POLYTECHNIC OF BARI

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING

AUTOMATION ENGINEERING MASTER'S DEGREE

Course on EMBEDDED CONTROL

Prof. Engr. Luca **DE CICCO**

Project work:

Furuta Pendulum

Students:
CARAMIA Donato
D'ALESSANDRO Vito Ivano
SOLETI Giovanni
VENEZIA Antonio

ACADEMIC YEAR 2019-2020

Abstract

This report presents a physical and theoretical implementation of Furuta Pendulum. Matematical model is obtained using Autodesk Inventor combined with Simscape Multibody, then, in Simulink, a LQR controller has been implemented. Due to a non-perfect data identification of the system, the LQR controller developed in Matlab does not work on the real system so a cascade PID controller tuned using Ziegler-Nichols method is used in order to keep the pendulum in the upward equilibrium point and the arm's angular velocity null.
In addition, all the produced materials are available on this link:
https://drive.google.com/drive/folders/1v1NON1EUI4vkj21B9rY8gF79eYCKx3qO?usp=sharing

Contents

1 Introduction

Among the under-actuated systems, the Furuta Pendulum, also known as rotary inverted pendulum, is a classic example of nonlinear controller. More in depth, it is a mechanism that has two degrees of freedom (DOF) and two revolute joints. It is essentially composed by three elements: a motor and two bars called arm and pendulum. The motor's shaft is connected to one end of the arm, which allows the arm to rotate in the horizontal plane, whereas the pendulum is joined to the free end of the arm through a revolute joint, allowing the rotation of the pendulum in the vertical plane. The objective of Furuta pendulum is to keep both pendulum angle and arm velocity null.

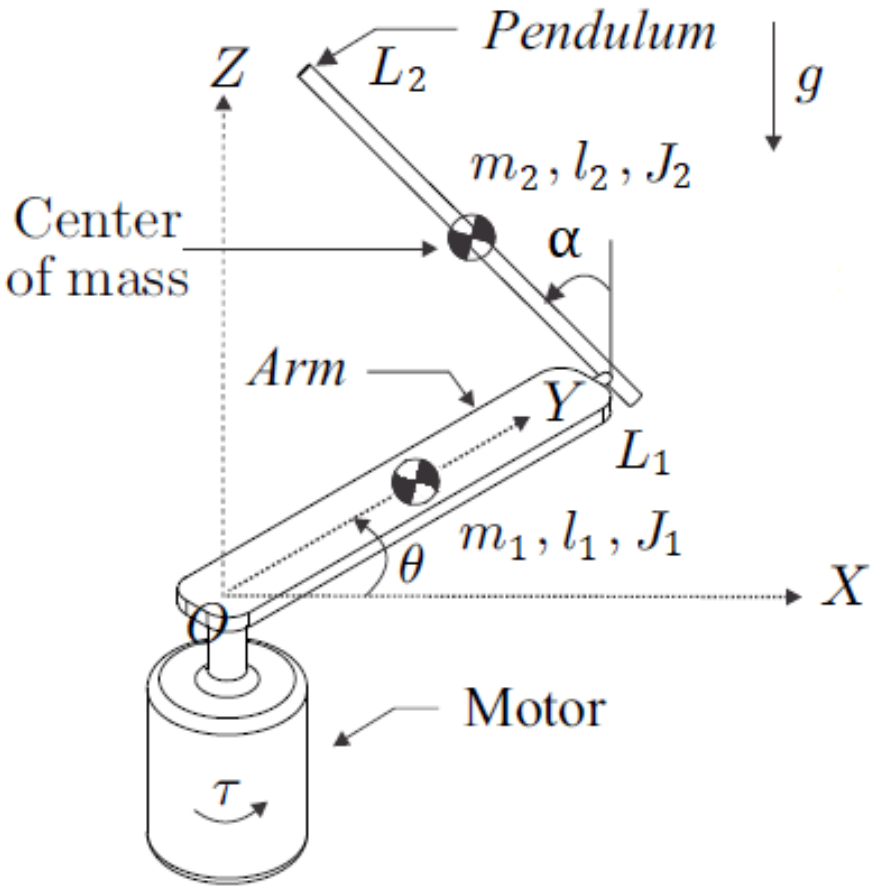


Figure 1.1: Schematic of Furuta Pendulum

2 Software Prototyping

The use of a computer-aided design application for 3D mechanical design such as Autodesk Inventor Professional 2020 allows to realize a 3D model of Furuta Pendulum, specifying all the physical parameters of each rigid bodies of the system, avoiding the hand-writing of the system's equations of motion. Thus, the 3D Cad model can be imported in Simulink (using an add-on, Simscape Multibody) obtaining, after a proper linearization in the desired pendulum position, the state-space model in order to make numerical simulations.

Figure 2.1: Furuta pendulum 3D CAD model

The control system implemented in Simulink is shown in Fig. 2.2 where the block named as "Furuta Pendulum" contains all the sub-blocks describing the physical system. These sub-blocks are shown in Fig. 2.3.

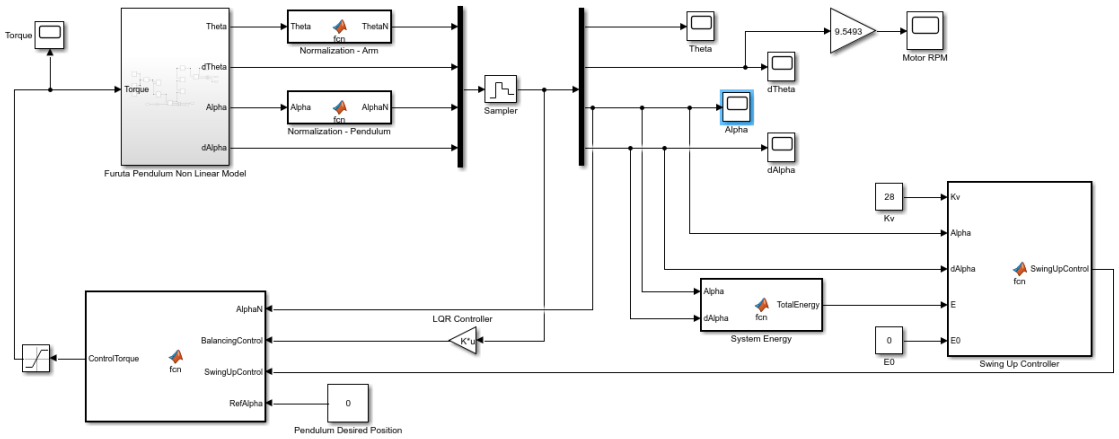


Figure 2.2: Simulink Model

The control architecture is based on the implementation of two main controllers:

- balancing controller, based on LQR feedback controller, used to stabilize the pendulum on the desired set-point;
- swing-up nonlinear controller, based on monitoring the energy of the system, used to swing the pendulum up from the downward to the upward position.

Simulink model also presents different blocks and Matlab functions that accomplish different tasks:

- a gain block which calculates the motor torque for LQR control;
- a Matlab function that determines the motor torque required in the swing-up controller, feeded by a block that calculates the energy of the system at each iteration;
- a block that switches between balancing and swing-up controller according to the angular position of the pendulum;
- blocks that normalize the angular position of the pendulum and the arm in range $[-\pi;\pi]$;
- a sampler simulating states data acquisition from sensors with a frequency of 200 Hz.

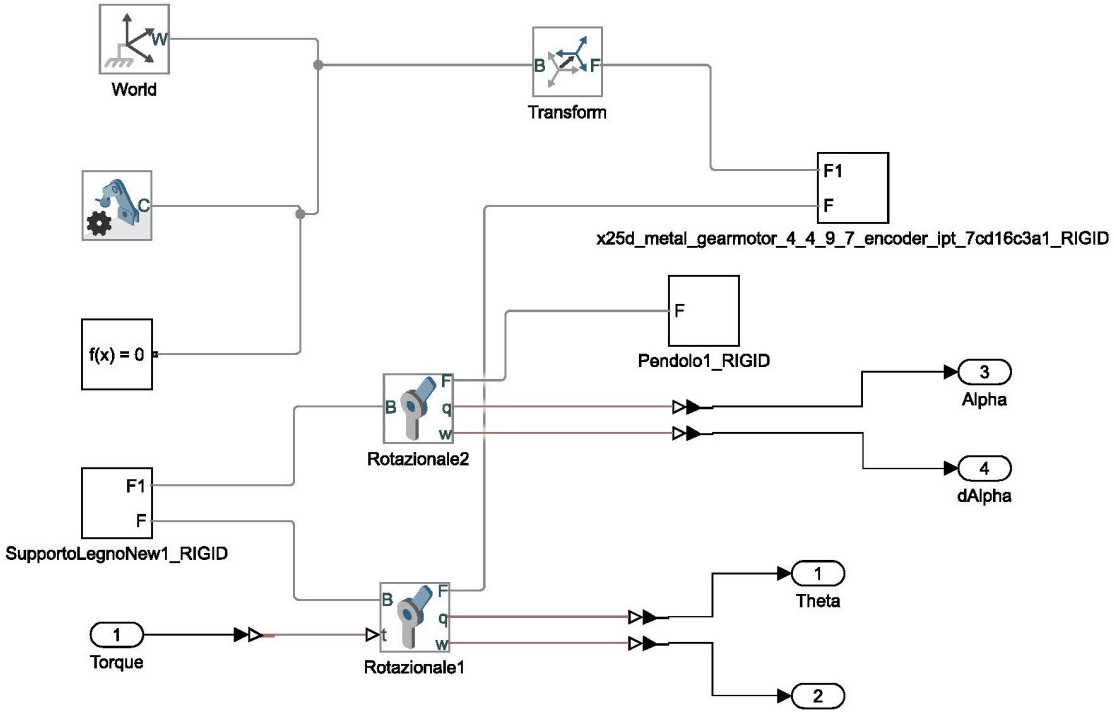


Figure 2.3: Furuta Pendulum Sub-Block

After:

- empirically tuning damping coefficients for the two revolute joints,
- linearizing the system around the unstable equilibrium point in order to set the LQR controller,
- tuning the proportional coefficient K_v of the swing-up controller properly,

the control system can swing the pendulum up from the downward position and stabilizing it in the upward position in few seconds.
As can be seen in Fig. 2.5, the pendulum reaches the upward position and keep it stable, with a low average torque required.

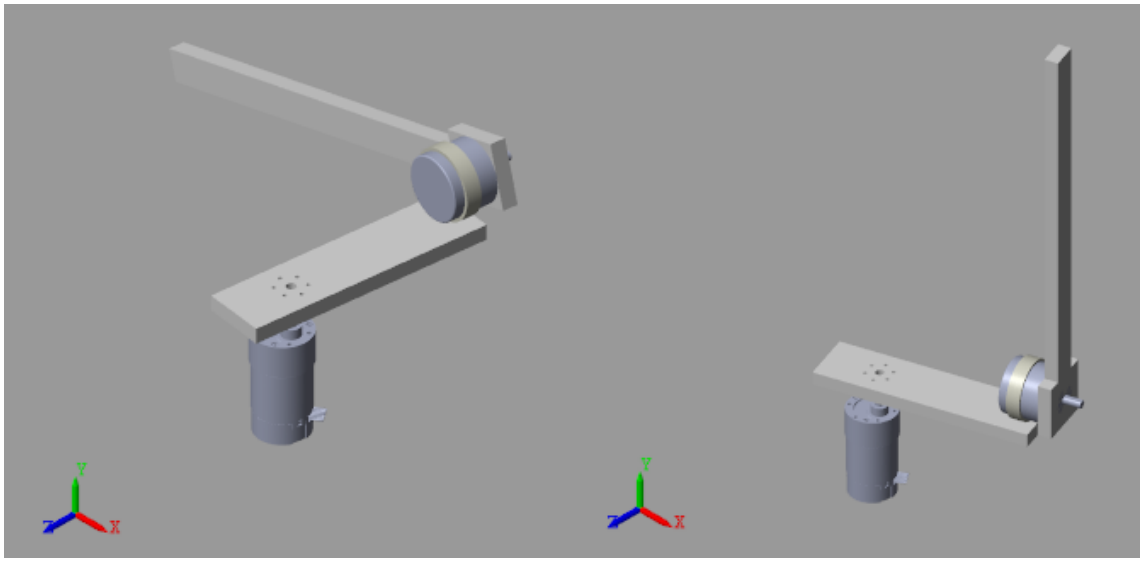


Figure 2.4: Pendulum from the swing-up to balancing using LQR

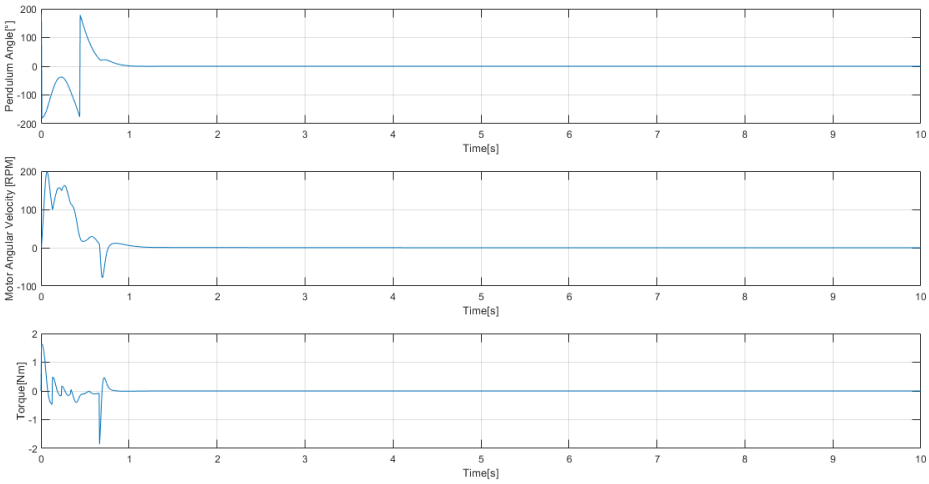


Figure 2.5: Pendulum angle, motor angular velocity and torque in the swing-up and balancing

At the beginning, a LQR controller combined with a nonlinear controller based on energy have been chosen due to their efficiency in this type of control system. After few tests on the real test bed, it has been noticed that the response of the real system did not match the software simulations. This may happen for the following reasons:

- inaccurate model identification;
- structural problems (presence of vibrations, elevation gain) due to the test bed.

Consequently, a new control system based on PID controllers has been implemented.

3 Control System

The actual control system implemented is shown in Fig. 3.1.

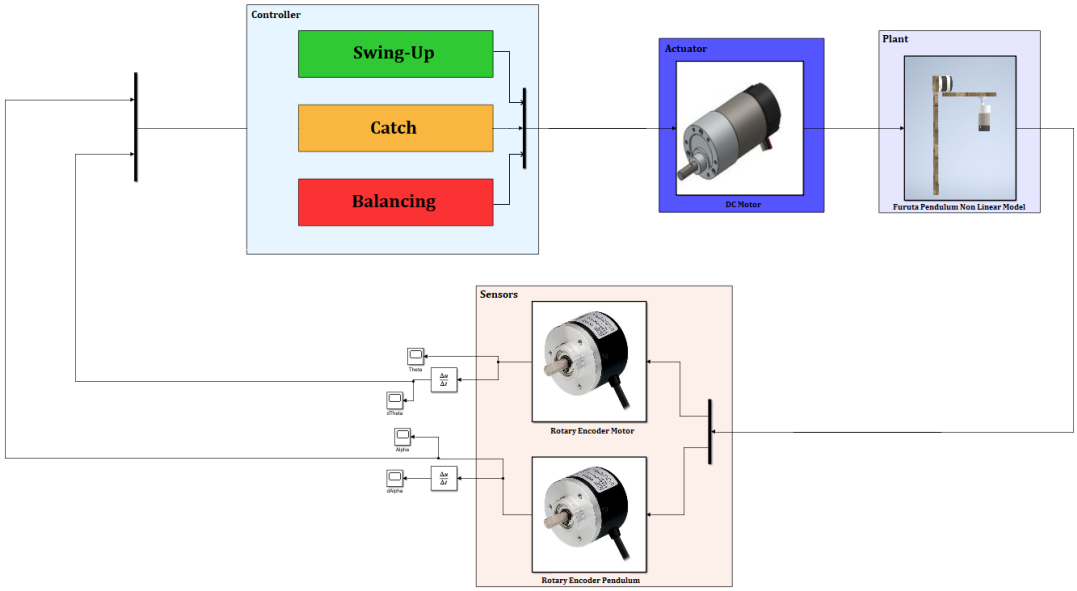


Figure 3.1: Block diagram of Furuta Pendulum's control system

It is based on using, consecutively, three main sub-controllers:

- swing-up controller, to drive the pendulum from the downward position to the upward unstable position;
- catch controller, to slow down the pendulum when nearby to the unstable position;
- balancing controller, to keep the pendulum in the upward position and the motor with null velocity, even if disturbances are applied.

The control action (torque applied by the DC motor) depends on the rotational angle α:

- if $-10° < \alpha < 10°$ the control action is dictated by the balancing controller;
- if $-35° < \alpha < -10°$ or $10° < \alpha < 35°$ the control action is dictated by the catch controller;
- otherwise, the control action is dictated by the swing-up controller.

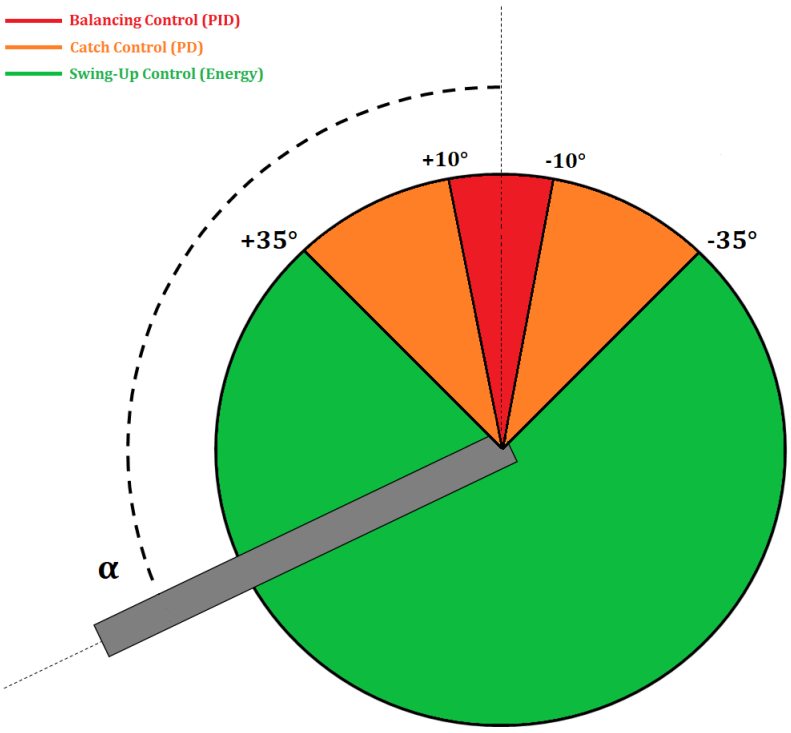


Figure 3.2: Different kind of algorithm control

3.1 Controller

In the following sections, each controller will be analyzed more in depth.

3.1.1 Swing-Up Control

The objective of the swing-up control is to move the system from the stable equilibrium point to the unstable one. It calculates the total energy based on the kinetic energy of both links, and potential energy of the pendulum. This value is compared to a defined quantity of energy when the pendulum is balanced (upward equilibrium point). The difference between desired energy and actual energy is multiplied by a gain, obtaining a torque that can be applied to the motor. The energy level in the balanced position is defined to be equal to the potential energy in that point. The control law to realize the above idea is given by the following equation:

$$\tau = K_v \cdot (E - E_0) \cdot sign(\dot{\alpha} \cdot cos(\alpha)) \tag{1}$$

The first two terms are the aggressivity gain K_v and the difference between actual and desired system energy $(E - E_0)$. These two terms provide the magnitude of energy that has to be added to the system at any given time. The aggressivity gain determines what proportion of the available input will be used to increase or decrease the system energy.
Normalized actual energy is determined by:

$$E = \frac{m_p g l_p}{2} \left(\left(\frac{\dot{\alpha}}{w_0} \right)^2 + cos\,(\alpha) - 1 \right) \tag{2}$$

where w_0 is the frequency of small fluctuations of the pendulum in the nearby the downward position.

$$w_0 = \sqrt{\frac{m_p g l_p}{I_p}}$$

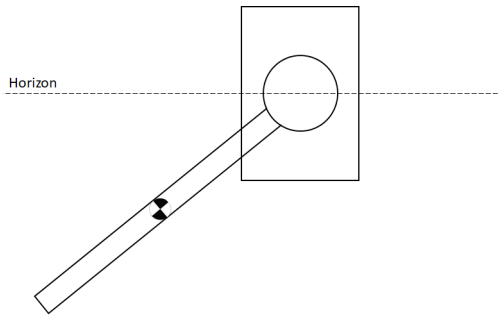
where

- I_p is the moment of inertia of the pendulum;
- l_p is the distance between the pendulum center of mass and the revolute joint that connect the pendulum to the arm;
- m_p is the mass of the pendulum.

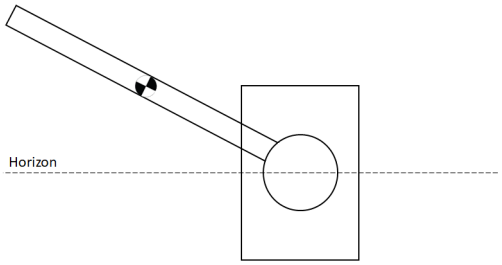
The value of the total energy depends on the selection of the "base" with zero energy. There are more options for the base: downward equilibrium position, horizontal position or up-right equilibrium position. The most suitable choice is the up-right equilibrium, since there is no reason to add more energy to the pendulum in this position. According to (2), the energy of the system in the downward position is $-2m_p g l_p$ and it is also the energy that needs to be delivered to the system, whereas the energy of the system in the upward position is equal to 0 .
Therefore, it is necessary to apply a control law that would consider the current gained energy of the system and would add energy in the right direction, at the same time.
For this choice, the desired system energy is the energy of the pendulum when it reaches the upward equilibrium point that corresponds to the only potential energy of the pendulum, so

$$E_0 = 0$$

The second half of the energy swing-up equation determines the direction the input should be applied to increase the energy of the system. The velocity term causes the input to change directions when the pendulum stops and begins to swing in the opposite direction. The cosine term is negative when the pendulum is below horizontal and positive above it. This helps the driven link to get under the pendulum and catch it as shown in Fig. 3.1 [1].



(a) Pendulum under the horizon



(b) Pendulum over the horizon

Figure 3.1: Sign function effect on swing-up

3.1.2 Balancing Control

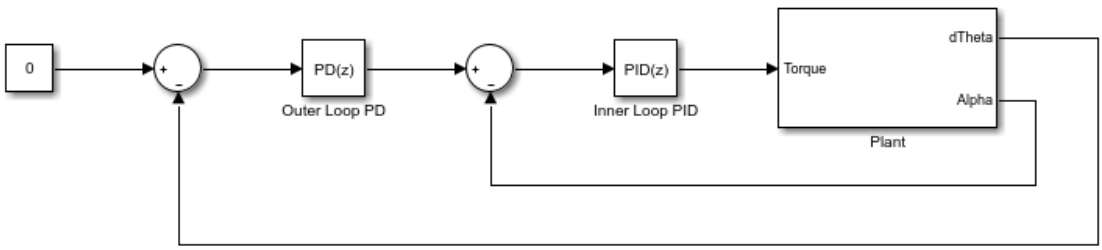


Figure 3.2: Cascade PID

To balance the pendulum in the up-right position and not move the arm, a cascade PID [2] controller has been used:

- the outer loop PD controller has as set-point the angular velocity of the motor shaft $\dot{\theta} = 0$. The error will be the difference between the set-point and the actual angular velocity of the motor shaft, so the set-point of the inner PID is variable;
- the inner PID loop controller has as set-point the difference between the output of the outer loop controller and the actual value of the angular position of the pendulum.

The first step in tuning a PID controller is setting the K_i and K_d values to 0 and only focusing on the K_p value in order to find the critical gain K_c. The proportional gain has been increased until

the pendulum was able to hold its up-right position for a few seconds while rapidly oscillating. Using a tool of STMCUBEIDE, DataTrace, the system's response has been determined while the pendulum is oscillating near the unstable equilibrium point: in this way the period of oscillations has been computed. Known the critical gain and the period of oscillations, Ziegler-Nichols PID tables can be used to tune the inner loop PID.

Control Type	K_p	K_i	K_d
P	$0.5K_u$		
PI	$0.45K_u$	$0.54K_u/T_u$	
PD	$0.8K_u$		$K_u\ T_u/10$
PID	$0.6K_u$	$1.2K_u/T_u$	$3K_u\ T_u/40$

Table 3.1: Ziegler-Nichols tuning PID controller

More in depth, one major flaw of the P controller was that it would begin to lean to one side and eventually begin to plummet down. Adding the integral term will stop the pendulum from over extending on either side of the axis. If the pendulum starts to lean one way, the error on that side will begin to accumulate and the integral term of the controller will slowly add more force over time to lean in the other direction. Finally, to decrease the rapid oscillation, we can introduce the derivative term. The derivative term will decrease the output if the pendulum is moving towards the set-point and increase the output if the pendulum is moving away, producing a "dampening" effect when the pendulum is moving to the vertical position. This will decrease overshoot and give a smoother and reliable controller. The outer loop PD controller instead, has been tuned manually, according to the fact that this controller has to be less aggressive respect to the inner loop one.

3.1.3 Catch Control

Once the cascade PID and Energy controllers were both working, it must combine them together. The goal for combining the controllers is to make the pendulum automatically swing-up after initialization and begin balancing. The swing-up controller is clearly able to swing the pendulum to the up-right position. However, a problem occurs in making the PID catch the pendulum when it is nearby the top: in fact, if the pendulum has too high velocity, it may get out from the balancing area. The optimal operation of the balancing PID is 10 °around the set-point. Overshooting the set-point will cause the cascade PID to react with large oscillations that will eventually cause the pendulum to fall again. To remedy this, an intermediate area has been set up between the swing-up and balancing zone just to catch the swing-up. When the pendulum is between 10 and 35 degrees on either side, a PD controller with high K_d will be first executed. The reasoning here is to attempt to dampen the pendulum's movement after the swing-up phase. Once the pendulum reaches the optimal operation range for the stabilizing cascade PID, it will kick in and begin balancing the pendulum.

3.2 Bill Of Materials

One of the aim of this project is to realize a working Furuta Pendulum with a very low budget respect to the working rotary inverted pendulum can be bought (e.g Quanser). After a deep research, a Bill of Material has been defined according to the budget (about 100 €). Some of components have been recycled and, consequently, not payed. In the following sections the complete BoM is presented, with a detailed description of the main components like sensors, actuators and mechanical parts.

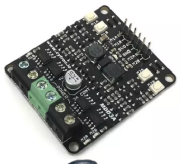
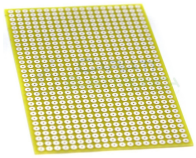
Components	Price	Figure
DC Gear Motor (70:1) + Rotary Incremental Encoder (64 CPR)	40 €	
Rotary Incremental Encoder (2400 CPR)	15 €	
2 Mounting Hubs	8 €	
Driver	7 €	
Ring Clamp	Not Payed	
Wood Arm	Not Payed	
Wood Pendulum	Not Payed	
Millefori Card	Not Payed	
STM32F446RE	17 €	
Power Supply 12V	Not Payed	

Table 3.2.1: BoM

3.3 Rotary Inverted Pendulum

The base of Furuta Pendulum is made by pieces of galvanized iron rigidly connected with a welding as shown in Fig. 3.3.1.

Figure 3.3.1: Base of Furuta Pendulum

In order to keep the base parallel to the ground, four adjustable pins have been used, as shown in Fig. 3.3.2.

Figure 3.3.2: Adjustable pins for the base

At the center of the base there is a galvanized iron pylon used to place the DC motor which is connected to the pylon using a ring clamp, as can be seen in Fig. 3.3.3.
A mounting hub allows to rigidly connect the shaft of the DC motor with the wooden arm. To the tip of the wooden arm, another rotary incremental encoder is placed using a ring clamp, too. The pendulum is connected to the shaft of the rotary encoder using another mounting hub. The whole

physical system is shown in Fig. 3.3.3.

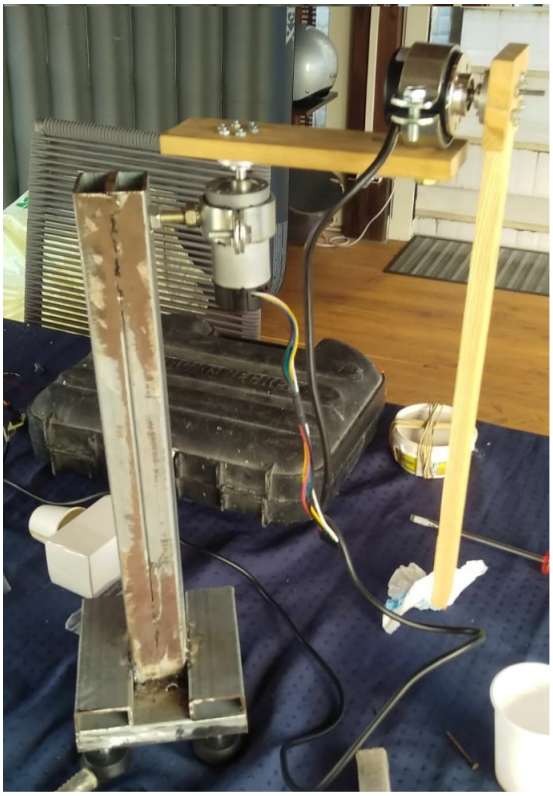


Figure 3.3.3: Test bed

3.3.1 Physical System

In Tab. 3.3.1 the main test bed parameters are reported.

	Name	Value	Unit
Arm parameters	mass	0.025	kg
	length	0.16	m
	thickness	0.01	m
	center of mass	0.08	m
Pendulum parameters	mass	0.04	kg
	length	0.32	m
	thickness	0.01	m
	center of mass	0.15	m
	Inertia	0.0003	$kg \cdot m^2$
DC Motor parameters	Ra	2.1818	Ω
	Ka	0.509434	
	Vm	12	V
	I_m No Load	0.2	A
	P_m No Load	2.4	W
	w_m No Load	150	RPM
	Stall Torque	2.64	Nm
Rotary Encoder Motor	CPR	64	
Rotary Encoder Pendulum	CPR	2400	

Table 3.3.1: Physical system parameters

3.4 Sensors

Initially, an Inertial Measuring Unit (IMU) was chosen to get pendulum states because it was available without paying it. After few tests, this type of solution appeared to be worthless due to the complexity of getting the desired measurements and the presence of high noise on the feedback signal. Thus, states of the system are read using two rotary incremental encoders: in particular, the rotary encoder attached on the pendulum has higher accuracy respect to the motor one due to the gear wheels. This choice allows to apply a more accurate control action to the angular position and velocity of the pendulum. In particular, the rotary encoder related to the DC motor has a resolution of 64 CPR so the maximum resolution error (considering the gear ratio of DC motor) is equal to

$$e_q = (360/(64 \cdot 70))/2 \simeq 0.04^\circ$$

while the rotary encoder of the pendulum has a resolution of 2400 CPR so the maximum resolution error is equal to:

$$e_q = (360/2400)/2 \simeq 0.05^\circ$$

3.5 Actuator

In the study case, the actuator is a DC gear motor driven by a PWM driver.
The advantages of using a DC motor are:

- Controlling a DC motor is as simple as a switch. Simply apply a voltage to start driving it. It slow down when the voltage is lowered, and spin in the other direction when the voltage is reversed;
- High torque is achieved at low speeds;

- DC motors are about 75-80 % efficient;
- DC motors are not expensive.

On the other hand, the disadvantages are:

- Aside from the audible noise from the rubbing parts, electromagnetic noise is also generated as a result of the strong sparks that occur at areas where the brushes pass over the gaps in the commutator. This can potentially cause interference in other parts of the system;
- Brushes could get easily worn out as a result of continuous moving contact and require constant maintenance. Speed could be limited due to brush heating;
- DC gear motors have a low angular resolution due to the presence of gear wheels that amplify the torque of the only DC motor.

The motor driver used has as major feature the maximum input frequency of PWM signals equal to 20 kHz, so it means that the driver can receive a PWM input signal with a frequency $\leq 20\ kHz$.

3.6 Electrical Connections

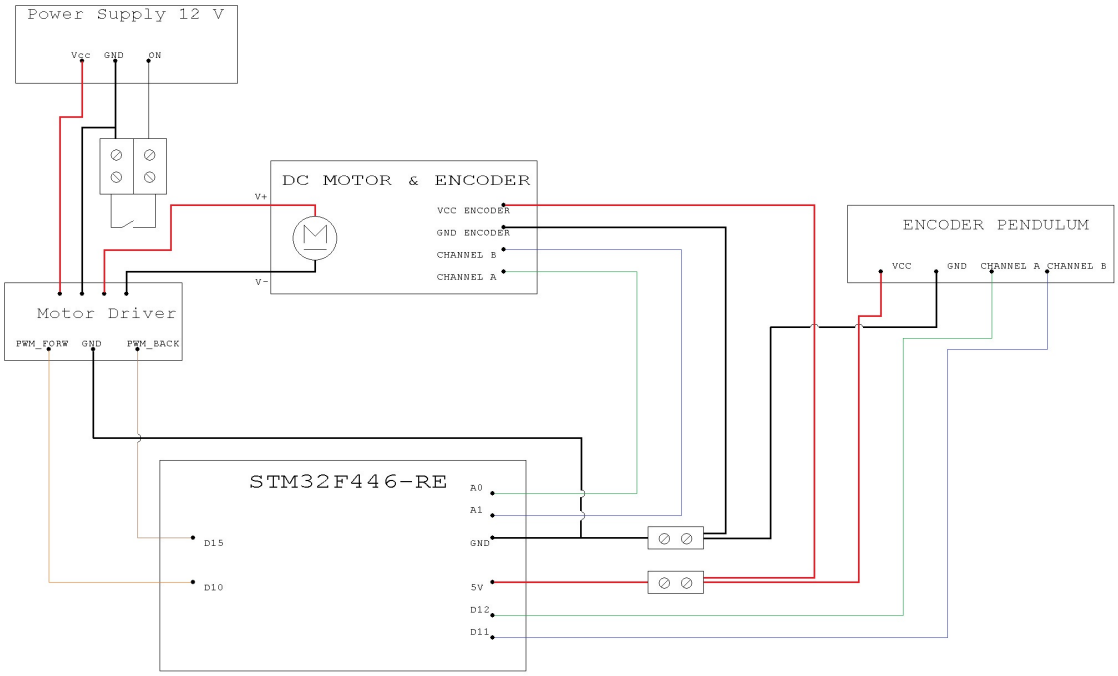


Figure 3.6.1: Circuit board for Furuta pendulum

As shown in Fig. 3.6.1, a 12V power supply has been used to supply, using a switch, the whole system. The motor driver receives two different PWM signals from TIMER pins of STM32F446RE set in PWM mode:

- the PWM signal that allows to rotate the shaft of the motor in counterclockwise direction is given by the pin D10;
- the PWM signal that allows to rotate the shaft of the motor in clockwise direction is given by the pin D15.

It's important to notice that the reference of PWM signals of motor driver is the GND given by the Nucleo board. Depending on the duty cycle of PWM input signals of the motor driver, the DC motor can be supplied with different level of voltages, according to the control action. Both rotary incremental encoders have two channels: Channel A (CHA), Channel B (CHB) and are connected to 5V and GND given by the Nucleo board.
Both sensors are quadrature encoders so CHA and CHB outputs are square waves from 0V to Vcc approximately 90°out of phase. The speed of the motor can be determined from their frequency, and the direction of rotation can be determined from the order of the transitions. PWM signals of two rotary encoders are the input signals of timer pins of Nucleo board, set in Encoder Mode. In this way, it's possible to compute the angular position and velocity of the DC motor and the pendulum.

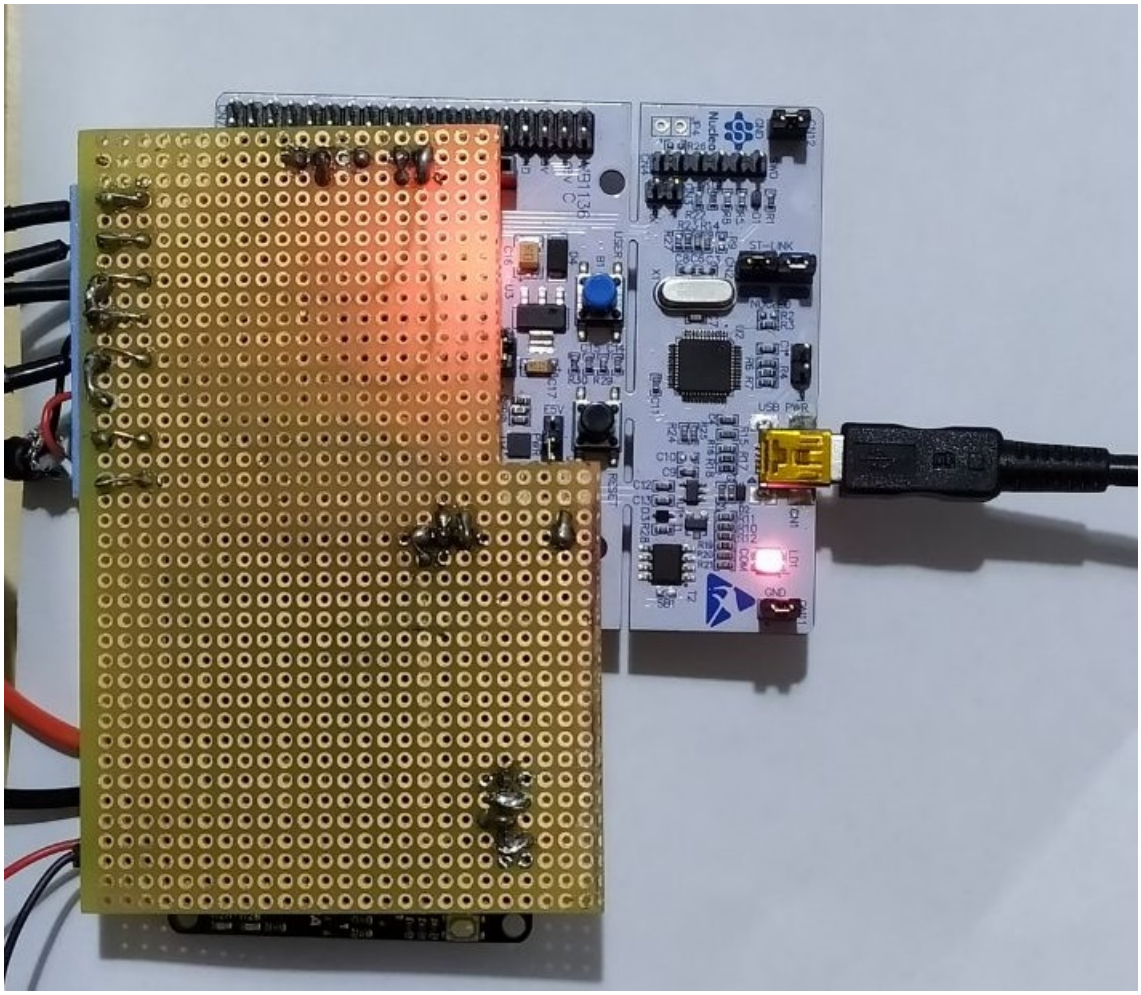


Figure 3.6.2: Shield that facilitates electrical connections

4 Firmware Design

In the following sections, the firmware, flashed on a Nucleo STM32F446RE board by STMicroelectronics, is shown. It has been developed using STMCubeIDE ver. 1.4.2.

4.1 High-level firmware overview

In Fig. 4.1.1 a flowchart has been shown that gives an high-level overview of the firmware design.

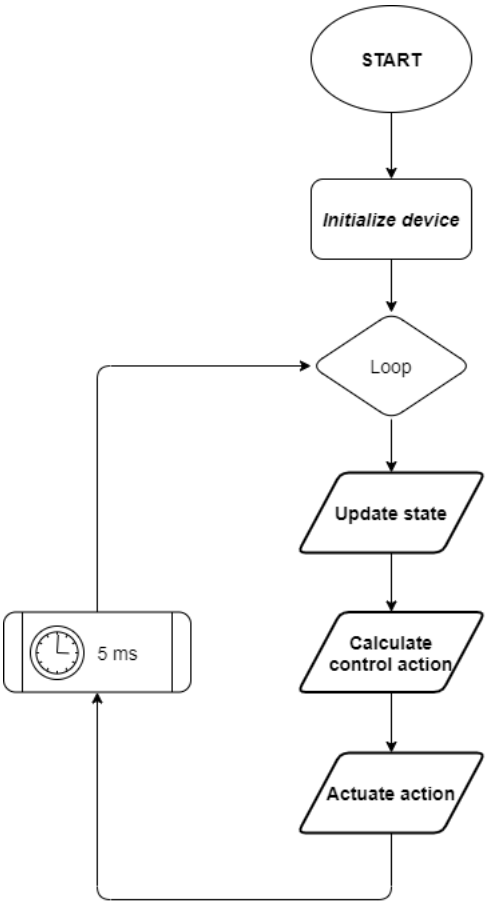


Figure 4.1.1: Firmware overview flowchart

At the beginning of the firmware, there is a section in which all devices are initialized.
In particular, there is a function called "start_detection" which allows to start reading from sensors and the time base, as shown in Fig. 4.1.2.

```
void start_detection()
{
    /* Start counting in X4 Encoder mode rising and falling edges of encoders' output signals */
    HAL_TIM_Encoder_Start(&htim2, TIM_CHANNEL_ALL);
    HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
    /* Time base start in IT mode */
    HAL_TIM_Base_Start_IT(&htim7);
    /* First count */
    count1_encoder_motor = __HAL_TIM_GET_COUNTER(&htim2);
    count1_encoder_pendulum = __HAL_TIM_GET_COUNTER(&htim3);
}
```

Figure 4.1.2: start_detection function

After that, a time-base generates an interrupt (IT) event every 5 ms and the function used to manage interrupt event is the Period Elapsed Callback. As shown in Fig. 4.1.3 in the Period Elapsed Callback are called the function showed in Fig. 4.1.1 where:

- **update_state**: get the current states of the system (α, $\dot{\alpha}$,$\dot{\theta}$);
- **calculate_control_action**: calculate the input torque;
- **actuate_control_action**: drives the motor with PWM output signals from MCU.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim -> Instance == TIM7)
        {
            update_state();
            float torque = calculate_control_action();
            actuate_control_action(torque);
        }
}
```

Figure 4.1.3: Period Elapsed Callback

4.2 Time-Base Generator

TIM7 is used in IT mode to generate an event every T_c. In the study case, $T_c = 5\ ms$, so the setting parameters of this timer are shown in Fig. 4.2.1.

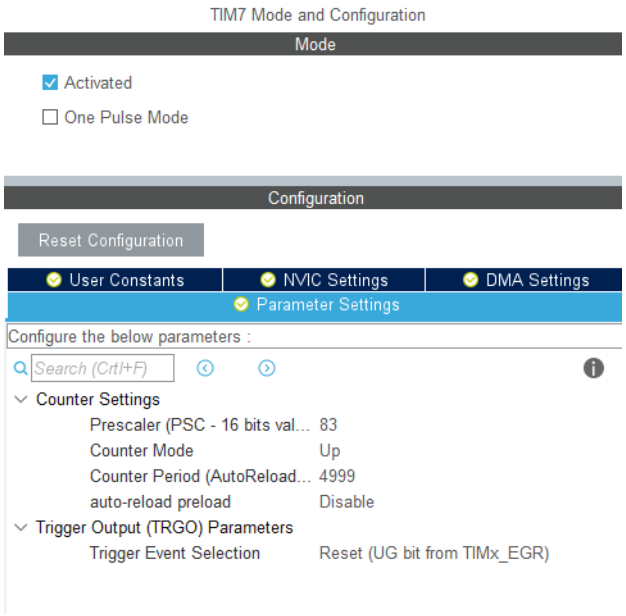


Figure 4.2.1: TIM7 Parameter settings

In Fig. 4.2.2 NVIC Mode and configuration window is shown. In this case, it has been set only one interrupt related to TIM7 with the highest preemption priority.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
TIM2 global interrupt	☐	0	0
TIM3 global interrupt	☐	0	0
TIM4 global interrupt	☐	0	0
USART2 global interrupt	☐	0	0
EXTI line[15:10] interrupts	☐	0	0
TIM7 global interrupt	☑	0	0
FPU global interrupt	☐	0	0

Figure 4.2.2: NVIC Mode and Configuration

4.3 Sensor reading

In order to update the value of the system's states, two general purpose timers (TIM2 & TIM3) have been used. The first one senses the data in output from the motor rotary encoder, the other one senses the data from the encoder connected to the pendulum. Both timers are used in Encoder Mode, so the Counter Register (CNT) increases by one when the rotational speed is positive and decreases by one otherwise, as it is shown in Fig. 4.3.1. Moreover, Direction Register (DIR) is used to get the direction of rotation.

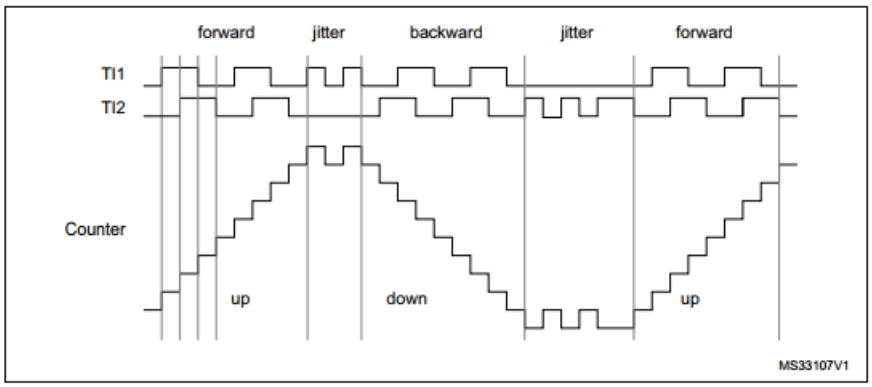


Figure 4.3.1: Encoder mode description [3]

4.3.1 Angular Position Reading

The angle α (rad) in output from a generic rotary encoder is given by:

$$\alpha = \frac{count \cdot 2\pi}{GR \cdot CPR} \tag{3}$$

where:

- count is the value of CNT Register;
- GR is the gear ratio of the motor;
- CPR is the count per rotation of the encoder.

The normalized angle in the domain $[-\pi, \pi]$ is given by:

$$\alpha = \text{atan2}(\sin(\alpha), \cos(\alpha)) \tag{4}$$

To avoid overflow the value of ARR Register must be multiple of CPR of the corresponding encoder. As shown in Fig. 4.3.2, the ARR Register is set as the maximum multiple of the Encoder CPR. Moreover, to use the Encoder mode in the right way PSC Register must be 0 [4].

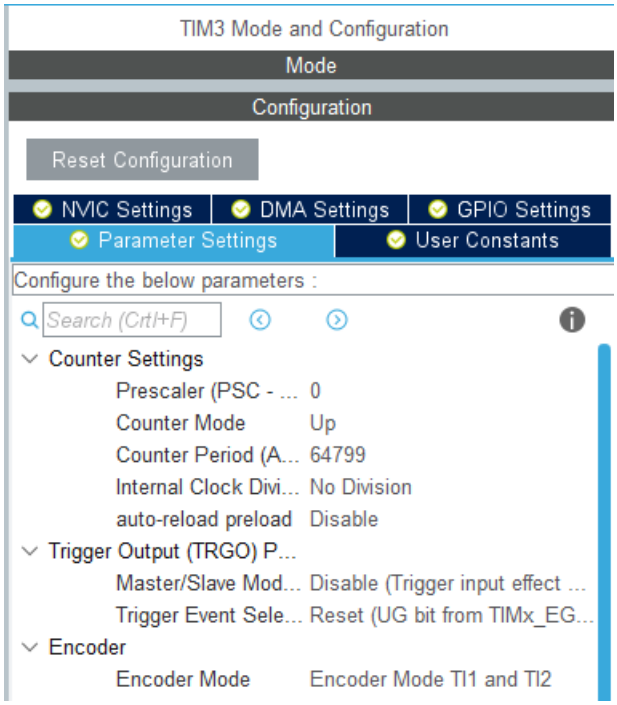


Figure 4.3.2: Parameter setting of TIM3 in Encoder mode

4.3.2 Angular Velocity Reading

Rotational speed w (rad/s) is given by:

$$w = \frac{(count2 - count1) \cdot 2\pi}{T_c \cdot GR \cdot CPR} \tag{5}$$

where:

- T_c is the sampling time;
- count2 and count1 are two consecutive value of CNT Register.

Moreover, as can be seen in Fig. 4.3.4, when the counter reaches the ARR it is reset.
For this reason, to avoid the counter overflow when the difference between count2 and count1 is computed, the function showed in Fig. 4.3.3 has been implemented.

```
int32_t difference_count(uint32_t cnt2,uint32_t cnt1,TIM_HandleTypeDef htim)
{
    /* Function used to avoid count overflow */
    int32_t difference = cnt2 - cnt1;
    if (__HAL_TIM_IS_TIM_COUNTING_DOWN(&htim) && (cnt2 > cnt1))
        {
            difference =  cnt2 - __HAL_TIM_GET_AUTORELOAD(&htim) - cnt1;
        }
    else if ((__HAL_TIM_IS_TIM_COUNTING_DOWN(&htim) == 0) && (cnt1 > cnt2))
    {
            difference = cnt1 +  (__HAL_TIM_GET_AUTORELOAD(&htim) - cnt2);
    }

    return difference;
}
```

Figure 4.3.3: Function implemented to avoid overflow when the difference is computed

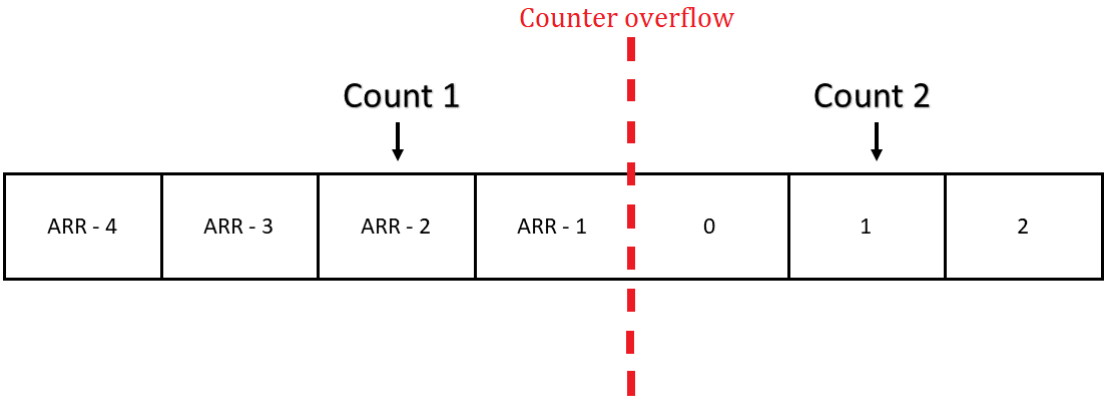


Figure 4.3.4: Counter overflow when encoder is counting forward

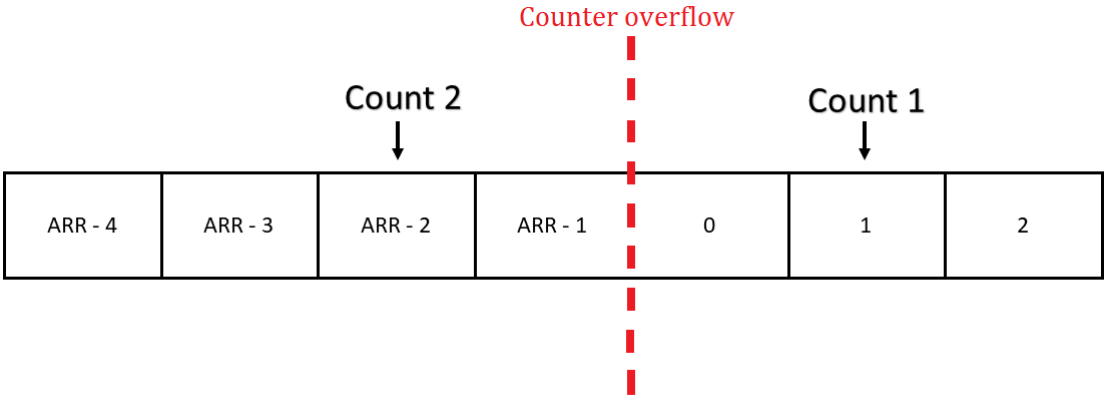


Figure 4.3.5: Counter overflow when encoder is counting backward

In Fig. 4.3.4 and 4.3.5 the overflow event when timer is counting up and down are shown respec-

tively.

4.4 Calculate control action

Due to the presence of three different controllers (as explained in 3.1), the function shown in Fig. 4.4.1 has been implemented to calculate the input torque in the three different situations.

```
float calculate_control_action()
{
    /*Function that switch between controllers*/

    /* Swing-up control action in ]35°;-35°[ */
    if ((pendulum_angle < - 0.61) || (pendulum_angle > 0.61))
    {
        integral_error = 0;
        return (swing_up());

    }

    /* Catch control action in ]10°;35] || [-35°;-10°[ */
    else if ((pendulum_angle < - 0.174 && pendulum_speed > 0) || (pendulum_angle > 0.174 && pendulum_speed < 0))
    {
        integral_error = 0;
        return (catch());

    }

    /* Balancing control action in [-10°;10°] */
    else if (pendulum_angle > - 0.174 || pendulum_angle < 0.174)
    {
        return (balancing_control());
    }
    else
    {
        return 0;
    }
}
```

Figure 4.4.1: Calculation of control action

Notice that the catch controller is fired only if the pendulum is rising whereas if the pendulum is falling down it's disabled.

4.4.1 Swing-up control

Fig. 4.4.2 shows the function implemented in order to apply a nonlinear proportional control when the pendulum angle is in range $[35°, -35°]$.

```
float swing_up()
{
    /*Function that implements the swing up non-linear controller*/
    float w0 = (float)sqrt((double)(mP*g*Lk)/Inertia);
    float E = ((mP*g*Lk)/2)*(((pow((double)(pendulum_speed/w0),2))) + cos(pendulum_angle) - 1);
    float torque_swing = Kv*(E-E0)*sign(pendulum_speed*cos(pendulum_angle));
    return torque_swing;
}
```

Figure 4.4.2: Implementation of Swing-Up control

4.4.2 Catch control

Fig. 4.4.3 shows the function that applies a catch when the pendulum angle is in range $[10°; 35°]$ or $[-35°; -10°]$. In order to reduce oscillations in output from swing-up, a catch control is used in order to produce a "dampening effect", and consist in a proportional-derivative controller, with an high derivative coefficient.

```
float catch()
{
    /*Function that implements the catch controller*/
    float proportional_error = -pendulum_angle;
    float derivative_error = -pendulum_speed;
    float P_term = Kp_catch * proportional_error;
    float D_term = Kd_catch * derivative_error;
    return (P_term + D_term);
}
```

Figure 4.4.3: Implementation of Catch control

4.4.3 Balancing control

The Fig. 4.4.4 shows how balancing control has been implemented using a cascade PID. This control is applied when the pendulum angle is in range $[-10°, 10°]$, as was explained in the previous section.

```
float balancing_control()
{
    /*Motor PID on motor_speed*/
    float proportional_motor_error = setpoint_m - motor_speed;
    motor_speed_2 = motor_speed;
    float derivative_motor_error = -(motor_speed_2 - motor_speed_1)/time_elapsed;
    motor_speed_1 = motor_speed_2;
    float setpoint_PD_out = Kp_o * proportional_motor_error + Kd_o*derivative_motor_error;

    /*Pendulum PID*/
    float proportional_error = setpoint_PD_out - pendulum_angle;
    float derivative_error = -pendulum_speed;
    integral_error += (proportional_error*time_elapsed);
    float P_term = Kp * proportional_error;
    float I_term = Ki * integral_error;
    float D_term = Kd * derivative_error;

    return((P_term + I_term + D_term));
}
```

Figure 4.4.4: Implementation of Cascade-PID Control

4.5 Actuate control action

The input torque (τ) is used to calculate the voltage (V_{out}) must be applied to the motor. In particular, from DC Motor law:

$$V_{out} = R_a \cdot \frac{\tau}{K_a} + K_a \cdot \omega$$

Known V_{out}, the duty cycle D is:

$$D = \frac{V_{out}}{V_m}$$

where V_m, K_a and R_a are given by Tab. 3.3.1.
As shown in Fig. 4.5.1, to set a correct duty cycle, the value of CCR Register must be:

$$CCR = D \cdot (1 + ARR)$$

The period of PWM signal is defined as:

$$PWM\_Period = \frac{(1 + ARR) \cdot (1 + PSC)}{f_{CLK}}$$

where $f_{CLK} = 84\ MHz$, according to the clock configuration in Fig. 4.6.1.

```c
void actuate_control_action(float torque)
{
    /* Calculate duty cycle from torque */
    float v_out = Ra*torque/Ka + Ka*motor_speed;
    float duty = v_out/Vm;
    uint16_t ccr = (uint16_t)(fabs(duty)*(float)(1+__HAL_TIM_GET_AUTORELOAD(&htim4)));
    /* Duty cycle saturation block */
    if (duty > 1)
    {
        duty = 1;
    }
    else if (duty < -1)
    {
        duty = -1;
    }
    /* Setting duty cycle to PWM Channels */
    if (duty > 0) {
        __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3, 0);
        __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,ccr);
    }
    else
    {
        __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3, ccr);
        __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,0);
    }
}
```

Figure 4.5.1: Actuate Control Action

In the case of study a frequency of 20 kHz has been chosen to reduce the motor current ripple and torque ripple. This is the maximum operating frequency of the driver. Fig. 4.5.2 shows ARR and PSC, obtaining 20 kHz PWM signals.

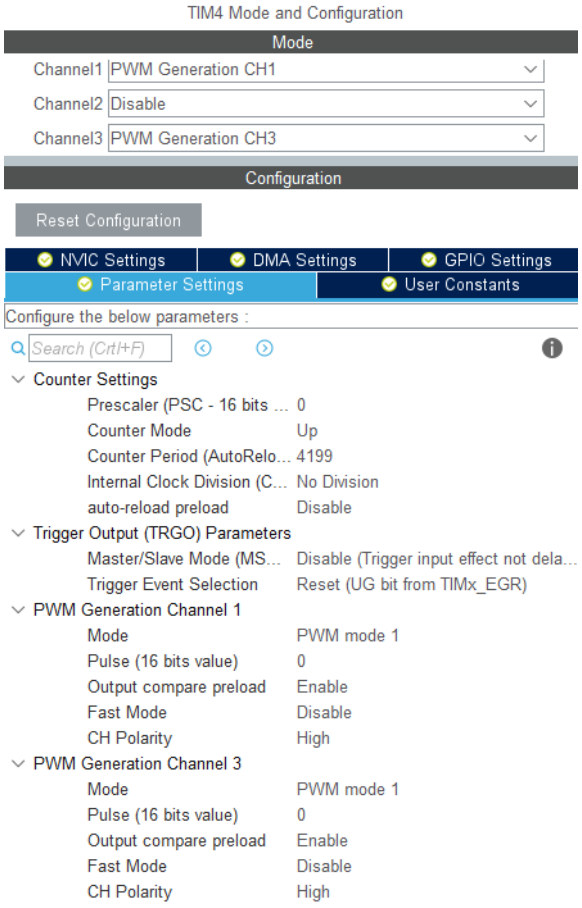


Figure 4.5.2: PWM mode.

4.6 Clock Configuration

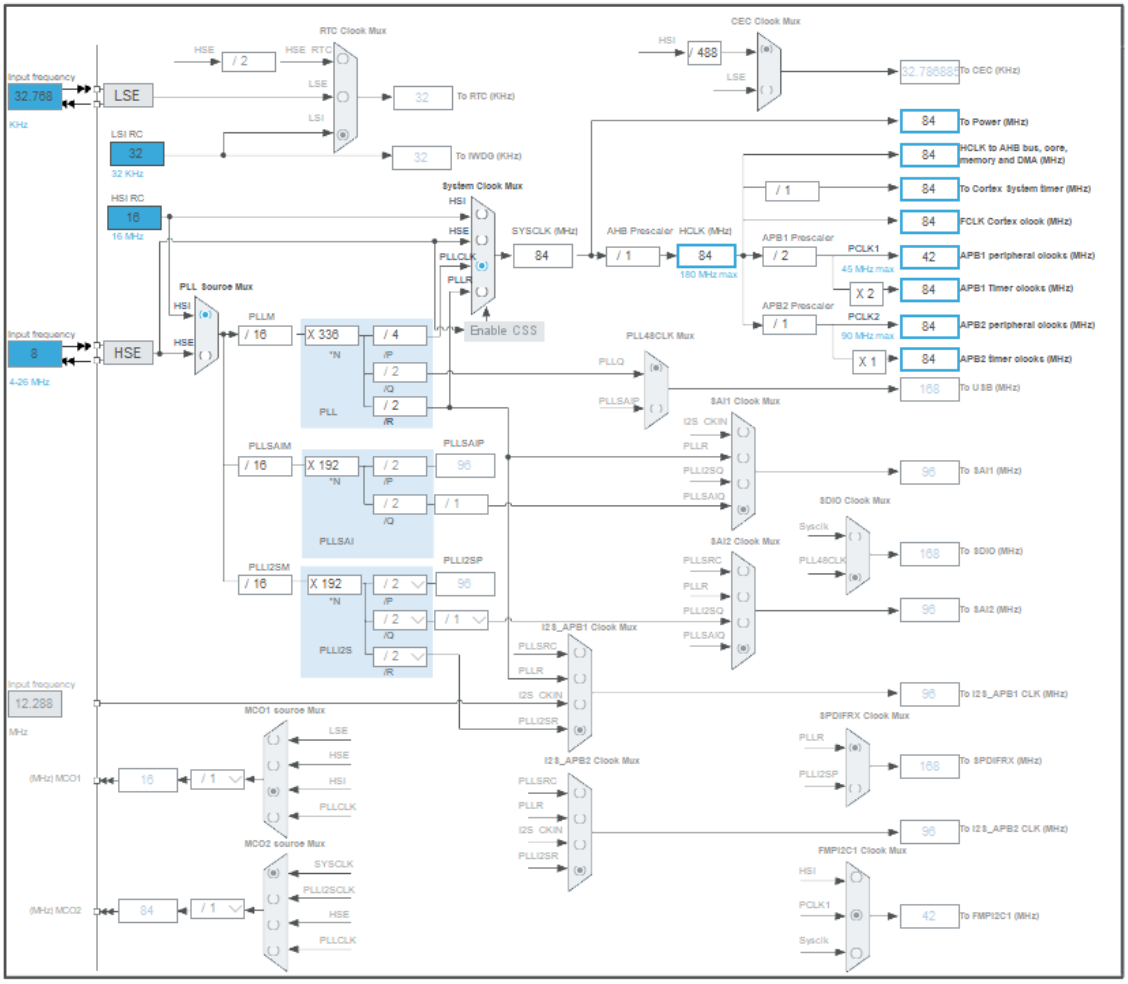


Figure 4.6.1: Clock Configuration

4.7 Pinout View

In Fig.4.7.1 has been shown the .ioc file of the firmware.

Timer	Function	Channel	I/O	Signal	Pin	Label
TIM3	Encoder Mode	Channel_1	I	PWM_forw pendulum	PA6	ChannelA_Pendulum
		Channel_2		PWM_back pendulum	PA7	ChannelB_Pendulum
TIM2	Encoder Mode	Channel_1	I	PWM_forw motor	PA0	ChannelA_Motor
		Channel_2		PWM_back motor	PA1	ChannelB_Motor
TIM4	PWM Mode	Channel_1	O	PWM_forw	PB6	PWM_ChannelForward
		Channel_3		PWM_back	PB8	PWM_ChannelBackward

Table 4.7.1: Table of pinout configuration

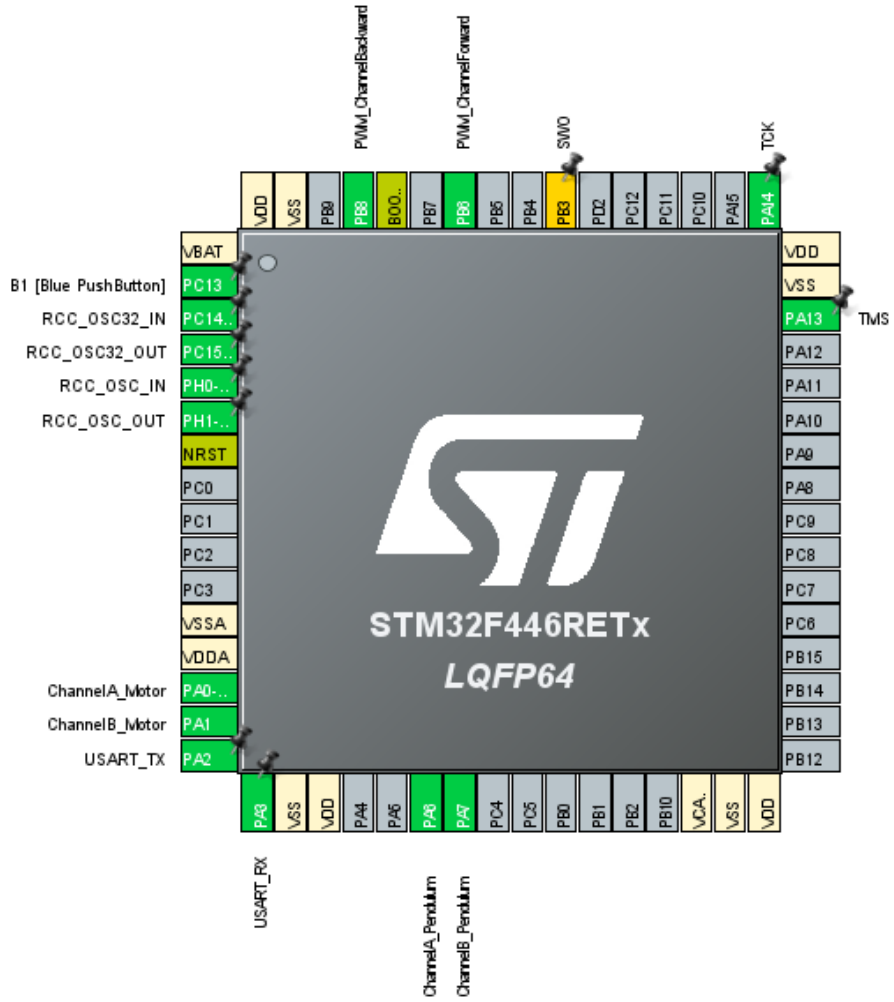


Figure 4.7.1: Pinout view in STMCUBEMX

5 Control System Tuning and Results

The swing-up controller has been the first one to be tuned. As discussed in Swing-Up Control, tuning the gain K_v allows the pendulum to reach the upward position. After several tests,

$$K_v = 28$$

has been chosen. Various tests on the test bed have shown that a catch controller would be necessary in order to slow down the pendulum near the set-point in the upward position. A Ziegler-Nichols tuning in the catching area results to be impractical, so a only-manual tuning has been done, according to the fact that setting an high value of K_d of the PD could increase the "dampening effect". At the end:

$$K_p = -11.682$$

$$K_d = -0.4$$

As mentioned in Balancing Control, Ziegler-Nichols tuning method has been used. For the PID in the inner loop, the critical gain K_c has been increased until the pendulum has shown sustainable oscillations, having:

$$K_c = -17.700$$

From DataTrace, the response of the system has been plotted (Fig.5.1), where the period of oscillations result in:

$$T_c = 0.166$$

Figure 5.1: System response with K_c

Having the value of K_c and T_c, a PID tuning using Ziegler-Nichols table can be done. After an extra manual tuning, the PID gains for the inner loop of the control system are:

$$K_p = -10.620$$

$$K_i = -90.382$$

$$K_d = -0.312$$

Thus, the angular position of the pendulum and the angular velocity of the motor have been monitored during various tests on the test bed. Results are shown in Fig.5.2.

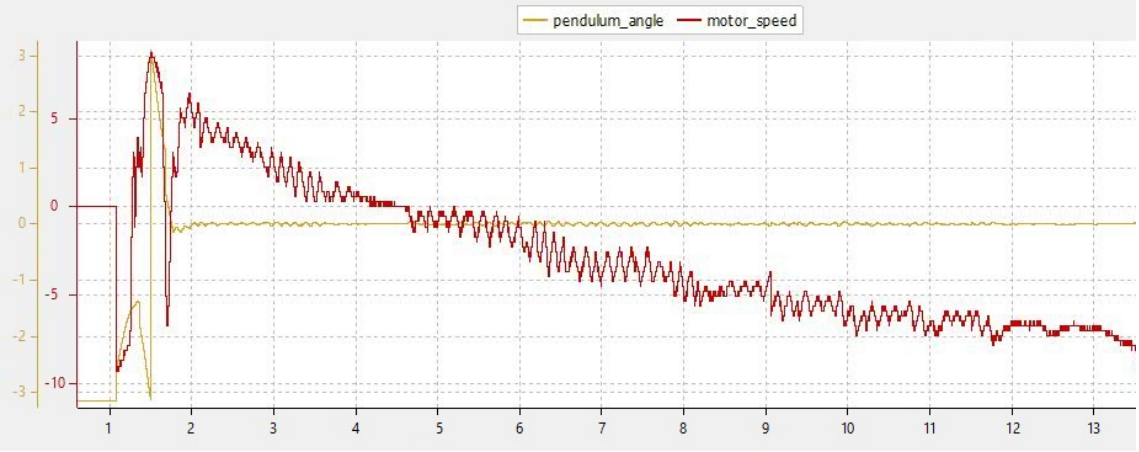


Figure 5.2: System response with PID

As can been seen, in the first instants, the value of the motor angular velocity changes in order to swing-up the pendulum until it reaches the unstable equilibrium point. Then, the pendulum keeps the set-point position while the motor angular velocity increase linearly. The reason of this behavior can be found in the absence, in this tests, of a controller that slow down the motor until stopping it.
Consequently, a PD controller in an outer loop has been added, where the set-point correspond to a null motor angular velocity, composing a cascade PID controller. In order to set the gains for the controller, a manual tuning has been done, resulting in:

$$K_p = -0.007$$

$$K_d = -0.000162$$

Notice that the PD gain values are very low because the control action of the outer loop has to be slower than the inner loop one in order to set an higher priority on stabilizing the pendulum first. Feasible result is shown in Fig.5.3. As can be seen, after the swing-up, the pendulum keeps its position around the desired set-point while the motor angular velocity slow down around 0 $[rad/s]$ as desired.

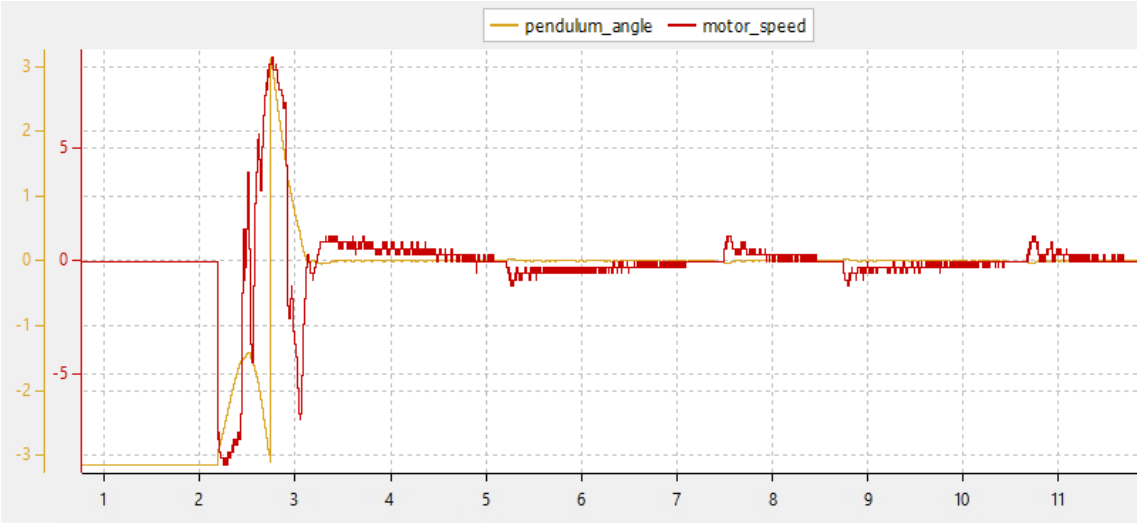


Figure 5.3: System response with cascade PID

6 Conclusions

The aim of this project was to create a working Furuta pendulum made by ourself, keeping an eye on the budget. As shown in the previous sections, the goal has been achieved, not very straightforward indeed. The main problems encountered were in choosing the most suitable control system. Despite of the efficiency of a control system based on LQR as shown in the software prototyping, the model identification results to be very inaccurate due to the presence of various parameters to be identified (like damping coefficients of the revolute joints ecc.), resulting in a very unstable system on the real test bed. For these reasons, the best solution consisted in the use of a cascade PID, combined with a swing-up and a catch controllers, easier and more intuitive to tune, improving the performance of the control system. Finding the controller's gains resulted to be not so easy due to the presence of vibrations of the gearbox motor shaft, unstable test bed support and non perfect mechanical connections, resulting sometimes in not ensuring constantly repeatability between tests.

"Get your hands dirty"

was one of the achieved goal of this project. Cut in pieces wooden rods, welding iron stuffs together make the job done. Realizing a circuit board that replaced a common breadboard resulted to be very useful in order to avoid connection errors. Another important skill improved is the team work: working together means talk about a problem, finding a solution together and scheduling different tasks in an efficient way in order to reduce the time cost of the project.

"Get your code working"

was another achieved goal of this project. The acknowledge of using timers, interrupts event, good casting of variables, the use of pointers has been developed during the realization of the Furuta Pendulum firmware. An important tool used to realize a working firmware surely is the debug mode with which is possible to find errors, bugs and so avoid them.
Future works can be focused on building a more solid structure, improving the mechanical connection between parts of the system and setting it up on a professional test bed support in a laboratory. With a model identification, the PID controller can be replaced by LQR controller. Increasing the budget, less bulky and higher quality encoders can be bought, improving reading resolution and suppress the noise, as well as replacing the actuator with a brushless motor, improving the control action (but increasing the control system architecture complexity).
The best experience that we will remember is the application of the theory, studied in the last years of automation course, to practice.

37D Metal Gearmotors

Pololu 37D Metal Gearmotors are powerful brushed DC motors paired with 37mm-diameter gearboxes. There are nine different gearbox options available, ranging from 6.3:1 to 150:1, and two different motor options: 12 V and 24 V. The 24 V versions offer approximately the same speed and torque at 24 V as their 12 V counterparts do at 12 V, with approximately half the current draw. This datasheet includes two sets of performance graphs for each version, one at its nominal voltage and one at half of its nominal voltage. Each version is available with an integrated 64 CPR quadrature encoder on the motor shaft.

Note: The original versions of these gearmotors had gearboxes with all spur gears. In August 2019, these were replaced by functionally identical “Helical Pinion” versions that feature helical gears for the first stage of the gearbox, which reduces noise and vibration and improves efficiency. The picture on the right shows the helical pinion gear and first mating gear.

Performance summary and table of contents

Rated Voltage	Pololu Item #	Gear Ratio	No Load		At Maximum Efficiency				Max Power	Stall Extrapolation[2]		Graph Pages
			Speed	Current	Speed	Torque	Current	Output		Torque	Current	
		:1	RPM	A	RPM	kg·mm	A	W	W	kg·mm	A	
12 V	4750 [1]	1	10,000	0.2						5	5.5	
	4747, 4757	6.25	1600		1300	4.9	1.2	6.4	12	30		5, 6
	4748, 4758	10	1000		850	6.6	0.91	5.7	12	49		7, 8
	4741,4751	18.75	530		470	10	0.76	5.0	12	85		9, 10
	4742, 4752	30	330		280	18	0.78	5.1	12	140		11, 12
	4743, 4753	50	200		180	22	0.66	4.0	10	210		13, 14
	4744, 4754	70	150		130	32	0.68	4.2	10 [3]	270		15, 16
	4745, 4755	102.08	100		87	42	0.72	3.8	8 [3]	340		17, 18
	4746, 4756	131.25	76		66	60	0.74	4.1	6 [3]	450		19, 20
	2828, 2829	150	67		58	65	0.72	3.8	6 [3]	490		21, 22
24 V	4690 [1]	1	10,000	0.1						5.5	3.0	
	4688, 4698	6.25	1600		1300	5.5	0.58	7.4	14	35		23, 24
	4689, 4699	10	1000		850	7.5	0.49	6.6	14	55		25, 26
	4681,4691	18.75	530		450	13	0.49	6.1	13	95		27, 28
	4682, 4692	30	330		280	19	0.46	5.5	13	150		29, 30
	4683, 4693	50	200		170	27	0.41	4.9	12	230		31, 32
	4684, 4694	70	140		120	39	0.42	5.0	10 [3]	310		33, 34
	4685, 4695	102.08	100		86	51	0.42	4.5	8 [3]	390		35, 36
	4686, 4696	131.25	79		68	63	0.40	4.4	6 [3]	470		37, 38
	4687, 4697	150	68		59	73	0.41	4.4	6 [3]	560		39, 40

Notes:

(1) Max efficiency data and performance graphs currently unavailable for the motors without gearboxes (items #4750 and #4690).

(2) Listed stall torques and currents are theoretical extrapolations; units will typically stall well before these points as the motors heat up. Stalling or overloading gearmotors can greatly decrease their lifetimes and even result in immediate damage. The recommended upper limit for continuously applied loads is 100 kg·mm, and the recommended upper limit for instantaneous torque is 250 kg·mm. Stalls can also result in rapid (potentially on the order of seconds) thermal damage to the motor windings and brushes; a general recommendation for brushed DC motor operation is 25% or less of the stall current.

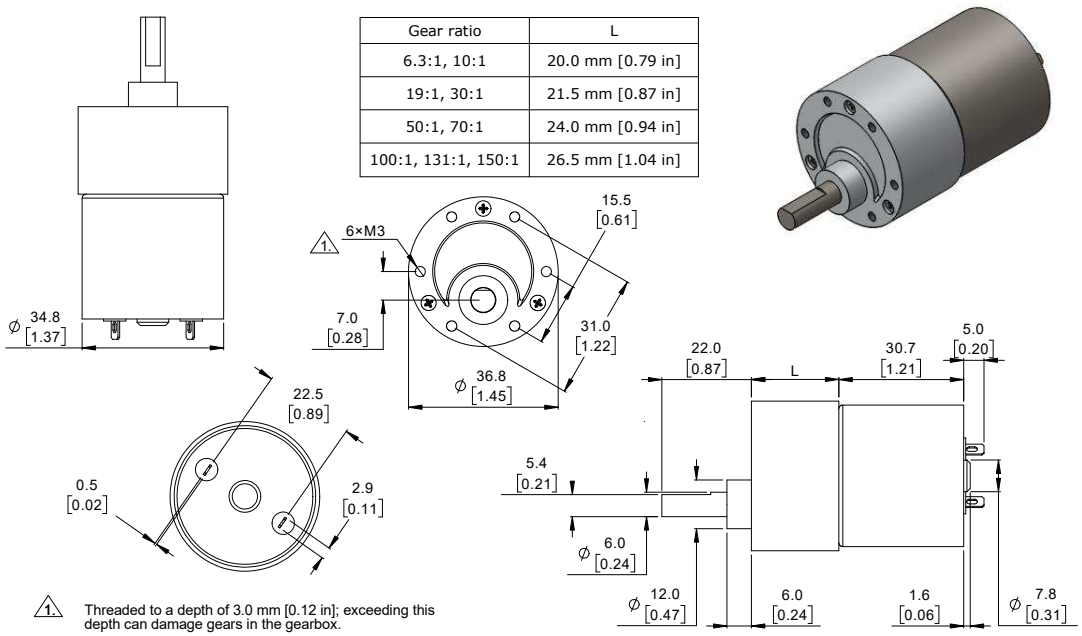
(3) Output power for these units is constrained by gearbox load limits; spec provided is output power at max recommended load of 100 kg·mm.

37D Metal Gearmotors

Dimensions (units: mm over [inches])

Gearmotor versions without encoders (items #2829, 4681–4689, 4741–4748) weight: 175 g to 195 g

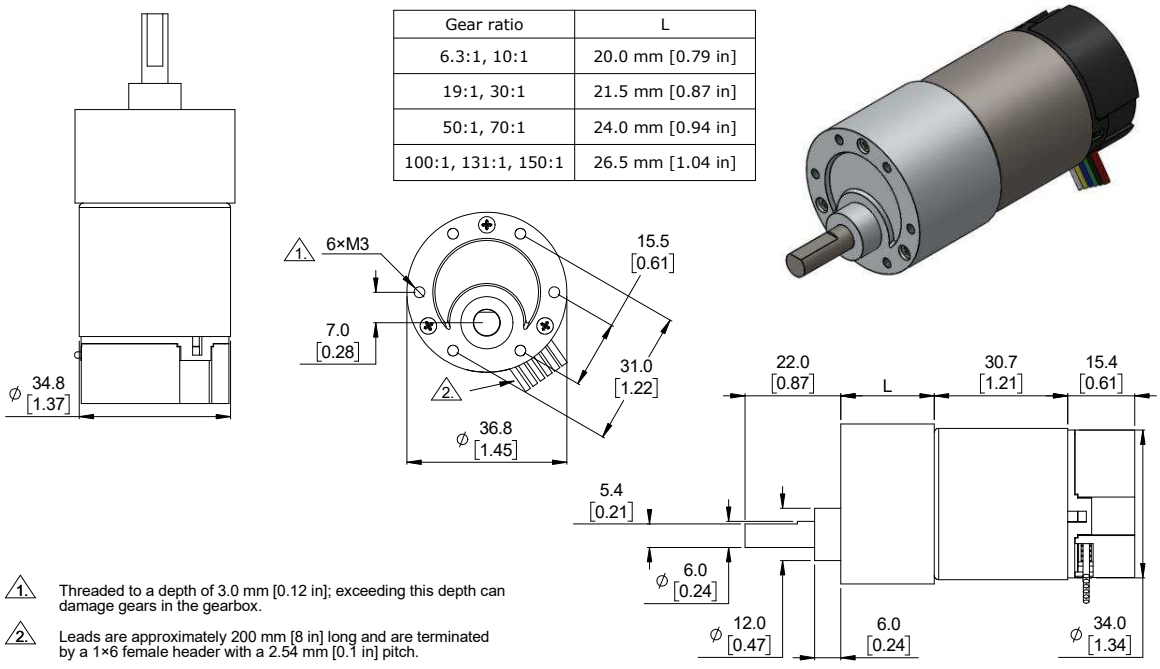
Gear ratio	L
6.3:1, 10:1	20.0 mm [0.79 in]
19:1, 30:1	21.5 mm [0.87 in]
50:1, 70:1	24.0 mm [0.94 in]
100:1, 131:1, 150:1	26.5 mm [1.04 in]



1. Threaded to a depth of 3.0 mm [0.12 in]; exceeding this depth can damage gears in the gearbox.

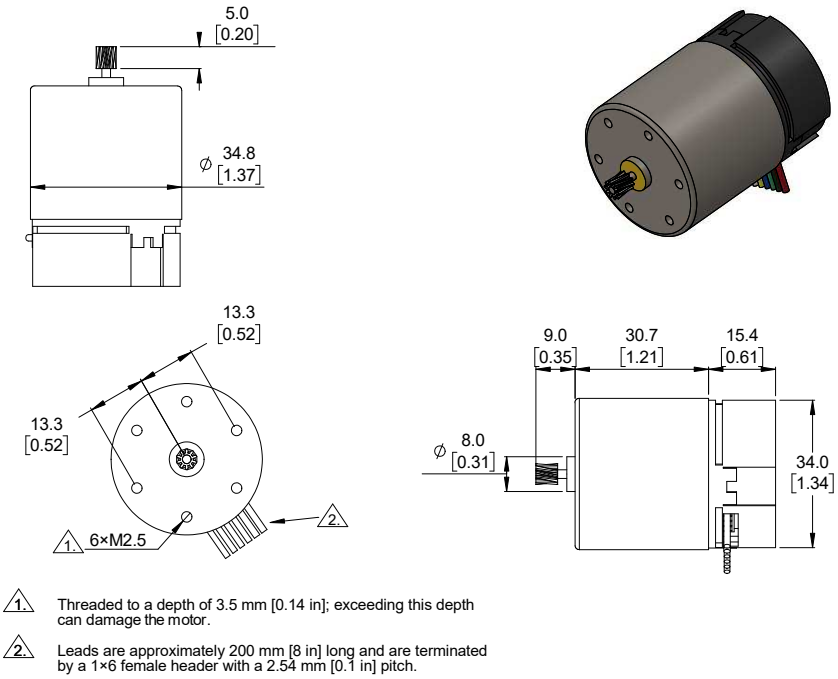
Gearmotor versions with encoders (items #2828, 4691–4699, 4751–4758) weight: 190 g to 210 g

Gear ratio	L
6.3:1, 10:1	20.0 mm [0.79 in]
19:1, 30:1	21.5 mm [0.87 in]
50:1, 70:1	24.0 mm [0.94 in]
100:1, 131:1, 150:1	26.5 mm [1.04 in]



1. Threaded to a depth of 3.0 mm [0.12 in]; exceeding this depth can damage gears in the gearbox.
2. Leads are approximately 200 mm [8 in] long and are terminated by a 1×6 female header with a 2.54 mm [0.1 in] pitch.

Motor with encoder and no gearbox (items #4690, 4750) weight: 110 g



1. Threaded to a depth of 3.5 mm [0.14 in]; exceeding this depth can damage the motor.
2. Leads are approximately 200 mm [8 in] long and are terminated by a 1×6 female header with a 2.54 mm [0.1 in] pitch.

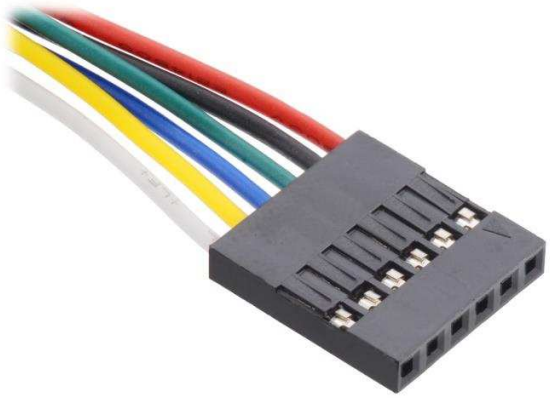
Using the encoder

Versions with encoders have additional electronics mounted on the rear of the motor. Two Hall-effect sensors are used to sense the rotation of a magnetic disc on a rear protrusion of the motor shaft. The encoder electronics and magnetic disc are enclosed by a removable plastic end cap. The following pictures show what the encoder portion looks like with the end cap removed:

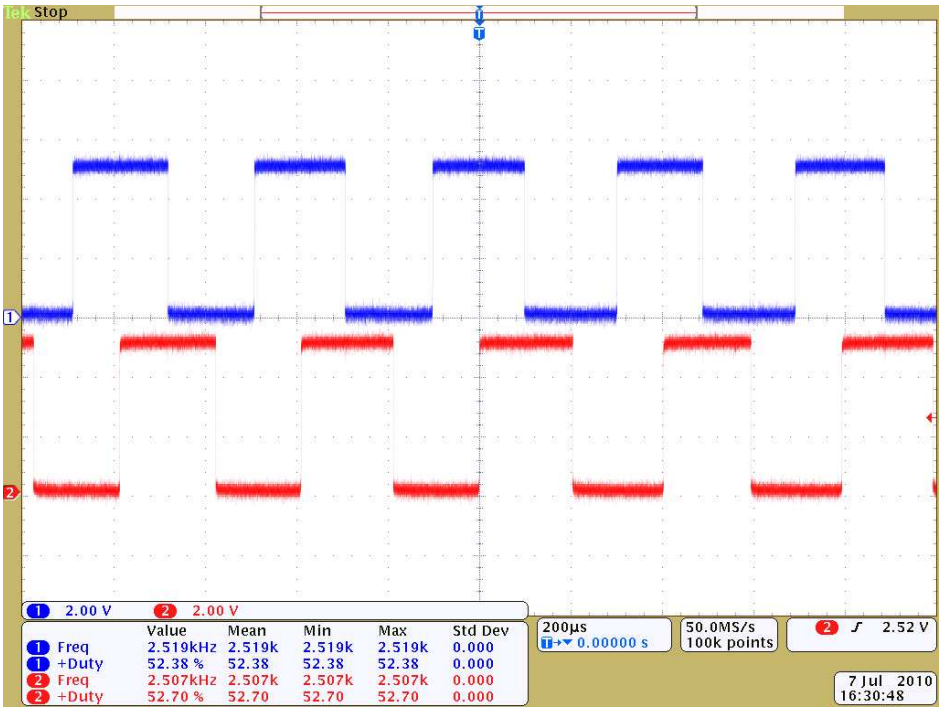
The quadrature encoder provides a resolution of 64 counts per revolution (CPR) of the motor shaft when counting both edges of both channels. To compute the counts per revolution of the gearbox output, multiply the gear ratio by 64.

The motor/encoder has six color-coded, 20 cm (8″) leads terminated by a 1×6 female connector with a 2.54 mm (0.1″) pitch. This connector works with standard 0.1″ male breakaway headers and Pololu male premium jumper and precrimped wires. If this header is not convenient, the crimped wires can be pulled out of the 1×6 housing and used with different crimp connector housings instead (e.g. 1×2 for the motor power and 1×1 housings for the other four leads), or the connectors can be cut off entirely.

Lead Color	Function
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20 V)
Yellow	Encoder A output
White	Encoder B output

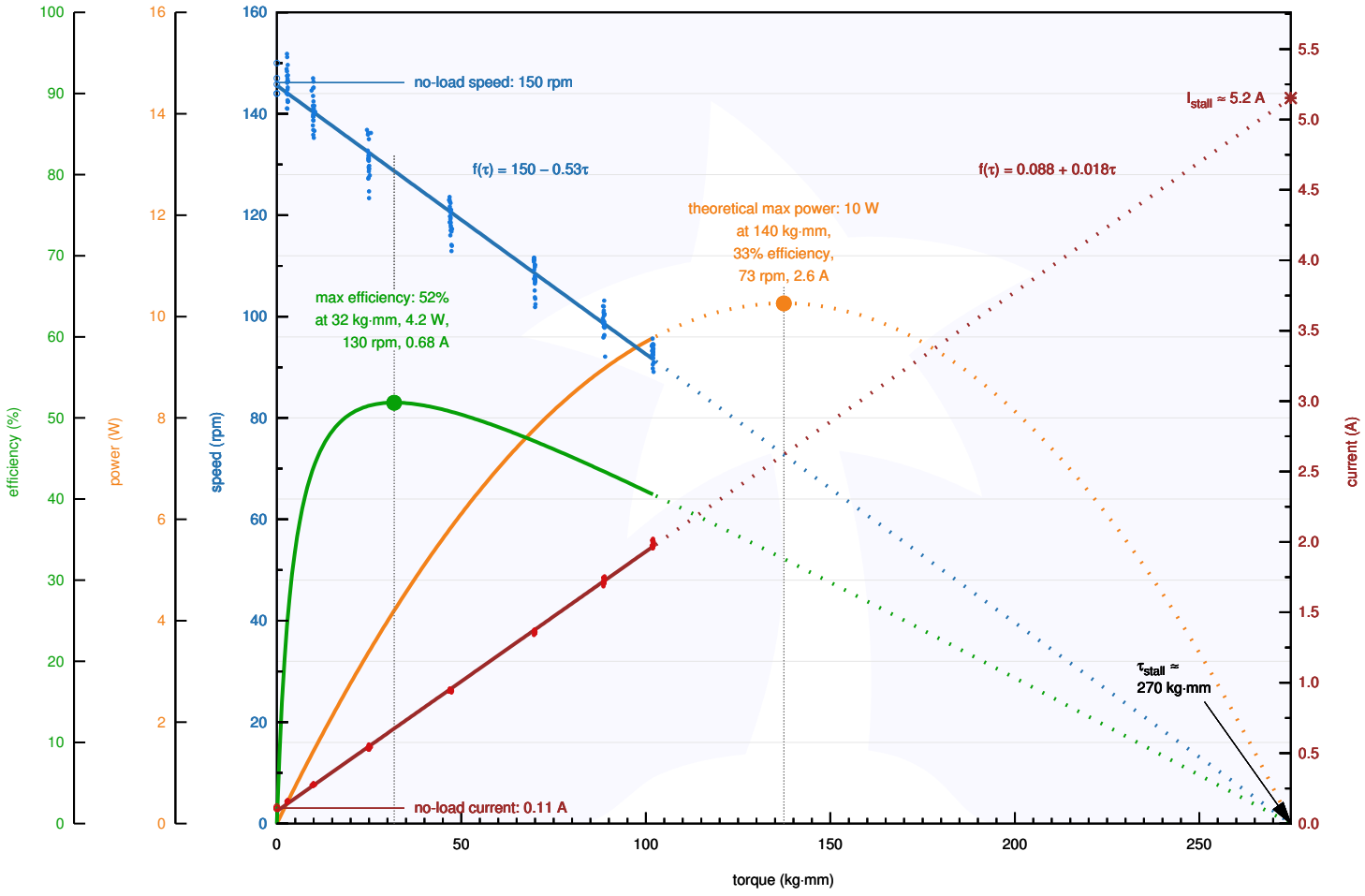


The Hall sensors require an input voltage, Vcc, between 3.5 V and 20 V and draw a maximum of 10 mA. The A and B outputs are square waves from 0 V to Vcc approximately 90° out of phase. The speed of the motor can be determined from the frequency, and the direction of rotation can be determined from the order of the transitions. The following oscilloscope capture shows the A and B (yellow and white) encoder outputs using a 12 V motor at 12 V and a Hall sensor Vcc of 5 V:



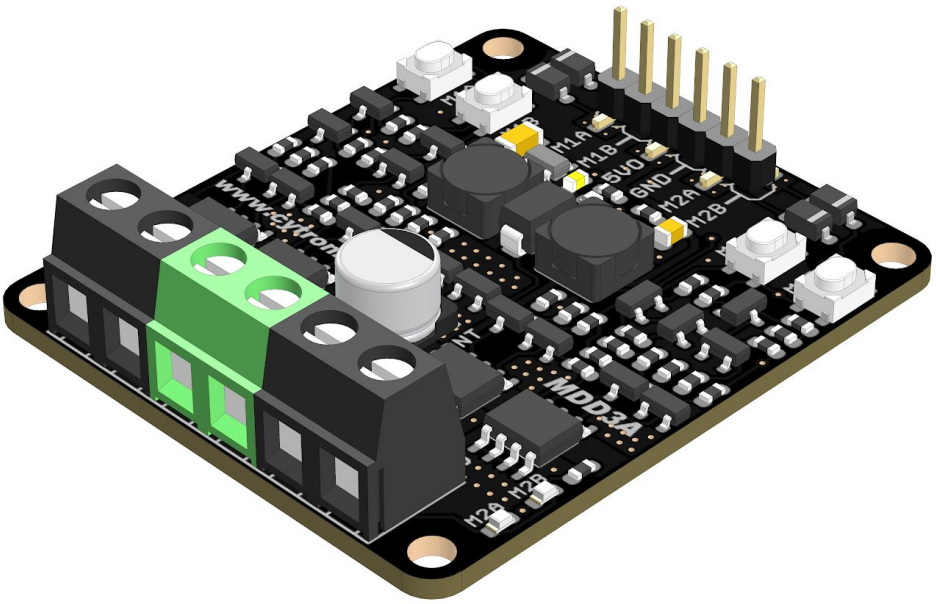
Counting both the rising and falling edges of both the A and B outputs results in 64 counts per revolution of the motor shaft. Using just a single edge of one channel results in 16 counts per revolution of the motor shaft, so the frequency of the A output in the above oscilloscope capture is 16 times the motor rotation frequency.

Pololu Items #4744, #4754 (70:1 Metal Gearmotor 37D 12V) Performance at 12 V



MDD3A

3Amp 4V-16V DC Motor Driver (2 Channels)



Datasheet

Rev 1.0
March 2019

1. BOARD LAYOUT & FUNCTION

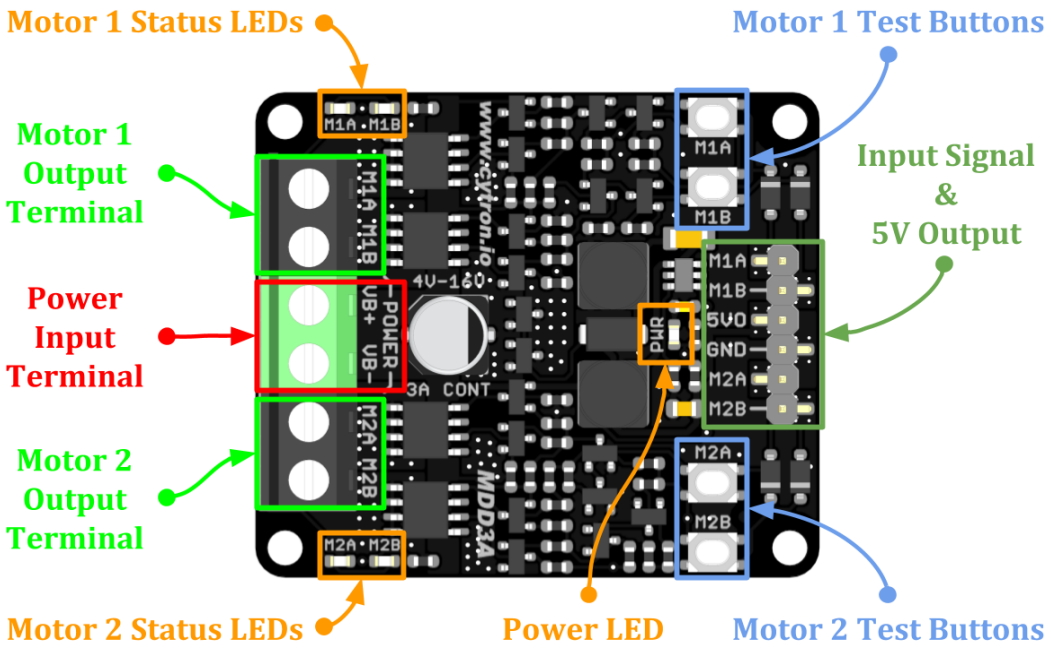


Figure 1: MDD3A Board Functions

Function	Description
Power Input Terminal	Connect to battery. ● VB+ : Positive ● VB- : Negative
Motor Output Terminal	Connect to motor terminal. Motor direction depends on the polarity.
Power LED	Turn on when power up.
Motor Status LEDs	Turn on when the motor is running. ● M1A / M2A : Forward* ● M1B / M2B : Backward*
Test Buttons	Press to test the functionality of the motor driver. Motor will run at full speed. ● M1A / M2A : Forward* ● M1B / M2B : Backward*
Input Signal & 5V Output	Input signal from microcontroller to control the motor. +5V output can be used to power the microcontroller. ● M1A : PWM Input A for motor 1. ● M1B : PWM Input B for motor 1. ● 5VO : DC +5V Output (Maximum 200mA) ● GND : Ground ● M2A : PWM Input A for motor 2. ● M2B : PWM Input B for motor 2.

Table 1: MDD3A Board Functions

* *Actual motor direction is depending on the motor connection.*
Swapping the connection (MA & MB) will reverse the direction.

2. SPECIFICATIONS

No	Parameters		Min	Max	Unit
1	Power Input Voltage (Vin)		4	16	VDC
2	Maximum Motor Current	Continuous	-	3	A
		Peak (< 5 seconds)	-	5	A
3	Logic Input Voltage (M1A, M1B, M2A, M2B)	Low Level	0	0.5	V
		High Level	1.7	12	V
4	PWM Frequency *(Output frequency is same as input frequency)*		DC	20	KHz
5	DC +5V Output Maximum Current		-	200	mA

Table 2: MDD3A Absolute Maximum Ratings

3. DIMENSION

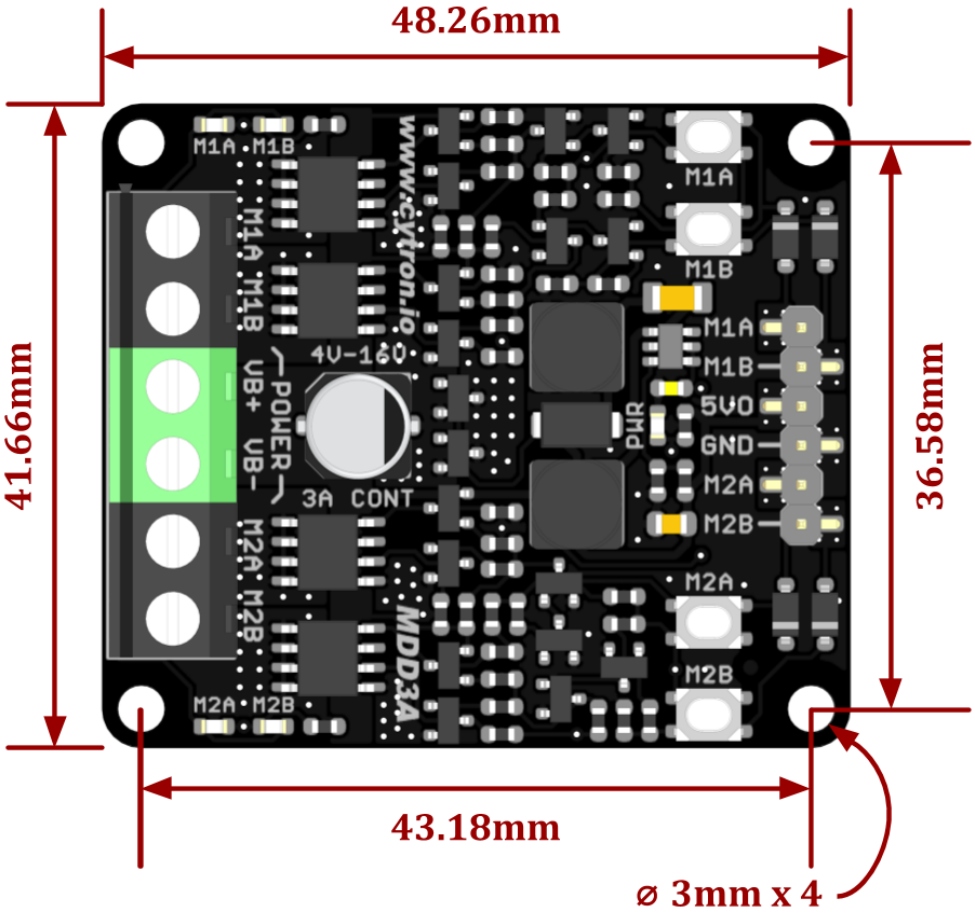


Figure 2: MDD3A Dimension

4. INTERFACE

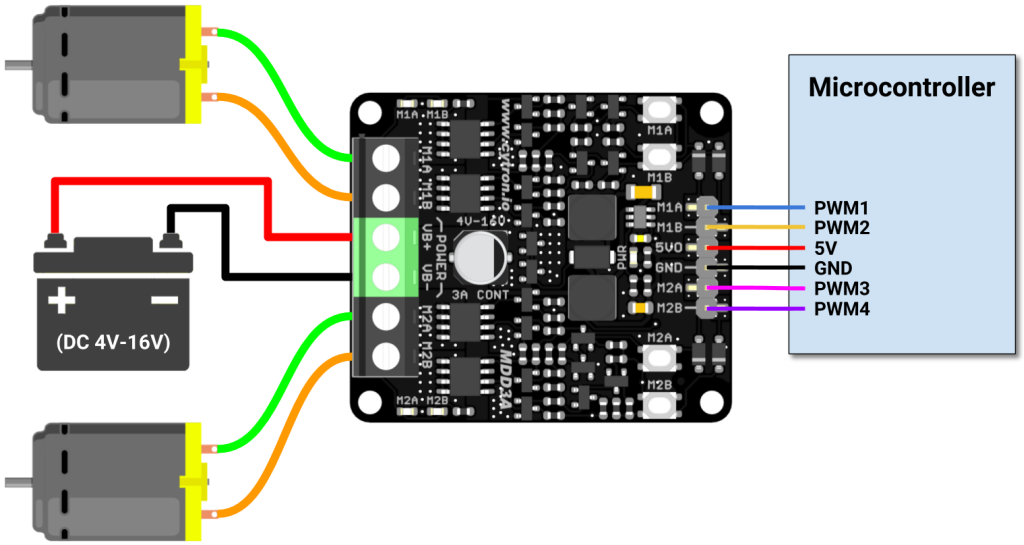


Figure 3: Connection Diagram for Brushed DC Motor

Input A (M1A / M2A)	Input B (M1B / M2B)	Output A (M1A / M2A)	Output B (M1B / M2B)	Motor
Low	**Low**	Low	Low	Brake
High	**Low**	High	Low	Forward*
Low	**High**	Low	High	Backward*
High	**High**	High	High	Brake

Table 3: Input Truth Table

* *Actual motor direction is depending on the motor connection.*
Swapping the connection (MA & MB) will reverse the direction.

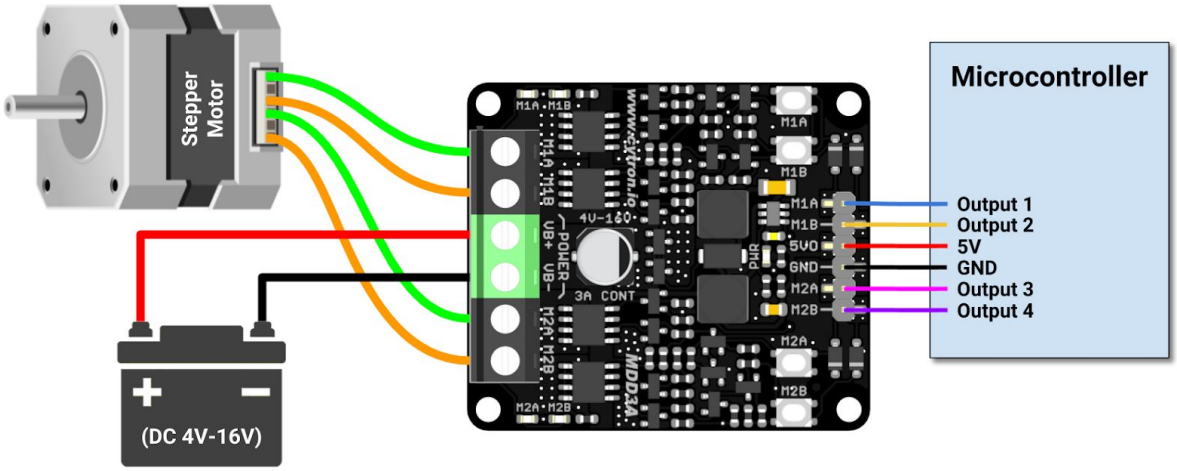


Figure 4: Connection Diagram for Stepper Motor

Appendix

LQR Controller

```
clear all
clc

%% Sample time
Tc = 0.005;
load('furuta_pendulum_Data.mat');
run('FURUTA_PENDULUM_DataFile')
%% Linearized model
sysD = c2d(linsys1,Tc,'zoh');
A = sysD.A;
B = sysD.B;
C = sysD.C;
D = sysD.D;
%% LQR stabilization algorithm
Q = C'*C;
R = 1;
[K,¬,¬] = dlqr(A,B,Q,R);
%% DC motor parameters
Kt = 0.48;
Ke = Kt;
Ra = 22.1896;
La = 0.044;
J = 0.0963;
```

Polulu DC motor 70:1 parameters and plots

```
%% Polulu DC motor 70:1 parameters and plots
clear all
clc
% We will use this number of bins for plotting and calculating
% all functions such as Torque, speed etc.
discreteBins = 500;
% Input part of the main function
StallTorque = 2.7; % [N*m]
StallCurrent = 5.5; % [A]
RatedVoltage = 12; % [V]
NoLoadCurrent = 0.2; % [A]
NoLoadSpeed = 150; % [RPM]

% Compute resistance of DC motor considered
Resistance = RatedVoltage / StallCurrent;

% Torque line
TorqueLine = 0:(StallTorque/discreteBins):StallTorque;

% Current Line
CurrentLine = NoLoadCurrent:(StallCurrent-NoLoadCurrent)/...
                            discreteBins:StallCurrent;

% Speed Line
SpeedLine = NoLoadSpeed: (0-NoLoadSpeed)/discreteBins : 0;

% Torque Constant in Torque per current
SlopeOfTorqueVsCurrent = (StallCurrent - NoLoadCurrent) / (StallTorque);


% Output Mechanical Power in watts is Torque (N*m) * Speed (RPM) *
% 0.10472 (rad/(s*RPM))
OutputPower = TorqueLine .* SpeedLine * 0.10472;
% Input Electrical Power to the motor is Voltage * Current
InputPower = CurrentLine * RatedVoltage;

% Plot part of the functions
subplot(2,2,1)
% This is the TorqueLoad vs. Motor Speed graph
[hAx, hLine1, hLine2] = plotyy([0 StallTorque], [NoLoadSpeed 0],...
    [0 StallTorque], [NoLoadCurrent StallCurrent]);

title('Torque vs. Speed & Torque vs. Current');
xlabel('Torque (N*m)');
ylabel(hAx(1), 'Speed-RPM');
ylabel(hAx(2), 'Current-A');

% Plot of the Output Mechanical power in Watts vs. Input
% Electrical power in Watts.
subplot(2,2,2);

[h2Ax, h2Line1, h2Line2] = plotyy(TorqueLine, OutputPower,...
                                  TorqueLine, InputPower);

xlabel('Torque (N*m)');
ylabel(h2Ax(1), 'OutputPower-Watts');
ylabel(h2Ax(2), 'InputPower-Watts');
title('Torque vs. Output Power & Torque vs. Input Power');

% Plot of the Power Efficiency of the motor.
subplot(2,2,3);
```

```
62  PowerEff = OutputPower ./ InputPower;
63  plot(TorqueLine, PowerEff);
64  xlabel('Torque (N*m)');
65  ylabel('Power Efficiency');
66
67  % Output information part of the function
68  fprintf('\n\n\nSlope of TorqueVsCurrent is %f. The recprocal is %f\n',...
69                      SlopeOfTorqueVsCurrent, (1/SlopeOfTorqueVsCurrent));
70
71  %Max Output Power is at
72  [V,I] = max(OutputPower);
73  fprintf('Maximum output mechanical power is %f(Watts).\nThis happens at the ...
        Torque load of %f(N*m), with Current %f(A)\n', OutputPower(I), TorqueLine(I), ...
        CurrentLine(I));
74  fprintf('Resistance of the motor is %f (ohm)\n', Resistance);
```

Slope of TorqueVsCurrent is 1.962963. The recprocal is 0.509434
Maximum output mechanical power is 10.602900(Watts).
This happens at the Torque load of 1.350000(N*m), with Current 2.850000(A)
Resistance of the motor is 2.181818 (ohm)

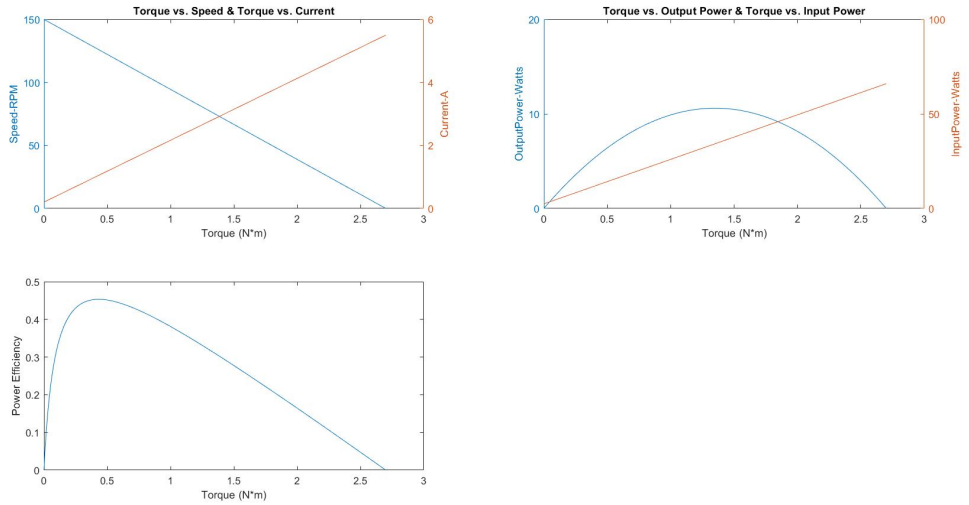


Figure 7.1: Graphs of DC Motor

References

[1] Pavol Seman, Martin Juh, Michal Salaj, et al. "Swinging up the Furuta pendulum and its stabilization via model predictive control". In: *journal of Electrical Engineering* 64.3 (2013), pp. 152–158.

[2] Fuat Peker et al. "Cascade Control Approach for a Cart Inverted Pendulum System Using Controller Synthesis Method". In: *2018 26th Mediterranean Conference on Control and Automation (MED)*. IEEE. 2018, pp. 1–6.

[3] STMicroelectronics. *RM0390 Reference manual*. English. Version Rev 4. 2018. 1328 pp.

[4] STMicroelectronics. *STM32F4 Encoder count is changing when it should not*. Sept. 2016. URL: `https://stackoverflow.com/questions/39450610/stm32f4-encoder-count-is-changing-when-it-should-not` (visited on).

List of Figures

List of Tables