YRKESHÖGSKOLAN
NOVIA

Control Process: Development of an Arduino-Based Simulation and Interface System for NOVIA University of Applied Sciences

Albert Llobera Gorgues

Degree Thesis for Bachelor of Engineering

Degree Programme in Energy Technology

Vaasa 2024

DEGREE THESIS

Author: Albert Llobera Gorgues
Degree Programme and place of study: Electrical Engineering and Automation, Lleida
Specialisation: Energy Technology
Supervisor(s): Joachim Böling and Philip Hollins
Title: Control Process: Development of an Arduino-Based Simulation and Interface System for NOVIA University of Applied Sciences

---

Date: 17.5.2024 Number of pages: 49 Appendices: 2

Abstract

This thesis outlines the creation and implementation of an Arduino-based simulation tool designed to enhance the practical training of control systems for engineering students at NOVIA University. Given the complex nature of theoretical control systems, the project sought to provide a tangible, interactive platform to facilitate a deeper understanding of these systems through real-time simulation and manipulation.

The project encompassed both hardware integration and software development phases, utilizing Arduino technology alongside various sensors and actuators to construct a flexible and interactive educational tool. The tool allows students and instructors to dynamically alter parameters and observe the immediate effects on the system, thereby linking theoretical knowledge with practical application.

The implementation phase highlighted challenges such as limitations in the Arduino's output voltage range (0-10V or 0-5V limited to 0-2.25V) and programming barriers that prevented the creation of a password-protected interface. Despite these hurdles, the tool successfully achieved its educational objectives within the constraints of a budget, totalling around €85, showcasing the potential for cost-effective educational aids in engineering.

Ultimately, the simulation tool has proven to be a valuable addition to the control systems curriculum, providing an accessible platform for students to engage with complex concepts in a hands-on manner. This project not only bridges the gap between theory and practice but also sets a foundation for future enhancements that could further broaden the tool's capabilities.

Language: English
Key Words: Arduino, Control Systems, Simulation Tool, Educational Technology, Hardware Integration

Table of Contents

List of Figures

1 Introduction

Control systems are integral to modern engineering applications, encompassing a variety of sectors from manufacturing to aerospace. Despite their ubiquity, the complex nature of these systems poses a significant challenge in educational settings, particularly in effectively bridging the gap between theoretical knowledge and practical application. This thesis addresses this challenge by developing an Arduino-based simulation tool tailored for control systems education at NOVIA University.

The need for practical educational tools in control systems is evident. Students often struggle to grasp theoretical concepts without hands-on experience, limiting their ability to apply knowledge in real-world scenarios. This project seeks to mitigate this issue by providing a dynamic, interactive simulation environment that allows students to explore various control system dynamics and their applications.

The objective of this thesis is twofold: firstly, to design and implement a configurable, cost-effective simulation tool using Arduino technology, and secondly, to enhance the learning experience by allowing teachers to interact directly with the parameters so students study the behaviors of the system. This approach not only aids in consolidating theoretical concepts but also fosters analytical and problem-solving skills crucial for future engineers.

This introduction sets the stage for the detailed exposition of the project's aims, background theory, design methodology, implementation, and the results achieved. It will be followed by a discussion on the implications of the findings and suggestions for future enhancements.

By providing a comprehensive and accessible educational tool, this thesis contributes to the advancement of educational methods in control systems, aligning with NOVIA University’s commitment to delivering cutting-edge engineering education.

1.1 Aims and Objectives

This thesis has been commissioned by Novia University of Applied Sciences (www.novia.fi). The aim is to bridge the educational gap in the field of control systems at NOVIA University.

The primary goal is to develop an Arduino-based simulation tool that provides students with practical, hands-on experience with control systems, preparing them for real-world engineering challenges.

To achieve this goal, the thesis is structured around several specific objectives:

- Develop diverse and configurable simulation models that allow students to explore various control systems using Arduino hardware.
- Create an intuitive and user-friendly interface that simplifies the selection and modification of control models for both students and educators.
- Establish a robust connection framework for seamless integration with Programmable Logic Controllers (PLCs).
- Construct a dynamic environment where users can adjust parameters such as time constants and delays to enhance learning through hands-on experimentation.

1.2 Document structure

This thesis is structured to systematically explore the development of an Arduino-based simulation tool, guiding the reader from the project's inception through to its evaluation. The initial chapters introduce the project's aims and provide the necessary theoretical background on control systems. The subsequent chapters detail the design and development of the simulation tool, covering both hardware and software aspects. Implementation and testing are discussed next, highlighting the setup procedures and challenges faced. The results are then presented, analyzing the tool's effectiveness in enhancing control systems education and discussing its limitations. The thesis concludes with a summary of findings and contributions, along with recommendations for future work. Supplementary materials, detailed documentation, and references are provided in the appendices to support the research presented.

2 Study background

This section of the thesis is devoted to establishing a solid theoretical foundation necessary for understanding the principles of control systems, which are critical to the development of an effective Arduino-based simulation tool. It begins with an exploration of fundamental control system theories, tracing their evolution over time to highlight how these systems have become integral to modern engineering practices. The section will cover a range of topics from basic concepts, such as the classification of control systems into open-loop and closed-loop systems, to more complex topics such as the mathematical modelling and analysis of these systems.

Additionally, historical milestones in the development of control technologies will be reviewed to appreciate their impact on current practices. This background will not only enrich the reader's understanding but will also bridge theoretical concepts with practical applications, emphasizing their relevance in today's technological landscape. The aim is to equip the reader with a comprehensive understanding that supports the practical sections of this thesis, where these principles are applied to build and test a simulated control environment using Arduino technology.

2.1 Introduction to Control Systems

A control system is a network of connected devices designed to manage, command, direct, or regulate other machines or entire systems (Electrical4U, 2020). By leveraging control loops, these systems ensure specific variables, like temperature or pressure, stay within desired ranges, ultimately achieving a predetermined goal. This technology is crucial in various industries, from manufacturing and robotics to transportation and energy management.

2.2 Historical Development of Control Systems

The concept of control systems, with their intricate networks of devices regulating machines and processes, is not a new invention. As the fascinating historical account reveals, the seeds of this technology were planted millennia ago, even before the Industrial Revolution. The earliest evidence comes from ancient civilizations, where sophisticated

water clocks were designed around the 3rd century BC. These timekeeping devices, particularly those by the Greeks and Arabs, employed feedback control mechanisms to regulate water flow, showcasing the early understanding of control principles. (History | IEEE Control Systems Society, 2020).

While these examples were impressive, a breakthrough arrived in 1788 with James Watt's flyball governor (Electrical4U, 2023). This invention, considered the first truly "automatic" system, automatically regulated the speed of steam engines, marking a turning point and paving the way for advancements in the Industrial Revolution illustrated in Figure 1.

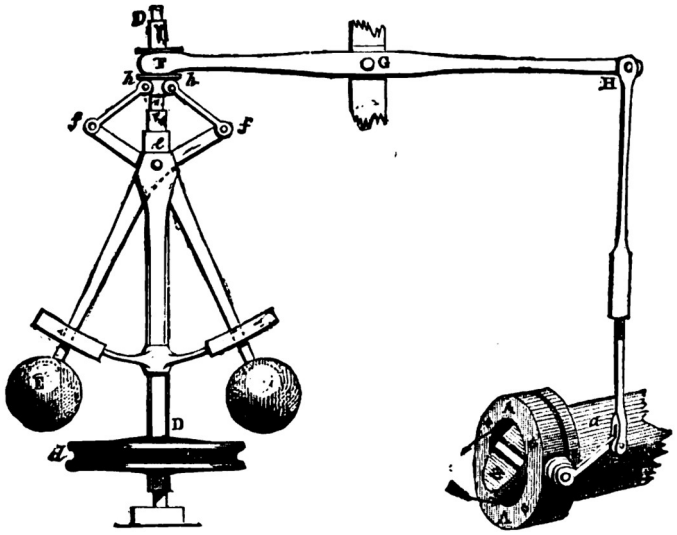


Figure 1: Flyball governor Source: https://upload.wikimedia.org

However, the impact of the flyball governor extended beyond its practical application. In 1868, James Clerk Maxwell, a renowned physicist, analyzed its behaviour using differential equations, laying the foundation for the development of mathematical control theory. This theoretical framework became crucial for understanding and designing control systems. (History | IEEE Control Systems Society, 2020).

The 19th century witnessed further advancements in mathematical modelling, essential for control systems, thanks to the work of Leonhard Euler, Pierre-Simon Laplace, and Joseph Fourier (Electrical4U, 2023).

The 20th century, often called the "golden age" of control engineering, saw significant developments in control methods. Bell Labs became a hub for innovation, with Hendrik Wade Bode and Harry Nyquist spearheading the development of "classical control" methods. Meanwhile, Nicholas Minorsky, a Russian-American mathematician, made crucial

contributions by developing automatic controllers for ships and introducing the concepts of integral and derivative control in the 1920s. Alongside these advancements, Harry Nyquist and Walter Evans established the concept of stability, while Oliver Heaviside pioneered the use of transforms in control systems. (Electrical4U, 2023).

However, the limitations of classical methods became apparent, leading to the development of "modern control" methods after the 1950s by Rudolf Kalman. This marked a further evolution in the field. The introduction of Programmable Logic Controllers (PLCs) in 1975 further revolutionized the field, making control systems more accessible and adaptable. (Electrical4U, 2023).

Today, automatic control systems are ubiquitous, playing a vital role in various fields, from space exploration and communication satellites to safer aircraft, cleaner automobiles, and efficient chemical processes. Their evolution represents a continuous journey of innovation, driven by both ancient ingenuity and modern scientific advancements.

2.3 Classification of Control Systems

Open-loop: In an open-loop control system, the control action initiated by the controller remains unaffected by the system's output or process variable. Put differently, alterations in the input variable do not translate into changes in the output variable. This configuration is often termed a non-feedback control loop system since the comparison between input and output responses is absent as shown in Figure 2. (The Engineering Concepts, 2021).

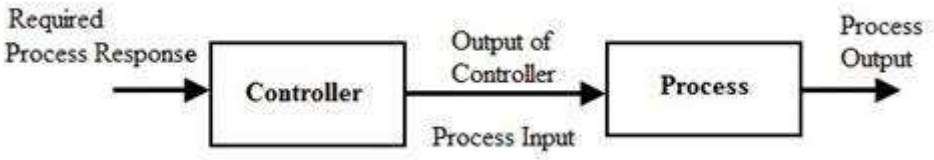


Figure 2: Open-loop block diagram Source: https://www.elprocus.com

Illustratively, in a washing machine, open-loop control is employed to determine the duration of each washing cycle. Users select a washing program, and the machine operates through predetermined sequences without adapting to the actual state of the clothes or water conditions.

Closed-loop: In a closed-loop control system, the input control action originating from the controller is contingent upon the system output or process variable. Any variation in the controller's input triggers a corresponding change in the output or process variable.

This system incorporates a feedback loop, ensuring that the controller adjusts its control action to maintain the process variable or output at the setpoint value illustrated in Figure 3. Consequently, a closed-loop control system is commonly referred to as a feedback controller. (The Engineering Concepts, 2021).

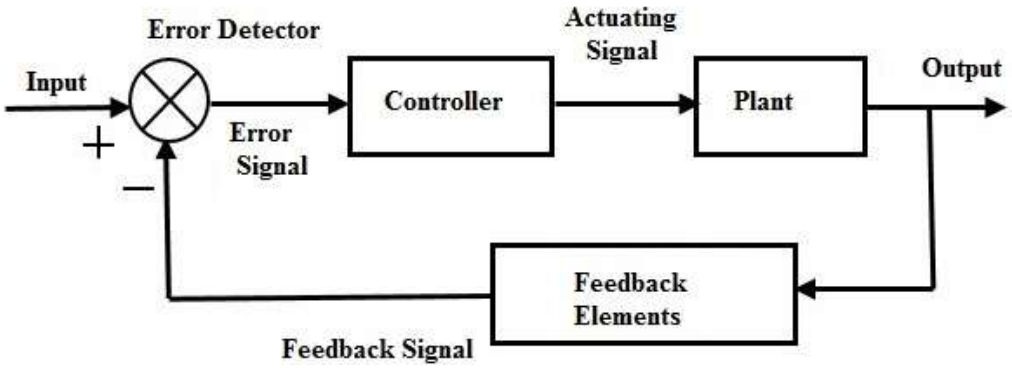


Figure 3: Closed-loop block diagram Source: https://www.elprocus.com

For instance, in a heating system's thermostat, a closed-loop control mechanism is employed to sustain a constant temperature. Sensors measure the current temperature, prompting the heating system to adjust its output to uphold the desired temperature level.

Linear: Linear control systems demonstrate a linear correspondence between input and output variables. This adherence to the principle of superposition ensures that the system's response to a combination of multiple inputs equals the sum of their individual responses. Due to their mathematical tractability, linear control systems are advantageous for analysis and design purposes, finding widespread application in various electronic devices and systems. (GfG, 2023a).

For example, in speed control for electric motors, linear control techniques are employed to regulate motor velocity. By modulating the input signal, whether it's voltage or current, the system can linearly adjust the motor speed.

Nonlinear: Nonlinear control systems, in contrast, feature nonlinear connections between input and output variables. The behaviour of these systems is notably intricate, often governed by nonlinear equations dictating their dynamics. They are encountered in

applications where linear approximations fall short, such as highly dynamic systems, chaotic systems, and those characterized by substantial nonlinearity. (GfG, 2023a).

For instance, in aerospace manoeuvring, nonlinear control systems navigate aircraft through dynamic and complex manoeuvres by adapting to nonlinear aerodynamic forces. This adaptation ensures stability across various flight scenarios, where linear models may not adequately capture the system's behavior.

Continuous: Continuous-time control systems serve as the backbone of dynamic systems, facilitating control over their long-term behaviour. By continuously adjusting input signals, these systems regulate and modify a process's output, ensuring sustained adherence to performance standards as can be seen in Figure 4. Treating all inputs, including signals and variables, as smoothly changing over time, continuous-time control systems effect changes constantly and seamlessly. (GfG, 2024).

Discrete: Discrete-time control systems play a vital role in designing and testing systems that operate at specific, distinct moments in time. Unlike continuous-time control systems, which handle gradual changes over time, discrete-time control systems manage signals measured at predetermined intervals as shown in Figure 4. Many real-world systems, particularly those involving digital electronics, communication networks, and computer-based controls, inherently operate in discrete time, highlighting the importance of this temporal handling approach. (GfG, 2024).

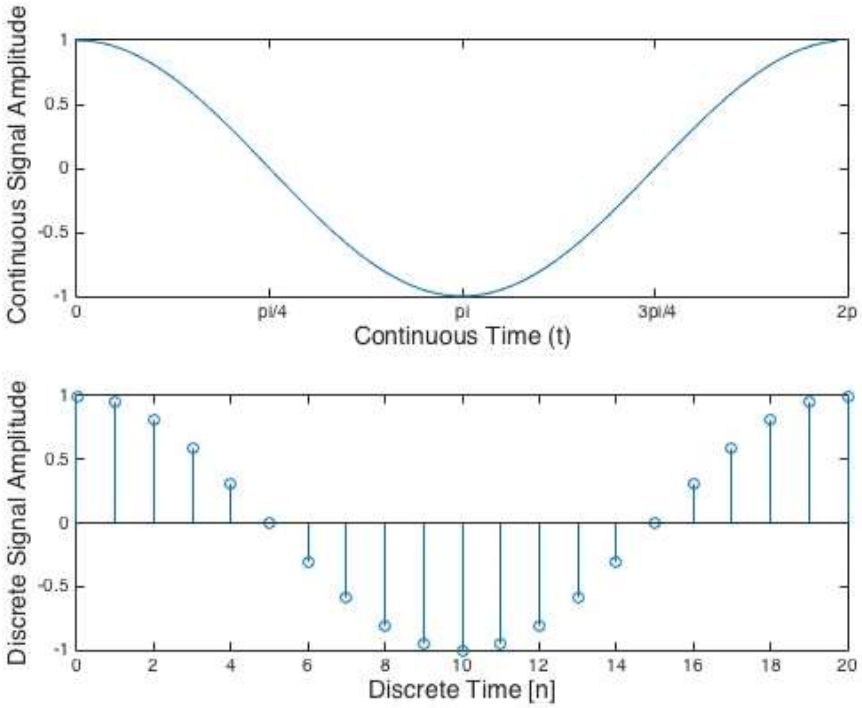


Figure 4: Continuous vs Discrete time controls Source: **https://www.theengineeringprojects.com**

Comprising various components such as controller design, stability evaluation, and behaviour representation using difference equations, discrete-time control systems find application in fields like robotics, industrial automation, digital signal processing, and telecommunications. Electronic engineers leverage these systems to address time-related challenges and devise reliable solutions for electronic systems and devices. Ensuring the accuracy and consistency of discrete-time control systems is paramount for the effective functioning of technology across diverse applications.

2.4 Modelling and Analysis of Control Systems

Modeling and analyzing control systems are crucial for predicting and understanding system behavior before real-world implementation. This section focuses on different methods used to model systems and introduces analysis techniques that reveal essential system parameters such as lag, time constant and gain.

2.4.1 Mathematical Equations

Control systems can be described through mathematical equations, collectively referred to as a mathematical model. These models are integral for both analyzing and designing

control systems. Analyzing a control system involves determining the output based on known inputs and the mathematical model. Conversely, designing a control system entails developing the mathematical model when the inputs and desired outputs are known.

The most commonly utilized mathematical models in control systems include:

Differential Equation Model: This model uses differential equations to represent the dynamic relationships within the system, typically describing how inputs influence outputs over time. (Control Systems - Mathematical Models, 2024)

Transfer Function Model: A transfer function model is a powerful S-domain representation used for analyzing Linear Time-Invariant (LTI) systems in control engineering. Defined as the ratio of the Laplace transform of the output to the Laplace transform of the input, the transfer function assumes that all initial conditions are zero. (Control Systems - Mathematical Models, 2024)

State Space Model: This model provides a framework to model the dynamics of a system using vectors and matrices that encapsulate all relevant state variables and their interactions. This method is particularly useful for systems where the relationship between inputs and outputs is complex, including multi-input and multi-output (MIMO) scenarios.

2.4.2 Transfer Functions

The transfer function, denoted as G(s), stands as a cornerstone in control systems, forging a connection between a system's input and output. Operating within the s-domain, it presents a complex portrayal of system dynamics, diverging from the time-domain framework of differential equations. Below we can see what a graphical representation looks like with a block diagram (Figure 5).

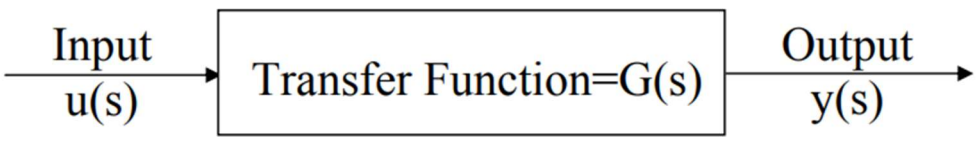


Figure 5: Transfer Function diagram Source: https://by.genie.uottawa.ca

This function, G(s), is inherently independent of the specific input applied to the system. It solely reflects the system's potential responses based on its structural dynamics. While it

offers a high-level view of the output relative to the input, it does not reveal detailed information about the internal structure or mechanisms of the system. As a result, different systems can have identical transfer functions but different internal configurations.

The transfer function is instrumental in predicting how a system will respond to various inputs. It allows for the computation of outputs or responses for a range of inputs, making it invaluable in system design and analysis. (*MATHEMATICAL MODELING of DYNAMIC SYSTEMS*, n.d.)

2.4.3 Block Diagrams

Block diagrams are visual representations that illustrate the functional relationships within a system model. Each block within the diagram symbolizes either a physical or functional component of the system and typically contains the transfer function of that component, which describes how the component's input is transformed into its output. (*MATHEMATICAL MODELING of DYNAMIC SYSTEMS*, n.d.)

Block diagrams are made up of several key elements, a graphical representation of them is also attached:

Blocks (Figure 6)

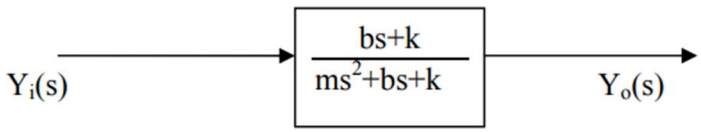


Figure 6: Blocks Source: https://by.genie.uottawa.ca

These are the primary elements that represent the functions performed by system components. Each block is labelled with a transfer function that mathematically relates the input to the output of the component.

Summation Junctions (Figure 7)

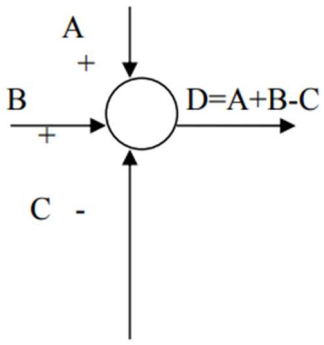


Figure 7: Summation Junctions Source: https://by.genie.uottawa.ca

These points within the diagram indicate where multiple signals are summed or subtracted, depending on their respective signs, to form a single output signal.

Paths (Figure 8)

Figure 8: Paths Source: https://by.genie.uottawa.ca

These lines illustrate the direction of signal flow between blocks and junctions, indicating how each part of the system is interconnected.

Branching Points (Figure 9)

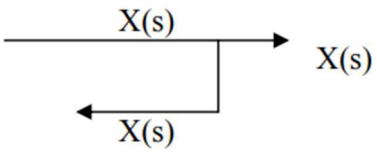


Figure 9: Branching Points Source: https://by.genie.uottawa.ca

These are nodes in the diagram where a signal path diverges, splitting a single input signal into multiple branches that can be routed to different parts of the system.

2.4.4 First Order Response Analysis

First-order systems are among the simplest and most common types of dynamic systems encountered in control engineering. These systems are characterized by a single energy storage element, which results in a first-order differential equation describing the system dynamics. The key parameters used to analyze first-order systems are lag, time constant, and gain. Understanding these parameters helps in predicting how the system will respond to various inputs. Here's a detailed look at each of these aspects:

Gain (K):

The gain of a first-order system indicates the steady-state change in output per unit change in input. In other words, it is a scalar factor that describes how much the output amplitude is scaled from the input amplitude under steady-state conditions.

Time Constant (τ):

The time constant of a first-order system is a measure of the time it takes for the system's response to reach approximately 63.2% of its final (steady-state) value after an input step change. This parameter is fundamental because it describes the speed at which the system responds to changes. In practical terms, the time constant determines how quickly a system can reach its new equilibrium after a disturbance. Shorter time constants mean faster response but potentially less stability, depending on other system parameters.

Lag (L):

Lag, or dead time, in a first-order system, refers to the delay between the input being applied and the output starting to respond. This is especially critical in processes where timing is crucial, as it affects how quickly a system can adjust to changes. A significant lag can lead to slower response times, making a system less efficient in applications requiring quick adjustments. In control systems, compensating for lag is crucial for maintaining stability and accuracy.

For a first-order system, the transfer function that includes the system's gain, time constant, and lag (assuming a step input) can be expressed as follows:

$$g(t) = K\left(1 - e^{-\frac{t-L}{\tau}}\right) u(t-L) \qquad (1)$$

Here, $u(t-L)$ represents the Heaviside step function, which is zero for $t < L$ and one for $t \geq L$. This function ensures that the system's response, g(t) is zero for $t < L$ and follows the formula $K\left(1 - e^{-(t-L)/\tau}\right)$ for $t \geq L$.

To obtain the Laplace Transform of this response, consider the function's behavior in different time segments:

For $t < L$, $g(t) = 0$, and it's contribution to the Laplace Transform is zero.

For $t \geq L$, the function active is $f(t) = K\left(1 - e^{-\frac{t-L}{\tau}}\right)$.

Recognizing as a shifted function, we can use the Laplace Transform properties for time shifting and for exponential functions:

1. The Laplace Transform of 1 is $\frac{1}{s}$.
2. For a function $f(t-a)u(t-a)$ where $u(t)$ is the unit step function, the transform is $e^{-as}\mathcal{L}\{f(t)\}$.

Tranforming $f(t)$, the exponential function $e^{-\frac{t-L}{\tau}}$ transforms to $\mathcal{L}\left\{e^{-\frac{t}{\tau}}\right\} \cdot e^{\frac{L}{\tau}} = \frac{e^{\frac{L}{\tau}}}{s+\frac{1}{\tau}}$

Combining these, the Laplace Transform of the function $f(t)$ is:

$$F_a(s) = K\left(\frac{1}{s} - \frac{e^{\frac{L}{\tau}}}{s+\frac{1}{\tau}}\right) \quad (2)$$

Using the shifting property, the Laplace Transform of the function $g(t)$ is:

$$G_a(s) = K \cdot e^{-Ls}\left(\frac{1}{s} - \frac{e^{\frac{L}{\tau}}}{s+\frac{1}{\tau}}\right) \quad (3)$$

To focus solely on the process function, we consider the input as a step function represented by $\frac{1}{s}$. Consequently, the overall transfer function $G_a(s)$ is expressed as the product of $\frac{1}{s}$ and the process-specific transfer function $H_a(s)$:

$$G_a(s) = \frac{1}{s} \cdot H_a(s) \quad (4)$$

After simplifying the process function $H_a(s)$, it is derived to encapsulate the system's dynamics, including the system's gain, time constant, and lag:

$$H_a(s) = K\frac{e^{-L\,s}}{\tau\,s+1} \tag{5}$$

Since we are implementing the control system on an Arduino, which operates in a discrete-time environment, it's necessary to convert the continuous-time Laplace Transform to the equivalent Z-transform. There are several methods for performing this transformation, the most well-known being Forward Euler, Backward Euler, and Tustin (or bilinear). The latter is the best at handling the tranformation which maps the s-plane into the z-plane. The specifics of the Tustin transformation can be visualized in the accompanying diagram (Figure 10).

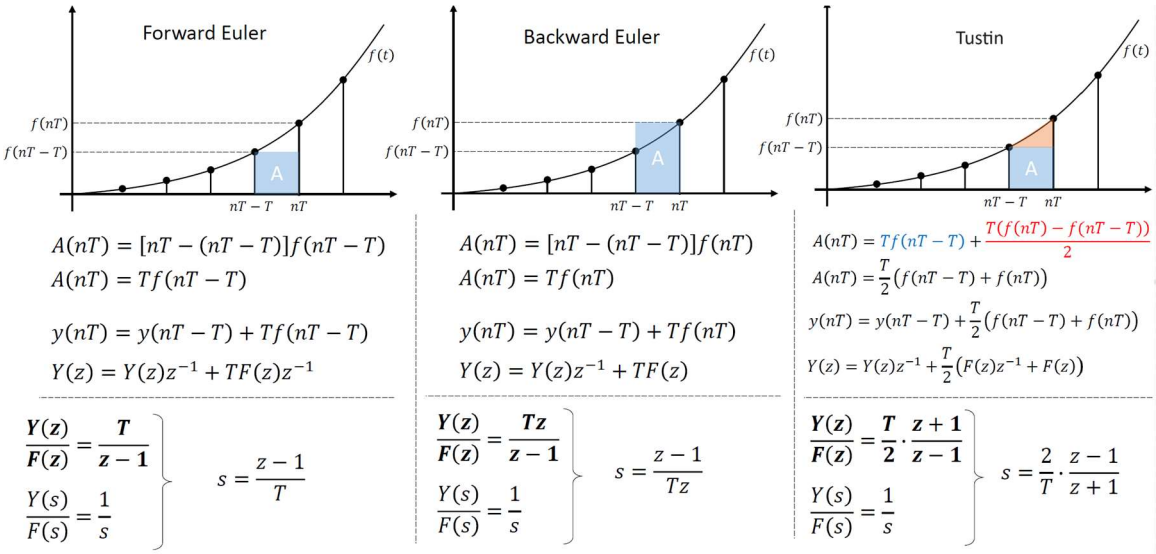


Figure 10: Z-transforms methods Source: Notes from a class at the University of Lleida

Let's break down the transformation of our Laplace Transform into the Z-domain in a step-by-step process:

For the delay term $e^{-L\,s}$ can be approximated in the Z-domain as z^{-L}, where L represents the delay in number of sampling periods.

For the denominator $\frac{1}{\tau\,s+1}$, we apply the Tustin transformation where T is the sampling time:

$$\tau\,s + 1 \rightarrow \tau\left(\frac{2}{T}\cdot\frac{z-1}{z+1}\right) + 1 = \frac{(2\tau+T)z-(2\tau-T)}{T(z+1)} \tag{6}$$

Combining these transformations, the Z-transform of our system is:

$$H_d(z) = K\cdot z^{-L}\cdot\frac{T(z+1)}{(2\tau+T)z-(2\tau-T)} \tag{7}$$

When programming in coding environment, it is advantageous to express all Z-transform terms using z^{-1} as it is considered a backward shift operator. This means if $X_d(z)$ is the Z-tranform of the sequence $x[n]$, being n the number of the sample, then $z^{-1}X_d(z)$ corresponds to $x[n-1]$, effectively shifting the entire sequence backward by one sample period. Similarly, $z^{-2}X_d(z)$ would represent $x[n-2]$, shifting the sequence backward by two sample periods. This manipulation is critical for formulating the difference equations that link current outputs $y[n]$ to both previous outputs and both current and past inputs.

Given the transfer function $H_d(z)$ and considering the above transformation strategy, the relationship between the output and input in the Z-domain can be expressed as:

$$H_d(z) = \frac{Y_d(z)}{X_d(z)} = K \cdot z^{-L} \cdot \frac{T(1+z^{-1})}{2\tau+\mathrm{T}-(2\tau-\mathrm{T})z^{-1}} \quad (8)$$

This leads to a specific form of the difference equation:

$$Y_d(z)[2\tau + \mathrm{T} - (2\tau - \mathrm{T})z^{-1}] = X_d(z)[K \cdot z^{-L} \cdot T(1 + z^{-1})] \quad (9)$$

$$(2\tau + \mathrm{T})y[n] - (2\tau - \mathrm{T})y[n-1] = (K \cdot T)x[n-L] + (K \cdot T)x[n-L-1] \quad (10)$$

Rearranging terms to solve for y[n], we find:

$$y[n] = \frac{(2\tau-\mathrm{T})}{(2\tau+\mathrm{T})}y[n-1] + \frac{(K\cdot T)}{(2\tau+\mathrm{T})}x[n-L] + \frac{(K\cdot T)}{(2\tau+\mathrm{T})}x[n-L-1] \quad (11)$$

In this context, since we are working with discrete samples, L represents the lag expressed in terms of the number of samples. To determine the value of L, divide the time lag by the sampling period, T. This calculation will give L as the number of samples by which the signal is delayed.

2.5 Types of Controllers

Within the intricate web of components constituting a control system, the controller emerges as a pivotal element responsible for optimizing performance and ensuring accuracy. This critical component functions as a regulatory mechanism, consistently striving to minimize the deviation between the system's actual value (process variable) and its predefined desired value (setpoint) (Jain D.V., 2021). This deviation, termed the error signal, serves as the fundamental input driving the controller's operations (Roshni Y, 2019).

To achieve the desired state, the controller employs a meticulous process, analyzing input signals and subsequently generating control signals based on the error signal and the predetermined setpoint. These intricate control signals are then fed back into the system, effectively influencing its behaviour and guiding it towards the targeted output. (GfG, 2023b).

The visualization of the controller's functionality often takes the form of a block diagram (Figure 11). This diagram offers a clear representation of the various components within the control system and their interconnected communication pathways. This visual aid enhances the understanding of how the controller processes inputs and generates outputs, ultimately maintaining the system's desired operation. (GfG, 2023b).

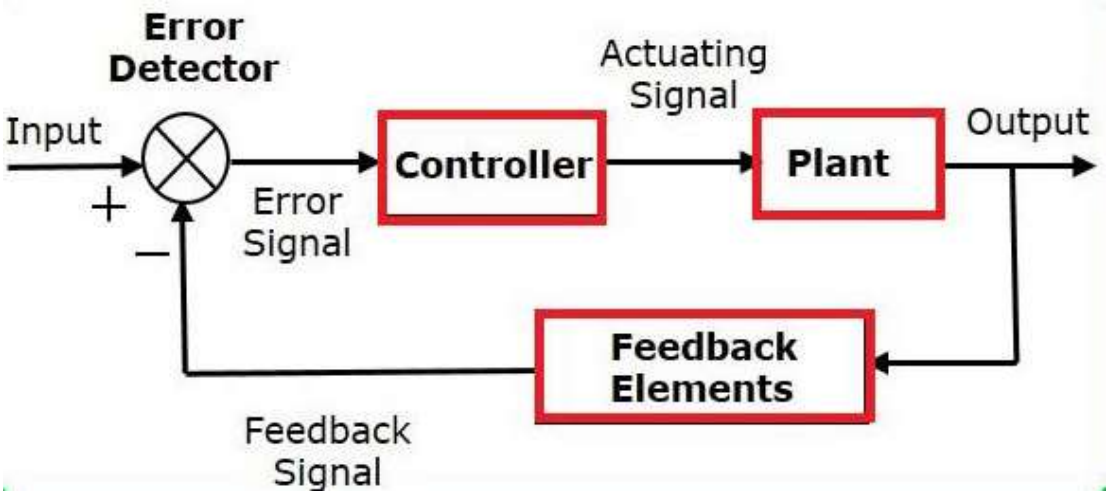


Figure 11: Block diagram form Source: https://www.geeksforgeeks.org

By implementing various control actions, the controller effectively minimizes the deviation between the actual and desired values, ensuring the system delivers the precise output. This essential component underpins the achievement of the exquisite control required in diverse engineering and automation applications.

Prior to delving into the diverse categories of controllers, it's essential to understand their operational modes. This understanding is crucial as the different types of controllers stem from distinct modes of operation. (Roshni Y, 2019). So, basically, there are two modes of operation: continuous and discontinuous.

2.5.1 Discontinuous Mode

In Discontinuous Mode, the controller outputs discrete values rather than exhibiting smooth variations in accordance with the generated signal. This results in fluctuations between discrete output levels. (Roshni Y, 2019). Within this category, two primary controller types are distinguished:

On-Off/Two-Position Controllers: The output fluctuates between two states—fully on or fully off. This type of controller, also known as an on/off controller, functions by adjusting the manipulated variable only when the measured variable crosses the setpoint. It operates as a two-position control, acting as a switch that turns off (or on as needed) when the error is positive and on (or off as needed) when the error is negative, this can be seen in Figure 12. This control method, exemplified in applications like oven and alarm systems, is suitable when precise maintenance of controlled process variables is not necessary. Despite its simplicity and cost-effectiveness, on-off control works best in systems where changes in the manipulated variable produce slow rates of change in the measured variable. (The instrument guru, 2020).

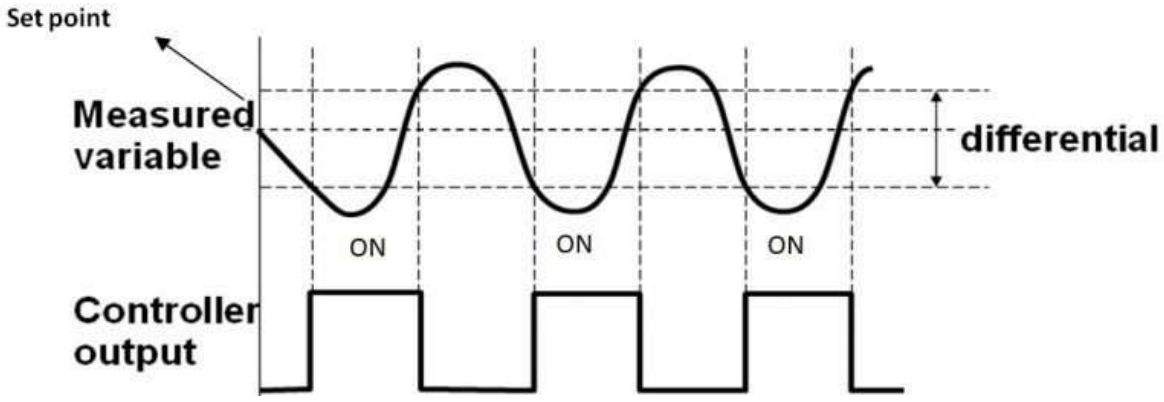


Figure 12: Output On-Off Controller vs Measured variable Source: https://theinstrumentguru.com

Multiposition Controllers: These controllers offer more than two discrete output levels, allowing for a wider range of control actions and enabling greater flexibility in system response (The instrument guru, 2020), this can be observed in Figure 13.

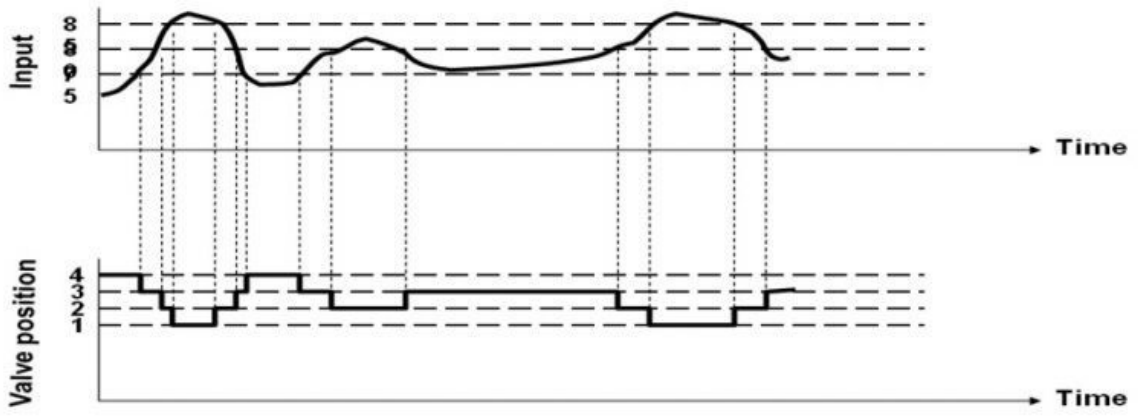


Figure 13: Output Multiposition Controller vs Input Source: **https://theinstrumentguru.com**

2.5.2 Continuous Mode

In Continuous Mode, the controller facilitates continuous variation of the output signal across its entire operational range. This translates to a proportional change in the controlled output relative to the error signal or some form of it. (Roshni Y, 2019). Based on this mode, three main types of controllers are categorized: Proportional (P), Integral (I), and Derivative (D) are parameters within PID controllers that can be adjusted or weighted to modify their impact on the process.

PID controllers offer more practicality compared to standard on/off controllers, as they enable finer adjustments within the system. However, on/off controllers have their advantages, including simplicity in design and execution, as well as the reliability and cost-effectiveness of binary sensors and actuators. (Libretexts, 2020).

Despite these advantages, utilizing an on/off controller scheme presents significant drawbacks. These include inefficiency (akin to driving with full throttle and full brakes), potential noise generation during stability-seeking phases (resulting in dramatic overshoot or undershoot of set-points), and accelerated physical wear on valves and switches due to continuous full-on/full-off cycling. (Libretexts, 2020).

The Process Gain (K) denotes the ratio of change in the output variable (response) to the change in the input variable (forcing function). It quantifies the sensitivity of the output to a given input alteration. Gain, a steady-state parameter, offers no insights into process dynamics and remains independent of design and operating variables. It comprises three components: sign, value, and units. The sign indicates the output's response to the input—

positive for output increase with input rise and negative for output decrease with input rise. Units vary based on the specific process and variables involved. (Libretexts, 2020).

Dead Time (t_0) signifies the delay between an input change and the onset of the output response. It significantly impacts the controllability of a system, as changes in setpoint are not immediate due to this parameter. Dead time must be carefully considered during tuning and process modelling. (Libretexts, 2020).

Proportional Controller: the simplest form of continuous control in closed-loop systems, utilizes a linear relationship between the error of a system and the controller output. P-only control minimizes process variable fluctuations, it may not always reach the desired setpoint precisely. Despite its faster response compared to other controllers, it may introduce a deviation from the setpoint known as offset, similar to a systematic error in a calibration curve. This offset, inherent in each equation, cannot be fully eliminated but can be mitigated by combining P-only control with other forms such as I- or D-control. (Libretexts, 2020).

Consider a scenario where a proportional controller continuously adjusts manipulated variables to balance process input with demand. Its output, proportional to the error, is determined by the instantaneous process error (e(t)), the Controller output with zero error (p0), and the proportional gain (Kp). This control mechanism ensures that any oscillations are removed, and the system returns to a steady state. However, as the system becomes more complex, the response time difference may accumulate, allowing the P-controller to potentially respond even faster. (The instrument guru, 2020).

$$P_{out} = K_p e(t) + p0 \quad (12)$$

To illustrate, let's examine the control of fluid level in a tank using P-only control. If outflow decreases, causing the level to rise, the P-only control adjusts the outflow to restore equilibrium. However, the steady-state level differs from the initial setpoint, representing the P-control offset. (Libretexts, 2020).

Integral Controller: A controller exhibiting integral control action adjusts the output proportionally to the integral of the error signal. Mathematically, it can be represented as:

$$I_{out} = K_i \int_0^t e(t)\, dt \quad (13)$$

Where (K_i) denotes the integral constant or controller gain. Integral control, also referred to as reset control, responds to the cumulative sum of past errors, aiming to eliminate steady-state errors over time. Unlike proportional control, the integral controller's output depends on the integrating time constant, making it comparatively slower. (Libretexts, 2020).

The integral controller continuously integrates the error signal, aiding in the maintenance of setpoints and the elimination of steady-state errors. It integrates the error signal over time, amplifying it by the integral gain, and adds this accumulated error correction to the control signal. However, improper tuning of the integral controller can lead to slower responses and potential overshoots. (GfG, 2023b).

It serves as the second form of feedback control, effectively removing deviations and ensuring the system returns to its original setting. While I-only controllers guarantee no offset in the system, they inherently possess slower response times due to their dependency on more parameters. Combining integral control with other forms such as proportional or proportional-derivative (PD) control can mitigate these drawbacks. (Libretexts, 2020).

Integral control, based on accumulated past error, adjusts deviations in proportion to their cumulative magnitude. Its key advantage lies in offset elimination, crucial for maintaining precise control within narrow ranges. Nonetheless, it can destabilize the controller and may lead to integrator windup, prolonging the time required for control adjustments. (Libretexts, 2020).

Derivative Controller: The derivative controller operates by reacting to the rate of change of the error signal, providing control action to counteract anticipated future error trends (GfG, 2023b). Its output is derived from the error signal with respect to time, expressed as:

$$D_{out} = K_d \frac{d}{dt} e(t) \quad (14)$$

Where (K_d) denotes the derivative constant. This control mechanism aids in damping oscillations and enhancing system stability, as it anticipates future errors based on the error rate of change (Roshni Y, 2019).

Unlike proportional and integral controllers, which focus on present and past states of the error, respectively, derivative control anticipates future states and acts accordingly. By

analyzing the change in error, minimizes deviations and maintains system consistency. However, D-control alone is rarely used due to its potential for sudden large changes in controller output, especially with fast error rate changes. Therefore, it is typically employed in conjunction with proportional or proportional-integral control actions. (Libretexts, 2020).

Mathematically, derivative control is the opposite of integral control, as it measures the change in error over time. It is more complex than proportional control, which can affect the controller's response time. While derivative control is effective in processes with rapidly changing outputs, its implementation requires careful consideration of system dynamics and potential oscillations. (Libretexts, 2020).

3 Practical Framework

This section details the tools used to carry out the Arduino-based simulation designed for control systems education at NOVIA University. It begins with an overview of the essential hardware components used, including Arduino boards, sensors and actuators, explaining their integration and functional roles within the system. The software environment, especially the use of the Arduino IDE and other interface tools, is also briefly discussed to describe how interactions and system responses are handled.

3.1 Arduino

Arduino is an open-source electronics platform that features user-friendly hardware and software. It's designed to read various inputs like light on a sensor, a finger on a button, or a Twitter message and convert them into outputs such as activating a motor, lighting up an LED, or posting online. Users can control their Arduino boards by writing commands in the Arduino programming language (based on Wiring) using the Arduino Software (IDE), which is based on Processing.

Originating from the Ivrea Interaction Design Institute, Arduino was created as a simple tool for rapid prototyping, accessible to students without prior experience in electronics or programming. As its popularity expanded beyond academic circles, the Arduino platform evolved to meet diverse requirements and challenges. It has grown to include a range of products from basic 8-bit boards to advanced offerings suitable for IoT projects, wearable technology, 3D printing, and embedded applications. (What is Arduino?, 2022).

3.1.1 Arduino Nano 33 IoT

We selected the Arduino Nano 33 IoT (Figure 14) for our project as there were ample units available for use in our laboratory. Additionally, this Arduino model perfectly aligns with our project requirements. Below is a brief overview of its capabilities. While we may not utilize all of its features initially, understanding its full range of functions is beneficial for potential future enhancements.

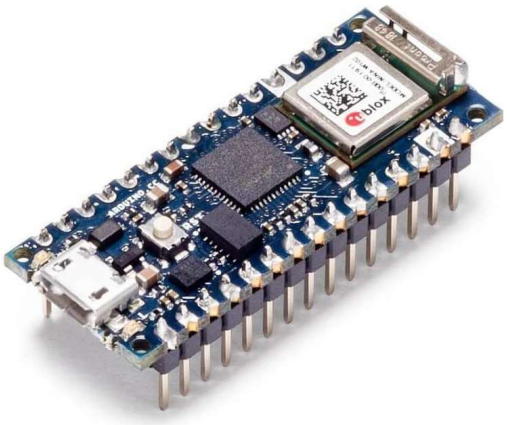


Figure 14: Arduino Nano 33 IoT Source: https://my.cytron.io

The Arduino Nano 33 IoT features a powerful SAMD21 Cortex-M0+ ARM MCU, with 256KB of flash memory and 32KB of SRAM, supporting a wide range of applications. It offers WiFi and Bluetooth connectivity via the ESP32-based Nina W102 module, compatible with IEEE 802.11b/g/n standards and includes both Bluetooth® BR/EDR and BLE capabilities for diverse wireless applications.

This board is rich in input/output options, including UART, I2C, and SPI interfaces for easy device and sensor integration. It also boasts several timers, a 12-bit ADC, a 10-bit DAC, and a PWM output, suitable for everything from simple LED operations to complex robotic controls. Security is enhanced with an ATECC608A Crypto Chip for secure key and certificate storage, essential for secure IoT projects.

Its compact size allows for flexible deployment in space-constrained environments, operable as a DIP or SMT component. Power management features like Power-on Reset (POR) and Brown-out Detection (BOD) ensure stable operation across varying power conditions. Additionally, it includes a 6-axis IMU with a 3D accelerometer and gyroscope, ideal for motion tracking in wearable devices and drones.

For more detailed information on this Arduino's features, please refer to Appendix 1, which contains the datasheet.

3.1.2 Arduino Cloud Editor

Initially, we utilized the offline Arduino IDE, which was convenient for coding and uploading directly to the board. However, the need to download numerous libraries presented a

challenge. To streamline our workflow and reduce dependency on local installations, we transitioned to an online version of the editor.

The Cloud Editor, part of Arduino Cloud, is an online platform where you can develop Arduino projects. It allows you to write, compile, and upload sketches directly to your Arduino board. The editor automatically saves your progress to the cloud. Key features include a compiler to check code compatibility with your board, an upload tool that uploads a sketch to your board, a Serial Monitor for reading serial data from your board, and access to all necessary board packages and libraries without the need for downloading. (Cloud Editor, 2024).

3.2 Liquid Crystal Display (LCD)

A Liquid Crystal Display (LCD) is used to display the adjustable parameter values set by the operator, in this instance, the teacher. For this purpose, we have chosen the RC2004A-BIW-CSX LCD module (Figure 15), which is widely used at NOVIA University.

Figure 15: RC2004A-BIW-CSX LCD module Source: https://www.adelaida.ro

The RC2004A-BIW-CSX LCD module is a character-type display developed by Raystar Optronics. It features a dimension of 98.0 x 60.0 x 13.6 mm with a viewing area of 77.0 x 25.2 mm. The active display area is 70.4 x 20.8 mm. This LCD can display up to 20 characters across 4 lines, with each character formed in a 5x8 dot matrix.

The display uses a blue STN negative transmissive LCD with LED backlighting, offering a viewing direction at 6 o'clock. It supports a 16-level duty cycle for the display and includes a built-in controller capable of handling both display data RAM and character generator RAM, allowing for customizable character patterns.

The module is designed with various interface pins for power, data input, and control signals, including options for instruction and data interfacing, making it versatile for different uses in embedded systems. It operates on a 5V supply voltage for logic and has specific requirements for LCD drive voltage, influenced by operating temperature.

3.3 Rotary Encoder

A rotary encoder is a position sensor that determines the angular position of a rotating shaft by generating electrical signals, either analogue or digital, based on rotational movements. Here's a breakdown of its operation:

The encoder features a disk with evenly spaced contact zones connected to a common pin (C) and two other contact pins (A and B). As the disk rotates incrementally, pins A and B make contact with the common pin C, producing two square wave output signals.

To determine the position, one could simply count the pulses from either output signal. To ascertain the direction of rotation, however, both signals must be analyzed together. These output signals are phase-shifted by 90 degrees relative to each other. During clockwise rotation, output A leads output B. By monitoring the transitions from high to low or low to high in these signals, where they show opposite values, one can deduce the direction of rotation. Conversely, if the encoder rotates counterclockwise, the output signals will match. Both ways are illustrated in Figure 16. This phase relationship allows for programming a controller to read both the position and direction of the encoder's rotation. (Dejan, 2016).

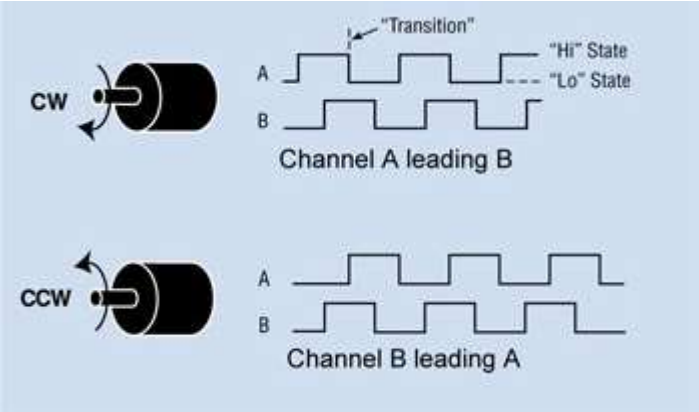


Figure 16: Relation between rotation and outputs pins A and B Source: https://electronics.stackexchange.com

We will use the EC12E244407 encoder, commonly used in NOVIA's university, for incrementing or decrementing the values of the parameters. The rotary encoder that we use is the EC12E244407, also very popular. This particular encoder also includes a push-button function, allowing us to switch between configuring different parameters seamlessly.

4 Methodology

This section outlines the systematic approach taken to design and implement the Arduino-based simulation tool for control systems education at NOVIA University. It details the step-by-step hardware installation and configuration processes that form the backbone of the project. We will explore how the Arduino platform, along with various sensors and actuators, are configured to simulate realistic control system scenarios. This section also delves into the software development aspect, explaining how the Arduino IDE and custom scripts are utilized to create a dynamic simulation environment. By providing a clear description of the setup procedures, configuration settings, and the rationale behind each decision, this section aims to impart a comprehensive understanding of the practical application of theoretical control system concepts through hands-on experimentation.

4.1 Hardware installation and configuration

In this subsection, we detail the hardware setup critical for the functioning of the Arduino-based simulation tool. Here, you'll find comprehensive information on assembling the physical components, including the Arduino board, various sensors, actuaries, and the connections between them. This part of the methodology explains how each component is integrated into the overall system to ensure smooth operation and data flow, setting the stage for the software integration that follows.

4.1.1 Setting Up the LCD Module for Arduino

In order to get the LCD operational, a series of steps must be followed. The module consists of 16 pins, which are categorized into four groups as seen in Figure 17: power pins, control pins, data pins, and LED pins. These are detailed below:

Power pins:

Pin 1 (VSS) serves as the ground connection.

Pin 2 (VDD) is connected to a +5V supply to power the LCD.

Control pins:

Pin 3 (V0) is used for adjusting the display's contrast.

Pin 4 (RS) toggles between command and data modes; setting it low for command mode and high for data mode.

Pin 5 (R/W) controls read and write operations, typically set to write (low) mode.

Pin 6 (E) enables the LCD, requiring a high-to-low pulse to process commands or data.

Data pins (pins 7 to 14):

These are used for transferring data to and from the LCD. In 4-bit mode, only D4 to D7 (pins 11 to 14) are used, which reduces the number of required GPIO pins by sending data in 4-bit nibbles. In 8-bit mode, all eight pins are utilized, allowing data to be sent in bytes.

LED pins:

Pin 15 (LED+) is the anode of the LED backlight.

Pin 16 (LED-) is the cathode of the LED backlight.

(Liquid Crystal Display with Arduino, 2023)

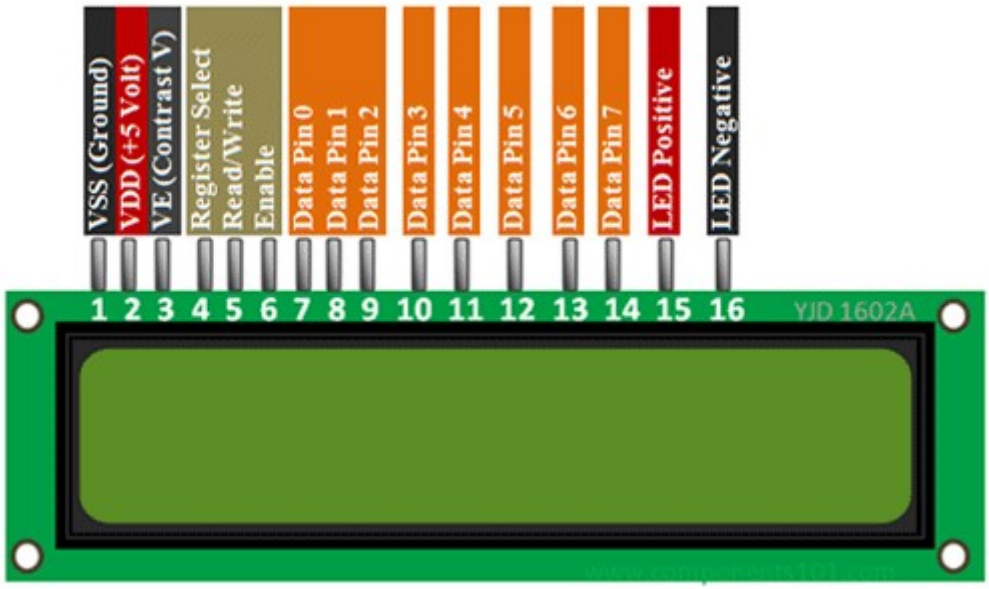


Figure 17: LCD pins Source: https://microcontrollerslab.com

Prior to making connections, I soldered a pin connector onto the LCD module, as illustrated in Figure 18. This step was crucial for facilitating straightforward and stable connections between the LCD and other components.

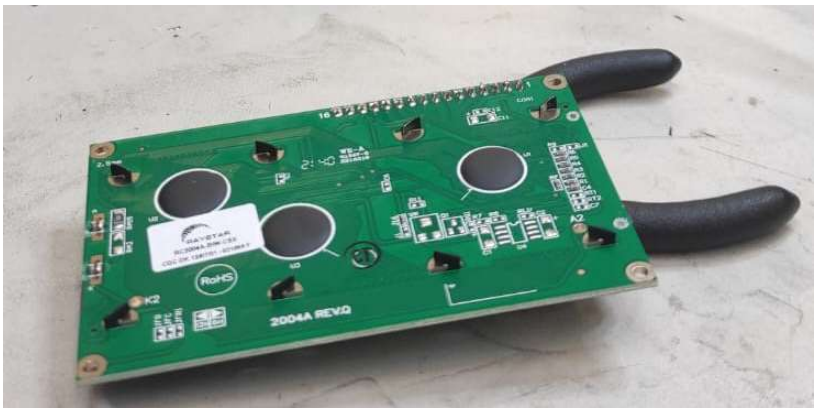


Figure 18: LCD module soldered with the pin connector Source: authors' own

Mostly all LCD modules have the same connections, as we can see in Figure 19 there is shown a schematic between the LCD and the Arduino. The Arduino is not the same as the one we have but is still valid for the pins connections. For knowing the pins of the Arduino we are using see Figure 20.

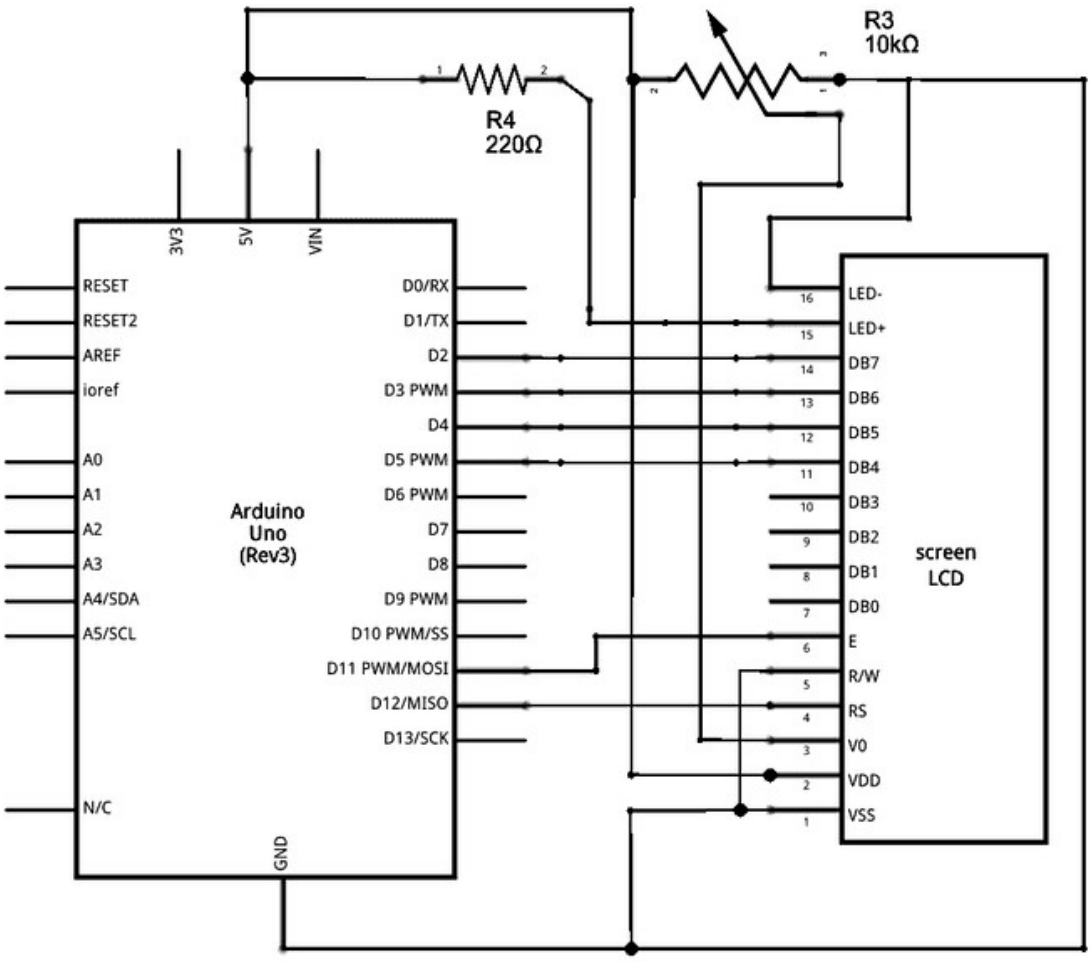


Figure 19: Schematic connection between LCD and Arduino Source: https://docs.arduino.cc

The RS pin connects to digital pin 12.

The Enable pin to digital pin 11.

Data transmission is handled via pins D4 to D7, which connect to digital pins 5, 4, 3, and 2, respectively.

The R/W, VSS, and LED- pins are grounded.

The VDD pin is connected to 5V.

The LED+ pin is connected to 5V through a 220 ohm resistor.

Contrast is adjusted via a 10k potentiometer linked to the LCD's V0 pin.

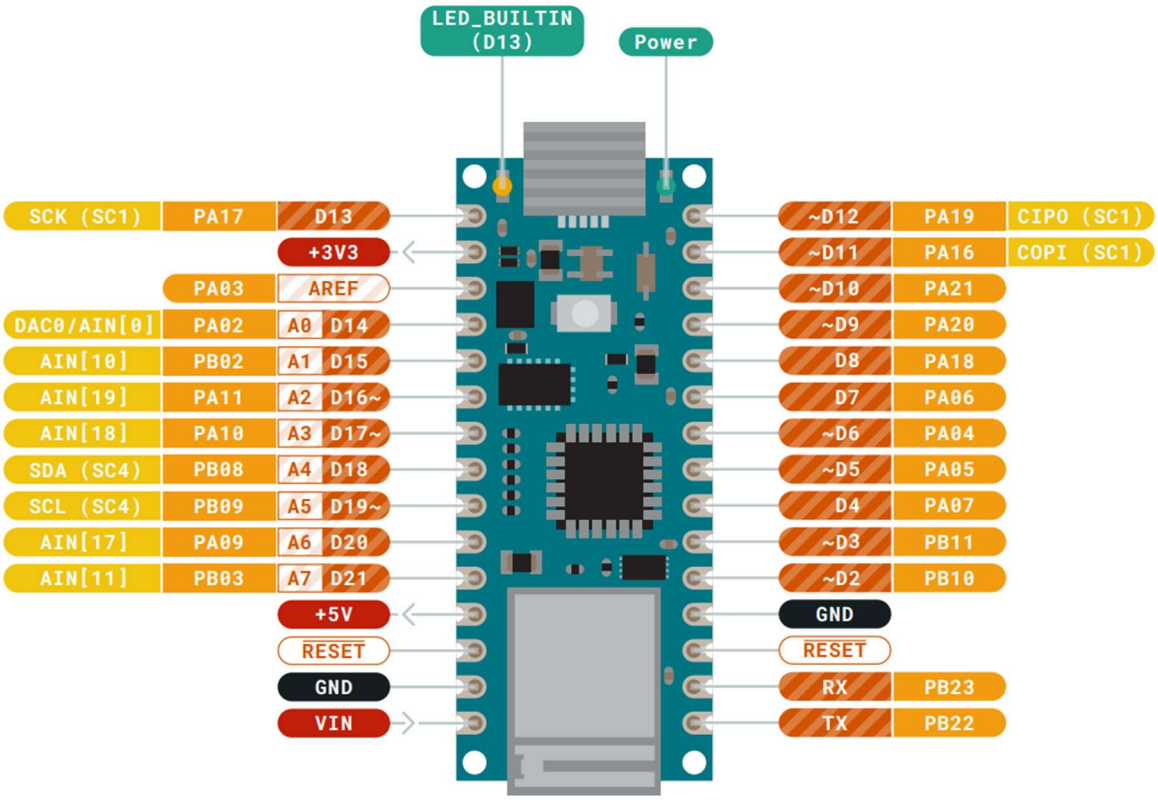


Figure 20: Pinout Arduino Nano 33 IoT Source: https://docs.arduino.cc

As it is mentioned before we use +5v, but reading the datasheet of the Arduino Nano 33 IoT (Appendix 1) it is said that the microcontroller runs at 3.3v, which means that we must never apply more than 3.3v to its Digital and Analog pins as if not this could damage the Arduino. To avoid such risk this microcontroller has a 5v pin but it is not connected as the default factory setting. So as we needed them we had to solder a bridge on the two pads marked as VSUB (Figure 21).

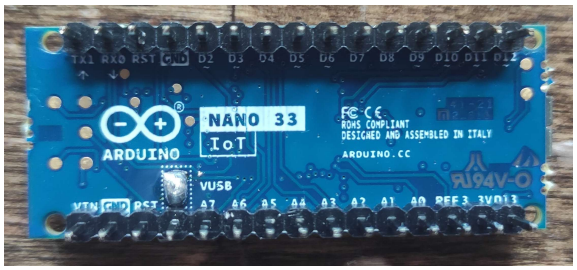


Figure 21: Solder bridge on the two pads VUSB Source: authors' own

4.1.2 Rotary encoder with a push button

This rotary encoder with a push button features 5 pins (labelled A, B, C, D, and E) where the first three are dedicated to the encoder functionality, and the last two are connected to the push button.

Pins A and B, often referred to as channels, are essential for determining the direction of rotation, either clockwise or counterclockwise. These are connected through a 10k ohm pull-up resistor to digital pins 6 and 7 on the microcontroller, respectively. Pin C and pin E are both grounded. Meanwhile, pin D connects to digital pin 8.

Due to issues with poor connections, we had to solder the wires directly to the pins, as depicted in Figure 22. In this configuration, the green wire corresponds to pin A, black to pin B, yellow to pin C, white to pin D, and red to pin E.

Figure 22: Rotary encoder connections Source: authors' own

4.1.3 Input and output of the simulation plant

Connections for both the input and output of the simulation plant need to be established. The input is connected to the analogue pin A1, while the output is linked to the digital-to-analogue converter pin DAC0. To simplify the connections between the external cables of the simulation plant (specifically, the connections between the PID controller and the plant), we soldered the input and output, along with their respective grounds, to a convenient connector, as shown in Figure 23.

Figure 23: Solder between input/output and the connector Source: authors' own

4.1.4 Hardware Encapsulation and System Layout

After confirming that everything was functioning properly, it was time to enhance the connections further. Initially, we removed the connectors that had been soldered onto the LCD display and directly soldered the cables onto it instead, as illustrated in Figures 24 and 25.

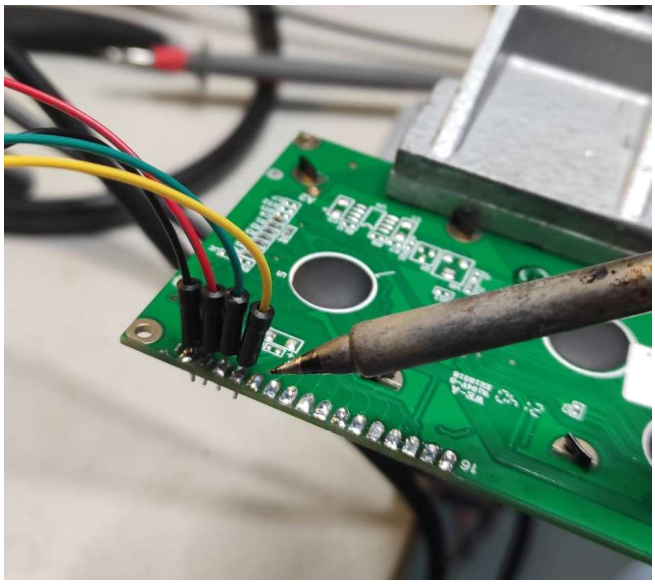


Figure 24: Soldering cables on the LCD display Source: authors' own

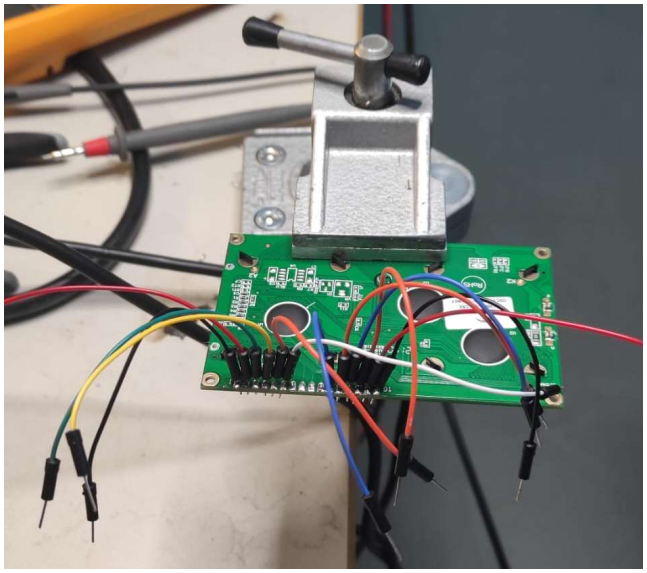


Figure 25: LCD display solder connections Source: authors' own

We also improved the connections for the potentiometer by soldering them directly (Figure 26).

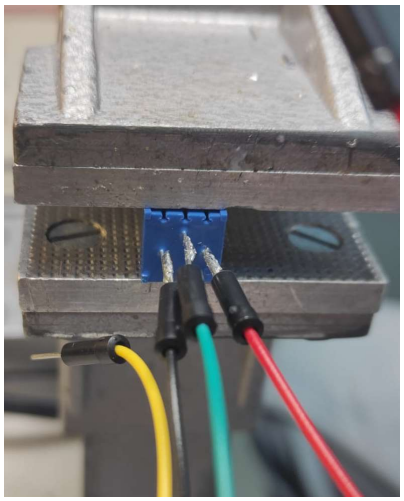


Figure 26: Solder connections of the potenciometer Source: authors' own

For the project enclosure, we marked the transparent box lid for drilling. These marks indicated where holes would be made for the potentiometer, the LCD display, the rotary encoder, and the input and output connectors (Figure 27).

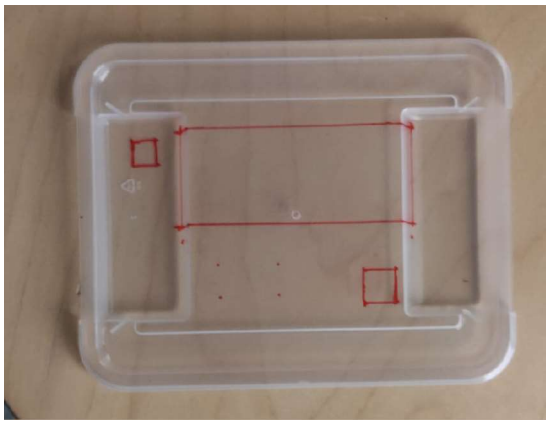


Figure 27: Lid measurements Source: authors' own

Using a drill, we created four holes for the input, output, and both grounds, ensuring the drill bit size matched the size of the connectors (Figure 28).

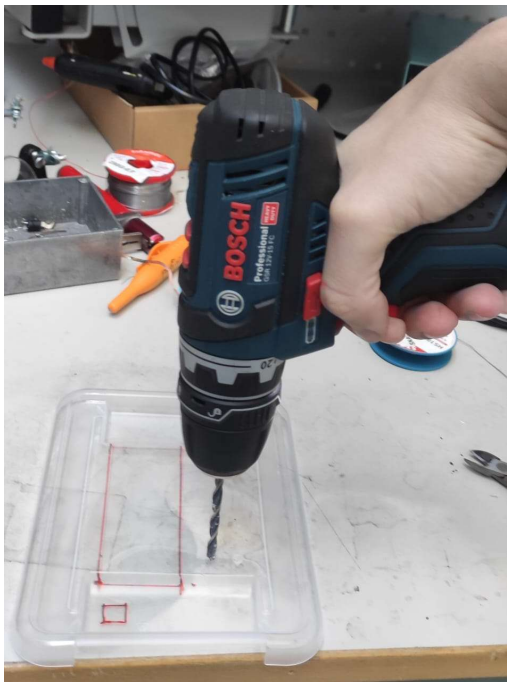


Figure 28: Drilling the lid Source: authors' own

For the rotary encoder, potentiometer, and LCD display, which required rectangular holes, we used a soldering iron to melt the plastic and form the necessary openings (Figure 29).

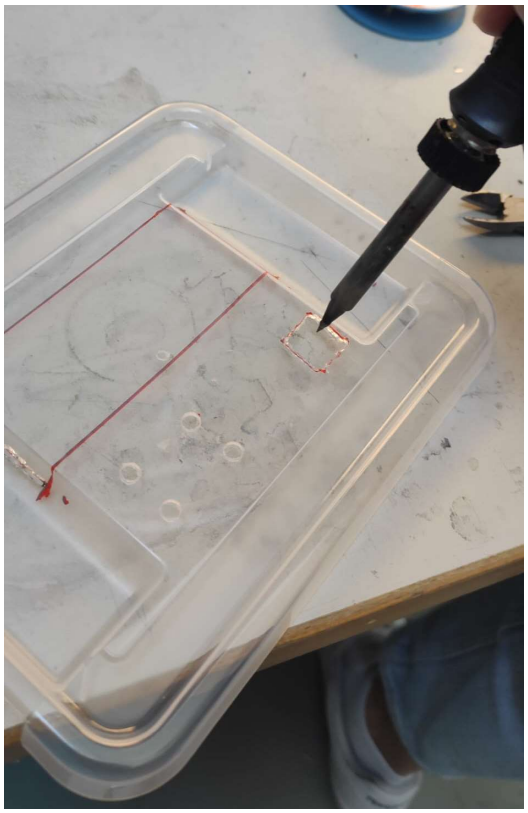


Figure 29: Piercing the lid Source: authors' own

Once all holes were made, we positioned all components in place, securing them with glue to ensure stability and prevent short circuits (Figure 30).

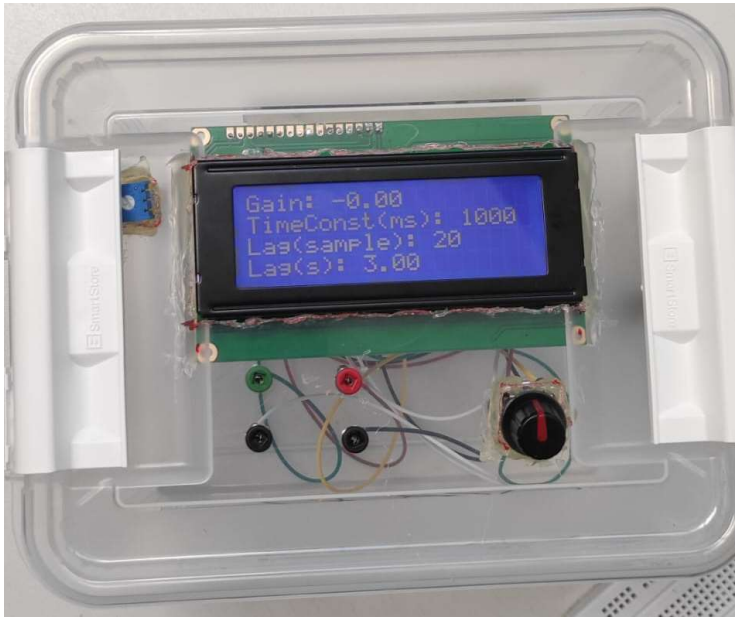


Figure 30: Top view of the final result Source: authors own

Finally, we installed the remaining connections and the Arduino inside the box and mounted it on the wall. We also made a hole in the side of the box to route the power cable through (Figure 31). After securing everything, we conducted a final test to confirm that all components were working correctly.

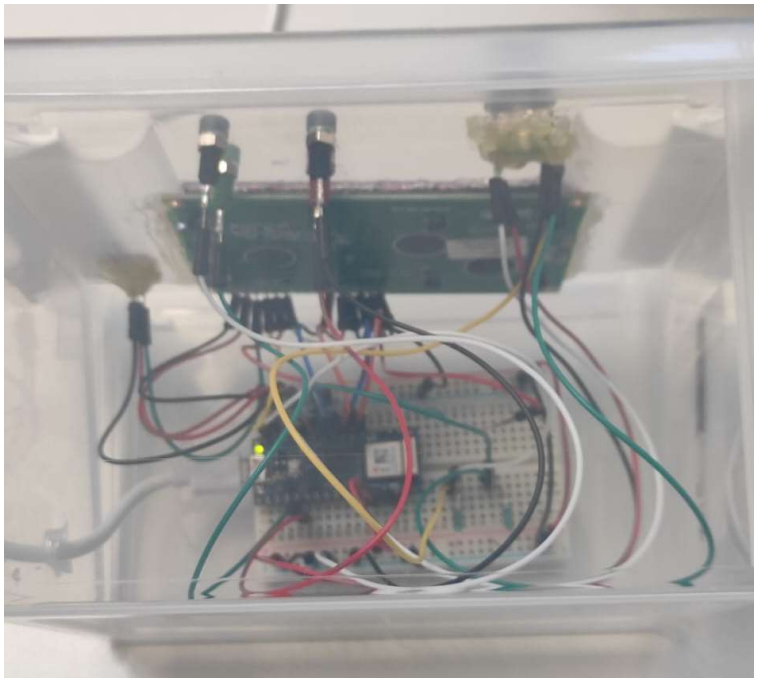


Figure 31: Side view of the final result Source: authors' own

4.2 Software configuration

In terms of the software, it is entirely programmed in C, utilizing just a single library (LiquidCrystal.h) to manage the LCD display. We will detail each segment of the code, which is fully documented and available in Appendix 2.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

In this section of the code, the LiquidCrystal.h library is included for later use. Additionally, the LCD display is initialized with the Arduino pins it is connected to, as specified in the function called LiquidCrystal lcd(pinRS, pinE, pinD4, pinD5, pinD6, pinD7`.

```
const int pinA = 6;
const int pinB = 7;
const int buttonPin = 8;
```

In this section, we define the pin numbers for the rotary encoder's dial and button as integer constants because they will remain connected to the same pins. The pins for the rotary encoder A and B are named pinA and pinB, respectively, and the button's pin is named buttonPin for pin D.

```
const int dacPin = A0;
const int analogInputPin = A1;
```

We also define integer constants for the input and output of our simulation plant. The DAC output, located at A0, is named dacPin, and the pin that reads the analogue input signals, located at A1, is named analogInputPin.

```
#define MAX_BUFFER_SIZE 34
```

Here, we define the maximum buffer size for storing past samples, which is calculated based on the maximum expected lag. Currently set at 34, this implies that with a sampling time of 150ms, the longest lag that can be accommodated would be 5.1 seconds.

```
float gain = 1.0;
int timeConstantms = 1000;
int lagSample = 20;
```

We also define the system parameters as variables, not constants, to allow for adjustments during operation. The gain is defined as a float to facilitate small incremental adjustments

and is initially set to 1. The timeConstantms is defined as an integer since it's measured in milliseconds and decimals are unnecessary; it starts at 1000ms. The lagSample represents the lag in terms of the number of samples, and since this pertains to sample counts, it is appropriately an integer, initialized to 20.

```
float timeConstant = timeConstantms / 1000.0;
```

For use in the equation, we need the time constant in seconds, so we convert it from milliseconds to seconds. This is why we declare it as a float variable to accommodate the decimal value resulting from this conversion.

```
float sampleTime = 0.15; // 150 ms
```

The sampleTime variable is specified in seconds, which is why it is declared as a float. We believe that a sampling time of 150ms (0.15 seconds) is sufficient for the system to function effectively.

```
float inputBuffer[MAX_BUFFER_SIZE];
```

Here, we create an array to store past input samples, functioning as a circular buffer. The array is declared as float type because we want to retain the decimal parts of the input, ensuring accuracy in data handling.

```
int bufferIndex = 0;
```

We also create a bufferIndex variable to track the current position in the circular buffer. It is initialized to 0 and is defined as an integer since it represents index positions within the array.

```
float y_prev = 0;
```

We initialize a variable to store the previous output, which is essential for recursive computation. This variable is declared as a float because we want to retain the decimal part of the output for precise calculations.

```
int lastStateA;
int selectedParameter = 0;
```

We have defined variables to manage the encoder: lastStateA, an integer that stores the last state of pinA to detect changes, and selectedParameter, an integer initially set to 0, used to track the current parameter being adjusted by the encoder.

```
void setup() {
  pinMode(dacPin, OUTPUT);
  pinMode(analogInputPin, INPUT);
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  Serial.begin(9600);
  lcd.begin(20, 4);
  analogWriteResolution(10);
  memset(inputBuffer, 0, sizeof(inputBuffer));
  lastStateA = digitalRead(pinA);
  updateDisplay();
}
```

The setup function initializes the Arduino's hardware configuration. First, we define the pin modes: dacPin is set as an output, while analogInputPin, pinA, and pinB are set as inputs. buttonPin is also configured as an input but in pull-up mode. When configured with INPUT_PULLUP, the Arduino activates an internal pull-up resistor, which keeps the pin in a HIGH state under normal conditions. Pressing a button wired to ground this pin changes its state to LOW, and releasing it allows the pull-up resistor to return it to HIGH.

Next, we start serial communication to assist with debugging. We initialize the LCD display using the lcd.begin(20, 4) function, where 20 represents the number of columns and 4 the number of rows. We then set the DAC resolution to 10 bits.

We also initialize the buffer by filling it with zeros, as there is no prior signal data available at startup. Additionally, we set lastStateA to the current state by reading the signal from pinA.

Finally, we call the updateDisplay function, which will be described later, to manage the LCD updates based on system changes and user inputs.

```
void loop() {
  static unsigned long lastUpdateTime = 0;
  if (millis() - lastUpdateTime > sampleTime * 1000) {
    lastUpdateTime = millis();

    float x_curr = float(analogRead(A1)) *3.5 / 1023;
    inputBuffer[bufferIndex] = x_curr;

    int indexL = (bufferIndex - lagSample + MAX_BUFFER_SIZE) %
    MAX_BUFFER_SIZE;
    int indexL1 = (indexL - 1 + MAX_BUFFER_SIZE) % MAX_BUFFER_SIZE;
```

```
    float y_curr = calculateDiscreteOutput(inputBuffer[indexL],
    inputBuffer[indexL1]);
    if (y_curr > 2.25) {
      y_curr = 2.25;
    }

    int y_out = (y_curr / 3.3) * 1023;
    analogWrite(A0, y_out);

    bufferIndex = (bufferIndex + 1) % MAX_BUFFER_SIZE;
  }

  handleEncoder();
  handleButton();
}
```

The main loop of the program begins by tracking the last update time. We then check if it's time to take a sample; if so, we update lastUpdateTime with the current time. The input is read as a float and scaled to a voltage value since the analogRead function returns a value between 0 and 1023 on a 10-bit scale, and the maximum input voltage is 3.5 volts. We perform this conversion to appropriately scale the input signal.

We store the current input in a variable named x_curr and also place it into the circular buffer. With the buffer updated, we can retrieve the correct past sample values required for our computation. Specifically, we need two samples: the one at n-L (the lag time ago) and n-L-1 (the sample just before the lag time). We calculate the indices for these samples, indexL and indexL1, and store these values in the respective variables.

Using these indices, we call a function that computes the current output, stored in y_curr. This output computation will be detailed later. If y_curr exceeds 2.25 volts, we limit it to 2.25 volts to match the DAC's maximum output capability.

The DAC operates on a 10-bit scale where the maximum voltage should be 3.3 volts. Therefore, we convert y_curr to match this scale using the analogWrite function on pin A0, which is connected to the DAC.

Within this conditional block, we also update bufferIndex to prepare for the next sample. Additionally, within the loop, functions handleEncoder and handleButton are called to manage user inputs and adjustments, which will be explained in further detail later. This loop effectively manages sampling, signal processing, and output while handling user interactions through the encoder and button.

```
float calculateDiscreteOutput(float x_L, float x_L1) {
  float a = (2 * timeConstant - sampleTime) / (2 * timeConstant +
  sampleTime);
  float b = (gain * sampleTime) / (2 * timeConstant + sampleTime);

  float y_curr = a * y_prev + b * x_L + b * x_L1;

  y_prev = y_curr;

  return y_curr;
}
```

In this function, we compute the output value based on the input values x[n-L] and x[n-L-1], as well as the previous output value y[n-1]. If we remember the equation it was like this:

$$a \cdot y[n-1] + b \cdot x[n-L] + b[n-L-1] \quad (15)$$

In this process, we first calculate the constants a) and b), then correctly apply the equation and store the resulting output in the variable y_curr. After computing this value, we update the previous output for use in the next cycle and return y_curr from the function.

```
void handleEncoder() {
  int stateA = digitalRead(pinA);
  if (stateA != lastStateA) {
    if (digitalRead(pinB) != stateA) {
      incrementValue();
    } else {
      decrementValue();
    }
    timeConstant = timeConstantms / 1000.0;
    updateDisplay();
  }
}
```

The handleEncoder function activates when the encoder is rotated. It begins by reading the current state of pinA and comparing it to lastStateA to determine if the encoder has been turned. A change between these states indicates a rotation. To establish the direction of rotation, we compare the current state of pinA with pinB. A difference between these states signifies a clockwise rotation, while identical states indicate a counter-clockwise rotation. Depending on the direction:

Clockwise: We increment certain values using a designated function.

Counter-Clockwise: We decrement the values using another specified function.

After adjusting these values, we update the timeConstant and call the updateDisplay function to reflect these changes on the display.

```
void incrementValue() {
  switch (selectedParameter) {
    case 0: gain += 0.1; break;
    case 1: timeConstantms += 10; break;
    case 2:
    if (lagSample < MAX_BUFFER_SIZE) lagSample += 1;
    break;
  }
}
```

The incrementValue function determines the current selectedParameter and accordingly increments its value based on predefined steps. If gain is the selected parameter, it increases by 0.1. If timeConstantms is selected, it increments by 10 milliseconds. For lagSample, it increases by 1, but only if the new value does not exceed the maximum size of the buffer. This ensures that each parameter is adjusted appropriately within its operational limits.

```
void decrementValue() {
  switch (selectedParameter) {
    case 0:
      if (gain > 0) gain -= 0.1;
      break;
    case 1:
      if (timeConstantms > 0) timeConstantms -= 10;
      break;
    case 2:
      if (lagSample > 0) lagSample -= 1;
      break;
  }
}
```

The decrementValue function assesses the current selectedParameter and appropriately reduces its value based on specific criteria. If the gain is selected, it is decreased by 0.1, provided that it is greater than 0, since a negative gain is not permissible. If the timeConstantms is selected, it is reduced by 10 milliseconds, but only if it is also above 0 to avoid negative time values. For the lagSample, it is decremented by 1, again only if it is greater than 0 to maintain valid sample indices. This function ensures that parameter adjustments do not result in invalid settings.

```
void handleButton() {
```

```
  if (digitalRead(buttonPin) == LOW) {
    selectedParameter = (selectedParameter + 1) % 3;
    delay(250);
    updateDisplay();
  }
}
```

The handleButton function is used to cycle through different parameters when the button is pressed. Initially, it reads the signal from the buttonPin. If the signal is LOW, this indicates that the button has been pressed. Upon recognizing a press, the function proceeds to a conditional segment where it alters the selectedParameter, allowing the user to adjust a different parameter next. To manage button press sensitivity, a delay is implemented while the button is held down. Finally, the updateDisplay function is called to update the display, reflecting the change in the selected parameter.

```
void updateDisplay() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Gain: ");
  lcd.print(gain, 2);
  lcd.setCursor(0, 1);
  lcd.print("TimeConst(ms): ");
  lcd.print(timeConstantms);
  lcd.setCursor(0, 2);
  lcd.print("Lag(sample): ");
  lcd.print(lagSample);
  lcd.setCursor(0, 3);
  lcd.print("Lag(s): ");
  lcd.print(float(lagSample*sampleTime));
}
```

The updateDisplay function begins by clearing any previous content displayed. Then, it proceeds with the following updates: on row 0, it displays "Gain: " followed by the value of the gain. On row 1, it shows "TimeConst(ms): " along with the value of the timeConstantms. Row 2 displays "Lag(sample): " and the value of the lagSample. Finally, on row 3, it prints "Lag(s): " and then the calculated value of the lag in seconds, which is the product of the lagSample and the sampleTime. This setup ensures that all relevant parameters are visibly updated on the display. Results and discussion

5 Results and Discussion

This section provides a detailed review and assessment of the results achieved in this thesis. The project was centred around the primary goal of developing an Arduino-based simulation tool for control systems education at NOVIA University. The outcomes have been elaborated and demonstrated throughout Section 4.

The outcomes of this thesis involve the integration of both hardware assembly and software development. Assuming that essential tools (such as screwdrivers, wires, and a soldering iron) are readily available and an Arduino license is obtained at no cost, the projected expense for this project is roughly €85, inclusive of VAT. This figure represents an estimate, calculated by considering each component as an individual purchase, though NOVIA University typically acquires multiple components collectively. The selection of components was guided by considerations of performance and availability. Detailed descriptions of the components used and their configurations can be found in Section 3.

If this project were to be replicated, the estimated time for configuring and assembling the Arduino-based simulation tool would be approximately 3 hours. The Arduino IDE and the user interface should be capable of adaptation with no significant issues beyond configuring the connection settings in the software. In cases where the code requires modification, the time cannot be precisely estimated, as the extent and complexity of the changes would significantly influence the duration.

Since the development of the original Arduino-based simulation tool for control systems at NOVIA University, Arduino has continued to release new hardware and software updates.

The advantage of using an officially supported platform includes access to comprehensive documentation and community support, which can facilitate the integration of advanced features and ensure more reliable configurations during setup, thereby improving the quality and accuracy of the simulations.

Although newer Arduino models offer upgraded features, the core functionality in terms of hardware and programmability remains largely similar. The primary benefit of the custom code and interface developed in this thesis is their tailored design to meet specific educational objectives set by the instructors at NOVIA University. The customized solution

ensures that the simulation tool directly addresses the pedagogical needs of the courses it is intended to support.

Initial aims and objectives

The primary goal of this project was to develop an Arduino-based simulation tool that bridges the educational gap in control systems at NOVIA University by providing students with hands-on, practical experience. Although the objectives were ambitious, the main aim has largely been met, despite some objectives not being fully achievable.

The first objective was to develop diverse and configurable simulation models that allow students to explore various control systems using Arduino hardware. This objecctive has been successfully completed. The flexibility in selecting different parameters enables the creation of new models tailored to the instructor's preferences.

The second and final objectives focused on creating an intuitive and user-friendly interface to simplify the selection and modification of control models for educational purposes and constructing a dynamic environment where users can adjust parameters such as time constants and delays. These goals have been achieved: the interface is straightforward, featuring an LCD that displays parameter values continuously, and an encoder that allows users to adjust values by turning it clockwise to increase or counterclockwise to decrease. Switching between editing different parameters is as simple as pressing the encoder.

The third objective was to establish a robust connection framework for seamless integration with Programmable Logic Controllers (PLCs). This was accomplished by matching the output and input wire colours with those used in PLCs and using compatible connection types to ensure straightforward integration.

Limitations

During the implementation phase, several technical challenges emerged. A primary issue was my limited proficiency in the programming language, which restricted some of the implementation to less-than-ideal approaches. Moreover, the ambition to establish a password-protected configuration to shield system settings from students could not be fulfilled due to these programming limitations.

The most significant technical constraint involved the voltage range of our Arduino setup. Although the Arduino platform can theoretically handle outputs from 0-5 volts, the actual output was capped at 0-2.25V to avoid internal damage. This limitation was further compounded by the Arduino's internal tolerance issues, which posed challenges in achieving precise conversions from a 10-bit range to the actual voltage output.

While these limitations did not critically impact the core objectives of this thesis, they could restrict the broader application and future utility of the developed program.

Consistencies and inconsistencies

Throughout the development of this thesis, the project's structure and proposed solutions underwent significant changes due to various misunderstandings, leading to some inconsistencies in the documentation. Initially, I was under the impression that I needed to develop both the PID controller and the plant, which is explained the extensive detail in Section 2. This misunderstanding was eventually clarified when I ordered materials from the laboratory and discussed the project scope with the laboratory staff.

Additionally, there was a challenge regarding the methodological approach to solving the problem. We initially agreed that the equation could be solved over time. However, it was not until the final week, when I discussed the approach with my tutor, that I realized this method was unfeasible for achieving the project's objectives. This realization led to a disagreement with my tutor about the necessity of using the Z-transform for working with a microprocessor, as it operates in discrete time rather than continuous time.

Improvement suggestions

Regarding the project, although the initial goals have been met, there is room for improvement to address the limitations encountered. One potential enhancement would involve sourcing an Arduino model capable of handling a wider voltage range, specifically 0-5 volts or 0-10 volts. Additionally, implementing a password-protection feature to secure the configuration settings would ensure that students cannot modify parameters during laboratory sessions.

Another useful improvement would be the ability to save certain models so that they can be readily accessed and deployed without the need to adjust each parameter individually.

This enhancement would streamline laboratory practices and increase efficiency during experiments.

6 Conclusion

The primary objective of this thesis was to develop an Arduino-based simulation tool for control systems education at NOVIA University, aiming to bridge the educational gap in this field and equip students with practical, hands-on experience. As the project concluded, it became evident that not only was this goal largely achieved, but the results also highlighted significant potential for future educational enhancements.

The project was undertaken with a keen focus on integrating hardware and software to facilitate a comprehensive learning experience. This integration was successfully realized through the development of a dynamic, user-friendly interface and configurable simulation models. The interface allows teachers to interact with the system efficiently, adjusting parameters such as gain, time constants and delays, which enriches the students learning experience by simulating various control system scenarios.

Financially, the project was completed on a modest budget of approximately €85, which includes VAT. This cost-efficiency is particularly notable considering the educational benefits provided by the tool. The expenditure was kept low by utilising readily available Arduino hardware and free software, which underscores the project's feasibility and replicability in similar educational settings.

Despite these successes, the project encountered some technical limitations that impacted its full potential. The most significant was the hardware's inability to achieve the intended output signal range of 0-10V or 0-5V, settling instead for a maximum of 2.25V due to Arduino's inherent limitations. Additionally, the initial aim to implement a password-protected configuration to safeguard system settings from unauthorized student modifications was not realized. This was primarily due to my limited programming experience, which underscored a critical area for further development and education.

Looking to the future, several enhancements could be made to expand the tool's capabilities and educational value. Upgrading to a more capable Arduino model could address the current voltage range limitation, thereby broadening the range of control scenarios that can be simulated. Furthermore, enhancing the software to include password protection would not only secure the system against unintended alterations but also add an extra layer of professionalism to the educational environment.

In conclusion, the Arduino-based simulation tool developed through this thesis represents a valuable educational asset for NOVIA University. It stands as a testament to the potential of integrating practical simulation tools in an academic setting, providing a solid foundation for students to understand and apply control systems theory. The experience gained from this project, coupled with the documented successes and limitations, provides a robust framework for future projects aimed at further enhancing control systems education. The recommendations for improvement suggest a pathway not only to refine this tool but also to innovate new solutions that could continue to evolve the educational landscape in engineering.

7 References

Cloud Editor (2024). Retrieved May 1, 2024, from Arduino.cc website:

https://docs.arduino.cc/arduino-cloud/guides/editor/

Control Systems - Mathematical Models. (2024). Retrieved April 21, 2024, from Tutorialspoint.com website:

https://www.tutorialspoint.com/control_systems/control_systems_mathematical_models.htm

Dejan. (2016, July 25). How Rotary Encoder Works and How To Use It with Arduino. Retrieved May 1, 2024, from How To Mechatronics website:

https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/?utm_content=cmp-true

DMM. (2023, January 4). Control System Definition, Types, Applications, and FAQs. Retrieved March 21, 2024, from Electronics For You website:

https://www.electronicsforu.com/technology-trends/learn-electronics/control-system-definition-types-applications-and-faqs

Electrical4U. (2020, December 27). What is a Control System? (Open Loop & Closed Loop Control Systems Explained). Retrieved March 21, 2024, from Electrical 4U website:

https://www.electrical4u.com/control-system-closed-loop-open-loop-control-system/?utm_content=cmp-true

Electrical4U. (2023, June 25). Control Engineering: What is it? (And its History) | Electrical4U. Retrieved March 21, 2024, from Electrical4U website: https://www.electrical4u.com/control-engineering-historical-review-and-types-of-control-engineering/?utm_content=cmp-true#google_vignette

GfG. (2023a, December 25). Classification of Control Systems. Retrieved March 21, 2024, from GeeksforGeeks website:

https://www.geeksforgeeks.org/classification-of-control-systems/

GfG. (2023b, October 19). Control Systems Controllers. Retrieved March 26, 2024, from GeeksforGeeks website: https://www.geeksforgeeks.org/control-systems-controllers/

GfG. (2024, February 8). Continuous Time and Discrete Time Control Systems. Retrieved March 21, 2024, from GeeksforGeeks website:

https://www.geeksforgeeks.org/continuous-time-and-discrete-time-control-systems/

What is Arduino? (2022). Retrieved May 1, 2024, from Arduino.cc website: https://docs.arduino.cc/learn/starting-guide/whats-arduino/

History | IEEE Control Systems Society. (2020). Retrieved March 21, 2024, from Ieeecss.org website: https://ieeecss.org/history

Jain D.V. (2021, June 27). Types of Controllers | Proportional Integral and Derivative Controllers | Electrical4U. Retrieved March 26, 2024, from Electrical4U website: https://www.electrical4u.com/types-of-controllers-proportional-integral-derivative-controllers/

Libretexts. (2020, May 19). 9.2: P, I, D, PI, PD, and PID control. Retrieved March 26, 2024, from Engineering LibreTexts website:

https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_%28Woolf%29/09%3A_Proportional-Integral-Derivative_%28PID%29_Control/9.02%3A_P%2C_I%2C_D%2C_PI%2C_PD%2C_and_PID_control

MATHEMATICAL MODELING OF DYNAMIC SYSTEMS 3. (n.d.). Retrieved from:

https://by.genie.uottawa.ca/~necsules/MCG_3306/MCG%203306%20%20pdf/A3.pdf

Liquid Crystal Display with Arduino (2023). Retrieved May 9, 2024, from Arduino.cc website: https://docs.arduino.cc/learn/electronics/lcd-displays/

Roshni Y. (2019, December 28). What are Controllers? Types of Controllers - Electronics Coach. Retrieved March 26, 2024, from Electronics Coach website:

https://electronicscoach.com/types-of-controllers.html

The Engineering Concepts. (2021, January 6). Types Of Control System - The Engineering Concepts. Retrieved March 21, 2024, from The Engineering Concepts website: https://www.theengineeringconcepts.com/types-of-control-system/

The instrument guru. (2020, December 15). Process control. Retrieved March 26, 2024, from THE INSTRUMENT GURU website: https://theinstrumentguru.com/process-control/

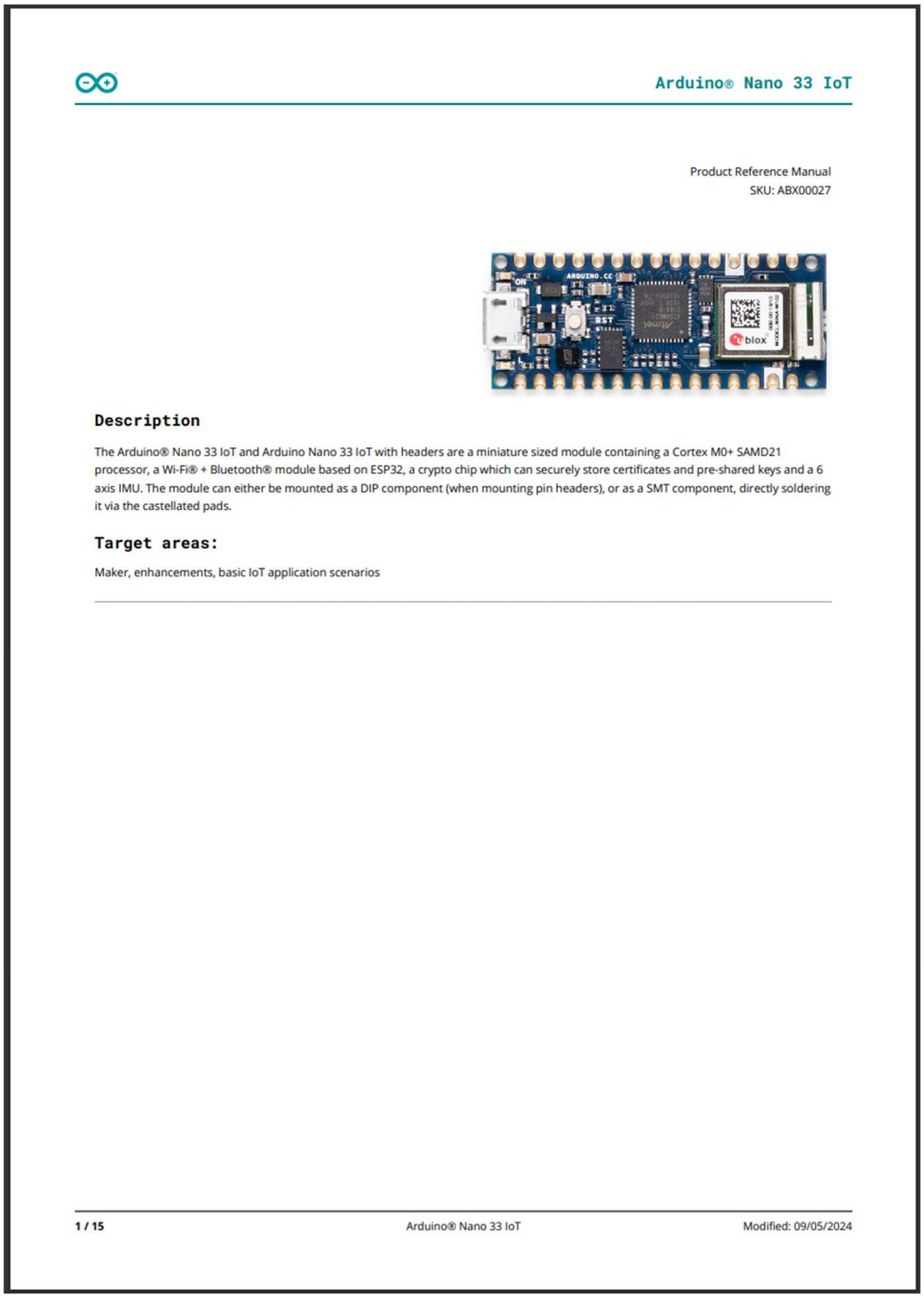
8 Appendices

8.1 Appendix 1: Arduino Nano 33 IoT datasheet

The datasheet of Arduino Nano 33 IoT can be found at the following link:

https://docs.arduino.cc/resources/datasheets/ABX00027-datasheet.pdf

Document size 1,493 KB (15 pages)



Arduino® Nano 33 IoT

Product Reference Manual
SKU: ABX00027

Description

The Arduino® Nano 33 IoT and Arduino Nano 33 IoT with headers are a miniature sized module containing a Cortex M0+ SAMD21 processor, a Wi-Fi® + Bluetooth® module based on ESP32, a crypto chip which can securely store certificates and pre-shared keys and a 6 axis IMU. The module can either be mounted as a DIP component (when mounting pin headers), or as a SMT component, directly soldering it via the castellated pads.

Target areas:

Maker, enhancements, basic IoT application scenarios

1 / 15 Arduino® Nano 33 IoT Modified: 09/05/2024

8.2 Appendix 2: Projet code in C

Here it is code of the project that is in C language:

```
#include <LiquidCrystal.h>

// Initialize the LCD display with the pins it's connected to on the Arduino
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// Define pin numbers for the rotary encoder's dial and button
const int pinA = 6;
const int pinB = 7;
const int buttonPin = 8;

// Define pin for DAC output. A0 is used for analogue output on Arduino Nano
33 IoT
const int dacPin = A0;
const int analogInputPin = A1; // Pin for reading analog input signals

// Define maximum buffer size for storing past samples, calculated based on
maximum expected lag
#define MAX_BUFFER_SIZE 34

// System parameters
float gain = 1.0;  // Initial gain factor
int timeConstantms = 1000; // Time constant in milliseconds
int lagSample = 20; // Lag in terms of the number of samples

// Convert time constant from milliseconds to seconds for processing
float timeConstant = timeConstantms / 1000.0;

// Define the sampling time in seconds
float sampleTime = 0.15; // 150 ms

// Array to store past input samples, serving as a circular buffer
float inputBuffer[MAX_BUFFER_SIZE];

// Index for the current position in the circular buffer
int bufferIndex = 0;

// Variable to store the previous output, required for recursive computation
float y_prev = 0;

// Variables to handle encoder input
int lastStateA;  // Store last state of pinA to detect changes
int selectedParameter = 0;  // Current parameter being adjusted by the
encoder

// Setup function initializes the Arduino's hardware configuration
void setup() {
```

```
  pinMode(dacPin, OUTPUT); // Set the DAC pin as an output
  pinMode(analogInputPin, INPUT); // Set the analog input pin
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  Serial.begin(9600); // Start serial communication for debugging
  lcd.begin(20, 4); // Start the LCD
  analogWriteResolution(10);
  //analogReadResolution(10);
  memset(inputBuffer, 0, sizeof(inputBuffer)); // Initialize buffer with
zeros
  lastStateA = digitalRead(pinA); // Initialize lastStateA with current
state
  updateDisplay();  // Update LCD display
}

// Main program loop
void loop() {
  static unsigned long lastUpdateTime = 0;  // Track the last update time
  if (millis() - lastUpdateTime > sampleTime * 1000) {  // Check if it's
time to sample
    lastUpdateTime = millis(); // Update last update time

    // Read the current input and scale it to a voltage value
    float x_curr = float(analogRead(A1)) *3.5 / 1023;
    inputBuffer[bufferIndex] = x_curr;  // Store current input in buffer

    // Compute indices for x[n-L] and x[n-L-1] accessing lagged samples in
the buffer
    int indexL = (bufferIndex - lagSample + MAX_BUFFER_SIZE) %
MAX_BUFFER_SIZE;
    int indexL1 = (indexL - 1 + MAX_BUFFER_SIZE) % MAX_BUFFER_SIZE;

    // Compute the current output using a discrete filter equation
    float y_curr = calculateDiscreteOutput(inputBuffer[indexL],
inputBuffer[indexL1]);
    if (y_curr > 2.25) {  // Clamp the output to a maximum of 2.25 volts
      y_curr = 2.25;
    }
    int y_out = (y_curr / 3.3) * 1023;  // Convert output voltage to DAC
value
    analogWrite(A0, y_out);  // Send output to DAC

    Serial.print("input: "); Serial.println(x_curr);
    Serial.print("output: "); Serial.println(y_curr);

    bufferIndex = (bufferIndex + 1) % MAX_BUFFER_SIZE;  // Move to the next
buffer index

  }
```

```
  handleEncoder(); // Process encoder inputs
  handleButton(); // Process button press
}

// Calculate the output of the filter based on previous output and lagged
inputs
float calculateDiscreteOutput(float x_L, float x_L1) {
  float a = (2 * timeConstant - sampleTime) / (2 * timeConstant +
sampleTime);
  float b = (gain * sampleTime) / (2 * timeConstant + sampleTime);

  // Calculate the current output
  float y_curr = a * y_prev + b * x_L + b * x_L1;

  // Update the previous output for the next cycle
  y_prev = y_curr;

  return y_curr;
}

// Process encoder input to adjust parameters
void handleEncoder() {
  int stateA = digitalRead(pinA); // Read the current state of pinA
  if (stateA != lastStateA) { // If the state has changed, a rotation has
occurred
    if (digitalRead(pinB) != stateA) { // The direction of rotation depends
on the states of pinA and pinB
      incrementValue();  // Clockwise rotation
    } else {
      decrementValue(); // Counter-clockwise rotation
    }
    timeConstant = timeConstantms / 1000.0;
    updateDisplay();  // Update the display with new parameter selection
  }
}

// Increment the currently selected parameter
void incrementValue() {
  switch (selectedParameter) {
    case 0: gain += 0.1; break; // Increment gain by 0.1
    case 1: timeConstantms += 10; break; // Increment timeConstantms by 10
    case 2:
    if (lagSample < MAX_BUFFER_SIZE) lagSample += 1;  // Only increment if
below maximum
    break;
  }
}

// Decrement the currently selected parameter
void decrementValue() {
  switch (selectedParameter) {
```

```
    case 0:
      if (gain > 0) gain -= 0.1; // Decrement gain by 0.1 if greater than 0
      break;
    case 1:
      if (timeConstantms > 0) timeConstantms -= 10; // Decrement
timeConstantms by 10 if greater than 0
      break;
    case 2:
      if (lagSample > 0) lagSample -= 1; // Decrement lagSample by 10 if
greater than 0
      break;
  }
}

// Process button presses to switch between parameters
void handleButton() {
  if (digitalRead(buttonPin) == LOW) {
    selectedParameter = (selectedParameter + 1) % 3;
    delay(250);  // Debounce delay
    updateDisplay();  // Update the display with new parameter selection
  }
}

// Update the LCD display with the current system parameters
void updateDisplay() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Gain: ");
  lcd.print(gain, 2);
  lcd.setCursor(0, 1);
  lcd.print("TimeConst(ms): ");
  lcd.print(timeConstantms);
  lcd.setCursor(0, 2);
  lcd.print("Lag(sample): ");
  lcd.print(lagSample);
  lcd.setCursor(0, 3);
  lcd.print("Lag(s): ");
  lcd.print(float(lagSample*sampleTime));
}
```