cut-off frequency to pass while stopping all others (Bhatt, 2014). When designing a low pass filter, the first choice you make is whether to design an FIR or IIR filter (MathWorks, n.d.). If the impulse response of the filter falls to zero after a finite

period of time, it is an FIR (Finite Impulse Response) filter. However, if the impulse response exists indefinitely, it is an IIR (Infinite Impulse Response) filter. The advantage of IIR filters over FIR filters is that IIR filters usually require fewer coefficients to execute similar filtering operations, that IIR filters work faster, and require less memory space (National Instruments, n.d.). These are the reasons why this project implements an IIR low pass filter. Common filter types include the Butterworth filter, Chebyshev filter, and Bessel filter.

The initial speed response graph derived from this project is coarse and there is high frequency oscillations between values. There is a need to filter the response to make it smoother. A Low Pass Filter (LPF) is required to attenuate the strength of the high frequency component. The transfer function of the designed filter is shown in equation (3.2):

$$H(z) = \frac{0.0728z + 0.0728}{z - 0.854} \tag{3.2}$$

The discrete update equation for the designed LPF is shown in equation (3.3).

$$v\_filt[n] = 0.854 * v\_filt[n-1] + 0.0728 * v[n] + 0.0728 * v[n-1] \tag{3.3}$$

where *v_filt[n]* is the value of the filtered velocity for the nth iteration, *v_filt[n - 1]* is the filtered velocity for the (n - 1)th iteration, *v[n]* is the unfiltered velocity for the nth iteration, and *v[n-1]* is the unfiltered velocity for the (n - 1)th iteration.

Filtering reduces the jitter and makes the response much smoother, however, it also introduces phase lag (delay) into the response. This delay occurs because the filter gives more weight to past data (previous samples) when calculating the current output, causing the response to lag behind the actual changes in the input. So designing a filter involves a tradeoff between smoother response and phase lag (delay). The speed response in the graphs shown in this report moving forward has been filtered. The exact filter implementation can be seen in Appendix A.

3.6.3 Implement Proportional Control

Proportional Control is a simple and effective control strategy used in various applications, such as motor speed control, temperature regulation, and position control. In P control, the output of the controller is directly proportional to the error, which is the difference between a desired setpoint and a measured process variable.

3.6.4 Implement Proportional-Integral Control

Proportional-Integral (PI) Control is a widely used control strategy in industrial applications for systems requiring precise regulation. A PI controller helps in the situation where proportional control is necessary to speed up the settling and integral control is necessary to reduce the error that is constant over time (Sandeep, 2021). It combines the benefits of proportional control with an integral action to eliminate steady-state error. The integral component addresses accumulated error over time. It integrates the error to eliminate steady-state error, improving accuracy. The control output for PI control is given in equation (3.4).

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)dt \tag{3.4}$$

where *u(t)* is the control output, *Kp* is the proportional gain, *Ki* is the integral gain, and *e(t)* is the error. The *integral_error* variable accumulates the error over time. By summing the error multiplied by the elapsed time (the time since the last update), the integral term helps account for the accumulated error that has occurred over time. The *elapsedTime* variable stores the time interval since the last control update. Multiplying the error by the elapsed time gives the integral term's contribution based on how long the error has persisted. The integral term is essential in eliminating steady-state error, helping to ensure the system reaches and maintains the target speed.

3.6.5 Implement Proportional-Integral-Derivative (PID) Control

PID Control is a widely used control strategy that combines three control actions—proportional, integral, and derivative—to achieve precise control of dynamic systems. The derivative component predicts future error by considering the rate of change of the error. It helps dampen the system response, reducing overshoot and improving stability. The control output *u(t)* for a PID controller is expressed in equation (3.5).

$$u(t) = Kp \cdot e(t) + Ki \cdot \int e(t)dt + Kd \cdot \frac{de(t)}{dt} \tag{3.5}$$

where *Kd* is the derivative gain.

The error value from the previous iteration of the control loop is stored. It is then used to compute how much the error has changed over time. So the derivative error is the rate of change of the error over time. It is obtained by subtracting the previous error from the current error and

dividing by the time that has elapsed. This term helps predict the future behaviour of the error by examining how quickly it is changing. If the error is increasing rapidly, the derivative term will provide a larger corrective action to counteract this trend. The load response can then be gotten and plotted. Finally, the performance indices for the different control techniques can be compared and analysed.

CHAPTER FOUR

RESULTS AND DISCUSSIONs

4.1 Data Capture and Visualization

Figure 4.1 shows a CoolTerm window reading serial data from the Arduino. The data is logged in a CSV format and the text file is converted to a CSV file and it is opened in Microsoft EXCEL for visualisation. The data generated is readings of time in milliseconds and the speed at that time. This can be plotted in Excel using a “Scatter plot with straight lines”.

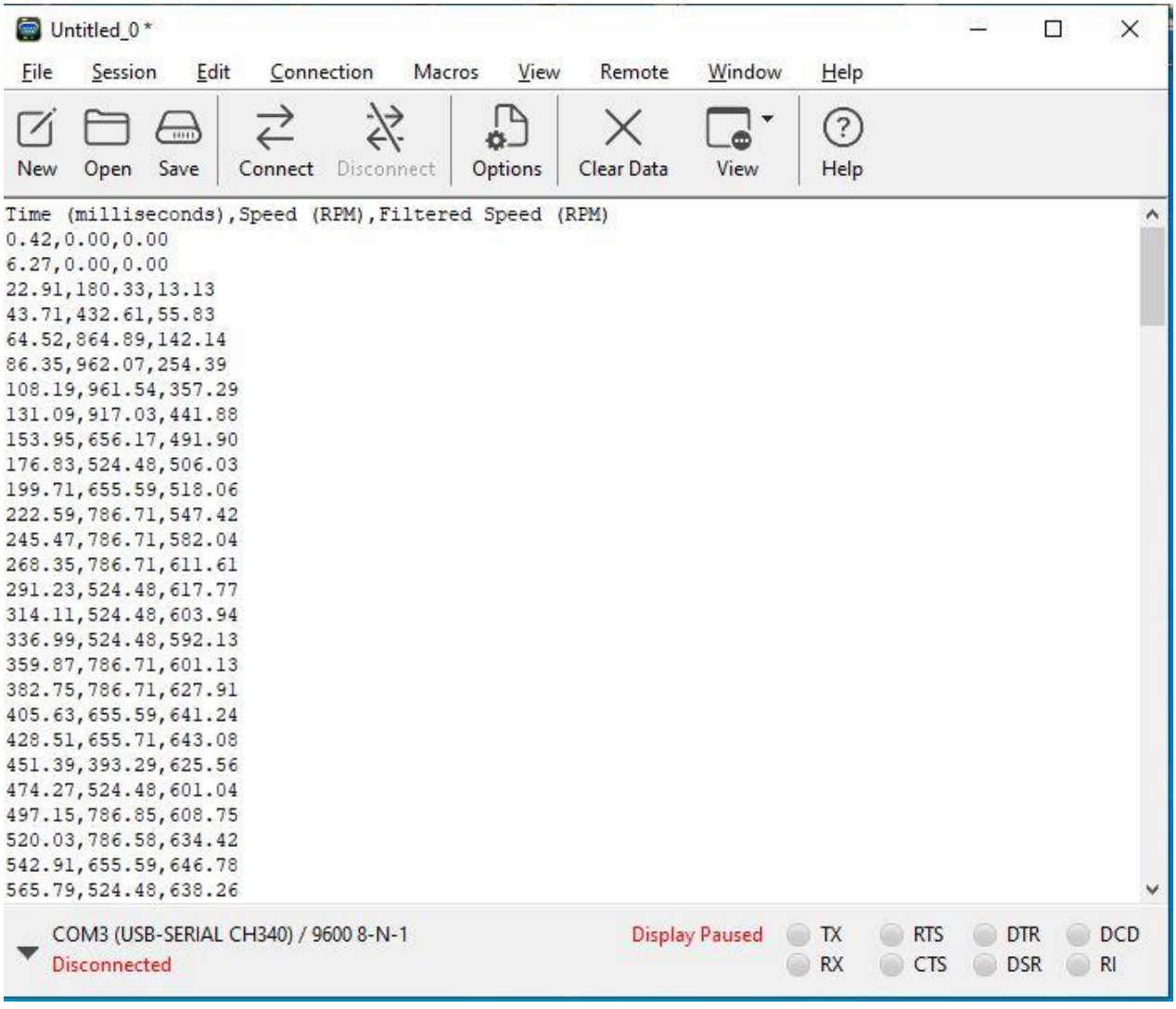

Figure 4.1: Reading data from a CoolTerm window

4.2 Open Loop Response

The open loop response is shown in Figure 4.2. The motor's speed increases sharply at the beginning, indicating a rapid response to the applied voltage. The rise time is short, showing that the motor reacts quickly. IT can be observed that there are high frequency bounces between levels due to the discrete measurements. To alleviate this effect, a low pass filter is required to smoothen the graph.

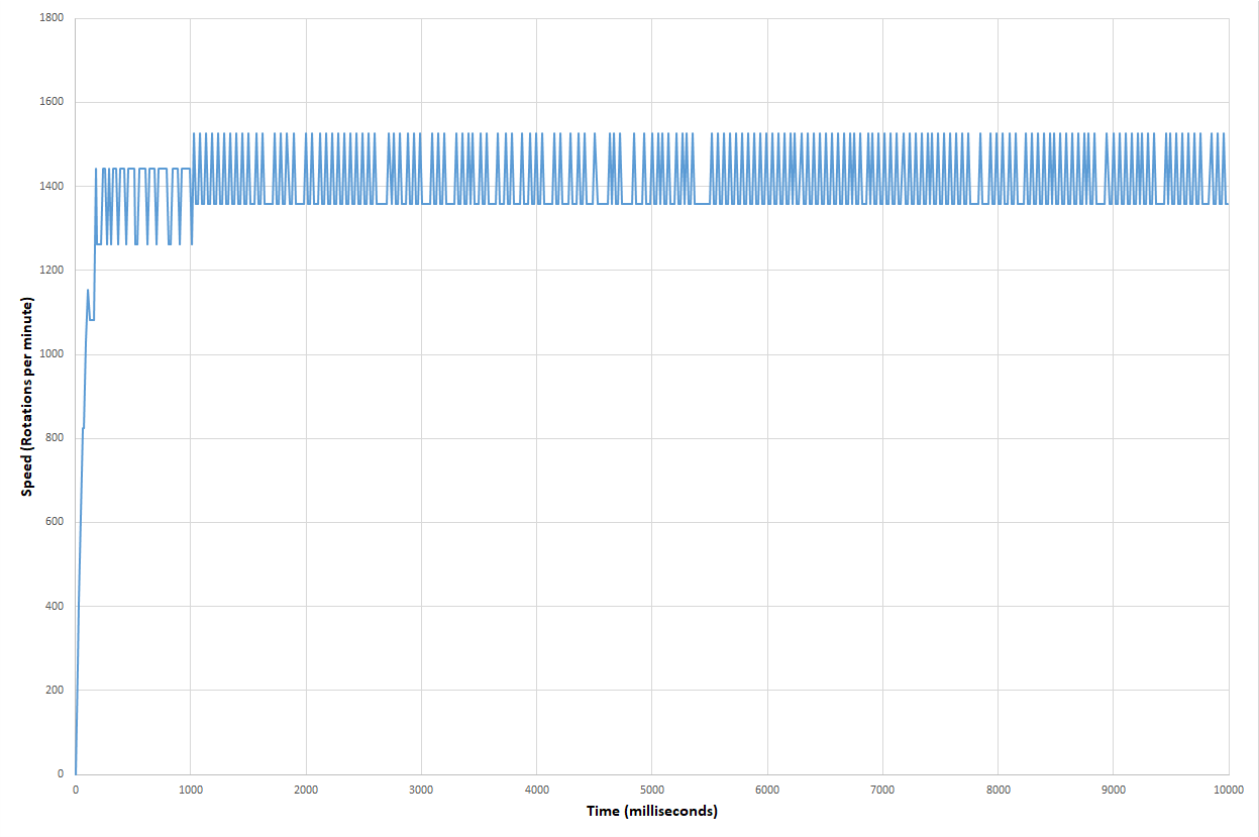

Figure 4.2: Open loop response of the motor

4.3 Low Pass Filter Response

After implementing the low pass filter, the response of the system becomes much smoother as shown in Figure 4.3. The unfiltered response exhibits significant oscillations and noise, particularly after the motor speed stabilises. The filtered response demonstrates a much smoother curve, with the oscillations from the raw speed data significantly reduced. The filtering process has effectively minimised the noise, providing a cleaner representation of the motor's speed. The filtered data better represents the actual motor performance since it eliminates high-frequency noise and provides a clearer picture of the motor's speed over time. Using filtered data would lead to more stable and reliable motor control.

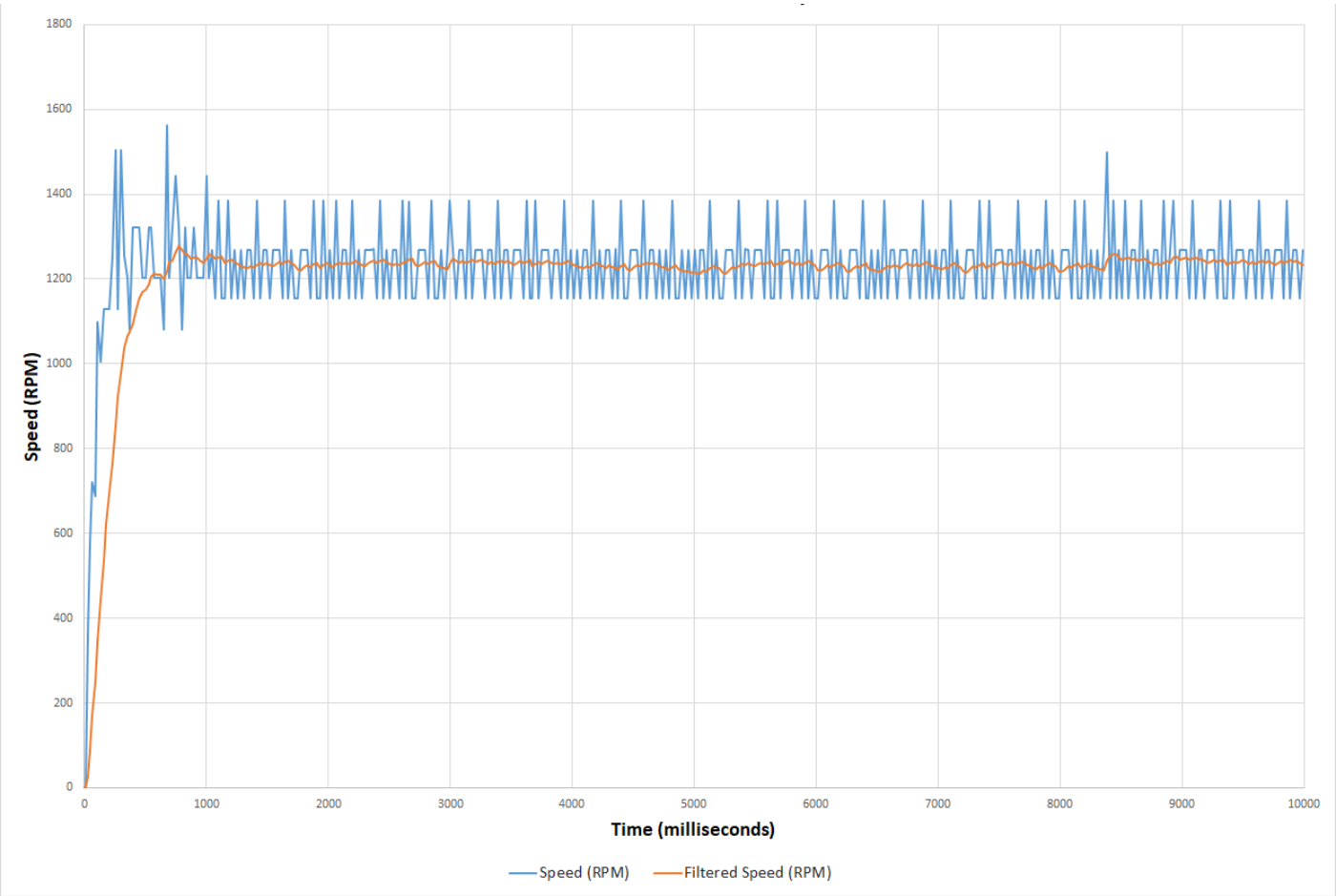

Figure 4.3: Filtered and unfiltered response of the motor

4.4 Proportional Control Response

The response with proportional control with the set point set to 600 and the proportional gain set to 0.72, is shown in Figure 4.4. It can be observed that the system experiences oscillations about a steady value. The average steady-state value is 528, which gives a steady–state error of 72 (12% error).

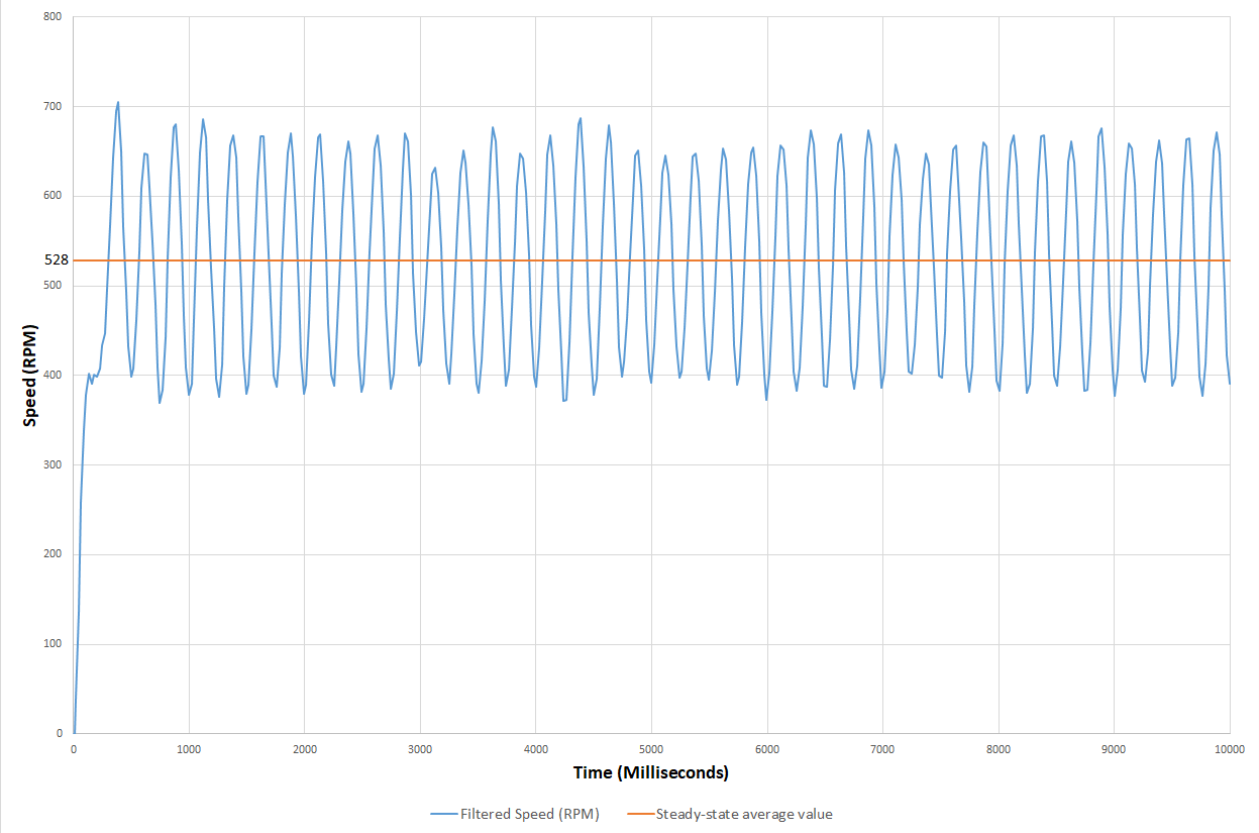

Figure 4.4: Response with P control

4.5 Proportional-Integral Control Response

The response to PI control with the set point set to 600, the proportional gain to 0.72, and the integral gain to 1.6, is shown in Figure 4.5. It can be observed that the oscillations are reduced in amplitude and frequency compared to the proportional control response.

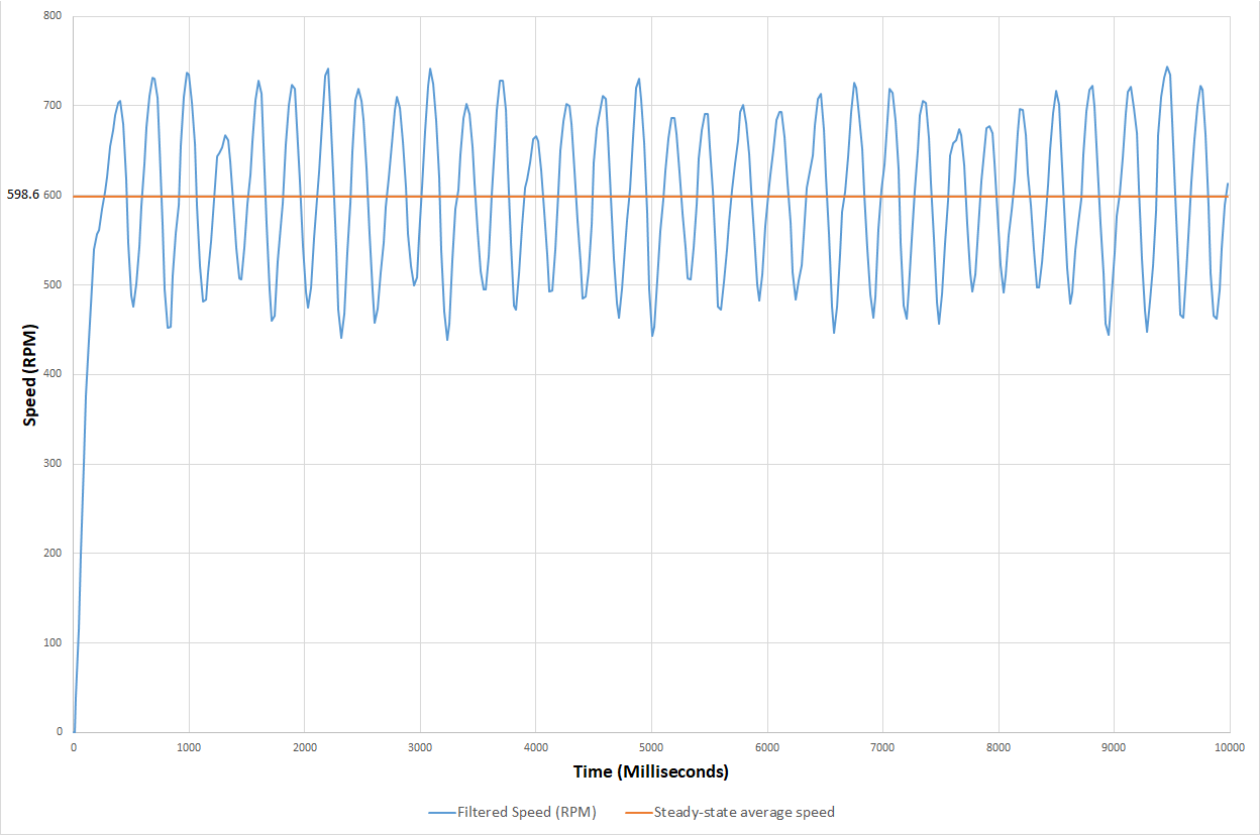

Figure 4.5: Response with PI control

4.6 Proportional-Integral-Derivative Control Response

The response to PID control with the set point set to 600, the proportional gain to 0.72, the integral gain to 1.6, and derivative gain to 0.05, is shown in Figure 4.6. The oscillations about the steady state value are much reduced and the overall trend can be easily seen. The rise time and overshoots are also reduced.

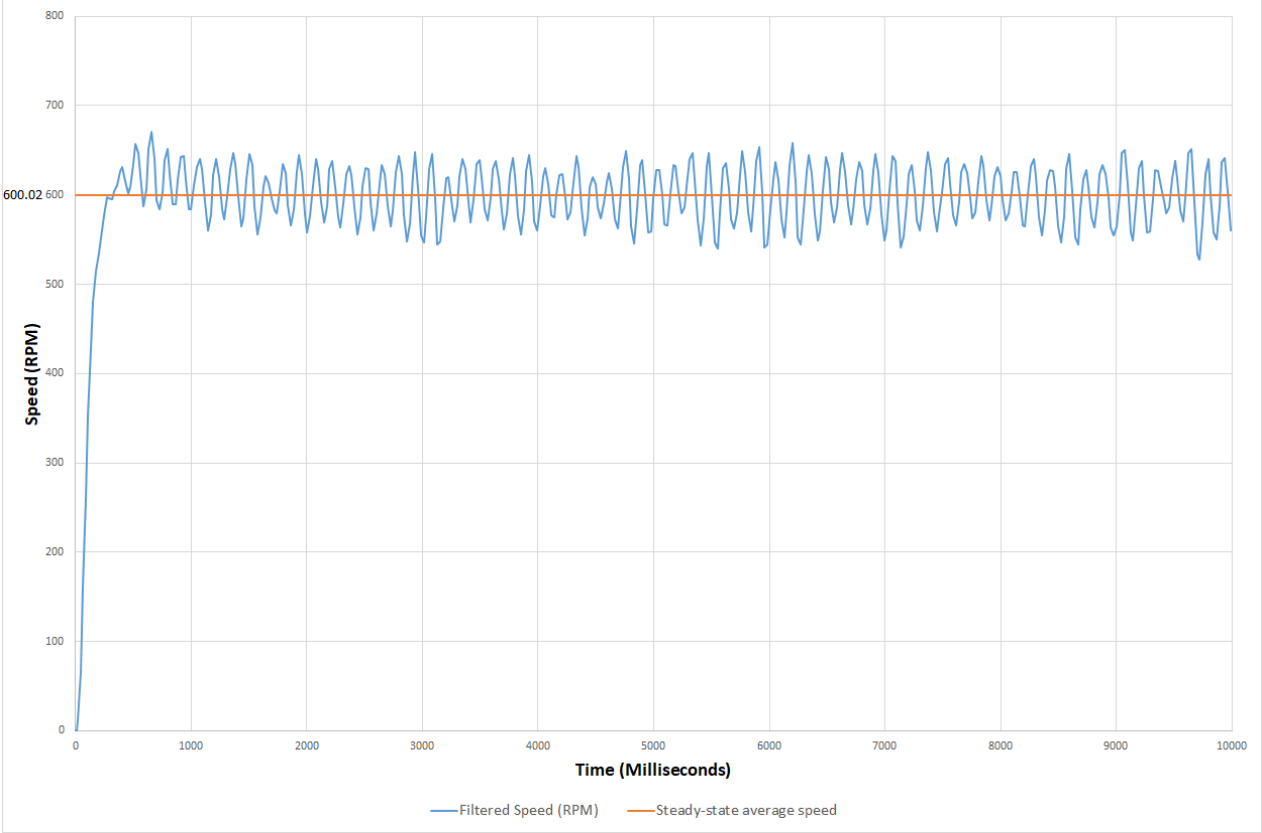

Figure 4.6: Response with PID control

4.7 Load Response

The response of the system when it is loaded must be considered. Figure 4.7 shows the system response to load. The speed and actuating signal U(t) are plotted with time as a load is added and later removed.

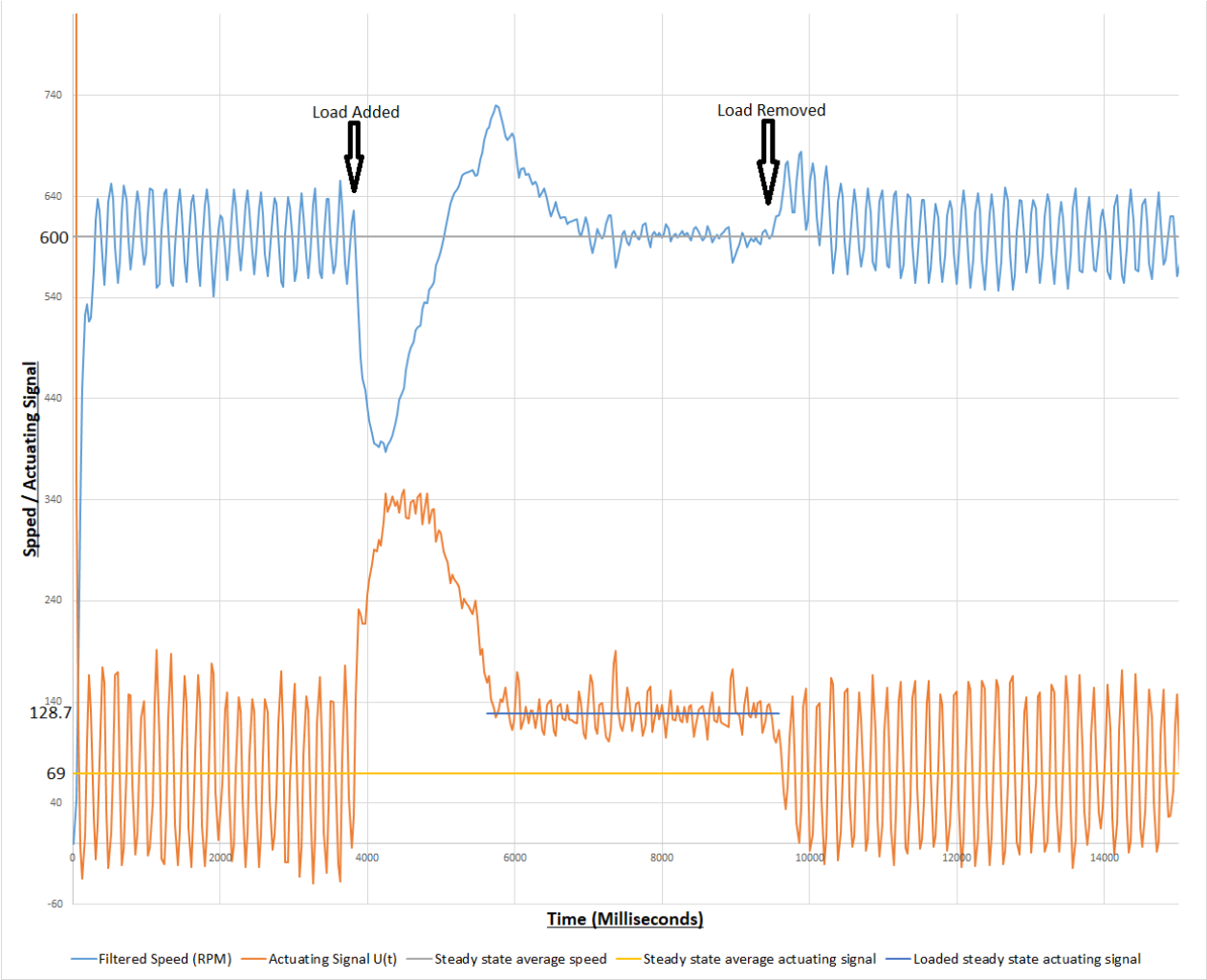

Figure 4.7: Load response with PID control

a. **Initial Response (0 - 500 ms)**: The system starts by attempting to reach the set point of 600 rpm. There is an initial oscillation in the speed as the system stabilises. The actuating signal of the controller begins at a very high value of 1,875,432 when the speed is at 0. This is typical behaviour of a PID controller as it is reacting aggressively to the initial large error (the difference between the desired speed of 600 rpm and the actual speed of 0 rpm). The

actual applied value of the actuating signal is capped in the code to a maximum of 140 to avoid instability.

b. **Steady-State Operation (500 - 3800 ms)**: Before the load is added, the system settles at 600 rpm, showing only minor oscillations. This suggests that the PID controller has brought the system to a fairly stable operation with minimal steady-state error. The actuating signal also stabilises at a value of 69 during this period, indicating that the control effort has also stabilised once the system is near the set point.
c. **Load Added (~3800 ms)**: A significant drop in speed is observed when the load is applied, indicating that the added load causes a large disturbance in the system. The speed initially drops to below **400 rpm** as the system reacts to the sudden change. In response to the increased error, the actuating signal spikes upward to counteract the speed drop, applying more power to the motor to bring the system back to the set point.
d. **Recovery After Load Addition (~4200 - 7000 ms)**: The system gradually recovers from the disturbance, with the speed overshooting the steady state of 600 rpm before recovery. The actuating signal decreases correspondingly as the speed overshoots and the PID controller adjusts to reduce the power supplied to the motor.
e. **New Steady-State Reached (`7000 - 9500 ms):** The system settles into a new steady-state balance where the speed is still 600 rpm but the steady actuating signal is increased from 69 to 128.7. The increase is what is required to keep the speed steady at 600 rpm while driving the load.
f. **Load Removed (~9500 ms)**: After the load is removed, the system experiences another disturbance. The speed increases briefly above the set point, but the system quickly returns to 600 rpm. The actuating signal shows a dip as the PID controller reduces the control effort to compensate for the sudden drop in load.
g. **Final Steady-State (10000 ms and beyond)**: After the load is removed, the system returns to initial steady-state operation.

4.7.1 Load Response Analysis

a. **Damping Effect of Load**: The addition of load influences the oscillatory behaviour of the system by reducing the amplitude and frequency of oscillations in the speed of the system. In control systems, damping is a critical factor that affects system stability and responsiveness. The loading increases the damping factor of the system. From this, it can be seen that the damping factor of the initial system can be improved by further tuning of the controller parameters.

b. **Load Handling**: The system can recover from load disturbances but experiences notable speed drops and overshoots when the load is added or removed. This indicates that the PID controller is effective but could be fine-tuned to minimise the impact of such disturbances.

c. **Response Speed**: The system took about 3 seconds to achieve a new steady state when the load was added. Depending on the application, this could be unsuitable and it indicates that the system parameters can be adjusted to speed up the response.

d. **System Stability**: Once the load is applied, the system becomes more stable, as the additional mass dampens any rapid fluctuations in speed. However, while the load increases stability, it may also cause a longer settling time, where the system takes longer to reach steady state after a disturbance. This trade-off between stability and response time is typical of systems with significant damping.

Overall, the system performs reasonably well, but improvements in tuning (e.g., fine-tuning PID parameters or adding more advanced filtering) could enhance its response to load changes and reduce oscillations.

4.5 Comparing the Performance Indices

Table 4.1 shows the performance comparison for the different control strategies using the following values for the proportional, integral, and derivative gain.

Table 4.1: Performance comparison of open loop, P, PI, and PID control for set point 600 rpm.

Performance Indices	Open Loop	P Control	PI Control	PID Control
Steady-state error	-	12%	0.23%	0%
Maximum overshoot	3.75%	17.5%	23.92%	11.86%
Peak time (Milliseconds)	772	387	407	657
Rise time (Milliseconds)	310	312	178	200
Kp = 0.72, Ki = 1.6, Kd = 0.05, Set point = 600 rpm				

The responses obtained are compared based on the performance indices. In control systems, performance indices are essential metrics used to evaluate and compare the responses of different systems or control strategies. These indices provide a quantitative means to assess how well a system performs, particularly when subjected to a given input such as a step, ramp, or impulse. By analysing the performance indices, we can determine if the system meets desired specifications, optimise system parameters, or compare the effectiveness of different controllers.

A smaller steady-state error signifies better accuracy and performance, while a larger error indicates that the system may not reach the desired output precisely. A smaller steady-state error is preferred, indicating that the system is accurately tracking the desired input. The maximum overshoot reflects how well the system can control or limit oscillations before reaching a steady state. High overshoot may indicate an underdamped system, while low or no overshoot suggests better control and stability. A shorter peak time indicates that the system reaches its maximum

output quickly, while a longer peak time suggests a slower response. It is determined by the system's damping ratio and natural frequency. The rise time of a control system is the time it takes for the system's output to rise from a specified low percentage to a specified high percentage of its final steady-state value when subjected to a step input. It is typically defined as the time it takes for the output to rise from 10% to 90% of its final value.

4.6 Summary of Analysis of Results

From the results derived, the following reaction was observed for the different control strategies:

A. **Steady-State Error**:
   a. **Open Loop**: No target value is provided as there is no feedback, but in open-loop control systems, steady-state error typically depends on system dynamics and external disturbances.
   b. **P Control**: There is a 12% steady-state error, indicating that proportional control alone cannot fully eliminate the steady-state error.
   c. **PI Control**: The PI control action reduces the steady-state error to 0.23%, as the integral action compensates for the error over time.
   d. **PID Control**: PID control action completely eliminates the steady-state error (0%), making it the most accurate in maintaining the desired set point.

B. **Maximum Overshoot**:
   a. **Open Loop**: The open loop has a minimal overshoot of 3.75%, which is due to the system's natural dynamics.
   b. **P Control**: The PI control action has a higher overshoot of 17.5%, indicating that proportional control leads to more aggressive responses.
   c. **PI Control**: The PID control action exhibits the highest overshoot (23.92%), which is a typical drawback of adding an integral action without sufficient damping.
   d. **PID Control**: PID control reduces overshoot to 11.86%, offering better control of the transient response while still minimising steady-state error. The PID control reduces the oscillations in the response.

C. **Peak Time**:

a. **Open Loop**: It has the longest peak time (772 milliseconds), showing the slow response of the system without feedback control.

b. **P Control**: P control significantly reduces the peak time to 387 milliseconds, resulting in a faster response.

c. **PI Control**: PI control slows down slightly compared to P control with a peak time of 407 milliseconds, due to the effect of the integral term.

d. **PID Control**: PID control has a peak time of 657 milliseconds, longer than P and PI controls but faster than the open-loop system.

D. **Rise Time**:

a. **Open Loop**: Open loop has a moderate rise time of 310 milliseconds.

b. **P Control**: PI control has a similar rise time (312 milliseconds), indicating that proportional control alone doesn't greatly affect rise time in this system.

c. **PI Control**: PI control achieves the fastest rise time of 178 milliseconds, making it highly responsive.

d. **PID Control**: PID control has a slightly longer rise time than PI control (200 milliseconds), but still faster than the open-loop and P controllers.

Open loop control lacks feedback, leading to large steady-state errors, high overshoots, and slow responses. Proportional control reduces the peak time and rise time but results in a higher steady-state error and overshoot. PI control provides excellent reduction in steady-state error and has the fastest rise time, but suffers from significant overshoot. PID control offers a balanced performance with no steady-state error, moderate overshoot, and reasonable peak and rise times, making it the most effective overall controller in this scenario.

4.7 Challenges Encountered

Some challenges were encountered during the course of this project. They are:

A. **Sensor Noise:**

During the course of the project, there were instances when the sensor pin was triggering the Arduino interrupt arbitrarily due to noise. This resulted in inaccurate readings and caused a lot of frustration. A 10kΩ pull-down resistor was used to resolve the issue. Pull-down resistors are resistors used in logic circuits to ensure a well-defined logical level at a pin under all conditions. Digital logic circuits have three logic states: high, low and floating (or high impedance). The floating state occurs when the pin is not pulled to a high or low logic level, but is left “floating" instead. It is neither in a high or low logic state, and the microcontroller might unpredictably interpret the input value as either a logical high or logical low. Pull-down resistors are used to solve the dilemma for the microcontroller by pulling the value to a logical low state (EE Power, n.d.).

A pull-down (pr pull-up) resistor is commonly used in digital circuits to stabilise the voltage at an input pin and prevent unintended changes due to noise or floating inputs. When a sensor’s output is connected to a microcontroller or digital circuit, sensor noise, particularly in the form of electrical fluctuations, can cause unstable or erroneous readings. The pull-down resistor helps mitigate this issue by ensuring that the input pin is firmly connected to a defined low (ground) voltage level when the sensor output is inactive or floating. The lower the resistance value the larger that current is, but the stronger the pull-down is. For most applications a 10K pull-down resistor is perfect (Arduino Forum, 2023).

B. **Parts procurement:**

Sourcing the necessary components for building the PID-controlled conveyor belt system was a significant challenge. Components such as optical slot sensors for speed measurement, motor driver, ball bearings, and motor coupler are unavailable in my location and they had to be ordered. Delivery times also slowed down the process.

C. **Software limitations:**

I encountered an issue when trying to copy the readings from the Arduino serial monitor. The readings could not be copied due to the scroll limits, limited buffer size, and copying

limits. To resolve this issue, CoolTerm, a freeware serial port terminal app and networking program, is used to capture the serial data directly into a text file.

D. **Hardware limitations:**

The system cannot accurately measure the highest speed of the open loop motor. This is due to the fact that the interrupts trigger at a faster rate than the sensitivity of the sensor. The processing speed of the Arduino microcontroller may also be a factor. This is why the maximum PWM output is capped below 255 (which is the actual maximum possible PWM output) in the code.

E. **Acoustic Noise:**

The physical conveyor belt when connected produces a lot of noise. This is partly due to the fact that the shaft is not tightly coupled and it is not perfectly aligned. To minimise this, the coupling must be tight and the alignment should be perfect.

F. **System instability:**

One of the core challenges in implementing PID control is achieving system stability. The constructed system experiences high frequency oscillations in the response even at steady state. The Low Pass Filter (LPF) implemented mitigates this problem to an extent. Also, the physical conveyor belt system has an unstable response. This is because the physical system is imperfectly constructed. The roller is not properly centred, the shaft is not perfectly aligned, and the shaft is not tightened to the rollers. All these cause the system to shake and jitter. It also causes a lot of noise and puts stress on the motor and the motor coupler. To model ideal conditions, I took the main readings for tuning the controller from the motor without connecting it to the physical system with the hopes of fine-tuning it afterwards. Figure 4.8 shows the jitter and instability in the physical system response.

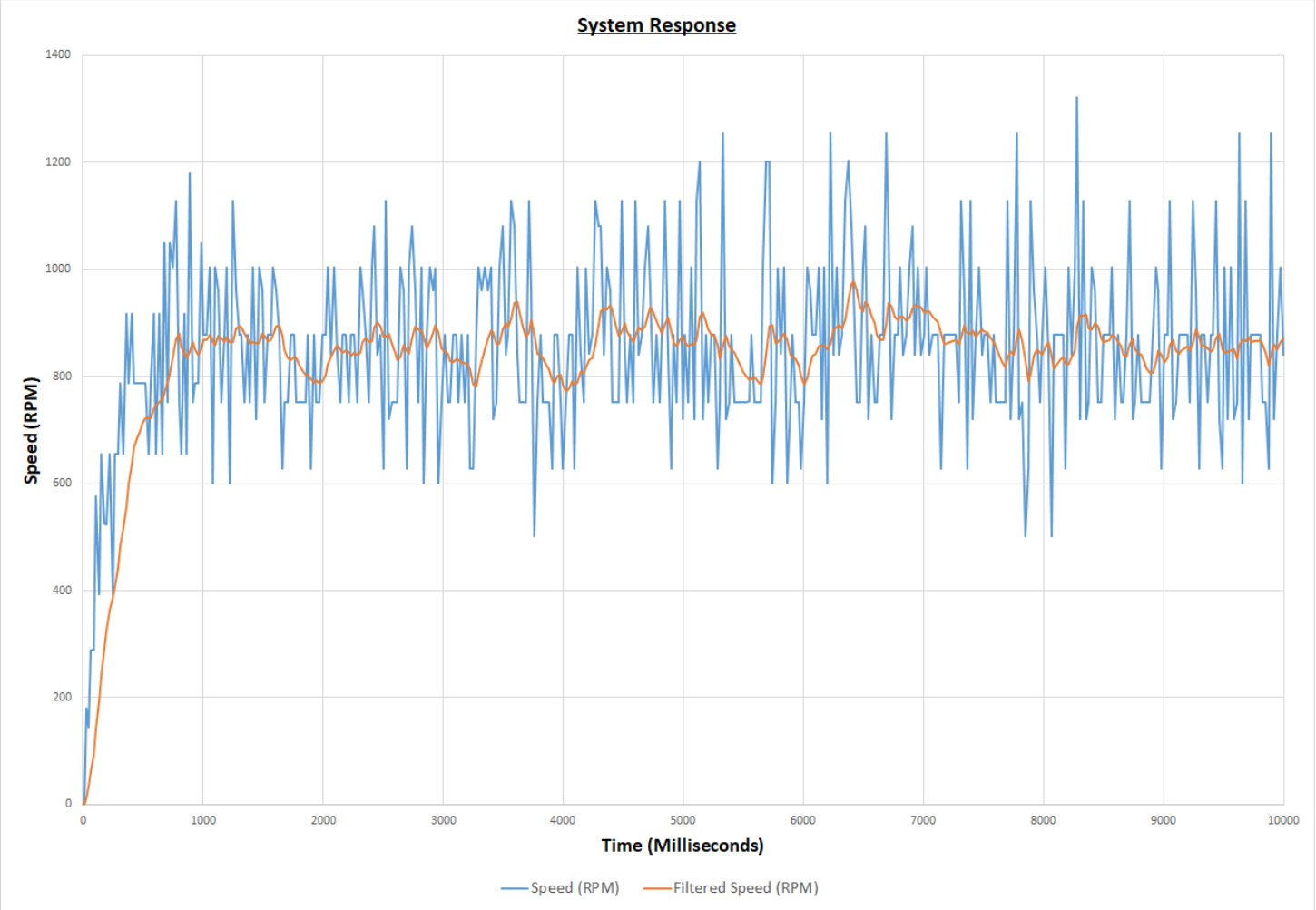

Figure 4.8: Physical system response

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

This project has successfully demonstrated the effectiveness of PID control in managing the speed and stability of a conveyor belt system. By integrating a PID controller with a DC motor and utilising the Arduino Uno as the control platform, the system was able to achieve precise regulation of conveyor speed with zero steady-state error, even in the presence of disturbances and load variations. Throughout the project, the key objectives of minimising the steady-state error, reducing overshoot, and ensuring fast response times were met through careful tuning of the PID gains. The system's performance was evaluated based on critical performance indices, such as rise time, peak time, and steady-state error, revealing that the PID controller provided significant improvements over other control methods like proportional and PI control. During the project, several challenges were encountered such as software limitations, hardware limitations, and system instability during tuning. Despite these challenges, the project demonstrated the potential for low-cost, efficient control systems using widely accessible hardware like the Arduino Uno. The implementation also showcased the adaptability of PID control in handling variations in load and speed, making it an ideal choice for conveyor belt systems in industrial applications.

In conclusion, this project highlights the importance of PID control in automated systems, offering a practical and relatively cheap solution for maintaining precise control of motor-driven processes. The experience gained through this project provides a solid foundation for further exploration and optimization of control systems, with potential applications beyond conveyor systems to processes in different areas such as temperature control, home and industrial automation, and robotics.

5.2 Recommendations and Future Directions

Based on the results and challenges encountered during this project, several recommendations can be made for future improvements and developments:

a. **Advanced Noise Filtering:** This project incorporates a first order, infinite impulse response (IIR) low pass filter to reduce the high frequency oscillations and jitter in the speed response of the conveyor. The filter smoothes the response. However it also introduces a small phase lag to the system. Increasing the order of the filter will increase the smoothing effect but it will also increase the phase lag.

b. **Microcontroller Improvement:** There are two possible directions to take for improving the microcontroller: increasing processing power and reducing cost. While the Arduino Uno is an accessible and flexible platform, it has limitations in terms of processing power, clock speed, and memory. For more complex systems or higher precision control, upgrading to a more powerful microcontroller, such as the Arduino Mega or Raspberry Pi, could allow for more sophisticated control algorithms, faster data processing, and better handling of multiple sensors. On the other hand, to deploy such systems at scale, the Arduino Uno could prove too expensive and a more cost-effective solution may be necessary. Smaller, lower-cost alternatives include the ATtiny series or ESP8266 for simpler, dedicated control tasks. When considering these options, there is often a tradeoff between cost and functionality.

By following these recommendations, the PID-controlled conveyor belt system can be made more reliable, scalable, and efficient for a wider range of applications. These improvements will enhance its value both in academic research and practical industrial environments, contributing to more effective automation solutions.

REFERENCES

Arduino, (2024). *Arduino Uno Rev3*. Arduino. Online. Available from: https://docs.arduino.cc/hardware/uno-rev3/, [Accessed 8/10/2024].

Arduino Forum. (2023). *Pull-up / pull-down resistor*. Arduino Forum. Online. Available from: https://forum.arduino.cc/t/pull-up-pull-down-resistor/1264205, [Accessed 1/10/2024].

Bansal, U. K., and Narvey, R. (2013). Speed control of DC motor using fuzzy PID controller. *Advances in Electronic and Electric Engineering*, 3(9), 1209–1220.

Bhatt, A. (2014). *Designing an active low-pass filter*. Engineers Garage. Online. Available from: https://www.engineersgarage.com/designing-of-low-pass-filter/, [Accessed 2/10/2024].

Borase, R.P., Maghade, D.K., Sondkar, S.Y., and Pawar, S. N. (2021). A review of PID control, tuning methods and applications. *International Journal of Dynamics and Control, 9*, 818–827.

Components101. (n.d.). *L293N motor driver module*. Components101. Online. Available from: https://components101.com/modules/l293n-motor-driver-module, [Accessed 8/10/2024].

CoolTerm. (n.d.). *CoolTerm*. Online. Available from: https://coolterm.en.lo4d.com/windows, [Accessed 27/9/2024].

Davila, D. J. M., Oyedele, L., Ajayi, A., Akanbi, L., Akinade, O., Bilal, M., and Owolabi, H. (2019). Robotics and automated systems in construction: Understanding industry-specific challenges for adoption. *Journal of Building Engineering, 26*, 100868.

EE Power. (n.d.). *Pull-up resistor vs. pull-down resistor*. EE Power. Online. Available from: https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor, [Accessed 1/10/2024].

Ellis, G. (2012). *Control System Design Guide: Using Your Computer to Understand and Diagnose Feedback Controllers* (4th ed.). Elsevier.

Hammoodi S. J., Flayyih K. S., and Hamad A. R. (2020). Design and implementation speed control system of DC Motor based on PID control and Matlab Simulink. *International Journal of Power Electronics and Drive System, 11*(1), 127-134.

Hassan, S. (2008). *Automatic control systems*. Katson Books.

IQS Directory (2024). *Belt Conveyors.* Online. Available from: https://www.iqsdirectory.com/articles/conveyors/belt-conveyors.html, [Accessed 1/10/2024].

Katsioula, A. G., Karnavas, Y. L., and Boutalis, Y. S. (2018). An enhanced simulation model for DC motor belt drive conveyor system control. In *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST).* IEEE

Kondaveeti, H. K., Kumaravelu, N. K., Vanambathina, S. D., Mathe, S. E., and Vappangi, S. (2021). A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations. *Computer Science Review, 40*, 100364.

Kumar, D., Hussain, M., Tyagi, S. V., Gupta, R., and Salim. (2020). LabVIEW Based Speed Control of DC Motor Using PID Controller. *International Journal of Electrical and Electronics Research, 3*(2), 293-298.

Maghfiroh, H., Nizam, M., and Praptodiyono, S. (2020). PID optimal control to reduce energy consumption in DC-drive system. *International Journal of Power Electronics and Drive Systems*, 11(4), 2165–2172.

MathWorks. (n.d.). *Lowpass filter design*. Online. Available from: https://www.mathworks.com/help/dsp/ug/lowpass-filter-design.html#LowpassFilterDesignExample-7, [Accessed 1/10/2024].

Nagrath, I. J., and Gopal, M. (2007). *Control systems engineering* (5th ed.). New Age International (P) Ltd., Publishers.

National Instruments. (n.d.). *FIR and IIR filter design*. Online. Available from: https://www.ni.com/docs/en-

US/bundle/diadem/page/genmaths/genmaths/calc_filterfir_iir.htm?srsltid=AfmBOor6j1JxIJSRG7cw3FH1lAcACUF7Gz2gBxq5WQ8XtubObhe007ZZ, [Accessed 1/10/2024].

Ogata, K. (2010). PID controllers and modified PID controllers. In K. Ogata, Modern Control Engineering. (5th ed., pp. 567-647). Prentice Hall.

Sandeep, V. M. (2021). *When and why to use P, PI, PD, and PID controller*. Medium. Online. Available from: https://medium.com/@svm161265/when-and-why-to-use-p-pi-pd-and-pid-controller-73729a708bb5, [Accessed 1/10/2024].

Somefun, O. A., Akingbade, K., and Dahunsi, F. (2021). The dilemma of PID tuning. *Annual Reviews in Control*, 52, 65–74.

Taghavi, N., Luecke, G., and Jeffery, N. (2020). A technical review on development of an advanced electromechanical system. *Computers*, *9*(1), 19.

Todkar, S., Ramgir, M., and Tathwade, J. R. (2018). Design of belt conveyor system. *International Journal of Science, Engineering and Technology Research*, *7*(7), 458-462.

Wang, L. (2020). PID Control System Design and Automatic Tuning using MATLAB/Simulink. *John Wiley & Sons Ltd.*

APPENDIX A

COMPLETE ARDUINO CODE FOR THE PROJECT

```
// Title: Implementing PID control
#include <util/atomic.h>
#include <LCD_I2C.h>

LCD_I2C lcd(0x27, 16, 2);

int sensorPin = 2;
int motorPin1 = 6;
int motorPin2 = 7;
int motorEnPin = 9;

unsigned long startTime = 0;
unsigned long lastLcdUpdateTimeMillis = 0;

// Variables for calculating speed of revolution
unsigned long prevTime = 0;
unsigned int counter = 0;

// Variables for the Low pass filter filter
float filtered_speed = 0;
float prev_filtered_speed = 0;

// Set target speed (RPM)
float target_speed = 600;

// Set the PID gains
float Kp = 0.72;
float Ki = 1.6;
float Kd = 0.01;

// Variables for calculating the errors
float error = 0;
float prevError = 0;
float integral_error = 0;
float derivative_error = 0;

// Increment the counter on every loop pass to calculate speed
void incrementCount() {
  counter++;
}
```

```
// actuate the motor with a pwm value and direction
void actuateMotor(int new_pwm_value, int new_direction) {
  if (new_direction > 0) {
    analogWrite(motorEnPin, new_pwm_value);
    digitalWrite(motorPin1, HIGH);
    digitalWrite(motorPin2, LOW);
  } else {
    analogWrite(motorEnPin, new_pwm_value);
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, HIGH);
  }
}

void stopMotor() {
  analogWrite(motorEnPin, 0);
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
}

void setup() {
  Serial.begin(9600);
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorEnPin, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(sensorPin),      incrementCount,
RISING);

  // Print CSV headers
  Serial.println("Time    (milliseconds),Speed    (RPM),Filtered    Speed
(RPM),Actuating Signal U(t)");

  prevTime = micros();
  startTime = micros();

  lcd.begin();
  lcd.backlight();
}

void loop() {
  // Condition to run the loop for only 10 seconds
  if (micros() - startTime <= 10 * 1.0e6) {
    // Read the counter in an atomic block to avoid potential misreads
    int counterCopy = 0;
```

```
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
      counterCopy = counter;
    }

    // Compute the velocity using method 1: calculating from how much
time
    // has elapsed between each iteration of the loop
    unsigned long currTime = micros();
    float elapsedTime = ((float)(currTime - prevTime)) / 1.0e6;
    float speed = (((float)counter / 20) / elapsedTime);  // rotations
per second
    speed *= 60;                                          // rotations
per minute

    // Applying a Low Pass Filter to the velocity
    filtered_speed = 0.854 * filtered_speed + 0.0728 * speed + 0.0728 *
prev_filtered_speed;
    prev_filtered_speed = speed;

    // Calculate the errors
    error = target_speed - filtered_speed;
    integral_error = integral_error + error * elapsedTime;
    derivative_error = (error - prevError) / elapsedTime;

    // Compute the actuating signal
    float actuating_signal = (Kp * error) + (Ki * integral_error) + (Kd
* derivative_error);

    // Set the motor speed and direction
    int direction = 1;
    if (actuating_signal < 0) {
      direction = -1;
    }
    int pwm_value = (int)fabs(actuating_signal);  // PWM ranges from 0
(0% duty cycle) to 255 (100% duty cycle)
    if (pwm_value > 170) { // The interrupt cannot measure maximum speed
      pwm_value = 170;
    }

    // Actuate the motor
    actuateMotor(pwm_value, direction);

    // Resetting the values for the next loop
    counter = 0;
    prevTime = currTime;
```

```
    prevError = error;

    unsigned long currTimeMillis = currTime / 1.0e3;

    // update the lcd every 200ms
    if (currTimeMillis - lastLcdUpdateTimeMillis >= 200) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Speed: ");
      lcd.print(filtered_speed);
      lcd.setCursor(0, 1);
      lcd.print("U(t): ");
      lcd.print(actuating_signal);
      lastLcdUpdateTimeMillis = currTimeMillis;
    }

    // Log the values in a CSV format
    Serial.print(currTimeMillis);
    Serial.print(",");
    Serial.print(speed);
    Serial.print(",");
    Serial.print(filtered_speed);
    Serial.print(",");
    Serial.println(actuating_signal);

    // Add a delay of 1ms to maintain a continuous sampling freq
    delay(1);
  } else {
    stopMotor();
  }
}
```