HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF INTERNATIONAL EDUCATION

FINAL REPORT

Application of PID controller in balancing an inverted pendulum

Major: MECHATRONICS ENGINEER

Subject: PROJECT OF MECHATRONICS

Lecturer: CÁI VIỆT ANH DŨNG, PhD.

Members: BÙI MINH THẮNG 21146414

Ho Chi Minh City, December 2024

Table of Contents

List of Tables

List of Figures

Chapter 1. INTRODUCTION

- ❖ Nowadays, there are many control algorithms used such as Fuzzy Logic Control, Neural Network Control, Adaptive Control, and Optimal Control. Optimal Control), ... But PID (Proportional Integral-Derivative) control is still an automatic control method is widely used in many industrial applications, especially in servo motor control because of its simplicity, ease of application and also the basis for development many other algorithms. To achieve this precise control, a PID controller is applied to adjust the motor output to respond properly to the target input signal.
- ❖ Main objective of the project:
  - Use stm32 microprocessor to control DC motor
  - Use a PID controller to keep the pendulum upside down
- ❖ Expected product:
  - Inverted pendulum model with 12V DC drive motor
- ❖ Input data:
  - DC motor:
  - Voltage: 12V
  - No-load current :120 mA
  - No-load speed: 1590 RPM
  - Pendulum travel: 700 mm
  - Mass of the load: 0.7 kg

Chapter 2. THEORETICAL BASIS

2.1. What is PID control?

- ❖ PID (Proportional Integral Derivative) is a feedback mechanism for control loops, they are applied widely used in modern industrial control systems.
- ❖ This controller is widely used in closed-loop control systems with feedback signals. The PID's task is to help calculate the error value which is the difference between the measured value and the desired set value.

2.2. Operating principle of PID controller

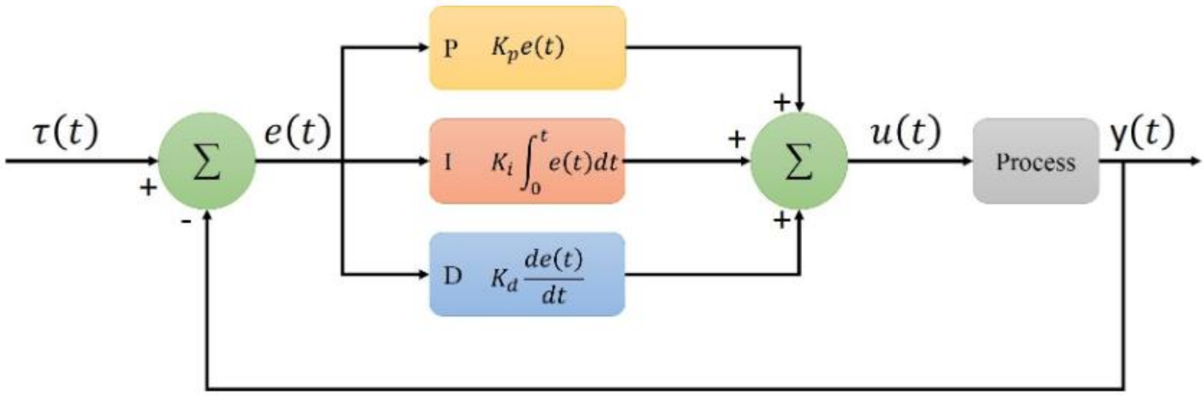

Figure 2.1. Principle diagram of PID controller

- ❖ The PID controller has 3 components, the sum of which forms the MV control variable

$$MV(t) = P_{out} + I_{out} + D_{out}$$

- ❖ The proportional, integral, and differential components are added together to calculate the output of the PID controller. Define that u(t) is the output of the controller, the final expression of the PID algorithm is:

$$u(t) = MV(t) = K_p * e(t) + K_i \int_0^t e(\tau)d\tau + K_d * e(t)\frac{d}{dt}$$

2.2.1. P (Proportional)

- ❖ The proportional component changes the output value, proportional to the current error value. The proportional response can be adjusted by multiplying that error by a constant Kp, called the proportional coefficient.

$$P_{out} = K_p * e(t)$$

Which

P_{out}: *output of the proportional component*

K_p: proportional coefficient

e: *error* $=$ *Setpoint* $-$ *Current value*

t: *instantaneous time*

- ❖ If the Pout coefficient is too high, the system will be unstable. On the contrary, a small coefficient is due to the small output response while the input error is large, causing the controller to be less sensitive or respond slowly. If the coefficient Kp is too low, the control action may react too little to system disturbances.
- ❖ The larger the Kp value, the faster the response, so the larger the error, the larger the proportional component compensation. A proportional gain value that is too large will lead to process instability and vibration dynamic.

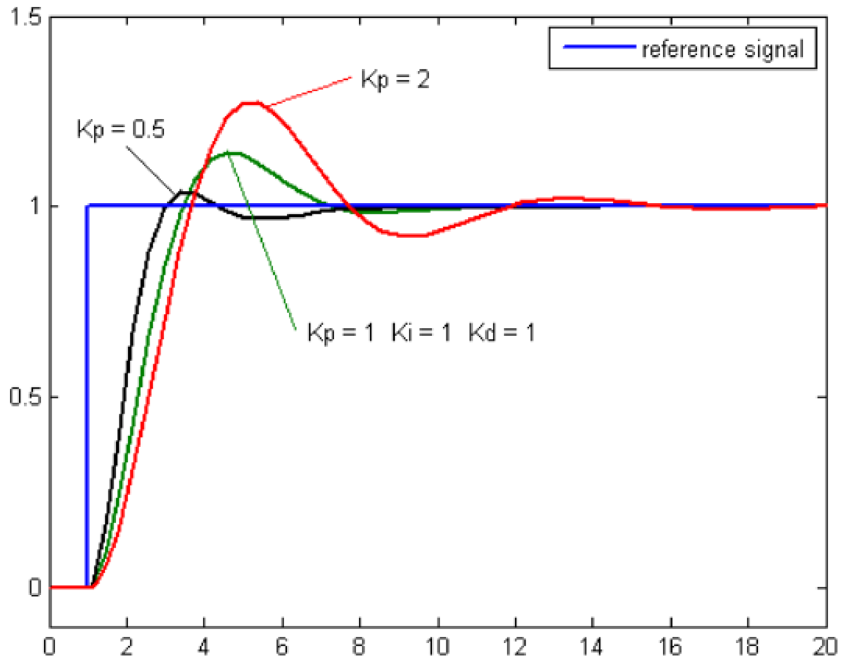

Figure 2.2. CV over time corresponds to 3 constant values of Kp and Ki, Kd

2.2.2. I (Integral)

- ❖ The distribution of the integration component is proportional to both the error amplitude and the time during which the error occurs. The sum of instantaneous errors over time (error integration) gives us the previously corrected cumulative compensation. The accumulated error is then multiplied by the integral gain and added to the controller output signal. The distribution amplitude of the integration component over all tuning actions is determined by the integration gain, Ki.

$$I_{out} = K_i \int_0^t e(\tau)d\tau$$

Which

$$I_{out}: output\ of\ the\ integral\ component$$

$$K_i: integral\ \text{coefficient}$$

$$\int e : error\ integral = error + Previous\ error$$

- ❖ The integral component will speed up the process movement to the set point and eliminate the stability error balance at a rate that depends only on the controller. However, because the integral component is the response of the previous accumulated error, it can cause the current value to overshoot.
- ❖ The larger the Ki value, the faster the stability error is eliminated. The disadvantage is the larger the overshoot: any negative error integrated during the transient response must be integrated by a positive error before reaching steady state.

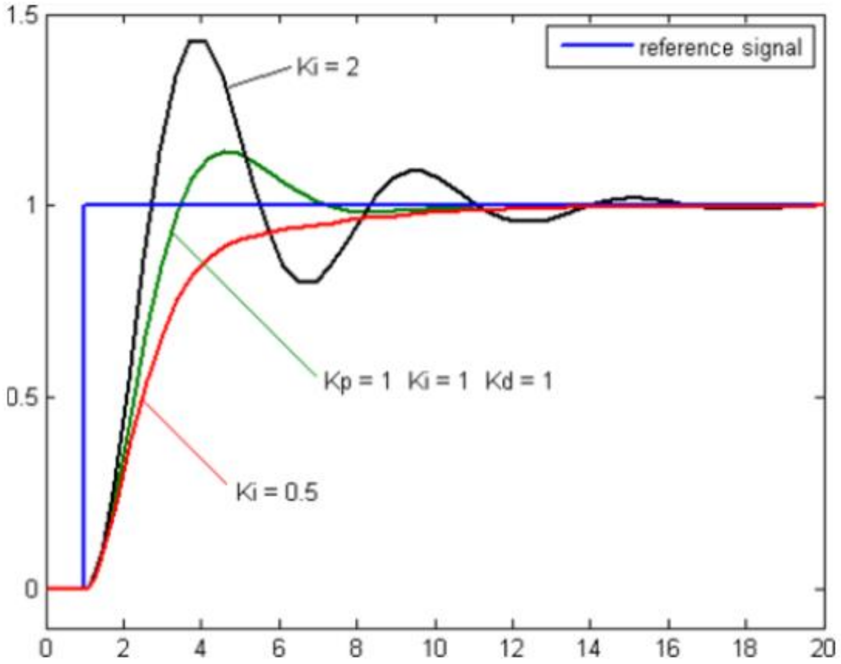

Figure 2.3. CV over time corresponds to 3 constant values of Kp and Ki, Kd

2.2.3. D (Derivative)

- The rate of change of the process error is calculated by determining the slope of the error with respect to time (first derivative with respect to time) and multiplying this rate by the proportional gain Kd. The amplitude of the differential component distribution over all control acts is limited by the differential gain, Kd.

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

Which

$$D_{out}: output\ of\ the\ differential\ component$$

$$K_d: differential\ \text{coefficient}$$

$$e\frac{d}{dt}: differential\ error = error - Previous\ error$$

- The differential component slows down the rate of change of the controller output, and this characteristic is most noticeable when the controller setpoint is reached. From there, differential control is used to reduce the overshoot amplitude produced by the integral component and enhance the stability of the composite controller. However, it will amplify the noise and thus the component is more sensitive to noise when errors occur and can cause the process to become unstable if the noise and differential gain are large enough.
- Larger Kd values reduce overshoot, but slow down the transient response and can lead to instability due to signal noise amplification in error differentiation.

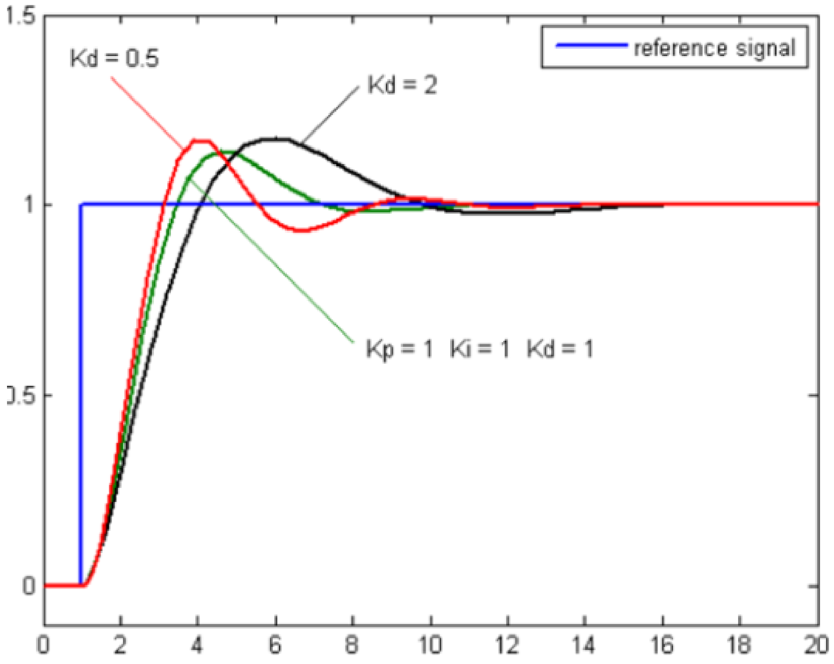

Figure 2.4. CV over time corresponds to 3 constant values of Kp and Ki, Kd

2.2.4. Anti-windup

❖ Anti-windup is a method or mechanism integrated into a PID controller to limit or prevent windup. The purpose of anti-windup is to stop the accumulation of integral errors when the controller output is limited, making the system more stable and more responsive.

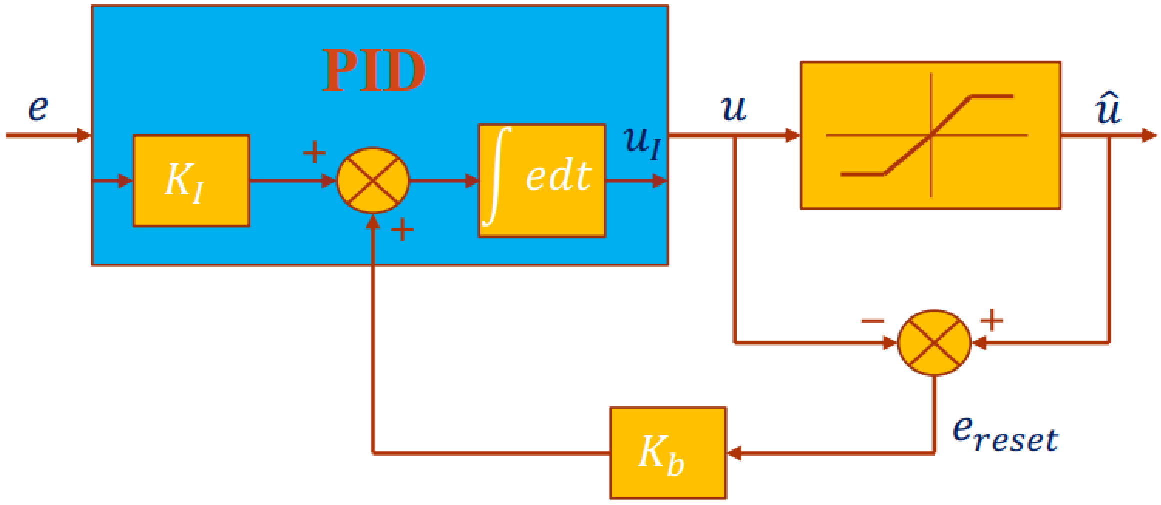

Figure 2.5. The anti-windup scheme for I-term

❖ The equation of I-term with anti-windup structure can be described as follows:

$$I_{out} = K_i \int_0^t e(\tau)d\tau + K_b \int_0^t e_{Reset}(\tau)d\tau$$

Which $e_{Reset} = \hat{u} - u$

- When the controller is at its limit, the anti-windup will adjust the value of the integral component (I) so that it does not continue to increase or decrease beyond the allowable limit.

2.2.5. Low pass filter

- Low-pass filters are often used to reduce noise and smooth the input signal or error signal before being processed by the components in the PID controller, especially the derivative component (D). This is a popular technique to improve the performance and stability of control systems.

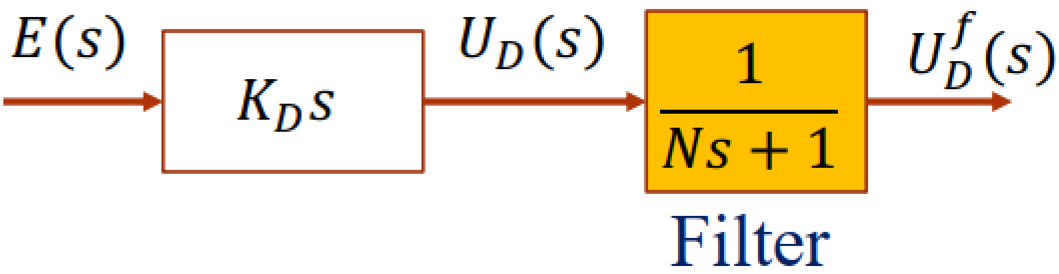

Figure 2.6. D-term with the low pass filter

$$u_f^D(k) = \frac{N}{N+T}.u_f^D(k-1) + \frac{T}{N+T}.u^D(k)$$

$$where\ \alpha = \frac{T}{N+T}\ (0 < \alpha \leq 1): coefficient\ of\ low\ pass\ filter$$

- The equation of D-term with low pass filter structure can be described as follows:

$$u_f^D(k) = (1-\alpha).u_f^D(k-1) + \alpha.u^D(k)$$

- The derivative component (D) in PID is very sensitive to rapid changes in error. If the error signal is noisy (especially short-term oscillations), the derivative component can produce excessive response, causing biases, oscillations, or unstable response. The low pass filter helps reduce these rapid changes, making the derivative component respond more smoothly and reducing the risk of unnecessary oscillation.

2.3. Methods to find parameters Kp, Ki, Kd

Table 2.1. Methods to find parameters Kp, Ki, Kd

Method	**Advantage**	**Drawback**
Manual adjustment	No math knowledge required.	Experience required. Online method
Ziegler – Nichols	Easy to experiment	Only effective for linear or near-linear systems, cannot effectively deal with obvious non-linearities. Online method
Software tools	PID parameters can be tested without having to change the actual system, helping to minimize risk. Online or offline method	Accurate simulation is required and cannot completely replace actual testing.
Cohen - Coon	Control the models well	Requires mathematical knowledge. Only effective for first-order processes. Offline method

2.3.1. Manual adjustment

- ❖ If the system must remain online, one method of adjustment is to set the initial value of Ki, Kd to zero. Gradually increase Kp until the output of the control loop oscillates, then Kp can be set to approximately half that value to drive "1/4 of the amplitude attenuation value" response is achieved. Then increase Ki to an appropriate value to allow enough processing time. However, Ki that is too large will cause instability. Finally, increase Kd, if necessary, until an acceptably fast control loop quickly regains its set value after disturbance. However, too large a Kd will cause noise and inaccuracy. A rapid adjustment of the PID control loop is often slightly overshoot as it approaches the set point rapidly; However, some systems do not tolerate overshoot, in which case we need a closed-loop overshoot system, setting a Kp value less than half of the Kp value that causes the oscillation.

Table 2.2. Impact of increasing an independent parameter

Parameters	Rise time	Overshoot	Settling time	Settling error
K_p	Decrease	Increase	Small change	Decrease
K_i	Decrease	Increase	Increase	Eliminated
K_d	Small change	Decrease	Decrease	Small change

2.3.2. Ziegler–Nichols's method

- Similar to the above method, Ki and Kd are initially set to zero. The gain P is increased until it approaches a critical gain, Kgh, where the output of the control loop begins to oscillate (closed system at the stable boundary). Kgh and oscillation time Tgh are used to assign the gain as follows:

Table 2.3. Ziegler–Nichols's method

Controller	K_p	K_i	K_d
P	0.5*Kgh	0	0
PI	0.45*Kgh	1.2*Kgh/Tgh	0
PID	0.6*Kgh	2*Kgh/Tgh	0.125*Kgh*Tgh

2.3.3. PID tuning software

- Most modern industrial applications no longer adjust control loops using manual calculation methods like the above. Instead, PID tuning and loop optimization software such as MATLAB Simulink are used to ensure robust results. These software packages will collect data, develop processing models, and recommend optimal adjustment methods. Some software packages can even develop tuning by collecting data from reference changes.
- PID tuning mathematically generates a pulse in the system, and then uses the control system's frequency response to design the control loop values PID. In loops with response times lasting many minutes, mathematical tuning should be chosen, because trial and error can actually take days to find a stable point for the loop. The optimal value is more difficult to find. Some digital controllers also have a self-

tuning function, in which very small changes in the set point are also sent process, allowing the controller to calculate the optimal adjustment value on its own.

- ❖ Other types of adjustments are also used depending on different outcome assessment criteria. Many of today's inventions are already embedded in part modules software and hardware for PID tuning.

2.4. What is an inverted pendulum?

Figure 2.7. Types of inverted pendulums

- ❖ The inverted pendulum is a complex model with high nonlinearity in the field of automation control. This model will help operators verify many theoretical bases and different algorithms in automatic control.
- ❖ The inverted pendulum system being researched today includes several types as follows: linear inverted pendulum, rotating inverted pendulum, inverted pendulum with slider system, inverted pendulum with cart system, 2, 3 degree of freedom inverted pendulum,...

Chapter 3. MODEL DESCRIPTION

3.1. Mechanical model

- ❖ Consists of a pendulum bar (mica) rotating around a vertical axis. The pendulum bar is indirectly attached to a vehicle through an encoder for measurement inclined angle. The drive motor has another built-in encoder to determine where the vehicle is moving.

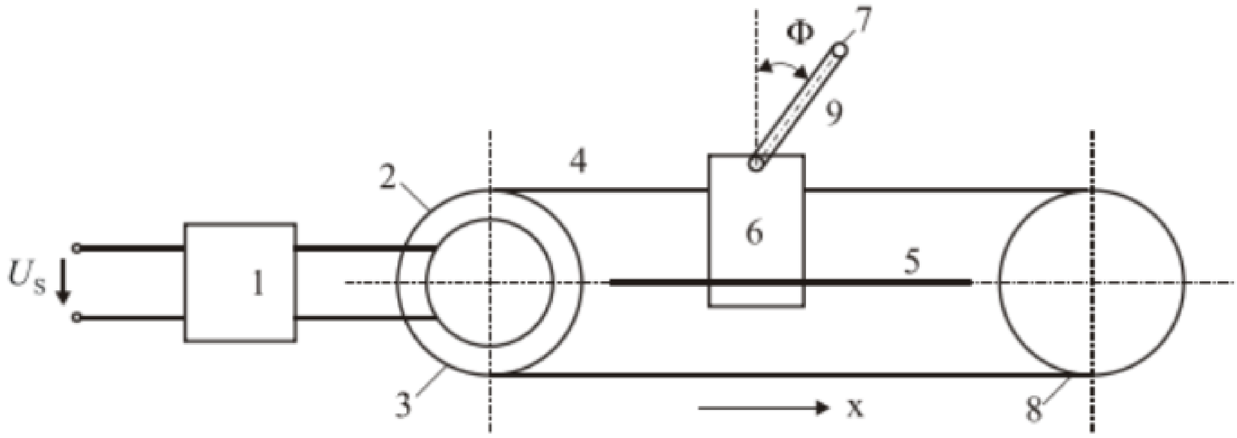

Figure 3.1. Kinematic structure of the inverted pendulum model

Which

1-Power supply block for the motor

2- DC motor

3- Drive pulley

4.8 -Drive belt

5-The bar guides the movement of the cart

6-Wagon

7-The pendulum

9- The lever arm of the pendulum

3.2. Electrical circuit

3.2.1. Introducing H-bridge circuits

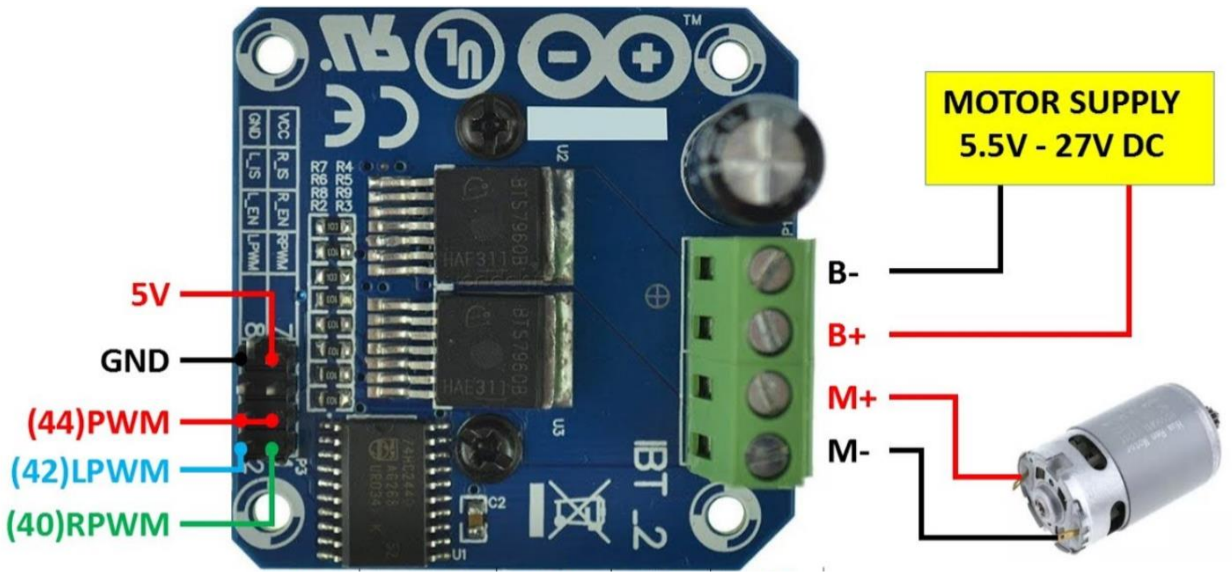

Figure 3.2. BTS7960 DC motor control circuit

Specifications:

Power supply: 6 ~ 27VDC

Control logic signal: 3.3 ~ 5VDC

Maximum control frequency: 25KHz

Pin diagram:

VCC: Source to create control logic level (3.3~5VDC)

GND: Ground

R_IS and L_IS: combined with resistors to limit the current through the H-bridge

R_EN and L_EN: Motor speed control pin

RPWM and LPWM: motor direction control pin

RPWM = 1 and LPWM = 0: Motor rotates forward.

RPWM = 0 and LPWM = 1: Motor rotates in reverse

RPWM = 1 and LPWM = 1 or RPWM = 0 and LPWM = 0 : Stop.

Normally, RPWM & LPWM are connected to the GPIO pins to control the direction of motor rotation.

The R_EN and L_EN pins are connected together and then connected to the PWM pin to control the motor speed.

3.2.2. Introducing rotary encoder

Figure 3.3. OMRON Rotary Encoder E6B2-CWZ6C

Specifications:

Resolution: 1000 pulse/round

Output type: NPN open collector

Frequency response: 100KHz (maximum)

Output phases: A (black wire), B (white wire), Z (orange wire)

Supply voltage: 5~24 VDC

Pin diagram:

Brown: 5-24V

GND: ground

Black: signal feedback channel A

White: signal feedback channel B

Orange: signal feedback channel Z

A rotary encoder is a type of sensor that converts the angular position or motion of a shaft into an analog or digital signal. It is commonly used in various applications for precise control and measurement of rotational movement.

Channels A and B produce square wave signals that are 90 degrees out of phase (quadrature). This phase difference allows the system to determine both the direction and the amount of rotation. For example, if Channel A leads Channel B, the encoder is rotating in one direction (e.g., clockwise). If Channel B leads Channel A, it indicates rotation in the opposite direction (e.g., counterclockwise).

Each channel generates pulses as the shaft rotates. The number of pulses per revolution (PPR) indicates the encoder's resolution. Higher PPR means finer resolution and more accurate position feedback.

In this project, we will use encoder mode integrate in stm32 timer peripheral to read encoder at x4 mode.

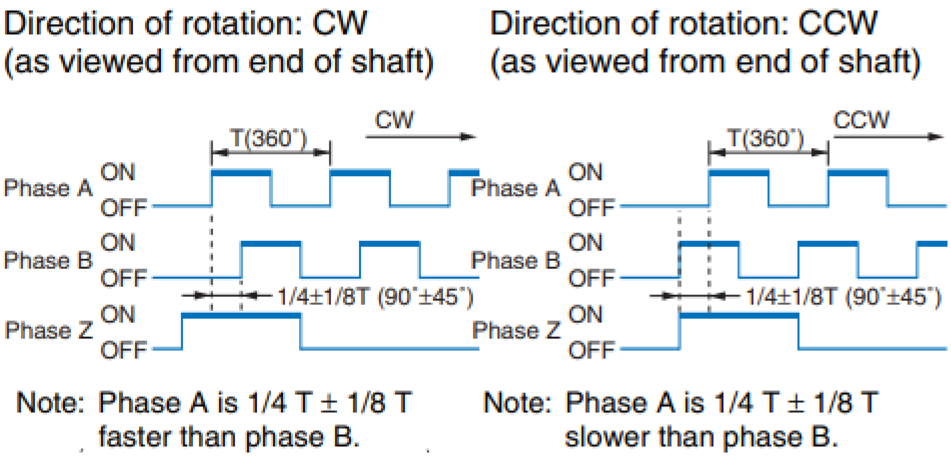

Figure 3.4. Output of Rotary Encoder E6B2-CWZ6C

3.2.3. Introducing DC servo motor with encoder

Figure 3.5. DC Servo Gear Reducer Motor

Motor specifications:

Rated voltage: 12VDC

Transmission ratio: 1:56

Speed after reduction gearbox (main shaft): 1590 RPM

Shaft diameter: 6mm

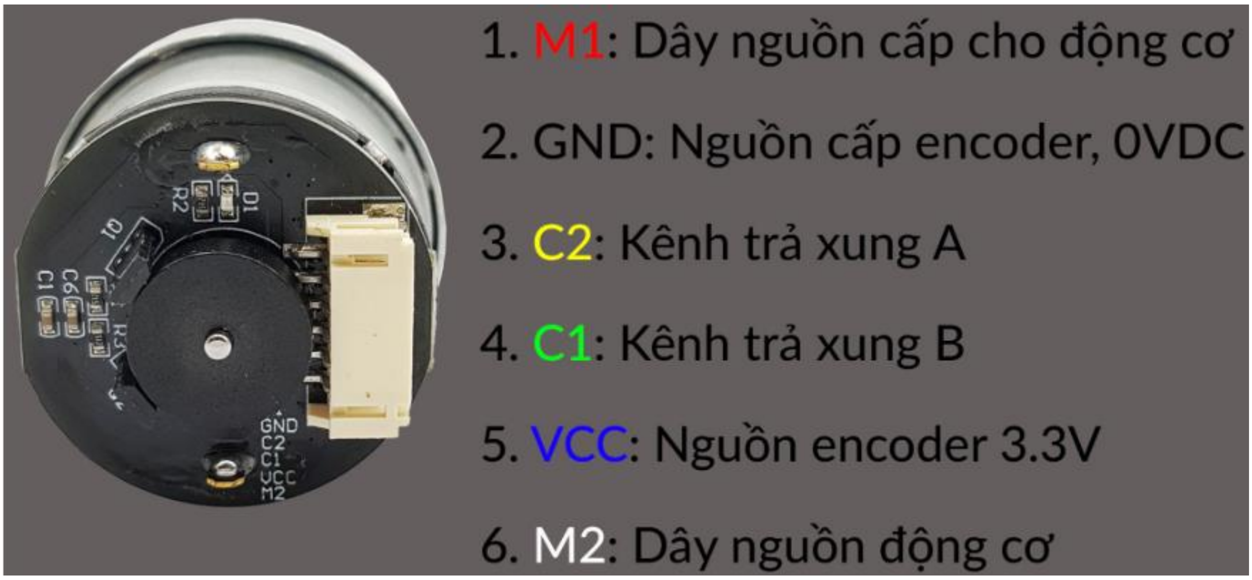

Figure 3.6. Pinout of gear reduction motor with encoder

Encoder specifications:

Supply voltage: 3.3VDC

Encoder: 2 AB pulse channels, with each channel returning 11 pulses/round

Number of Encoder pulses after deceleration: 11×56 = 616 pulses/ revolution

3.3. Wiring and connection

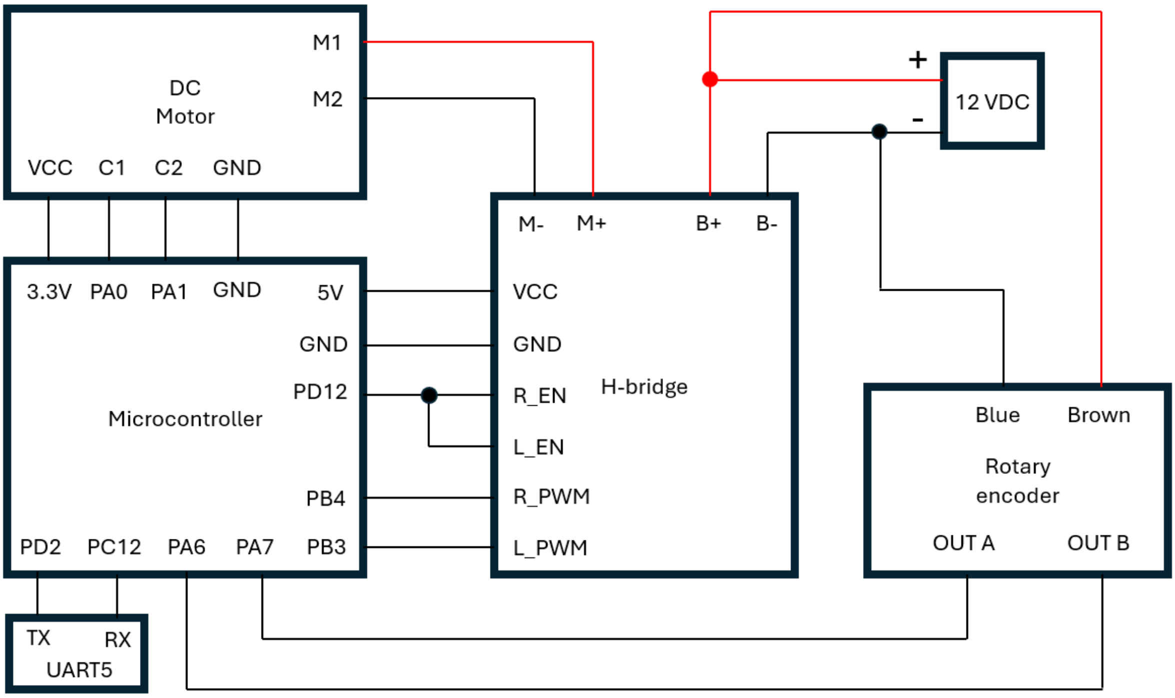

Figure 3.7. Wiring and connection diagram

Chapter 4. APPLYING PID CONTROLLER TO AN INVERTED PENDULUM

4.1. Building a block diagram

- The main goal is to control the position of the pendulum, which will return to the vertical after the initial disturbance, so the reference signal r will be 0. The external force F acting on the vehicle can be considered as gravity, friction or many external factors impact. Meanwhile, the output of the controller is the traction force from the motor. The feedback signal to the controller is the tilt angle Φ of the pendulum at a time. The difference between the reference signal and the feedback signal is called the error, which will update and adjust the controller parameters accordingly so that the error approaches 0.
- The block diagram of an inverted pendulum can be constructed as follows:

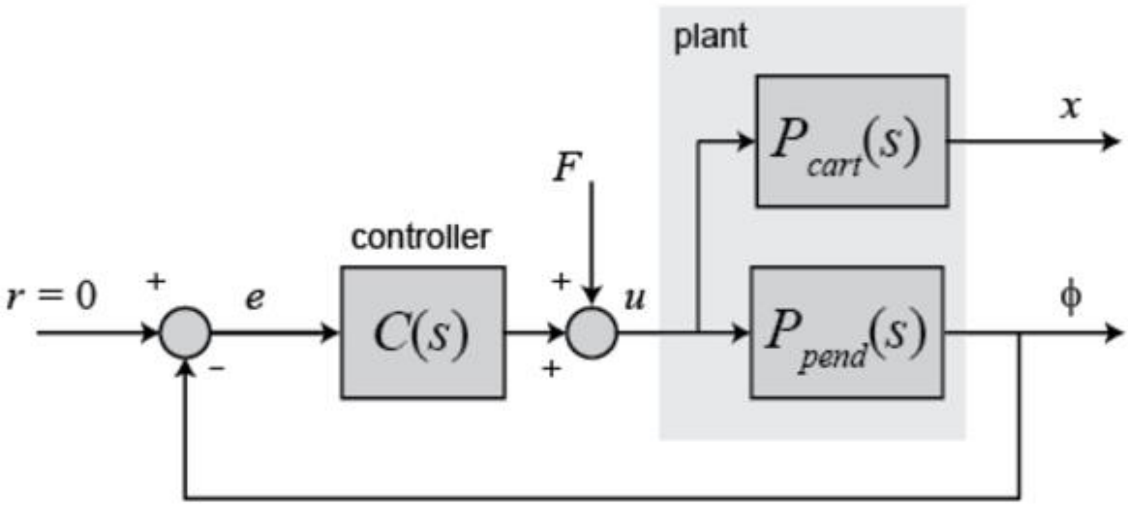

Figure 4.1. System structure of inverted pendulum controller (1 feedback loop)

- The system only has one feedback loop from the encoder reading the tilt angle. Since the vehicle position has no feedback loop, this variable will not affect the PID controller.

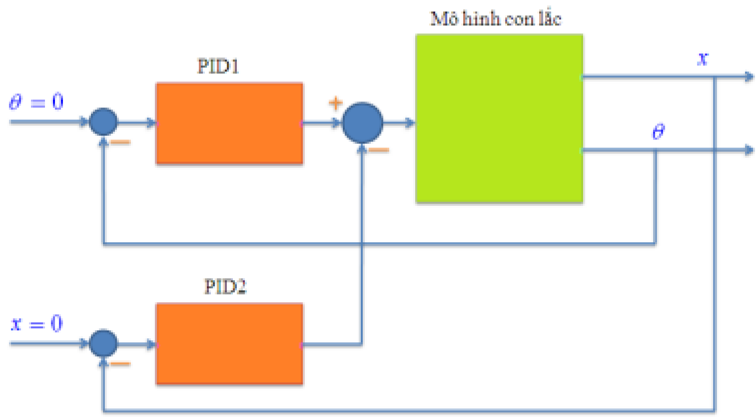

Figure 4.2. System structure of inverted pendulum controller (2 feedback loop)

- ❖ The system has 2 feedback loops from 2 different encoders. Two separate pid controllers will run simultaneously and respond to 2 independent signals. Controller 1 will adjust so that the angle of the pendulum is always vertical. Meanwhile, controller 2 will adjust to slowly bring the cart to the desired position.
- ❖ Controller 1 needs to respond quickly to keep the pendulum balanced, so the output of PID1 must be large. The output of controller 2 should be small, having little effect on the total output of the 2 controllers. The goal is to slowly move the cart to the desired position, while always keeping the pendulum vertically upward.

4.2. Algorithm of PID controller

4.2.1. P term

- ❖ In the experiment, gradually increase Kp until the sled starts moving when the error is at the minimum allowable level (0.72 degrees). Save the value Kp_1L. Continue increasing from Kp_1L, then slowly tilt the pendulum and observe the reaction of the sled: If the system has a large overshoot or gradually becomes unstable after exiting the steady state. Save the Kp_1H value.
- ❖ Kp adjustment is in the range [Kp_1L; Kp_1H] so that the system reacts quickly enough but remains stable, with little overshoot
- ❖ After checking and concluding that the parameters of PID1 controller are suitable, continue to find the parameters for PID2 controller. In the 2 closed loop control system, there is another PID controller for the cart position. The sum of the 3 terms of PID2 is small and has a slight impact on PID1.

- Any small Kp2 value can be chosen arbitrarily, but the output of P2-term and PID2 should not be allowed to be too large.

4.2.2. I term

- After increasing Kp from PID1, continue to increase Ki until the overshoot is large enough and causes system instability. Save the Ki_1H value. Slowly tilt the pendulum to observe the sled's reaction.
- Adjust Ki in the range [0; Ki_1H] so that the system responds faster but remains stable.
- After obtaining the appropriate Ki, use Anti-windup to reduce overshoot, by narrowing the allowable limit of the I component (including upper and lower limit of term I).

4.3. PID parameters table

Table 4.1. PID parameters

Parameter	**Value**
K_{p1}	1800
K_{i1}	0.01
K_{d1}	0
K_{p2}	2
K_{i2}	0
K_{d2}	0

Chapter 5. PROGRAM CODE IN STM32

5.1. STM32 CubeMX Configuration

5.1.1. System clock configuration

❖ Select HSE as the system clock source. The frequency multiplier PLLCLK is used for configuration as follows:

Table 5.1. System clock configuration

Clock source	Frequency (MHz)
HCLK	84
APB1	84
APB2	84

5.1.2. GPIO pins configuration

Table 5.2. GPIO pins configuration

Pin	Mode	Maximum output speed
PB3	Output push-pull	High
PB4	Output push-pull	High

5.1.3. Timers configuration

❖ Configure timer 1 and 2 as follows:

Table 5.3. Timers configuration

Parameter settings	Mode (value)
Clock source	Internal Clock
Prescaler (PSC)	84
Auto Reload Register (ARR)	1000
NVIC settings	**Mode**
TIM1 break interrupt	Enabled
TIM1 update interrupt	Enabled

TIM2 global interrupt	Enabled

5.1.4. PWM signal source configuration

❖ Configure PD12 as the PWM pulse source at timer 4 as follows:

Table 5.4. PWM configuration

Parameter settings	Mode (value)
Channel 1	PWM Generation CH1
Prescaler (PSC)	8400
Auto Reload Register (ARR)	100

5.1.5. Encoder configuration

❖ Configure PA6, PA7 as the Encoder Mode at timer 3 as follows:

Table 5.5. Encoder mode configuration for pendulum

Parameter settings	Mode (value)
Prescaler (PSC)	0
Auto Reload Register (ARR)	65535
Encoder	Encoder Mode TI1 and TI2
NVIC settings	**Mode**
TIM5 global interrupt	Enabled

❖ Configure PA0, PA1 as the Encoder Mode at timer 5 as follows:

Table 5.6. Encoder mode configuration for DC motor

Parameter settings	Mode (value)
Prescaler (PSC)	0
Auto Reload Register (ARR)	4294967295
Encoder	Encoder Mode TI1 and TI2
NVIC settings	**Mode**

TIM5 global interrupt	Enabled

5.1.6. UART configuration

❖ Configure UART 5 as follows:

Table 5.6. UART configuration

Parameter settings	**Mode (value)**
Baud rate (bits/s)	9600
Parity	None
Stop bits	1

5.2. Program code

```
#include "main.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "stdbool.h"

/* Private includes -------------------
/* USER CODE BEGIN Includes */
#define GPIO_NN_Thuan GPIOB
#define GPIO_PIN_NN_Thuan GPIO_PIN_13
#define GPIO_NN_Nghich GPIOB
#define GPIO_PIN_NN_Nghich GPIO_PIN_14
#define GPIO_NN_Dung GPIOB
#define GPIO_PIN_NN_Dung GPIO_PIN_15
#define GPIO_IN1 GPIOB
#define GPIO_PIN_IN1 GPIO_PIN_3
#define GPIO_IN2 GPIOB
#define GPIO_PIN_IN2 GPIO_PIN_4
#define GPIO_ENA GPIOB
#define GPIO_PIN_ENA GPIO_PIN_5
#define pi 3.1416
#define r 3
#define d_pulley 14
int TrangThai = 2, start=0;
```

Figure 5.1. Pins definition and library declaration

❖ Call libraries and define pins, constants

```
volatile float giatrido_hientai, error  , error_truoc  ,pre_pre_Error, sai_so, pre_saiso, duc=0, deg=0;
volatile float giatrido_hientai1, giatrido_hientai2, error1, error2, error_truoc1, error_truoc2;

volatile static float integral = 0, derivative  , p_part=0, i_part, d_part, i_truoc, uDf, uDf_truoc;
volatile static float p_part1 = 0, p_part2 = 0, i_part1, i_part2, d_part1, d_part2;
volatile float setpoint =0, Kp = 0, Ki = 0, Kd = 0;// 2000 0 0
volatile float Delta_t = 0.001;
volatile float output,pre_out, DesPos = 0, Des_Angle = pi;
volatile float Kp1 = 1800, Ki1 = 0, Kd1 = 0, Kp2 = 0, Ki2 =0, Kd2 = 0, out1=0, out2;
volatile static float xung_vantoc_hientai, xung_vantoc_truoc, xung_angle_hientai, xung_angle_truoc,angle;
volatile static float Delta_time_vantoc = 0.001;
volatile static float vantoc, vitri, vantoc_cu, err_vantoc, vantoc_goc,vantoc_goc_cu ,theta, distance;
volatile static float HILIM = 100, LOLIM =-100, eReset, uHat, Kb = 0.1, alpha = 0;
uint16_t x=0;
int Tick = 0;
uint32_t count_a,count_b=0;
int16_t counter_a, counter_b=0;
```

Figure 5.2. Variable declaration

❖ Declare some important variables such as:

p_part1, i_part1, d_part1: 3 components of pendulum tilt PID controller

p_part2, i_part2, d_part2: 3 components of cart position PID controller

output: pwm variable to control the motor, calculated from PID functions

Delta_t: differential of time (equal to the timer interrupt)

Kp1, Ki1, Kd1: 3 parameters of inverted pendulum PID controller

Kp2, Ki2, Kd2: 3 parameters of cart position PID controller

theta: tilt angle of pendulum (radians)

distance: position of the cart (millimeters)

```
void QuayThuan() {
HAL_GPIO_WritePin(GPIO_IN1, GPIO_PIN_IN1, GPIO_PIN_SET);    /
HAL_GPIO_WritePin(GPIO_IN2, GPIO_PIN_IN2, GPIO_PIN_RESET);
//HAL_GPIO_WritePin(GPIO_ENA, GPIO_PIN_ENA, GPIO_PIN_SET);
}
void QuayNghich() {
HAL_GPIO_WritePin(GPIO_IN1, GPIO_PIN_IN1, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIO_IN2, GPIO_PIN_IN2, GPIO_PIN_SET);
//HAL_GPIO_WritePin(GPIO_ENA, GPIO_PIN_ENA, GPIO_PIN_SET);
}
void Dung() {
HAL_GPIO_WritePin(GPIO_IN1, GPIO_PIN_IN1, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIO_IN2, GPIO_PIN_IN2, GPIO_PIN_RESET);
//HAL_GPIO_WritePin(GPIO_ENA, GPIO_PIN_ENA, GPIO_PIN_RESET);
}
```

Figure 5.3. Declare a function that controls the direction of rotation

❖ Create H-bridge activation functions to control motor rotation direction, by triggering high or low level for PB3, PB4 pin.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
count_a    =__HAL_TIM_GET_COUNTER(&htim3);
counter_a =(int16_t)count_a;

count_b    =__HAL_TIM_GET_COUNTER(&htim5);
  counter_b =(int16_t)count_b;
  theta = counter_a*2*pi/4000.00;
  deg=counter_a*360.00/4000.00;
  distance = counter_b*14.00*pi/(44.00*6.29);// mm
  vitri = counter_b*360.00/44.00;
}
```

Figure 5.4. Call the encoder pulse counting function and process the signal

❖ Use Encoder Mode to read signals from motor encoder and pendulum, with timer 5 for motor and timer 3 for pendulum.

❖ Encoder pulse returns to microprocessor with x4 mode, then save into 2 variables count_a and count_b. These two variables are converted to 16-bit unsigned integers used to calculate the pendulum angle (theta) and cart position (distance).

```
void DieuKhienLuc() {
if (output > 0) {
QuayThuan();
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, output);
//__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1,output);
//__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2,0);
}
else if (output < 0) {
QuayNghich();
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, -output);
//__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2,-output);
//__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1,0);
}
else {
Dung();
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, 0);
}
}
```

Figure 5.5. Declare the motor control function

❖ Build motor control functions, including pulse width and direction adjustment.

```
void PIDVT(volatile  float DesPos, volatile float distance){

  error2 = DesPos - distance;
p_part2 = Kp2*Delta_t*(error2 - error_truoc2);
i_part2 = Ki2*Delta_t/2*(error2 + error_truoc2);
d_part2 =  Kd2/Delta_t*( error2 - 2*error_truoc2 + pre_pre_Error);
out2 = pre_out + p_part2 + i_part2 + d_part2 ;

pre_pre_Error = error_truoc2;
error_truoc2 = error2;
pre_out = out2;

if (out2 > 9)
out2 = 9;
if (out2 < -9)
out2 = -9;

}
```

Figure 5.6. Declare the PID function to control cart position

❖ Cart position PID controller, which out2 is the output of the PIDVT function that adjusts the vehicle position.

❖ The setpoint of cart position is the input variable of the PIDVT function, called 'DesPos'.

❖ Cart position at each moment is called 'distance'.

❖ p_part2, i_part2, d_part2 are 3 components of the cart position PID controller respectively.

```
void PID_Luc(volatile  float Des_Angle,volatile  float angle){

  error1 = Des_Angle - angle;
  p_part1 = Kp1*error1;
  d_part1 =  Kd1/Delta_t*( error1 - error_truoc1);
//  uDf = (1-alpha)*uDf_truoc + alpha*d_part1;
  i_part1 = i_truoc + Ki1*Delta_t*error1;
//  i_part1 = i_truoc + Ki*Delta_t*error + Kb*eReset*Delta_t;// anti wind up
  error_truoc1 = error1;
  i_truoc = i_part1;
  out1 = (int)(p_part1 + i_part1 + d_part1);
//  output = (p_part + i_part + uDf);// low pass
//  uDf_truoc = uDf;
  if (out1 > 100)
    {
    out1 = 100;
    }
  else if (out1 < -100)
    {
    out1 = -100;
    }
//  if (uHat > HILIM){
//    uHat = HILIM;
//  }
//  else if (uHat < LOLIM){
//    uHat = LOLIM;
//  }
//  else{
//    uHat = output;
//  }
//  eReset = uHat - output;
}
```

Figure 5.7. Declare the PID function to control the pendulum tilt angle

- Inverted pendulum PID controller, which out1 is the output of the PID_Luc function that adjusts the pendulum tilt angle
- The setpoint of tilt angle is the input variable of the PID_Luc function, called 'Des_Angle'.
- The angle of the pendulum at each moment is called 'angle'
- p_part1, i_part1, d_part1 are 3 components of the inverted pendulum PID controller respectively

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
if (htim == &htim2)
  {
    if(start==1)
    {
  if(TrangThai ==0){ // pid vantoc
    PIDTD(vantoc);
    DieuKhienTocDo(); }
  else if(TrangThai == 1){ // pid vitri
    PIDVT(200, distance);
    output = out2;
    DieuKhienLuc();
//    DieuKhienViTri();
    }
  else if(TrangThai == 2){// pid goc 1,2 vong
    PID_Luc(pi,theta);
//    PIDVT(0, distance);
    output = out1-out2;
    DieuKhienLuc();
  }
    Tick++;
    }
    else
    {
    Dung();
    }
}
}
```

Figure 5.8. Enter the timer interrupt function to calculate and control the system

- ❖ Updates motor control and PID parameters after each timer 2 interrupt.
- ❖ Set variable 'start ' = 1 to calculate output and control motor.
- ❖ If variable 'TrangThai' =2, control inverted pendulum pid with 1 feedback loop. The control signal is the output of PID_Luc
- ❖ If variable 'TrangThai' =3, control inverted pendulum pid with 2 feedback loop. The control signal is the output of PID_Luc and PIDVT

```c
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_TIM4_Init();
  MX_TIM3_Init();
  MX_TIM5_Init();
  /* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim1);
HAL_TIM_Encoder_Start_IT(&htim3, TIM_CHANNEL_1 | TIM_CHANNEL_2);
HAL_TIM_Encoder_Start_IT(&htim5, TIM_CHANNEL_1 | TIM_CHANNEL_2);
```

Figure 5.9. Activate the functions in the library

- ❖ In the main function, call the necessary configuration initialization functions.
- ❖ Turn on the timer1, 2 to calculate and control the inverted pendulum.
- ❖ Turn on timer 4 to enable pwm output.
- ❖ Turn on timer 3, 5 to read encoder pulses from pendulum and motor

```c
  while (1)
  {
    /* USER CODE END WHILE */
char buffer[50];
sprintf(buffer, "Vantoc:%d Vitri:%d Angle:%.9f\n", (int)vantoc, (int)vitri,theta);
HAL_UART_Transmit(&huart5, (uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);
    /* USER CODE BEGIN 3 */

  }
```

Figure 5.10. UART reading

- ❖ Read variables sent from microprocessor via uart

Chapter 6. ANALYZE AND EVALUATE THE MODEL RESULTS ACHIEVED

6.1. Configuration

6.1.1. Advanced Serial Port Terminal

❖ Configure Terminal to sync with CubeMX as follows:

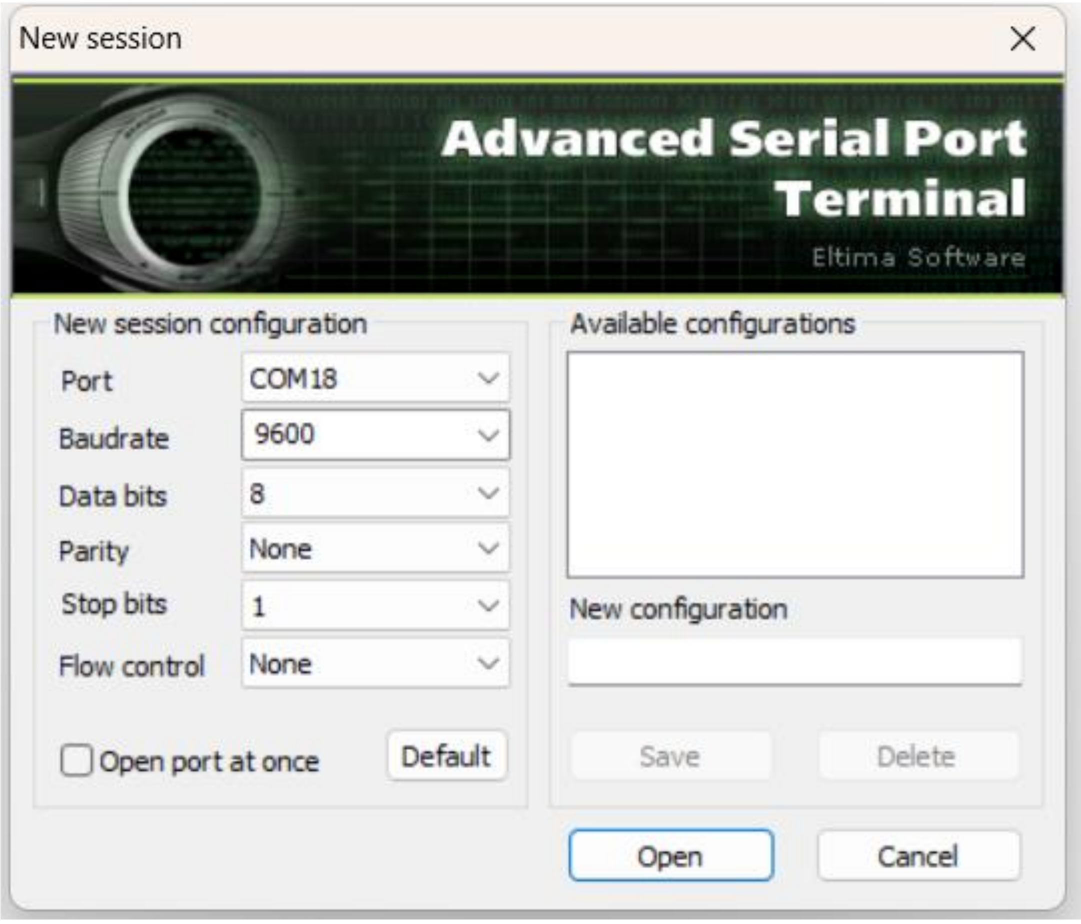

Figure 6.1. Terminal configuration

6.1.2. STM32 Cube Monitor

❖ Go to the path and select the .axf executable file, then select the variables to monitor: "Des_Angle", "theta". Finally, click "DEPLOY" and open the Dashboard to see the graph drawn.

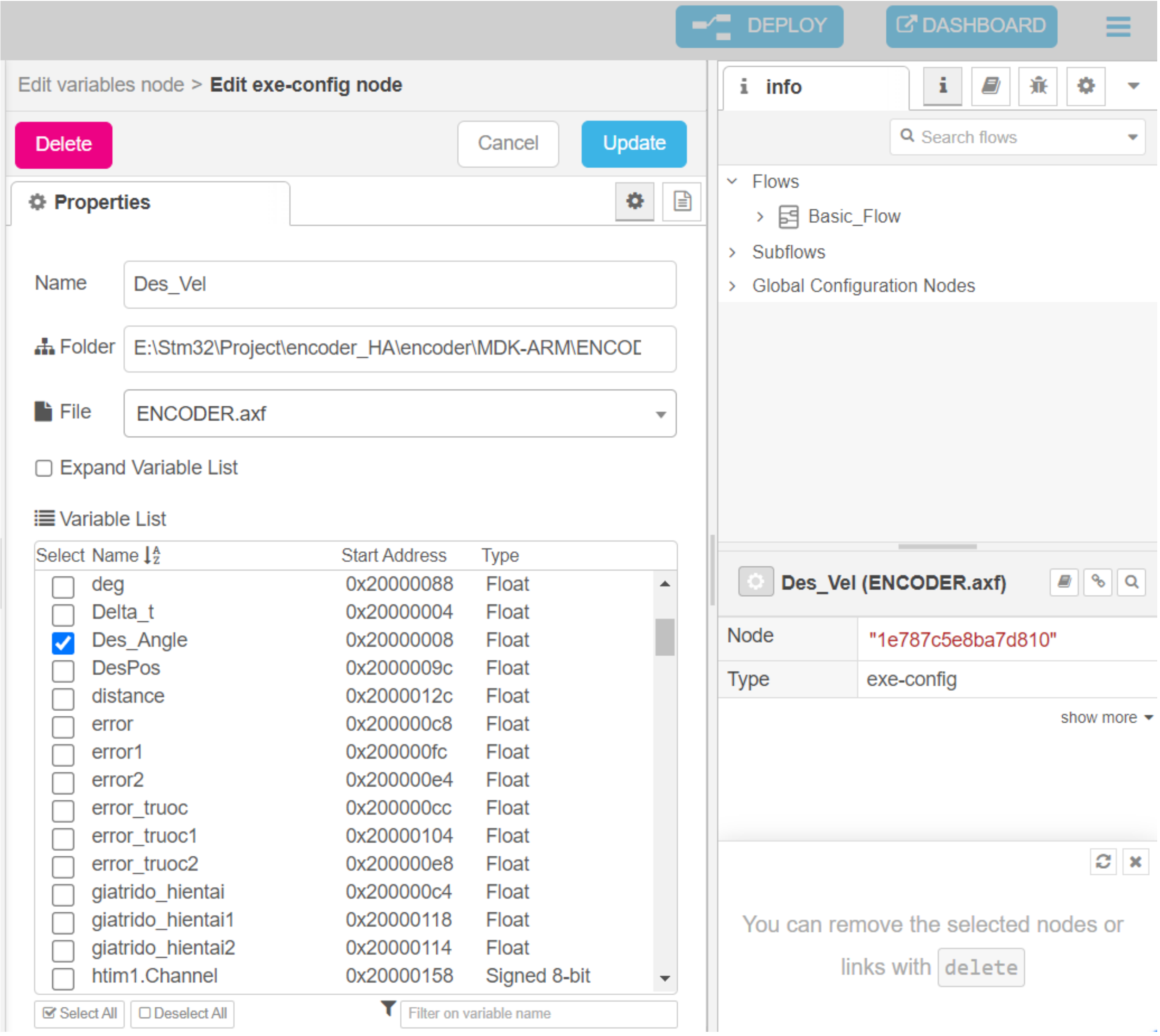

Figure 6.2. STM32 Monitor configuration

6.2. Experimental results

6.2.1. Advanced Serial Port Terminal

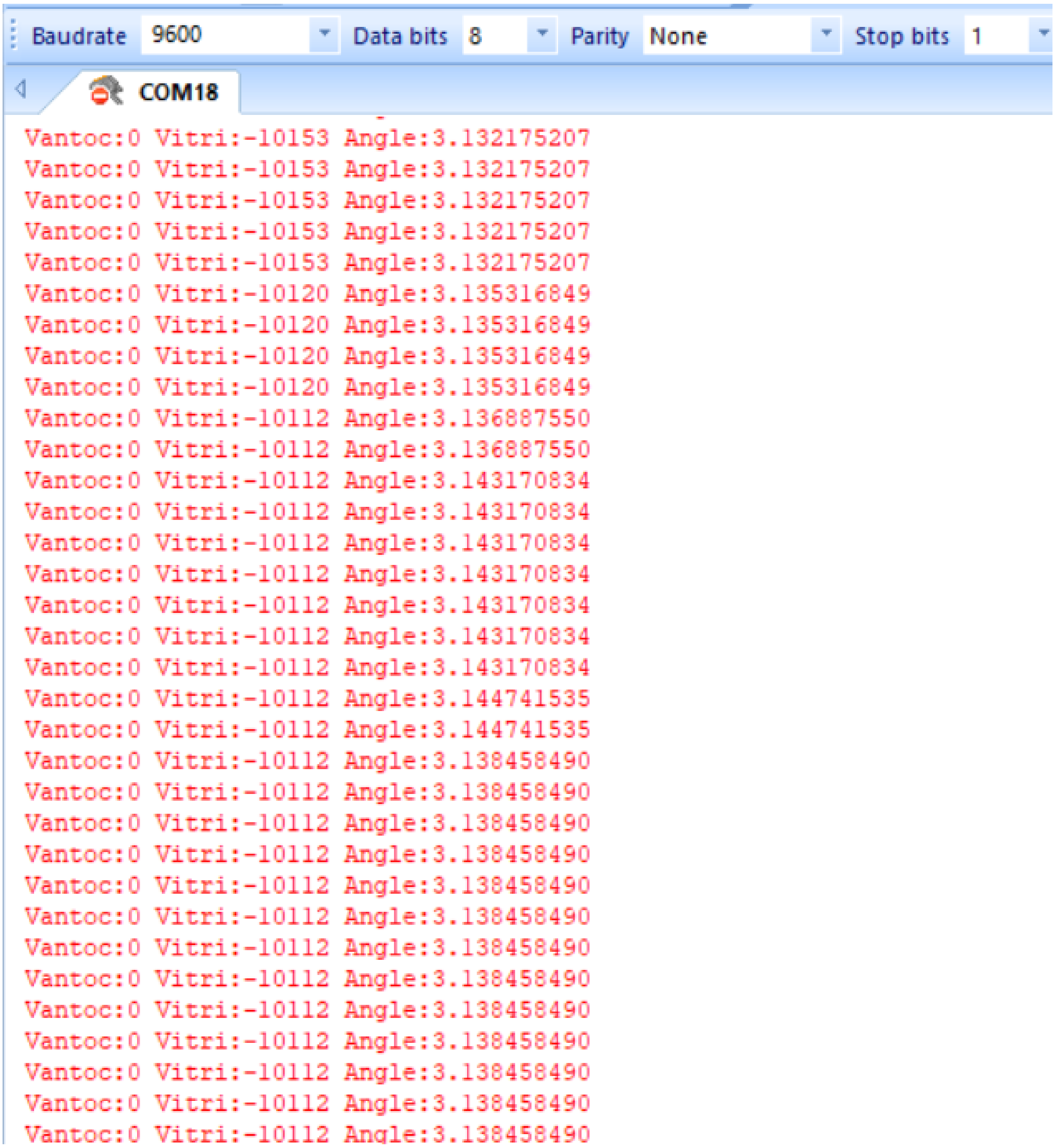

Figure 6.3. Pendulum tilt angle data from terminal

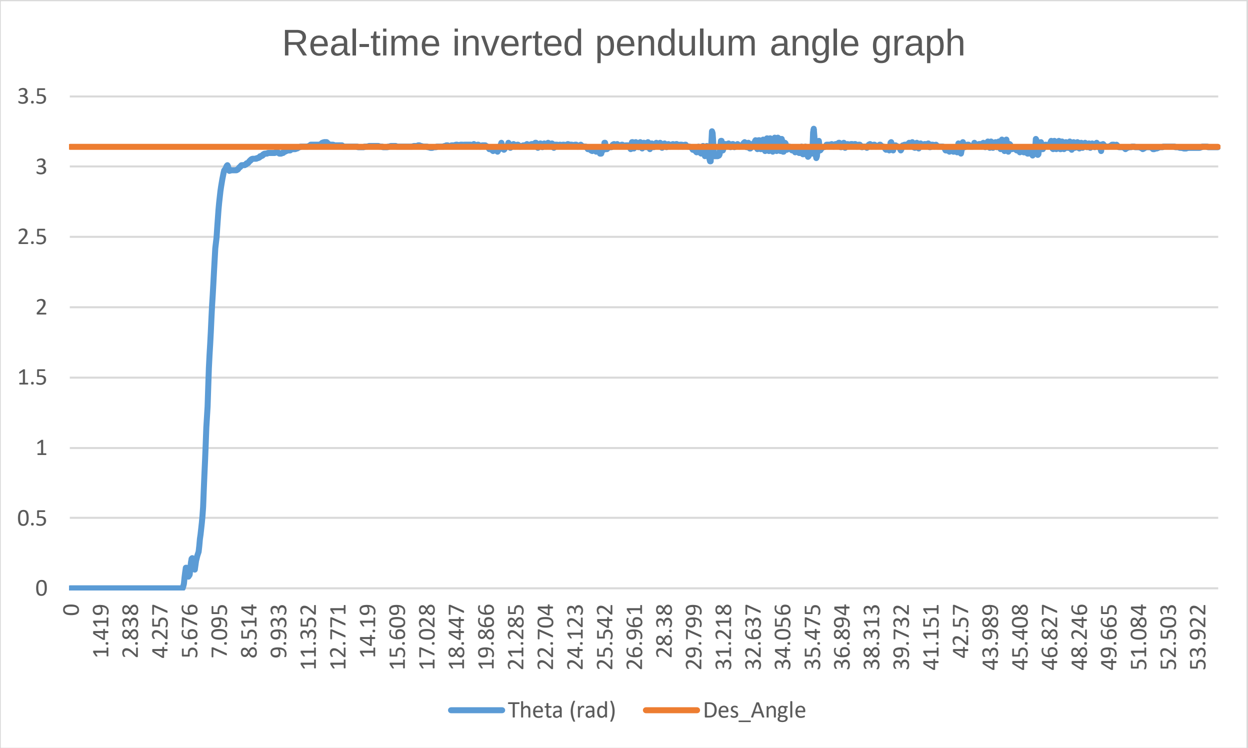

Figure 6.4. Graph of inverted pendulum angle

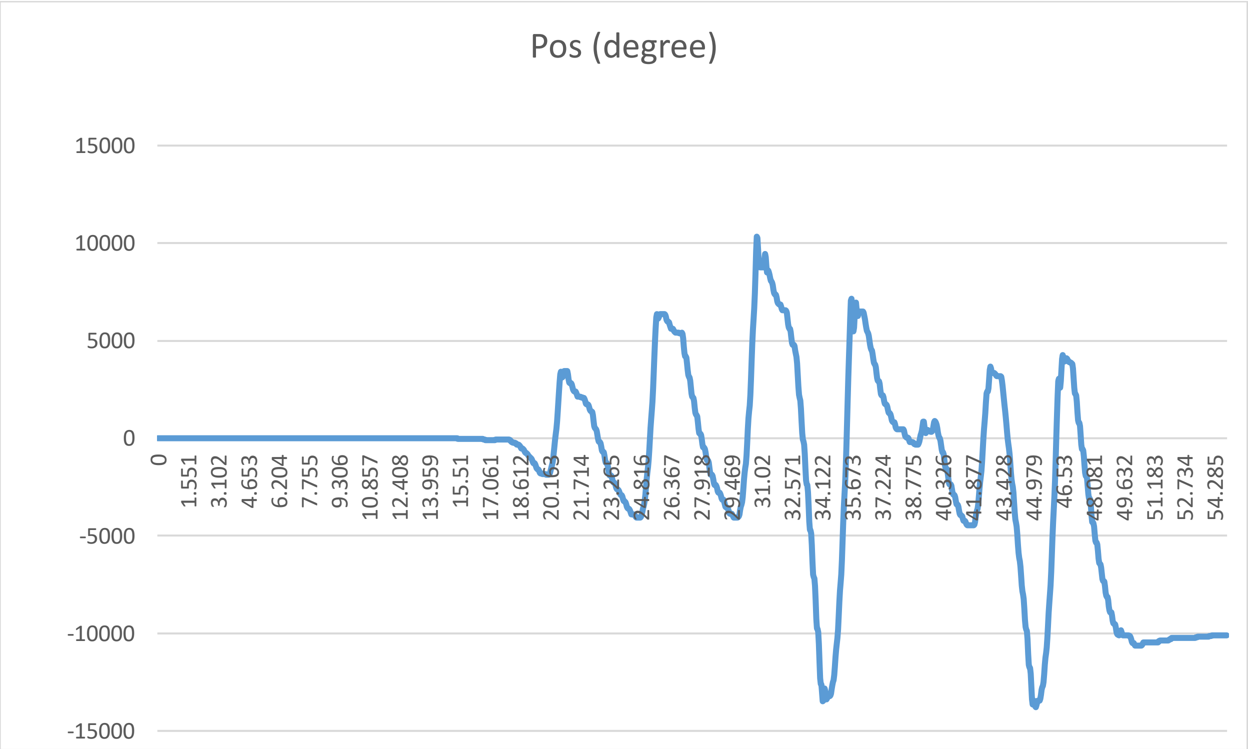

Figure 6.5. Graph of cart position

6.2.2. STM32 Cube Monitor

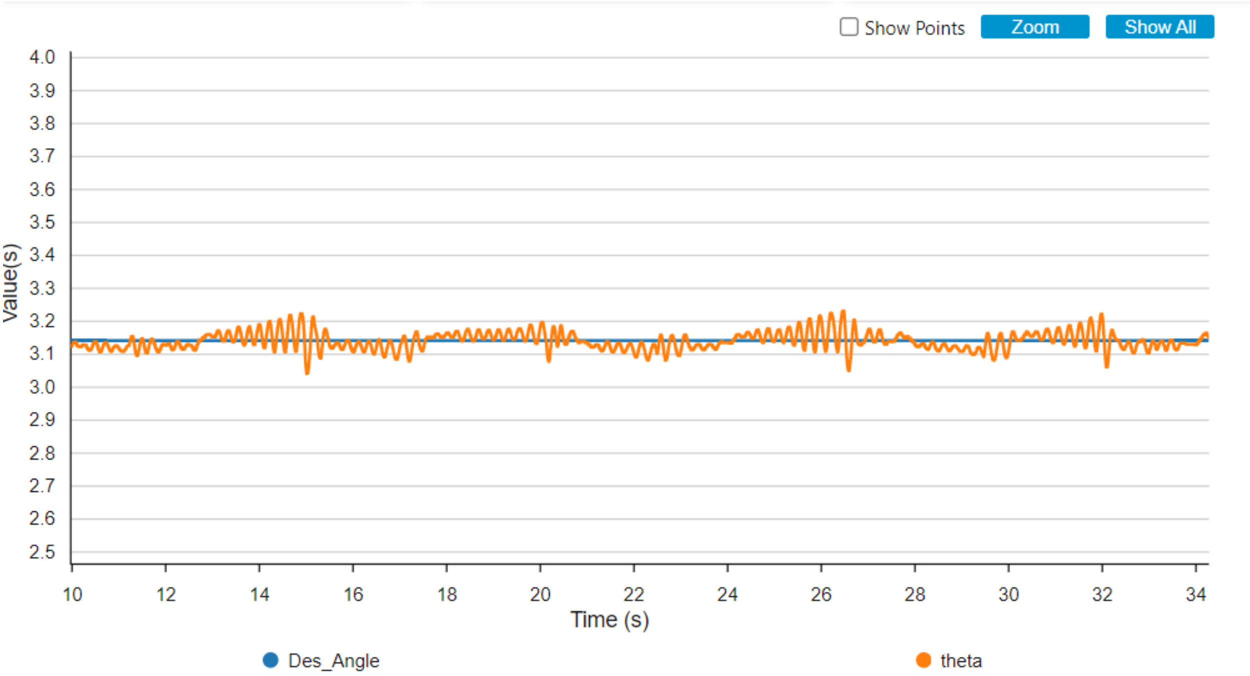

Figure 6.6. Graph of inverted pendulum angle

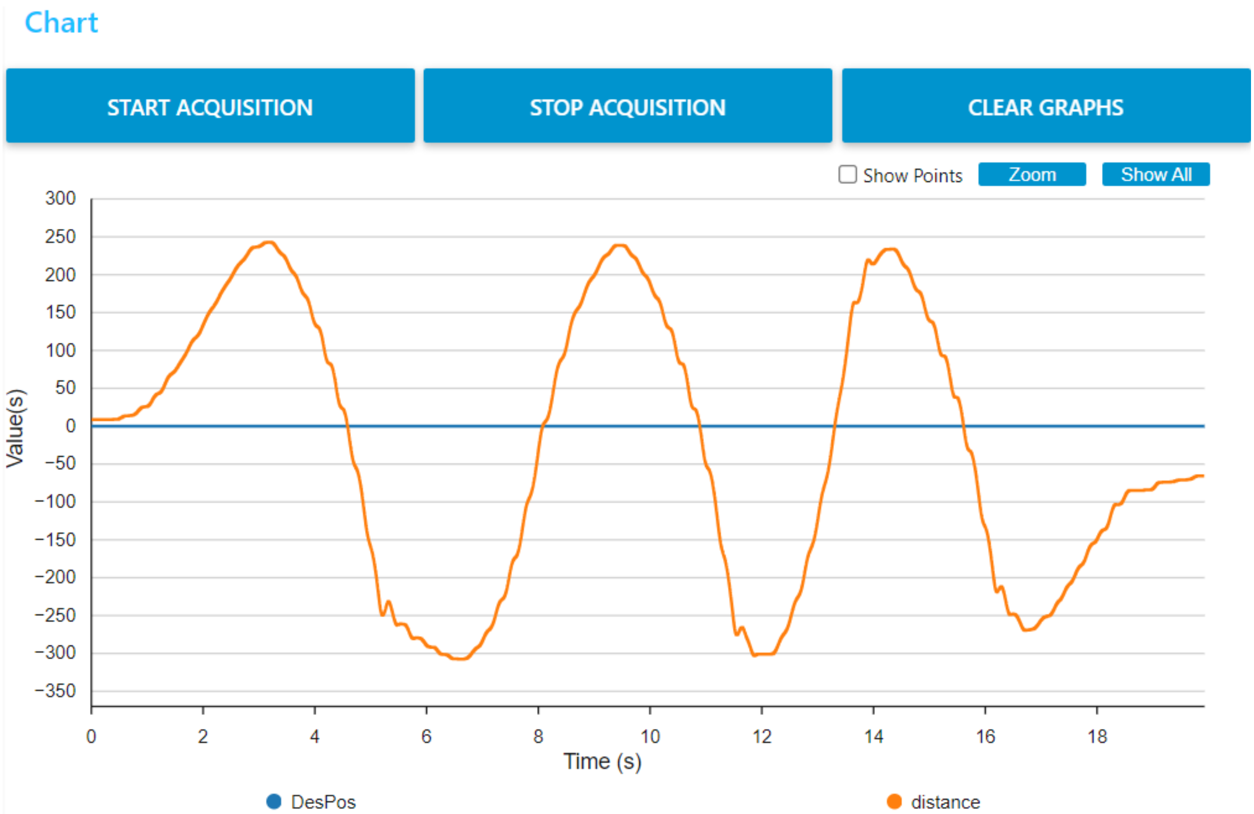

Figure 6.7. Graph of cart position

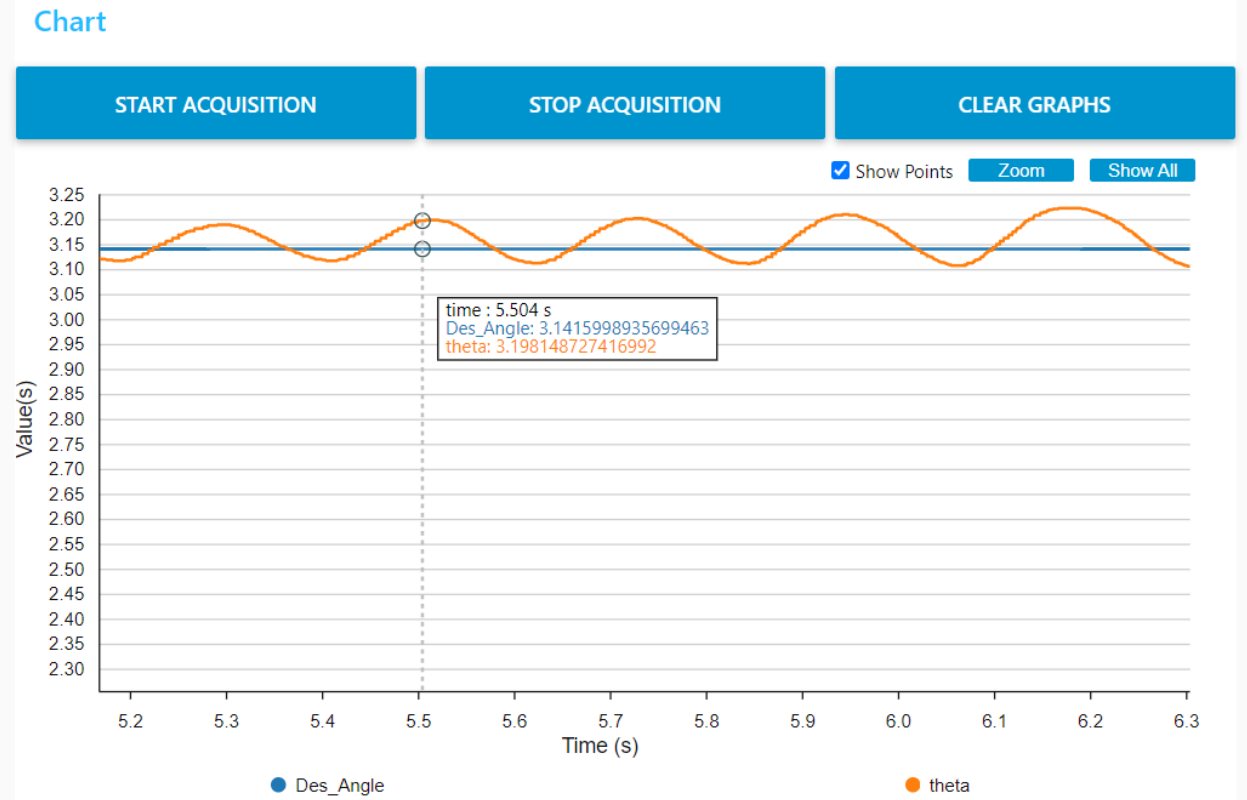

Figure 6.8. Graph of inverted pendulum angle

6.3. Analysis and evaluation of results

- With "Des_Angle" = 3.1416 rad (angle setting value), the system responded and responded quickly through the "theta" variable.
- After many experiments, the system has a response time of 0.1 seconds, with a maximum positive deflection angle of 3.270406 rad and a maximum negative deflection angle of 3.037927 rad.

Chapter 7. CONCLUSION AND RECOMMENDATION

7.1. Conclusion

- ❖ After the process of research, calculation and construction, the model meets most of the requirements. In terms of hardware, the blocks are connected and operate in accordance with the design principle. Peripherals communicating with the STM32 F407 VGT-Disc board are stable and have almost no latency. In terms of software, the model control program has been completed with the set goal: balancing an inverted pendulum with a PID controller. The model has operated successfully and fully with the proposed requirements.
- ❖ Although it has met the requirements, the model still has noise and does not work well in reality, even affecting the stability of the system.

7.2. Recommendation

In order for the model to be complete and consistent with reality, some recommendations for improvement and development are made as follows:

Improvements:

- ❖ Improved hardware to reduce noise such as adding joints to firmly fix the pendulum
- ❖ Increase resolution or reduce timer interrupt time to make the microprocessor control the system smoother

Developments:

- ❖ Use other controllers specialized for nonlinear systems to balance the inverted pendulum.
- ❖ Developing the current topic into a 2-degree-of-freedom inverted pendulum
- ❖ Create an inverted pendulum control interface using Raspberry

REFERENCES

[1] RM0090, STM32F4 Series Manual Reference (Datasheet)

https://www.st.com/resource/en/reference_manual/rm0090-stm32f405415-stm32f407417-stm32f427437-and-stm32f429439-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2] Mạch điều khiển động cơ DC BTS7960 43A High-power Motor Driver

https://hshop.vn/mach-dieu-khien-dong-co-dc-bts796043a-1-dong-co

[3] E6B2-CWZ6C Rotary Encoder 1000 Xung NPN

https://www.thegioiic.com/e6b2-cwz6c-rotary-encoder-1000-xung-npn

[4] Module Chuyển Đổi USB UART TTL CP2102

https://dientutuyetnga.com/products/module-chuyen-doi-usb-uart-ttl-cp2102

[5] Động cơ giảm tốc kèm encoder 12V 1590RPM GA37

https://dientunguyenhien.vn/show/3490

[6] WIKIPEDIA. PID controller. Accessed 10/13/2024

https://vi.wikipedia.org/wiki/B%E1%BB%99_%C4%91i%E1%BB%81u_khi%E1%BB%83n_PID

[7] Control Tutorials for MATLAB and Simulink, Inverted Pendulum: PID Controller Design

https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlPID

[8] Thiết kế mô hình cân bằng con lắc ngược

https://drive.google.com/drive/folders/1m2Lm2WwPbAEuUQ2nN8JwNEULB7nGLSCn

[9] Thanh Dong Ngo, ĐIỀU KHIỂN ĐỘNG CƠ ENCODER DC_STM32 control DC motor

https://www.youtube.com/watch?v=XLm-EK-jN_Y