Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Bachelor Thesis

Control Implementation for a Rotary Inverted Pendulum

Supervised and examined by:
Prof. Dr.-Ing. Heide Brandtstädter
Dipl.-Ing. Tobias Lipke

Author:
Michael Arvin
568686

Submitted on:
August 30, 2022

Fachbereich 1 - Energie und Information
Elektrotechnik
HTW Berlin

Acknowledgements

I would like to take this opportunity to thank all the people who have supported me in the processing and preparation of this thesis.

First and foremost, I would like to thank Prof. Dr.-Ing. Heide Brandtstädter and Dipl.-Ing. Tobias Lipke for their great help in formulating and structuring the thesis.

My special thanks go to my parents Teddy Sadeli and Sandy Aurellia Tjhan for their emotional support throughout my studies, especially during the entire process of writing this thesis. I would also like to thank my siblings Albert Nicholas and Tania Ardelia.

Contents

List of Figures iv

List of Tables v

List of Listings vi

Nomenclature vii

1 Introduction 1

2 Hardware 2
2.1 Microcontroller . . . 2
2.1.1 Timer . . . 2
2.1.2 Serial Communication . . . 3
2.2 BLDC Motor . . . 4
2.2.1 SimpleFOCShield . . . 5
2.3 Encoder . . . 6
2.3.1 Incremental Encoder . . . 6
2.3.2 Resolution and Accuracy . . . 7
2.4 Printed Circuit Board . . . 8
2.5 Pin Configuration . . . 8

3 Software 9
3.1 MathWorks . . . 9
3.2 STMicroelectronics . . . 9
3.2.1 API . . . 10
3.3 Arduino . . . 10

4 Model Formulation and Control System Design 12
4.1 System Identification . . . 13
4.1.1 Model Validation . . . 13
4.2 State-space-model . . . 14
4.3 Controllability and Observability . . . 15
4.4 Controller . . . 16
4.4.1 Linear Matrix Inequality . . . 16
4.4.2 Linear Quadratic Regulator . . . 17
4.5 Observer Feedback Control Design . . . 18

5 Code Generation and Implementation 20
5.1 Method of Integration . . . 20
5.1.1 STM32-MAT . . . 20
5.1.2 Simulink Support Packages . . . 21
5.2 Encoder . . . 22
5.3 Data Transmission . . . 22
5.3.1 Simulink-controlled STM32 Board . . . 22
5.3.2 Arduino-programmed STM32 Board . . . 23
5.4 Motor Control . . . 25
5.5 Simulink Model . . . 26

5.6 Results 28

6 Conclusion **30**

Literature **ix**

Eidesstattliche Erklärung **xi**

Appendix **xii**
A. STM32L476RG Datasheet xii
B. Arduino program xiii
C. S-function builder for encoder xv

List of Figures

Figure 1.1 Test Rig Rotary Wheel Pendulum 1
Figure 2.1 STM32-L476RG Board and Pinout 2
Figure 2.2 UART wiring 3
Figure 2.3 UART packet 3
Figure 2.4 Field Oriented Control 5
Figure 2.5 SimpleFOCShield v2 board 5
Figure 2.6 Configured Soldering Pad of SimpleFOCShield 6
Figure 2.7 Schematic representation of an encoder 6
Figure 2.8 Pulse diagram of incremental encoder 7
Figure 2.9 Accuracy and Resolution 7
Figure 2.10 Printed Circuit Board 8
Figure 4.1 Rotary Inverted Pendulum model 12
Figure 4.2 Model Validation 14
Figure 4.3 LQR standard control loop 18
Figure 4.4 Observer-based state feedback control 19
Figure 5.1 Configuration for COM Port Simulink 20
Figure 5.2 STM32-MAT/Target MCU Configuration Block 21
Figure 5.3 S-function for reading encoder values 22
Figure 5.4 SCI Pin Configuration in Simulink 23
Figure 5.5 Serial Communication in Simulink 23
Figure 5.6 STM32CubeMX's communication protocols 24
Figure 5.7 Simulink Model for RIP simulation 26
Figure 5.8 Simulated Pendulum Angle 27
Figure 5.9 Simulink Model for RIP in external mode 28
Figure 5.10 Result of the pendulum angle in external mode 29

List of Tables

Table 2.1 GM6208 Datasheet 4
Table 2.2 Pin Mapping 8
Table 4.1 Parameter notations of the RIP 12
Table 4.2 System parameters 13

List of Listings

Lists 4.1 Matlab code for controllability ad observability of a system 16
Lists 5.1 Reading encoder values STM32 22
Lists 5.2 Hardware Serial Initialization 24
Lists 5.3 Receiver function for UART 24
Lists 5.4 Calling function to decode bits 25
Lists 5.5 Defining and initializing motor and encoder 25
Lists 5.6 Control loop type setting 25
Lists 5.7 FOC routine code 26

Nomenclature

Latin Letters

Symbol	Unit	Definition
f	Hz	Frequency
c_1	Nms/rad	Inertia moment of pendulum
c_2	Nms/rad	Inertia moment of wheel
J_1	kgm^2	Inertia moment of pendulum
J_2	kgm^2	Inertia moment of wheel
K_b	V/rad/s	Back emf constant
l_1	m	Length from origin to COG of pendulum
l_2	m	Length from origin to COG of wheel
L_t	N m/A	Motor torque constant
m_1	kg	Mass of pendulum and stator
m_2	kg	Mass of wheel and rotor
R_a	Ω	Armature winding resistance
t	s	Time

Greek Letters

Symbol	Unit	Definition
ϕ	°[degree]	Angle of the motor relative to the start reference angle
θ	°[degree]	Angle of pendulum

Abbreviations

ADC	Analog Digital Converter
API	Application Programming Interface
BLDC	Brushless Direct Current
COG	Center of Gravity
FOC	Field Oriented Control
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
I/O	Input/Output
LED	light-emitting diode
LL	Low-Layer
LMI	Linear Matrix Inequality
PCB	Printed Circuit Board
PPR	Pulse per Revolution
PWM	Pulse-width Modulation
RIP	Rotary Inverted Pendulum
RPM	Rotation per Minute
SCI	Serial Communication Interface
SIMO	Single Input Multiple Output
STM	STMicroelectronics
UART	Universal Asynchronous Receiver Transmitter

1 Introduction

The main goal of this thesis is to develop and implement suitable control system for a newly built reaction wheel pendulum test rig, which can be used for teaching and research purposes. This system will be able to run in real-time with Simulink. The test rig is comprised of two STM32-L476RG microcontrollers, two AMT103-V encoders, a SimpleFOCShield, a GM6208 BLDC Motor, and a custom printed circuit board to adapt the shield to the STM32 board (as illustrated in Figure 1.1).

To achieve these goals, this thesis approaches the final objective progressively. In Chapter 2, the main hardware and the components of the Rotary Inverted Pendulum are examined to determine the decisions made in further sections of the work. The required software, support packages, and the programmable features of the STM32 microcontrollers for this project are explained in Chapter 3. It is recommended for the reader to use the same software version to avoid compatibility problems with the provided programming files. In Chapter 4, the model formulation, the controller design, and system identification are achieved through theoretical control principles and deduction of mathematical equations. The implementation of the generated code from the Simulink model, Arduino IDE generated code, and its analysis are reflected in Chapter 5.

Figure 1.1: Test Rig Rotary Wheel Pendulum

2 Hardware

2.1 Microcontroller

The STM32L476RG board is a low-cost and flexible microcontroller based on the high-performance Arm® Cortex®-M4 32-bit RISC core produced by STMicroelectronics[21]. It includes an on-board ST-LINK debugger, 1 user LED, and push buttons. It operates at a frequency of up to 80 MHz. To program and power the board, a USB 'type A to mini B' cable must be plugged. The STM32L476RG has two types of extension resources:

- ARDUINO® Uno V3 connectivity
- ST morpho extension pin headers for full access to all STM32 I/Os

The complete specifications, can be acquired from the STM32L476RG datasheet[21]. A picture of the board and its pinout can be found in Figure 2.1.

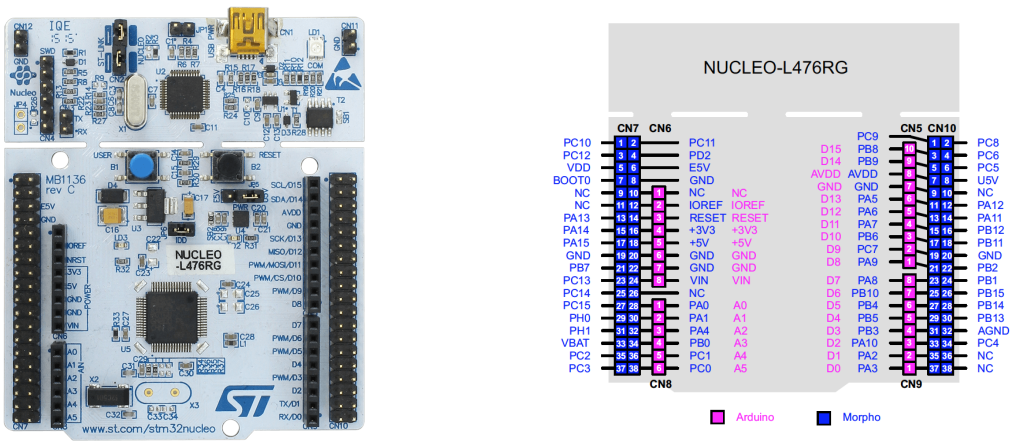


Figure 2.1: STM32-L476RG Board and Pinout [22]

2.1.1 Timer

The STM32L476RG contains two advanced control timers, up to nine general-purpose timers, two basic timers, two low-power timers, two watchdog timers, and a SysTick timer. In the STM32 microcontroller documentation, a general purpose timer peripheral is always referred to as "TIMx", where "x" can be any number and does not indicate the number of timer peripherals built into a particular microcontroller[24]. Some of the timers feature auto-reload up/down-counter and Prescaler, which could be configured through STM32CubeMX.

In this project two timers are used, which are the general purpose timers TIM3 and TIM4. These timers feature 16-bit auto-reload up/down-counter and 16-bit Prescaler. They have 4 independent channels capable of input capture, output compare, PWM mode, and one-pulse mode output. Both timers support quadrature encoders through

the use of "encoder-mode", which interprets the incoming signal from the encoder into counts[21].

2.1.2 Serial Communication

There are various types of hardware communication protocols available in an STM32 microcontroller, such as serial communication. In serial communication, there are different methods of exchanging data using serial digital binary. The data transmitted between sub-components in serial communication is in the form of binary pulses, where 1 represents a logic HIGH and 0 represents a logic LOW. All bits can be packed into one conductor with serial communication and transmitted one after the other in a fixed clock. The operating modes of serial communication can be divided into three types

- **Simplex**: one-way communication technique.
- **Half Duplex**: bidirectional non-simultaneous data transmission.
- **Full Duplex**: bidirectional simultaneous data transmission.

It is also important to consider endianness, which is the order of storing data at a particular memory address. Endianness can be big or little, depending on which value is stored first.

In this work, a serial communication method known as universal asynchronous receiver transmitter (UART) will be used. The term "asynchronous" in UART means that it does not require an additional clock signal to synchronize the data transmission between the transmitter and receiver. Instead, the devices are configured to have the same baud rate to synchronize the bits delivered from the transmitter to the receiver. As shown in Figure 2.2, the connection between two UART devices is established through the wiring of two pins, known as Rx (receiver) and Tx (transmitter).

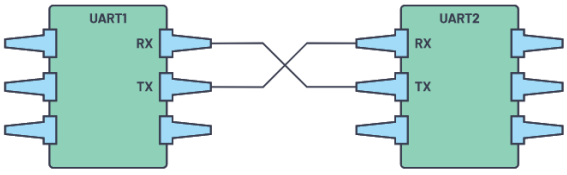


Figure 2.2: UART wiring [19]

In UART, the data is transmitted in the form of a packet. It consists of a start bit, five to nine data bits, a parity bit, and one to two stop bits.

Figure 2.3: UART packet[19]

Start bit is used to signal to the transceiver and receiver that the data is ready for transmission and acquisition. By changing this from 1 to 0, the transmission of the data can be started.

Data bits contain the actual data that is transmitted from the transmitter to the receiver. Normally there are 5 to 8 long bits in these data bits. An exception of 9 long bits is present if there is no parity bit in the entire data frame.

Parity bit checks the accuracy of the transmitted data by comparing the evenness and oddness of this data with the parity bit defined before the transmission.

Stop bit signals the end of the data frames and aborts the UART protocol by changing the transceiver signal from 0 to 1. [19]

To ensure that the correct data is being transmitted, received, and read, a header-data-trailer packet format is applied. The header consists of bytes, which indicates that a new packet has started. While the trailer is used as a packet separator or terminator.

2.2 BLDC Motor

The control of the reaction wheel pendulum will be done over a brushless DC (BLDC) motor, in this case, the GM6208 Brushless Gimbal Motor by iPower Motors will be used. The technical specification of the motor is as follows:

Weight [g]	249
Dimension [mm]	69.5*24
Maximum rotation speed [RPM]	209-231
Torque [g]	2600-3600
Configuration	24N28P
Hollow Shaft Diameter [mm]	22

Table 2.1: GM6208 Datasheet [14]

Field-oriented control(FOC) is a mathematically advanced method of controlling the electronic commutation of a 3-phase-BLDC motor. It is similar to sinusoidal commutation, but it has better efficiency and accuracy at higher speeds.[7]

In the FOC method, there are two current loops (as illustrated in Figure 2.4), one for the q reference frame and another for the d reference frame. The q loop is controlled by the target torque value set by the controller. The d loop is set to zero to minimize unwanted torque. Through the use of *Clarke and Park transformation*, the vectorized phase angle is converted to and from the de-referenced d and q reference frames.[16]

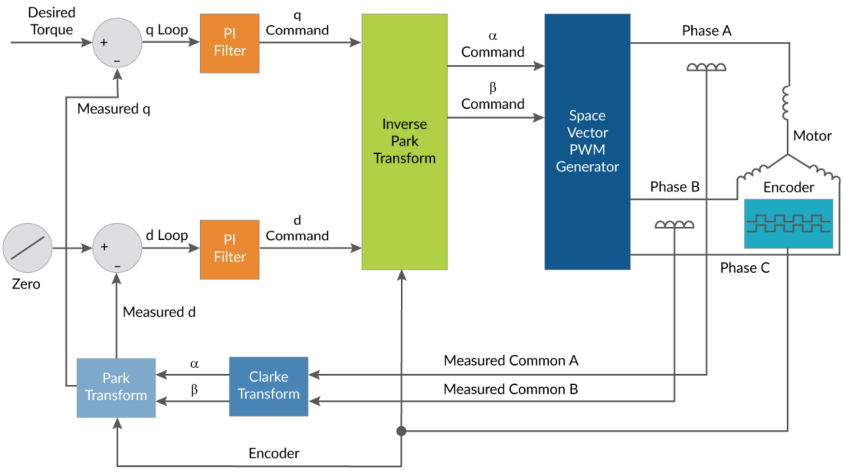


Figure 2.4: Field Oriented Control [16]

2.2.1 SimpleFOCShield

SimpleFOCShield is a low-cost open-source driver board, intended for FOC applications of BLDC motors. The board is designed to be fully compatible with Arduino UNO and other microcontrollers with the standard Arduino headers, such as the STM32L476RG used in this project. In conjunction with the SimpleFOCShield, an Arduino library that implements the FOC algorithm, called SimpleFOCLibrary, is available for users to control the motor by hardware and software.[20]

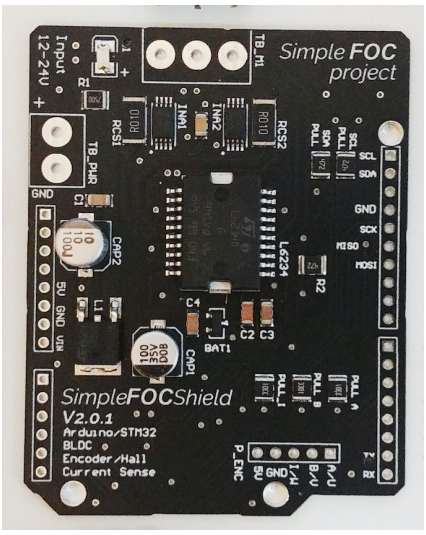


Figure 2.5: SimpleFOCShield v2 board [20]

The SimpleFOCShield can be configured through a set of solder pads on the bottom side of the board. The configuration of the board for this project can be seen in Figure 2.6. Since STM32 microcontrollers operate with a 3.3 V power supply, the shield is set to the same voltage.

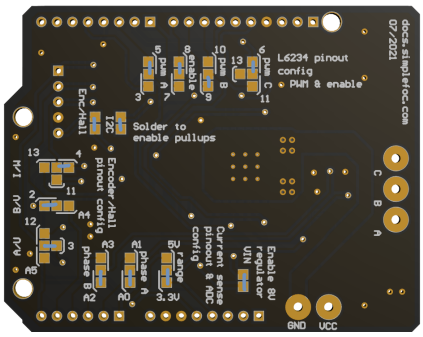


Figure 2.6: Configured Soldering Pad of SimpleFOCShield

2.3 Encoder

An encoder is a type of position sensor used to determine the angular position of a rotating shaft. It generates an analog or digital electrical signal corresponding to the rotational motion. In general, an encoder consists of three components:

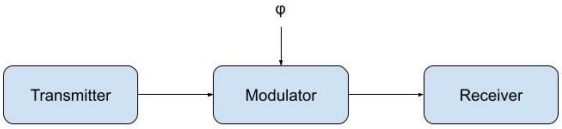


Figure 2.7: Schematic representation of an encoder [12]

The *transmitter* supplies the system with energy, e.g. in the form of LED for an optical encoder. The *modulator* changes the supplied energy in proportion to the mechanical angle ϕ and thus serves as an etalon or measuring standard. The *receiver* converts the modulated physical quantity into an electrical signal[12].

There are different types of encoders, distinguished by either the output signal or the sensing technique. In this project, two AMT103-v quadrature encoders are used. The encoder can be supplied with a voltage from 3.6V-5.5V and its resolution has been set to 2048. For further details, it can be found in CUI Device's AMT10 Datasheet[6].

2.3.1 Incremental Encoder

An incremental encoder, also known as quadrature encoder, transmits the output position signal digitally or analog. A certain number is generated per revolution of pulses, where each pulse represents an increment corresponding to the specified resolution. An incremental encoder can measure the change in position, but not the absolute position.[10]

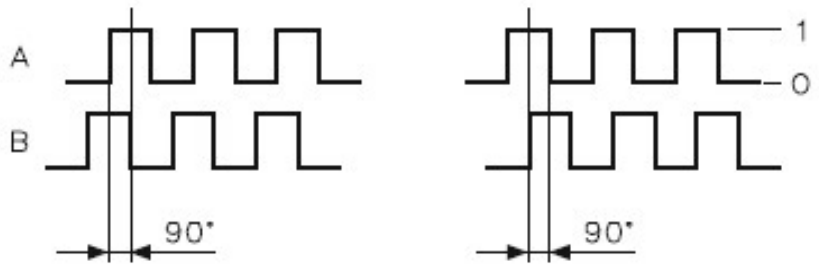


Figure 2.8: Pulse diagram of incremental encoder [11]

The output position signal is transmitted via two digital signals A and B, commonly referred to as the "A/B signal states". When signal A comes *before* signal B, each edge corresponds to an increment in the *positive* direction; when signal A comes *after* signal B, it corresponds to an increment in the *negative* direction [12].

2.3.2 Resolution and Accuracy

The resolution and accuracy of an encoder are important characteristics that describe the performance of the encoder.

The resolution describes the possible number of measuring units per mechanical revolution of the encoder. For digital incremental encoders, the resolution is measured as the number of pulses per revolution (PPR).

The accuracy is a measure of the error between the value acquired from the encoder and the real physical value. The resolution of an incremental encoder is based on the number of strokes on the encoder disk. The resolution is physically set. However, the resolution of an incremental encoder can be doubled (x2) or quadrupled (x4) by adjusting which leading and trailing edges of the two output signals are counted [5].

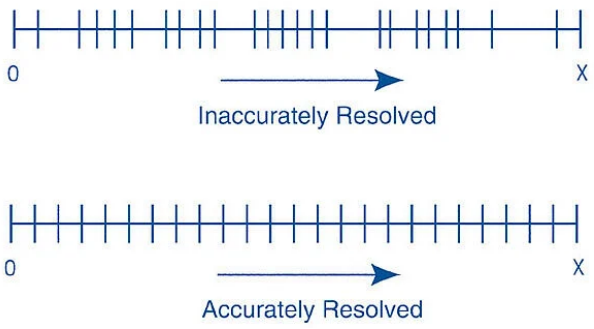


Figure 2.9: Accuracy and Resolution [5]

In Figure 2.9, the two encoders have the same resolution of 24 counts but have different accuracy. This shows that an encoder's resolution and accuracy are independent of each other.

2.4 Printed Circuit Board

An STM32 microcontroller is operated with a 3.3 V power supply. On the other hand, the encoder generates signals with 5 V. If the microcontroller and the encoder are connected without any shift in logic levels, the components can be damaged. To prevent that, a custom printed circuit board will be used in this work to shift the logic levels of the encoders from 5 V to 3.3 V and configure some of the pin configurations of the SimpleFOCShield so that it is suitable with the STM32 board.

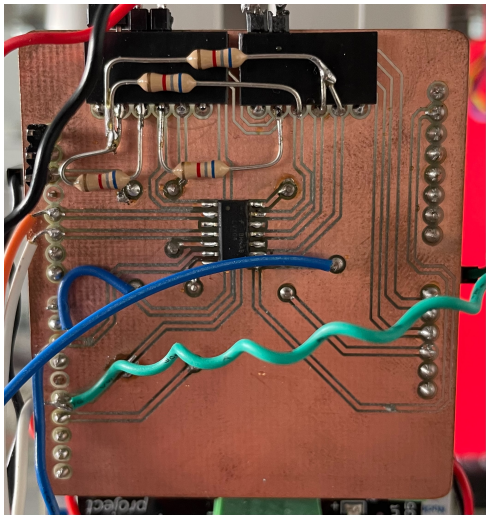


Figure 2.10: Printed Circuit Board

2.5 Pin Configuration

The pin configuration for the microcontroller connected to the shield is shown in Table 2.2.

Pin	Function
PA2	UART2 transmitter
PA3	UART2 receiver
PA8	PWM 1
PA9	PWM 2
PA10	PWM 3
PB4	Motor position encoder A
PB5	Motor position encoder B
PB6	Pendulum angle encoder A
PB7	Pendulum angle encoder B
PC4	UART3 transmitter
PC5	UART3 receiver

Table 2.2: Pin Mapping

Similar to the aforementioned pin configuration, the microcontroller responsible for the Simulink model has a similar pin mapping.

3 Software

3.1 MathWorks

MATLAB is a high-level software, designed for mathematical calculation, analysis, and modeling. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulation of dynamic systems. It is integrated with MATLAB and allows MATLAB algorithms to be incorporated into models and simulation results to be exported to MATLAB for further analysis. The STM32 microcontroller can be connected to Simulink and programmed by downloading the appropriate support packages. In Simulink, the STM32 microcontroller is connected in "real-time" and data processing takes place, which is called "external mode".

An overview of all essential software, toolboxes, and additional hardware support packages from Mathworks is described as follows:

Licensed base software:

- MATLAB 2022a and Simulink

Toolbox with licensing requirement:

- Simulink Coder
- MATLAB Coder
- Embedded Coder

Available Hardware Support Packages:

- Simulink Coder Support Package for STMicroelectronics Nucleo Boards
- Embedded Coder Support Package for STMicroelectronics STM32 Processors

3.2 STMicroelectronics

To program an STM32 microcontroller, several integrated development environments (IDEs) are available from STMicroelectronics. The following programs must be downloaded from STMicroelectronics website:

- STM32CubePogrammer 2.6.0
- STM32CubeMX 6.2.0

- STM32CubeIDE 1.9.0
- STM32-MAT/Target

STM32CubeProgrammer is a software tool to read, write, and verify the device memory of the microcontroller through both the debug interface (JTAG and SWD) and the bootloader interface (UART, USB DFU, I2C, SPI, and CAN). It provides firmware upgrades for the microcontroller. In this project, the firmware version of the ST-LINK of the microcontrollers is V2J37M26.

STM32CubeMX is a GUI-based initialization code C-code generator that embeds Hardware Abstract Layer (HAL) drivers into the base project. The generated .ioc project file can then be imported into the workspace of STM32CubeIDE or other supported IDE.

STM32CubeIDE is an integrated development environment developed by STMicroelectronics. The Arm toolchain is supported by this open-source IDE, which also offers a number of practical debugging tools including the Serial Wire Debug (SWD) based on the GNU Debugger (GDB).

STM32-MAT/Target is a software package developed by STMicroelectronics to extend target support for Simulink and MATLAB. This software comes various peripheral driver blocks, such as the GPIOs and timers. However, it is has been succeeded by MathWorks Embedded Coder Support.

3.2.1 API

An API is an application programming interface that defines a set of routines, protocols, and tools for building an application. It is created to be generic and implementation-independent. This allows the API to be used in multiple applications, changing only the implementation of the API and not the general interface or behavior[1].

The STM32 microcontrollers have two types of APIs that are commonly used: Hardware Abstraction Layer and Low-Level. The Hardware Abstraction Layer (HAL) is an abstraction layer intended for maximized portability across the STM32 microcontrollers. While the low-layer APIs (LL) offer a fast lightweight layer at the register level [23]. The Hardware Abstraction Layer will be utilized for the code generation of the STM32 microcontroller.

3.3 Arduino

The SimpleFOClibrary must be installed to program the SimpleFOCShield. Currently, the SimpleFOClibrary is only available in the *Arduino IDE*. This Arduino library provides the user with torque, velocity, and position control through the implementation of the field-oriented control algorithm for BLDC motors. STM32 microcontrollers are generally supported by the Arduino IDE through the STM32Duino package, which is open-source

and can be installed directly through the Arduino board manager. To check if SimpleFOCShield is compatible, it is first tested and monitored using the available sample codes via SimpleFOCStudio, which is a graphical user interface for the SimpleFOClibrary. This software allows the controlled motor to be tuned and configured via the serial port and "Commander" interface.

4 Model Formulation and Control System Design

A mechanical system can be described through the use of differential equations. From these equations, can go through model verification to adjust the parameters and the state space model can be derived. The model formulation of the test stand is based on a paper by Quang Khanh Ho and Cong Bang Pham[13].

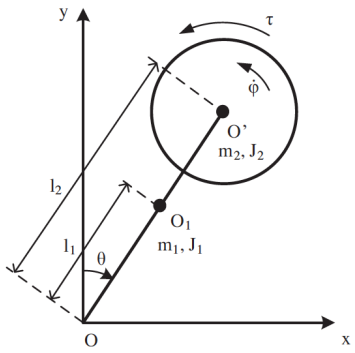


Figure 4.1: Rotary Inverted Pendulum model [13]

Figure 4.1 represents a model of the rotary inverted pendulum, where θ and ϕ are the pendulum angle and wheel angle respectively. Table 4.1 lists the parameter notations for the rotary inverted pendulum.

Symbol	Unit	Definition
$theta$	degree	Angle of pendulum
ϕ	degree	Angle of wheel
J_1	kgm^2	Inertia moment of pendulum
J_2	kgm^2	Inertia moment of wheel
c_1	Nms/rad	Friction factor of pendulum
c_2	Nms/rad	Friction factor of motor
m_1	kg	Mass of pendulum and stator
m_2	kg	Mass of wheel and rotor
l_1	m	Length from origin to COG of pendulum
l_2	m	Length from origin to COG of wheel
K_b	$V/rad/s$	Back-emf constant
L_t	Nm/A	Motor torque constant
R_a	Ω	Armature winding resistance

Table 4.1: Parameters notations of the RIP [13]

Based on Figure 4.1, the system's overall nonlinear dynamic equation is[13]:

$$\begin{bmatrix} m_1 \cdot l_1^2 + m_2 \cdot l_2^2 + J_1 + J_2 & J_2 \\ J_2 & J_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} + \begin{bmatrix} c_1 + 0 & \\ 0 & \frac{K_t \cdot K_b}{R_a} + c_2 \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} .(m_1 \cdot l_1 + m_2 \cdot l_2^2) \cdot g \cdot \sin\theta \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{K_t}{R_a} \end{bmatrix} V_m \tag{1}$$

The system parameters are as shown in Table 4.2.

Parameter	Value
J_1	0.0013 $(kg \cdot m^2)$
J_2	0.0001 $(kg \cdot m^2)$
c_1	0.004$(\frac{N \cdot m \cdot s}{rad})$
c_2	0.0007 $(\frac{N \cdot m \cdot s}{rad})$
m_1	0.52 (kg)
m_2	0.195 (kg)
l_1	0.12(m)
l_2	0.1020(m)
K_b	0.0987 $(\frac{V}{\frac{rad}{s}})$
L_t	0.0987 $(\frac{N \cdot m}{A})$
R_a	32.6 (Ω)

Table 4.2: System parameters

4.1 System Identification

System identification is the process of developing a model of a dynamic system through the use of data. In this project, a grey box approach is used to create a model. The gray box approach allows the model to be developed through the use of data and system identification by estimating the parameters of the model[8][18].

4.1.1 Model Validation

Validation is an indispensable part of constructing an appropriate simulation model. By changing the parameters, the model can perform as intended. These parameters can be adjusted through trial and error to fit the real-world output.

In this work, this is done by letting go the pendulum from an angle and adjusting the parameters of the model to fit the the values of the angle of the pendulum of the test stand as illustrated in Figure 4.2.

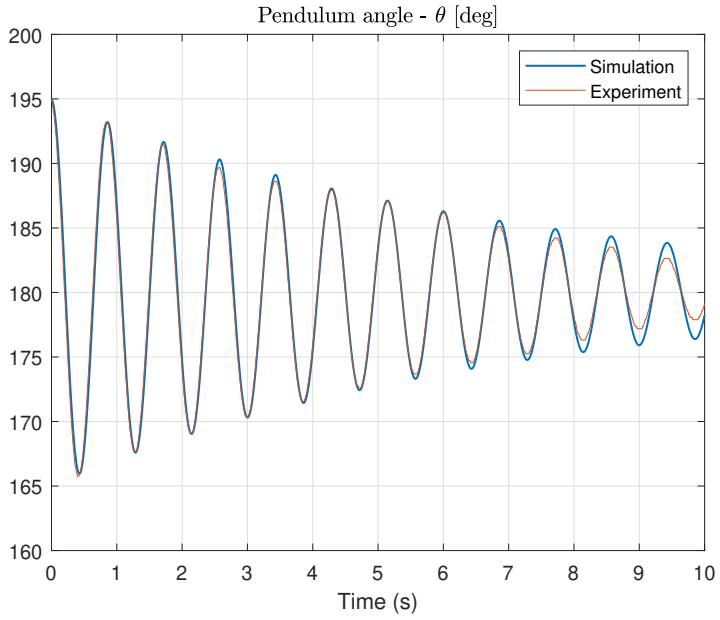


Figure 4.2: Model Validation

Through this process, the parameters in 4.2 have been adapted to the test stand that is used in this work.

4.2 State-space-model

The dynamic equation of a model can be represented in a state-space form as:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{2}$$

The state vectors of the system are selected as desired. Thus, the chosen state vectors for this particular system are expressed as:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} := \begin{pmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{pmatrix} \tag{3}$$

where θ, $\dot{\theta}$, ϕ, and $\dot{\phi}$ are the pendulum angle, angular velocity of the pendulum, wheel angle, and the angular velocity of the wheel respectively.

With the state vector and u, the state space model $\dot{x}$ = f(x,u) can be derived from the differential equation of motion[13].

$$
\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= \frac{b \cdot g \cdot \sin x_1 - c_1 \cdot x_2 + c_2 \cdot x_4 - u}{a - J_2} \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= \frac{\frac{a \cdot u}{J_2} + c_1 \cdot x_2 - b \cdot g \cdot \sin x_1 - \frac{a \cdot c_2 \cdot x_4}{J_2}}{a - J_2}
\end{aligned}
\tag{4}
$$

with

$$
\begin{aligned}
a &= m_1 \cdot l_1^2 + m_1 \cdot l_2^2 + J_1 + J_2 \\
b &= m_1 \cdot l_1 + m_2 \cdot l_2
\end{aligned}
\tag{5}
$$

Due to the nonlinearity of the pendulum motion, the pendulum dynamics need to be linearized. Linearization can be done through approximation of the nonlinear dynamics using a Taylor series expansion around the operative point. After linearization with an operating point of (0,0), the linear state space model of the differential equation is:

$$
\begin{aligned}
\Delta\dot{\underline{x}} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{b \cdot g}{a - J_2} & \frac{-c_1}{a - J_2} & 0 & \frac{c_2}{a - J_2} \\ 0 & 0 & 0 & 1 \\ \frac{-b \cdot g}{a - J_2} & \frac{c_1}{a - J_2} & 0 & \frac{c_2}{J_2(a - J_2)} \end{bmatrix} \Delta\underline{x} + \begin{bmatrix} 0 \\ \frac{1}{a - J_2} \\ 0 \\ \frac{a}{J_2(a - J_2)} \end{bmatrix} \Delta\underline{u} \\
\Delta\dot{\underline{x}} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 52.8 & -0.28 & 0 & 0.49 \\ 0 & 0 & 0 & 1 \\ -52.8 & 0.28 & 0 & -70.1 \end{bmatrix} \Delta\underline{x} + \begin{bmatrix} 0 \\ -4.47 \\ 0 \\ 638.79 \end{bmatrix} \Delta\underline{u}
\end{aligned}
\tag{6}
$$

$$
\Delta\underline{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \Delta\underline{x} \tag{7}
$$

In the case of a controller that is to return to the origin states of the system, the input u is replaced by a controller gain matrix K, which is multiplied by the state vector replaced

$$
\underline{u} = -K\underline{x} \tag{8}
$$

4.3 Controllability and Observability

Controllability and observability are two crucial properties of a control system that are dependent on each other. Controllability refers to the ability to control a system so that

it can be made stable. Observability is important because if a system is observable, the system determine whether it is stable or not. Controllability and observability are coupled together because they are dependent on each other.

If a system can be controlled by changing the inputs, or parameters, but its behavior or performance cannot be observed, the system cannot be controlled effectively. Similarly, if a system can be observed to determine whether it is stable or not, but it cannot be controlled, the state of the system is not changeable.

To determine the controllability and observability of a system, Kalman's test is used. If system matrix A has an order of $n \times n$, the controllability matrix Q_c and observability matrix Q_o are formed, where B is a column matrix and C is a row matrix.[9]

$$Q_0 = \begin{bmatrix} B & BA & BA^2 & .. & BA^{n-1} \end{bmatrix} \tag{9}$$

$$Q_c = \begin{bmatrix} C \\ CA \\ CA^2 \\ .. \\ CA^{n-1} \end{bmatrix} \tag{10}$$

If the rank of Q_c and Q_o are n, the system is controllable and observable. In Matlab, the following code can be used to determine the controllability and observability of the system using the state space model that has been constructed in Equation 4.2.

```
% Check for controllability
Co = ctrb(sys);
unco = length(A) - rank(Co); % if 0 then controllable
% Check for observability
Ob = obsv(sys);
unob= length(A) - rank(Ob); % if 0 then observable
```

Lists 4.1: Matlab code for controllability ad observability of a system

The system is both controllable and observable.

4.4 Controller

4.4.1 Linear Matrix Inequality

The prerequisites for the controller design through pole placement with the help of the Linear Matrix Inequality (LMI) are that all states of the system x are known and the controllability of the system is given.

The controller design is represented through the transformation of the linear state-space model from (2) using the controller law from (8) as

$$\dot{\underline{x}} = (A - BK)\underline{x}. \tag{11}$$

The required matrix K guarantees the stability of the system. If a higher form of stability is required, such as exponential stability, the resulting LMIs can be adjusted to obtain it.

Through the derivation of the Lyapunov function, the following equation can be acquired[3].

$$\begin{aligned} XA^T + AX - M^T B^T - BM + 2\alpha X \prec 0 \\ X \succ 0 \end{aligned} \tag{12}$$

Matrix K can be calculated with the help of M and X.

$$K = MX^{-1} \tag{13}$$

The parameter α determines how far to the left (negative) the real value of the poles are from the imaginary axis of the pole-zero diagram.

Through the use of MATLAB codes, the controller gains K was found as

$$K_{LMI} = \begin{bmatrix} -50.51 \\ -6.58 \\ -0.02 \\ -0.14 \end{bmatrix} \tag{14}$$

4.4.2 Linear Quadratic Regulator

Linear Quadratic Regulator or LQR is a type of optimal control based on the state space representation. An LQR minimizes the deviations, i.e. in this case the deviations from the target angle.

$$J(u) = \int_0^\infty (x^T Q x + u^T R u + 2x^T N u) dt \tag{15}$$

depending on the system dynamics

$$\dot{x} = Ax + Bu \tag{16}$$

Figure 4.3 shows a standard LQR control system.

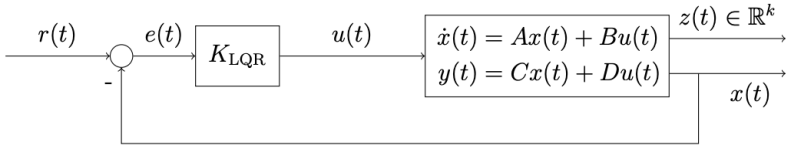


Figure 4.3: LQR standard control loop

In LQR, the Q matrix defines the weights on the states, while the R matrix defines the weights on the control input in the cost function[4]. Both Q and R are adjustable and can be altered through trial and error.

With the help of MATLAB, the closed-loop controller gains were found as

$$K_{LQR} = \begin{bmatrix} -913.39 \\ -123.7 \\ -0.35 \\ -0.58 \end{bmatrix} \tag{17}$$

with the Q and R values set as

$$\begin{aligned} Q &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ R &= 8 \end{aligned} \tag{18}$$

4.5 Observer Feedback Control Design

In this project, observers are necessary because not all states are measurable. Using an observer will allow the model to compensate for the missing measurements through estimation. The Luenberger observer is a type of asymptotic observer for the following plant model:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{19}$$

The dynamic of the Luenberger observer is represented by the following equations[2]:

$$\begin{aligned} \dot{\hat{x}} &= A\hat{x} + Bu + L(y - \hat{y}) \\ \hat{y} &= C\hat{x} \end{aligned} \tag{20}$$

where $\hat{x} \in \mathbb{R}$ represents the estimation of the reconstructed states. The observer error differential equation is

$$\dot{e} = (A - LC)e \tag{21}$$

Figure 4.4 shows the observer-based state feedback control structure with a Luenberger observer.

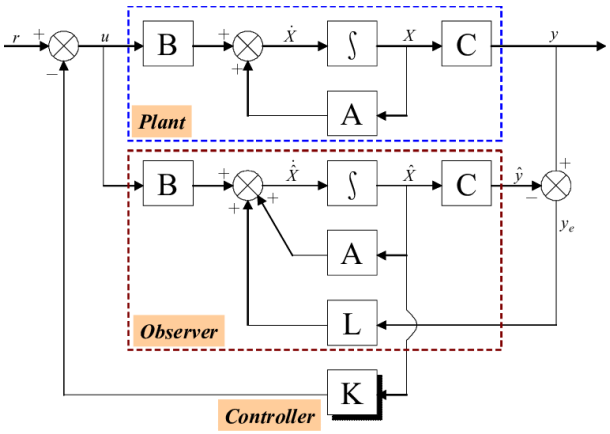


Figure 4.4: Observer-based state feedback control [15]

To determine the value of L, a similar method can be applied as seen from the pole placement method to determine K[3]. The observer gain L was obtained as

$$L = \begin{bmatrix} 29.25 & -0.27 \\ 425.98 & -3.06 \\ 0.37 & 10.16 \\ -58.09 & 40.63 \end{bmatrix} \tag{22}$$

5 Code Generation and Implementation

In this work, two STM32-L476RG microcontrollers are used. The first microcontroller will be reading the encoder values, running the Simulink model in external mode, and sending values to the other microcontroller through UART. While the second microcontroller will be flashed with an Arduino program that contains the SimpleFOC library and a serial communication decoder.

It is important to identify which communication port is responsible for each microcontroller. After identifying them, the communication port for Simulink can be configured in the Hardware Implementation as shown in Figure 5.1. In Arduino IDE, this can be done with the "Port" option under "Tools".

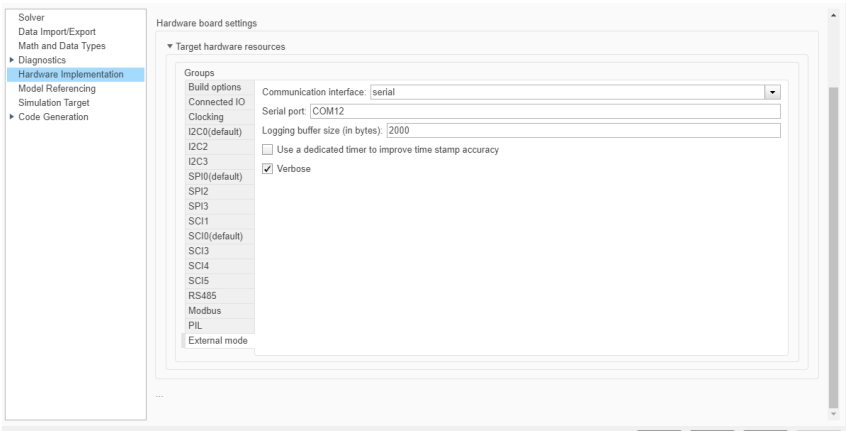


Figure 5.1: Configuration for COM Port Simulink

5.1 Method of Integration

Despite the availability of hardware support packages for STM32 microcontrollers in Simulink, there are limitations to it. The lack of target support for STM32 microcontrollers limits the capability of configuring individual peripherals of the board. To mitigate this issue there are two ways around this.

5.1.1 STM32-MAT

First is through the use of another support package called STM32-MAT/Target, which would allow users to integrate the configuration code file generated by STM32CubeMX and use more customizable driver blocks.

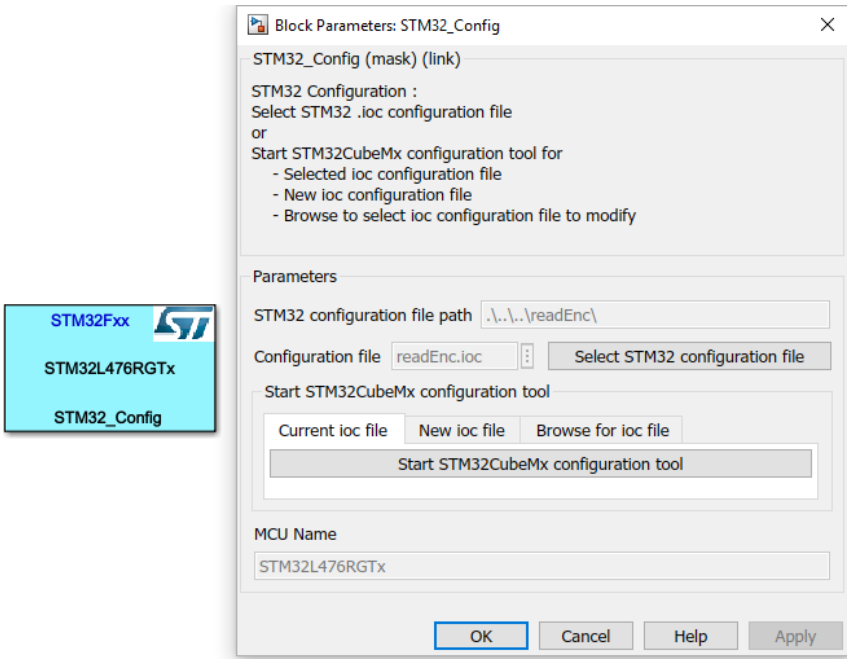


Figure 5.2: STM32-MAT/Target MCU Configuration Block

However, at the time of this work, this support package has a few drawbacks. As the support package is no longer supported and has been succeeded by Mathworks Embedded Coder Support, there are no further updates to it and to download and install it, the user has to find a third-party website.

Another drawback is the complexity of its workflow when generating the code for Simulink in external mode. After integrating the STM32CubeMX generated file, the model created in Simulink must be built using the Build button under Deployment and a series of dialog windows will appear to configure the communication ports. When the build is complete, the user will have to press the Generate Code button, which will allow STM32CubeMX to integrate the code file generated by Simulink with the original configuration file. When the code is generated, the project can be run with the debugger in the IDE that has been selected in STM32CubeMX. As the code is being debugged, the board can be connected to the Simulink model using the Connect button under Monitor & Tune.

Due to the complexity of this workflow, the process of code generation after changing a particular component of the model will be time-consuming. There are a few existing bugs, where the Simulink model in external mode cannot be terminated unless the debugger in the IDE is stopped first and the run-time of the model is inaccurate.

5.1.2 Simulink Support Packages

The second method is by using the available support packages listed in 3.1. The limitation with this is that not all peripherals are accessible through the available block sets. Some STM32 microcontrollers are more supported than others. To bypass this limitation, an S-function builder block can be used. Through the use of an S-function block, the previously inaccessible peripherals of the board can be configured through hard-coding

the initialization of the peripherals[17]. The initialization process done in the S-function block builder is done by cross-referencing an example code file that has been configured generated by STM32CubeMX.

5.2 Encoder

To access the encoder using an STM32 microcontroller, the timers are able to be set to "Encoder Mode" as previously discussed in 2.1.1. Since the timer peripheral is not accessible and configurable with the available support packages, the S-function builder block is used to initialize and configure the timers.

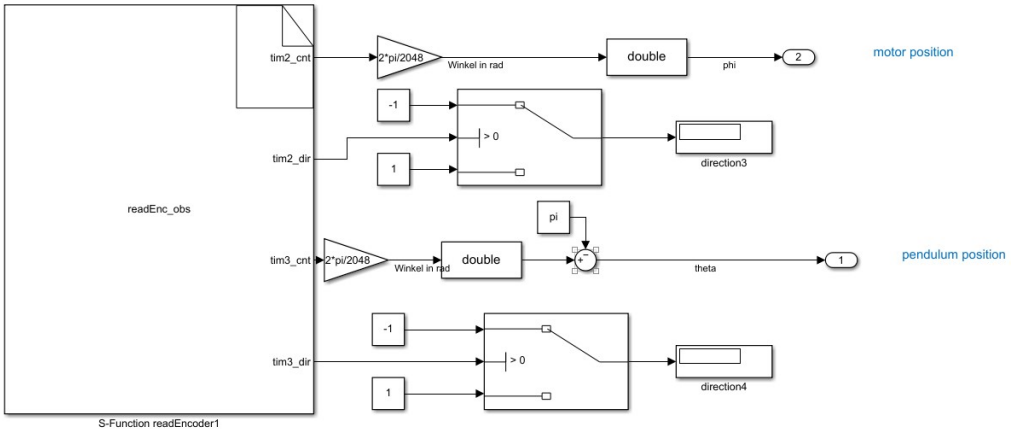


Figure 5.3: S-function for reading encoder values

In the S-function builder block, the HAL library for the particular microcontroller is also integrated through the " #include" command. Reading the encoders' values can be done by accessing STM32's encoder mode, with the following commands.

```
1 & Start encoder-mode
2 HAL_TIM_Encoder_Start(&htim2 ,TIM_CHANNEL_ALL);
3 HAL_TIM_Encoder_Start(&htim3 ,TIM_CHANNEL_ALL);
4
5 % Read encoder counts
6 tim2_cnt[0] = __HAL_TIM_GET_COUNTER(&htim2);
7 tim3_cnt[0] = __HAL_TIM_GET_COUNTER(&htim3);
```

Lists 5.1: Reading encoder values STM32

The complete code for the S-function builder block can be found in the Appendix C.

5.3 Data Transmission

5.3.1 Simulink-controlled STM32 Board

In Simulink, the term of the block set for UART protocol is represented by "SCI", which is an abbreviation for serial communication interface. When an STM32 microcontroller

is connected to a computer host, its default pins for UART are already occupied for the serial communication between the computer and the board. As a result, there is a need to select alternative pins for the Rx and Tx. The available SCI pins can be found in Simulink's Hardware Implementation as shown in Figure 5.4.

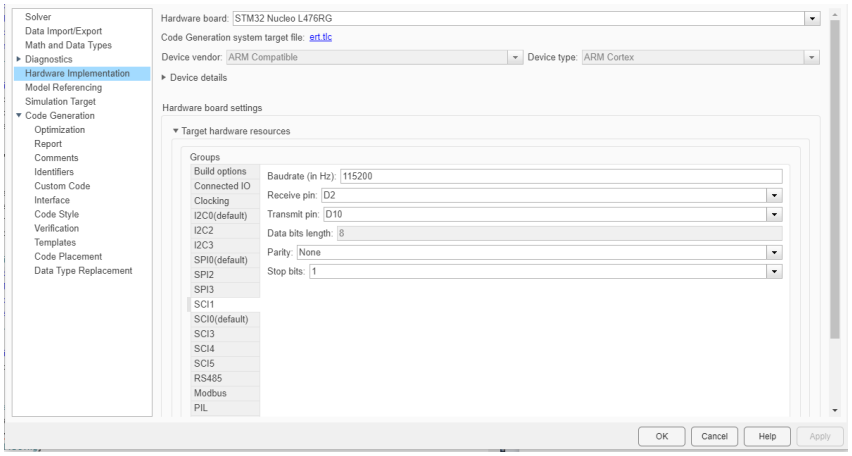


Figure 5.4: SCI Pin Configuration in Simulink

The first microcontroller sends the voltage target value to the second microcontroller in Simulink's external mode. This value's data type is initialized as an unsigned integer 16. With the "Byte Pack" block, the value is packed as a unsigned integer 8 data type. To ensure that the right bits are being sent and read, a header and a terminator are two essential indicators. In this case, as the value sent has a upper and lower limit of 12 and -12, the header and terminator are set to 100 and 200 respectively.

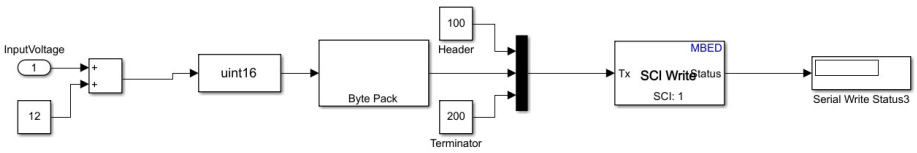


Figure 5.5: Serial Communication in Simulink

5.3.2 Arduino-programmed STM32 Board

The second microcontroller is programmed via Arduino IDE to receive the transmitted data and decode it. Similar to the previous problem, the pins responsible for UART needs to be changed from the default pins. This can be done by referencing the pin configuration of the UART protocol of STM32 from the STM32CubeMX, as shown in Figure 5.6.

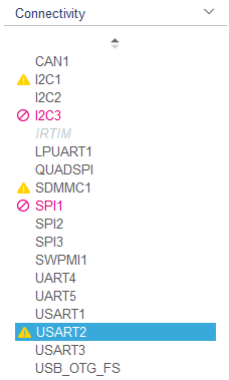


Figure 5.6: STM32CubeMX's communication protocols

The USART3 pins (PC5 and PC4) from STM32CubeMX are used as the alternative UART pins. To define and initialize the new pins in Arduino IDE, the "Hardware Serial" command can be used like the following:

```
HardwareSerial Serial3(PC5, PC4); //Rx, Tx

void setup()
{
    Serial3.begin(115200);
}
```

Lists 5.2: Hardware Serial Initialization

To decode the transmitted data, a function called recvWithStartEndMarker() is used as shown below.

```
void recvWithStartEndMarker() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    uint8_t startMarker = 100;
    uint8_t endMarker = 200;
    uint8_t rc;
    while (Serial3.available() > 0 && newData == false) {
        rc = Serial3.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                numReceived = ndx;
                ndx = 0;
                newData = true;
            }
        }

        else if (rc == startMarker) {
            recvInProgress = true;
        }
    }
}
```

Lists 5.3: Receiver function for UART

In this program, a variable called recvInProgress is an essential variable, because there needs to be a distinction made between data or values that arrive before the start marker and those that arrive after it. Each time the function recvWithStartEndMarker() is called, it reads the incoming data in the serial input buffer and places them in the array receivedChars. This function discards all incoming data until a header or start-marker is detected.

```
1 void loop()
2 {
3   recvWithStartEndMarker();
4   showNewNumber();
5 }
```

Lists 5.4: Calling function to decode bits

5.4 Motor Control

The motor control is executed by the second microcontroller that is programmed through Arduino IDE. The algorithm is referenced from SimpleFOC's sample projects[20]. The idea of the implemented motor control is to control the torque or voltage that is being used by the motor through the available SimpleFOCLibrary. The value of the torque or voltage is calculated through a Simulink model designed with the aforementioned equations and sent via UART to the second microcontroller, which is then decoded and set as the target value of the motor.

Using the SimpleFOCLibrary, the number of pole pairs and the pins can be defined to suit the user's specifications as shown below

```
1 // init BLDC motor
2 BLDCMotor motor = BLDCMotor(14); //number of pole pairs
3 BLDCDriver3PWM driver = BLDCDriver3PWM(PA8, PA9, PA10, PC7);
4
5 // init encoder
6 Encoder encoder = Encoder(PB4, PB5, 2048);
```

Lists 5.5: Defining and initializing motor and encoder

SimpleFOCLibrary implements 3 different control methods[20]:

- Torque control using voltage
- Velocity motion control
- Position control

In this work, the torque control will be implemented as illustrated in the following code

```
1 void setup()
2 {
3     motor.torque_controller = TorqueControlType::voltage;
4     motor.controller = MotionControlType::torque;
5 }
```

Lists 5.6: Control loop type setting

In the following code, the FOC routine that is available in the library is called. The variable "dataNumber" is the decoded value that was transmitted from Simulink. With the "motor.move()" command, the motor will move according to the input value and control loop type.

```
1 void loop() {
2 // iterative FOC function
3 motor.loopFOC();
4 motor.move(dataNumber);
5 }
```

Lists 5.7: FOC routine code

The full code can be seen in Appendix B..

5.5 Simulink Model

The construction of a simulation model is essential before creating a working model for the actual test station. This is done to ensure that the control parameters are correct. The plant model is first designed with reference to the state space model in Equation (4).

To simulate the real model, the available state vector will be set to only θ and ϕ.

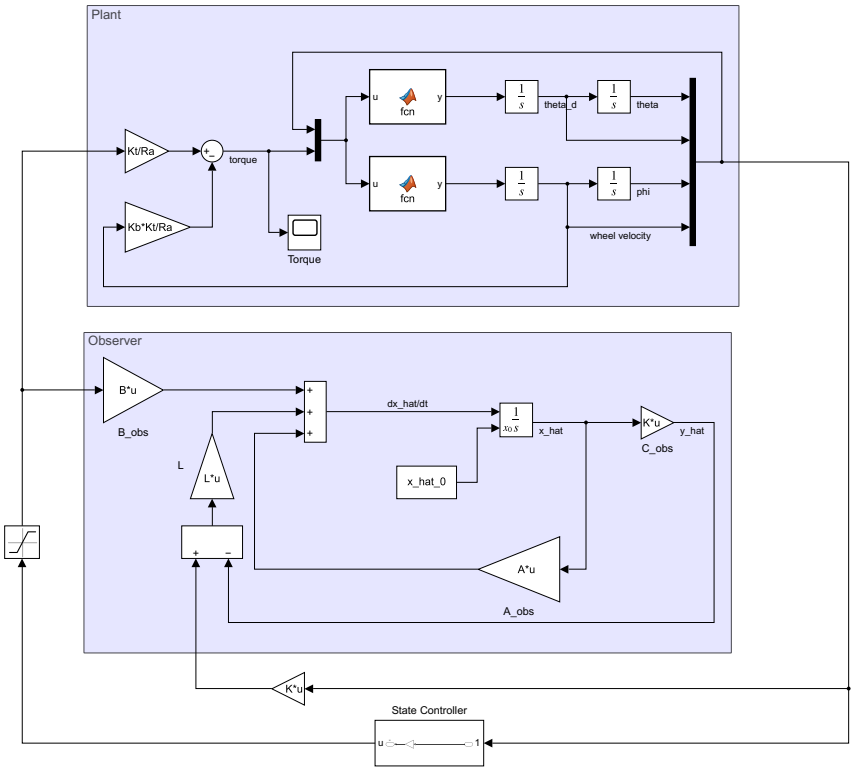
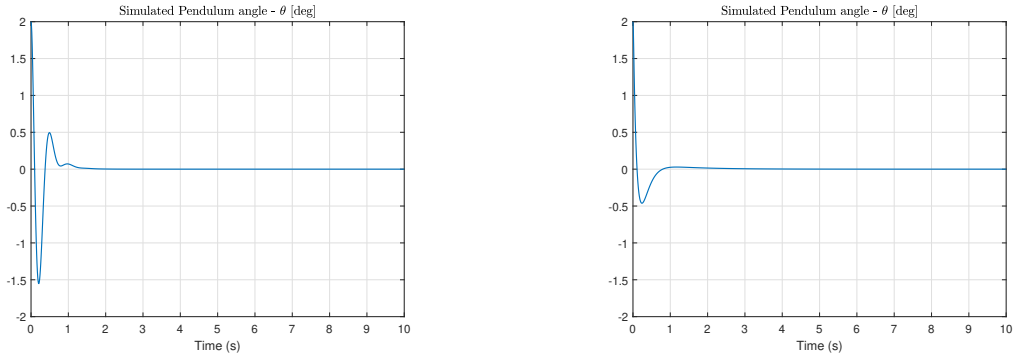


Figure 5.7: Simulink Model for RIP simulation

Figure 5.8 shows a comparison between the pendulum angle with gains that were calculated using LMI and LQR respectively. In this work, the LQR method is preferred because it allows more flexibility in determining the weight of each state vector. It also

gives a better performance to reach the target angle with minimal overshoot and settling time.



(a) With Pole Placement

(b) With LQR

Figure 5.8: Simulated Pendulum Angle

After several simulations, it can be concluded that the simulation model can balance itself from $\pm 4°$ and enters a steady state as illustrated in Figure 5.8.

After adjusting the control parameters, the model can be tested in real-time through Simulink's external mode. To do so, the "Monitor & Tune" button is selected, and Simulink will generate the appropriate code for the STM32 microcontroller to execute the model.

Since there are only two state vectors that are measurable, the remaining two state vectors are estimated through the observer model. With the complete state vectors, the voltage needed to be generated is then determined through the state controller and sent to the microcontroller responsible for the motor control. Figure 5.9 shows the complete model that is executed in external mode.

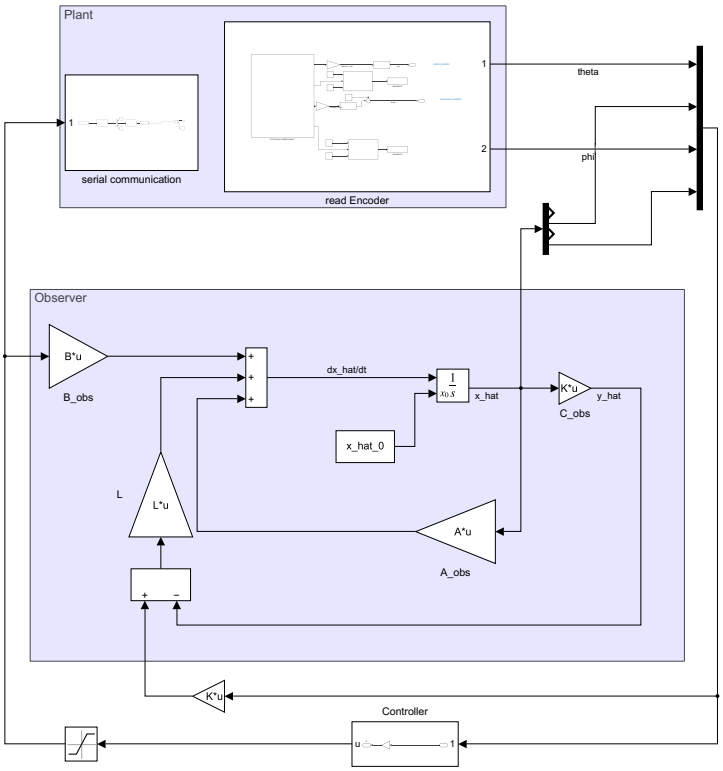


Figure 5.9: Simulink Model for RIP in external mode

5.6 Results

The results are not as good as expected. The swing-up algorithm performed as it should be. However, the test stand is having trouble balancing the pendulum. It can be seen in Figure 5.10 that instead of balancing at the zero degrees, it overshoots and restarts the swing-up process.

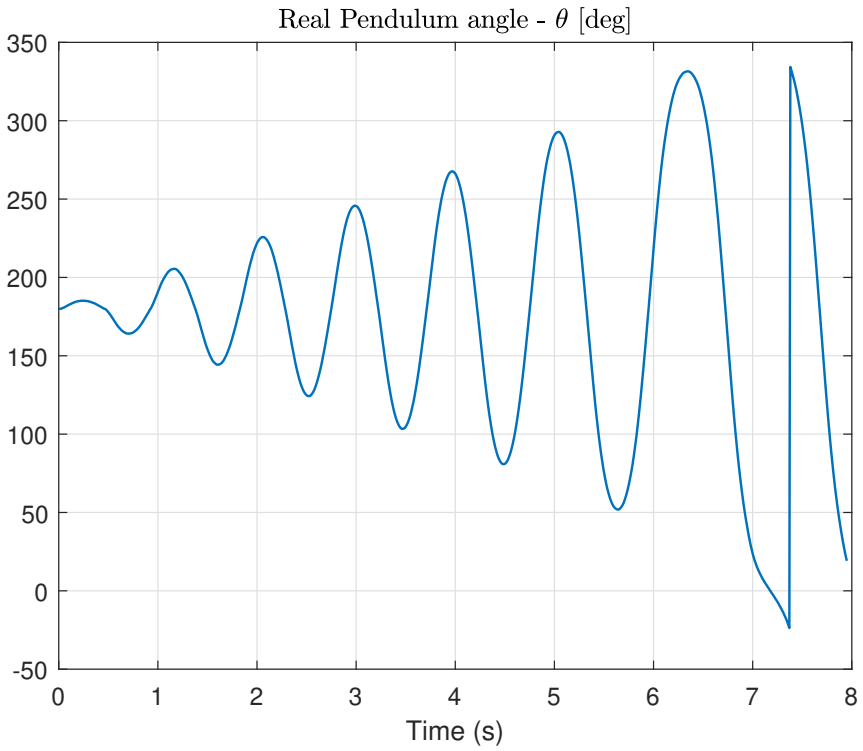


Figure 5.10: Result of the pendulum angle in external mode

It can be observed that the target voltage calculated through the Simulink model is either the maximum or minimum value of the voltage when limited by a saturation block. To mitigate this, the values of Q and R of the LQR controller are adjusted. However, after several iterations, the parameters have been tuned, but the results are still similar to Figure 5.10.

6 Conclusion

The results attained throughout the project's development are summarized in this section. Although the initial goals have been met, there is still room for improvement in a few areas, which will be examined in more detail later.

In this thesis, the development of the rotary inverted pendulum can be achieved by using two STM32 microcontrollers and SimpleFOCShield to control the BLDC motor and read the position of the pendulum. The first STM32 board is then programmed using Simulink and operated in external mode or real-time. Despite the lack of support for STM32 microcontrollers by MATLAB/Simulink, the integration of Simulink and STM32 is done through the use of the support packages and S-function builders instead of STM32-MAT due to its limitations. While the second board is attached to the SimpleFOCShield and programmed using C in Arduino IDE. This allows the development process of the motor control to be simplified due to the available libraries such as the SimpleFOCLibrary.

A simulation test of the whole model is also carried out to check whether the parameters are appropriate. The tests are performed by using the model validation and adjusting the LQR values of Q and R.

To run the test bench, two steps need to be done. First, the program created in Arduino IDE should be uploaded to the STM32 microcontroller with the SimpleFOCShield. Then through Simulink, the target voltage can be generated and sent via UART from the first to the second microcontroller, where it is decoded and set as the target value for the motor. The swing-up process relies on the effect of inertia through the change in the applied voltage or torque of the motor. The required voltage to be applied by the motor is calculated by using an LQR controller.

The swing-up process requires at least 7 seconds and at least 6 swings for it to reach the upright position. However, the pendulum overshoots and is unable to balance. This can be caused by the calculated target value of the voltage, which is only either the minimum or maximum value (in this case it is -12V and 12V). Another possible factor is the control parameters of the LQR state controller. Due to the trial and error method when determining the Q and R values, both variables may not have the most optimal values.

In summary, the method and simulation test described above can be used to develop a test bench to demonstrate the control implementation of the RIP using STM32, SimpleFOCShield, and a BLDC motor. However, further adjustments can be performed to improve the test bench. For example, the values of Q and R of the LQR controller can be further adjusted to allow the balancing of the pendulum in an upright position. And more simulations or tests can be done to find out why the target values of the voltage are either the minimum or the maximum values.

References

[1] Beningo, Jacob. *Embedded Basics – API's vs HAL's*. Website. Available: https://www.beningo.com/embedded-basics-apis-vs-hals/; Accessed 2 June 2022.

[2] Boutat, Driss and Zheng, Gang. *Observer Design for Nonlinear Dynamical Systems*. Springer, 2022. ISBN: 978-3-030-73741-2. URL: https://link.springer.com/book/10.1007/978-3-030-73742-9.

[3] Johannes Brunner. *Reglerentwurf für Inverses Pendel und Buck Converter*. 2022.

[4] Gaurav Chaudhary et al. "Review of Energy Storage and Energy Management System Control Strategies in Microgrids". In: *Energies* 14.16 (2021). ISSN: 1996-1073. DOI: 10.3390/en14164929. URL: https://www.mdpi.com/1996-1073/14/16/4929.

[5] Collins, Danielle. *Encoder resolution and accuracy: What's the difference?* Website. Available: https://www.motioncontroltips.com/encoder-resolution-and-accuracy-whats-the-difference/; Accessed 27 May 2022.

[6] CUI Devices. *AMT10 Datasheet*. Website. Available: https://www.cuidevices.com/product/resource/pdf/amt10-v.pdf; Accessed 8 July 2022.

[7] digikey. *How to Power and Control Brushless DC Motors*. Website. Available: https://www.digikey.de/de/articles/how-to-power-and-control-brushless-dc-motors; Accessed 9 June 2022.

[8] Douglas, Brian. *System Identification*. Website. Available: https://resourcium.org/journey/companion-resources-what-system-identification-system-identification-part-1; Accessed 20 July 2022.

[9] Electricalvoice. *Controllability and Observability in Control System*. Website. Available: https://electricalvoice.com/controllability-and-observability-in-control-system/; Accessed 18 August 2022.

[10] Gonzalez, Carlos. *What's the Difference Between Absolute and Incremental Encoders?* Website. Available: https://www.machinedesign.com/mechanical-motion-systems/article/21836107/whats-the-difference-between-absolute-and-incremental-encoders; Accessed 23 May 2022.

[11] Herbert, Bernstein. *Messelektronik und Sensoren*. Springer Vieweg, 2014. ISBN: 978-3-658-00549-8. URL: https://link.springer.com/book/10.1007/978-3-658-00549-8.

[12] Hering, Ekbert and Schönfelder, Gert. *Sensoren in Wissenschaft und Technik*. Ed. by Gert Schönfelder. Funktionsweise und Einsatztgebiet. SpringerVieweg, 2018.

[13] Quang Khanh Ho and Cong Bang Pham. "Study on Inertia Wheel Pendulum Applied to Self-Balancing Electric Motorcycle". In: *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*. 2018, pp. 687–692. DOI: 10.1109/GTSD.2018.8595698.

[14] iFlight. *iPower Motor GM6208 150T Brushless Gimbal Motor*. Website. Available: https://shop.iflight-rc.com/ipower-motor-gm6208-150t-brushless-gimbal-motor-pro208; Accessed 9 June 2022.

[15] Ting-En Lee, Juhng Su, and Ker-Wei Yu. “Implementation of the State Feedback Control Scheme for a Magnetic Levitation System”. In: June 2007, pp. 548–553. ISBN: 978-1-4244-0737-8. DOI: 10.1109/ICIEA.2007.4318468.

[16] Lewin, Chuck. *Field Oriented Control (FOC) - A Deep Dive*. Website. Available: https://www.pmdcorp.com/resources/type/articles/get/field-oriented-control-foc-a-deep-dive-article; Accessed 9 June 2022.

[17] Mathworks. *Build S-Functions Automatically Using S-Function Builder*. Website. Available: https://www.mathworks.com/help/simulink/sfg/s-function-builder-dialog-box.html; Accessed 13 June 2022.

[18] Mathworks. *System Identification Overview*. Website. Available: https://www.mathworks.com/help/ident/gs/about-system-identification.html; Accessed 20 July 2022.

[19] Peňa, Eric and Legaspi, Mary Grace. *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. Website. Available: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html; Accessed 13 June 2022.

[20] SimpleFOC. *SimpleFOCDocs*. Website. Available: https://docs.simplefoc.com/; Accessed 9 June 2022.

[21] STM32. *STM32L476xx*. Website. Available: https://www.st.com/resource/en/datasheet/stm32l476rg.pdf; Accessed 13 June 2022.

[22] STM32. *User manual - STM32 Nucleo-64 boards (MB1136)*. Website. Available: https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf; Accessed 13 June 2022.

[23] STMicroelectronics. *Description of STM32F7 HAL and low-layer drivers - User Manual*. Website. Available: https://www.st.com/resource/en/user_manual/dm00189702-description-of-stm32f7-hal-and-lowlayer-drivers-stmicroelectronics.pdf; Accessed 2 June 2022.

[24] STMicroelectronics. *General-purpose timer cookbook for STM32 microcontrollers*. Website. Available: https://www.st.com/resource/en/application_note/dm00236305-generalpurpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf; Accessed 3 June 2022.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass

- ich die vorliegende wissenschaftliche Arbeit selbständig und ohne unerlaubte Hilfe angefertigt habe,
- ich andere als die angegebenen Quellen und Hilfsmittel nicht benutzt habe,
- ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe,
- die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfbehörde vorgelegen hat.

Berlin, August 30, 2022

Appendix

A. STM32L476RG Datasheet

life.augmented

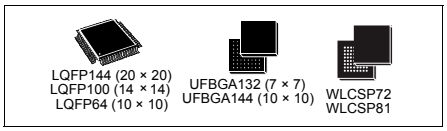
STM32L476xx

Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, ext. SMPS

Datasheet - production data

Features

- Ultra-low-power with FlexPowerControl
  - 1.71 V to 3.6 V power supply
  - -40 °C to 85/105/125 °C temperature range
  - 300 nA in V_{BAT} mode: supply for RTC and 32x32-bit backup registers
  - 30 nA Shutdown mode (5 wakeup pins)
  - 120 nA Standby mode (5 wakeup pins)
  - 420 nA Standby mode with RTC
  - 1.1 µA Stop 2 mode, 1.4 µA with RTC
  - 100 µA/MHz run mode (LDO Mode)
  - 39 µA/MHz run mode (@3.3 V SMPS Mode)
  - Batch acquisition mode (BAM)
  - 4 µs wakeup from Stop mode
  - Brown out reset (BOR)
  - Interconnect matrix
- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait-state execution from Flash memory, frequency up to 80 MHz, MPU, 100DMIPS and DSP instructions
- Performance benchmark
  - 1.25 DMIPS/MHz (Drystone 2.1)
  - 273.55 CoreMark® (3.42 CoreMark/MHz @ 80 MHz)
- Energy benchmark
  - 294 ULPMark™ CP score
  - 106 ULPMark™ PP score
- Clock Sources
  - 4 to 48 MHz crystal oscillator
  - 32 kHz crystal oscillator for RTC (LSE)
  - Internal 16 MHz factory-trimmed RC (±1%)
  - Internal low-power 32 kHz RC (±5%)
  - Internal multispeed 100 kHz to 48 MHz oscillator, auto-trimmed by LSE (better than ±0.25 % accuracy)
  - 3 PLLs for system clock, USB, audio, ADC



- Up to 114 fast I/Os, most 5 V-tolerant, up to 14 I/Os with independent supply down to 1.08 V
- RTC with HW calendar, alarms and calibration
- LCD 8× 40 or 4× 44 with step-up converter
- Up to 24 capacitive sensing channels: support touchkey, linear and rotary touch sensors
- 16x timers: 2x 16-bit advanced motor-control, 2x 32-bit and 5x 16-bit general purpose, 2x 16-bit basic, 2x low-power 16-bit timers (available in Stop mode), 2x watchdogs, SysTick timer
- Memories
  - Up to 1 MB Flash, 2 banks read-while-write, proprietary code readout protection
  - Up to 128 KB of SRAM including 32 KB with hardware parity check
  - External memory interface for static memories supporting SRAM, PSRAM, NOR and NAND memories
  - Quad SPI memory interface
- 4x digital filters for sigma delta modulator
- Rich analog peripherals (independent supply)
  - 3x 12-bit ADC 5 Msps, up to 16-bit with hardware oversampling, 200 µA/Msps
  - 2x 12-bit DAC output channels, low-power sample and hold
  - 2x operational amplifiers with built-in PGA
  - 2x ultra-low-power comparators
- 20x communication interfaces
  - USB OTG 2.0 full-speed, LPM and BCD
  - 2x SAIs (serial audio interface)
  - 3x I2C FM+(1 Mbit/s), SMBus/PMBus
  - 5x USARTs (ISO 7816, LIN, IrDA, modem)
  - 1x LPUART (Stop 2 wake-up)

June 2019 DS10198 Rev 8 1/270

This is information on a product in full production. www.st.com

B. Arduino program

```
#include <SimpleFOC.h>

HardwareSerial Serial3(PC5, PC4); //Rx, Tx

const byte numChars = 6;
uint8_t receivedChars[numChars];   // an array to store the received data
byte numReceived = 0;
boolean newData = false;

float dataNumber = 0;             // new for this version


// init BLDC motor
BLDCMotor motor = BLDCMotor( 14 );

BLDCDriver3PWM driver = BLDCDriver3PWM(PA8, PA9, PA10, PC7);
// init encoder
Encoder encoder = Encoder(PB4, PB5, 2048);
// channel A and B callbacks
void doA(){encoder.handleA();}
void doB(){encoder.handleB();}

void setup()
{

  // initialize motor encoder hardware
  encoder.init();
  encoder.enableInterrupts(doA,doB);

  // driver config
  driver.voltage_power_supply = 12;
  driver.init();

  // set control loop type to be used
  motor.torque_controller = TorqueControlType::voltage;
  motor.controller = MotionControlType::torque;


  // link the motor to the encoder
  motor.linkSensor(&encoder);
  // link the motor to the driver
  motor.linkDriver(&driver);

  Serial.begin(115200);
  Serial3.begin(115200);

  // initialize motor
  motor.init();
  // align encoder and start FOC
  motor.initFOC();
}

void loop()
{
  recvWithStartEndMarker();
  showNewNumber();

  motor.loopFOC();
  // pendulum sensor read
  pendulum.update();

  motor.move(dataNumber);
}


void recvWithStartEndMarker() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    uint8_t startMarker = 100;
    uint8_t endMarker = 200;
    uint8_t rc;
```

```
72
73      while (Serial3.available() > 0 && newData == false) {
74          rc = Serial3.read();
75
76          if (recvInProgress == true) {
77              if (rc != endMarker) {
78                  receivedChars[ndx] = rc;
79                  ndx++;
80                  if (ndx >= numChars) {
81                      ndx = numChars - 1;
82                  }
83              }
84              else {
85                  receivedChars[ndx] = '\0'; // terminate the string
86                  recvInProgress = false;
87                  numReceived = ndx;
88                  ndx = 0;
89                  newData = true;
90              }
91          }
92
93          else if (rc == startMarker) {
94              recvInProgress = true;
95          }
96      }
97  }
98
99  void showNewNumber() {
100     if (newData == true) {
101         dataNumber = 0;
102         dataNumber = (receivedChars[0] | receivedChars[1] << 8) - 12;
103
104         Serial.print("This␣just␣in␣...␣");
105         for (byte n = 0; n < numReceived; n++) {
106             Serial.print(receivedChars[n]);
107             Serial.print('␣');
108         }
109         Serial.println();
110         Serial.print("Data␣as␣Number␣...␣");
111         Serial.println(dataNumber);
112         newData = false;
113     }
114 }
```

C. S-function builder for encoder

```
#include <math.h>
#ifndef MATLAB_MEX_FILE
#include <stm32l4xx_hal.h>

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;

void Error_Handler(void)
{
    __disable_irq();
    while (1)
    {
    }
}

void HAL_MspInit(void)
{
    __HAL_RCC_SYSCFG_CLK_ENABLE();
    __HAL_RCC_PWR_CLK_ENABLE();
}

void HAL_TIM_Encoder_MspInit(TIM_HandleTypeDef* htim_encoder)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(htim_encoder->Instance==TIM2)
  {
    __HAL_RCC_TIM2_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  }
  else if(htim_encoder->Instance==TIM3)
  {
    __HAL_RCC_TIM3_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF2_TIM3;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  }

}

void HAL_TIM_Encoder_MspDeInit(TIM_HandleTypeDef* htim_encoder)
{
    if(htim_encoder->Instance==TIM2)
  {
    __HAL_RCC_TIM2_CLK_DISABLE();
    HA_GPIO_DeInit(GPIOA, GPIO_PIN_0|GPIO_PIN_1);
  }
  else if(htim_encoder->Instance==TIM3)
  {
    __HAL_RCC_TIM3_CLK_DISABLE();
    HAL_GPIO_DeInit(GPIOB, GPIO_PIN_4|GPIO_PIN_5);
  }
}

static void MX_TIM2_Init(void)
{
  TIM_Encoder_InitTypeDef sConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 3;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
```

```
72    htim2.Init.Period = 2048;
73    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
74    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
75    sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
76    sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
77    sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
78    sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
79    sConfig.IC1Filter = 0;
80    sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
81    sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
82    sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
83    sConfig.IC2Filter = 0;
84    if (HAL_TIM_Encoder_Init(&htim2, &sConfig) != HAL_OK)
85    {
86      Error_Handler();
87    }
88    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
89    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
90    if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
91    {
92      Error_Handler();
93    }
94  }
95
96  static void MX_TIM3_Init(void)
97  {
98      TIM_Encoder_InitTypeDef sConfig = {0};
99      TIM_MasterConfigTypeDef sMasterConfig = {0};
100
101     htim3.Instance = TIM3;
102     htim3.Init.Prescaler = 3;
103     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
104     htim3.Init.Period = 2048;
105     htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
106     htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
107     sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
108     sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
109     sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
110     sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
111     sConfig.IC1Filter = 0;
112     sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
113     sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
114     sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
115     sConfig.IC2Filter = 0;
116     if (HAL_TIM_Encoder_Init(&htim3, &sConfig) != HAL_OK)
117     {
118     Error_Handler();
119     }
120     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
121     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
122     if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
123     {
124     Error_Handler();
125     }
126
127 }
128
129 void readEnc_initialize(void)
130 {
131   HAL_TIM_Encoder_Start(&htim2,TIM_CHANNEL_ALL);
132   HAL_TIM_Encoder_Start(&htim3,TIM_CHANNEL_ALL);
133 }
134 #endif
135
136 void readEnc_obs_Start_wrapper(real_T *xD)
137 {
138 }
139
140 void readEnc_obs_Outputs_wrapper(uint16_T *tim2_cnt,
141                                  uint16_T *tim2_dir,
142                                  real_T *tim3_cnt,
143                                  real_T *tim3_dir,
144                                  const real_T *xD)
```

```
145 {
146     if(xD[0]==1){
147         #ifndef MATLAB_MEX_FILE
148             tim2_cnt[0] = __HAL_TIM_GET_COUNTER(&htim2);
149             tim2_dir[0] = __HAL_TIM_IS_TIM_COUNTING_DOWN(&htim2);
150             tim3_cnt[0] = __HAL_TIM_GET_COUNTER(&htim3);
151             tim3_dir[0] = __HAL_TIM_IS_TIM_COUNTING_DOWN(&htim3);
152         #endif
153     }
154 }
155
156 void readEnc_obs_Update_wrapper(uint16_T *tim2_cnt,
157                                 uint16_T *tim2_dir,
158                                 real_T *tim3_cnt,
159                                 real_T *tim3_dir,
160                                 real_T *xD)
161 {
162     if(xD[0]!=1){
163         #ifndef MATLAB_MEX_FILE
164             MX_TIM2_Init();
165             MX_TIM3_Init();
166             readEnc_initialize();
167          #endif
168     }
169     xD[0]=1;
170 }
171
172 void readEnc_obs_Terminate_wrapper(real_T *xD)
173 {
174 }
```