Hochschule Bremerhaven

MasterofScienceinEmbeddedSystems Design

EmbeddedSystemProjectReporton

CartPendulumControl

UndertheGuidanceof

Prof. Dr.-Ing. Kai Muller

Prof.Dr.-Ing.KarstenPeter

Submitted by

Bansi Thummar(40267)

Hardik Dudhat (40310)

Ishabahen Polara (40273)

Parth Savaliya (40278)

Ritik Mungalpara (40274)

Rutik Lunagariya (40245)

Submittedon

29.02.2024

Acknowledgements

We extend our heartfelt gratitude to our respected professors and colleagues who provided invaluable assistance and support throughout the completion of our academic project, Cart Pendulum Control. Special thanks are due to our project guides, Prof. Dr.-Ing Karsten Peter, and Prof. Dr.-Ing Kai Mueller, whose guidance and suggestions were instrumental in shaping our project and stimulating our group's intellectual pursuits. We also acknowledge the staff of Embedded Systems Design at Hochschule Bremerhaven for their generous support and provision of necessary equipment, without which this project would not have been possible. Lastly, we thank our colleagues for their unwavering support, camaraderie, and collaboration throughout this challenging yet rewarding journey.

Work Organization

Sr no	Team members	Work Contribution
1	Bansi Thummar (40267)	Ch: 8, 9, 10
2	Hardik Dudhat (40310)	Ch: 14, 15, 16, 17
3	Ishabahen Polara (40273)	Ch: 1, 2, 3
4	Parth Savaliya (40278)	Ch: 11, 12, 13
5	Ritik Mungalpara (40274)	Ch: 4, 5, 6.1, 6.2
6	Rutik Lunagariya (40245)	Ch: 6.3, 6.4, 7

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **2**

LIST OF FIGURES **7**

LIST OF TABLES **10**

ABSTRACT **11**

1 INTRODUCTION **12**

2 PROJECT LIFE CYCLE AND REQUIREMENTS ANALYSIS INTEGRATION **14**

3 MATHEMATICAL MODELLING OF THE CART PENDULUM SYSTEM **16**

3.1 DERIVATION OF DIFFERENTIAL EQUATION OF THE SYSTEM 16
3.2 NON-LINEAR EQUATIONS 17
3.3 LINEAR EQUATIONS 19

4 STATE SPACE REPRESENTATION OF LINEAR MODEL **21**

5 MATLAB MODELLING AND SIMULATION OF CART PENDULUM SYSTEM **23**

5.1 NONLINEAR MODEL 23
5.1.1 MATLAB code 23
5.1.2 Simulink Model 23
5.1.3 Output Graphs 25
5.2 LINEAR MODEL 27
5.2.1 Matlab Code 27
5.2.2 Simulink Model 27

6 CART POSITION CONTROLLER **30**

6.1 STATE FEEDBACK SYSTEM 30
6.2 PROPORTIONAL CONTROLLER 31
6.3 TIME CONTINUOUS PI CONTROLLER 34
6.3.1 MATLAB script for calculation of PI controller 35
6.3.2 Simulink Model 36
6.3.3 Output Graphs 37
6.3.4 Disturbance Effect 41
6.4 TIME DISCRETE PI CONTROLLER 46

7 CART SPEED CONTROLLER **53**

7.1 MATHEMATICAL MODELLING OF CART SPEED CONTROLLER 53
7.2 TIME CONTINUOUS PI CONTROLLER 55
7.2.1 MATLAB Code 55
7.2.2 Simulink Model 55
7.2.3 Output graph 56
7.3 TIME DISCRETE PI CONTROLLER 57
7.3.1 Simulink Model 57
7.3.2 Output graph 57

8 HARDWARE COMPONENTS 58

8.1 INCREMENTAL ENCODER 59
8.2 VIOLIN AMPLIFIER 60
8.3 LINEAR ACTUATOR LEZ3 60
8.4 DC SERVO MOTOR 61
8.5 END SWITCHES 62
8.6 ARTY Z7 BOARD 62

9 HARDWARE INTERFACE BOARD 65

9.1 DIGITAL TO ANALOG CONVERSION (DAC) 66
9.2 ANALOG-TO-DIGITAL CONVERTOR (ADC) 67
9.3 UP & DOWN COUNTER 67
9.3.1 Up Counting 67
9.3.2 Down Counting 67
9.4 SPI PROTOCOL 68

10 CART PENDULUM SOC DESIGN 70

10.1 CART PENDULUM CUSTOM IP COMPONENT 70
10.2 SPI VHDL MODULE 71
10.2.1 Sequential process VHDL code for spi pha='0' 72
10.2.2 Start/Stop and Clock Handling 73

11 SCALING FOR THE POSITION AND SPEED CONTROLLER 75

11.1 INCREMENTAL ENCODER 75
11.2 LEZ3 ACTUATOR 76
11.3 PHYSICAL SCALING 78
11.3.1 Physical Scaling for Cart Position Controller: 79
11.3.2 Fixed Point Position Controller: 82
11.3.3 Physical Scaling for Cart Speed Controller: 85
11.3.4 Fixed Point Speed Controller: 88

12 S-FUNCTION DEVELOPMENT 90

12.1 C CODE TEST FOR CART POSITION CONTROLLER 90
12.2 C CODE TEST FOR CART SPEED CONTROLLER 95

13 IMPLEMENTATION OF C CODE 99

13.1 REFERENCING PHASE FOR CART SPEED CONTROLLER 99
13.1.1 State Diagram 99
13.1.2 Code Explanation for Referencing Phase 100
13.2 CART POSITION CONTROLLER 105
13.2.1 State Diagram 105
13.2.2 Code Explanation for Controlling Phase 106

14 C-CODE FUNCTIONS FOR GUI 108

14.1 REFERENCE POSITION CODE 108
14.2 START CONTROLLING CODE 109

14.3 STOP CONTROLLING CODE 110
14.4 DATA RECORDING ON/OFF CODE 110

15 GUI FOR CART PENDULUM SYSTEM 112

15.1 GRAPHICAL USER INTERFACE 112
15.2 DEVICE CONNECTION (HARDWARE) 113
15.3 CONTROL ACTIONS BUTTON 113
15.4 DATA DISPLAY BOX 114
15.5 DATA RECORDER BUTTON 114
15.6 GRAPH WINDOW 114
15.7 COMMAND PANEL 115

16 MAIN GUI CODE EXPLANATION 117

16.1 HARDWARE CONNECTION CODE 117
16.2 CONTROL ACTIONS BUTTON CODE 119
16.3 DATA DISPLAY BOX CODE 120
16.4 DATA RECORDER BUTTON CODE 123
16.5 GRAPH WINDOW CODE 125
16.6 DOWNLOAD RECORD DATA FILE CODE 129
16.7 CHECK BOX 130

17 TEST RESULTS ANALYSIS 133

17.1 ACTIVE CONTROL 133
17.1.1 Pendulum Angle 133
17.1.2 Pendulum Velocity 134
17.1.3 Cart Displacement 135
17.1.4 Cart Velocity 136
17.1.5 Force 137
17.2 RECORDED DATA FILE 137

CONCLUSION 142

REFERENCES 143

List of Figures

Figure 1: Typical Overhead Crane [24] 12
Figure 2: Test Stand 13
Figure 3: V-Model for porject[25] 14
Figure 4: Cart Pendulum System [14] 16
Figure 5: Pendulum Co-ordinates 17
Figure 6: Nonlinear subsystem model of cart pendulum system 23
Figure 7: Detailed nonlinear model of cart pendulum system 24
Figure 8: Angular Displacement of Pendulum (Non-Linear Model) 25
Figure 9: Angular velocity of Pendulum (Non-Linear Model) 25
Figure 10: Velocity of Cart (Non Linear Model) 26
Figure 11: Displacement of Cart (Non Linear Model) 26
Figure 12: State space representation of cart pendulum system 27
Figure 13: Angular Velocity of Pendulum (Linear Model) 28
Figure 14: Angular Displacement of Pendulum (Linear Model) 28
Figure 15: Velocity of Cart (Linear Model) 29
Figure 16: Displacement of cart (Linear Model) 29
Figure 17: Proportional controller 32
Figure 18: cart displacement of Proportional controller (linear) 33
Figure 19: Continuous PI controller 36
Figure 20: Linear angular velocity of pendulum (Continuous PI) 37
Figure 21: Linear angular displacement of pendulum (Continuous PI) 37
Figure 22: Linear cart velocity (Continuous PI) 38
Figure 23: Linear cart displacement (Continuous PI) 38
Figure 24: Nonlinear angular velocity of pendulum (Continuous PI) 39
Figure 25: Nonlinear angular displacement of pendulum (Continuous PI) 39
Figure 26: Nonlinear cart velocity (Continuous PI) 40
Figure 27: Nonlinear cart displacement (Continuous PI) 40
Figure 28: Continuous PI controller with disturbance effect 41
Figure 29: Disturbance effect on pendulum angular velocity (Linear continuous PI) 42
Figure 30: Disturbance effect on pendulum angle (Linear continuous PI) 42
Figure 31: Disturbance effect on cart Velocity (Linear continuous PI) 43
Figure 32: Disturbance effect on cart displacement (linear continuous PI) 43
Figure 33: Disturbance effect on pendulum angular velocity (nonlinear continuous PI) 44
Figure 34: Disturbance effect on pendulum angle (Linear continuous PI) 44
Figure 35: Disturbance effect on pendulum angle (nonlinear continuous PI) 45
Figure 36: Disturbance effect on cart displacement (Nonlinear continuous PI) 45
Figure 37: Discrete PI controller 48
Figure 38: Linear angular velocity of pendulum with disturbance (Discrete PI) 49
Figure 39: Linear angular displacement of pendulum with disturbance (Discrete PI) 49
Figure 40: linear cart velocity with Disturbance (Discrete PI) 50
Figure 41: Linear cart displacement with Disturbance (Discrete PI) 50
Figure 42: Nonlinear angular velocity of pendulumwith Disturbance (Discrete PI) 51

Figure 43: Nonlinear angular displacement of pendulum with Disturbance (Discrete PI) 51
Figure 44: Nonlinear cart velocity with Disturbance (Discrete PI) 52
Figure 45: Nonlinear cart displacement with Disturbance (Discrete PI) 52
Figure 46: Cart velocity (Continuous cart speed controller) 56
Figure 47: Cart velocity (Discrete cart speed controller) 57
Figure 48: Hardware setup of Cart Pendulum System [10] 58
Figure 49: Hardware components and their interface 59
Figure 50: Incremental Encoder [2] 59
Figure 51: Output signals of the incremental encoder [2] 60
Figure 52: LEZ3 linear actuator [3] 60
Figure 53: End Switches [5] 62
Figure 54: Arty Z7 board [6] 63
Figure 55: The interface of the interface board and the components 65
Figure 56: DAC Input Shift Register [7] 66
Figure 57: Up & Down Counter 68
Figure 58: SPI master and slave interface [9] 68
Figure 59: DAC SPI write cycle [7] 69
Figure 60: Conversion Start and Data Read Timing Diagram of ADC [8] 69
Figure 61: Cart Pendulum SoC Design 70
Figure 62: IP Block diagram 71
Figure 63: IP port and interface 71
Figure 64: VHDL code for spi pha='0' 72
Figure 65: State diagram for spi pha='0' 72
Figure 66: SPI start and stop logic state machine 73
Figure 67: Experiment to derive relationship between DAC values and force(N) 77
Figure 68: Relationship between DAC values and force(N) 77
Figure 69: Physically scaled cart position controller 79
Figure 70: Physically scaled position controller 81
Figure 71: Cart velocity and Cart position graph after physically scaled 81
Figure 72: Pendulum angle and angular velocity after physically scaled 82
Figure 73: Fixed point position controller 83
Figure 74: Required force to the cart for fixed point model 84
Figure 75: Cart position for fixed point model 84
Figure 76: Pendulum angle for fixed point model 85
Figure 77: Physically scaled cart speed controller 86
Figure 78: Physically scaled speed controller 87
Figure 79: Cart velocity graph after physically scaled 88
Figure 80: Fixed point speed controller 89
Figure 81: Cart velocity for fixed point model 89
Figure 82: Simulink model and C code legacy block comparison for position controller 93
Figure 83: Cart position comparison between legacy block and simulink model 94
Figure 84: Pendulum angle comparison between legacy block and Simulink model 94
Figure 85: Simulink model and C code legacy block comparison for speed controller 97
Figure 86: Cart velocity comparison between legacy block and simulink model 98

Figure 87: Block diagram for referencing phase 100
Figure 88: Block diagram for controlling phase 105
Figure 89: Graphical user interface 112
Figure 90: COM port selection and connect and disconnect radio buttons 113
Figure 91: Control action button and Reference position display 113
Figure 92: Data display box 114
Figure 93: Data recorder button 114
Figure 94: Graph window 115
Figure 95: Command panel 115
Figure 96: Command line 116
Figure 97: Pendulum angle graph 133
Figure 98: Pendulum velocity graph 134
Figure 99: Cart displacement graph 135
Figure 100: Cart velocity graph 136
Figure 101: Force graph 137

List of Tables

Table 1: Parameter of Motor MV 300 [4] 61
Table 2: Command Definition of the DAC [7] 66
Table 3: DAC Input bits 66
Table 4: Scaling variables for control of cart pendulum system 78
Table 5: All Model gains and their calculated scaled gain 80
Table 6: All scaled gains and their selected fixed point formats 82
Table 7: Proportional and Integral gains for model and their calculated scaled gain 87
Table 8: Proportional and Integral gains and their selected fixed-point formats 88

Abstract

The primary objective of this project was to develop a Cart Pendulum System capable of controlling the cart's position while damping the pendulum's oscillation. This involved mathematically describing the system and designing a nonlinear model. To enable the application of various control algorithms, the system was linearized at a specific operating point. Controllers, including PI and state-feedback, were devised for both position and sway control of the cart pendulum system. Validation of the designed controller was performed using MATLAB/Simulink with the system model. Additionally, the same control algorithm was implemented on a Zynq board in C language and connected to the test stand to evaluate real-world performance. System states, such as cart displacement and angular displacement of the pendulum, were tracked using incremental rotary encoders. Furthermore, a graphical user interface (GUI) was developed to facilitate user interaction and display essential data such as cart displacement, cart velocity, angular displacement, and angular velocity of the pendulum.

1 Introduction

In modern industrial operations, the efficient loading and unloading of cargo, weighing millions of tonnes, is facilitated by a variety of cranes deployed across ships, trucks, and industrial facilities. Given that these loads are suspended by flexible ropes, precision in their movement is imperative to minimize swinging during transportation. The oscillation of loads poses significant challenges and risks to crane operators during loading and unloading operations, necessitating robust control strategies to ensure safety. One commonly encountered crane type in industrial settings is the overhead crane, often referred to as a bridge crane. This type of crane features parallel runways with a traveling bridge spanning the gap between them. A cargo is typically attached to a hoist located at the bottom of the bridge, enabling vertical movement along the bridge's length. However, due to the nature of the load attachment, achieving precise positioning with minimal swing is inherently challenging.

Figure 1: Typical Overhead Crane [24]

To address this challenge, our project focuses on the development of a control strategy to mitigate the oscillation of loads while in motion. We establish a test stand resembling an overhead crane, comprising a cart and a pendulum attached to it. Here, the cart represents the crane, and the pendulum's swing mirrors the oscillation effect observed in crane operations. The primary objective of our project is to devise a control method that effectively dampens the pendulum's oscillation as the cart moves.

Our project setup, illustrated in Figure 2, features a pendulum suspended from the cart, which is propelled by a motor via a belt mechanism. The motor serves to supply the external force required for cart movement. Crucially, the pendulum is affixed to the cart's center, allowing it to move freely within

the cart's plane of motion without encountering any restraining forces. To facilitate control, the system incorporates two incremental encoders—one attached to the belt to provide incremental cart position data, and the other affixed to the cart to furnish angular incremental information about the pendulum. Additionally, limit switches positioned at either end of the cart's travel path serve to detect the endpoints.

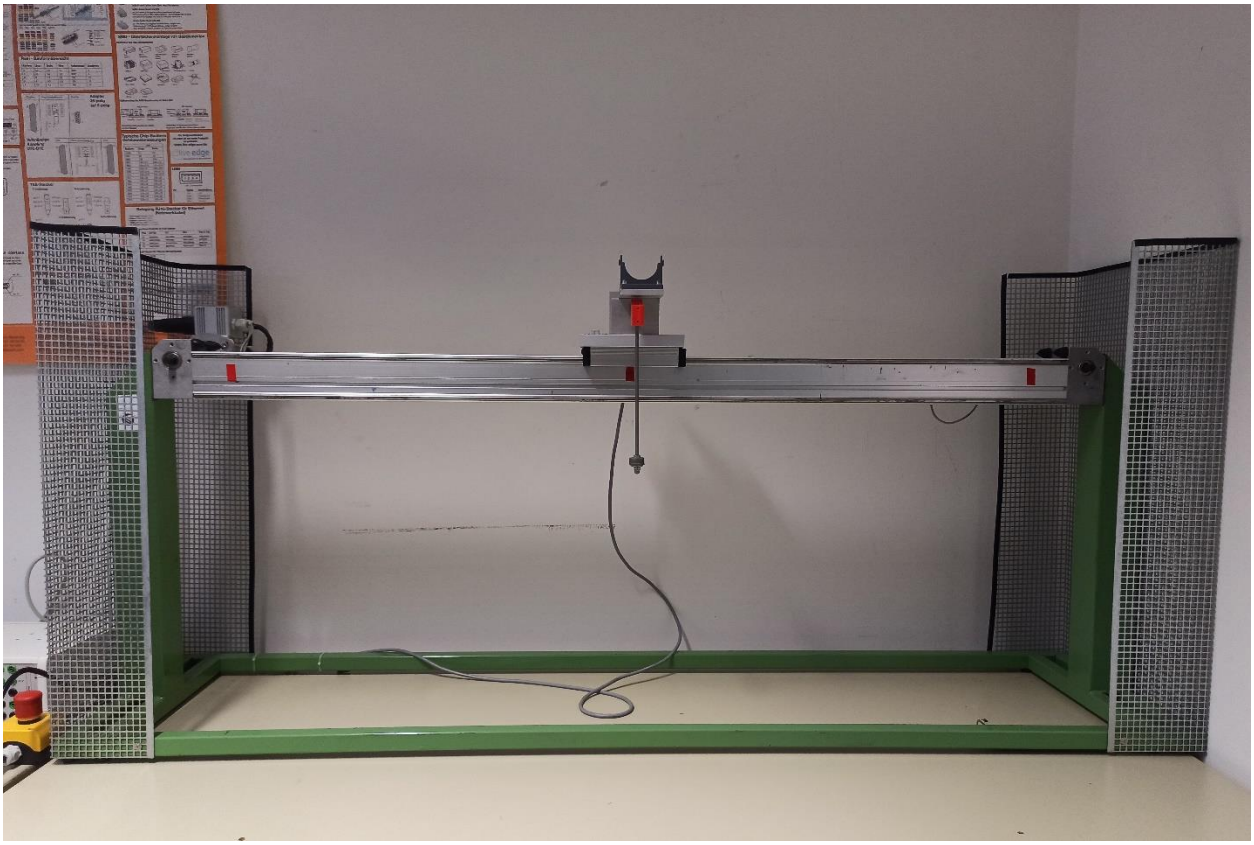


Figure 2: Test Stand

Throughout this report, we will delve into the mathematical design of the cart-pendulum system, encompassing both nonlinear and linear models. We will detail the development of a PI controller, the hardware design, and the implementation of the control algorithm on the Zynq board. Furthermore, we will discuss the creation of a user-friendly GUI for system interaction, alongside the visualization of pertinent data such as cart displacement and angular pendulum displacement. Through these efforts, we aim to enhance the safety and efficiency of crane operations by minimizing load oscillation during transportation.

2 Project Lifecycle and Requirements Analysis Integration

The execution of this project is managed with V-model.

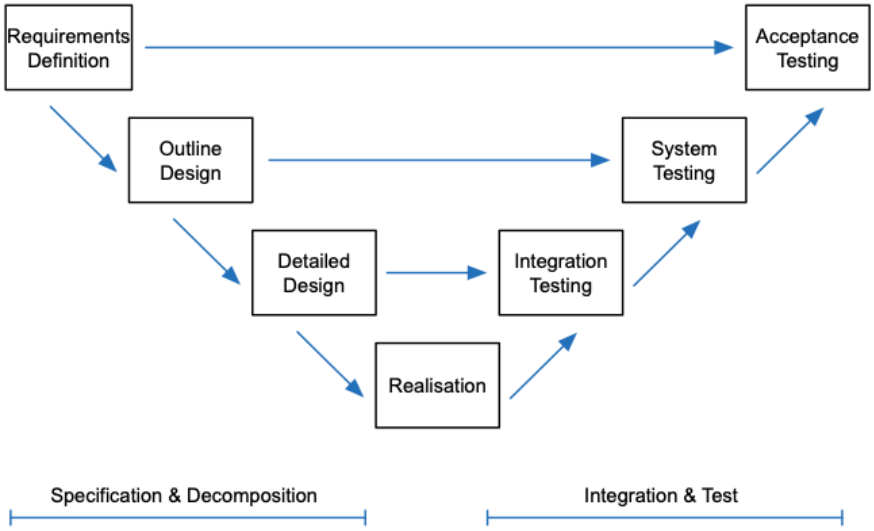


Figure 3: V-Modelfor porject[25]

The Cart-Pendulum Control project encompasses a systematic approach aimed at developing a robust control system for managing a cart-pendulum configuration. The project involves several phases, each crucial in ensuring the successful implementation of the system.

1. Top-Level Requirement: At the onset of the project, an in-depth analysis of overarching project requirements was conducted to discern key objectives and constraints. This preliminary stage laid the groundwork for subsequent phases by providing a clear understanding of the project's scope and goals.

2. Requirement Analysis and Architecture Design: Following the top-level requirement analysis, detailed high-level requirements were dissected into specific domains such as Modelling, Hardware, Software, and GUI. This phase also involved conceptualizing the basic architecture for the controller and system, establishing a blueprint for further design and development efforts.

3. Design: Conducting a mathematical analysis of system dynamics was integral to this phase. Simulink models were created to represent both the controller and plant, serving as the foundation for subsequent implementation efforts.

4. Implementation: Translating designs into tangible solutions marked the implementation phase. Controllers were carefully scaled to match hardware specifications, and S-functions were developed for Software-in-Loop (SIL) testing. Additionally, the Zynq board was programmed to execute control algorithms, and a user-friendly GUI was crafted using Java to enhance system interaction.

5. Module Testing: Module testing was conducted in parallel with the design phase to validate individual component functionality. This rigorous testing ensured the reliability and efficacy of components such as the plant, controllers, and cart speed controller, including the testing of S-functions.

6. Integration Testing: Once individual components were validated, integration testing was performed to evaluate interactions between different elements of the system. Through simulation, the overall performance of the system was assessed, and the control algorithm was tested on the Zynq board to ensure functionality in real-world scenarios.

7. Acceptance Testing: The final phase of the project involved comprehensive acceptance testing to confirm adherence to predefined criteria and desired specifications. Rigorous testing on the test stand verified the functionality of the complete system, ensuring it met all desired requirements.

Requirement Analysis: Requirements analysis stands as a critical juncture within the project life cycle, where accurately identifying needs is paramount. Thorough examination of top-level requirements provided a comprehensive understanding of project objectives, guiding the creation of a detailed low-level requirement document. This document, categorized into Modelling, Hardware, Software, and GUI requirements, facilitated clear comprehension for stakeholders and aided in task delineation.

Requirement Categories:

- Design requirements enable the transformation of concepts into design features.
- Functional requirements delineate system functions in terms of behavior.
- Non-functional requirements assess system operation beyond specific behaviors.
- Interface requirements necessitate interaction or communication with other systems.
- Test requirements encompass various testing procedures to ensure system reliability.
- Safety requirements mitigate risks and ensure safe system operation.

3Mathematical Modelling of the cart pendulum system

The Cart Pendulum System is mathematically characterized by differential equations, with the Lagrange equation serving as the primary framework for its description. Initially, a set of nonlinear equations is formulated to represent the system's dynamics. To facilitate control using diverse algorithms, the system undergoes linearization, transforming it into a State Space model. This linearization process not only simplifies the system but also enables the application of efficient control methodologies, enhancing the system's controllability and responsiveness to various control strategies.

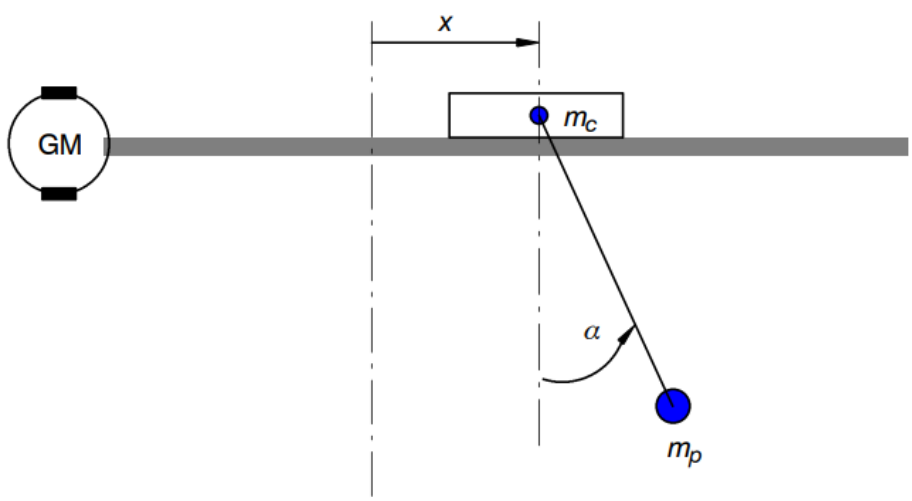


Figure 4: Cart Pendulum System [14]

3.1 Derivation of differential equation of the system

Using the Lagrange equation to characterize the system requires describing its total kinetic and potential energy. Given is the Lagrange function:

$$L = E_{kin} - E_{pot} \tag{1}$$

The system's dynamic behavior is fully described by the Lagrange equation. The Lagrange equation is expressed as:

$$\frac{\partial}{\partial t}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \left(\frac{\partial L}{\partial q_i}\right) = Q_i \tag{2}$$

where,

q_i: Coordinate vector.

Q_i: Non-conservative force vector.

3.2 Non-Linear equations

It is not necessary to use the Lagrange formalism in Cartesian Coordinates. Alternatively, the system's degrees of freedom may be employed. The two-dimensional level is where the cart pendulum control mechanism is located. The two degrees of freedom in the system to be built are the pendulum's rotation around the pivot α and the linear displacement of the cart x.

The vector coordinate q_i, given as follows, is created by combining the polar coordinates x and α, which are selected to characterize all energies and motions:

$$q_i = \begin{pmatrix} x \\ \alpha \end{pmatrix}$$

Drive force $F_c$is used to control the system; it is applied to the cart and operates in the x direction. A non-conversative force, $F_c$serves as an input to the nonlinear model that is being controlled. The non-conservative forces' vector, denoted as Q_i, is as follows:

$$Q_i = \begin{pmatrix} F_c \\ 0 \end{pmatrix}$$

The total kinetic energy of the system is calculated to derive the nonlinear equation of the cart pendulum system. This may be expressed as follows:

$$E_{\text{kin}} = \frac{1}{2} \cdot M \cdot v^2 + \frac{1}{2} \cdot m \cdot \left(\dot{x_p}^2 + \dot{y_p}^2\right) \tag{3}$$

The kinetic energy of the cart is represented by the first component in this instance, while the kinetic energy of the pendulum is represented by the second. Look at the following graphic to get the $x_p$andy_p coordinates.

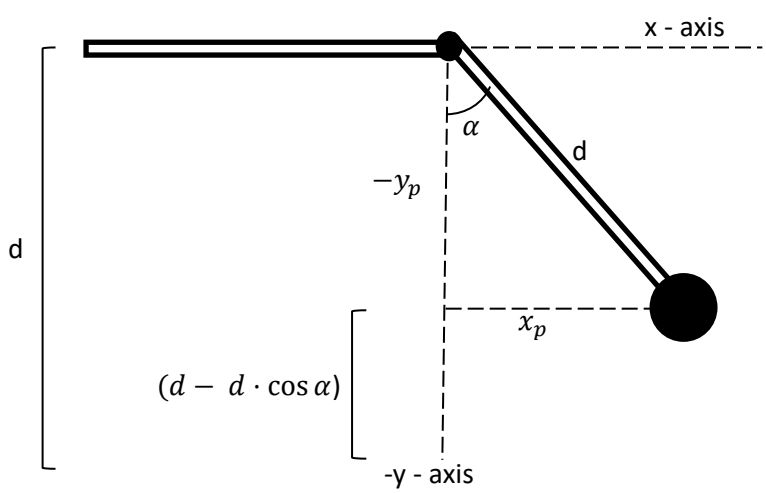


Figure 5: Pendulum Co-ordinates

From the above figure:

- The x - coordinate is given as: $x_p = d \cdot \sin\alpha$
- The y - coordinate is given as: $-y_p = d \cdot \cos\alpha$
- The x-component of the pendulum velocity is given as:

$$\dot{x}_p = v + d \cdot \omega \cdot \cos\alpha \tag{4}$$

The y-component of the pendulum velocity is given as:

$$\dot{y}_p = d \cdot \omega \cdot \sin\alpha \tag{5}$$

Substituting [4] and [5] in [3]

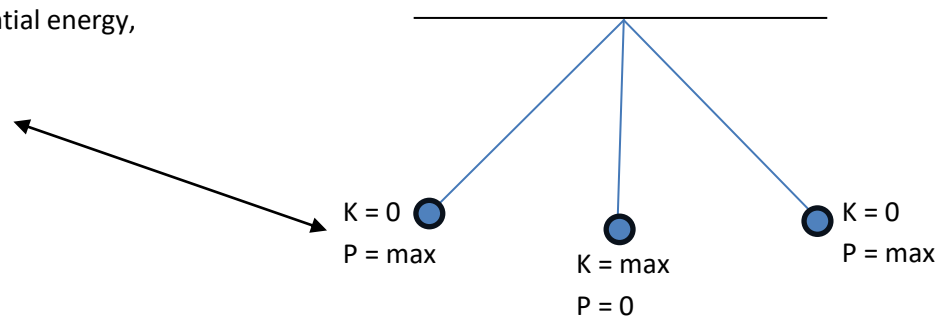
$$E_{\text{kin}} = \frac{1}{2} \cdot \mathrm{M} \cdot v^2 + \frac{1}{2} \cdot m \cdot ((v + d \cdot \omega \cdot \cos\alpha)^2 + (d \cdot \omega \cdot \sin\alpha)^2$$

After further simplifying the equation, kinetic energy is given as:

$$E_{\text{kin}} = \frac{1}{2} \cdot M \cdot v^2 + \frac{1}{2} \cdot m[v^2 + 2 \cdot v \cdot \omega \cdot d \cdot \cos\alpha + d^2 \cdot \omega^2] \tag{6}$$

Now, for the potential energy,

$E_{pot} = m \cdot g \cdot \mathrm{h}$



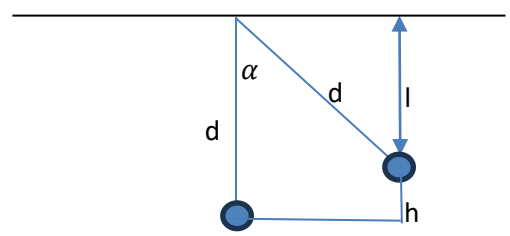
Thus,

$\cos\alpha = l/d$

$d\cos\alpha = l$

$d = h + l$

$h = d(1 - \cos\alpha)$



Hence, the potential Energy of the cart is zero and of the pendulum it is given as:

$$E_{pot} = m \cdot g \cdot d \cdot (1 - \cos\alpha) \tag{7}$$

Substituting E_{pot} and E_{kin} in the above equation [1],

$$L = \frac{1}{2} \cdot M \cdot v^2 + \frac{1}{2} \cdot m \cdot [v^2 + 2 \cdot v \cdot \omega \cdot d \cdot \cos\alpha + d^2 \cdot \omega^2] - m \cdot g \cdot d(1 - \cos\alpha) \tag{8}$$

At this point, friction force is introduced,

$$F_{fr} = \mathrm{c} \cdot v \tag{9}$$

The Lagrange equation [1], with our two generalized co-ordinates is now written as:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial v}\right) - \left(\frac{\partial y}{\partial x}\right) + F_{fr} = F_c \tag{10}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \omega}\right) - \left(\frac{\partial L}{\partial \alpha}\right) = 0 \tag{11}$$

Substitute and solve [8] in [10] and [11]

$$\dot{v} \cdot (M + m) = F_c - m \cdot \dot{\omega} \cdot d \cdot \cos\alpha + m \cdot \omega^2 \sin\alpha - c \cdot v$$
$$\dot{\omega} \cdot d^2 \cdot m = -\dot{v} \cdot d \cdot \cos\alpha \cdot m - m \cdot d \cdot g \cdot \sin\alpha$$

Rearranging equation [13] and [14]

$$\dot{v} = \frac{F_c}{(M + m)} - \frac{m \cdot \dot{\omega} \cdot d \cdot \cos\alpha}{(M + m)} + \frac{m \cdot \omega^2 \cdot d \cdot \sin\alpha}{(M + m)} - \frac{c \cdot v}{(M + m)} \tag{12}$$

$$\dot{\omega} = -\frac{\dot{v} \cdot \cos\alpha}{d} - \frac{g \cdot \sin\alpha}{d} \tag{13}$$

3.3 Linear Equations

Linearization at the operating point is a technique used in control engineering to simplify nonlinear equations by approximating them with linear equations. The equations (12) and (13) represents the behaviour of the system. By linearizing around a chosen operating point, typically zero, the nonlinear system can be represented by linear differential equations.

Function $f(\dot{\omega}, \omega, \alpha, \dot{v}, v, F_c)$--->(0,0,0,0,0,0) :

Now, by using series expansion for linearization around $f(\ddot{\alpha}, \omega, \alpha, \ddot{x}, \dot{x}, F_c)$--->(0,0,0,0,0,0)working-point, the following approximations are to be considered,

α ≈0, cos α ≈1, sin α ≈α, $\dot{\alpha}^2$≈0.

Taking a partial derivative of a function,

$$f(\dot{\omega},\omega,\alpha,\dot{v},v,F_c) \approx f\left(\dot{\omega}_0,\dot{\alpha}_0,\alpha_{0,},\dot{v}_0,v_0,F_{C0}\right) + \left.\frac{\partial \mathrm{f}(\dot{\omega},\omega,\alpha,\dot{v},v,F_c)}{\partial \ddot{\alpha}}\right|_{\omega_0,v_0,\alpha_0,\dot{v}_0,\dot{\omega}_0,\mathrm{F_{C0}}} (\dot{\omega}-\dot{\omega}_0) + \ldots\ldots\ldots$$

After substituting the values in [12] and [13], the following linear equation,

$$\dot{v} = \frac{F_c}{M} + \frac{m \cdot g}{M} \cdot \alpha - \frac{c \cdot v}{M} \tag{14}$$

$$\dot{\omega} = -\frac{F_c}{M \cdot d} - \frac{g \cdot (M+m)}{M \cdot d} \cdot \alpha + \frac{c \cdot v}{M \cdot d} \tag{15}$$

4State Space Representation of Linear Model

For linear systems, it is possible to write the state-space representation in the following way:

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t)$$

$$y(t) = C \cdot x(t) + D \cdot u(t)$$

where,

A is nxn System Matrix

B is nxp Input Matrix

C is qxn Output Matrix

D is qxp Feed-through Matrix

Here, n is the number of states, p is the number of inputs and q is the number of outputs. The state-space form is a first-order differential equation. The previous description of cart pendulum system consists of two second-order differential equations. Hence, the two second-order differential equations are converted into four first-order differential equations and represented in a matrix form.

From above equation state-space representation of linear cart pendulum system is given as:

$$\begin{bmatrix} \dot{\omega} \\ \omega \\ \dot{v} \\ v \end{bmatrix} = \begin{bmatrix} 0 & \frac{-g\cdot(M+m)}{M\cdot d} & \frac{c}{M\cdot d} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \frac{m\cdot g}{M} & -\frac{c}{M} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega \\ \alpha \\ v \\ x \end{bmatrix} + \begin{bmatrix} \frac{-1}{M\cdot d} \\ 0 \\ \frac{1}{M} \\ 0 \end{bmatrix} \cdot F_c$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \omega \\ \alpha \\ v \\ x \end{bmatrix} + 0\cdot u(t)$$

It represents the single input multiple output (SIMO) system. The notations in above equations are as follows:

$\dot{\omega}$ = angular acceleration $\dot{v}$ = acceleration

ω = angular velocity v = velocity

α = angle x = displacement

The measured physical parameters of the system on test stand as follows:

M= 4.676 Kg

m = 0.098 Kg

d = 0.285 m

g = 9.8 $m/_{s^2}$

c = 5

With these above values the state space matrices become as follows:

A = $\begin{bmatrix} 0 & -35.1066 & 0.3047 & 0 \\ 1.0000 & 0 & 0 & 0 \\ 0 & 0.2054 & -1.0693 & 0 \\ 0 & 0 & 1.0000 & 0 \end{bmatrix}$

B = $\begin{bmatrix} -0.7504 \\ 0 \\ 0.2139 \\ 0 \end{bmatrix}$

C = $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

D = 0

The linear cart pendulum system is fully represented by the matrices A, B, C, and D.

5MATLAB Modelling and Simulation of Cart Pendulum System

The linear and non-linear cart pendulum system models are shown in the previous chapter. It's time to use MATLAB/Simulink to study the system behavior at this point.

5.1 Nonlinear Model

5.1.1 MATLAB code

Below is the MATLAB script for nonlinear cart pendulum system.

```
%Model Parameters
M = 4.676;        %kg% Mass of the cart
m = 0.098;      %kg% Mass of the pendulum
d = 0.285;      %m% length of the pendulum rope
g = 9.8;      %m/s^2% gravity
c=5;        %friction constant
```

5.1.2 Simulink Model

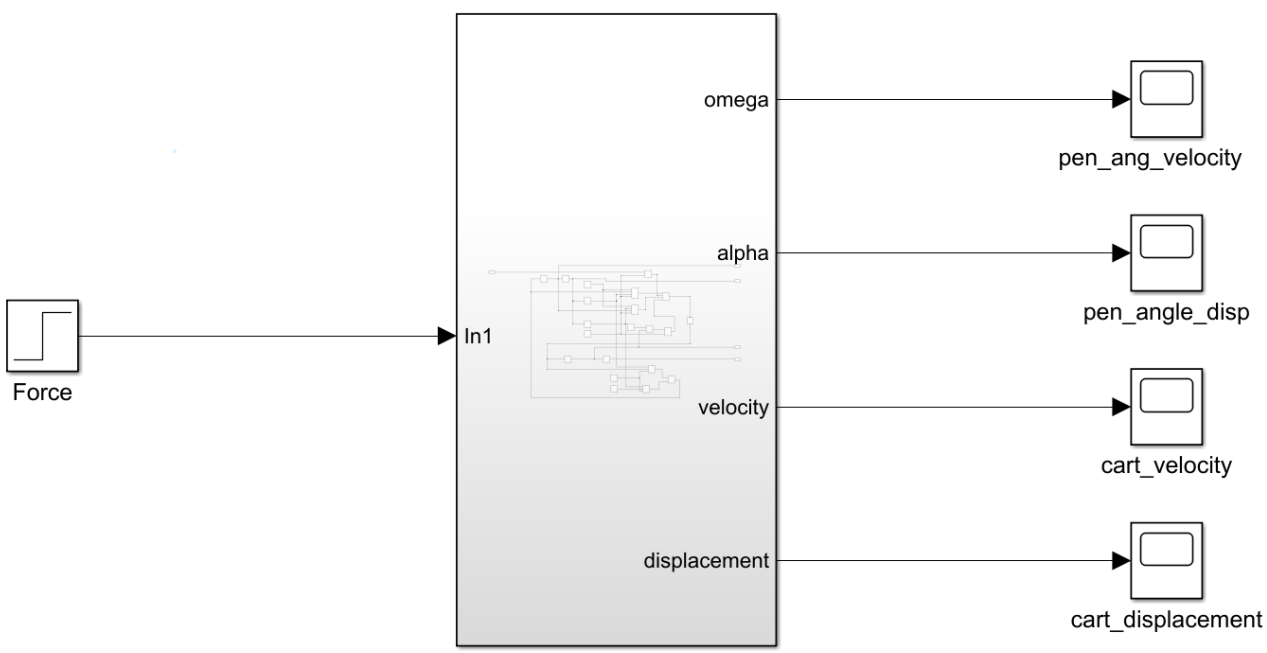


Figure 6:Nonlinear subsystem model of cart pendulum system

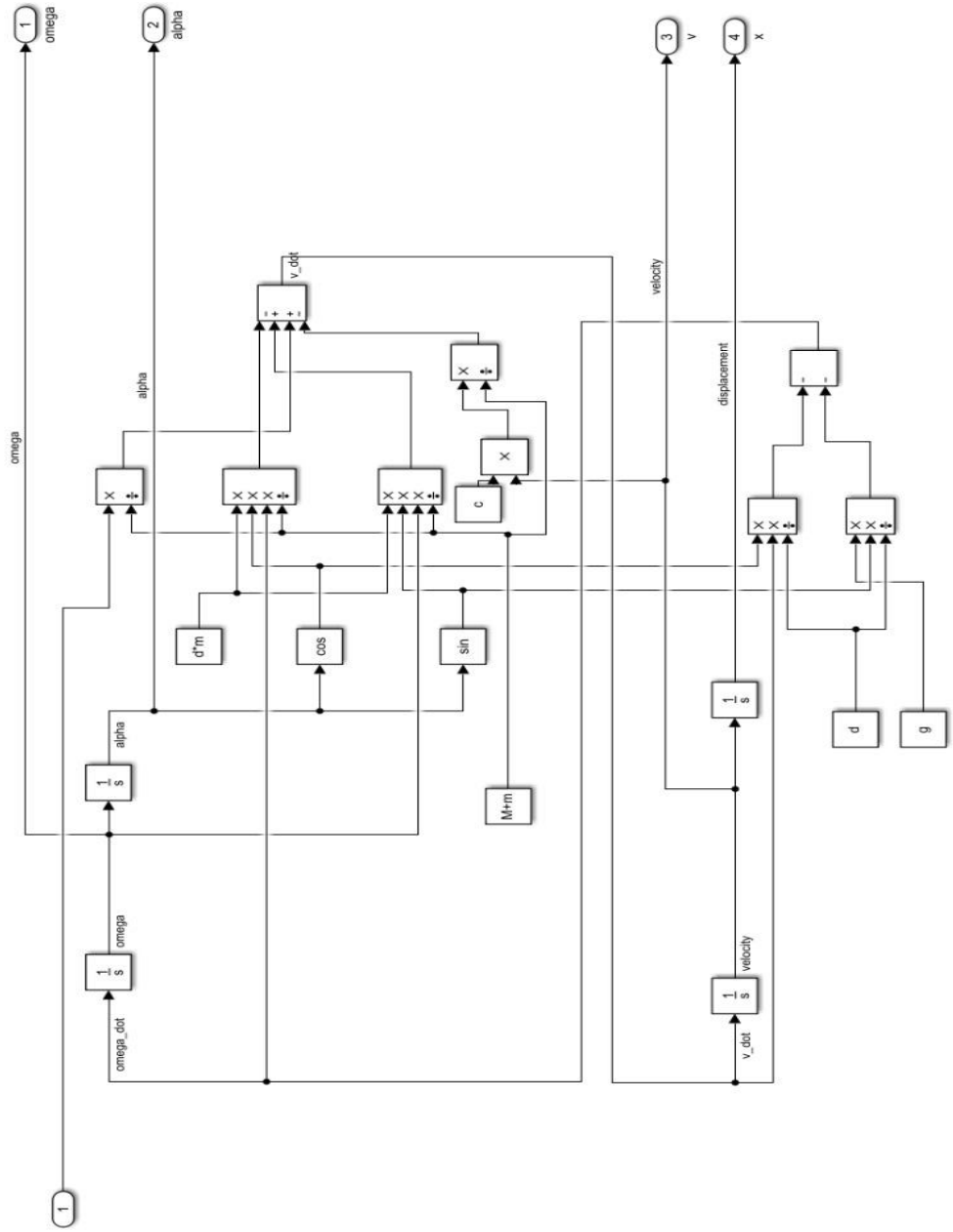


Figure 7: Detailed nonlinear model of cart pendulum system

Figure illustrates detailed nonlinear model of cart pendulum system. The input to the system is force F. The system output observed are cart displacement, cart velocity, pendulum angular displacement and pendulum angular velocity.

5.1.3 Output Graphs

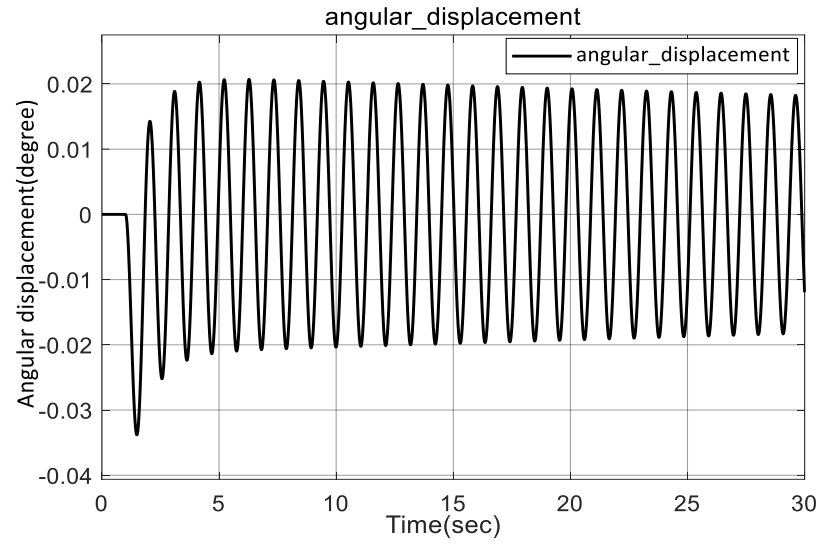


Figure 8:Angular Displacement of Pendulum (Non-Linear Model)

In the angular displacement graphs, initially a greater negative swing is observed as it is a nonlinear system. This is because when the cart begins to move, a sudden jerk is applied, and the pendulum oscillates with a greater swing. The range of the pendulum oscillates between 0.02 to -0.03 deg.

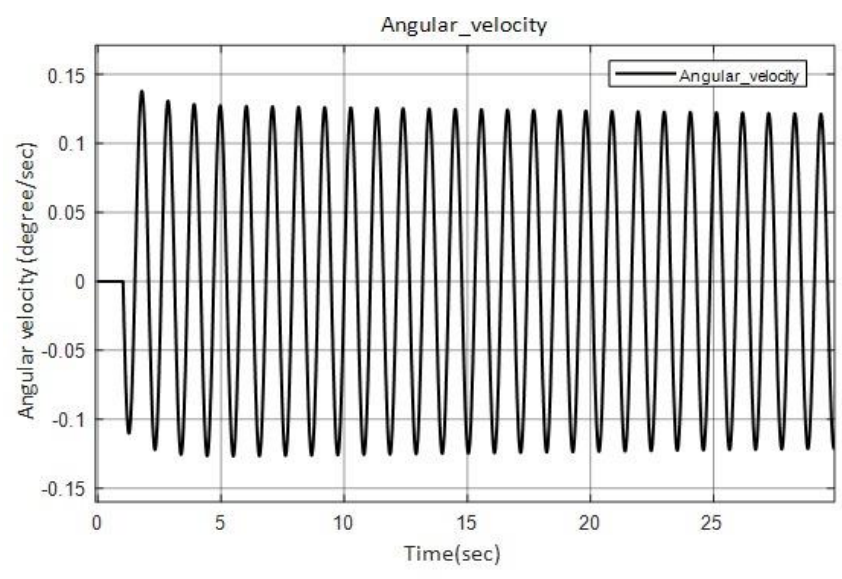


Figure 9:Angular velocity of Pendulum (Non-Linear Model)

Figure shows the angular velocity and angular displacement of the pendulum for nonlinear model. Input force of 1N is applied. The angular velocity of the pendulum shows no unexpected events or behaviour. The angular velocity ranges from 0.14 to – 0.12 deg/sec.

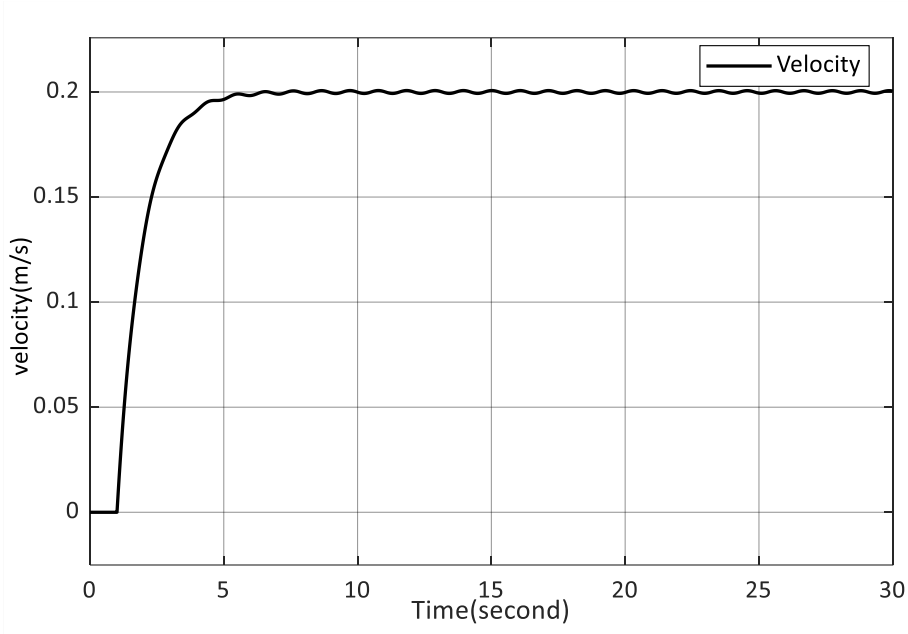


Figure 10:Velocity of Cart (Non Linear Model)

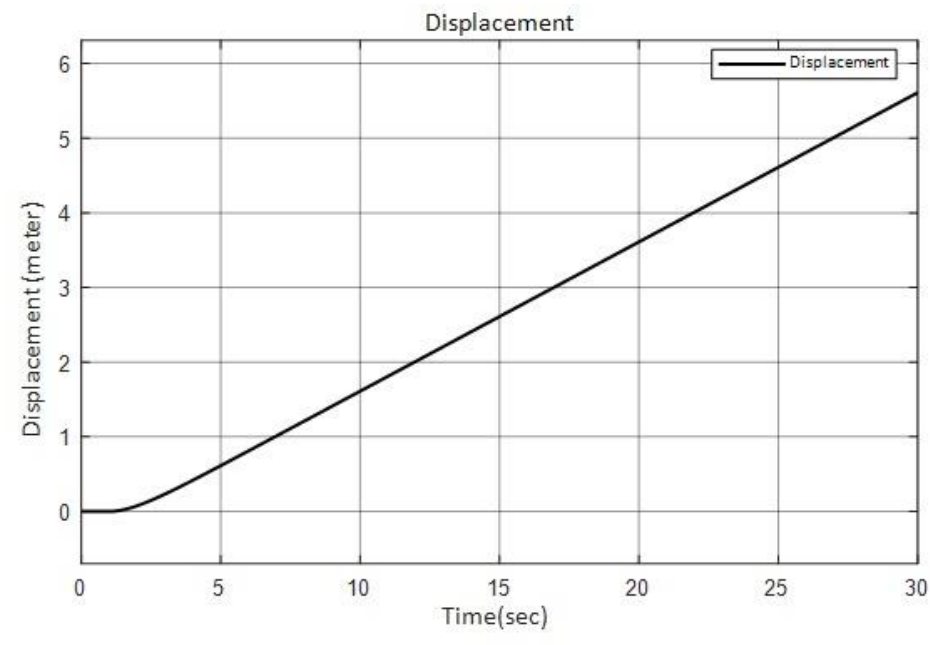


Figure 11:Displacement of Cart (Non Linear Model)

Figure shows the cart velocity and cart displacement. Input force of 1N is applied to accelerate the cart. It is observed from the graph that the cart velocity is constant after some time, due to which the displacement of the cart is increasing exponentially.

5.2 Linear Model

State-space representation is used to design the linear model for cart pendulum system. The State-space form of the linearized Cart Pendulum System is given as:

$$\begin{bmatrix} \dot{\omega} \\ \omega \\ \dot{v} \\ v \end{bmatrix} = \begin{bmatrix} 0 & \frac{-g\cdot(M+m)}{M\cdot d} & \frac{c}{M\cdot d} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \frac{m\cdot g}{M} & -\frac{c}{M} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \omega \\ \alpha \\ v \\ x \end{bmatrix} + \begin{bmatrix} \frac{-1}{M\cdot d} \\ 0 \\ \frac{1}{M} \\ 0 \end{bmatrix} \cdot F_c$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \omega \\ \alpha \\ v \\ x \end{bmatrix} + 0\cdot u(t)$$

5.2.1 Matlab Code

```
%state space

%System Matrix
A = [0,(-g*(M+m))/(M*d),c/M*d,0;1,0,0,0;0,(m*g)/M,-c/M,0;0,0,1,0];
%Input Matrix
B = [-1/(M*d);0;1/M;0 ]
%Output Matrix
C=eye(4);
%Feed-through Matrix
D=[0;0;0;0];
%ct = C(4,:);
```

5.2.2 Simulink Model

Figure illustrates the linear model of cart pendulum system. The linear system is modelled using state-space block in Simulink. Input to the system is force F. The initial condition is set to x=0, $\dot{x}$=0, $\dot{\alpha}$=0, α=0 i.e., the state of rest. The outputs of the system observed are cart displacement, cart velocity, pendulum angular displacement and pendulum angular velocity.

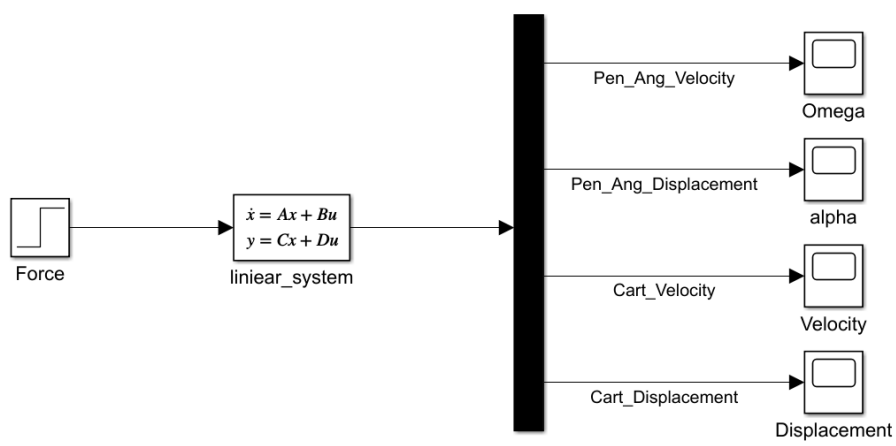


Figure 12:State space representation of cart pendulum system

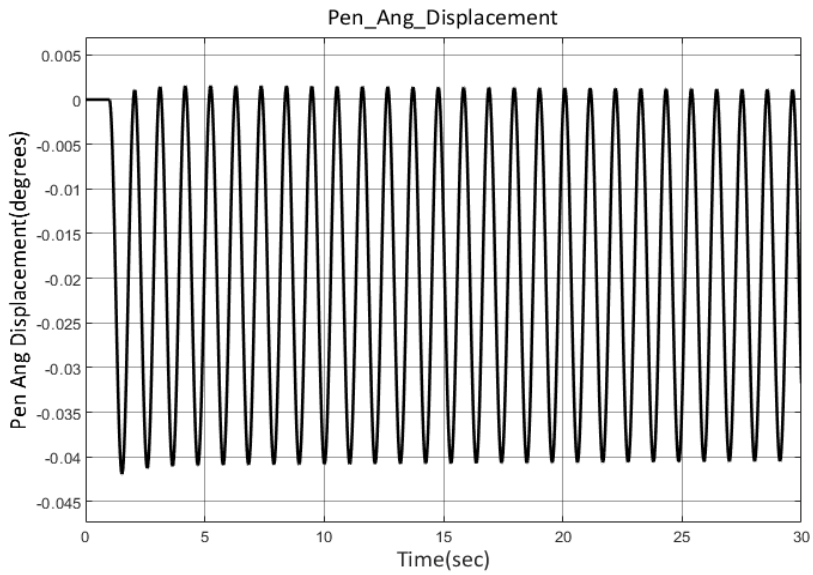


Figure 13:Angular Velocity of Pendulum (Linear Model)

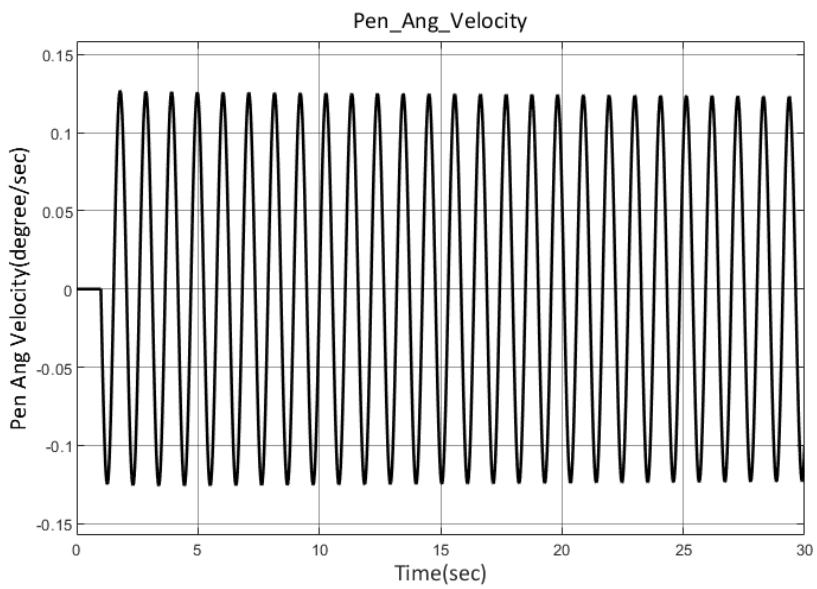


Figure 14:Angular Displacement of Pendulum (Linear Model)

Force applied to the cart is 1N. As the cart is applied with constant force, there is a consistent swing in pendulum angular velocity. The angular velocity of the pendulum for linear system showed no unexpected events or behavior, and it is very close to the behavior of nonlinear model shown in section.

The pendulum oscillations for linear system between the ranges 0 to -0.04 degrees which is small. Unlike nonlinear system the angular displacement of the pendulum is consistent with no initial spike in oscillations.

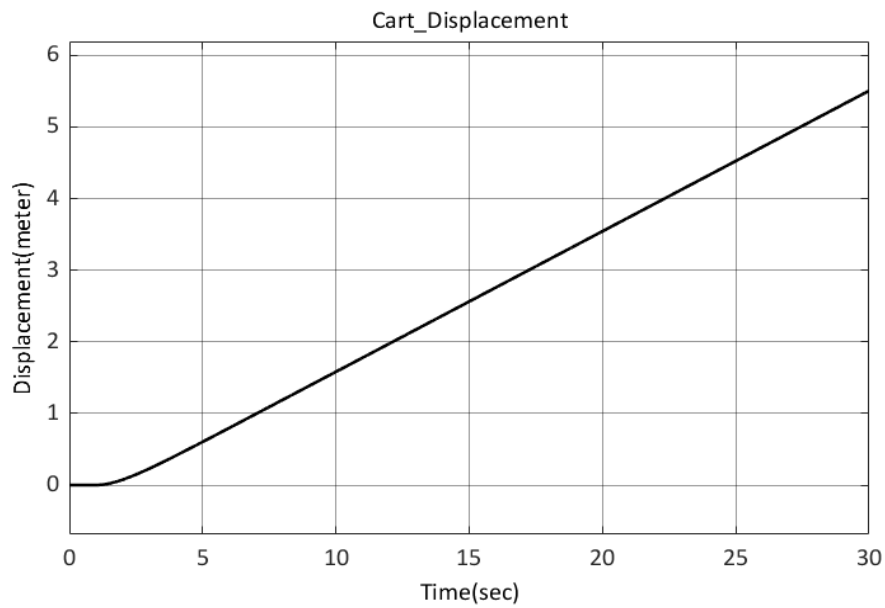


Figure 15:Velocity of Cart (Linear Model)

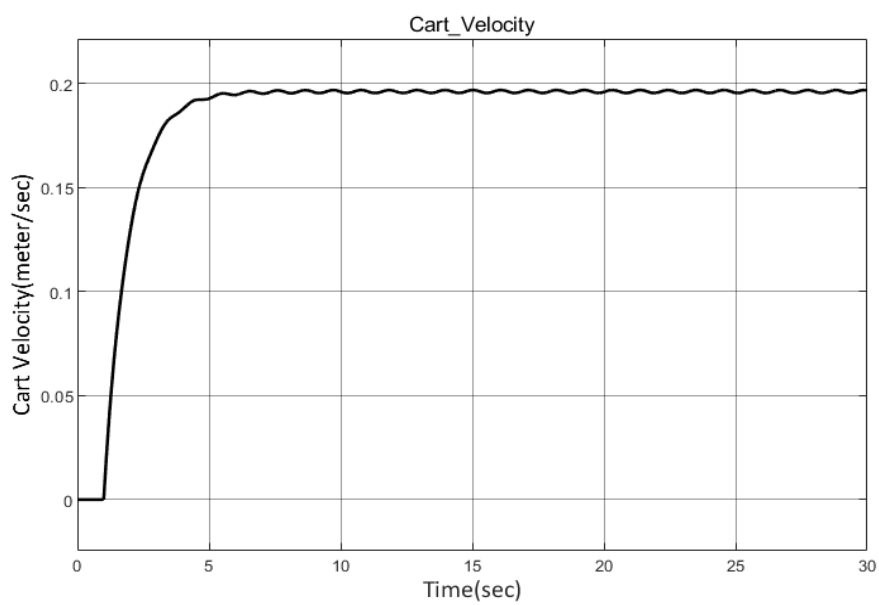


Figure 16:Displacement of cart (Linear Model)

Figure illustrates cart displacement and velocity of linear model. The cart is accelerated constantly due to constant unit step force. Due to the presence of friction, it is observed that the velocity increases initially to overcome the static friction and then it remains constant.

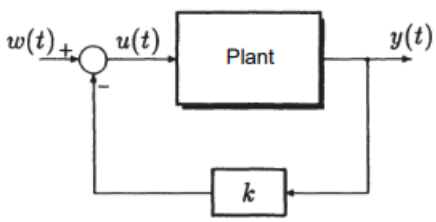
The cart is accelerated constantly due to constant unit step force, and this results in an exponential increase in cart position. From section 6.1.3, it is observed that the linear and nonlinear model are very close.

6Cart Position Controller

6.1 State Feedback System

The state-feedback controller is designed based on four states: cart displacement, cart velocity, pendulum angular displacement, and pendulum angular velocity. Figure illustrates the layout or structure of this system.

In such a system, matrix A typically represents the system dynamics, B represents the control input matrix, C represents the output matrix, and D represents the feedforward matrix. The state-feedback controller would utilize these matrices along with the desired control objectives to design a controller that can manipulate the system's states to achieve the desired behavior or performance.



For the cart pendulum system, the feed-through matrix D is taken as zero as it does not get directly influenced by the input. The A, B and C matrices represent the open loop system. w is the reference input to the system and u is the manipulating variable (force). The feedback gains for all the states are represented by the state-feedback vector K. With the help of state feedback, the system becomes a closed loop.

The state space equations for the closed loop system are given as:

$$\dot{x}(t) = (A - B \cdot K) \cdot x(t) + B \cdot w$$

$$y = C \cdot x(t)$$

The state-feedback gain vector affects the poles of the system. Feedback vector K is chosen such that all the eigen values of the matrix $(A - B \cdot K)$ are negative, to make the system stable. The fundamental benefit in state-feedback is the free placement of poles. State feedback allows to place every single pole at any desired location in the left half of the s-plane while preserving the stability of the system. The placement of poles is done using Ackerman's formula.

The Ackerman formula is given by:

$$K = q_t \cdot (\alpha_0 \cdot I + \alpha_1 \cdot A \cdots\cdots + \alpha_0 \cdot An{-}1 + A_n)$$

where,

K : State-feedback vector

q_t : Last row of the inverse of controllability matrix

α : Coefficients of the desired polynomial obtained by pole placement

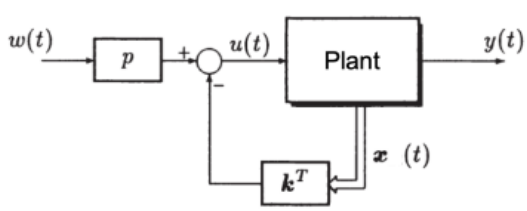
6.2 Proportional Controller

The state-feedback is not transparent to any errors at the output. If there is any disturbance at the output, the state-feedback controller cannot anticipate and correct it. This necessitates the use of output feedback.

The output feedback restores back the steady state accuracy even if there is a step disturbance at the output. The error obtained by calculating the difference between the reference value and the output value is amplified by the calculated proportional gain P and then given as input to the system. Because of the output feedback, the closed loop poles are affected. In order to compensate for this shift of poles, the state-feedback gain vector is modified as follows:

$$P = \frac{1}{((C_T * inv(B*K) - A) * B}$$

This is called P controller or proportional controller.



The state feedback controller and preamplifier calculations are shown in the MATLAB script file below.

```
% proportional controller

% Controllability matrix
s =[B,A*B,A^2*B,A^3*B];
% Rank of Controllability matrix
r = rank(s);
% Inverse of Controllability matrix
IS = inv(s);
% Last Row of Inverse Controllability matrix
q = IS(4,1:4);
% Desired Poles
pole = -4
alpha= poly([pole pole pole pole]);
% output matrix
ct = C(4,:);
% Ackermann?s formula
k = q*(alpha(5)*eye(4) + (alpha(4)*(A))+ (alpha(3)*(A^2)) + (alpha(2)*(A^3)) + (A^4));
% Proportional gain (Preamplifer Gain)
P = 1/(ct*((B*k-A)^-1)*B);
```

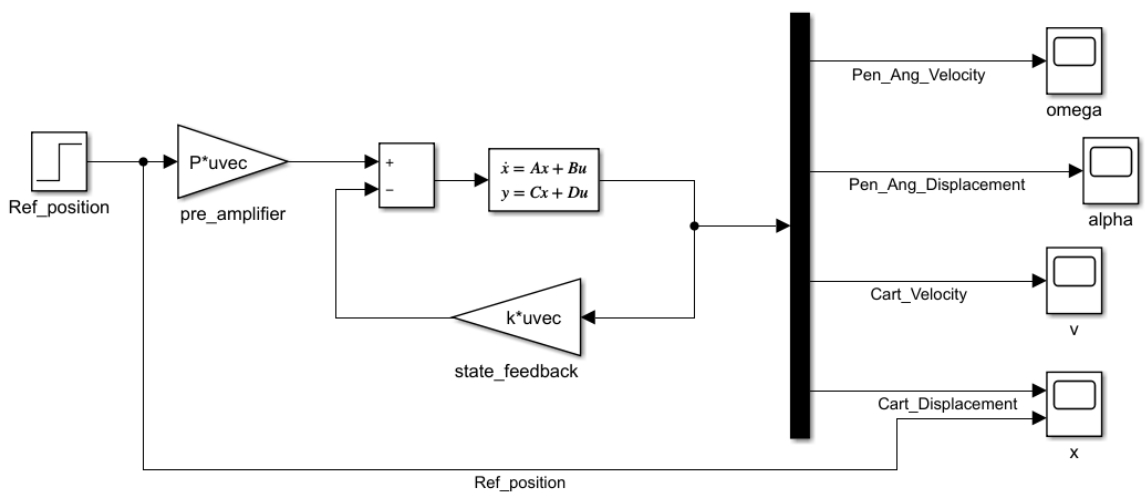


Figure 17:Proportional controller

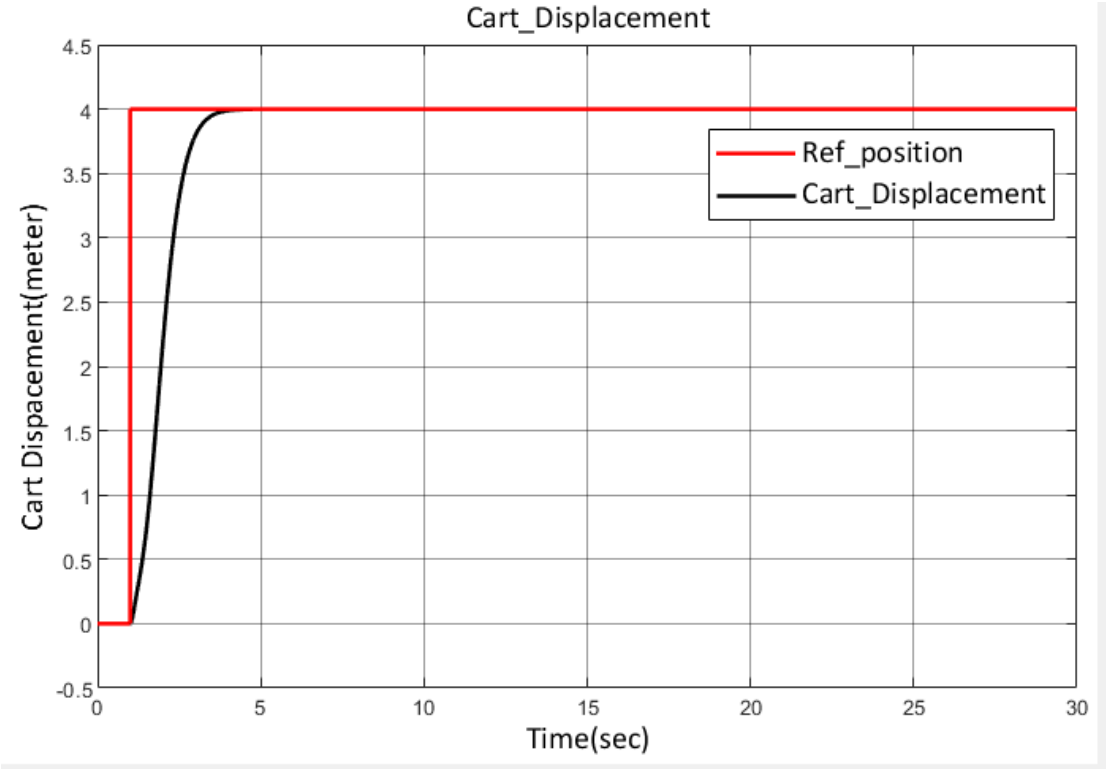
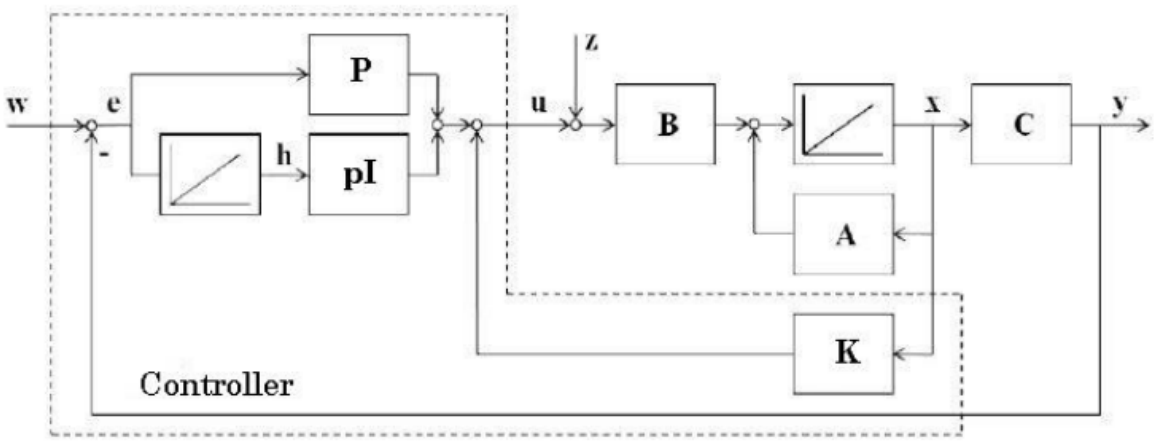


Figure 18:cart displacement of Proportional controller (linear)

6.3 Time Continuous PI Controller

To integrate the error brought on by the disturbance and return to steady state accuracy, the system needs an additional integrator before the input if it has distortions. Now that the integrator must process the error before taking any action, the system's response time gets noticeably slower. Figure depicts the full model of such a PI-controller.



PI state-feedback control for the cart pendulum system as plant is modelled. All poles are placed at -4. The state-feedback vector k based on the augmented system (5th order) is calculated. Nonlinear & linear PI-state-feedback control are verified for reference value steps Below are the steps for calculating the state-feedback vector.

- Modify matrices A and B to:

$$\underline{\bar{A}} = \begin{bmatrix} A & 0 \\ c^T & 0 \end{bmatrix} \& \underline{\bar{B}} = \begin{bmatrix} B \\ 0 \end{bmatrix}$$

- Obtain the controllability matrix S using the formula:

$$S = [B\ AB\ \ldots . A^{n-1}B]$$

- Obtain q_i which is the last row of the inverse controllability matrix S.

- Obtain the polynomial of the required poles [-4 -4 -4 -4 -4], also known as the desired characteristic polynomial, using:

$$P_\alpha\ (s) = (S - S_1)(S - S_2)(S - S_3)(S - S_4)(S - S_5)$$

- The Ackerman formula is given by:

$$K_{pi} = q_i \cdot (\alpha_0 \cdot I + \alpha_1 \cdot A \cdots\cdots + \alpha_0 \cdot A_{n-1} + A_n)$$

where, α is coefficients of the desired polynomial obtained by pole placement.

• Obtain the modified state-feedback vector k using Ackermann's formula, which will lead to:

$$K_{\mathrm{i}} = K_{\mathrm{pi}}\ (1{:}\ 4) - (P * C_{\mathrm{t}})$$

$$p_{\mathrm{i}} = -K_{\mathrm{pi}}\ (5)$$

where, p_{i} is the integrator gain.

k_{i} is the new state-feedback vector.

6.3.1 MATLAB script for calculation of PI controller

```
%Pi controller with state feedback

% System Matrix
A_pi = [A [0;0;0;0]; -ct 0];
% Input Matrix
B_pi = [B;0];
% Output Matrix
C_pi = eye(5);
% Feed-through Matrix
D_pi = [0;0;0;0;0];
%Controllability Matrix
S_pi =[B_pi A_pi*B_pi (A_pi^2)*B_pi (A_pi^3)*B_pi A_pi^4*B_pi ];
%Rank of Inverse Controllability Matrix
RI = rank(S_pi);
%Inverse Controllability Matrix
S_pi_inv = inv(S_pi);
% Last Row of Inverse Controllability matrix
Q_pi = S_pi_inv(5,1:5);
% Desired poles
alpha1= poly([pole pole pole pole pole]);
% Ackermann's formula for continous system with integrator
k_pi = Q_pi*(alpha1(6)*eye(5)+ alpha1(5)*A_pi + alpha1(4)*A_pi^2 + alpha1(3)*A_pi^3+ alpha1(2)*A_pi^4 + A_pi^5);
%Value of state-feedback
ksf =  k_pi(1:4)-(P*ct);
% Gain of Integral
PI = -k_pi(5);
```

6.3.2 Simulink Model

Figure shows the Simulink model of Time Continuous Position Controller. The states of the system are demultiplexed by a demux. The cart displacement is used as output feedback.

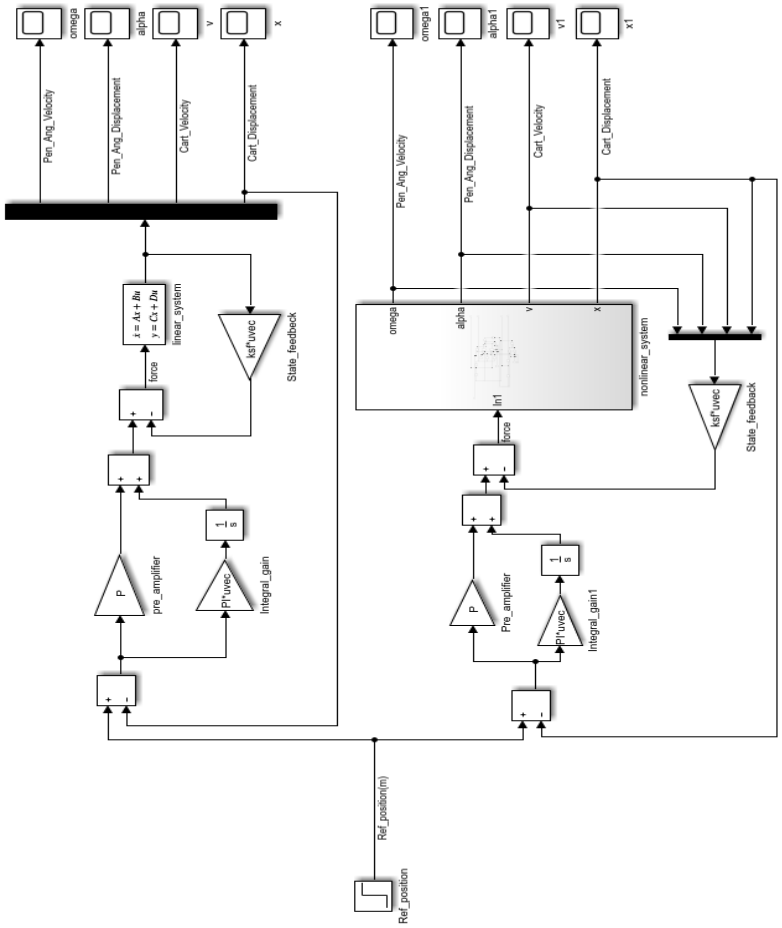


Figure 19: Continuous PI controller

The values of the gains & feedback vector are as follows:

Proportional Gain P = 34.8123
Integral Gain PI= 139.2494
Feedback vector ksf = [-6.1951 -110.7465 66.7828 139.2494]

6.3.3 Output Graphs

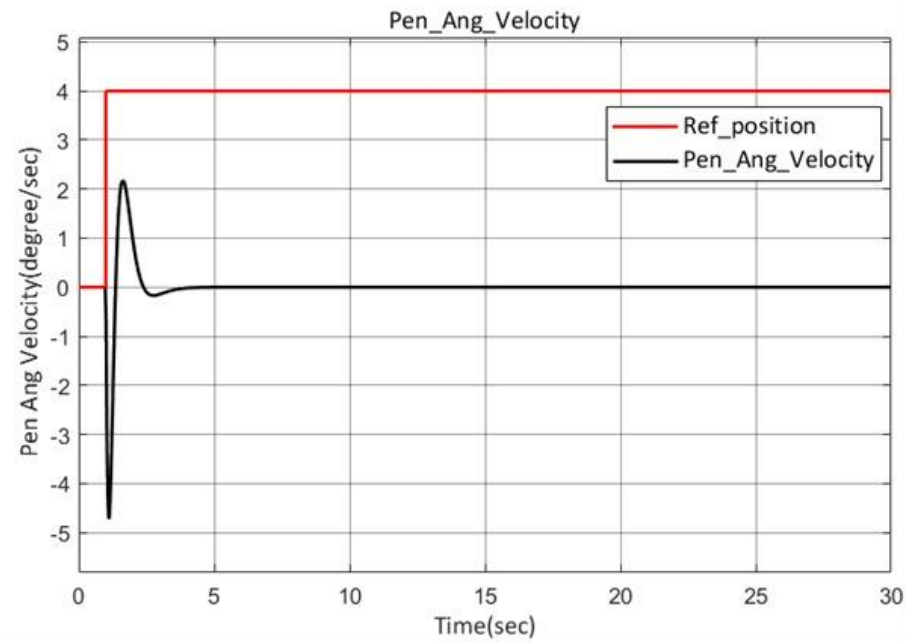


Figure 20: Linear angular velocity of pendulum (Continuous PI)

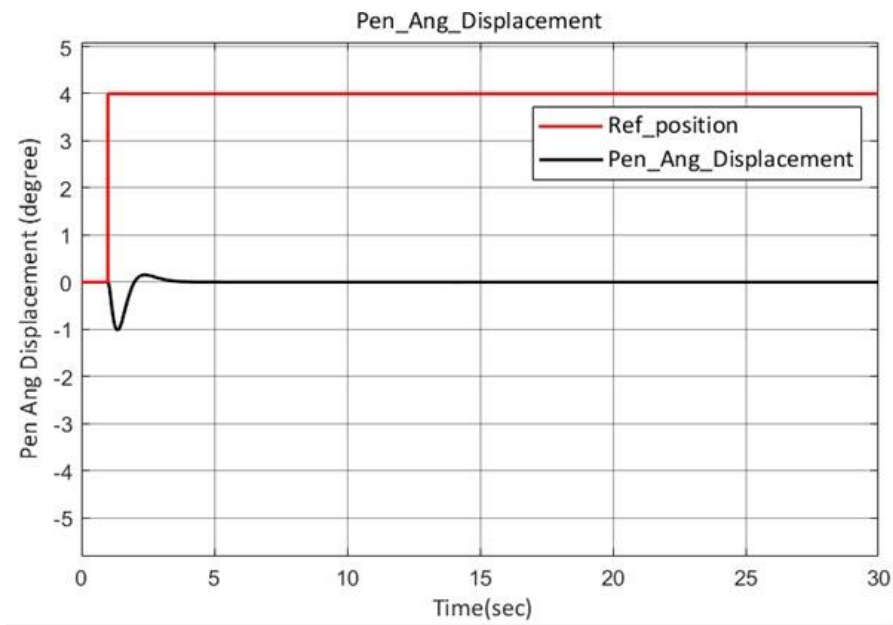


Figure 21: Linear angular displacement of pendulum (Continuous PI)

Above graphs show pendulum angular displacement and velocity of linear model without disturbances. Reference position of 4m is applied. It is seen that pendulum swings up-to -1 degrees initially because of the applied input, and the oscillation gradually settles to zero degree due to controller effect.

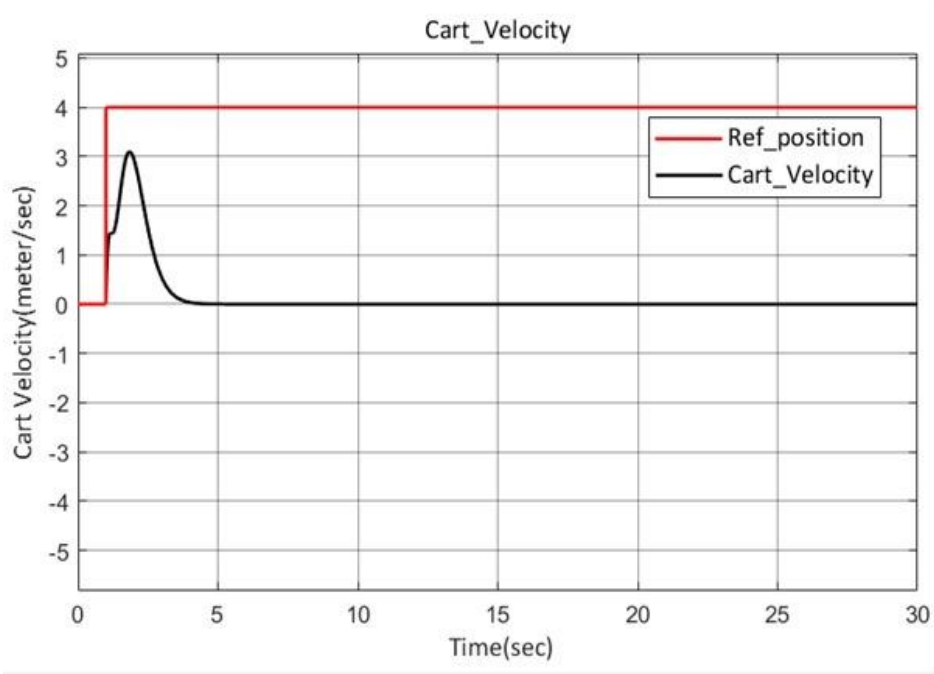


Figure 22: Linear cart velocity (Continuous PI)

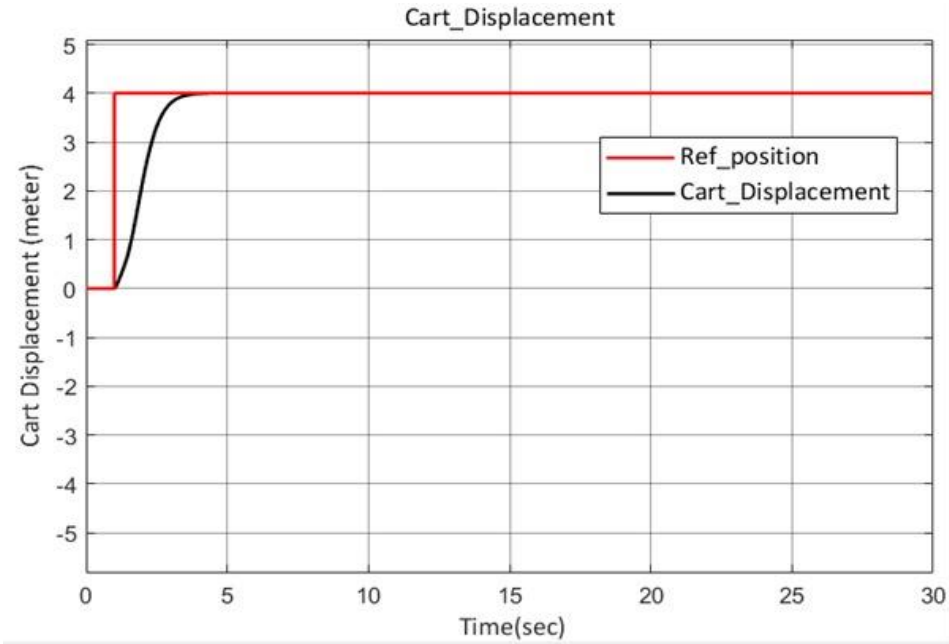


Figure 23: Linear cart displacement (Continuous PI)

The above figure illustrates the cart displacement and velocity of linear model without disturbances. Reference position of 4m is applied as input. The cart reaches the desired steady state accuracy in 4 secs.

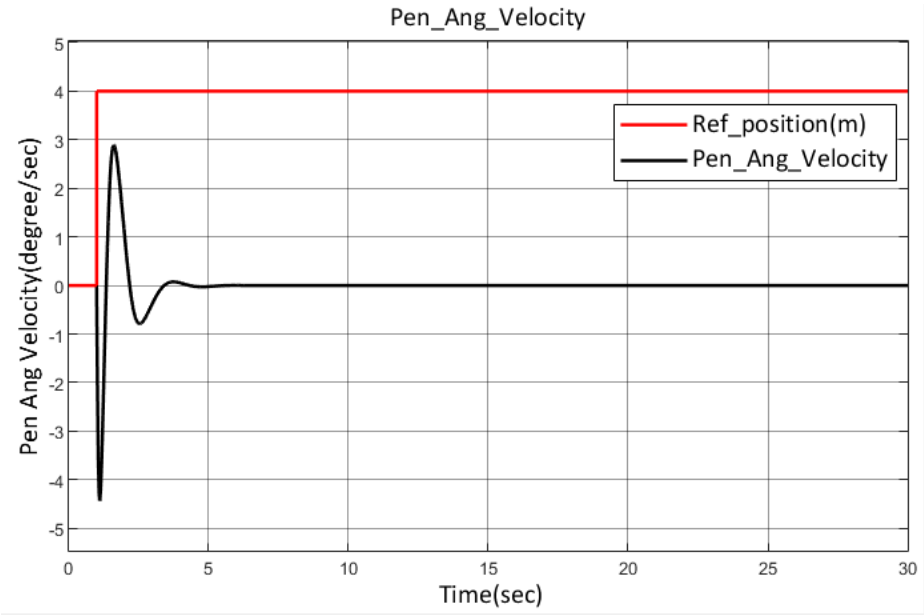


Figure 24: Nonlinear angular velocity of pendulum (Continuous PI)

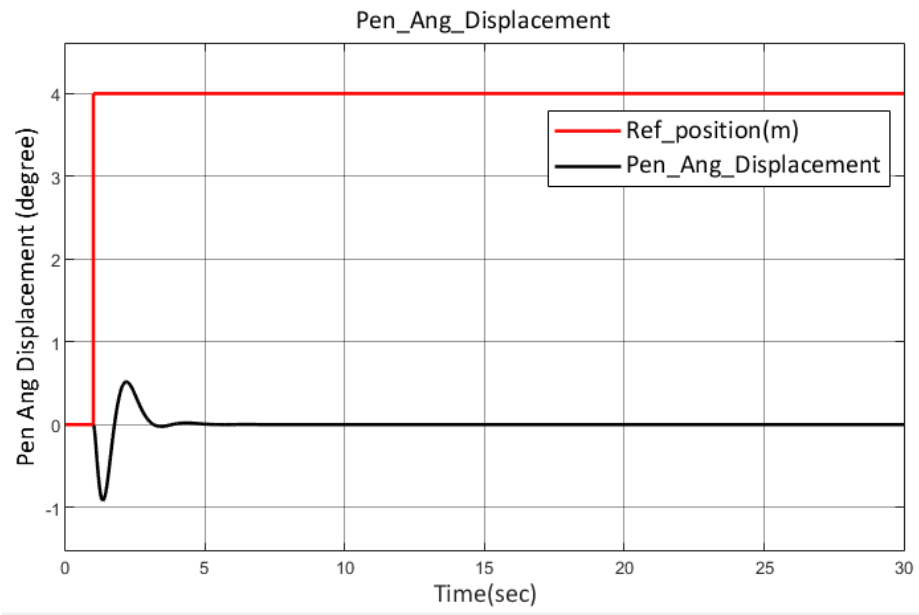


Figure 25: Nonlinear angular displacement of pendulum (Continuous PI)

Above figures show pendulum angular displacement and velocity of nonlinear model without disturbances. Reference position of 4m is applied. It is seen that pendulum swings up-to -1 degrees initially because of the applied input and the oscillation gradually settles to zero degree due to controller effect. In the nonlinear model it is observed that the oscillations are more when compared to linear model and oscillations settle down with some delay.

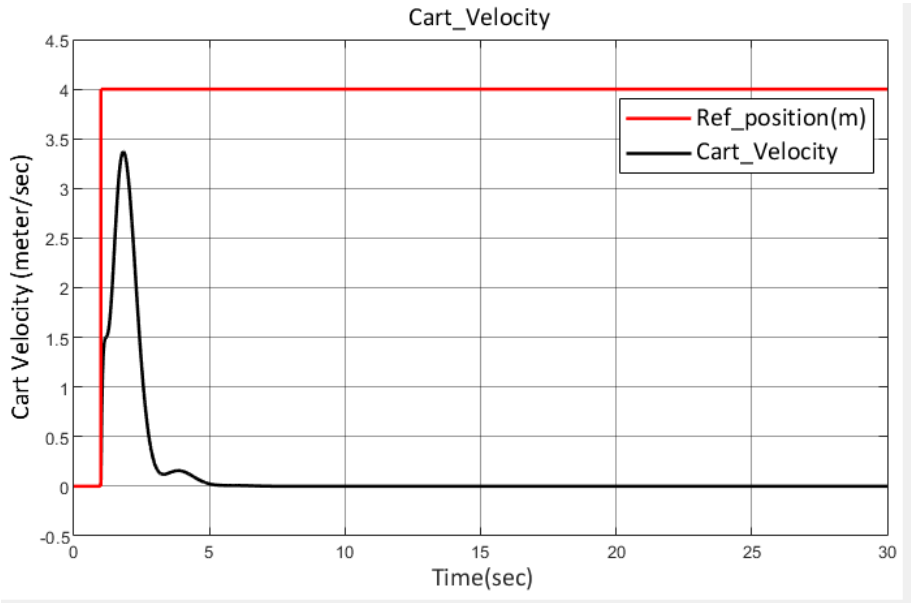


Figure 26: Nonlinear cart velocity (Continuous PI)

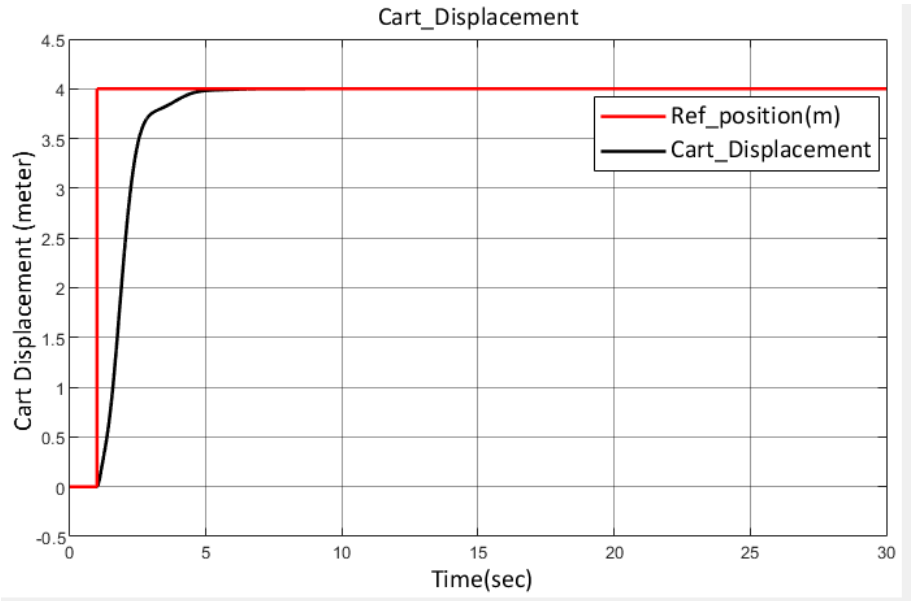


Figure 27: Nonlinear cart displacement (Continuous PI)

Above illustrates the cart displacement and velocity of nonlinear model without disturbances. Reference position of 4m is applied. The cart reaches the desired steady state accuracy in 5 secs. It take 1 more second to stable as compared to linear system.

6.3.4 Disturbance Effect

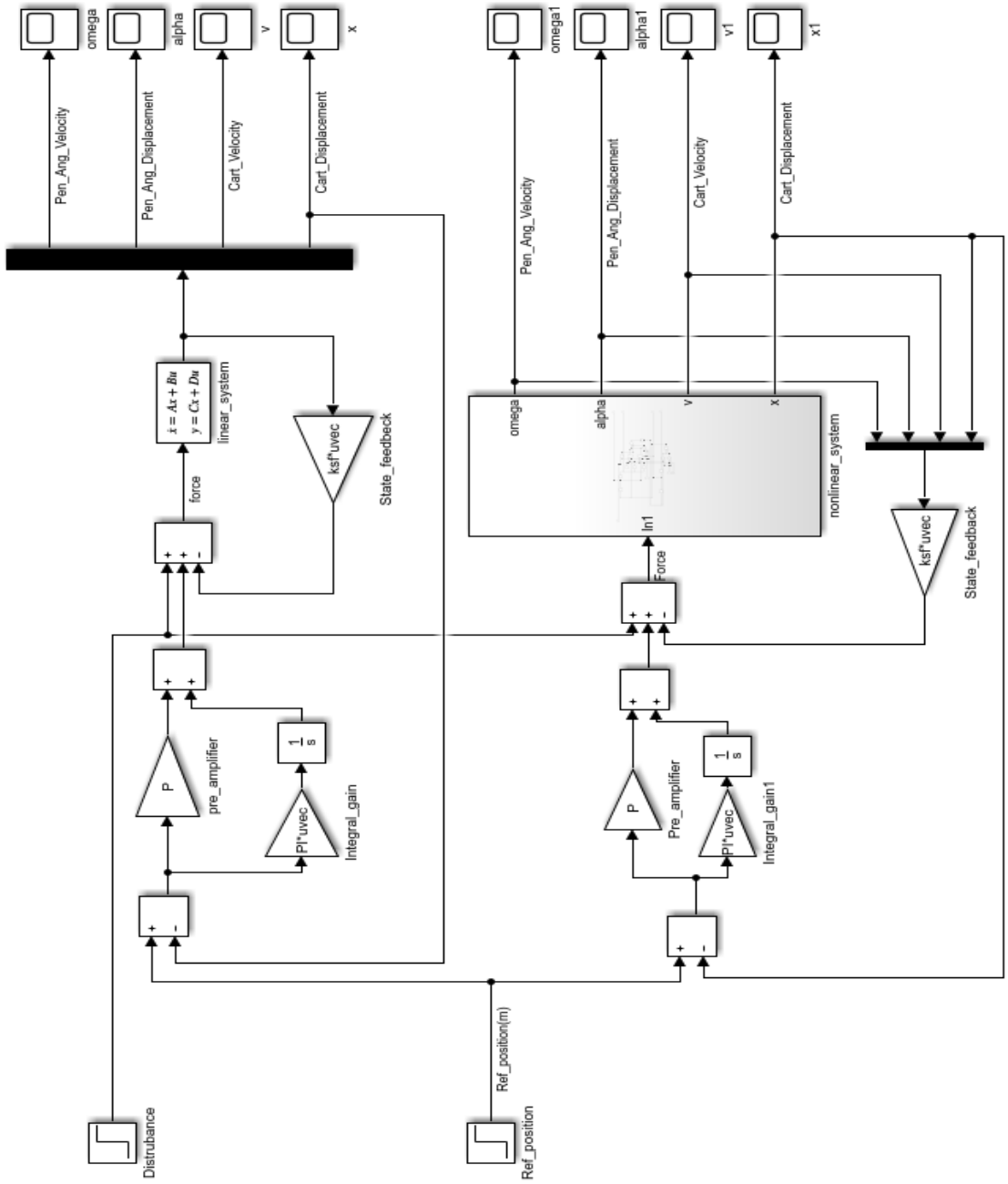


Figure 28:Continuous PI controller with disturbance effect

To study the effect of external disturbance on cart pendulum system, disturbance is applied to the pendulum angle. The below Figure illustrates continuous PI controller with disturbance added to pendulum angle. The disturbance is added for both linear and nonlinear models.

- Output graphs

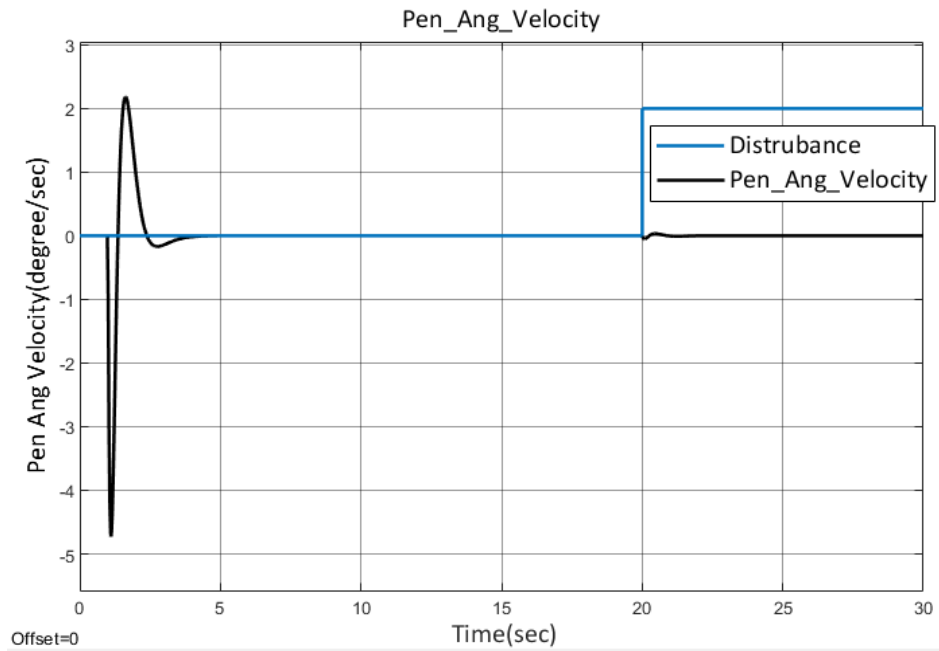


Figure 29: Disturbance effect on pendulum angular velocity (Linear continuous PI)

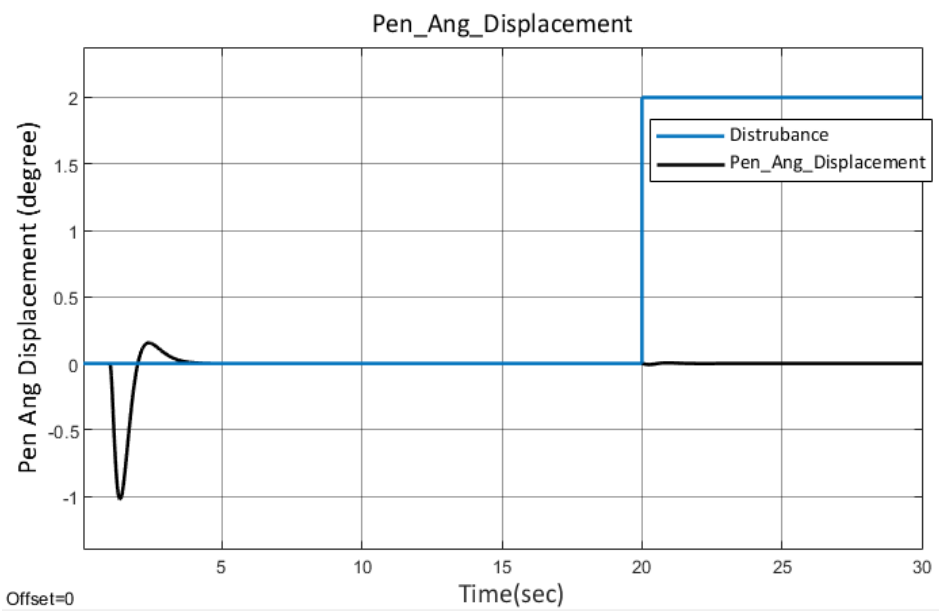


Figure 30: Disturbance effect on pendulum angle (Linear continuous PI)

Above Figure shows angular displacement and angular velocity of pendulum in the linear model. The disturbance of 20 second duration is applied. An external force induced with a value of 2, Even after disturbance is applied, the pendulum oscillation eventually settles down to 0 degree due to controller's effect.

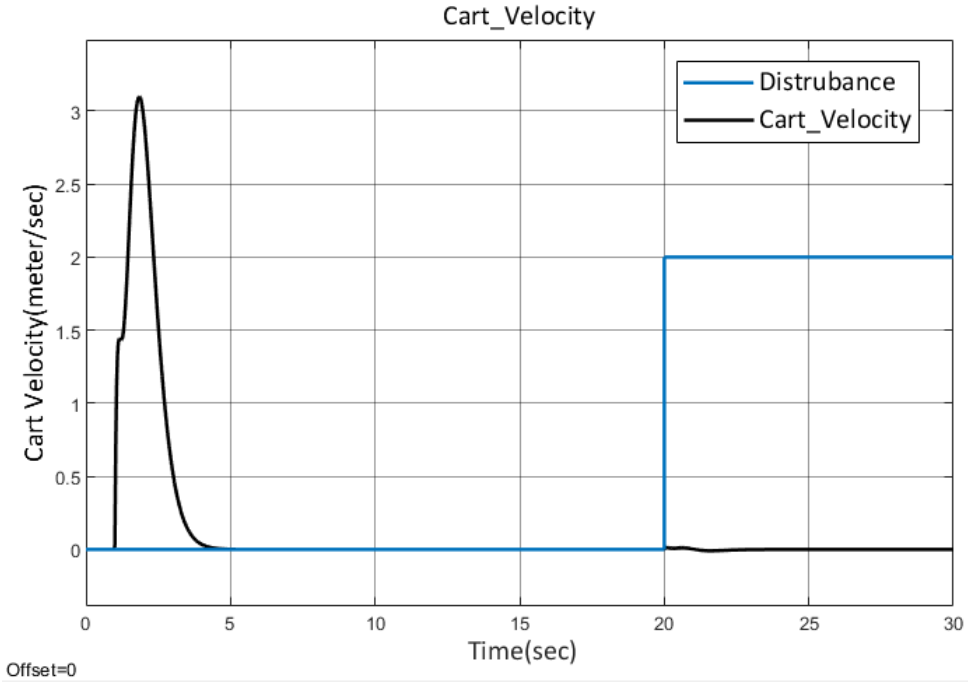


Figure 31: Disturbance effect on cart Velocity (Linear continuous PI)

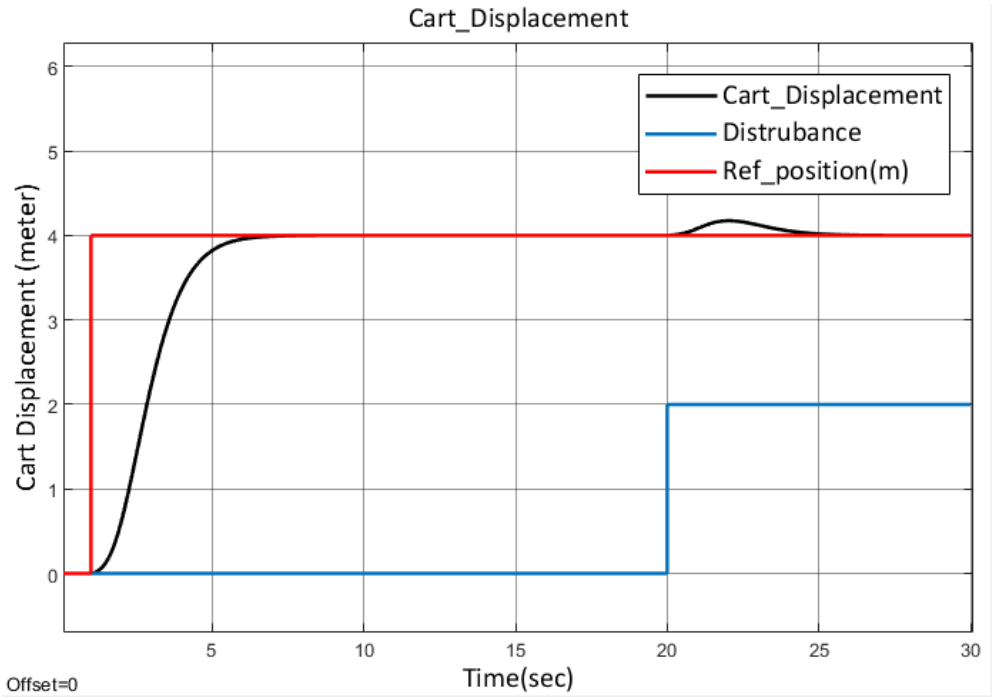


Figure 32: Disturbance effect on cart displacement (linear continuous PI)

Above Figure shows the cart displacement observed when disturbance to the pendulum is applied in the linear model. Reference input applied is 4m. The cart position oscillations around the reference position to control the pendulum oscillation and eventually settles at 4.

- For linear system

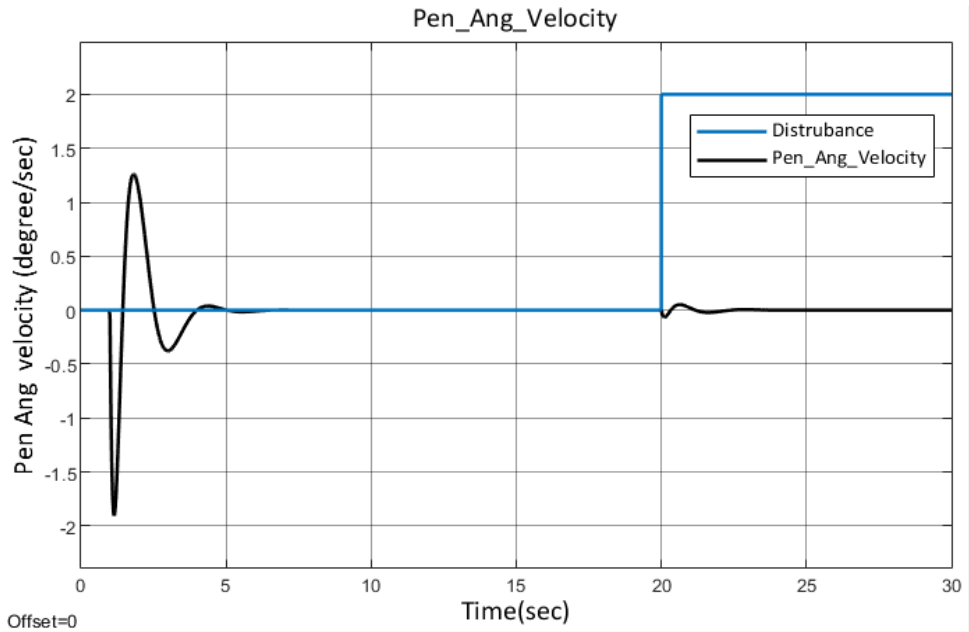


Figure 33: Disturbance effect on pendulum angular velocity (nonlinear continuous PI)

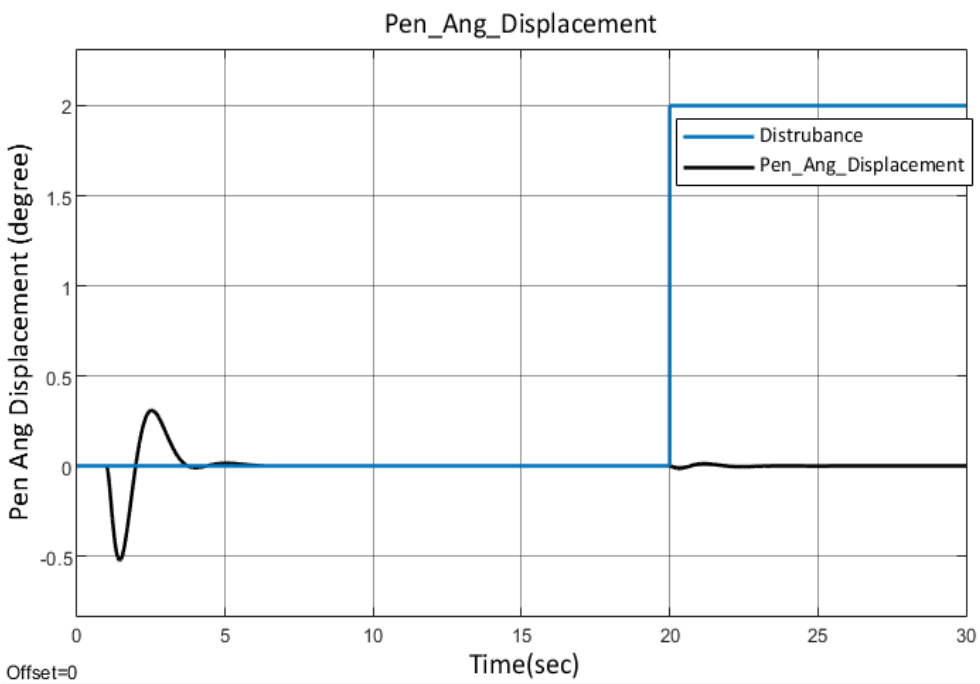


Figure 34: Disturbance effect on pendulum angle (Linear continuous PI)

Above Figure shows angular displacement and angular velocity of pendulum in the nonlinear model. The disturbance of 20 second duration is applied. An external force induced with a value of 2, Even after disturbance is applied, the pendulum oscillation eventually settles down to 0 degree due to controller's effect.

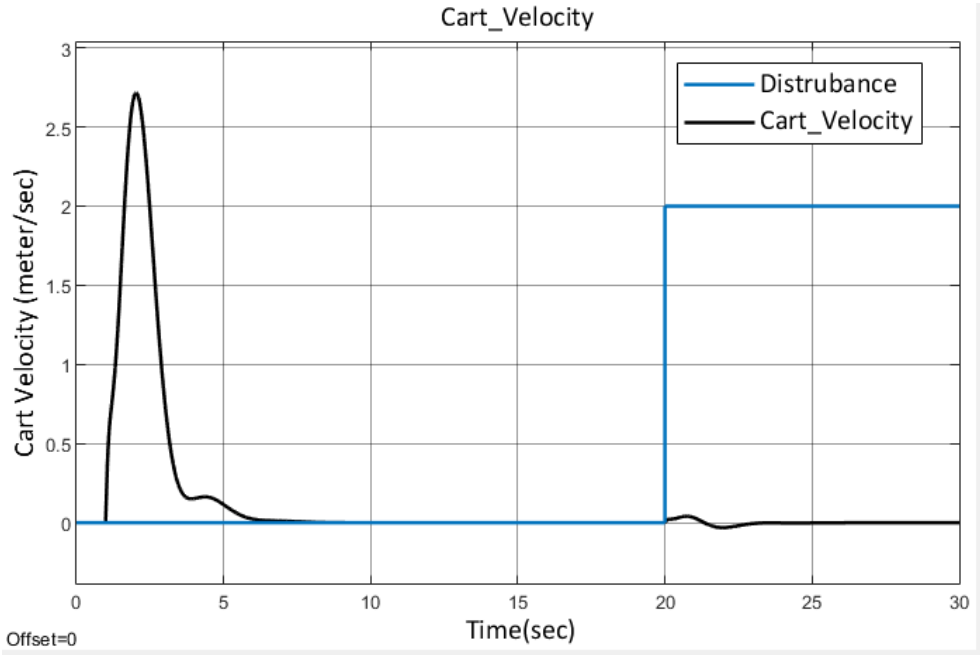


Figure 35: Disturbance effect on pendulum angle (nonlinear continuous PI)

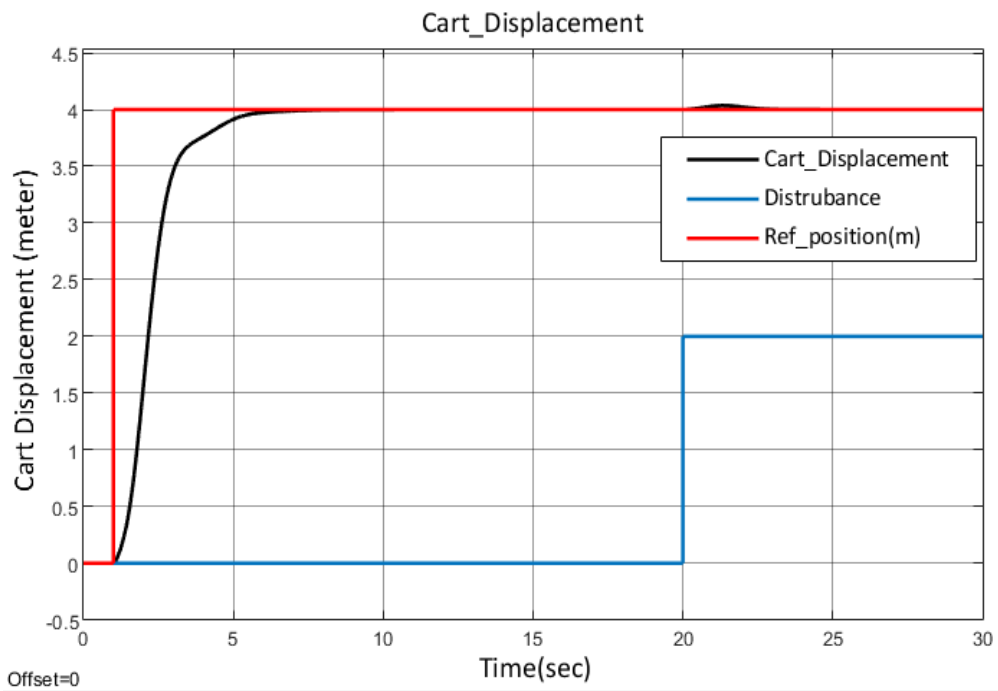


Figure 36: Disturbance effect on cart displacement (Nonlinear continuous PI)

Above Figure illustrates the cart displacement and velocity of nonlinear model with disturbances. Reference position of 4m is applied. The cart reaches the desired steady state accuracy in 5 secs.

6.4 Time Discrete PI Controller

Because digital systems are not continuous, the controller created for continuous time cannot be used for their implementation. Their foundation is sampling time. Hence, the controller must be transformed into its discrete form to be realized into any digital device. This is accomplished by employing "Zero Order Hold" and an appropriate sample period to transform a continuous time system into a discrete one. The same Ackerman formula can be used to calculate all discrete time gains. The continuous-time developed controller is converted to its equivalent.

discrete version using the following MATLAB script. Moreover, the continuous plant needs to be changed into its discrete form.

```
% State feedback with P Controller - Discrete

%sample period
Ts = 10e-3;
% Continuous to discrete system
sys1=c2d(ss(A,B,C,D),Ts);
% Accessing data of discrete system without integrator
[AD,BD,CD,DD]=ssdata(sys1);
% Discrete Controllability matrix
S_D = [BD AD*BD AD^2*BD AD^3*BD];
% Discrete Inverse of Controllability matrix
Si_D = inv(S_D);
% Discrete Last Row of Inverse Controllability matrix
q_D = Si_D(4, 1:4);
% Transfer function for desired poles
tf_p=tf([1],poly([pole pole pole pole]));
% Converting transfer function to discrete
tfd=c2d(tf_p,Ts);
[zeros_d,poles_d]=tfdata(tfd,'v') ;
%Ackermann?s formula for discrete system
K_D = q_D*(poles_d(5)*eye(4) + poles_d(4)*AD + poles_d(3)*AD^2
+ poles_d(2)*AD^3 + poles_d(1)*AD^4);
% Discrete Proportional gain value
P_D = 1/((ct*(eye(4)-(AD-BD*K_D))^-1)*BD);
```

```
% State feedback with PI Controller - Discrete

% Discrete system with integrator
sys = ss(A_pi,B_pi,C_pi,D_pi,0);
sys_d = c2d(sys,Ts);
% Accessing data of discrete system
[Adi,Bdi,Cdi,Ddi] = ssdata(sys_d);
% Controllability matrix
Sd=[Bdi Adi*Bdi (Adi^2)*Bdi (Adi^3)*Bdi (Adi^4)*Bdi];
% Rank of the controllability matrix
Rd=rank(Sd);
% Inverse of controllability matrix
Sd_i=inv(Sd);
% Last row of the inverse controllability matrix
qdi=Sd_i(5, 1:5);
% Converting transfer function to discrete
tf_pi=tf(1,poly([pole pole pole pole pole]));
tfd=c2d(tf_pi,Ts);
[zeros_d,poles_d]=tfdata(tfd,'v');
%Ackermann's formula for discrete system with integrator
Ktd= qdi*(poles_d(6)*eye(5) + poles_d(5)*Adi + poles_d(4)*Adi^2
+ poles_d(3)*Adi^3 + poles_d(2)*Adi^4 + Adi^5);
% Value of state-feedback
kt_d = Ktd(1:4)-(P_D*ct);
% Discrete integrator gain
pi_d = -Ktd(5);
```

The discrete system's sample period is Ts = 10 ms. It is also necessary to translate our chosen pole in the s-domain into its equivalent poles in the z-domain because the system is discrete. There isn't a big difference between the gain levels in continuous time and discrete time because the sampling time is tiny enough. The gain values change into,

State feedback gain vector kt_d = [-6.1571 -99.6894 61.2623 126.7680]

Proportional gain P = 34.8123

Integral gain pi_d = 126.7522

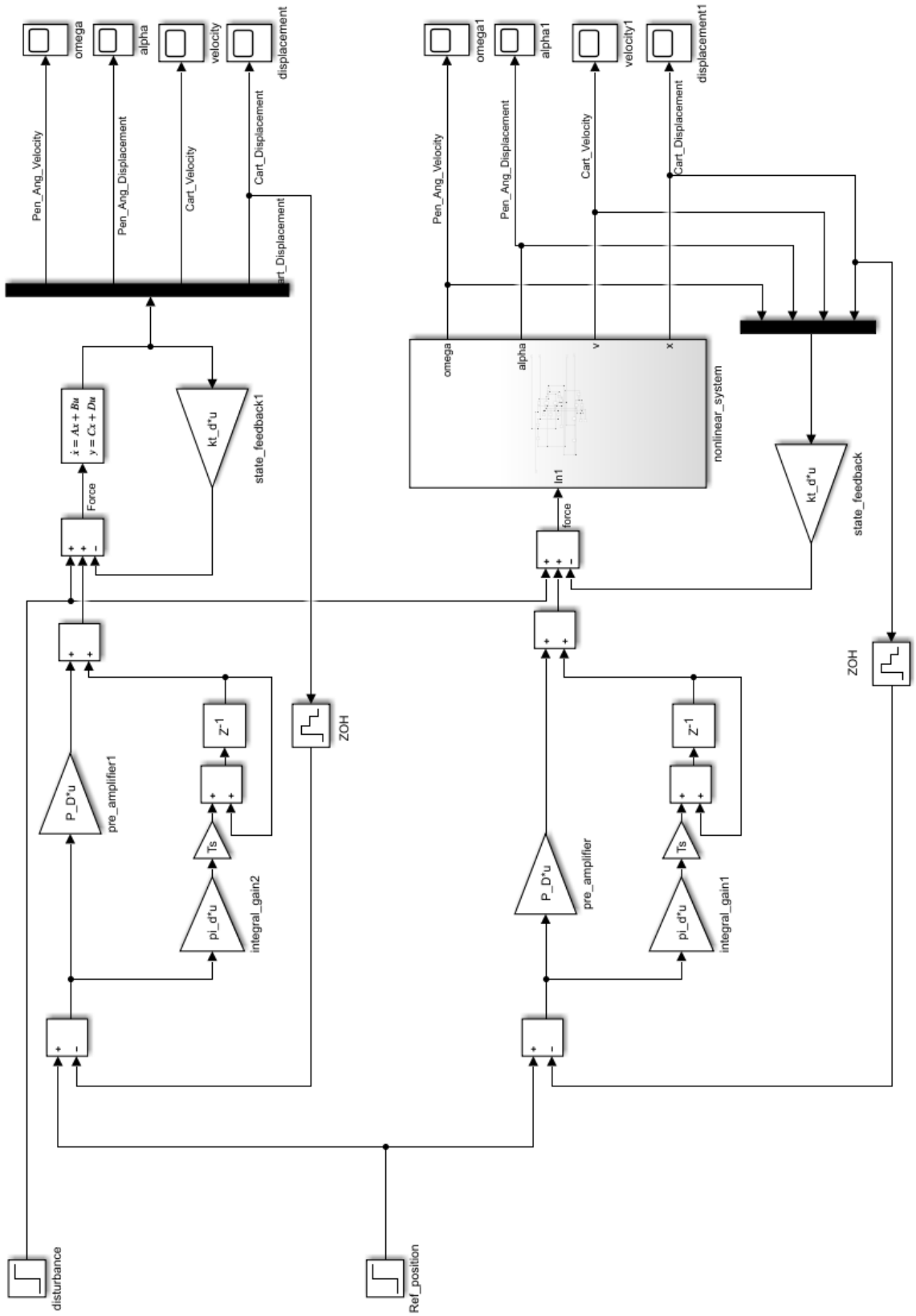


Figure 37: Discrete PI controller

Graph for linear system

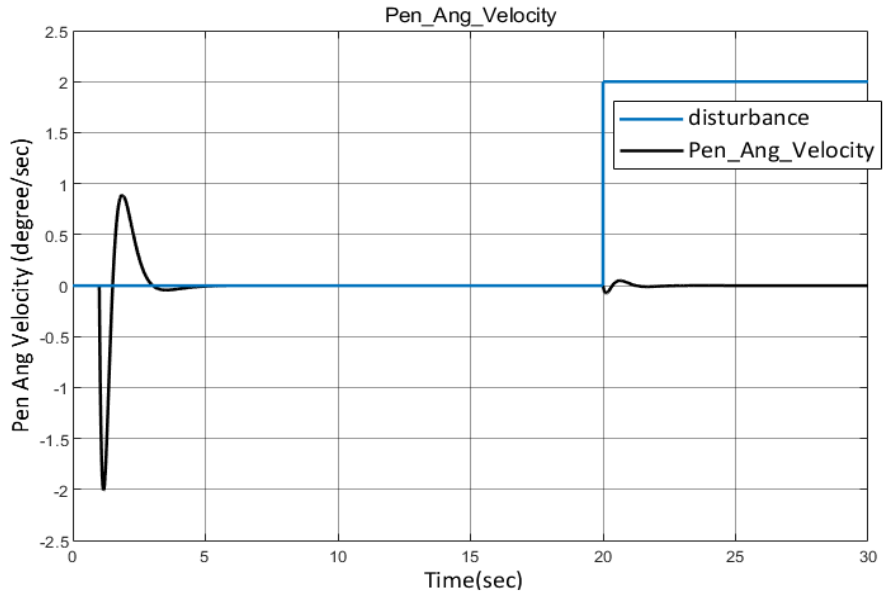


Figure 38: Linear angular velocity of pendulum with disturbance (Discrete PI)

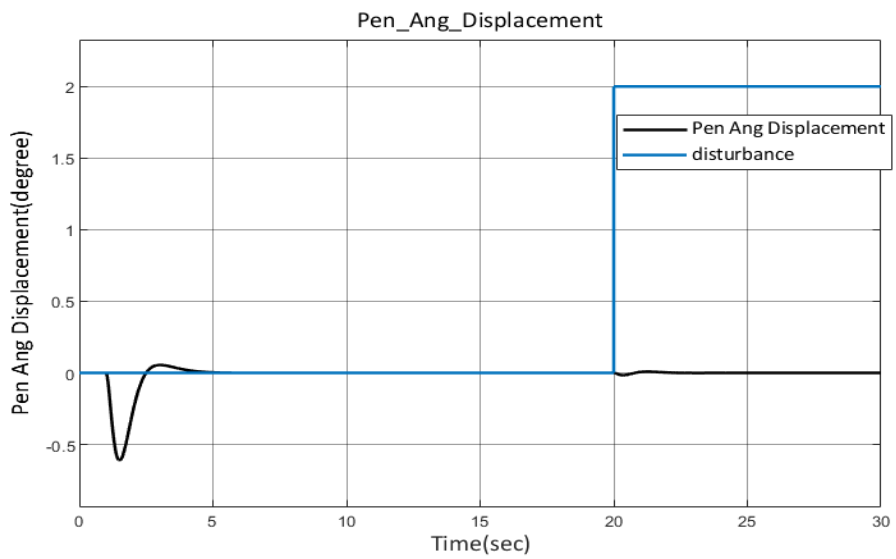


Figure 39: Linear angular displacement of pendulum with disturbance (Discrete PI)

Figures show the angular displacement and velocity of the pendulum of linear model. The graphs are like the time continuous controller. Reference position of 4m is applied. The pendulum oscillation gradually settles to 0 degree due to controller's effect.

And observed when disturbance to the pendulum is applied in the linear model. No reference input is applied. The cart position oscillates around the reference position to control the pendulum oscillation and eventually settles at zero.

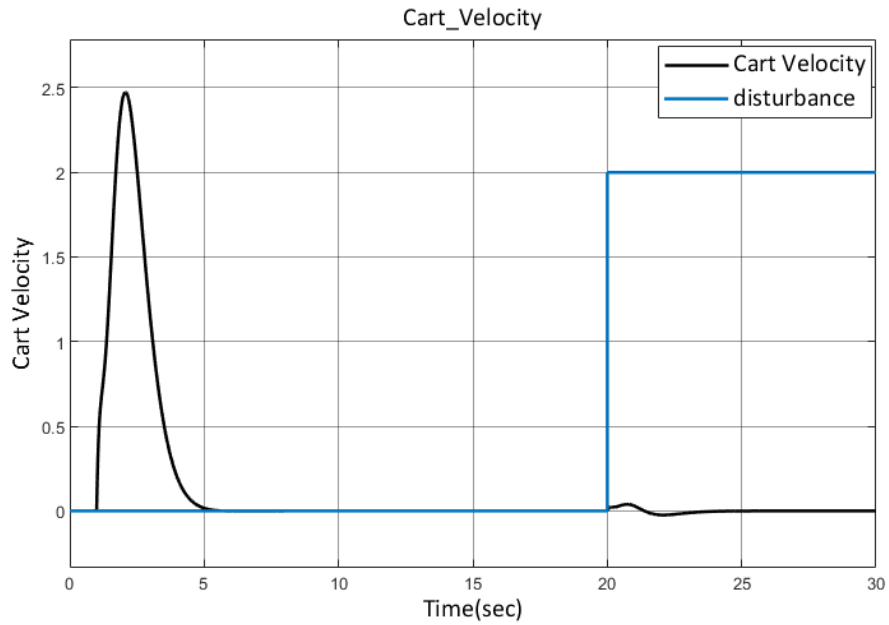


Figure 40: linear cart velocity with Disturbance (Discrete PI)

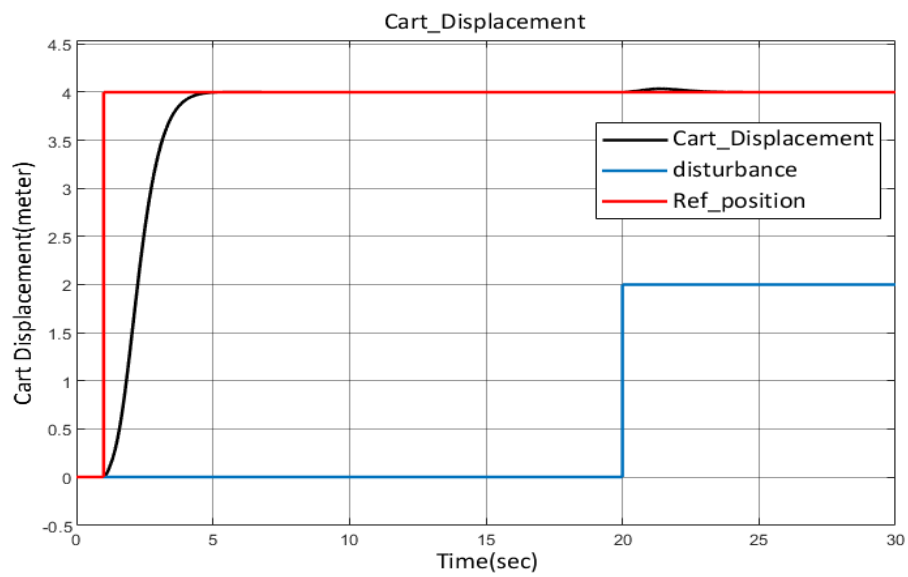


Figure 41: Linear cart displacement with Disturbance (Discrete PI)

Figures shows the cart displacement and velocity of linear model. The graphs are similar to the continuous controller. Reference position of 4m is applied. The cart reaches to desired steady state in 4 seconds.

Figure 41 shows the cart displacement observed when disturbance to the pendulum is applied in the linear model. No reference input is applied. The cart position oscillates around the reference position to control the pendulum oscillation and eventually settles at zero.

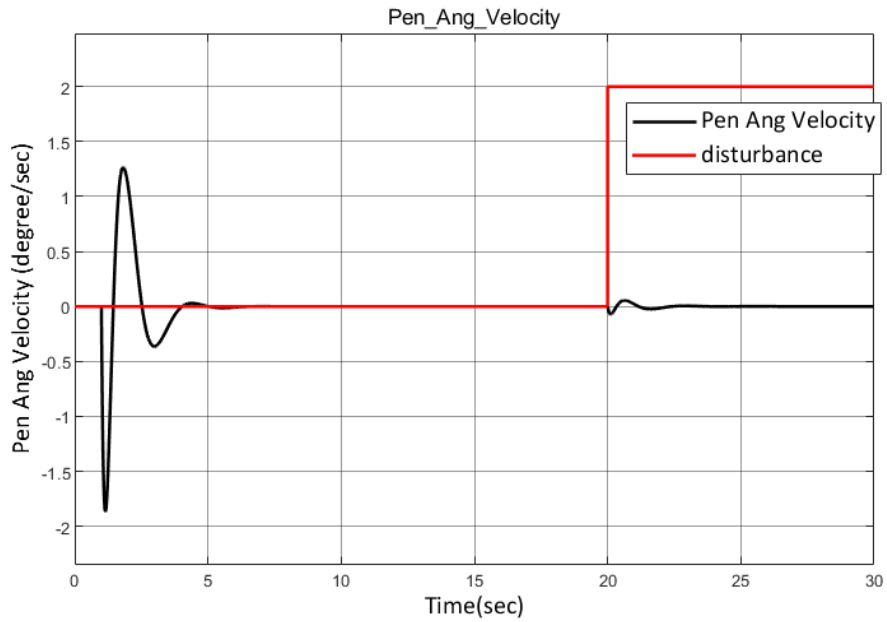


Figure 42: Nonlinear angular velocity of pendulumwith Disturbance (Discrete PI)

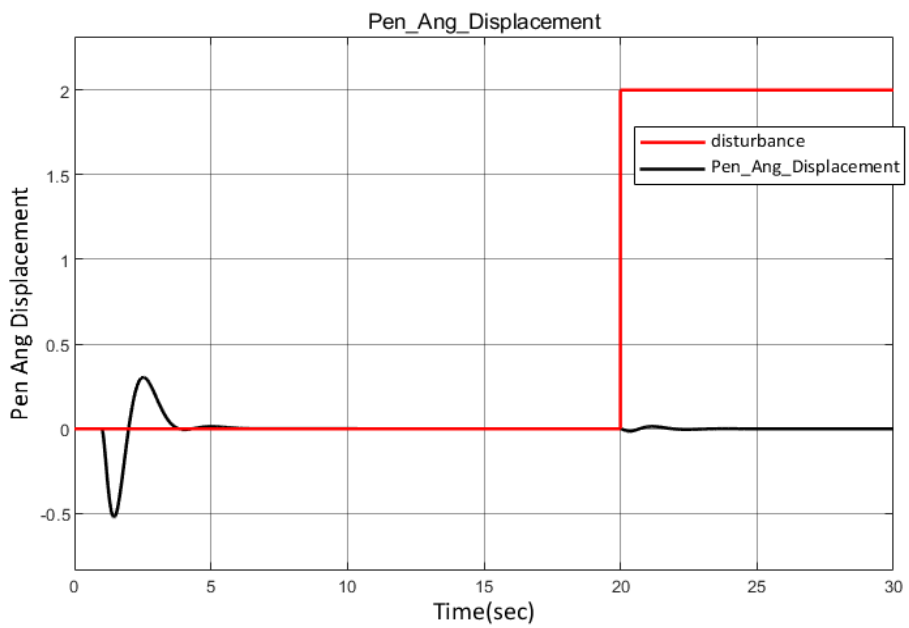


Figure 43: Nonlinear angular displacement of pendulum with Disturbance (Discrete PI)

Figures show pendulum angular displacement and velocity of nonlinear model with disturbances. Reference position of 4m is applied. The graphs are like continuous controller. Pendulum oscillation settles to zero degree due to controller effect.

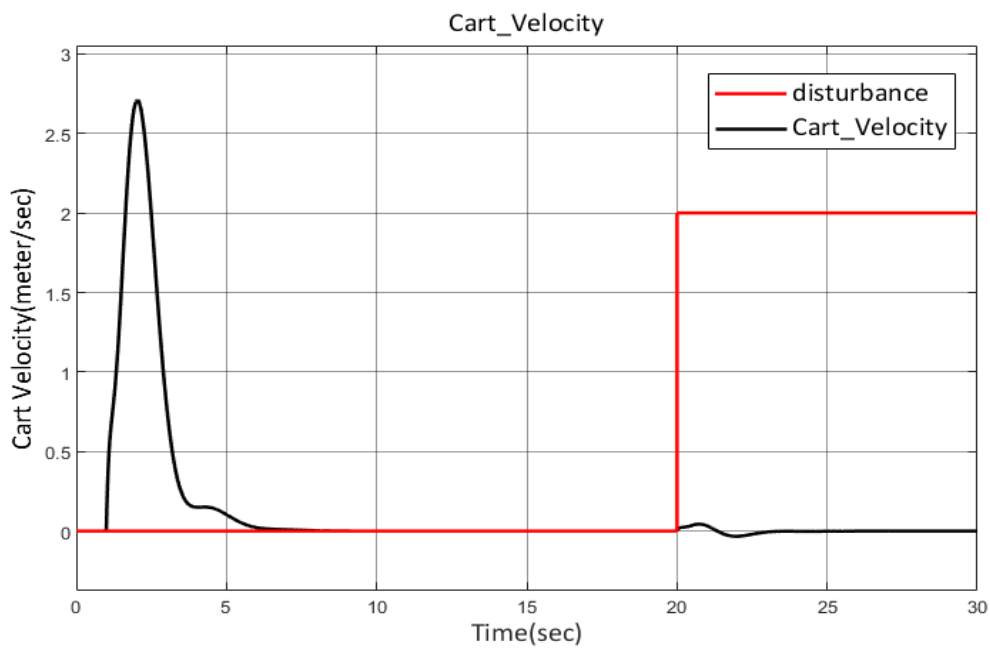


Figure 44: Nonlinear cart velocity with Disturbance (Discrete PI)

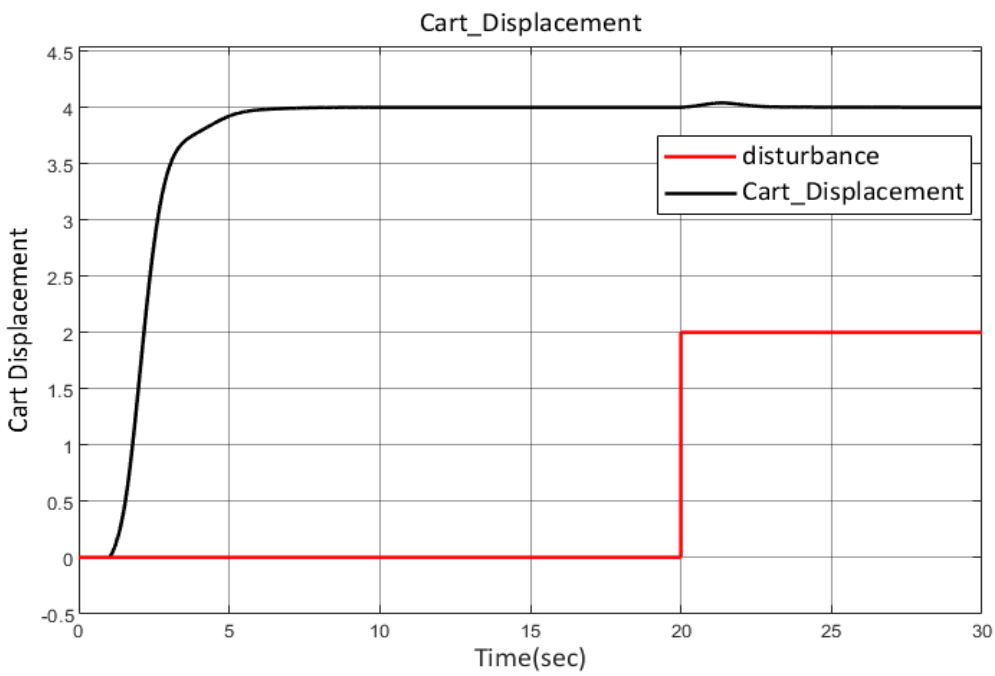


Figure 45: Nonlinear cart displacement with Disturbance (Discrete PI)

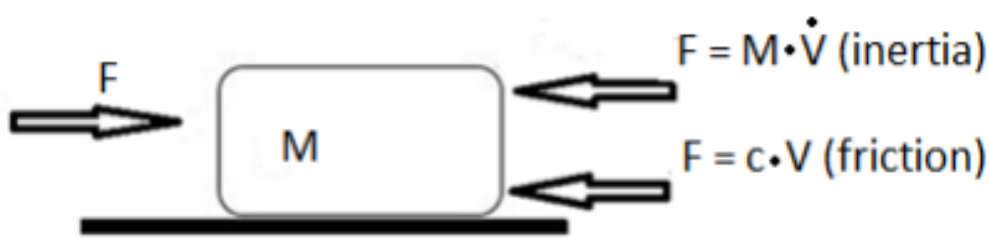
Figures 44 and 45 show the cart displacement and velocity of nonlinear model. The graphs are like the continuous controller. Reference position of 4m is applied. The cart reaches the desired steady state in 5 seconds.

7 Cart Speed Controller

The cart pendulum test stand consists of two limit switches, one on both ends. Initially when the input is given to system, the cart position is unknown. In order to know the position of the cart the reference phase is performed. During this phase the cart must move with constant low speed in order to detect the limit switches properly. If the speed of the cart is not controlled there are chances that the cart would go hit the limit switches hard and there could be safety issues. To avoid this, a speed controller is designed. To evaluate the speed control of the cart pendulum system a simple physical model is designed by neglecting the mass of a pendulum. A mathematical model is calculated by applying Newton's equation of motion.

7.1 Mathematical Modelling of Cart Speed Controller

A simple free-body diagram of a cart pendulum is drawn as shown in the figure. The direction of the arrow shows the direction that the force is acting upon a cart. According to Newton's second law, the sum of the forces is equal to zero.



The parameters of the cart pendulum systems are as follows:

M: Mass of the cart (kg)

c : Friction coefficient (kg/s)

V : Velocity of the cart (m/s)

F : External force applied to the cart (N)

From the Newton's equations of motion,

$$F - c \cdot V - M \cdot (dV/dt) = 0$$

$$M \cdot (dV/dt) = F - c \cdot V$$

In Laplace domain the equation is

$$M \cdot s \cdot V = F - c \cdot V$$

$$V = \frac{1}{M \cdot s + c} \cdot F$$

Then our transformation gain is

$$G = \frac{1}{M \cdot s + c}$$

Where

$$G = \frac{V}{F} \qquad [1]$$

Our total gain K is the sum of both proportional gain P and integral gain PI which is multiplied by the integral value 1/s(s-domain) which is given as:

$$K = P + \frac{PI}{s}$$

$$K = \frac{Ps + PI}{s} [2]$$

From equation 1 and 2

$$K \cdot G = \frac{Ps + PI}{s} \cdot \frac{1}{M \cdot s + c}$$

Now dividing the first term by PI and second term by 1/c

$$K \cdot G = \frac{\frac{P}{PI} \cdot s + 1}{\frac{s}{PI}} \cdot \frac{1}{C} \cdot \frac{1}{\frac{M}{c} \cdot s + 1}$$

By comparison we get,

$$\frac{P}{PI} = \frac{M}{c}$$

$$PI = P \cdot \frac{c}{M}$$

7.2 Time Continuous PI Controller

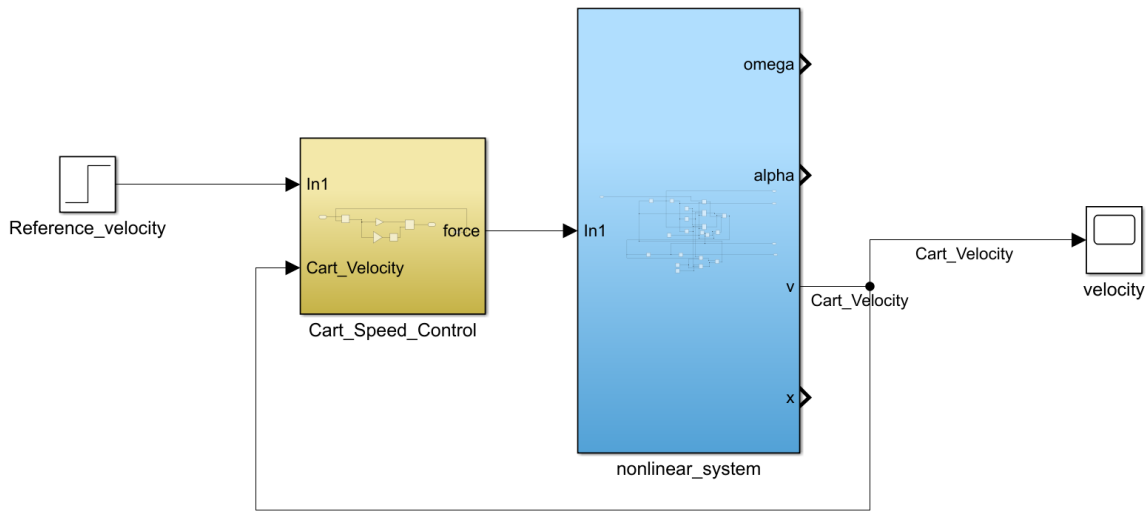
7.2.1 MATLAB Code

```
%Model Parameters
M = 4.676;        %kg% Mass of the cart
m = 0.098;      %kg% Mass of the pendulum
d = 0.285;      %m% length of the pendulum rope
g = 9.8;      %m/s^2% gravity
c=5;        %friction constant

%Speed controller
P_S = 34.8123
PI_S = P*(c/M);
```

7.2.2 Simulink Model

Reference velocity is given input to the controller. The state cart velocity is feedback to the controller. The output is observed using the graph below.



7.2.3 Output graph

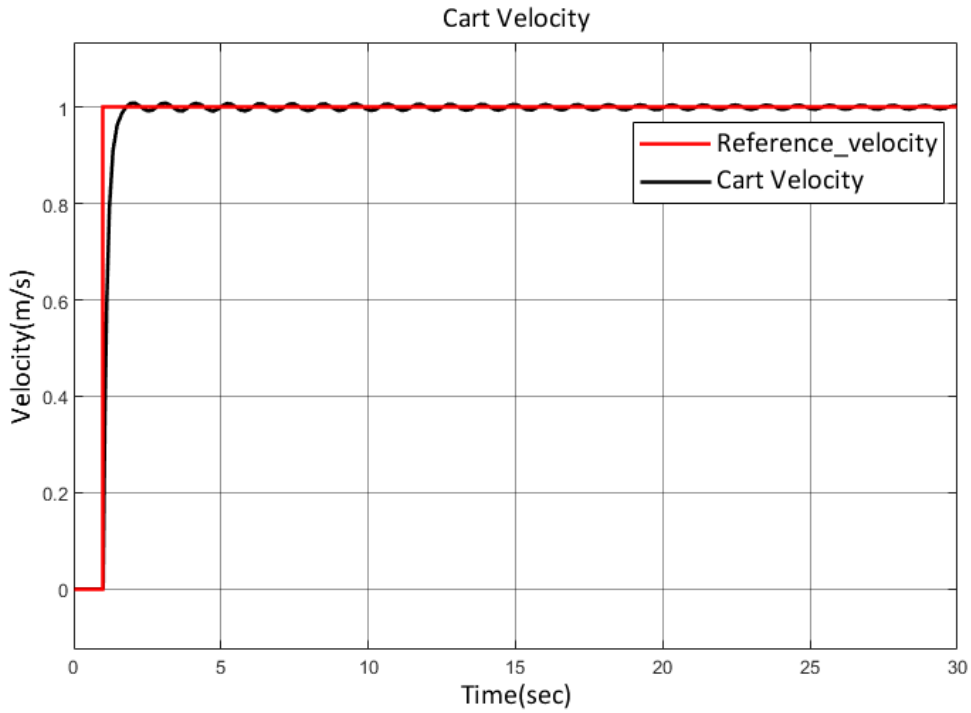


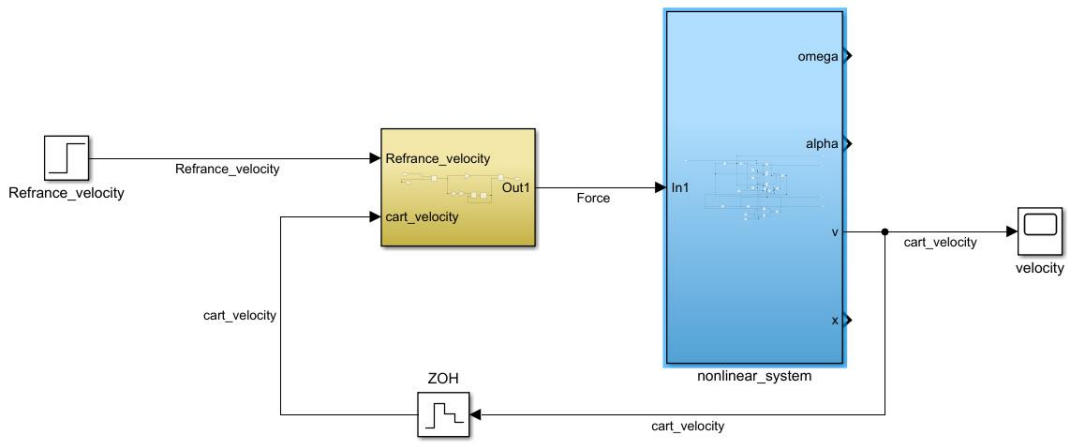
Figure 46:Cart velocity (Continuous cart speed controller)

Reference velocity of 1m/sec is applied, and it is observed that the cart velocity reaches reference velocity within 2 sec.

7.3 Time Discrete PI Controller

The figure shows the discrete speed controller model. The continuous speed controller is discretized using ZOH. Reference velocity is given input to the controller. The output is observed using the graphs below. Sampling time Ts = 0.01 is used.

7.3.1Simulink Model



7.3.2 Output graph

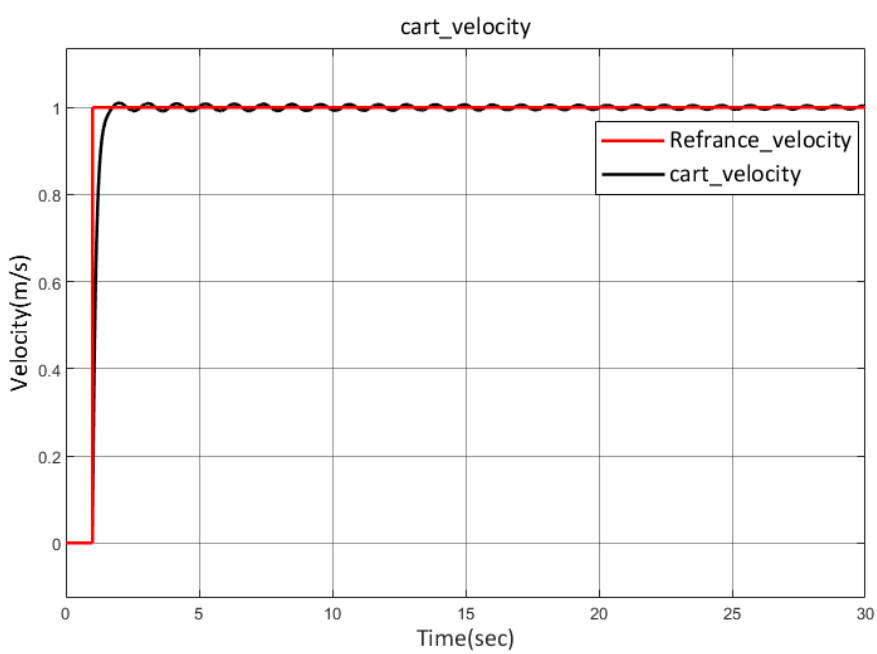


Figure 47:Cart velocity (Discrete cart speed controller)

In this graph Reference velocity of 1m/s is applied and cart reach this velocity within 2 sec.

8Hardware Components

The physical components are an integral part of any embedded system. Understanding the interaction between these hardware elements, whether with software or other hardware, is crucial for the development of a real-time application.

The Pendulum Cart system's parts and their exact locations on the test stand are shown in the figure below.

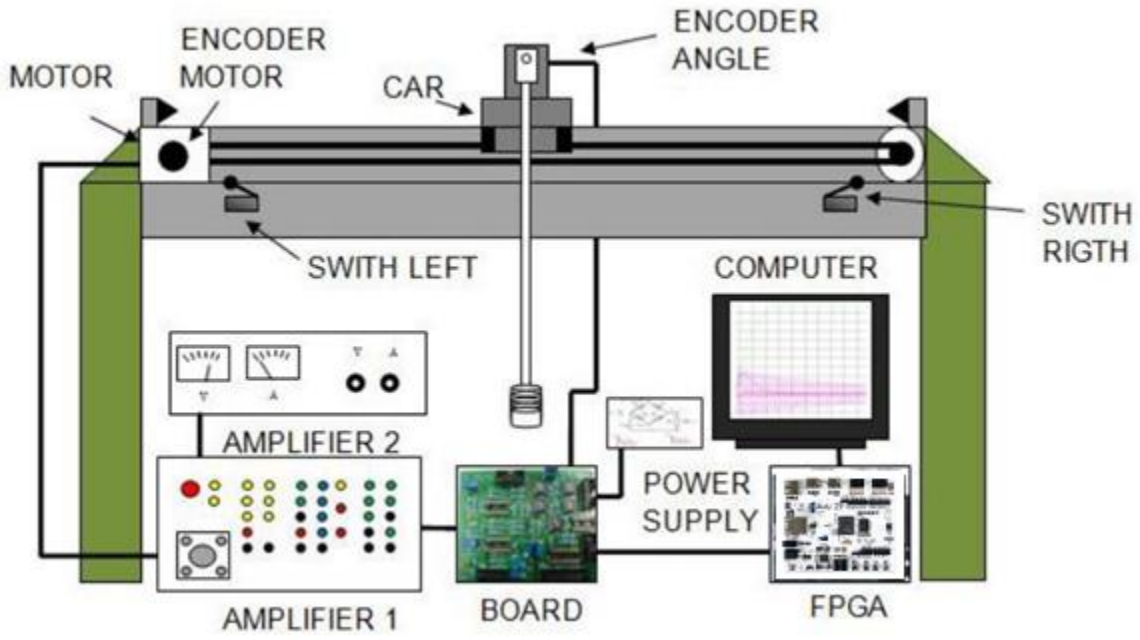


Figure 48:Hardware setup of Cart Pendulum System [10]

In this setup, the aim is to move the carriage to a certain position without triggering pendulum oscillations, considering the attachment of the pendulum to the carriage. Therefore, the cart can move in both ways and is restricted by limit switches at both ends of the cart path.

The user can control the position of the carriage by sending commands via the PC. The Arty Z7 processes these commands and determines the force signal required to reach the required position. This force signal is then transmitted via the violin amplifier to the motor, which adjusts the motor current.

Feedback on the position of the cart and the angle of the pendulum is provided by the motor or pendulum encoder. Communication takes place via the hardware interface card between the system-on-chip (SoC) and other components in the carriage pendulum system.

All hardware components and connections through the interface cardboard are shown in the Figure below.

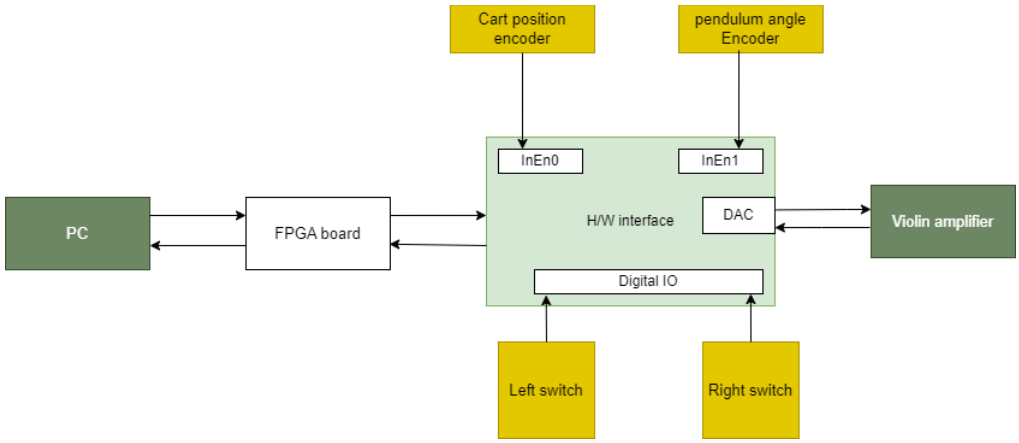


Figure 49: Hardware components and their interface

8.1 Incremental Encoder

An incremental encoder is a rotating instrument that responds to the movement of a rotating object with pulses. It can contain a reference mark signal (Z) and usually provides position and direction information via two output channels (A and B). It is commonly used in robotics, automation, and CNC systems where it is required for applications that require accurate position or velocity feedback.

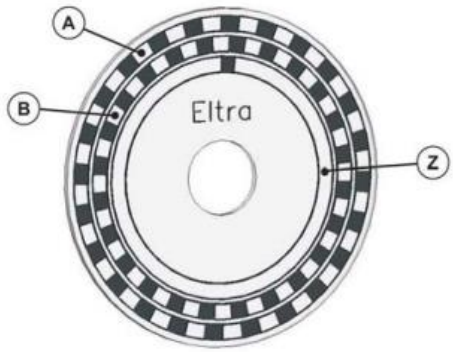


Figure 50: Incremental Encoder [2]

The servomotors are equipped with a built-in encoder. The incremental encoder generates an output signal that provides details about the movement of the shaft and is normally processed to derive information such as position, speed, and distance. The standard output signals of the incremental encoder are illustrated in the figure below.[2]

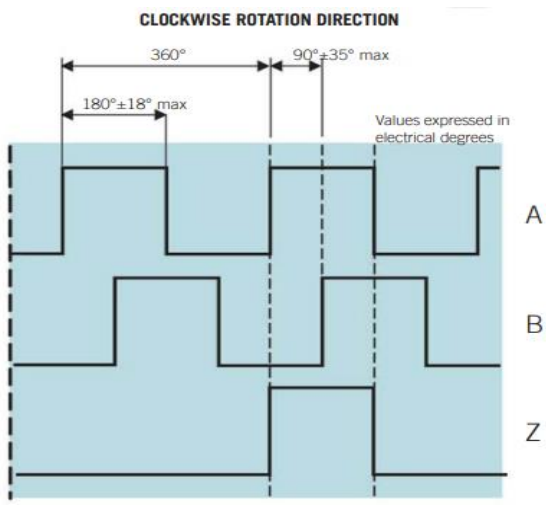


Figure 51:Output signals of the incremental encoder [2]

8.2 Violin Amplifier

The servo amplifier operates in two modes, with the current mode being used specifically for torque control. In this mode, the amplifier calculates the exact amount of current (measured in amps) supplied to the motor. For a DC motor, the current corresponds directly to the torque, allowing the amplifier to effectively control the torque applied to the motor.

The amplifier is controlled by two input signals: Amplifier Enable and Current Command Input, both of which are used to control amplifier operation. Furthermore, an output signal called Amplifier OK is generated by the amplifier to indicate the operating status of the amplifier.

8.3 Linear Actuator LEZ3

The LEZ3 linear actuator is equipped with a toothed belt drive. The linear guide rails and the associated shaft carriages serve as the basis for this linear actuator. The carriage of the linear actuator, the so-called carriage, moves to the desired position specified by the user.

Figure 52: LEZ3 linear actuator [3]

A toothed belt serves as the drive for positioning the cart. In comparison to conventional chain drives, the toothed belt requires no maintenance and no lubricant. A linear guide rail allows the upper run of the carriage to be supported, resulting in greater accuracy when positioning long axis lengths. There are rubber teeth on the inside of the belt that engage with a toothed wheel. The positive locking of the teeth allows high forces to be transmitted.

8.4 DC Servo Motor

A servo motor, which is designed for the high-precision rotation of machine parts, works with an electric motor, and contains a control circuit. This circuit provides feedback on the current position of the motor shaft and thus enables precise rotations. Servo motors are advantageous for rotating objects at specific distances or angles.

A servo motor can be of two main types: AC servomotors and DC servomotors. A servo motor is known as a DC servo motor if it is operated with DC power and as an AC servo motor if it is operated with AC power.

Motor MV 300	
Rated power	330 W
Rated speed	3300 (min-1)
Rated torque / peak torque	100 / 539 Nm
Rated current / peak current	6.5 / 30 A
Motor Encoder	
Power Supply	5 V DC
Resolution	1000 PPR

Table 1: Parameter of Motor MV 300 [4]

8.5 End Switches

This switch uses a mechanically driven lever to detect the position of a moving element and triggers an electronic signal when it reaches its end position. Typically, 3 of these energy efficient switches (with an option for up to 6) are connected to moving parts to detect the origins of the X, Y and Z axes. The limit switch can be operated with both direct and alternating current up to 125 V and can be securely attached to other components using two small holes. The screws required for fixing the limit switch must be available.

Figure 53: End Switches [5]

8.6 Arty Z7 Board

The Arty Z7 is a pre-configured development platform that is based on the Zynq-7000™ All Programmable System-on-Chip (AP SoC) from Xilinx. The Zynq-7000 architecture smoothly integrates a dual-core, 650 MHz ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. This combination allows a robust processor to be surrounded by a range of software-defined peripherals and controllers that can be customized for the specific application.

The Vivado, Petalinux and SDSoC toolsets each offer an easily accessible way to define custom peripherals and integrate their functionality into a Linux operating system or bare-metal program that runs on the processor. For those that prefer a more conventional digital logic design, however, it is also an option to bypass the ARM processors and program the Zynq FPGA similarly to any other Xilinx FPGA.

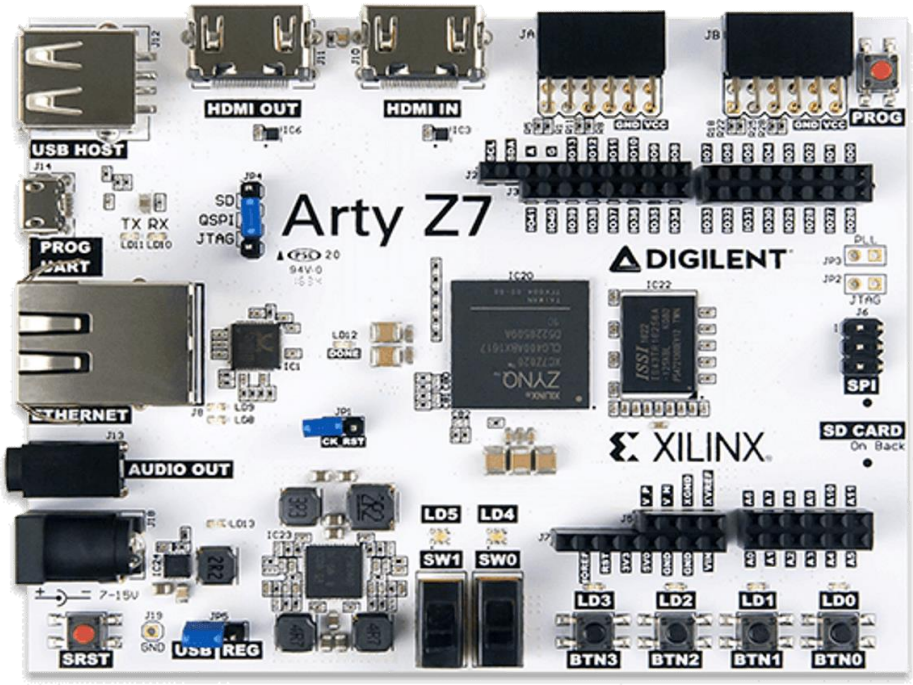


Figure 54: Arty Z7 board [6]

- Key features of Arty Z7 SoC:
  - Dual-core Cortex-A9 processor running at 650MHz
  - Peripheral controller with low bandwidth: SPI, UART
  - Peripheral controllers with high bandwidth: USB 2.0, SDIO
  - Powered through USB or any external power source within the range of 7V-15V
  - Programmable via JTAG, Quad-SPI flash, and microSD card
  - USB-UART bridge circuit
  - Circuitry for USB-JTAG programming

A system on chip (SoC) is a single integrated circuit such as the Zynq-7020, which consists of all the classic digital components: processor, memory, bus, etc.

- Application Processor Unit (APU):

  The Application Processor Unit (APU) offers a comprehensive range of high-performance functions and standards-compliant capabilities. It has two ARM Cortex-A9 MP Core CPUs with ARM v7 architecture:
  - 32 KB instruction and 32 KB data L1 caches with parity per MPCore
  - The Snoop Control Unit (SCU) ensures the maintenance of L1 and L2 coherency.
  - System-Level Control Registers (SLCRs) form a group of different registers that control the PS behavior
  - 512 KB shared L2 cache with parity

- Accelerator Coherency Port (ACP) from PL (master) to PS (slave) with a 64b AXI slave port. It can access the L2 and OCM, and the transactions are data coherent with L1 and L2 caches.
- 256 KB on-chip SRAM (OCM) with parity.
- General Interrupt Controller (GIC)

➢ Memory interfaces:

Quad SPI controller (which is the one used in the cart pendant system)

- I/O peripherals (IOP):
- The I/O peripherals (IOP) are a set of interfaces for external data communication.
- GPIO
- USB controllers
- UART control
- PS MIO I/O

Programmable logic (PL) offers an architecture with user-configurable functions. The most important features include:

➢ Configurable Logic Blocks (CLB):

A CLB element consists of a pair of slices, each slice containing four lookup tables (LUTs) with 6 inputs and eight memory elements.

➢ 36 KB block RAM:

The key features of block RAM are:

- Dual-port 36 KB block RAM with port widths of up to 72

➢ Configurable I/O:

- Includes High-Performance Select I/O technology.
- High-Range (HR) I/O supports a voltage range from 1.2V to 3.3V.

The SPI Quad Flash with a memory capacity of 16 MB is used to store the software for the controller. This SPI flash memory serves as non-volatile memory for code and data, for initializing the PS subsystem and for configuring the PL subsystem. Relevant features of the device include its 16 MB capacity and support for Zynq configuration rates at 100 MHz.

9Hardware Interface Board

There are integrated components in the hardware interface circuit that enable an input and output interface. This circuit is an essential component that connects the system on chip (SoC), the limit switches, the violin amplifier and the incremental encoders.

The CMOS (Complementary Metal-Oxide-Semiconductor) and TTL (Transistor-Transistor-Logic) technologies are clearly being utilized in the cart pendant system. A CMOS technology is used in the SoC for its low power consumption, high noise immunity, fast performance and long rise and fall times. It operates at a 3.3V level. These characteristics make digital signal processing in the CMOS architecture easier. Conversely, components operating in the 5V range choose TTL technology because of its lower cost compared to CMOS. As the system uses a motor that is susceptible to electromagnetic interference, TTL is a better option for connecting components than CMOS as it is more robust against this type of interference.

The data is converted into the required format for the hardware component in such a way that communication can take place between all components. A DAC converts digital data from the SoC into an analog signal, as the violin amplifier requires an analog signal.

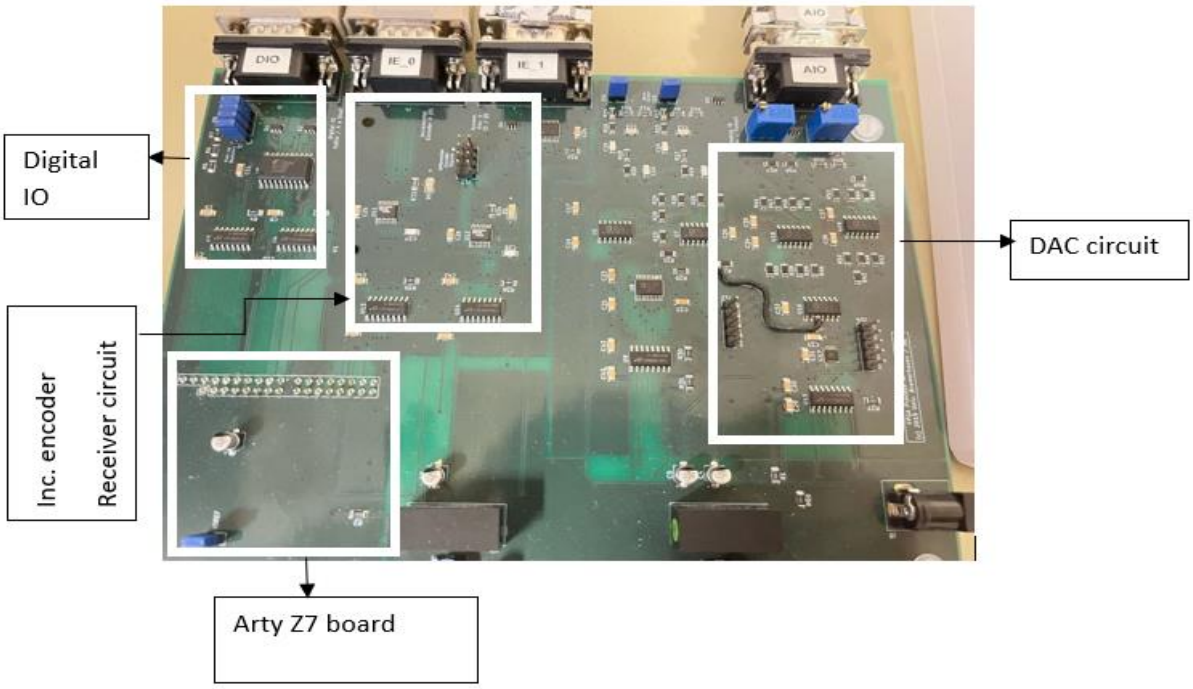


Figure 55: The interface of the interface board and the components

9.1 Digital to Analog Conversion (DAC)

In this application, the DAC AD5624R is used. The DAC fulfills two main functions in this context: First, it increases the light intensity via its four output channels, and second, it provides an internal reference voltage to the ADC.

To work effectively with the DAC, the FPGA must transmit the appropriate data bits to the DAC.

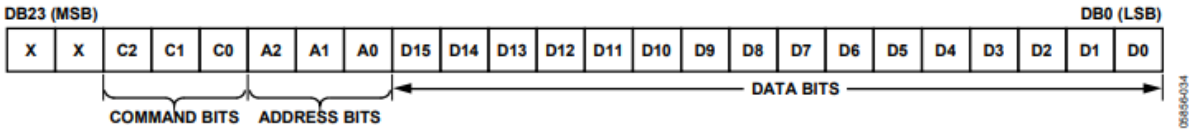


Figure 56: DAC Input Shift Register [7]

C2	C1	C0	Command
0	0	0	Write to input register n
0	0	1	Update DAC register n
0	1	0	Write to input register n, update all (software LDAC)
0	1	1	Write to and update DAC channel n
1	0	0	Power down DAC (power-up)
1	0	1	Reset
1	1	0	LDAC register setup
1	1	1	Internal reference setup (on/off)

Table 2: Command Definition of the DAC [7]

In the table below, the list of operating and input shift registers bits to be sent is shown. For more detailed information, please refer to the AD5624R data sheet (see reference).

Sr.No.	Operation	Input Shift register Bits
1	Reset DAC	0x280001
2	Power-Up DAC	0x20000F
3	LDAC Register Mode	0x300000
4	Internal Reference	0x380001

Table 3: DAC Input bits

9.2 Analog-to-Digital Convertor (ADC)

The AD7886 is a 12-bit ADC featuring a sample-and-hold amplifier, delivering a blend of high-speed performance and low power consumption. Utilizing a triple-pass flash ADC design with 15 comparators in a 4-bit flash technique, the AD7886 achieves 12-bit accuracy with a rapid conversion time of 1 µs/750 ns. An on-chip clock oscillator is integrated, providing the necessary timing for the three conversion stages, eliminating the requirement for external clocks. The sample-and-hold amplifier's acquisition time results in an effective throughput rate of 750 kHz/1 MHz.

In this project 16 bits of data are required, so two ADCs can be used here.

Operational within ±5 V power supplies, the AD7886 offers pin-strappable inputs that provide flexibility with three analog input ranges: 0 V to 5 V, 0 V to 10 V, or ±5 V. [8]

9.3 Up & Down Counter

9.3.1 Up Counting

The 90-degree phase shift between the signals of channels A and B provides additional insights into each pulse cycle. Essentially, each pulse cycle can be dissected into four distinct states. The diagram below illustrates a pulse cycle transmitted by the incremental encoder.

The pulse pattern shown in the diagram designates the up-pulse as one. To put it simply, the counter increments when channels A and B produce the same sequence of zeroes and ones simultaneously. The table provided below documents the flip-flop states for channels A and B during the incremental action.

9.3.2 Down Counting

The up-down counter undergoes decrementing when the incremental encoder transmits a pulse pattern resembling the figure below.

This pulse pattern designates the dn-pulse as one. Simply put, the counter decreases when channels A and B simultaneously generate the same sequence of zeros and ones. The table provided below documents the states of the flip-flops for channels A and B during the decremental action.

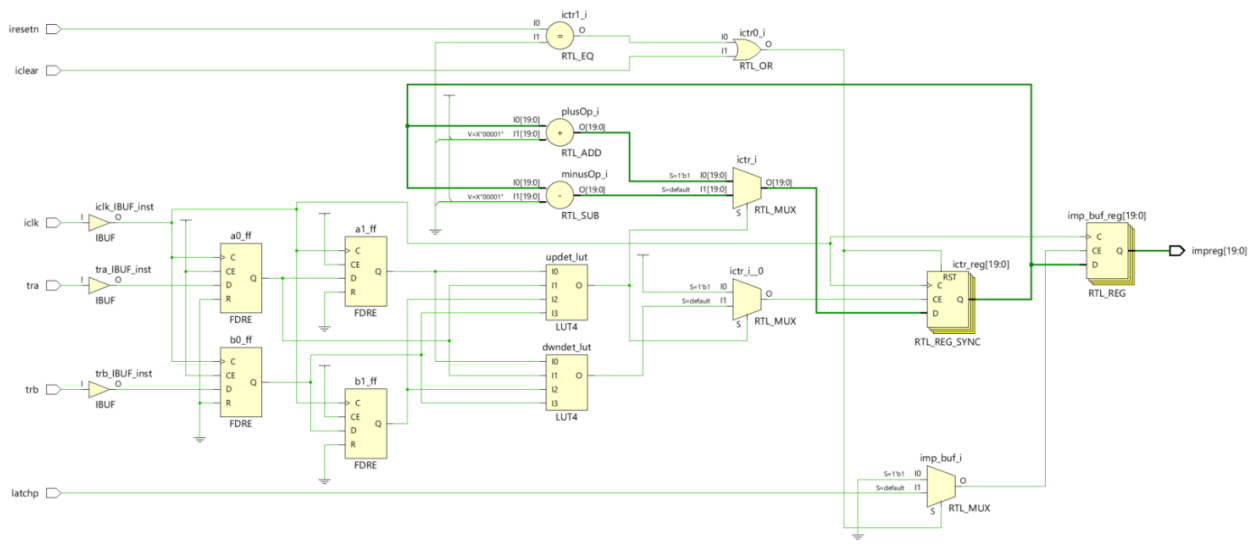


Figure 57: Up & Down Counter

9.4 SPI Protocol

Communication between electronic devices mirrors human communication in the sense that both parties must use the same language. In electronics, these languages are referred to as communication protocols. SPI (Serial Peripheral Interface) operates at a slower speed compared to protocols like USB, Ethernet, Bluetooth, and WiFi, but it is more straightforward, requiring fewer hardware and system resources.

SPI stands out as a widely adopted communication protocol employed across a diverse range of devices. Noteworthy examples include SD card modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers, all relying on SPI for seamless communication with microcontrollers. A key strength of SPI lies in its capacity to transfer data in a continuous stream, accommodating any number of bits sent or received. [9]

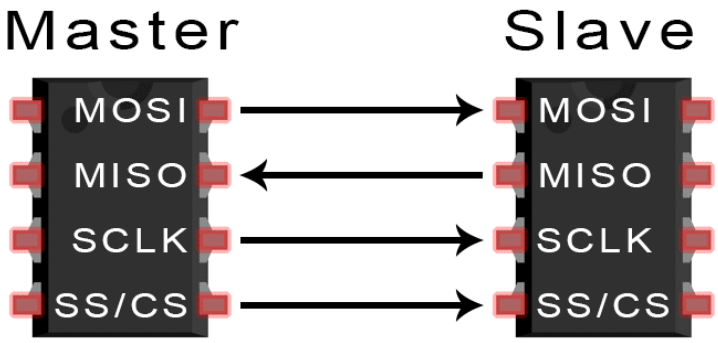


Figure 58: SPI master and slave interface [9]

Typically, devices communicating through SPI adhere to a master-slave relationship. In this dynamic, the master, often embodied by a microcontroller, takes charge of communication,

issuing instructions that the slave, which could be a sensor, display, or memory chip, dutifully follows. While the simplest SPI In the SPI communication protocol, several essential lines facilitate the exchange of data between the master and slave devices:

1. **MOSI (Master Output/Slave Input)**: This line serves as the channel through which the master transmits data to the slave.
2. **MISO (Master Input/Slave Output)**: Conversely, the MISO line enables the slave to send data back to the master.
3. **SCLK (Clock)**: The SCLK line carries the clock signal, synchronizing the timing of data transfer between the master and slave.
4. **SS/CS (Slave Select/Chip Select)**: This line allows the master to designate which specific slave it intends to communicate with, effectively selecting the target device.

In practical implementations, the number of slaves that can be effectively managed is constrained by the system's load capacitance. This limitation impacts the master's ability to accurately switch between voltage levels, influencing the reliability and efficiency of the communication process.

The timing diagram for SPI communication of DAC AD5624R and ADC AD7886 are shown in the figures below.

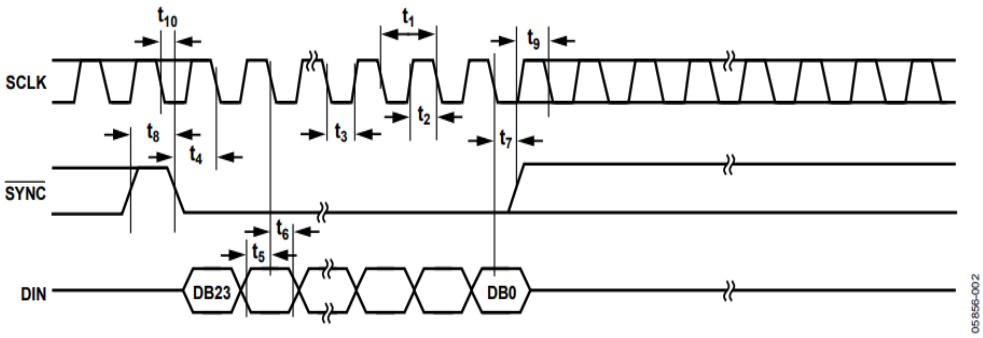


Figure 59: DAC SPI write cycle [7]

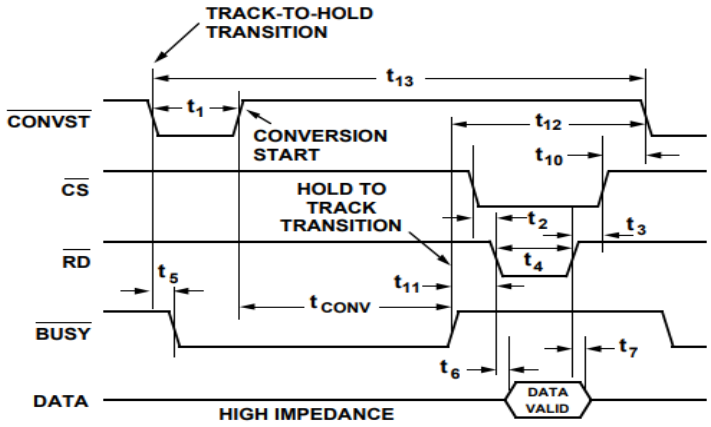


Figure 60: Conversion Start and Data Read Timing Diagram of ADC [8]

10Cart Pendulum SoC design

A processor system, an AXI bus, and a peripheral interface make up the core components of the Cart Pendulum SoC design.

Custom IPs were created in order to implement the peripheral interfaces. The custom IPs include an instance of the universal reconfigurable SPI and additional logic for specific purposes.

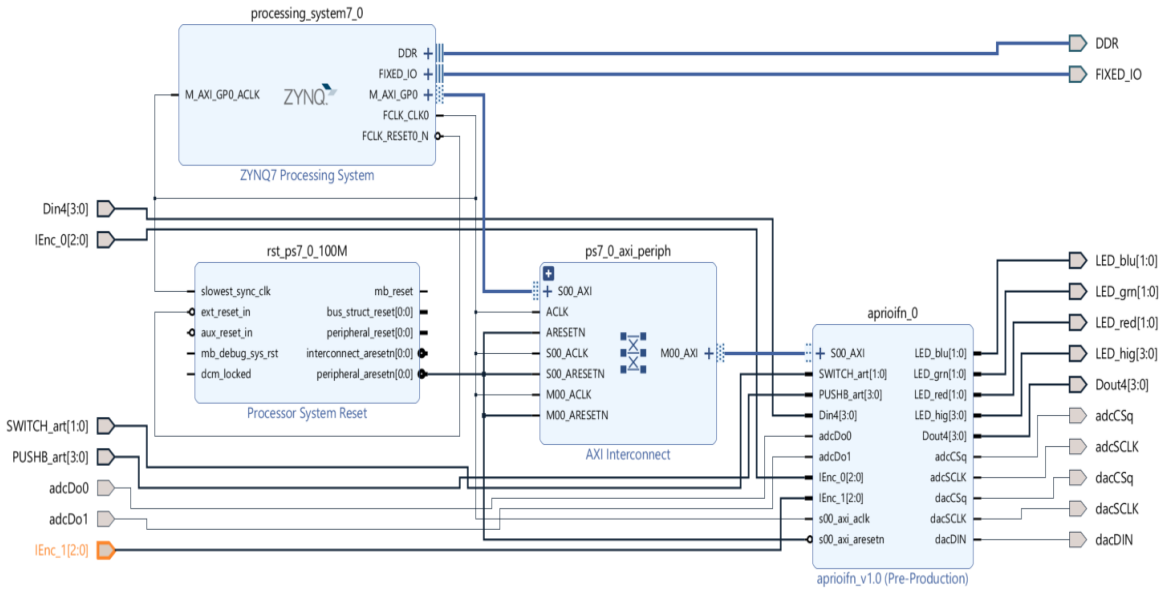


Figure 61: Cart Pendulum SoC Design

10.1 Cart Pendulum Custom IP Component

The main purpose of the aprioifn_0 block, which is shown in the picture below, is to make it easier for the Arty Z7 FPGA and the basic POXI interface board to communicate and operate together. This calls for the use of proprietary VHDL-developed IP, which includes an AXI peripheral with four 32-bit registers. Effective communication with the POXI board is made possible by these registers.

The chip select distribution logic and SPI module are integrated within the aprioifn_0 block. It also contains drivers for Arty Z7 board LEDs, pushbuttons, and switches, as well as drivers for red and infrared LEDs.

The implemented SPI module manages chip select distribution, controls the operation of red and infrared LED drivers, Arty Z7 board LEDs, switches, and pushbuttons, and streamlines communication with the POXI board.

For a detailed overview of the POXI ports and interfaces, see the image below.

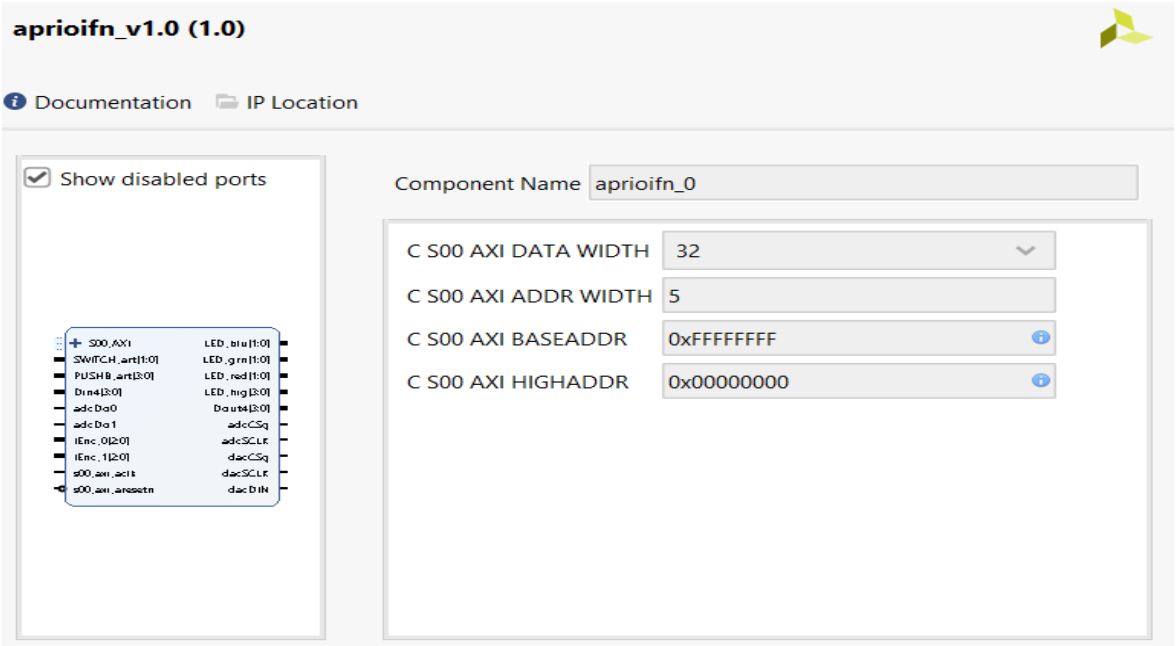


Figure 62: IP Block diagram

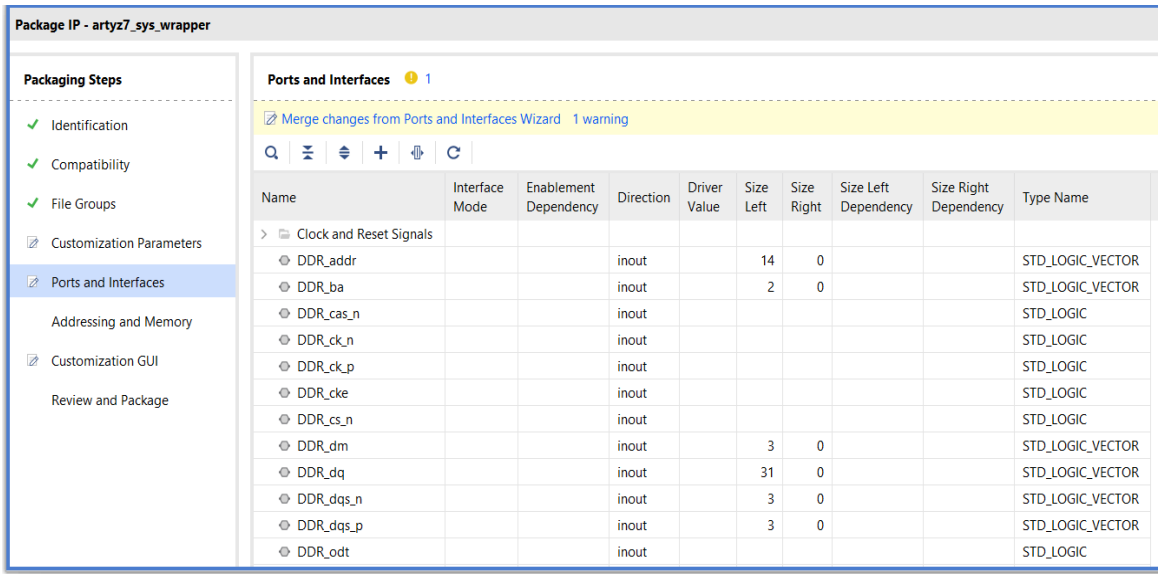


Figure 63: IP port and interface

10.2 SPI VHDL Module

The SPI protocol is used to establish communication between the Arty Z7 FPGA chip and external peripherals. A unique SPI module that has been created within the custom IPs (AXI peripherals) provides the following features:

- 16- and 24-bit configurable data length choices are available.
- Clock phase and polarity adjustments (CPHA and CPOL).
- complete duplex data transfer ability.

Appendix A contains specific information about the VHDL code that implements the SPI Protocol.

10.2.1 Sequential process VHDL code for spi pha='0'

```
    -- spi PHA=0 logic
rcvDataA <= rd_bufA;
rcvDataB <= rd_bufB;

    sseq_proc: PROCESS(resetn, bclk, next_state, sclk_i, scsq_i,
                                rd_bufA, rd_bufB, sdiA, sdiB, count, spi_clkp)
    BEGIN
            IF rising_edge(bclk) THEN
                    IF resetn='0' THEN
                            state <= sidle;
                            count <= USPI_SIZE-1;
                    ELSIF spi_clkp='1' THEN
                            IF next_state=sstartx THEN
            count <= USPI_SIZE - 1;
            ELSIF next_state=sclk_lo THEN
                count <= count - 1;
                rd_bufA <= rd_bufA(USPI_SIZE-2 downto 0) & sdiA;
                rd_bufB <= rd_bufB(USPI_SIZE-2 downto 0) & sdiB;
            ELSIF next_state=sstop_lo THEN
                rd_bufA <= rd_bufA(USPI_SIZE-2 downto 0) & sdiA;
                rd_bufB <= rd_bufB(USPI_SIZE-2 downto 0) & sdiB;
            END IF;
```

Figure 64: VHDL code for spi pha='0'

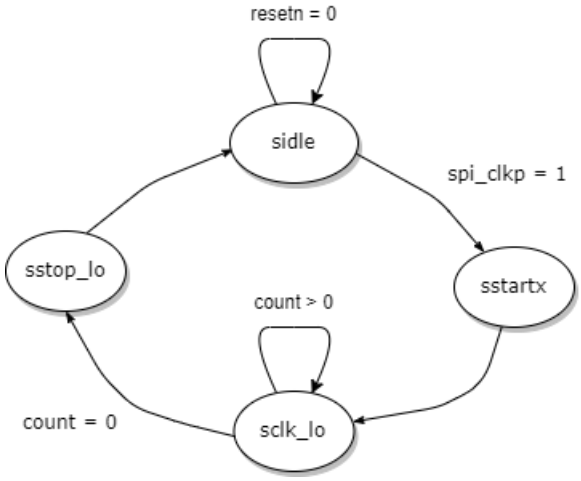


Figure 65:State diagram for spi pha='0'

It manages state transitions, data reception, and clock division. Upon the rising edge of the clock (bclk), it adjusts states based on reset and clock conditions, updates the count and buffers during data reception, and controls output signals (sclk and scsq). The process contributes to the overall SPI receiver functionality.

10.2.2 Start/Stop and Clock Handling

The provided state machine serves the purpose of initiating and concluding SPI communication while ensuring proper handling of clock division.

This state machine encompasses three distinct states: ssl_idle, ssl_start, and ssl_run. In the event that the reset signal is set to '0', the state machine seamlessly transitions to the ssl_idle state. Within this state, the done signal is asserted to '1,' signaling the successful transmission of 16 bits.

```
-- start/stop logic
-- proper handling of clock division
sslseq_proc: PROCESS(resetn, bclk, sslnxt_state)
BEGIN
        IF rising_edge(bclk) THEN
                IF resetn='0' THEN
                        ssl_state <= ssl_idle;
                ELSE
                        ssl_state <= sslnxt_state;
                END IF;
        END IF;
END PROCESS sslseq_proc;

sslcmb_proc: PROCESS(ssl_state, state, start)
BEGIN
        sslnxt_state <= ssl_state;
        start_i <= '0';
        done <= '0';
        CASE ssl_state IS
                WHEN ssl_idle =>
                        done <= '1';
                        IF start='1' AND state=sidle THEN
                                sslnxt_state <= ssl_start;
                        END IF;
                WHEN ssl_start =>
                        start_i <= '1';
                        IF state=sstartx THEN
                                sslnxt_state <= ssl_run;
                        END IF;
                WHEN ssl_run =>
                        IF state=sidle THEN
                                sslnxt_state <= ssl_idle;
                        END IF;
        END CASE;
```

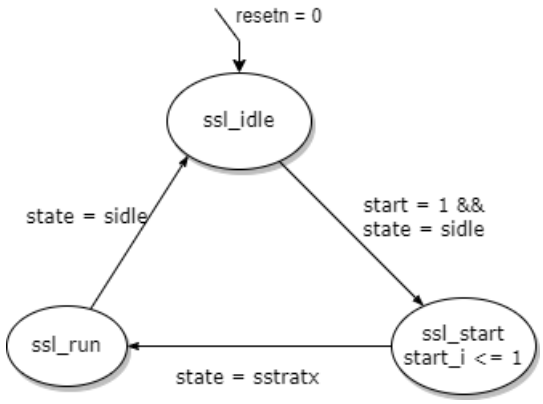


Figure 66:SPI start and stop logic state machine

Upon detecting a '1' in the start bit and the SPI master logic being in the sidle state, the state machine transitions to the ssl_start state.

Within the ssl_start state, the start_i signal is activated, and progression to the ssl_run state occurs when the SPI master logic state is sstartx.

The ssl_run state in the start/stop state machine facilitates the transmission of 16 bits of data. Upon completion of data transmission, denoted by the SPI master logic transitioning to the idle state, the state machine cyclically reverts to the ssl_idle state, perpetuating this operational cycle.

11Scaling for the Position and Speed Controller

For the position and speed controller, they must be physically scaled as it cannot interact directly with the Cart Pendulum system. It is the sensors and the actuators which help the controllers to interact with the system. Here we have the incremental encoders as sensors and linear actuator LEZ3 as actuator. Therefore, by physical scaling it is possible to adjust controllers to the hardware components and use them as required.

11.1 Incremental Encoder

The motor encoder has a resolution of 4000 incr. /rev., which means it sends 4000 pulses in one revolution. That is the way the encoder and controller interact.
The encoder reading for given cart position is derived by:

$$\frac{P_{0k}}{P_0} = \frac{R_E}{T_d}$$

Where,
$\mathrm{P_{0k}}$ = Encoder reading for cart position (increments)
$\mathrm{P_0}$ =Rated cart position (m)
$\mathrm{R_E}$ =Motor Encoder resolution (increments/revolution)
$\mathrm{T_d}$ = Traveled distance by cart per one revolution (m/rev.)
Now,

$$P_{0k} = \frac{R_E}{T_d} * P_0$$

$$\therefore \quad P_{0k} \approx 57147 * P_0$$

Therefore, for 1 meter cart position, encoder will give 57147 increments.

For pendulum angle there is one more incremental encoder. The pendulum angle can be calculated using the same techniques as for the cart position. The difference is that pendulum encoder has 4096 incr. /rev.
Hence, the encoder reading for given cart position is derived by:

$$A_{0k} = \frac{R_E}{2\pi} * A_0$$

Where,
$\mathrm{A_{0k}}$ = Encoder reading for Pendulum angle (increments)
$\mathrm{A_0}$ =Rated Pendulum angle (rad)
$\mathrm{R_E}$ = Pendulum Encoder resolution (increments/revolution)

A state feedback PI controller is designed for the Cart Pendulum system, as discussed in Chapter 7. The controller requires full state information, including cart position, cart velocity, pendulum angle, and pendulum angular velocity. Four sensors are required to measure four states, but the incremental encoder's high resolution allows for the calculation of cart velocity and pendulum angular velocity by differentiating cart position and pendulum angle. Therefore, only two sensors are required.

Cart velocity in sensor units can be calculated for rated cart velocity $V_0 = P_0$m/s as:

$$V_{0k} = P_{0k} * T_s$$

Where,
$V_{0k} =$ Encoder reading for rated cart velocity (incr./s)
$T_s =$ Sample time (s)

Similarly, pendulum angular velocity in sensor units can be calculated for rated pendulum angular velocity $Av_0 = A_0$ rad/s as:

$$Av_{0k} = A_{0k} * T_s$$

Where,
$V_{0k} =$ Encoder reading for rated pendulum angular velocity(incr./s)
$T_s =$ Sample time (s)

11.2 LEZ3 Actuator

We have one actuator in our system, which moves in both right and left directions. Therefore, the driving motor controls torque in both positive and negative directions. The DAC has a 12-bit resolution and can handle values ranging from 0 to 4095. The negative force ranges from 0 to 2047, while the positive force ranges from 2048 to 4095.

For physical scaling for the force applied to the cart, the relationship between DAC values and Force (N) must be derived. The experiment is performed on test bench to derive relation between DAC values and force (N):

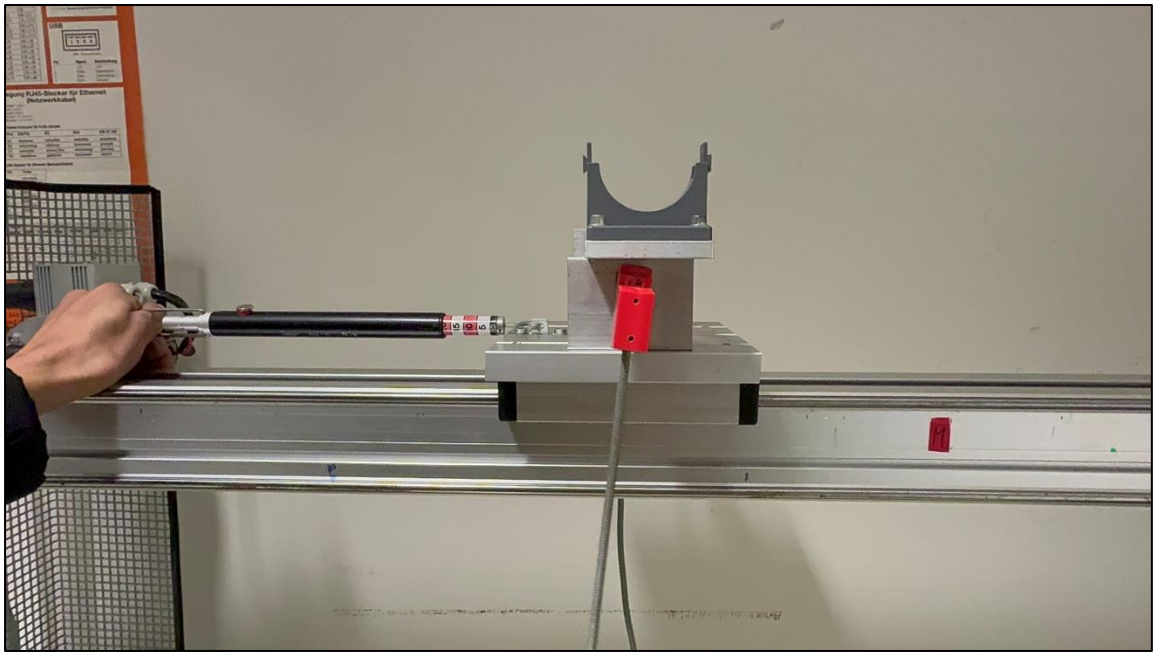


Figure 67: Experiment to derive relationship between DAC values and force(N)

In this experiment, force applied to the cart is derived using a provided force scale. The experiment is performed by holding a force scale at one end and cart is attached with force scale to another end. The experiment is done in the right (Positive) direction. The DAC values transmitted by Zynq is recorded, which shows the relation as below graph,

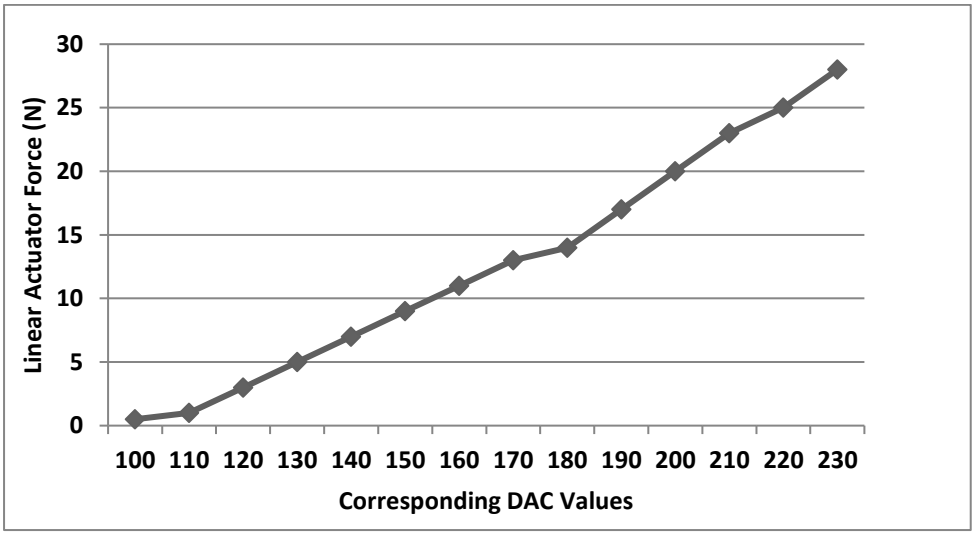


Figure 68: Relationship between DAC values and force(N)

From the above graph, it can be observed that the force scale gives almost zero reading for the DAC values below 100. This is because of the effect of static friction. It can also be noted that the graph shows the linear relationship betweenDAC values and force(N). Now we have to

select one relation for our scaling calculation, so the point (200, 20) is selected for simplicity. Hence, 20 N is our rated force (F_0) and 200 is corresponding DAC value (F_{0k}).

11.3 Physical Scaling

The physical scaling is done to adjust the controllers to interact with sensor's reading to control the actuator. The below table indicates all the physical scaling parameters required in cart pendulum system.

Scaling Variables	Values
Ts (Sample time(ms))	10
P_0 (Rated cart position(m))	1
P_{0k} (Rated position in sensor units)	57147
V_0 (Rated cart speed (m/s))	1
V_{0k} (Rated cart speed in sensor units)	571.47
A_0 (Rated pendulum angle (rad))	2*pi
A_{0k} (Rated pendulum angle in sensor units)	4095
Av_0 (Rated pendulum angular velocity (rad/s))	2*pi
Av_{0k} (Rated pendulum angular velocity in sensor units)	40.95
F_0 (Rated force(N))	20
F_{0k} (Rated force in actuator units)	200

Table 4: Scaling variables for control of cart pendulum system

11.3.1Physical Scaling for Cart Position Controller:

The position controller is now scaled physically shown in figure below:

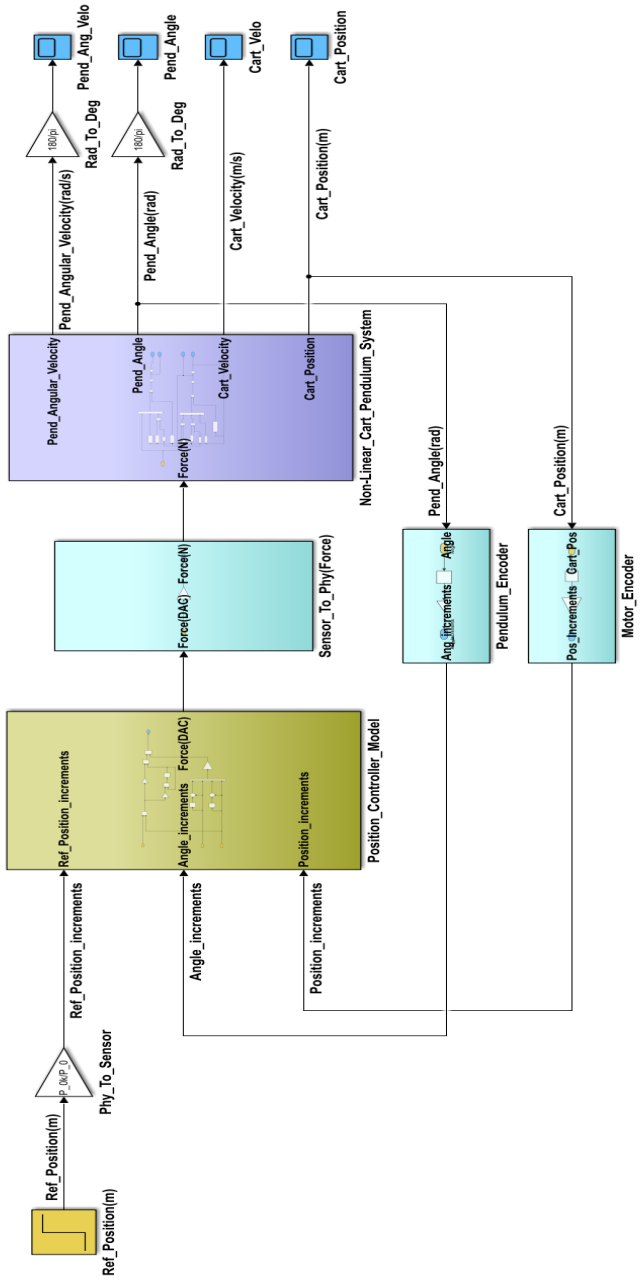


Figure 69: Physically scaled cart position controller

In above model, The Pendulum Encoder indicates the conversion of pendulum angle in radian to pendulum angle in increments. The Motor Encoder indicates the conversion of cart position in meter to cart position in increments. The force conversion block convert force from the DAC values to newton unit, which further applied to the non-linear system. The Position Cotroller

Model indicates the phisycally scaled PI controller. We have already discussed about all the scaling parameters used in above model except for proportional gain, integral gain, and state beedback gain vector.

Hence, the scaled proportional gain and integral gain iscalculated by the equations below.

$$G_p = \frac{P_0}{P_{0k}} * P_D * \frac{F_{0k}}{F_0}$$

$$G_i = \frac{P_0}{P_{0k}} * pi_d * T_s * \frac{F_{0k}}{F_0}$$

The state feedback gain vector has to be scaled for each gain as it has four states. The scaled state feedback gain vector is calculated by the equations below.

$$ktd_s(1) = \frac{Av_0}{Av_{0k}} * kt_d(1) * \frac{F_{0k}}{F_0}$$

$$ktd_s(2) = \frac{A_0}{A_{0k}} * kt_d(2) * \frac{F_{0k}}{F_0}$$

$$ktd_s(3) = \frac{V_0}{V_{0k}} * kt_d(3) * \frac{F_{0k}}{F_0}$$

$$ktd_s(4) = \frac{P_0}{P_{0k}} * kt_d(4) * \frac{F_{0k}}{F_0}$$

Thus, the scaled proportional gain, integral gain, and state Feedback gain vector can be now calculated.

Gains	Description	Values
P_D	Unscaled proportional gain	32.32
pi_d	Unscaled integral gain	126.75
kt_d	Unscaled state feedback vector gain	[-6.1571 -99.6894 61.2623 126.7680]
G_p	Scaled proportional gain	0.0057
G_i	Scaled integral gain	2.2180e-04
ktd_s	Scaled state feedback vector gain	[-9.4472 -1.5296 1.0720 0.0222]

Table 5: All Model gains and their calculated scaled gain

All the required variables are now scaled; hence these variables can be used to design pi controller for position control. The PI position controller can now be revealed from figure 70.

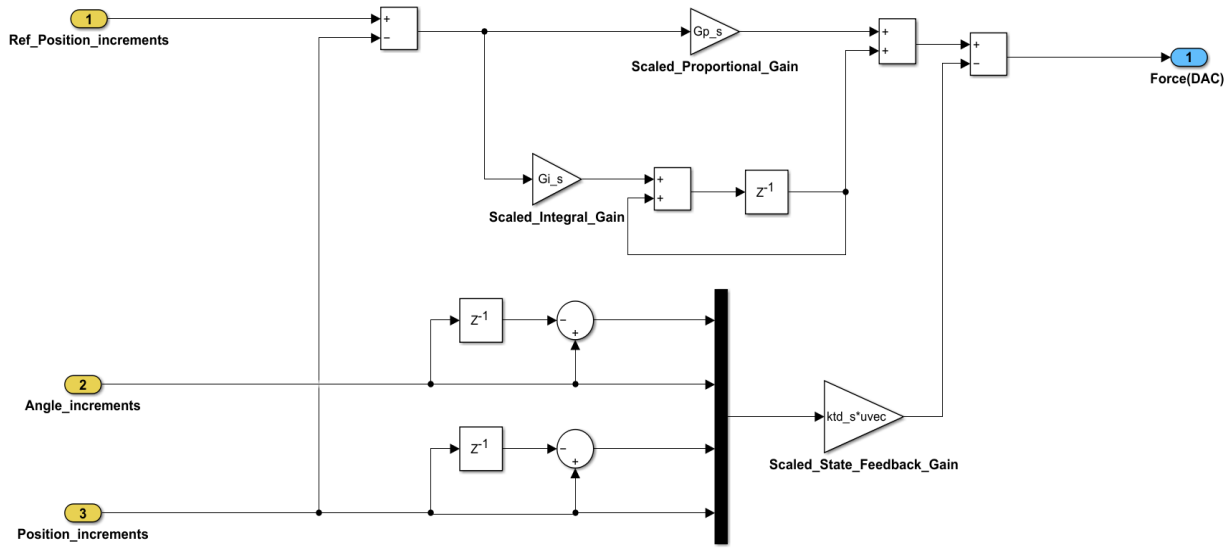


Figure 70: Physically scaled position controller

The below figures show the simulation's result of the cart position and cart velocity with a step time of 1 second and reference position of 1 m.

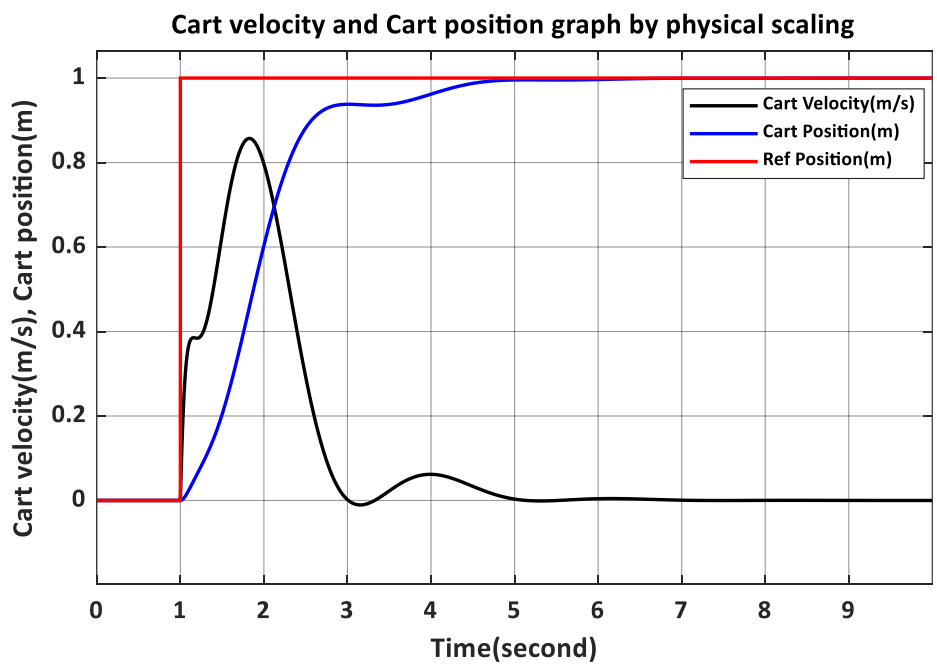


Figure 71: Cart velocity and Cart position graph after physically scaled

From the above graph it is clear that the system becomes stable after approximately 5 seconds. Both cart velocity and cart position graphs become stable after approximately 5 seconds. The

below figure shows the graphs of pendulum angle and pendulum angular velocity. The graph illustrates the degree of control achieved over a pendulum within a five-second timeframe.

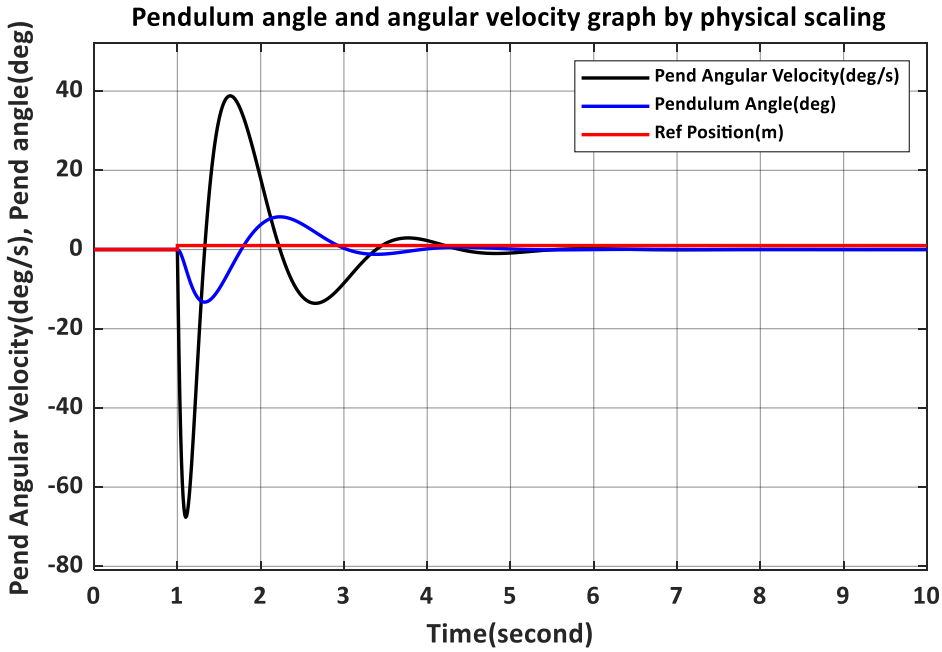


Figure 72: Pendulum angle and angular velocity after physically scaled

11.3.2Fixed Point Position Controller:

Fixed-point arithmetic provides more precision than floating-point arithmetic. For implementation of fixed point, the range of the numbers must be known.The controllers' signals are examined, and the maximum values are established.The Zynq device, equipped with a 32-bit microprocessor, necessitates integer sizes of 16 and 32 bits due to its architecture. Additionally, we also have to consider signed integer as signals could be positive or negative.

All the gains can be scaled as below table.

Gains	Values	Selected Fixed Point Format	Integer Value
G_p	0.0057	Int_16.15	185
G_i	2.2180e-04	Int_16.22	930
ktd_s(1)	-9.4472	Int_16.11	-19348
ktd_s(2)	-1.5296	Int_16.14	-25061
ktd_s(3)	1.0720	Int_16.14	17564
ktd_s(4)	0.0222	Int_16.14	363

Table 6: All scaled gains and their selected fixed point formats

In the above table, in case of integral gain 7-bits are added to increase its precision and for better result. Hence, format Int_16.22 is chosen for integral gain. All the other formats are selected by optimal fixed point format method.

The fixed-point PI controller model is now implemented in position controller as below,

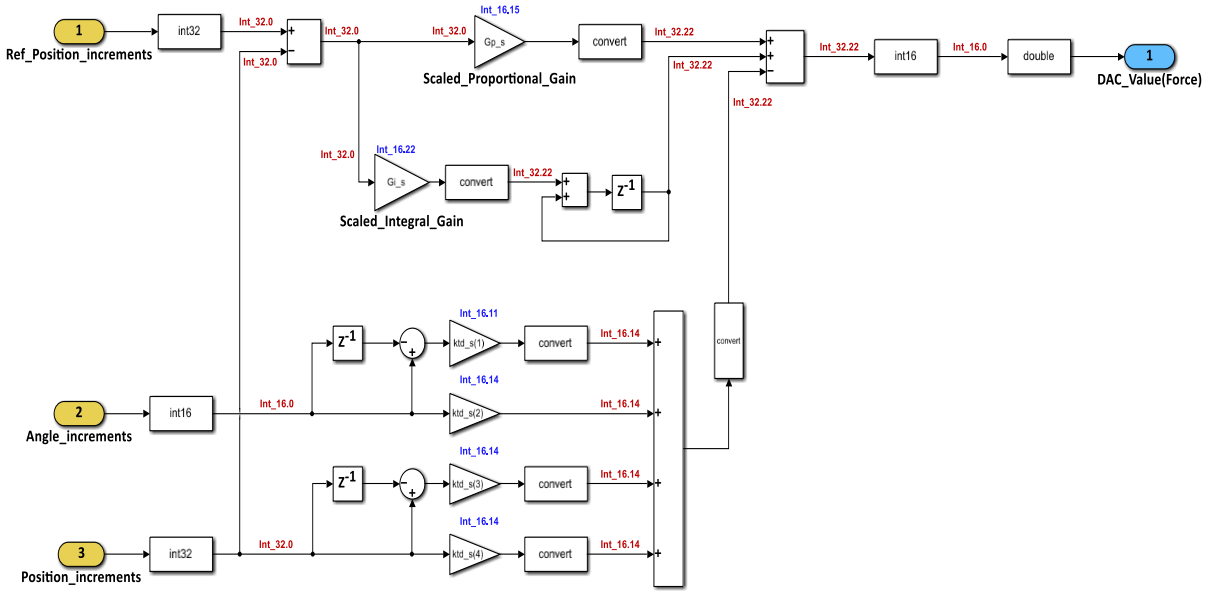


Figure 73: Fixed point position controller

The below figures indicates the simulation's result of the required force applied to the cart with a step time of 1 second and reference position of 1 m. from the graph it is clear that the force applies to the cart continuously for 5 seconds, therefore system becomes stable within 5 seconds.

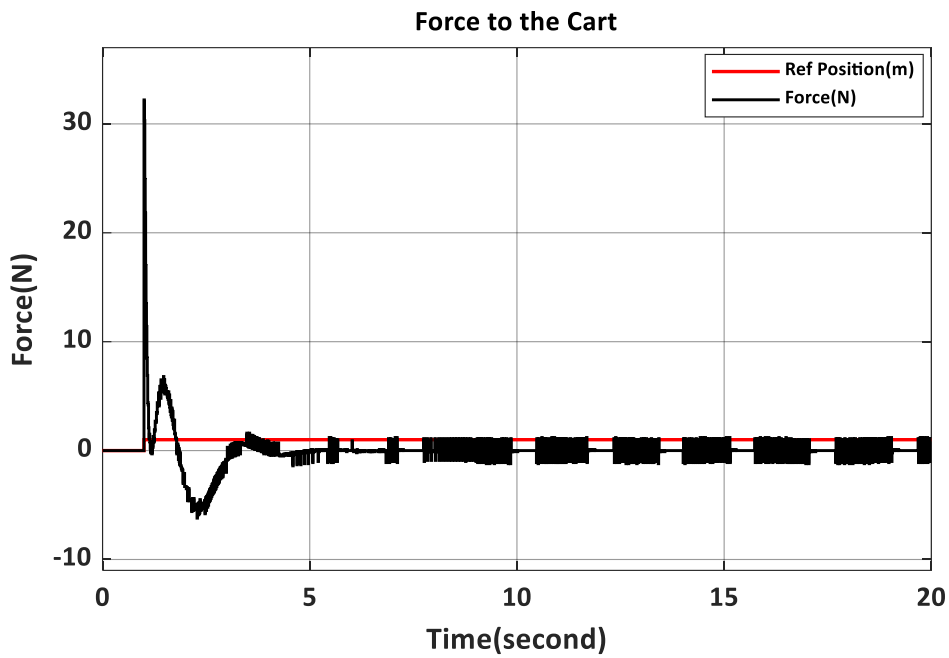


Figure 74: Required force to the cart for fixed point model

The below figures shows the simulation's result of the cart position with a step time of 1 second and reference position of 1 m.

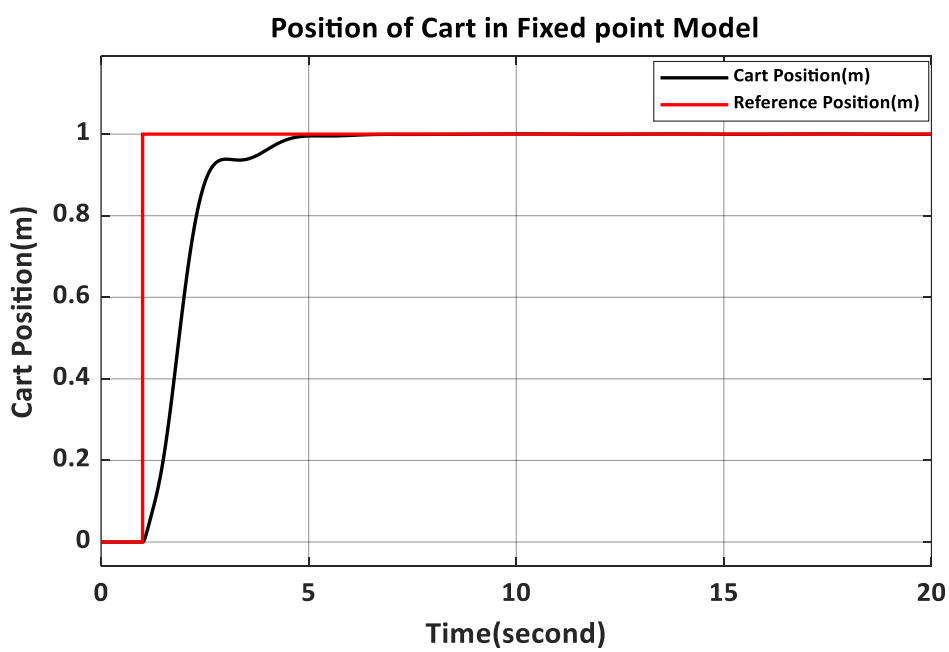


Figure 75: Cart position for fixed point model

From the above graph, it is clear that the system becomes stable after approximately 5 seconds, that means cart reaches to the desired position in approximately 5 seconds. In case of pendulum angle, it oscillates between -13.25 to 8.23 degrees (According to simulation output) and it further oscillates between -0.018 to +0.018 degrees as can be shown in below figure.

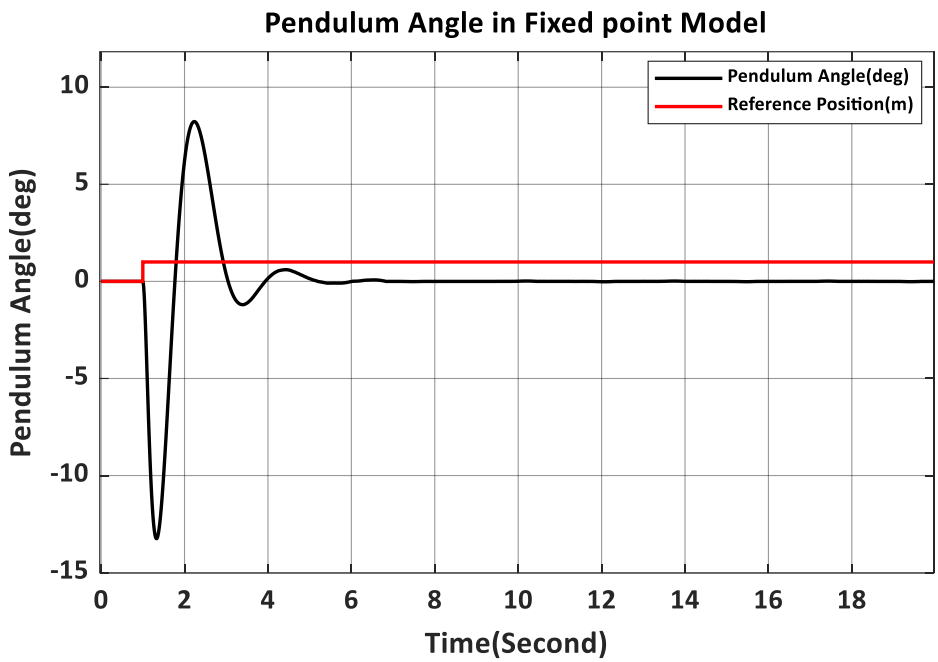


Figure 76: Pendulum angle for fixed point model

11.3.3Physical Scaling for Cart Speed Controller:

The speed controller is designed by following the same approach as in cart position controller. The position controller is scaled physically shown in figure below.

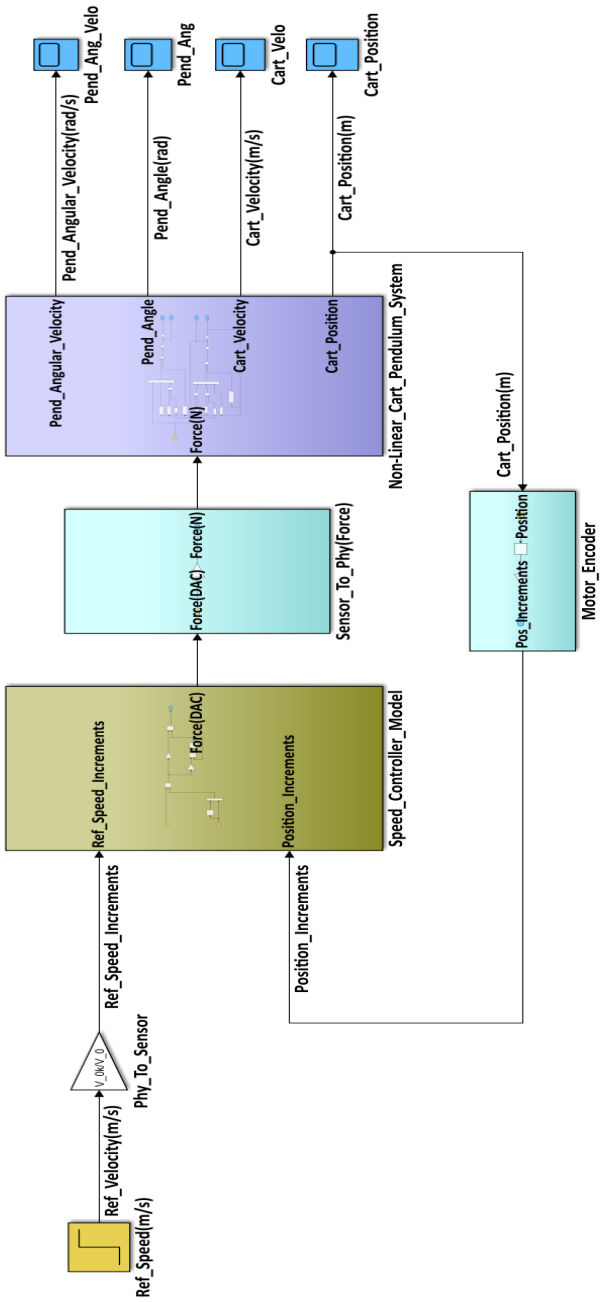


Figure 77: Physically scaled cart speed controller

The scaled proportional gain and integral gain is calculated by the equations below.

$$G_p_s = \frac{V_0}{V_{0k}} * P_D * \frac{F_{0k}}{F_0}$$

$$G_i_s = \frac{V_0}{V_{0k}} * pi_d * T_s * \frac{F_{0k}}{F_0}$$

The scaled proportional gain and integral gain can be now calculated by above equation.

Gains	Description	Values
P_D	Unscaled proportional gain	32.32
pi_d	Unscaled integral gain	34.56
G_p_s	Scaled proportional gain	0.5657
G_i_s	Scaled integral gain	0.0060

Table 7: Proportional and Integral gains for model and their calculated scaled gain

All the required variables is now scaled, hence these variables can be used to design pi controller for speed control. The speed PI controller can be now revealed from the figure 78.

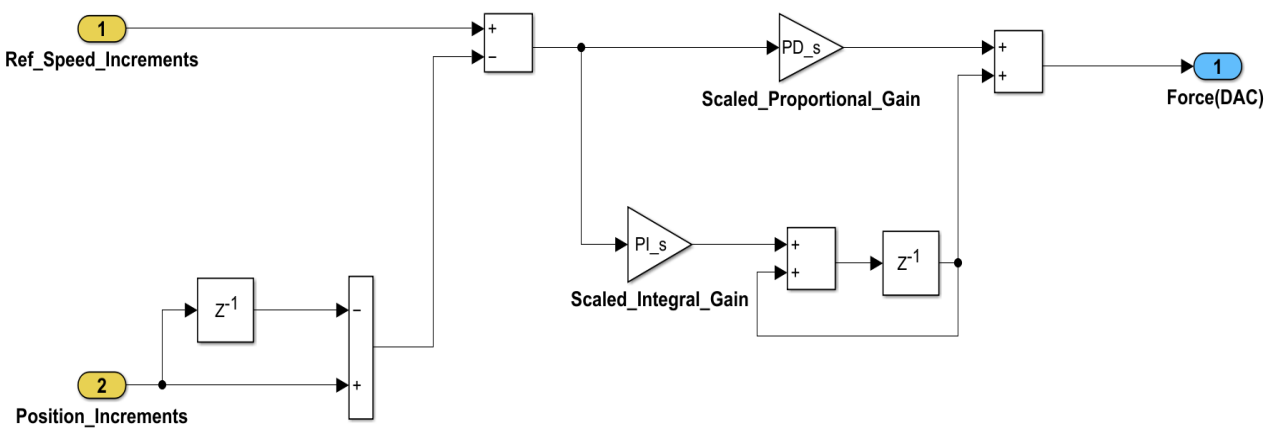


Figure 78: Physically scaled speed controller

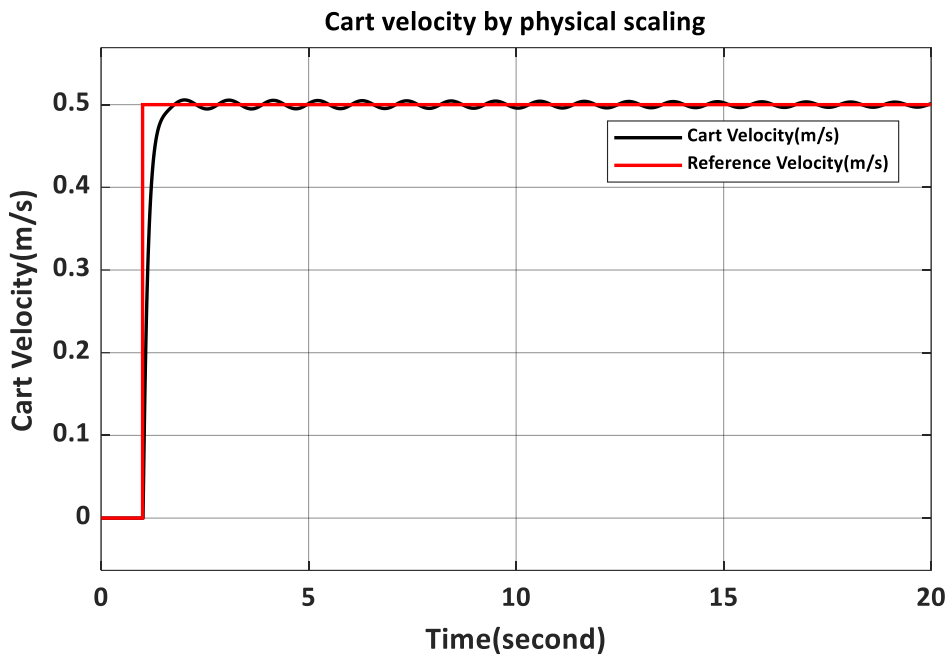


Figure 79: Cart velocity graph after physically scaled

The below figures shows the simulation's result of the cart velocity with a step time of 1 second and reference velocity of 0.5 m/s. It has been consistently noted that the velocity of the cart exhibits minimal deviation from a stable value when compared to reference velocity.

11.3.4Fixed Point Speed Controller:

The cart speed controller has to also be scaled to a fixed point. Hence, the gains for speed controller are converted in fixed point format in the table below.

Gains	Values	Selected Fixed Point Format	Integer Value
G_p_s	0.5657	Int_16.15	18536
G_i_s	0.0060	Int_16.15	198

Table 8: Proportional and Integral gains and their selected fixed-point formats

The fixed-point PI controller model is now implemented in speed controller as below,

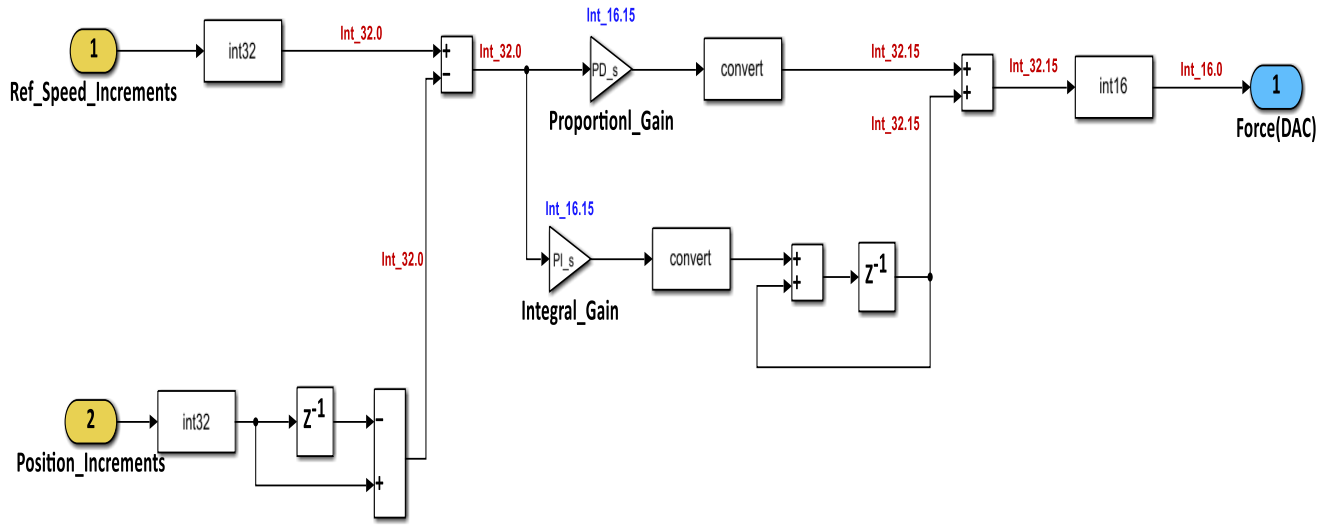


Figure 80: Fixed point speed controller

The below figures show the simulation's result of the cart velocity with a step time of 1 second and reference velocity of 0.5m/s. Upon analysis of the graph depicted below, it is evident that the velocityof the cart, as determined through physical scaling, closely aligns with that obtained through the utilization of a fixed-point controller.

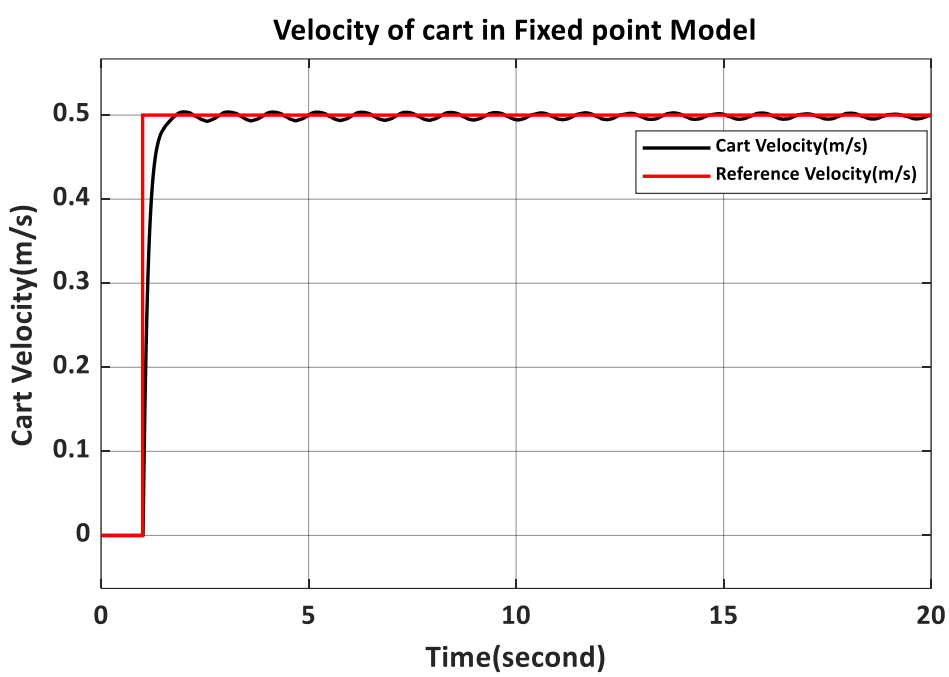


Figure 81: Cart velocity for fixed point model

12S-Function Development

The primary objective of this project is to implement both position and speed controllers within a hardware testbench environment. To achieve this goal, it is imperative to design C-code implementations for both the speed and position controllers. Subsequently, it is essential to verify the functionality and correctness of these C-code implementations to ensure they operate as intended. To facilitate this verification process, S-functions are generated.

The S-function serves as intermediary components that enable the simulation and testing of the C-code implementations within the MATLAB Simulink environment. By leveraging S-functions, we can validate the behavior and performance of the speed and position controllers prior to their deployment on the hardware testbench. This approach ensures that any potential issues or discrepancies in the C-code implementations are identified and addressed early in the development cycle.This methodology of simulating the C code alongside the plant model is called Software-in-the-Loop (SIL) testing.

12.1 C code test for Cart Position Controller

The c code for cart position controller is implemented using legacy code block with MATLAB and Simulink model. The legacy code description is given below,

```
% create legacy mex function for cart position control
def = legacy_code('initialize');
def.SourceFiles = {'spctrl_Pos.c','Fixedpt_Position_Control.c'};
def.HeaderFiles = {'Fixedpt_Position_Control.h'};
def.SFunctionName = 'ex_sfun_spctrl';
def.OutputFcnSpec = 'int16 y1 = spctrl_Pos(int16 u1, int32 u2, int16 u3, int32 u4)';
legacy_code('sfcn_cmex_generate', def)
legacy_code('compile', def)
legacy_code('slblock_generate', def)
disp('That`s all folks.')
```

To generate Simulinkblock using legacy code, two source files and one header file are required as can be seen in the legacy code description.After successfully obtaining satisfactory results from the cart position controller developed in a Simulink model, we proceeded to implement the same mathematical logic in C code. An S-function was then created to facilitate the simulation of this C code.

The C code received inputs consisting of the reference position, cart position, and pendulum angle. Within the code, we implemented Proportional-Integral (PI) logic using fixed-point arithmetic, which calculated the force required. Detailed explanations for each statement can be located within the code comments.

```
#include "xil_types.h"
#include "Fixedpt_Position_Control.h"
static PiObj_struct PiObj;

// Reset the C code
void UPiCtrlInit(s32 pgain, s32 igain) {
        PiObj.kp = pgain;
        PiObj.ki = igain;
        PiObj.xi = 0;
PiObj.Pend_Ang_Prev = 0;
PiObj.Cart_Position_Prev = 0;
}

s16 Fixedpt_Position_Control(s32 Ref_Pos,s16 Pend_Ang,s32 Cart_Pos)
{
   s32 Feedback_Sum=0;
   s16 Force;
   s32 Pend_Ang_Velo,Cart_Velo,PI_controller,Cntrl_Err, KI_mul, KP_mul;
//State beedbak gain
   s16 KTD_f[] = {-19348,-25061,17564,363};

//Calculated control error
   Cntrl_Err = Ref_Pos - Cart_Pos;
//Multiplication with integral gain
   KI_mul = Cntrl_Err * IG_f;
//Integral gain addition
   PiObj.xi = PiObj.xi + KI_mul;
//Multiplication with proportional gain and
//left shifted by 7-bits to match integral gain fixed point format
KP_mul = (Cntrl_Err * PG_f)<<7;
//Getting pendulum angular velocity by derivating pendulum angle
   Pend_Ang_Velo = ((Pend_Ang - PiObj.Pend_Ang_Prev)<<3) * KTD_f[0];
//Getting cart velocity by derivating cart position
   Cart_Velo = (Cart_Pos -  PiObj.Cart_Position_Prev) * KTD_f[2];
   PiObj.Pend_Ang_Prev = Pend_Ang;
   PiObj.Cart_Position_Prev = Cart_Pos;
//Addition of state feedback gain
   Feedback_Sum = Pend_Ang_Velo+Cart_Velo+Pend_Ang*KTD_f[1]+
PiObj.Cart_Position_Prev*KTD_f[3];
//left shifted by 8-bits to match integral gain fixed point format
   Feedback_Sum = Feedback_Sum<<8;
//Main PI controller logic
   PI_controller = KP_mul + PiObj.xi - Feedback_Sum;
//Right shift by 22 to convert back to int_16 format
   PI_controller=PI_controller>>22;
   Force = (PI_controller);
return (Force);
}
```

In cases where the reset function is absent within a C code implemented in a legacy block, MATLAB may produce varying outputs with each execution. To address this issue and ensure consistent behavior, the function "void UPiCtrlInit(s32 pgain, s32 igain)" is invoked to reset the C code. This initialization process helps stabilize the behavior of the C code and maintain repeatability in the output results across multiple runs.

The S-function block was connected in parallel with the Simulink model to verify the correspondence of the desired output graph between the model and the S-function simulation by manual switch. This configuration allowed for a comparative analysis to ensure consistency and alignment of the output graphs generated by both the Simulink model and the S-function simulation.

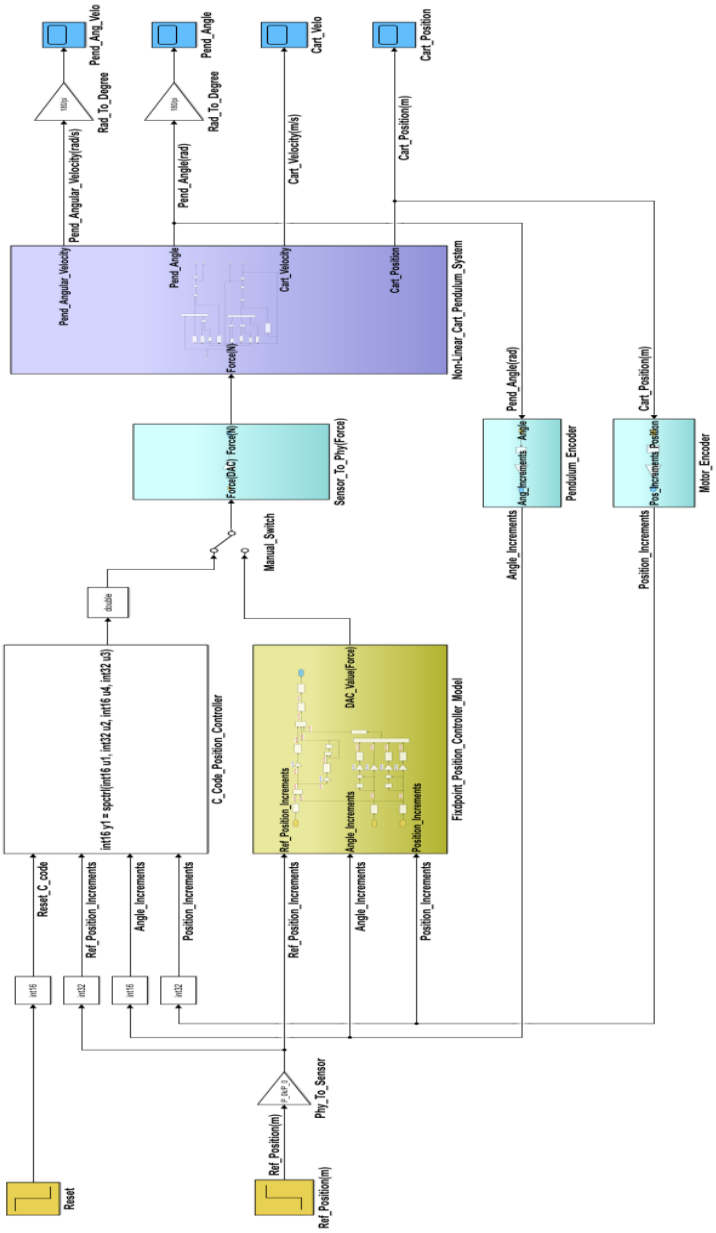


Figure 82: Simulink model and C code legacy block comparison for position controller

Output Graphs

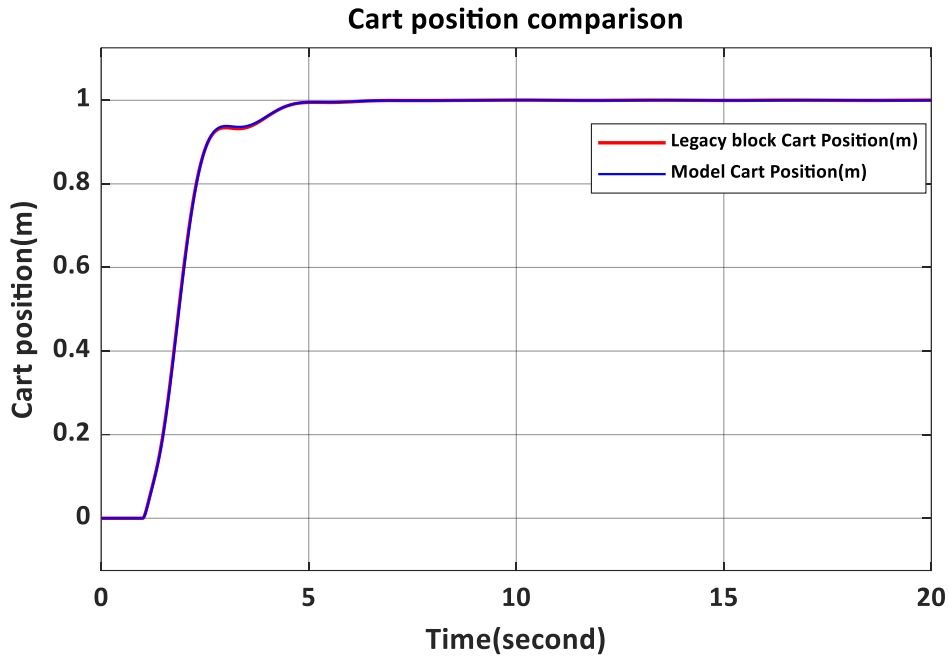


Figure 83: Cart position comparison between legacy block and simulink model

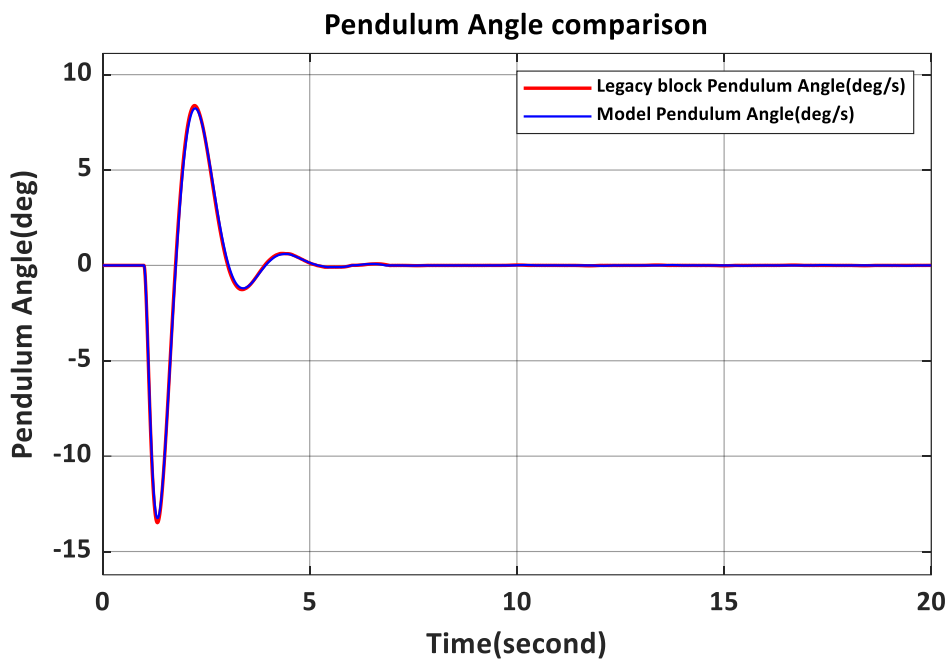


Figure 84: Pendulum angle comparison between legacy block and Simulink model

Based on the analysis of the graphs presented above, it is evident that the output graphs obtained from both the S-function and the Simulink model exhibit identical behavior. Consequently, this observation confirms the successful verification of the S-function through Software-in-the-Loop (SIL) testing. As a result, it can be concluded that the corresponding C-

code implementation is prepared for deployment on the Zynq board for system testing purposes.

12.2 C code test for Cart Speed Controller

The c code for cart speed controller is implemented using legacy code block with MATLAB and Simulink model. The legacy code description is given below,

```
% create legacy mex function for cart control
def = legacy_code('initialize');
def.SourceFiles = {'spctrl_Speed.c','Fixedpt_Speed_Control.c'};
def.HeaderFiles = {'Fixedpt_Speed_Control.h'};
def.SFunctionName = 'ex_sfun_spsctrl';
def.OutputFcnSpec = 'int16 y1 = spctrl_Speed(int16 u1, int32 u2, int32 u3)';
legacy_code('sfcn_cmex_generate', def)
legacy_code('compile', def)
legacy_code('slblock_generate', def)
disp('That`s all folks.')
```

To generate simulink block using legacy code, two source files and one header file are required as can be seen in the legacy code description.After successfully obtaining satisfactory results from the cart speed controller developed in a Simulink model, we proceeded to implement the same mathematical logic in C code. An S-function was then created to facilitate the simulation of this C code.

The C code received inputs consisting of the reference position, cart position, and pendulum angle. Within the code, we implemented PI logic using fixed-point arithmetic, which calculated the force required. Detailed explanations for each statement can be located within the code comments.

```
#include "xil_types.h"
#include "Fixedpt_Speed_Control.h"
static PiObj_struct PiObj;

// Reset the C code
void UPiCtrlInit(s32 pgain, s32 igain) {
PiObj.kp = pgain;
PiObj.ki = igain;
PiObj.xi = 0;
PiObj.Cart_Position_Prev=0;
}
s16 Fixedpt_Speed_Control(s32 Ref_Velo, s32 Cart_Pos)
{
s16 Force=0;
```

```
s32 KP_mul,KI_mul,Cntrl_Err,PI_controller,Cart_Velocity;

//Getting cart velocity by derivating cart position
Cart_Velocity = (Cart_Pos-PiObj.Cart_Position_Prev);
PiObj.Cart_Position_Prev = Cart_Pos;
//Calculated control error
Cntrl_Err = Ref_Velo-Cart_Velocity;
//Multiplication with proportional gain
KP_mul = (Cntrl_Err*PG_f_Speed);
//Multiplication with integral gain
KI_mul = Cntrl_Err*IG_f_Speed;
//Integral gain addition
PiObj.xi = (PiObj.xi) + KI_mul;
//Applying upper and lower limits
if (PiObj.xi>IG_Limit){
PiObj.xi=IG_Limit;
}
elseif(PiObj.xi < -IG_Limit){
PiObj.xi = -IG_Limit;
}
//Main PI controller logic
PI_controller = KP_mul + PiObj.xi;
//Applying upper and lower limits
if (PI_controller>Out_Limit){
PI_controller=Out_Limit;
}
elseif(PI_controller < -Out_Limit){
PI_controller = -Out_Limit;
}
Force = PI_controller>>15;
return (Force);
}
```

The reset function has been similarly employed in the speed controller as it was in the position controller, adhering to the same procedural approach.

The S-function block was connected in parallel with the Simulink model to verify the correspondence of the desired output graph between the model and the S-function simulation by manual switch. This configuration allowed for a comparative analysis to ensure consistency and alignment of the output graphs generated by both the Simulink model and the S-function simulation.

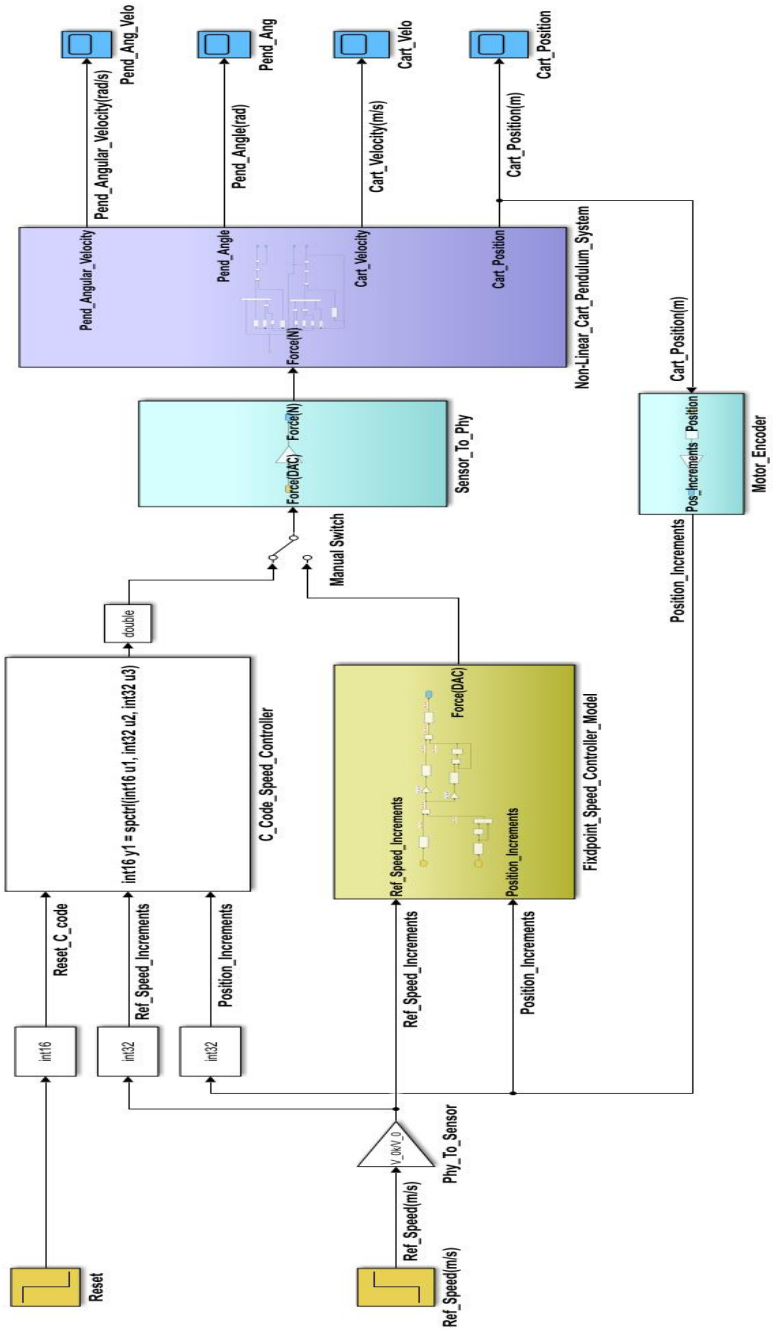


Figure 85: Simulink model and C code legacy block comparison for speed controller

Output Graphs

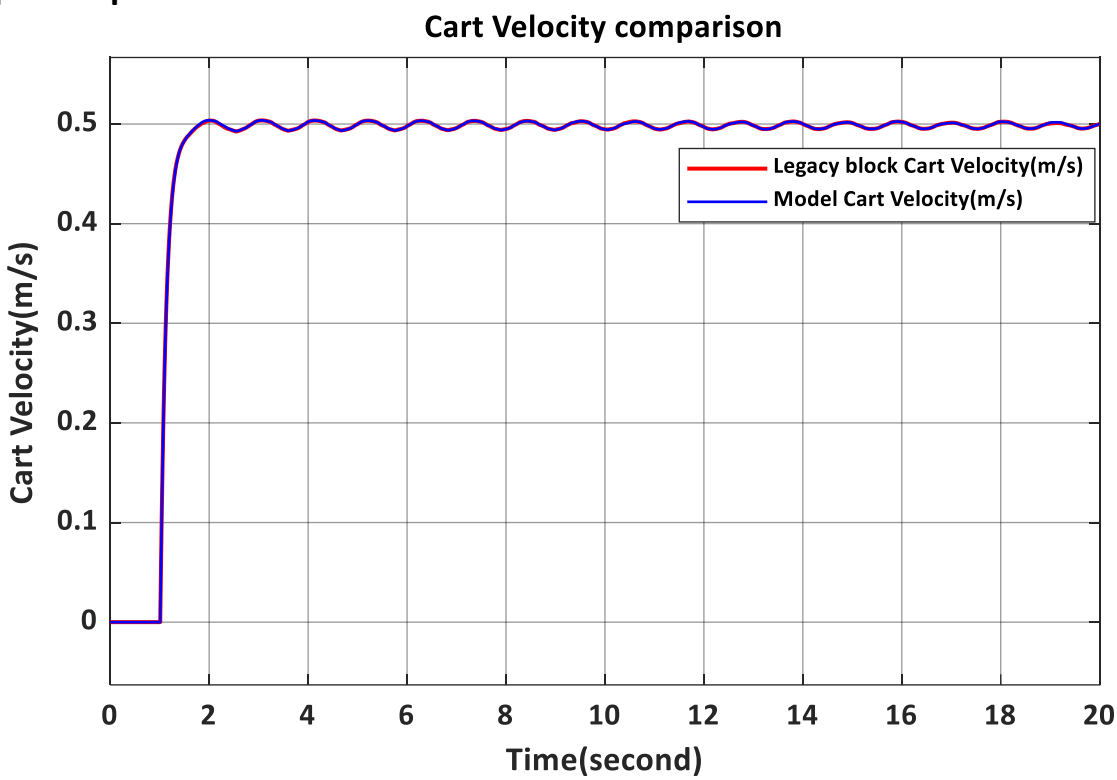


Figure 86: Cart velocity comparison between legacy block and simulink model

Based on the analysis of the graphs presented above, it is evident that the output graphs obtained from both the S-function and the Simulink model show almost identical behavior. Consequently, this observation confirms the successful verification of the S-function through Software-in-the-Loop (SIL) testing. As a result, it can be concluded that the corresponding C-code implementation is prepared for deployment on the Zynq board for system testing purposes.

13Implementation of C code

The software development for the cart pendulum controller primarily utilizes the Xilinx Software Development Kit (SDK), an Integrated Development Environment (IDE) based on Eclipse. Initially designed and validated in MATLAB Simulink using the Legacy Code Tool, the controller logic was later transferred to the SDK. The code is organized into three main files: cpcivpmn.c contains the main loop and interrupts service routine, Fixedpt_Position_Control.c implements the function for cart position control, and Fixedpt_Speed_Control.c implements the function for cart speed control. All calculations are performed using fixed-point arithmetic for improved performance and precision.

13.1 Referencing Phase for Cart Speed Controller

Initially, the position of the cart is indeterminate and could be anywhere on the shaft. Therefore, it is important to establish an initial positioning such that position control can be exercised efficiently in both right and left directions. This necessitates the initiation of a referencing phase, the objective of which is to align the cart precisely at the center position of the shaft. The initiation of the referencing phase can be executed either by clicking the "Referencing" button in GUI or by writing ".Ref" command through the command window interface.

13.1.1 State Diagram

The initial movement of the cart is directed leftwards until the detection of a switch. Subsequently, it transitions to a rightward trajectory until the detection of the switch positioned on the right. In cases where the cart is already positioned on a switch, it proceeds towards the right until the right switch is detected. Finally, the cart moves towards the center.

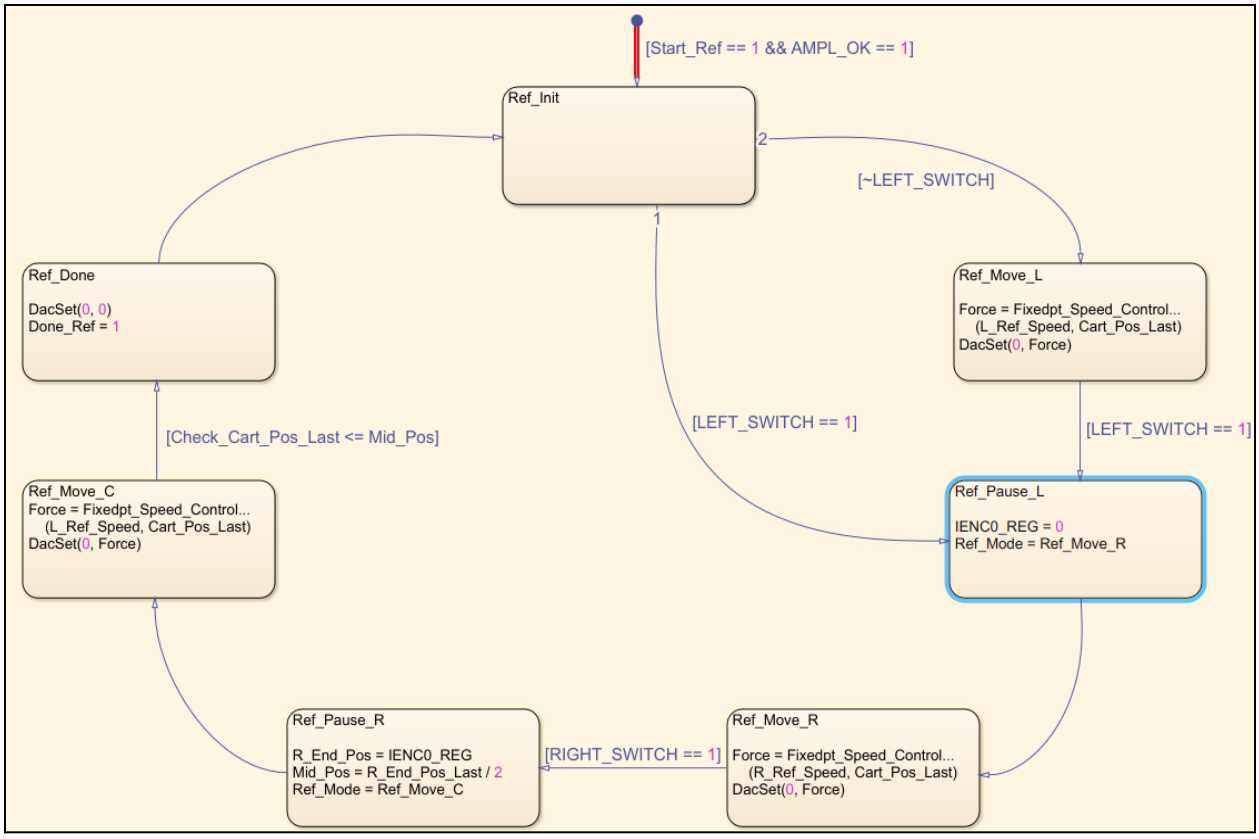


Figure 87: Block diagram for referencing phase

13.1.2 Code Explanation for Referencing Phase

The referencing phase is executed across seven different phases.

1. Ref_Init:

The default stage is activated when both the "Start_Ref" and "AMPL_OK" signals are set to 1. If the left switch is already active, the next stage will be "Ref_Pause_L"; otherwise, it will be "Ref_Move_L".

```
if (Start_Ref)
   {
switch (Ref_Mode)
      {
//Default mode
case Ref_Init:
         L_End_Pos = 0;
         R_End_Pos = 0;
         R_End_Pos_Last = 0;
         Check_Cart_Pos = 0;
         Check_Cart_Pos_Last = 0;
```

```
        Done_Ref = 0;
        Mid_Pos = 0;
//Check if cart is already on left switch
if ((Start_Ref) && (AMPL_OK))
        {
if (LEFT_SWITCH)
          {
            Ref_Mode = Ref_Pause_L;
          }
else
          {
            Ref_Mode = Ref_Move_L;
          }
        }
else
        {
          printf(" *** Inverter is not OK\n\r");
        }
break;
```

2. Ref_Move_L:

During this phase, the cart is set in motion towards the left direction by invoking the speed controller function responsible for regulating its velocity. This function provides the appropriate force values, subsequently provided as input to the "DacSet" function. Following this adjustment, the cart continues its movement until the left switch is detected. Upon detection of the left switch, the cart halts its motion, triggering the invocation of the "Ref_Pause_L" subroutine.

```
case Ref_Move_L:
// IENC1_REG write => latch counters
IENC1_REG = 0;
//Get cart position
Cart_Pos = IENC0_REG;
//Check if number is negative(19th bit is signed)
if ((Cart_Pos & 0x080000) != 0)
{
          Cart_Pos |= 0xFFF00000;
}
// Convert to IEEE standard
Cart_Pos_Last = *(int*)&(Cart_Pos);
//Recieve speed controller force value for left side cart movement
        Force = Fixedpt_Speed_Control(L_Ref_Speed, Cart_Pos_Last);
//Set DAC to control the speed of cart
        DacSet(0, Force);
//Check if left end switch is active
if (LEFT_SWITCH)
```

```
        {
//Set DAC to stop cart
            DacSet(0, 0);
            Ref_Mode = Ref_Pause_L;
        }
break;
```

3. Ref_Pause_L:

This phase is initiated upon activation of the left end switch. The cart is brought to a standstill. Furthermore, the motor position encoder value is reset to zero. This action is crucial to ensure the accurate alignment of the cart to the center position of the shaft. If this action is not performed here, then we will not able to stop the cart in the middle of the shaft.The next stage is set as " Ref_Move_R".

```
case Ref_Pause_L:
//Reset motor position encoder value
        IENC0_REG = 0;
// IENC1_REG write => latch counters
        IENC1_REG = 0;
        Ref_Mode = Ref_Move_R;
break;
```

4. Ref_Move_R:

During this phase, the cart is set in motion towards the right direction by invoking the speed controller function responsible for regulating its velocity. This function provides the appropriate force values, subsequently provided as input to the "DacSet" function. Following this adjustment, the cart continues its movement until the right switch is detected. Upon detection of the right switch, the cart halts its motion, triggering the invocation of the "Ref_Pause_R" subroutine. In this phase, the current cart position is stored in R_End_Pos to calculate the middle position of the shaft.

```
case Ref_Move_R:
// IENC1_REG write => latch counters
        IENC1_REG = 0;
Cart_Pos = IENC0_REG;
//Check if number is negative(19th bit is signed)
if ((Cart_Pos & 0x080000) != 0)
{
Cart_Pos |= 0xFFF00000;
}
// Convert to IEEE standard
Cart_Pos_Last = *(int*)&(Cart_Pos);
//Recieve speed controller force value for right side cart movement
        Force = Fixedpt_Speed_Control(R_Ref_Speed, Cart_Pos_Last);
//Set DAC to control the speed of cart
```

```
        DacSet(0,Force);
// IENC1_REG write => latch counters
        IENC1_REG = 0;
//Get cart position
        R_End_Pos = IENC0_REG;
//Check if right end switch is active
if (RIGHT_SWITCH)
        {
           DacSet(0, 0);
           Ref_Mode = Ref_Pause_R;
        }
break;
```

5. Ref_Pause_R:

This phase is initiated upon activation of the right end switch. The cart is brought to a standstill. The final position of the cart, which was previously recorded and stored during the previous stage, serves as the basis for determining the distance between the left and right end switches of the shaft. By dividing this distance by two, the midpoint of the shaft can be accurately determined. The next stage is set as "Ref_Move_C".

```
case Ref_Pause_R:
// IENC1_REG write => latch counters
        IENC1_REG = 0;
//Check if number is negative(19th bit is signed)
if ((R_End_Pos & 0x080000) != 0)
        {
           R_End_Pos |= 0xFFF00000;
        }
// Convert to IEEE standard
        R_End_Pos_Last = *(int*)&(R_End_Pos);
//Calculating the centre position of the shaft
        Mid_Pos = R_End_Pos_Last / 2;
        Ref_Mode = Ref_Move_C;
break;
```

6. Ref_Move_C:

During this phase, the cart is set in motion towards the left direction by invoking the speed controller function responsible for regulating its velocity. This function provides the appropriate force values, subsequently provided as input to the "DacSet" function. During the leftward movement, the position of the cart is continuously monitored. If the cart position count reaches or falls below the midpoint, the cart is immediately brought to a halt, and the next state is set as "Ref_Done".

```
case Ref_Move_C:
// Convert to IEEE standard
        IENC1_REG = 0;
Cart_Pos = IENC0_REG;
//Check if number is negative(19th bit is signed)
if ((Cart_Pos & 0x080000) != 0)
{
          Cart_Pos |= 0xFFF00000;
}
// Convert to IEEE standard
Cart_Pos_Last = *(int*)&(Cart_Pos);
//Recieve speed controller force value for left side cart movement
Force = Fixedpt_Speed_Control(L_Ref_Speed, Cart_Pos_Last);
//Set DAC to control the speed of cart
        DacSet(0, Force);
// IENC1_REG write => latch counters
        IENC1_REG = 0;
        Check_Cart_Pos = IENC0_REG;
//Check if number is negative(19th bit is signed)
if ((Check_Cart_Pos & 0x080000) != 0)
        {
           Check_Cart_Pos |= 0xFFF00000;
        }
// Convert to IEEE standard
        Check_Cart_Pos_Last = *(int*)&(Check_Cart_Pos);
//Check if the cart has reached the centre point of the shaft
if (Check_Cart_Pos_Last <= (Mid_Pos))
        {
           Ref_Mode = Ref_Done;
        }
break;
```

7. Ref_Done:

During this phase, all variables are initialized to zero, and the "Done_Ref" is set to 1. This is then assessed before starting the position control phase.

```
case Ref_Done:
//Set DAC to stop cart
        DacSet(0, 0);
//Update status of Done_Ref
        Done_Ref = 1;
//Reset all the variables
        L_End_Pos = 0;
        R_End_Pos = 0;
        R_End_Pos_Last = 0;
        Check_Cart_Pos = 0;
```

```
        Check_Cart_Pos_Last = 0;
        Mid_Pos = 0;
        Start_Ref = 0;
//Get pendulum Angle before motor encoder reset
        Angle = IENC1_REG;
//Reset motor position encoder value
        IENC0_REG = 0;
        Ref_Mode = Cntrl_Init;
        printf(" ***Referencing phase done.\n\r");
break;
```

13.2 Cart Position Controller

The referencing phase has been completed, and the cart has now reached the mid-position of the shaft. Consequently, upon clicking “Start Control” button on the graphical user interface (GUI), the controlling phase will become active.

13.2.1 State Diagram

During this phase, the "Fixedpt_Position_Control" function will be called to supply the required force for controlling the movement of the cart in response to any disturbances applied to the pendulum. This phase will be active only after Done_Ref is 1, Start_Control is 1, LEFT_SWITCH and RIGHT_SWITCH is inactive.

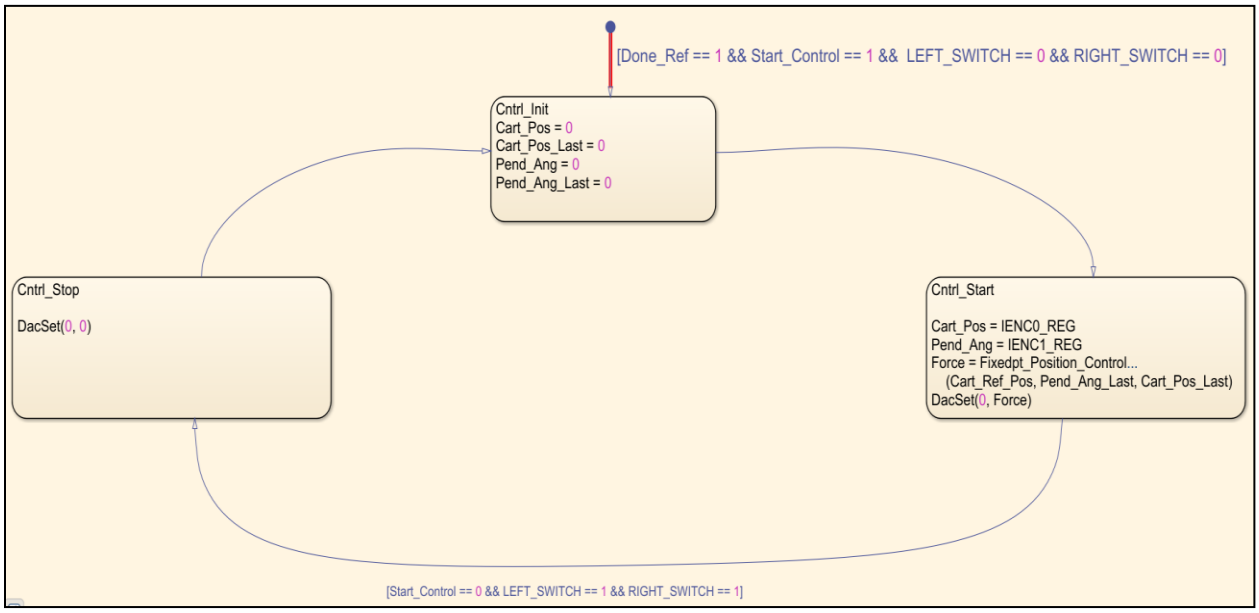


Figure 88: Block diagram for controlling phase

13.2.2 Code Explanation for Controlling Phase

The Controlling phase is executed across three different phases.

1. Cntrl_Init:

The default stage is activated when Done_Ref is 1, Start_Control is 1, LEFT_SWITCH and RIGHT_SWITCH is inactive. All the variables are reset in this stage, and the next state is set as "Cntrl_Start".

```
if(Start_Control)
        {
        if ((LEFT_SWITCH) || (RIGHT_SWITCH))
        {
                // Set DAC for 0 speed
                DacSet(0, 0);
                Inverter_Off();
                printf(" *** To start control phase left or right switch must be inactive.\n\r");
                printf(" *** To achieve this, first perform referencing phase.\n\r");
        }

        elseif (!AMPL_OK)
        {
                 printf("Inverter not OK\n\r");
                DacSet(0, 0);
                Inverter_Off();
    }
        elseif (Done_Ref)
        {
                switch (Control_Mode)
                {
                case Cntrl_Init:
//Reset all the required variables
                Cart_Pos = 0;
                Cart_Pos_Last = 0;
                Pend_Ang = 0;
                Pend_Ang_Last = 0;
                Control_Mode = Cntrl_Start;
                break;
```

2. Cntrl_Start

During this phase, the cart is set in motion by calling the function "Fixedpt_Position_Control". This function provides the appropriate force values, subsequently provided as input to the "DacSet" function. Following this adjustment, the cart continues its movement until the pendulum becomes standstill. If the user demands stop the controlling phase then by clicking stop control in GUI, the next state is set as "Cntrl_Stop".

```
                case Cntrl_Start:
                if (!Start_Control)
                {
//If requested, stop the cart
                        Control_Mode = Cntrl_Stop;
                        break;
                        }
                        // IENC1_REG write => latch counters
                        IENC1_REG = 0;
                        Cart_Pos = IENC0_REG;
//Check if number is negative(19th bit is signed)
                        if ((Cart_Pos & 0x080000) != 0)
                        {
                                Cart_Pos |= 0xFFF00000;
                        }
                // Convert to IEEE standard
                        Cart_Pos_Last = *(int*)&(Cart_Pos);
                        Pend_Ang = IENC1_REG;
//Check if number is negative(19th bit is signed)
                        if ((Pend_Ang & 0x080000) != 0)
                        {
                                Pend_Ang |= 0xFFF00000;
                        }
                         //Convert to IEEE standard
                        Pend_Ang_Last = -(*(short*)&(Pend_Ang));
//Recieve speed controller force value for left side cart movement
                        Force = Fixedpt_Position_Control(Cart_Ref_Pos, Pend_Ang_Last,
Cart_Pos_Last);
//Set DAC to control the speed of cart
                        DacSet(0, Force);
                        break;
```

3. Cntrl_Stop:

During this stage, cart is brought to zero speed immediately and the inverter is deactivated by calling "Inverter_Off" function.

```
        case Cntrl_Stop:
//Set DAC to stop cart
                DacSet(0, 0);
                        Inverter_Off();
                        Start_Control = 0;
                        Control_Mode = Cntrl_Init;
                        break;
```

14 C-Code functions for GUI

14.1 Reference Position Code

```
// reference position via a command from the GUI
elseif (!strncmp(cbuf, "Ref", 3))
      {
if (AMPL_OK)
         {
            L_End_Pos = 0;
            R_End_Pos = 0;
            Mid_Pos = 0;
if (!Start_Control)
            {
               Done_Ref = 0;
               printf(" ***Starting initiallizing phase....\n\r");
               Start_Ref = 1;
               Inverter_On();
            }

if (AMPL_ENABLE)
            {
               printf(" ***Turn on the inverter....\n\r");
               Start_Ref = 0;

            }
if (isr_run == 0)
            {
// start scu timer
               XScuTimer_Start(&pTimer);
               isr_run = 1;

            }
         }
      }
```

When the received command from the GUI is "Ref", the code initiates the reference positioning phase, resets position variables, and starts an inverter it's the initial phase. It enables an amplifier if conditions are met.

14.2 Start Controlling Code

```
// start controlling via a command from the GUI
if (!strncmp(cbuf, "cntrl", 5))
                {

                        printf("cart_ref_pos: (main): %d", Cart_Ref_Pos);
                        if (Cart_Ref_Pos != 1)
                        {
                                printf("cart_ref_pos: %d", Cart_Ref_Pos);
                                printf("I/p error in reference position of cart…\n\r");
                        }
                        else
                        {
                                if (!AMPL_OK)
                                {
                                        printf("Amplifier isn't ok\n\r");
                                }
                                elseif (!Done_Ref)
                                {
                                        printf("Referencing not finished\n\r");
                                }
                                else
                                {
                                        printf("Starting Position Control\n\r");
                                        Inverter_On();
                                        // start controlling
                                        Start_Control = 1;
                                        // value send via sendDataGUI function
                                        sendDataGUI();
                                }
                        }
                }
```

Upon matching the "cntrl" command received from the GUI, the code checks if the cart's reference position is properly set and if the initial referencing phase is complete. If conditions are met, it turns on the inverter, sets 'Start_Control' to 1 for starting the control process, and updates the GUI with the current state via 'sendDataGUI()'. This ensures the system is ready for precise cart position control based on validated preconditions.

14.3 Stop Controlling Code

```
// stop controlling via a command from the GUI
                    if (!strncmp(cbuf, "stop", 4))
                    {
                              DacSet(0, 0);
                              Inverter_Off();
                              Start_Control = 0;
                              printf("###Controlling stop\n");
```

When a "stop" command is detected from the GUI, the code stop the controlling process by setting DAC outputs to zero, turning off the inverter, and resetting the Start_Control flag, effectively stopping the control activity and notifying with a console message. Additionally, it signals the GUI with "###" to stop controlling operations and stop data recording.

14.4 Data Recording On/Off Code

```
// start data recording via a command from the GUI.
                    if (!strncmp(cbuf, "Recordon", 8))
                    {
                              // start data recording
                              RecordData = 1;
                              // value send via sendDataGUI function
                              sendDataGUI();
                              printf("Start data recording \n");
                              printf("Start data recording \n");
                    }
// stop data recording via a command from the GUI.
                    elseif (!strncmp(cbuf, "Recordoff", 9))
                    {
                              RecordData = 0;
                              // value send via sendDataGUI function
                              sendDataGUI();
                    printf("###Stop data recording\n");
printf("###Stop data recording\n");
                    }
                    elseif (cbuf[0] == 'p')
                    {
// value send via sendDataGUI function
                              sendDataGUI();
                    }
```

When the command from the GUI for data recording is detected, upon receiving "Recordon", it initiates data recording by setting ‘RecordData’ to 1, sending current state information to the GUI, and printing a message indicating the start of data recording. Conversely, when "Recordoff" is detected, it stops data recording by setting ‘RecordData’ to 0, updates the GUI, and outputs a message to signify the stop recording. Additionally, any command starting with 'p' triggers sending real-time values to the GUI without altering the recording state.

```
if (RecordData)
        {
        // A data string that starts with "~" is intended for graph plotting within the GUI
        printf("~ %d %d %d %d %d %d\n", Cart_Ref_Pos_Gui, Cart_Pos_Gui, Cart_Velo_Gui,
Pend_Ang_Gui, Pend_AngularVelocity_Gui, Force_Gui);
        printf("~ %d %d %d %d %d %d\n", Cart_Ref_Pos_Gui, Cart_Pos_Gui, Cart_Velo_Gui,
Pend_Ang_Gui, Pend_AngularVelocity_Gui, Force_Gui);
        }
// A data string initiating with “@” is intended to present value in the display box of the GUI
        printf("@ %d %d %d %d %d %d\n", Cart_Ref_Pos_Gui, Cart_Pos_Gui, Cart_Velo_Gui,
Pend_Ang_Gui, Pend_AngularVelocity_Gui, Force_Gui);
        printf("@ %d %d %d %d %d %d\n", Cart_Ref_Pos_Gui, Cart_Pos_Gui, Cart_Velo_Gui,
Pend_Ang_Gui, Pend_AngularVelocity_Gui, Force_Gui);
}
```

When the sendDataGUI function is invoked while recording is active, it transmits both types of data strings: those beginning with "~" and those starting with "@". The data string prefixed with "~" is utilized for plotting data within the graph window of the GUI, whereas the string commencing with "@" is employed to showcase values in the GUI's display box.

15GUI for Cart Pendulum System

15.1 Graphical User Interface

In the context of the cart pendulum system, executing tasks such as referencing, position control, and graph plotting requires a set of instructions to be input into the processor. This initiates the activation of the corresponding actuators to execute the given commands. The process could become very tedious if it required manual repetition of instruction sequences. Consequently, the primary motivation for developing a Graphical User Interface (GUI) is to facilitate a user-friendly platform for issuing commands to the processor for specific actions.

The GUI serves as an accessible portal to the system, acting as its visual representation rather than the system itself. It functions as an intermediary, enabling user interaction with the system's operations. Actions performed through the GUI will impact the system provided there is an established mechanism for the GUI to communicate with the underlying system.

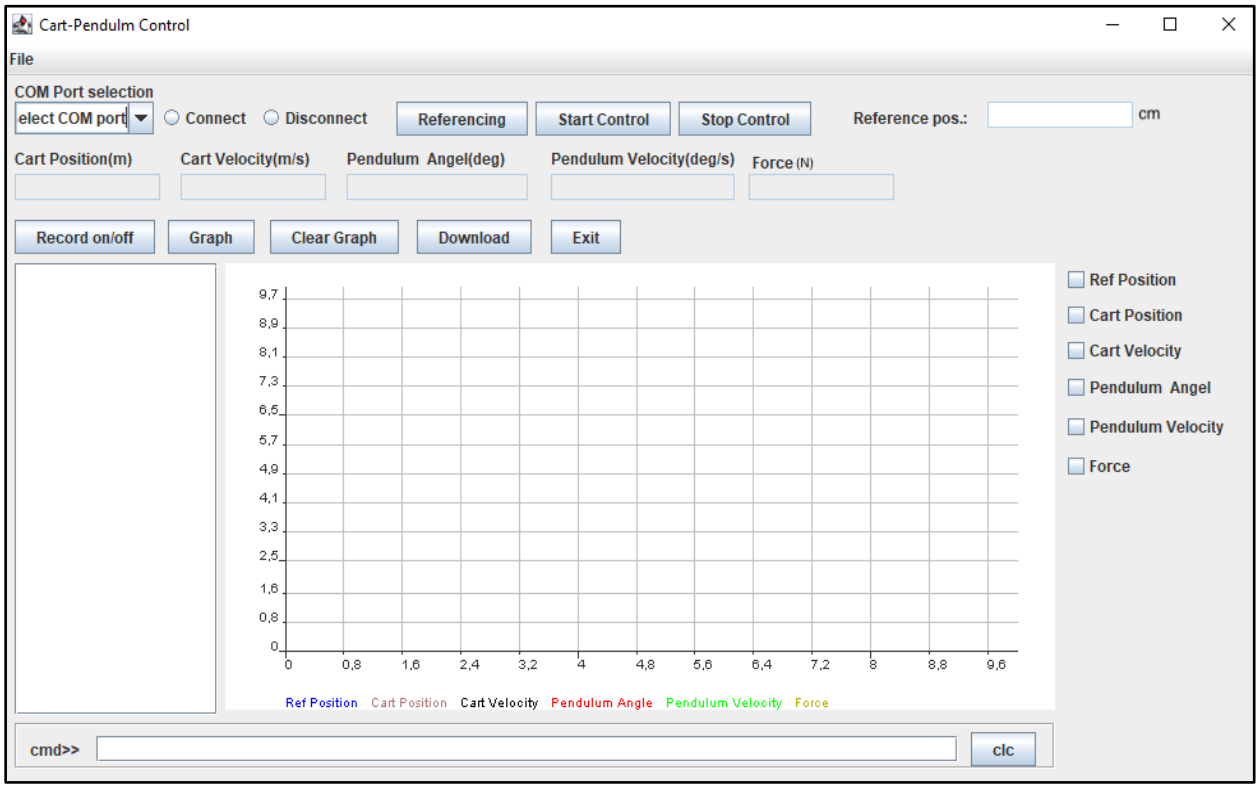


Figure 89: Graphical user interface

15.2 Device Connection (Hardware)

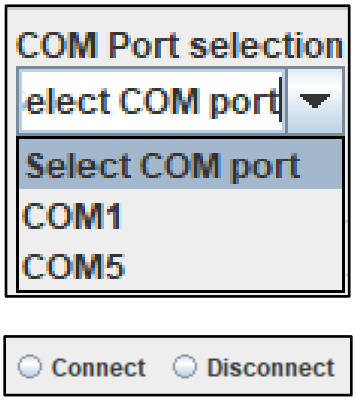


Figure 90: COM port selection and connect and disconnect radio buttons

For seamless experiments with the cart-pendulum system, it's essential to establish a reliable connection between the programmable device and the PC via the GUI, selecting the correct COM port for communication. The GUI features "Connect" and "Disconnect" options to control this connectivity, with the Zynq board relying on serial UART for interaction. Ensuring this connection is properly set up is crucial for smooth and uninterrupted data exchange during experiments, leveraging COM ports like COM1 and COM5 for effective serial communication.

15.3 Control Actions Button

Figure 91: Control action button and Reference position display

Referencing: - To start the system, it's important to measure the distance it will move and position the cart where needed. When the system turns on, a step called "reference mode" must be done. This is achieved using a button labeled "Referencing."

Start Control: - Pressing the start control button activates the control of the cart pendulum system.

Stop Control: - When the user presses the stop control button, it deactivates the control system, stopping the controlling of any disturbances in the pendulum.

Reference pos.: - The reference position of the cart is shown in a specific display box.

15.4 Data Display box

Figure 92: Data display box

This display box presents various metrics such as cart position, cart velocity, pendulum angle, pendulum velocity, and force. When the controller sends data to the GUI, the actual values for the cart pendulum system are then visible in this box. This feature enables users to observe and analyze the position data, which is scaled for easy interpretation on the display.

15.5 Data Recorder button

Figure 93: Data recorder button

Record on/off: - The "Record on/off" button is used to start record and stop record of the data recording.

Graph: - The graph is plotted based on the collected data, and specific graphs can be generated using the checkboxes located beside the graph window.

Download: - Retrieve and store the data for future user analysis by downloading it.

Clear graph: - Erases the recorded data and associated graphs.

Exit: - Disconnect the GUI from the hardware and close the GUI window.

15.6Graph Window

The Graph window is designed to assist in analyzing the system's output. When the ‘plot’ command is entered in the command text field or the ‘graph’ button is pressed, the GUI plots the graph based on the selected checkboxes. As illustrated in the figure below, all physical quantitiescan be plotted, including the reference position to assess the system's steady-state error.

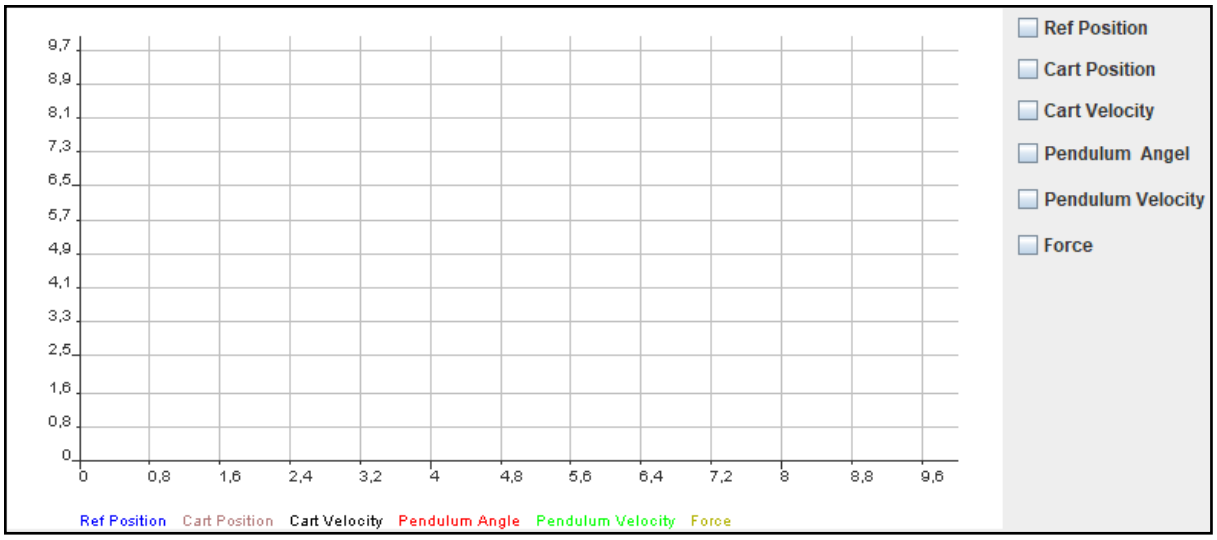


Figure 94: Graph window

Here, we observe checkboxes; by selecting any of these, we can view specific graphs. There are six checkboxes labeled as “Ref Position”, “Cart Position”, “Cart Velocity”, “Pendulum Angle”, “Pendulum Velocity”, and “Force”. It is possible to select more than one checkbox to compare and analyze the values side by side.

15.7 Command Panel

The Command Panel serves as a monitor that shows the system's progress and displays error messages if incorrect input is entered. The below figure shows the default Command Panel.

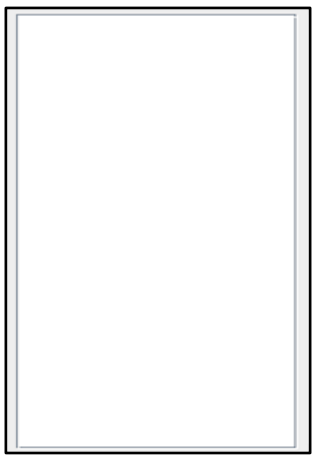


Figure 95: Command panel

To perform specific tasks, the user has the option to either input commands in the command editor window or utilize the provided buttons on the graphical user interface (GUI). If the commands are unknown, users can type the ‘help’ command to view a list of all available commands.

Figure 96: Command line

Command List:

clc - clear text window
clg - clear chart window
dev - list avaliable devices
conn {comX} - connect
disconn - disconnect
.{sendstring} - Sends command to hardware(Zynq board)
.Ref - Start referencing
.cntrl - Start control
.Recordon - Start recording
.Recordoff - Stop recording
.downl - Download data in .txt format
.stop - Stop control
.exit - Terminate
.dacoff - Shutdown DAC
.dacon - Enable DAC
.dac - DacSet
.adc - Read ADC
.pclr - Clear position count
.pos {value} - Display cart position
.r - Read registers
.w - Write register

16Main GUI CodeExplanation

16.1 Hardware Connection Code

```
// com port selection label text
JLabel comPortLbl = new JLabel("COM Port selection");
springLayout.putConstraint(SpringLayout.NORTH, comPortLbl, 4,
SpringLayout.NORTH, FG_frame.getContentPane());
springLayout.putConstraint(SpringLayout.WEST, comPortLbl, 10,
SpringLayout.WEST, FG_frame.getContentPane());
FG_frame.getContentPane().add(comPortLbl);
// com port selection text field
JComboBox comPortField = new JComboBox();
comPortField.setEditable(true);
springLayout.putConstraint(SpringLayout.NORTH, comPortField, 20,
SpringLayout.NORTH, FG_frame.getContentPane());
springLayout.putConstraint(SpringLayout.WEST, comPortField, 10,
SpringLayout.WEST, FG_frame.getContentPane());

/*
 * Combo box for selecting COM port of device
 */
comPortField.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                JComboBox cb = (JComboBox)e.getSource();
                port = (String)cb.getSelectedItem();
//PrintTxtWin("Test Combo Box Selection Detected " + port, 4, true);
        }
});
comPortField.setModel(new DefaultComboBoxModel(new String[] {"Select COM
port", "COM1", "COM5"}));
FG_frame.getContentPane().add(comPortField);


/**
* connect radio button
*/
JRadioButton radioConnect = new JRadioButton("Connect");
springLayout.putConstraint(SpringLayout.WEST, radioConnect, 123,
SpringLayout.WEST, FG_frame.getContentPane());
springLayout.putConstraint(SpringLayout.EAST, comPortField, -4,
SpringLayout.WEST, radioConnect);
springLayout.putConstraint(SpringLayout.NORTH, radioConnect, 0,
SpringLayout.NORTH, comPortField);

/*
 * Radio button to establish connection with the device.
```

```
*/
buttonGroup.add(radioConnect);
radioConnect.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		if(port == "Select COM port" || port == null ) {
			PrintTxtWin("select COM port", 3, true);
			buttonGroup.clearSelection();
		}
		else
		{
			PrintTxtWin("select port: " + port, 3, true);
			CommandHandler("conn " + port);

		}
	}
});
FG_frame.getContentPane().add(radioConnect);


/**
* disconnect radio button
*/
JRadioButton radioDisConnect = new JRadioButton("Disconnect");
springLayout.putConstraint(SpringLayout.NORTH, radioDisConnect, 0,
SpringLayout.NORTH, comPortField);
springLayout.putConstraint(SpringLayout.WEST, radioDisConnect, 6,
SpringLayout.EAST, radioConnect);

/*
 * Radio button for disconnecting from the device.
 */
buttonGroup.add(radioDisConnect);
radioDisConnect.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
		if(port == "Select COM port" || port == null ) {
		PrintTxtWin("COM port isn't selected", 3, true);
		buttonGroup.clearSelection();

		}
		else {
		PrintTxtWin("Disconnected: " + port, 3, true);
		CommandHandler("disconn");
		}
	}
});

FG_frame.getContentPane().add(radioDisConnect);
```

The Java Swing framework (GUI) is employed to construct a user interface for serial communication in the application. A JLabel component, ‘comPortLbl’, and a JComboBox, ‘comPortField’, facilitate user selection of the communication port from a dropdown menu pre-populated with options "COM1" and "COM5". JRadioButton components, ‘radioConnect’ and ‘radioDisConnect’, provide interactive elements for establishing or terminating the connection, with ActionListener implementations that invoke a custom ‘CommandHandler’ method to process the commands "conn" or "disconn" based on the user's selection.

16.2 Control Actions Button Code

```
    /**
                    *  Referencing button (Initialization button)
                    */
                    JButton refBtn = new JButton("Referencing");
                    springLayout.putConstraint(SpringLayout.NORTH, refBtn, 0, SpringLayout.NORTH,
comPortField);
                    springLayout.putConstraint(SpringLayout.WEST, refBtn, 19, SpringLayout.EAST,
radioDisConnect);

                    refBtn.addActionListener(new ActionListener() {
                       @Override
                       public void actionPerformed(ActionEvent arg0) {
                          PrintTxtWin("Referencing Initialize", 4, true);
                          CommandHandler(".Ref");
                       }
                    });
                    FG_frame.getContentPane().add(refBtn);


                    /**
                    * Start controlling button
                    */
                    JButton startCtrlBtn = new JButton("Start Control");
                    springLayout.putConstraint(SpringLayout.NORTH, startCtrlBtn, 0,
SpringLayout.NORTH, comPortField);
                    springLayout.putConstraint(SpringLayout.WEST, startCtrlBtn, 6, SpringLayout.EAST,
refBtn);

                    startCtrlBtn.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                    }
                    });
                    startCtrlBtn.addActionListener(new ActionListener() {
                       @Override
                       public void actionPerformed(ActionEvent arg0) {
                          PrintTxtWin("command sent : .cntrl ", 5, true);
```

```
                    CommandHandler(".cntrl ");
                    refPosTextField.setEnabled(true);
                }
            });
            FG_frame.getContentPane().add(startCtrlBtn);


            /**
            * stop controlling button
            */
            JButton stopCtrlBtn = new JButton("Stop Control");
            springLayout.putConstraint(SpringLayout.NORTH, stopCtrlBtn, 0,
SpringLayout.NORTH, comPortField);
            springLayout.putConstraint(SpringLayout.WEST, stopCtrlBtn, 6, SpringLayout.EAST,
startCtrlBtn);
            stopCtrlBtn.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                            refPosTextField.setEnabled(false);
                            CommandHandler(".stop");
                    }
            });

            FG_frame.getContentPane().add(stopCtrlBtn);
```

In the code mentioned above, three JButtons are indicated, "Referencing," "Start Control," and "Stop Control"were integrated. The ActionListener attached to each JButton triggers command execution through the CommandHandler method, affecting system states. "Referencing" invokes ‘.Ref’, "Start Control" sends ‘.cntrl’, and "Stop Control" issues ‘.stop’ and disables the same text field. SpringLayout manages component positioning, ensuring a structured GUI layout.

16.3 Data Display Box Code

```
/**
            * Cart velocity label text and text field
            */
            JLabel cartVelLbl = new JLabel("Cart Velocity(m/s)");
            springLayout.putConstraint(SpringLayout.NORTH, cartVelLbl, 55,
SpringLayout.NORTH, FG_frame.getContentPane());
            springLayout.putConstraint(SpringLayout.WEST, cartVelLbl, 140,
SpringLayout.NORTH, FG_frame.getContentPane());
            FG_frame.getContentPane().add(cartVelLbl);
```

```
                    // Cart velocity text field
                    cartVelTextField = new JTextField();
                    springLayout.putConstraint(SpringLayout.NORTH, cartVelTextField, 75,
SpringLayout.NORTH, FG_frame.getContentPane());
                    springLayout.putConstraint(SpringLayout.WEST, cartVelTextField, 140,
SpringLayout.NORTH, FG_frame.getContentPane());
                    FG_frame.getContentPane().add(cartVelTextField);
                    cartVelTextField.setEditable(false);
                    cartVelTextField.setColumns(10);


/**
                    * Cart position text field
                    */
                    cartPosTextField = new JTextField();
                    springLayout.putConstraint(SpringLayout.NORTH, cartPosTextField, 75,
SpringLayout.NORTH, FG_frame.getContentPane());
                    springLayout.putConstraint(SpringLayout.WEST, cartPosTextField, 10,
SpringLayout.NORTH, FG_frame.getContentPane());
                    cartPosTextField.setColumns(10);
                    cartPosTextField.setEditable(false);
                    FG_frame.getContentPane().add(cartPosTextField);
                    // Cart position label text
                    JLabel cartPosLbl = new JLabel("Cart Position(m)");
                    springLayout.putConstraint(SpringLayout.NORTH, cartPosLbl, 55,
SpringLayout.NORTH, FG_frame.getContentPane());
                    springLayout.putConstraint(SpringLayout.WEST, cartPosLbl, 10,
SpringLayout.NORTH, FG_frame.getContentPane());
                    FG_frame.getContentPane().add(cartPosLbl);


/**
                    * Pendulum angle text field
                    */
                    pendAngTextField = new JTextField();
                    springLayout.putConstraint(SpringLayout.NORTH, pendAngTextField, 75,
SpringLayout.NORTH, FG_frame.getContentPane());
                    springLayout.putConstraint(SpringLayout.WEST, pendAngTextField, 270,
SpringLayout.NORTH, FG_frame.getContentPane());
                    springLayout.putConstraint(SpringLayout.EAST, pendAngTextField, 0,
SpringLayout.EAST, refBtn);
                    pendAngTextField.setColumns(10);
                    pendAngTextField.setEditable(false);
                    FG_frame.getContentPane().add(pendAngTextField);
                    // Pendulum angle label text
                    JLabel pendAngLbl = new JLabel("Pendulum  Angel(deg)");
                    springLayout.putConstraint(SpringLayout.NORTH, pendAngLbl, 55,
SpringLayout.NORTH, FG_frame.getContentPane());
```

```
                springLayout.putConstraint(SpringLayout.WEST, pendAngLbl, 270,
SpringLayout.NORTH, FG_frame.getContentPane());
                FG_frame.getContentPane().add(pendAngLbl);


/**
                * Pendulum velocity text field
                */
                pendVelTextField = new JTextField();
                springLayout.putConstraint(SpringLayout.NORTH, pendVelTextField, 75,
SpringLayout.NORTH, FG_frame.getContentPane());
                springLayout.putConstraint(SpringLayout.WEST, pendVelTextField, 430,
SpringLayout.NORTH, FG_frame.getContentPane());
                pendVelTextField.setColumns(10);
                pendVelTextField.setEditable(false);
                FG_frame.getContentPane().add(pendVelTextField);
                // Pendulum velocity label text
                JLabel pendVelLbl = new JLabel("Pendulum Velocity(deg/s)");
                springLayout.putConstraint(SpringLayout.EAST, pendVelTextField, 0,
SpringLayout.EAST, pendVelLbl);
                springLayout.putConstraint(SpringLayout.NORTH, pendVelLbl, 55,
SpringLayout.NORTH, FG_frame.getContentPane());
                springLayout.putConstraint(SpringLayout.WEST, pendVelLbl, 430,
SpringLayout.NORTH, FG_frame.getContentPane());
                FG_frame.getContentPane().add(pendVelLbl);



/**
                * Force lable text
                */
                JLabel forceLbl = new JLabel("Force (N)");
                springLayout.putConstraint(SpringLayout.NORTH, forceLbl, 12,
SpringLayout.SOUTH, stopCtrlBtn);
                springLayout.putConstraint(SpringLayout.WEST, forceLbl, 14, SpringLayout.EAST,
pendVelLbl);
                FG_frame.getContentPane().add(forceLbl);
                // Force text field
                forceTextField = new JTextField();
                springLayout.putConstraint(SpringLayout.NORTH, forceTextField, 75,
SpringLayout.NORTH, FG_frame.getContentPane());
                springLayout.putConstraint(SpringLayout.WEST, forceTextField, 585,
SpringLayout.NORTH, FG_frame.getContentPane());
                forceTextField.setColumns(10);
                forceTextField.setEditable(false);
                FG_frame.getContentPane().add(forceTextField);
```

In the code, JLabels describe JTextFields, which are set to display real-time data in a non-editable format and are positioned using SpringLayout.JLabel components labeled "CartVelocity(m/s)", "Cart Position(m)", "Pendulum Angle(deg)", "Pendulum Velocity(deg/s)", and "Force (N)" are instantiated to provide clear identification of the respective data. Corresponding JTextField components are configured to display the values for cart velocity, cart position, pendulum angle, pendulum velocity, and force, with constraints set via a SpringLayout manager to organize their positions within the frame.

16.4 Data RecorderButton Code

```
/**
                    * Record On/Off button
                   */
                   JToggleButton tglRecordOnOff = new JToggleButton("Record on/off");
                    springLayout.putConstraint(SpringLayout.NORTH, tglRecordOnOff, 110,
SpringLayout.NORTH, FG_frame.getContentPane());
                    springLayout.putConstraint(SpringLayout.WEST, tglRecordOnOff, 10,
SpringLayout.NORTH, FG_frame.getContentPane());
                              tglRecordOnOff.addActionListener(new ActionListener() {
                              public void actionPerformed(ActionEvent e) {
                              if (tglRecordOnOff.isSelected()) {
                              if (SerialNetw.getConName() != null) {
                              Downl_Cnt = 0;
                              Pb_NValues = 0;
                              Pb_Ready = false;
                              } else {
                              PrintTxtWin("no connected device", 3, true);
                              }
                              CommandHandler(".Recordon");
                              }else{
                              CommandHandler(".Recordoff");
                              }}
                              });
                              FG_frame.getContentPane().add(tglRecordOnOff);


                    /**
                    * Graph button for plotting data in the graph panel
                   */
                   JButton GraphBtn = new JButton("Graph");
                   springLayout.putConstraint(SpringLayout.NORTH, GraphBtn, 110,
SpringLayout.NORTH, FG_frame.getContentPane());
                   springLayout.putConstraint(SpringLayout.WEST, GraphBtn, 130,
SpringLayout.NORTH, FG_frame.getContentPane());
```

```
FG_frame.getContentPane().add(GraphBtn);

GraphBtn.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
        PrintTxtWin("Graph Plot", 4, true);
        ClearChart();
        CommandHandler("plot");
        }
});


 /**
 * Clear Graph button to remove data from the graph panel
 */
JButton clcGraphBtn = new JButton("Clear Graph");
springLayout.putConstraint(SpringLayout.NORTH, clcGraphBtn, 110,
SpringLayout.NORTH, FG_frame.getContentPane());
springLayout.putConstraint(SpringLayout.WEST, clcGraphBtn, 210,
SpringLayout.NORTH, FG_frame.getContentPane());
FG_frame.getContentPane().add(clcGraphBtn);

clcGraphBtn.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent arg0) {
                CommandHandler("clg");
                PrintTxtWin("clear Graph", 3, true);
        }
});

 /**
 * Download button is used to download the data file
 */
JButton DownloadBtn = new JButton("Download");
springLayout.putConstraint(SpringLayout.NORTH, DownloadBtn, 110,
SpringLayout.NORTH, FG_frame.getContentPane());
springLayout.putConstraint(SpringLayout.WEST, DownloadBtn, 325,
SpringLayout.NORTH, FG_frame.getContentPane());
FG_frame.getContentPane().add(DownloadBtn);

DownloadBtn.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent arg0) {
      CommandHandler("downl");
                PrintTxtWin("Data file Downloaded", 14, true);
   }
});
```

```
/**
 * Exit button is used to close the GUI
*/
JButton exitBtn = new JButton("Exit");
springLayout.putConstraint(SpringLayout.NORTH, exitBtn, 110,
SpringLayout.NORTH, FG_frame.getContentPane());
springLayout.putConstraint(SpringLayout.WEST, exitBtn, 430, SpringLayout.NORTH,
FG_frame.getContentPane());
FG_frame.getContentPane().add(exitBtn);

exitBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                String dev_name;
                dev_name = SerialNetw.getConName();
                if (dev_name != null) {
                        SerialNetw.spDisconn(dev_name);
                }
    DispUpdate_Timer.stop();
    System.exit(0);
        }
});
```

The JToggleButton named 'tglRecordOnOff' toggles the recording state and interacts with the serial network, with checks for device connectivity before executing recording commands. The JButton 'GraphBtn' initiates data plotting on the graph panel, and 'clcGraphBtn' is configured to clear this panel, invoking command handlers for each action. Additionally, 'DownloadBtn' facilitates the downloading of data files, while 'exitBtn' implements the application's termination process, ensuring device disconnection and stopping display updates before exiting the system.

16.5Graph Window Code

```
// Create graph window
        private void CreateChart()
        {
                int  k;
                chart = new Chart2D();
                for (k = 0; k < NTRACES; k++) {
                mtraces[k] = new Trace2DSimple();
                chart.addTrace(mtraces[k]);
                }
```

```
        mtraces[0].setColor(Color.BLUE);                    mtraces[0].setName("Ref Position");
        mtraces[1].setColor(Color.PINK.darker());           mtraces[1].setName("Cart Position");
        mtraces[2].setColor(Color.BLACK);                   mtraces[2].setName("Cart Velocity");
        mtraces[3].setColor(Color.RED);                     mtraces[3].setName("Pendulum Angle");
        mtraces[4].setColor(Color.GREEN);                   mtraces[4].setName("Pendulum Velocity");
        mtraces[5].setColor(Color.YELLOW.darker());         mtraces[5].setName("Force");
        Gr_panel.setLayout(new BorderLayout(0, 0));
        Gr_panel.add(chart,BorderLayout.CENTER);
        GridBagLayout globalChart = new GridBagLayout();
        globalChart.columnWidths = new int[]{0};
        globalChart.rowHeights = new int[]{0};
        globalChart.columnWeights = new double[]{};
        globalChart.rowWeights = new double[]{};
        chart.setLayout(globalChart);
        SpringLayout chartSpringLayout = new SpringLayout();
        chart.setLayout(chartSpringLayout);
        chart.setGridColor(Color.lightGray);
        IAxis axisX = chart.getAxisX();
        axisX.setPaintGrid(true);
        IAxis axisY = chart.getAxisY();
        axisY.setPaintGrid(true);
        Gr_panel.add(chart);
        Gr_panel.setSize(100,200);
        chart.setVisible(true);
        Gr_panel.setVisible(true);
        Gr_panel.repaint();
            }

// received real-time data from hardware and update GUI continuously
private void UpdateDynamic()
            {
                    String  recv_s, string_num;
                    String  splits[];
                    String  splitValue[];
                    int  splitLineNo;
                    int k;
                    int p;

                    while ((recv_s = SerialNetw.ReadString()) != null)
// System.out.println("["+ recv_s + "]");
                    {
                        splits = recv_s.split("\n");
                        splitLineNo = splits.length;
                    for (k = 0; k < splitLineNo; k++)
                            {
                            if (splits[k].startsWith("$") || splits[k].startsWith("~"))
                            {
                             PrintTxtWin("+", 1, true);
```

```
                    Downl_Cnt++;
                    if (Downl_Cnt >= 25) {
                        Downl_Cnt = 0;
                        PrintTxtWin("\n   ", 1, false);
                        }
                    PrintTxtWin("\ndata recording started", 1, false);
                    if (Pb_NValues < NROWS)
        {
          splitValue = splits[k].split(" ");
    try {
    for (p = 1; p < splitValue.length; p++)
            {
            if (splitValue[p].contains("\r")) {
            string_num = splitValue[p].substring(0, splitValue[p].length()-1);
            } else {
            string_num = splitValue[p];
            }
            ReceivedData = (double)Integer.parseInt(string_num);
    Plot_Buffer[Pb_NValues][p-1] = ReceivedData * ScalingArray[p-1];
            }
            Pb_NValues++;
              } catch  (NumberFormatException e)
              {
                    System.out.println(e.toString());
              }
            }
            if (splits[k].startsWith("~")) {
            // SerialNetw.SendString("p\n");
             CommandHandler(".p");
              }
    }
    if(splits[k].startsWith("@"))
    {
      splitValue = splits[k].split(" ");
    try {
        RefPosition =  (double)Integer.parseInt(splitValue[1]) * RefScaling;
        CartPosition = (double)Integer.parseInt(splitValue[2]) * CartPosScaling;
        CartVelocity = (double)Integer.parseInt(splitValue[3]) * CartVelScaling;
        PendulmAngle = (double)Integer.parseInt(splitValue[4]) * PendAngScaling;
        PendulmAngulerVelocity = (double)Integer.parseInt(splitValue[5]) * PendAngVelScaling;
    if (splitValue[6].contains("\r")) {
            string_num = splitValue[6].substring(0, splitValue[6].length()-1);
         } else {
            string_num = splitValue[6];
                }
         Force = (double)Integer.parseInt(string_num) * ForceScaling;

    refPosTextField.setText(String.valueOf(fbb.format(RefPosition)));
```

```
        forceTextField.setText(String.valueOf(fbb.format(Force)));
        cartPosTextField.setText(String.valueOf(fbb.format(CartPosition)));
        cartVelTextField.setText(String.valueOf(fbb.format(CartVelocity)));
        pendAngTextField.setText(String.valueOf(fbb.format(PendulmAngle)));
        pendVelTextField.setText(String.valueOf(fbb.format(PendulmAngulerVelocity)));
                } catch(NumberFormatException e)
                {
                        System.out.println(e.toString());
                }
                        }
                        if (splits[k].startsWith("###"))
                        {
                PrintTxtWin("\nDownload finished.\n", 1, true);
                Pb_Ready = true;
                } else {
                PrintTxtWin(splits[k], 0, true);
                }
            }
        }
    }

      // Generate the specific graph that is selected by the corresponding checkbox
        else if (command.startsWith("plot")) {
        if (!Pb_Ready) {
          PrintTxtWin("*** no plot data available", 3, true);
          } else if (Pb_NValues < 2) {
          System.out.printf("pb_nvalue: %d\n",Pb_NValues);
          PrintTxtWin("*** no data points", 3, true);
          }
        else {
        if (command.equals("plot")) {
          ClearChart();
        for (k = 0; k < Pb_NValues; k++) {
        if (Pb_NValues < NROWS) {
        for (q = 0; q < NTRACES; q++) {

        //Plot Reference Position's graph
        if ((q==0) && checkBoxRefPos) {
        mtraces[0].addPoint((double)k*SampleTime, Plot_Buffer[k][q]);
        }
        //Plot cart Position's graph
        if ((q==1) && checkBoxCartPos) {
        mtraces[1].addPoint((double)k*SampleTime, Plot_Buffer[k][q]);
        }
        //Plot cart velocity's graph
        if ((q==2) && checkBoxCartVel) {
        mtraces[2].addPoint((double)k*SampleTime, Plot_Buffer[k][q]);
        }
```

```
            //Plot pendulum Angle's graph
            if ((q==3) && checkBoxPendAng) {
            mtraces[3].addPoint((double)k*SampleTime, Plot_Buffer[k][q]);
            }
            //Plot Pendulum velocity's graph
            if ((q==4) && checkBoxPendVel) {
            mtraces[4].addPoint((double)k*SampleTime, Plot_Buffer[k][q]);
            }
            Plot force's graph
            if ((q==5) && checkBoxForce) {
            mtraces[5].addPoint((double)k*SampleTime, Plot_Buffer[k][q]);
            }
           }
          }
         }
        }
       }
```

The ‘UpdateDynamic’ function, which runs at preset intervals, checks the data received from the controller through “SerialNetw.ReadString()” via UART. Data intended for recording, starting with the character "~", is processed upon reception; it is split, scaled, and stored in the ‘Plot_Buffer’ array. In contrast, data meant for display. box on the GUI, prefixed with "@", is scaled accordingly and populated into the display boxes. Decisions on data handling are made based on the initial character of the received strings.

To plot data on the graph, the desired states must be chosen using the checkboxes provided right side of the graph window, followed by clicking the Plot button on the GUI or issuing a “plot” command. Initiating the plotting process, the code first clears any pre-existing graphs and proceeds to iterate over the ‘Plot_Buffer’, plotting the data for the states that have been selected. The plotting is conditional, based on the user's checkbox selections for metrics such as "Reference Position" and "Cart Velocity," and it dynamically generates each graph by appending points using the specified ‘SampleTime’ intervals from ‘Plot_Buffer’.

16.6 Download Record Data File Code

```
else if (command.eqals("downl")) {
                                if (SerialNetw.getConName() != null) {
                                  SerialNetw.SendString("p\n");
                                  PrintTxtWin(" => starting download:\n ", 1, false);
                                // Download Recorded data file
                                try {
                                  GetDateTime();
```

```
                    String fname = "RecordData_"+DateTime+"_.txt";
                    FileWriter writeFile = new FileWriter(fname);
                    // File header names for the data
                    writeFile.write("CartRefPos(m)   CartPos(m)     CartVel(m/s)
PendAng(deg)          PendAngVel(deg/s)      Force(N)\n");
            for (k = 0; k < Pb_NValues; k++) {
                    if (Pb_NValues < NROWS) {
                    for (q = 0; q < NTRACES; q++) {
                    //One by one transfer all data from the plot buffer into the file
                    writeFile.write(String.valueOf(fbb.format(Plot_Buffer[k][q])) + "\t\t");
                    }
                    writeFile.write("\n");
                            }
                            }
                            writeFile.close();
                            } catch (IOException e) {
                            System.out.println(e.toString());
                            e.printStackTrace();
                            }
                    } else {
                            PrintTxtWin("*** no connected device", 3, true);

                    }
            }
```

Upon triggering the Download function, either through a button click or the "downl" command, the system verifies device connectivity. It then proceeds to create a uniquely named file, designated by the timestamp from the 'GetDateTime()' function. This file is crafted to include a header, after which data from the 'Plot_Buffer' is methodically entered, formatted into clearly defined columns that align with each system metric for ease of analysis.

16.7 Check box

```
            /*
            * Check Box Selection
            */
            // check Box for cart velocity graph
            JCheckBox graphCartVel = new JCheckBox("Cart Velocity");
            springLayout.putConstraint(SpringLayout.NORTH, graphCartVel, 200,
SpringLayout.NORTH, FG_frame.getContentPane());
            springLayout.putConstraint(SpringLayout.WEST, graphCartVel, 6,
SpringLayout.EAST, Gr_panel);
            graphCartVel.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent arg0) {
```

```
                checkBoxCartVel = graphCartVel.isSelected();
                }
                });
        FG_frame.getContentPane().add(graphCartVel);


        // check Box for Cart Position graph
        JCheckBox graphCartPos = new JCheckBox("Cart Position");
        springLayout.putConstraint(SpringLayout.WEST, graphCartPos, 6,
SpringLayout.EAST, Gr_panel);
        springLayout.putConstraint(SpringLayout.SOUTH, graphCartPos, -6,
SpringLayout.NORTH, graphCartVel);
        springLayout.putConstraint(SpringLayout.EAST, graphCartPos, -43,
SpringLayout.EAST, FG_frame.getContentPane());
        graphCartPos.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                        checkBoxCartPos = graphCartPos.isSelected();
                }
                });
        FG_frame.getContentPane().add(graphCartPos);


        // check Box for Pendulum Angel graph
        JCheckBox graphPendAng = new JCheckBox("Pendulum  Angel");
        springLayout.putConstraint(SpringLayout.SOUTH, graphCartVel, -6,
SpringLayout.NORTH, graphPendAng);
        springLayout.putConstraint(SpringLayout.NORTH, graphPendAng, 225,
SpringLayout.NORTH, FG_frame.getContentPane());
        springLayout.putConstraint(SpringLayout.WEST, graphPendAng, 6,
SpringLayout.EAST, Gr_panel);
        graphPendAng.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                        checkBoxPendAng = graphPendAng.isSelected();
                }
                });
        FG_frame.getContentPane().add(graphPendAng);


        // check Box for Pendulum Velocity graph
        JCheckBox graphPendVel = new JCheckBox("Pendulum Velocity");
        springLayout.putConstraint(SpringLayout.NORTH, graphPendVel, 6,
SpringLayout.SOUTH, graphPendAng);
        springLayout.putConstraint(SpringLayout.WEST, graphPendVel, 6,
SpringLayout.EAST, Gr_panel);
        graphPendVel.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                        checkBoxPendVel = graphPendVel.isSelected();
                }
```

```
                        });
                FG_frame.getContentPane().add(graphPendVel);


                // check Box for Force graph
                JCheckBox forceCB = new JCheckBox("Force");
                springLayout.putConstraint(SpringLayout.NORTH, forceCB, 6, SpringLayout.SOUTH,
graphPendVel);
                springLayout.putConstraint(SpringLayout.WEST, forceCB, 6, SpringLayout.EAST,
Gr_panel);
                forceCB.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent arg0) {
                                checkBoxForce = forceCB.isSelected();
                        }
                        });
                FG_frame.getContentPane().add(forceCB);

        // check Box for reference position graph
        JCheckBox graphRefPos = new JCheckBox("Ref Position");
        springLayout.putConstraint(SpringLayout.NORTH, graphRefPos, 0, SpringLayout.NORTH,
scrollPane);
        springLayout.putConstraint(SpringLayout.WEST, graphRefPos, 6, SpringLayout.EAST,
Gr_panel);

                forceCB.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent arg0) {
                        checkBoxRefPos = graphRefPos.isSelected();
                        }
                        });
                FG_frame.getContentPane().add(graphRefPos);
```

This code facilitates graph plotting in a graphical window based on the selections made through checkboxes. It allows for the display of specific graphs, ensuring only those corresponding to the selected checkboxes are plotted. Users have the flexibility to select multiple checkboxes, enabling the comparison and analysis of different graphs simultaneously.

17Test Results Analysis

Data was gathered across multiple scenarios, and the corresponding outcomes were graphically represented within the GUI's Graph Panel for detailed examination. This facilitated a comprehensive analysis of the different case results through visual representation.

17.1 Active Control

17.1.1 Pendulum Angle

Analyze the angle of the pendulum while the control system is operational, following the introduction of certain disturbances to the pendulum.

Pendulum angle:

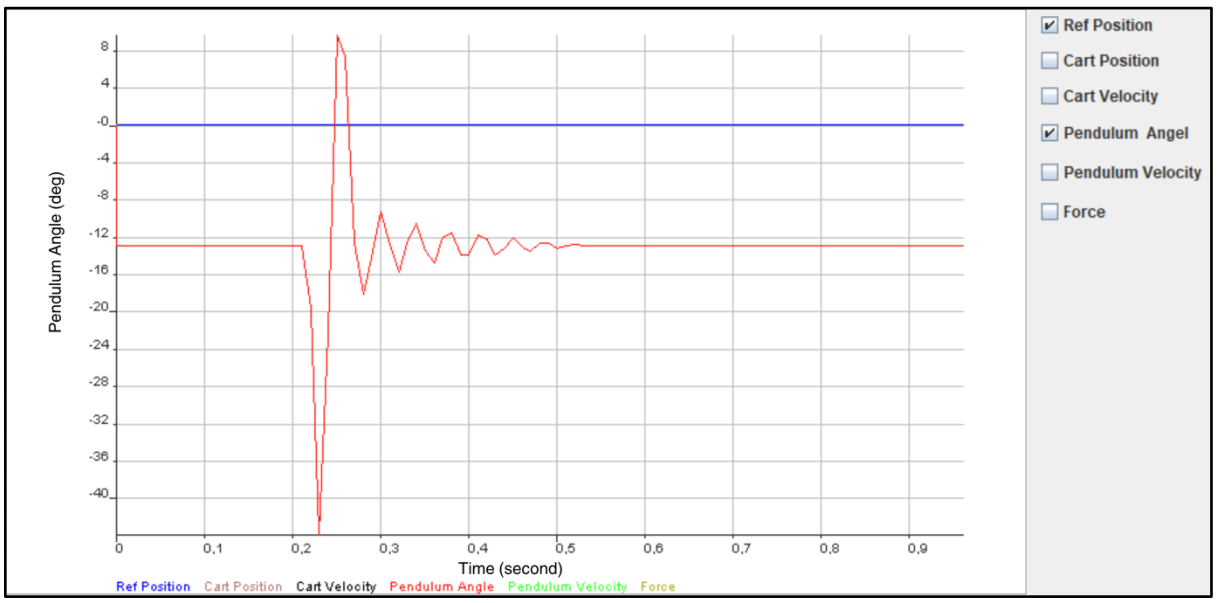


Figure 97: Pendulum angle graph

The presented graph shows the angle of the pendulum in response to an applied disturbance. According to the data sheet, the greatest disturbance recorded in the pendulum angle reached approximately-43.5degrees.The negative angle indicates that the disturbance caused the pendulum to move to the left, as leftward motion is assigned a negative value. The pendulum stabilized and returned to its settled position within 0.55 seconds.

17.1.2 Pendulum Velocity

Analyzethe velocity of the pendulum while the control system is active, after introducing some disturbances to the pendulum.

Pendulum velocity graph:

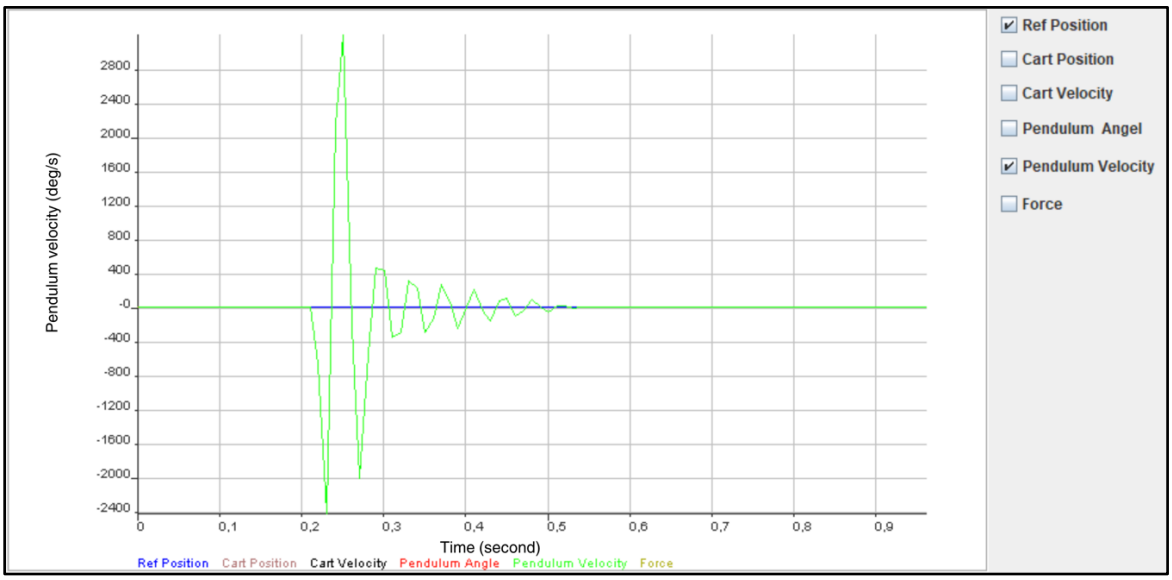


Figure 98: Pendulum velocity graph

The graph above depicts the velocity of the pendulum when a disturbance is applied to it. Initially, the pendulum's velocity is zero, but it increases significantly due to an external force exerted on it. The graph clearly shows a peak in initial velocity between 0.2 and 0.3 seconds, after which there is a marked decrease in velocity as the control system becomes active, indicating its role in stabilizing the pendulum. The system achieves stability within 0.55 seconds.

17.1.3 Cart Displacement

Analyze the displacement of the cart while the control system is active, after introducing some disturbances to the pendulum.

Cart Displacement graph:

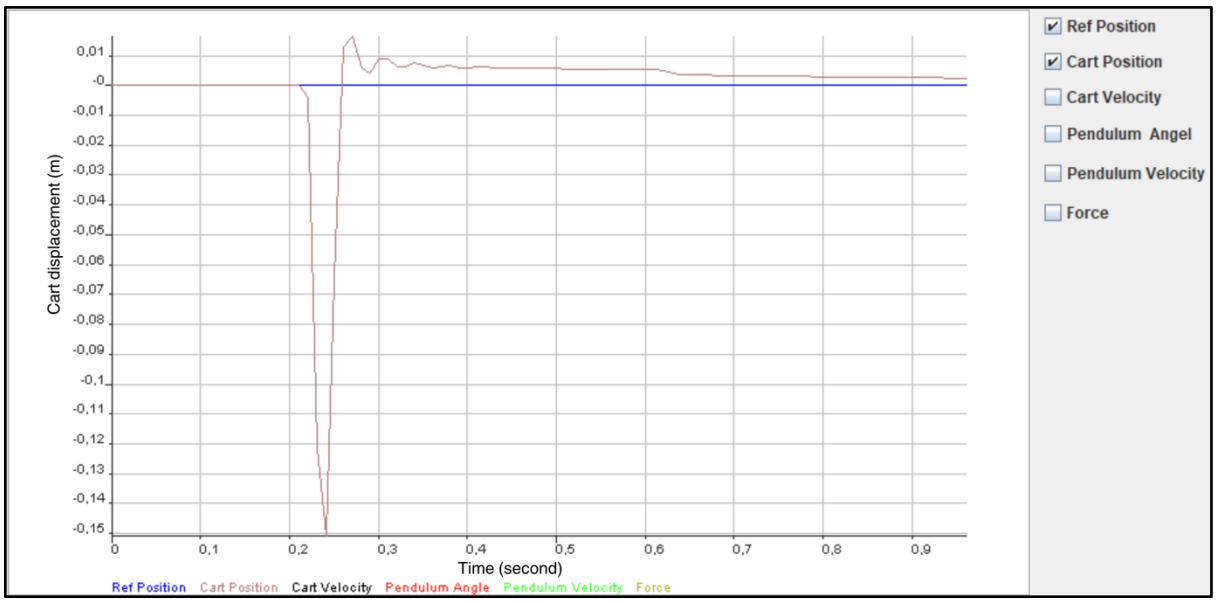


Figure 99: Cart displacement graph

The graph above displays the displacement of the cart due to a disturbance in the pendulum. At approximately 0.24 seconds, a disturbance of around -43.5 degrees is introduced to the pendulum, causing the cart to diverge from its reference position as it acts to dampen the oscillations of the pendulum. Subsequently, the cart moves back and re-aligns with the reference position, achieving damping of the pendulum oscillations. The cart resettles to its original position within roughly 0.8 seconds.

17.1.4 Cart Velocity

Analyze the velocity of the cart while the control system is active, after introducing some disturbances to the pendulum.

Cart velocity graph:

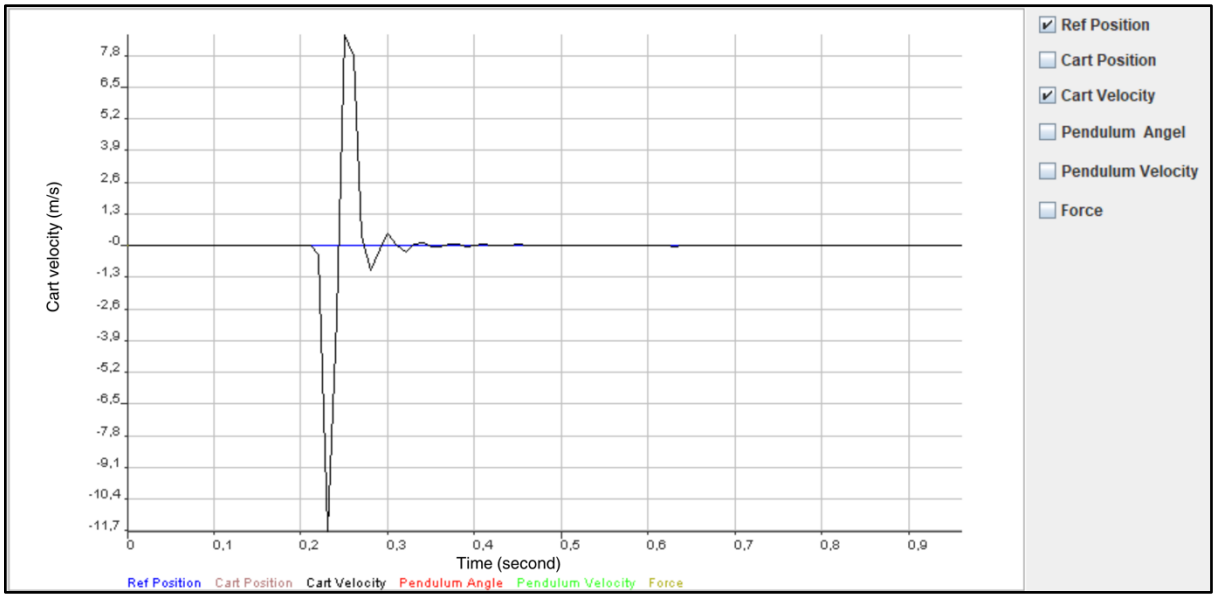


Figure 100: Cart velocity graph

The graph presented above illustrates the cart's velocity as a result of a disturbance. Initially, the cart velocity is zero due to the absence of external disturbances. However, subsequently, the velocity spikes as a result of an external force being applied to the pendulum. This surge in velocity is evident in the graph from 0.2 to 0.3 seconds. Subsequently, there is a significant reduction in velocity once the control system engages, which reflects its stabilizing effect on the pendulum. The system reaches a stable state in under 0.5 seconds.

17.1.5 Force

Analyze the force exerted by the control system after it becomes active and certain disturbances are introduced to the pendulum.

Force graph:

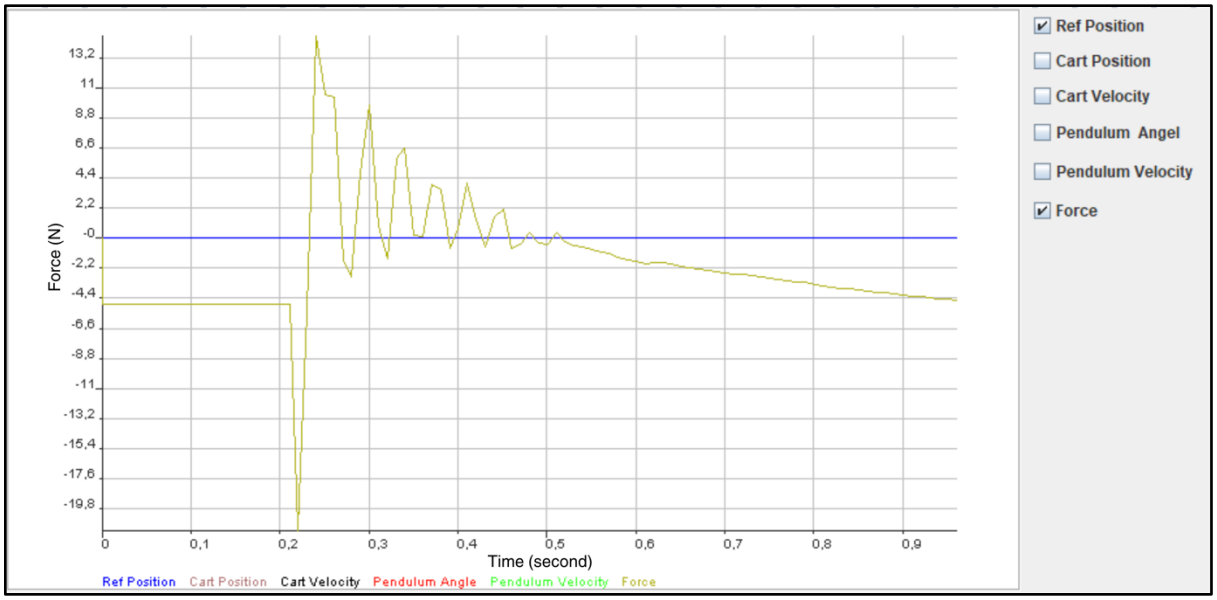


Figure 101: Force graph

The graph depicted above shows the force applied in response to a disturbance of the pendulum. Upon introducing this disturbance, a force was generated to move the cart and mitigate the pendulum's oscillations. Following an initial overshoot, the force stabilized within 0.55 seconds.

17.2 Recorded data file

The file mentioned below is available for download and contains the recorded data.

CartRefPos (m)	CartPos(m)	CartVel(m/s)	PendAng(deg)	PendAngVel(deg/s)	Force(N)
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9

0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	0	0	-13,011	0	-4,9
0,0001	-0,004	-0,3955	-19,6923	-668,1319	-21,5
0,0001	-0,1216	-11,7609	-43,956	-2426,3736	-5,4
0,0001	-0,151	-2,9485	-22,4176	2153,8462	14,8
0,0001	-0,0649	8,6146	9,5824	3200	10,5
0,0001	0,0127	7,7607	7,3846	-219,7802	10,3

0,0001	0,0162	0,3482	-12,6593	-2004,3956	-1,6
0,0001	0,0057	-1,0517	-18,1099	-545,0549	-2,9
0,0001	0,004	-0,1715	-13,5385	457,1429	4,8
0,0001	0,0087	0,476	-9,2308	430,7692	9,7
0,0001	0,0087	-0,0035	-12,8352	-360,4396	0,9
0,0001	0,006	-0,2712	-15,8242	-298,9011	-1,5
0,0001	0,0062	0,021	-12,7473	307,692	5,8
0,0001	0,0073	0,1085	-10,5495	219,7802	6,6
0,0001	0,0065	-0,0717	-13,5385	-298,9011	0,2
0,0001	0,0057	-0,084	-14,8571	-131,8681	0,1
0,0001	0,0061	0,042	-12,1319	272,5275	3,9
0,0001	0,0064	0,0262	-11,5165	61,5385	3,5
0,0001	0,0058	-0,0577	-13,978	-246,1538	-0,8
0,0001	0,0056	-0,021	-13,978	0	0,7
0,0001	0,0059	0,0332	-11,8681	210,989	4
0,0001	0,0059	-0,0017	-12,3077	-43,956	1,3
0,0001	0,0055	-0,0367	-13,978	-167,033	-0,7
0,0001	0,0055	0	-13,1868	79,1209	1,5
0,0001	0,0057	0,014	-12,1319	105,4945	2,1
0,0001	0,0056	-0,0105	-13,0989	-96,7033	-0,8
0,0001	0,0055	-0,0122	-13,5385	-43,956	-0,5
0,0001	0,0055	0,0035	-12,6593	87,9121	0,4
0,0001	0,0055	0	-12,6593	0	-0,4
0,0001	0,0054	-0,0087	-13,1868	-52,7473	-0,5

0,0001	0,0054	-0,0017	-13,011	17,5824	0,4
0,0001	0,0054	-0,0017	-12,8352	17,5824	-0,3
0,0001	0,0054	-0,0017	-12,9231	-8,7912	-0,6
0,0001	0,0054	0	-12,9231	0	-0,7
0,0001	0,0053	-0,0017	-12,9231	0	-0,9
0,0001	0,0053	-0,0035	-12,9231	0	-1,1
0,0001	0,0053	-0,0035	-12,9231	0	-1,2
0,0001	0,0052	-0,0035	-12,9231	0	-1,4
0,0001	0,0052	-0,0035	-12,9231	0	-1,6
0,0001	0,0051	-0,007	-12,9231	0	-1,7
0,0001	0,005	-0,0122	-12,9231	0	-1,9
0,0001	0,0046	-0,0385	-12,9231	0	-1,8
0,0001	0,004	-0,0577	-12,9231	0	-1,8
0,0001	0,0036	-0,0402	-12,9231	0	-1,9
0,0001	0,0034	-0,0245	-12,9231	0	-2,1
0,0001	0,0033	-0,0105	-12,9231	0	-2,2
0,0001	0,0032	-0,0052	-12,9231	0	-2,3
0,0001	0,0032	-0,0052	-12,9231	0	-2,4
0,0001	0,0031	-0,0052	-12,9231	0	-2,5
0,0001	0,0031	-0,007	-12,9231	0	-2,6
0,0001	0,003	-0,0052	-12,9231	0	-2,7
0,0001	0,003	-0,0035	-12,9231	0	-2,7
0,0001	0,0029	-0,0035	-12,9231	0	-2,8
0,0001	0,0029	-0,0052	-12,9231	0	-2,9

0,0001	0,0029	-0,0035	-12,9231	0	-3
0,0001	0,0028	-0,0035	-12,9231	0	-3,1
0,0001	0,0028	-0,0035	-12,9231	0	-3,2
0,0001	0,0028	-0,0017	-12,9231	0	-3,3
0,0001	0,0027	-0,0017	-12,9231	0	-3,3
0,0001	0,0027	-0,0035	-12,9231	0	-3,4
0,0001	0,0027	-0,0017	-12,9231	0	-3,5
0,0001	0,0027	-0,0035	-12,9231	0	-3,6
0,0001	0,0026	-0,0035	-12,9231	0	-3,7
0,0001	0,0026	-0,0035	-12,9231	0	-3,7
0,0001	0,0026	-0,0017	-12,9231	0	-3,8
0,0001	0,0025	-0,0035	-12,9231	0	-3,9
0,0001	0,0025	-0,0017	-12,9231	0	-4
0,0001	0,0025	-0,0035	-12,9231	0	-4
0,0001	0,0025	-0,0017	-12,9231	0	-4,1
0,0001	0,0024	-0,0035	-12,9231	0	-4,2
0,0001	0,0024	-0,0017	-12,9231	0	-4,3
0,0001	0,0024	-0,0035	-12,9231	0	-4,3
0,0001	0,0023	-0,0052	-12,9231	0	-4,4
0,0001	0,0023	-0,0052	-12,9231	0	-4,5
0,0001	0,0022	-0,0105	-12,9231	0	-4,5
0,0001	0,002	-0,0175	-12,9231	0	-4,6

Conclusion

The Cart Pendulum Control project successfully integrates various control system components and contemporary theories to manage the position and control of the system. Through meticulous engineering techniques, including numerical representation and the development of differential equations, the dynamic behavior of the system was analyzed. Utilizing state space control methods, the nonlinear model was linearized to facilitate the construction of controllers. Both simulation and real-time experimentation validated the effectiveness of the designed controllers in regulating the cart pendulum system's position and movement. Furthermore, the project transitioned seamlessly from simulation to practical implementation, employing hardware-software co-design approaches. The controllers, initially tested and refined through simulation, were translated into C code and implemented on the ZYNQ SoC board. A graphical user interface facilitated monitoring and system status checks during physical testing. The successful convergence of theoretical modeling, simulation validation, and practical implementation underscores the project's robustness and applicability in real-world scenarios.

References

[1] Embedded Systems Project (Revision: V1.2a, Release: October 2020, Prof. Dr.-Ing. Kai Mueller)

[2] Incremental Encoder : (https://www.eltra.it/assets/Uploads/EncoderPedia-immagini-e-file/PDF/en-INCREMENTAL-ENCODERS-140407.pdf)

[3] LEZ3 Linear Actuator (https://www.isel-us.com/LEZ-3-Belt-Actuator)

[4] DC Servo motor (https://www.yumpu.com/en/document/read/23535078/970471-dm005-49-2001-mv330-langewellepdf)

[5] End Switch (https://store.thingibox.com/en/electrical/548-endstop_switch_with_lever.html)

[6] Arty Z7 board (https://digilent.com/reference/programmable-logic/arty-z7/start)

[7] DAC Data sheet (https://www.analog.com/media/en/technical-documentation/data-sheets/ad5624r_5644r_5664r.pdf)

[8] ADC Data sheet (https://www.analog.com/media/en/technical-documentation/data-sheets/ad7886.pdf)

[9] SPI Protocol (https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/)

[10] https://elli.hs-bremerhaven.de/goto.php?target=file_284022_download&client_id=elli

[11] https://elli.hs-bremerhaven.de/goto.php?target=file_284023_download&client_id=elli

[12] https://elli.hs-bremerhaven.de/goto.php?target=file_284021_download&client_id=elli

[13] https://elli.hs-bremerhaven.de/goto.php?target=file_284026_download&client_id=elli

[14] https://elli.hs-bremerhaven.de/goto.php?target=file_284014_download&client_id=elli

[15] https://www.w3schools.com/c/c_getstarted.php

[16] http://www.c-jump.com/CIS77/CPU/Numbers/U77_0160_sign_extension.htm

[17] https://www.youtube.com/watch?v=7HChmKwOUzw

[18] https://www.w3schools.com/java/

[19]https://stackoverflow.com/questions/825678/what-is-the-best-way-to-generate-a-unique-and-short-file-name-in-java

[20]https://mkyong.com/java/java-how-to-split-a-string/#:~:text=7.-,Split%20a%20string%20by%20a%20space%20or%20multiple%20spaces,split%20a%20string%20by%20spaces.

[21] https://stackoverflow.com/questions/14386397/split-a-string-at-a-single-space

[22] https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html

[23] https://www.baeldung.com/java-write-to-file

[24] https://elli.hs-bremerhaven.de/goto.php?target=file_284017_download&client_id=elli

[25] https://www.linkedin.com/pulse/systems-engineering-tidbits-v-model-erik-puik