applied sciences

MDPI

Article

An Open-Source Software Framework for Direct Field-Oriented Control of a BLDC with Only One Sensor for ARM

Radu Bogdan Sabau [1,2] and Radu Etz [1,2,*]

[1] Applied Electronics Department, Technical University of Cluj-Napoca, 400114 Cluj-Napoca, Romania; sabau.gh.radu@student.utcluj.ro
[2] European University of Technology, European Union
* Correspondence: radu.etz@ael.utcluj.ro

Abstract

This paper introduces an open-source software framework for implementing Field-Oriented Control (FOC) on a Brushless DC Motor (BLDC) across its entire speed range. The control strategy employs a Direct FOC method with a single Hall sensor combined with Space Vector Pulse Width Modulation (SVPWM) and complementary sensorless techniques. The BLDC motor and supporting circuits are modeled and validated through both simulation and hardware implementation. A modular software architecture enables deployment via distinct system components, promoting hardware abstraction and reducing platform-specific dependencies. The entire setup is conceptualized and executed in MATLAB/Simulink R2024b and the framework supports remote experimentation through a web-based interface, requiring only a single MATLAB license. This scalable solution is designed for academic researchers and industry practitioners alike, offering an accessible low-cost platform for motor control development, validation, and early-stage prototyping.

Keywords: field-oriented control; space vector pulse width modulation; brushless DC motor; direct Clarke transform; direct Park transform; inverse Park transform

Academic Editors: Stijn Derammelaere and Amélie Chevalier

Received: 9 September 2025
Revised: 6 October 2025
Accepted: 10 October 2025
Published: 14 October 2025

Citation: Sabau, R.B.; Etz, R. An Open-Source Software Framework for Direct Field-Oriented Control of a BLDC with Only One Sensor for ARM. *Appl. Sci.* **2025**, *15*, 11018. https://doi.org/10.3390/app152011018

1. Introduction

Brushless DC motors [1] are synchronous machines that use a direct current power supply alongside an electronic controller for switching the DC currents to the motor windings, resulting in highly efficient controlled machines. The field winding is supplied from a rotating excitation system, typically referred to as a brushless excitation system. It is advantageous to have the single low-power field winding on the rotor while having the high-power (typically multiple-phase) armature winding on the stator [2].

Most common methods for controlling synchronous machines consist of Variable-Frequency Drive (VFD) [3] control methods such as Direct Torque Control (DTC) [4] or FOC [5]. The DTC can be either classic or space vector modulation-based [6]. On the other hand, Field-Oriented Control (FOC) can result in Direct FOC (DFOC) or Indirect FOC (IFOC); DFOC uses sensors to directly measure rotor position and flux, while IFOC estimates these values, potentially eliminating sensors but adding computational complexity. There are additional methods for controlling a synchronous machine, such as a Model Predictive Control [7]; these use a different kind of controllers than PID ones, implying even more computational complexity.

FOC, also known as vector control, has emerged as a powerful technique for achieving high-performance control of three-phase AC machines, including BLDC motors [8,9]. FOC

enables decoupled control of torque and flux by transforming stator currents into a rotating d–q reference frame aligned with the rotor magnetic field [10]. This transformation allows the torque (q-axis) and flux (d-axis) components to be regulated independently, effectively emulating the behavior of a separately excited DC motor [11]. Such decoupling enables linear control characteristics and rapid dynamic response capabilities that are unattainable with traditional scalar (or "V/f") control methods [12,13].

Although originally developed for permanent-magnet synchronous machines (PMSMs) [14] and induction motors, the principles of FOC can be effectively adapted to BLDC motors. This adaptation enhances torque smoothness and reduces acoustic noise, making FOC particularly advantageous in applications demanding superior performance metrics [15].

To achieve such control of the motor, rotor position knowledge is crucial for effective operation [16]. This task is accomplished by sensors such as Hall sensors, optical sensors, or other solutions used in industrial applications where high performance and accuracy are of significant importance; alternatively, sensorless control techniques use back-Electromotive Force (EMF) or saliency-based control.

Recent BLDC control strategies span a wide spectrum of complexity, sensor dependency, and implementation accessibility. While traditional FOC offers high precision, it demands multiple sensors and significant computational resources [17]. Sensorless FOC reduces hardware cost, but suffers from unreliable low-speed performance due to complex rotor estimation. Advanced methods such as DTC, MPC, and AI-based control provide fast response and adaptability, but are computationally intensive and rarely available in open-source platforms. In contrast, the proposed framework offers a balanced solution that is moderate in complexity, reliable at startup, and designed for rapid prototyping with minimal sensor requirements. This positions it as a practical alternative for embedded and low-cost applications where simplicity and scalability are prioritized over high-end precision.

This paper investigates the working of a BLDC motor in the case of direct FOC with minimal use of sensors. By using integrated circuits that help to compute the needed components for proper control of the BLDC, the proposed approach reduces disadvantages and cost while keeping performance as high as possible. In addition, we present an application that aids in understanding of how FOC works for BLDCs by providing a comparison between a simulation using an ideal sensor and a real-world implementation using a non-ideal sensor.

The rest of this article is organized as follows: the Field-Oriented Control (FOC) method and model representation used in this paper are presented in Section 2; in Section 3 explains the software and hardware implementation; Section 4 provides the model-in-the-loop implementation and interface development; Section 5 reports the results of our simulation and experiments; lastly, Section 6 concludes the paper.

2. Field-Oriented Control Method and Model Representation

The proposed FOC method takes the direct FOC approach and uses only a Hall Sensor to obtain the rotor angle value, which is connected to an external Integrated Circuit (IC) used to compute the speed of the rotor. The rotor angle is used in the transforms that take place inside the FOC for determining direct and quadrature components used for the flux and torque controllers, respectively. The direct Clarke, direct Park, inverse Park, and space vector pulse width modulation techniques and transforms are used for torque and flux control. The inverse and direct Park transforms use the electrical angle of the rotor and consider the motor pole–pair number for correct computation, as shown in Figure 1.

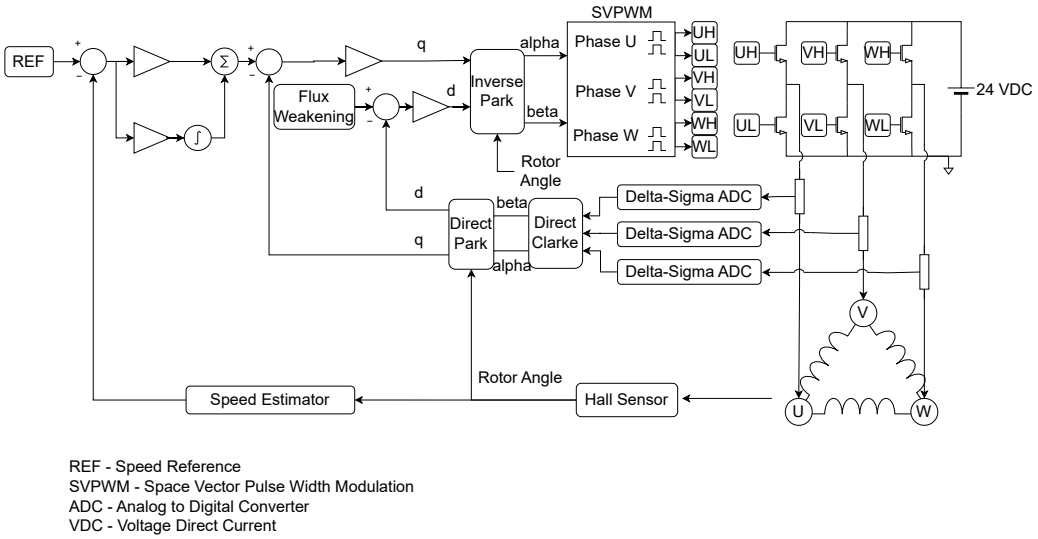

Figure 1. Block diagram of the proposed model.

As shown in the figure, the PI controller components, error computation for their inputs, and reference values are referred to as the control element of the proposed system, meaning that the d and q components are referred to as the control signals. A PI structure is used for speed control, as described in Equation (1). A P structure is used for the torque and flux controllers, as shown in Equations (2) and (3) below.

$$u_s(k) = [K_p + \frac{K_i * T_i * z}{z-1}] * e_s(k) \quad (1)$$

$$u_q(k) = K_p * e_q(k) \quad (2)$$

$$u_d(k) = K_p * e_d(k) \quad (3)$$

In our case, both execution and output signals are fed back to the loop, taking into consideration that execution signals that are the phase currents of the motor help in computing the direct and quadrature components for the controllers alongside the angle in the case of flux and torque, while the speed signal assists the speed controller. In this case, the previously mentioned Hall sensor helps in determining the angle, and consequently the speed. Thus, the element including the Hall sensor and speed calculator is referred to as the transducer element, and its output signals are used for error computation inside the control element, thereby closing the loops.

It can be observed that the speed loop is cascaded with the two parallel loops for the flux and torque, since the output of the speed PI controller serves as the reference to the torque controller. This promotes efficiency, as the optimal required torque is produced for the required speed value. For the flux, a weakening value is used as a reference in order to consume as little energy as possible.

In order to properly test and simulate the proposed model, MATLAB/Simulink software was used alongside Simscape libraries. The first block to be considered is the BLDC, which is parameterized using the motor's technical specifications in order to respect similarities between simulation and experimental results. The parameters provided by the datasheet can be found in Table 1.

Other parameters needed for modeling the BLDC are computed by using the datasheet [18] given ones as per Equations (4) and (5), and shown in Table 2.

$$\theta_F = \frac{\theta_H}{p} \quad (4)$$

$$L_d = L_q = \frac{L_{l-l}}{2} \quad (5)$$

Table 1. Datasheet BLDC parameters.

Parameter	Value	Symbol	Unit
Hall Effect Angle	120	Θ_H	Degree
Maximum rotor-induced back EMF	24	-	V
Number of pole pairs	4	p	-
Winding type	Delta	-	-
Line to Line Inductance	1.2	L_{l-l}	mH
Stator resistance per phase	0.72	R_s	Ω
Motor Torque Constant	0.036	K_t	$\frac{Nm}{A}$
Rotor Inertia	48×10^{-6}	J	$kg * m^2$

Table 2. Determined BLDC parameters.

Parameter	Value	Symbol	Unit
Rotor angle over for constant back EMF	30	Θ_F	Degree
Stator d-axis inductance	0.6	L_d	mH
Stator q-axis inductance	0.6	L_q	mH

Figure 2 illustrates the complete control and execution architecture of the proposed embedded motor drive system implemented for real-time control of a three-phase inverter-fed electric machine. The diagram is organized into hierarchical blocks representing the flow of signals and operations across the system's digital and power domains. Each functional block is modularly defined to reflect its role in the closed-loop control loop.

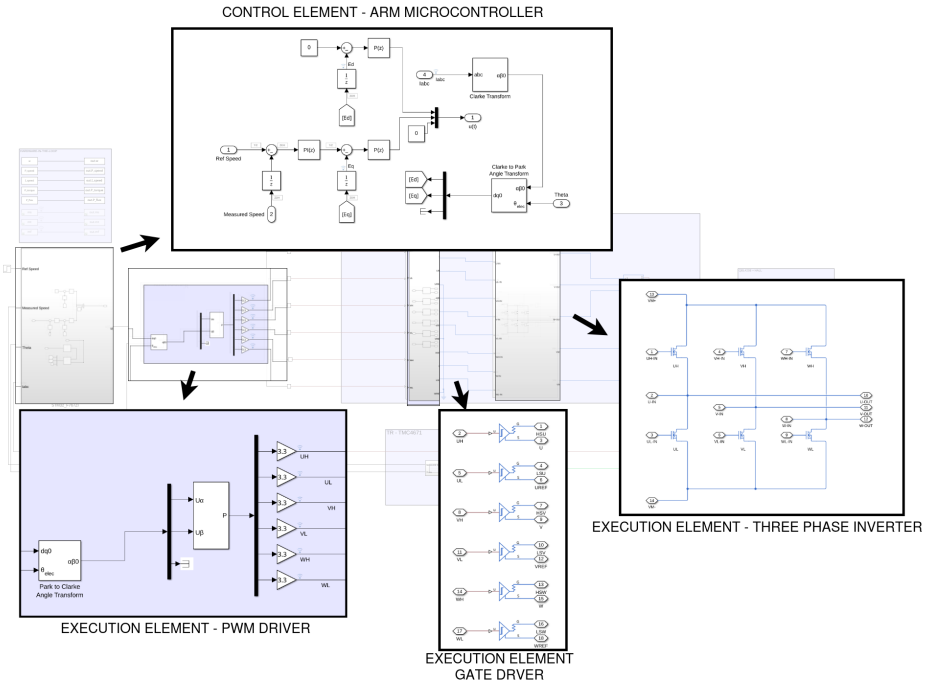

Figure 2. Control and execution element of the model.

At the top of the figure, the control element is implemented on an ARM-based microcontroller, which executes the core algorithm responsible for speed and current regulation. This subsystem includes standard FOC transformations such as Clarke and Park transforms

along with PI controllers for both the torque-producing current and the rotor speed. The processed control signals are converted into duty cycles and transmitted to the downstream execution chain as PWM references.

The PWM driver execution element receives these control signals and formats them into high-resolution pulse-width modulated signals. This module is critical in translating numerical control values into physical switching commands, operating with precise timing to maintain waveform integrity under high-frequency operation.

These PWM signals are passed to the gate driver block, which forms the interface between the low-power control logic and the high-power switching elements. The gate driver provides the electrical isolation and level shifting necessary for reliable operation of the inverter's semiconductor devices. It ensures appropriate gate charge delivery to the power switches, enabling safe and efficient switching.

At the final stage, the three-phase inverter execution element receives the gate signals and synthesizes the desired AC output from a DC supply. The inverter employs a standard six-switch topology to generate three-phase voltage waveforms corresponding to the reference signals provided by the controller. This output drives the connected machine, closing the control loop.

The arrows denote the unidirectional flow of information and control, from algorithmic computation to physical actuation. The modular structure of the system enables scalable deployment across various hardware configurations and facilitates Model-in-the-Loop (MIL) testing for validation.

This architectural decomposition demonstrates a tightly integrated hardware-software system, balancing control fidelity with execution speed. It enables deterministic operation and supports the rigorous timing requirements of embedded motor control applications such as electric propulsion, robotics, and power conversion systems.

3. Hardware Approach and Implementation

The chosen hardware parts for the physical model implementation are as follows:

- Controllers and direct transforms are implemented on an ARM based microcontroller.
- A specific IC is used for the PWM engine (part of the execution element) as well as the Hall sensor interpreter and current ADCs (part of the transducer element).
- For the gate driver and three-phase inverter used to drive the motor, another specific IC evaluation board is used (part of the execution element).
- For the motor, we use a brushless DC motor (process) incorporating a Hall sensor.

The hardware setup diagram, which implies the two previously-mentioned ICs along with the BLDC, ARM microcontroller, and Hall sensor, is shown in Figure 3.

For enhanced system performance, the angle value needs to be obtained quickly enough to transmit the correct α and β inputs to the SVPWM, although the D and Q components do not change in that time. Therefore, a single IC is used for both the PWM engine, which is part of the execution element, and the transducer element. By using this IC instead of the microcontroller's PWM channels, we ensure correct functionality of the motor regardless of the microcontroller's performance in terms of computation speed. The Hall sensor signal interpreter which estimates the mechanical angle of the rotor is incorporated into the IC, and the value is multiplied by the number of pole pairs from the previously-configured internal register in order to compute the electrical angle of the rotor. This value is then transmitted directly to the inverse Park transform inside the IC, ensuring a smooth operation of the BLDC.

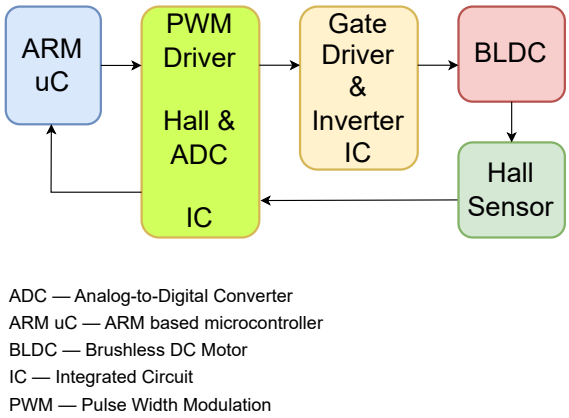

Figure 3. Hardware setup diagram.

The loops involved are the torque and flux, both of which work in parallel, and the speed loop (the so-called outer loop), which is cascaded with the other two. The loop times are very important, having a major impact on both the dynamic and static performance characteristics of the system, such as response time and steady-state error, respectively.

4. Software Infrastructure

An ARM architecture-based microcontroller is used as the control element, and uses the software infrastructure shown in Figure 4.

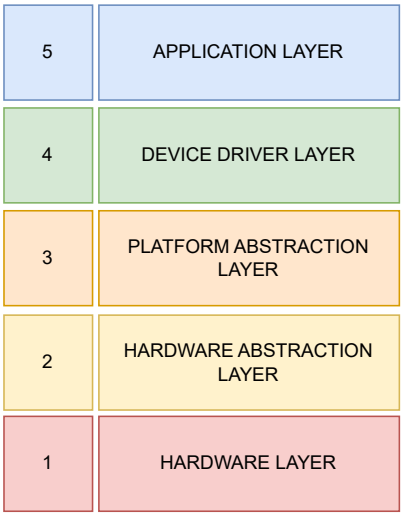

Figure 4. Software infrastructure.

The foundation is the hardware layer, consisting of microcontroller soft cores and their on-chip peripherals (timers, General Purpose Input/Output (GPIO), Universal Asynchronous Receiver/Transmitter (UART), Direct Memory Access (DMA)) as well as external components such as ADCs, gate drivers, and position sensors. For motor control systems, this layer is responsible for acquiring phase current and rotor position measurements and for driving the inverter stage through PWM outputs.

Directly above this lies the vendor Software Development Kit (SDK)/Board Support Package (BSP) layer, which provides the first level of software interaction with the hardware. This includes startup code, board initialization, and Hardware Abstraction Libraries (HALs) for peripherals such as Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I^2C), UART, and timer modules.

To ensure cross-platform portability, the platform abstraction layer defines a uniform API that normalizes these vendor differences. Functions such as SPI writing-then-reading and GPIO value update setting act as canonical entry points, allowing higher layers to remain independent of the underlying processor family. For a motor control system, this abstraction enables the same communication protocol driver code for the two used ICs to operate seamlessly on different processing platforms.

The device driver layer encapsulates the logic required to interface the previously-integrated circuits and their functionalities, such as ADC configuration for current sensing, PWM generation, and gate-driver setup. These drivers implement register maps, initialization sequences, calibration procedures, and data acquisition routines, typically relying on SPI or I^2C communication provided by the lower layers.

At the top is the application layer, where motor control algorithms are realized; this includes the implementation of FOC with its speed, torque, and flux regulation loops. The application layer integrates device drivers and platform services into a complete control solution while remaining agnostic to the underlying processing unit thanks to the layered design.

This structure not only promotes portability across heterogeneous platforms but also isolates time-critical motor control tasks from vendor-specific dependencies, enabling efficient reuse of algorithms and drivers across different hardware generations.

In this case, we used the IC's maximum SPI frequency of 40 MHz, meaning that each bit of information alongside the communication lanes takes exactly 0.025 µs to transmit, as shown in Figure 5.

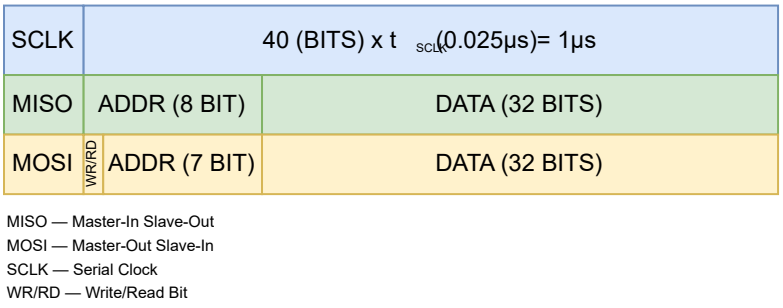

Figure 5. SPI datagram.

Taking into account that the register address is byte-sized and the register value is 32-bit sized, this means that the whole transfer consists of 40 bits; thus, a single SPI transfer requires precisely 1 µs. In our application, we perform a total of four readings: one each for the angle, speed, U and W currents, and V current. This results in a total SPI read time of 4 µs. For writing of the D and Q control signals to the execution element, we perform two writings, one for each of the D and Q components. In this way, we obtain a total SPI write time of 2 µs.

Considering that all three loops use controllers that are implemented inside a microcontroller with an ARM microprocessor, certain limitations appear when it comes to time and resources. These limitations are due to factors such as the SPI timing, software infrastructure's implication of abstraction layers, and calculus of transforms such as the direct Park and direct Clarke transforms within the microcontroller. All of these factors result in a 100 µs loop time, as shown in Figure 6.

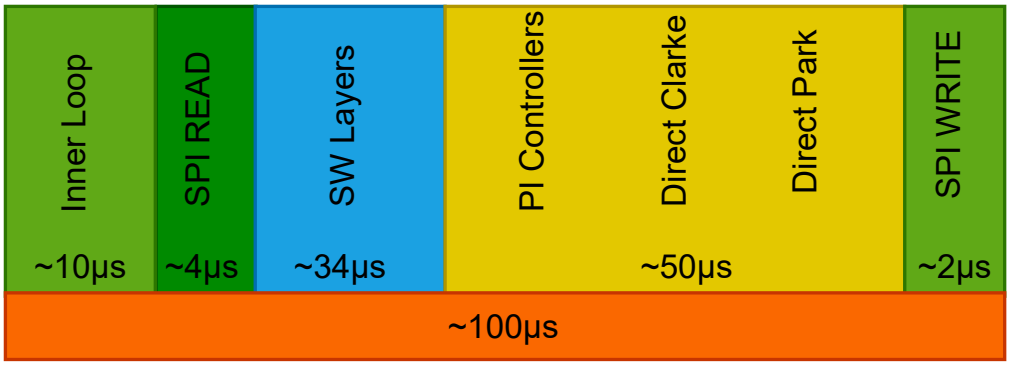

Figure 6. Loop timing.

The motor control software is organized into modules aligned with the layered architecture. The communication module forms part of the platform abstraction layer, standardizing access to peripheral interfaces such as SPI and UART while isolating higher-level modules from platform-specific details.

The device driver modules occupy the device driver layer, managing low-level interactions with the microcontroller's peripherals that interact with the hardware components critical for FOC, including ADCs for phase current sensing, PWM modules for inverter control, and position sensors.

The control software modules form the application layer. They implement the main FOC control loop, including Proportional-Integral (PI) controllers for torque and speed regulation and Clarke and Park transforms for converting three-phase currents into the d–q reference frame. This modular separation ensures maintainability, facilitates reuse across different motor types, and allows the control logic to operate independently of the underlying hardware.

Figure 7 illustrates these modules and their interactions, highlighting the flow of data from device drivers through coordinate transformations and control algorithms to the PWM outputs driving the three-phase inverter.

The Main Control Loop (Listing 1) firmware module begins by initializing the communication interface (UART) and the two mentioned ICs (IC1—PWM generator and Hall interpreter; IC2—Gate Driver), with error handling to safely terminate and release resources if initialization fails. The motor controller is then configured, including PWM setup and closed-loop control preparation.

The PI controllers for speed, torque, and flux are initialized alongside the Clarke and Park transforms to implement Field-Oriented Control (FOC). At this point, a reference speed is defined and the motor is started.

The closed-loop control executes with a 100 µs time step over 1 s. At each iteration, the motor speed, torque, flux, and rotor angle are measured. The speed error is processed by a PI controller to compute the torque demand, while the torque and flux errors are regulated through corresponding PI controllers. The resulting voltage commands are applied to the motor, with data logged at every 1 ms.

After the test time, the motor is safely stopped, the logged data are transmitted via UART, and all drivers and resources are released to ensure safe system shutdown.

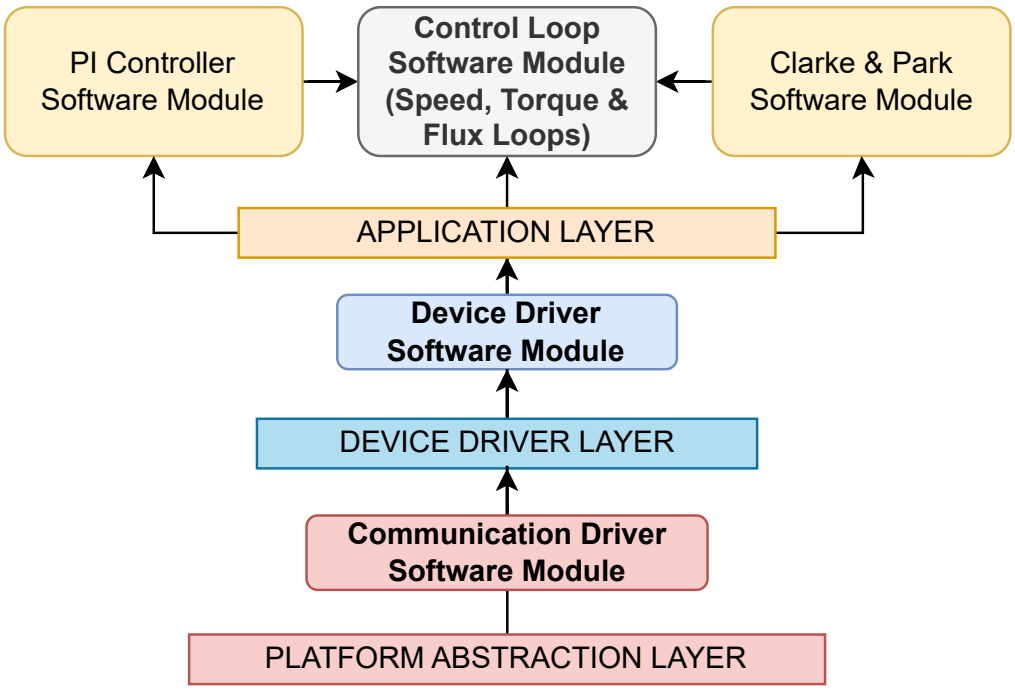

Figure 7. Developed and used software modules.

Listing 1. Main control loop.

```
START
/* platform driver module */
INITIALIZE_UART()
/* platform driver module */
INITIALIZE_IC1_AND_IC2()
/* pi controller module*/
INITIALIZE_PI_CONTROLLERS()
/* clarke & park module */
INITIALIZE_CLARKE_AND_PARK()
/* device driver module */
START_MOTOR()
REPEAT for 1s (with 100µs step):
    /* device driver module */
    MEASURE motor_speed_torque_and_flux
    MEASURE rotor_angle
    /* clarke & park module */
    CALCULATE PI_torque_and_flux_referrences
    /* pi controller module */
    CALCULATE PI_for_speed_torque_and_flux
    CALCULATE torque_and_flux_commands
    APPLY commands_to_motor
    LOG data every 1ms
END REPEAT
STOP_MOTOR()
SEND_LOG_DATA_OVER_UART()
REMOVE_DRIVERS()
END
```

5. Model-in-the-Loop and Interface Development

In most cases, an MIL implementation is used for a better comparison, and sometimes even to link between simulation and experimental results. Here, the proposed implementation is performed using Simulink's support for developing a Web Application (Web App) which is to be linked with the previously mentioned software algorithm, which assures a better perspective on hardware and simulation data while providing a way of comparing the two sets of signals. The diagram of the proposed model for the whole application is shown in Figure 8.

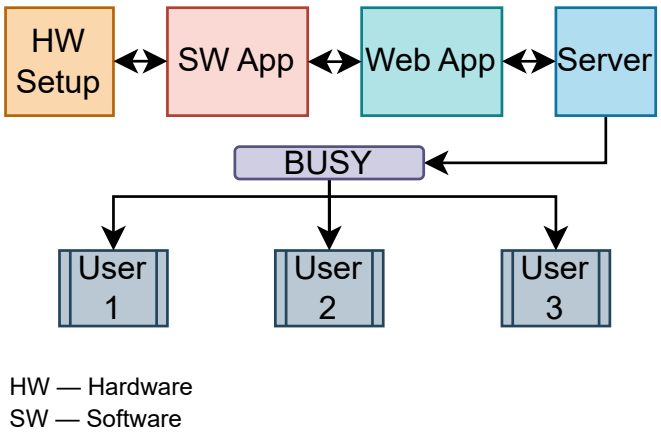

Figure 8. User application's general diagram.

The framework is built with modularity and scalability in mind, allowing users to adapt it to different motor ratings, power levels, and converter switching frequencies. Motor-specific parameters such as rated speed and voltage along with current limits are configurable via backend scripts. A hardware abstraction layer further decouples the control logic from platform-specific dependencies, enabling portability across microcontroller platforms and converter topologies. This design supports flexible deployment in both educational and industrial contexts.

The way the Web App works is that inside the software algorithm after BLDC has run for about 1 s (measured by assuring a total of 10,000 loops, each taking 100 µs), 1000 values of the speed, flux, and torque signals are logged approximately 1 ms apart from each other. Therefore, 1000 points of each signal previously mentioned are transmitted over UART to a server (which is connected to the microcontroller) in order for it to process, log, and plot the data for a 1 s run time. By transmitting the index (ranging from 0 to 999), speed, torque, and flux decimal-coded values on the serial port, the Web App accesses the terminal that captures the transmitted UART signals. The values are then parsed into a text file that is later processed by the internal MATLAB script inside the Web App in order to plot the hardware signals.

The way the Web App's user interface works is that the user is able to change the reference and PI controller values, after which the simulation and hardware models have to be run separately by pressing the corresponding button.

When the button of the corresponding hardware model is clicked, the script starts to build the program with user-specified data, uploads the binary file to the microcontroller, and runs the program. While the program runs, the MATLAB script waits for 10 s for the program to finish running; at the same time, the serial port is opened and waits for values to be captured.

The data are then read back, processed, and organized into timetables, thereby plotting each signal of the hardware setup. The interface enables the user to set a reference speed in Revolutions Per Minute (RPM), adjust the proportional and integral gains for the speed,

torque, and flux control loops, and control the motor operation through dedicated buttons, as shown in Figure 9.

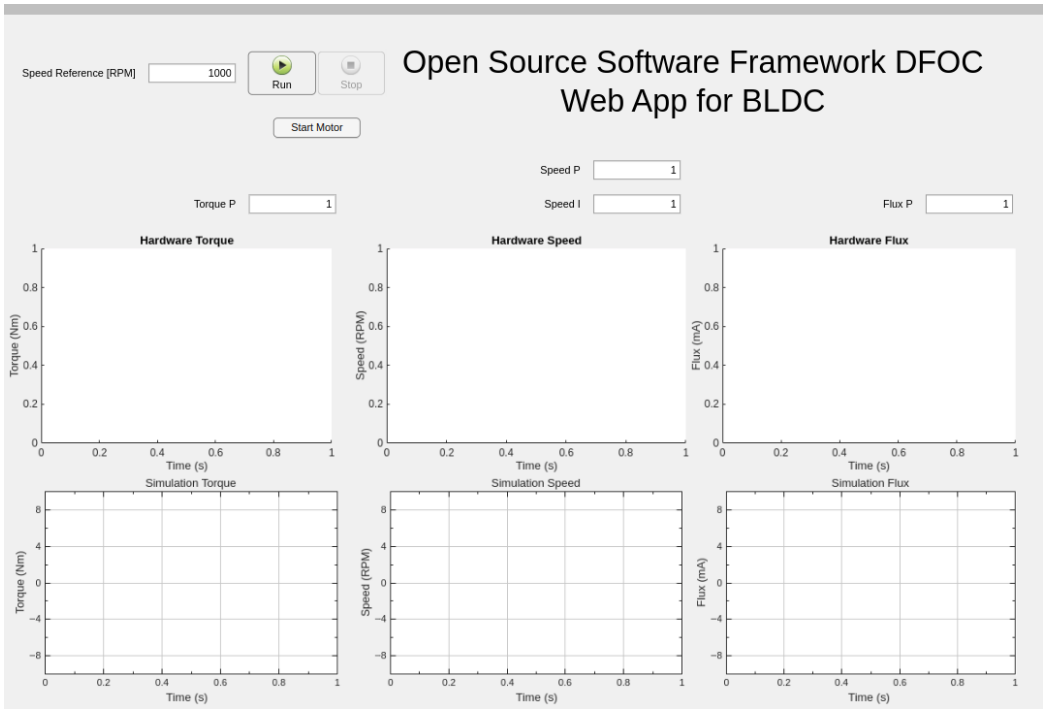

Figure 9. lWeb application's graphical user interface.

The top section of the GUI provides input fields for controller parameters and operational commands. The lower portion displays six real-time plots that visualize both hardware and simulation data. The first row presents hardware-acquired measurements of the torque, speed, and flux, while the second row displays the corresponding simulation results. Each plot maps amplitude versus time, enabling direct comparison between real-world behavior and simulated performance. This GUI environment supports interactive tuning and evaluation of control strategies, serving as a valuable tool for both development and experimental validation of FOC in BLDC motor applications.

The user can choose to start the physical model, which is handled by the MATLAB R2024b script that accesses the server's terminal in order to modify, compile, and re-upload the source code, with the data expected to be transmitted on the serial port at the end and then processed and plotted. If the simulation model is chosen, the simulation starts with the set parameters using the integrated Simulink model.

In the end, the application can be deployed by using the MATLAB Web Server application, which allows for specific Web Apps to be deployed on a server and grants remote access to the application. Therefore multiple users can access the same model to view the data, although only one user at a time can start or stop the simulation/hardware runs. While MATLAB is a licensed platform, the system architecture is designed such that only a single host machine requires a MATLAB license. Users can access the GUI remotely via a web interface, allowing them to test and deploy control algorithms on shared hardware without requiring individual MATLAB installations. This approach balances the robustness of MATLAB-based development with the accessibility of web-based deployment, enabling collaborative experimentation and reducing overall licensing costs. The Web App's internal structure is shown in Figure 10.

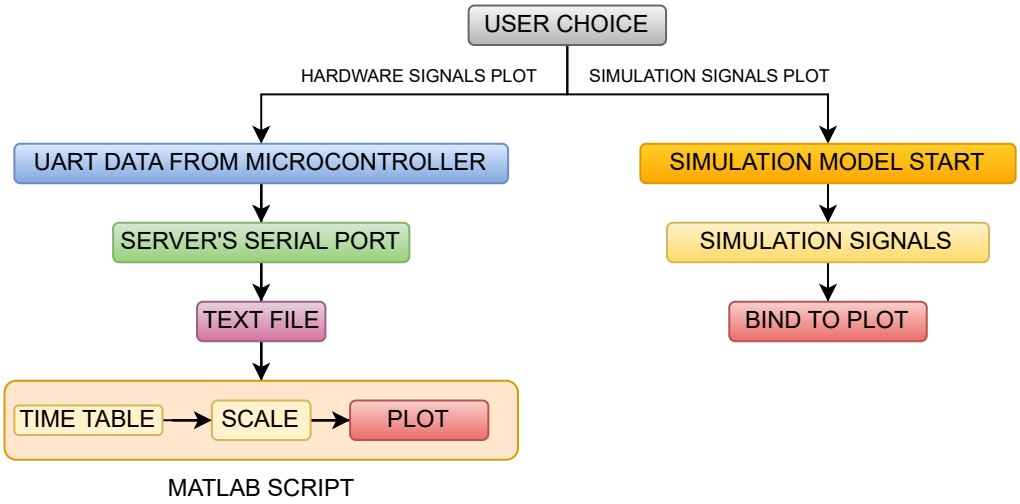

Figure 10. Web application's internal structure.

6. Simulation and Experimental Results

Figure 11 shows the physical model of the system used to test the whole application.

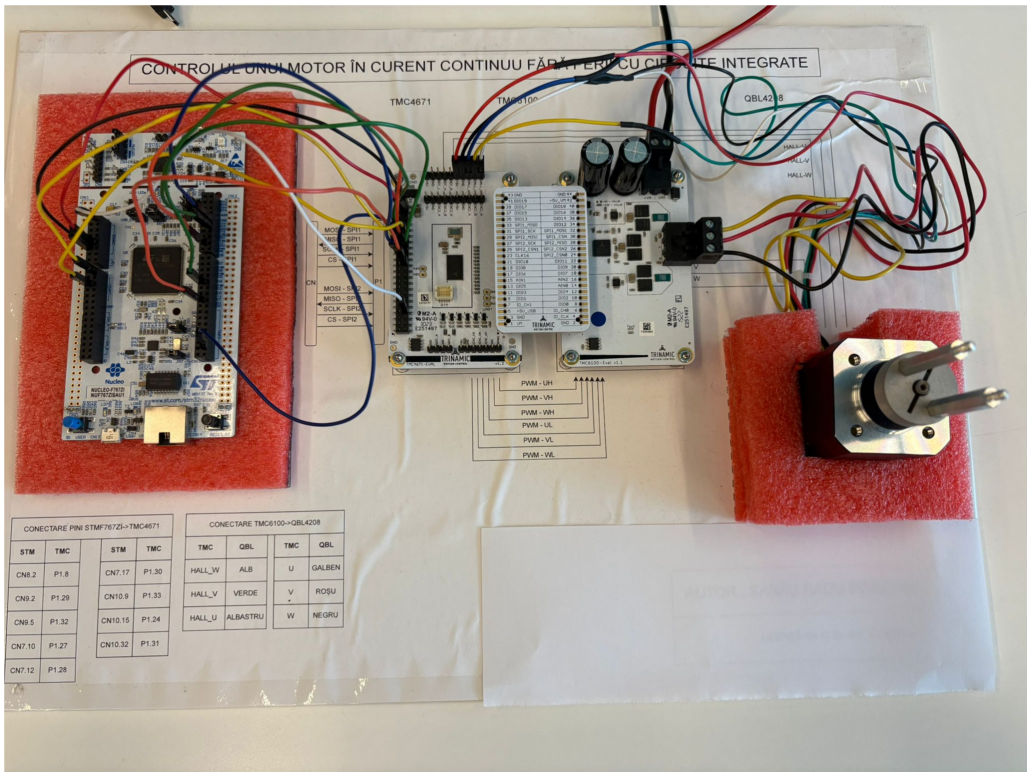

Figure 11. Physical model.

The parameters of the BLDC used for both simulation and experimental tests are described in Tables 1 and 2. The other parameters of the system, such as discretization time of the controllers (only for the speed controllers, since the torque and flux ones are proportional only, meaning that the discretization time can be irrelevant in this case), switching frequency, and other timing properties of the execution element can be found in Table 3.

Table 3. System parameters.

Parameter	Symbol	Value	Unit
Discretization Time	t_s	100	µs
Switching Frequency (SVPWM)	f_{SVPWM}	50	kHz
Gate Driver Propagation Delay	t_p	85	ns
Gate Driver Rise Time	t_{LH}	10	ns
Gate Driver Fall Time	t_{HL}	10	ns

To properly test the proposed system using a delta-wound BLDC motor, the proportional gains for torque (K_{pq}) and flux (K_{pd}) are assumed to be equal and derived as follows:

$$K_{pq} = \frac{L_s}{\tau}, \quad \tau = \frac{L_s}{R_s} \quad \Rightarrow \quad K_{pq} = K_{pd} = 0.72. \tag{6}$$

For the speed loop, a model-based tuning approach is employed using the motor's rotor inertia and torque constant. The controller crossover frequencies are defined as follows:

$$f_{ci} = \frac{f_{\text{PWM}}}{20}, \quad \omega_{ci} = 2\pi f_{ci} = 15.7\,\text{kHz}, \tag{7}$$

$$\omega_{cs} = \frac{\omega_{ci}}{20} = 0.785\,\text{kHz}. \tag{8}$$

The proportional and integral gains for the speed loop are calculated as follows:

$$K_{ps} = \frac{J \cdot \omega_{cs}}{K_t} \approx 1.04, \tag{9}$$

$$K_i = \frac{K_{ps} \cdot \omega_{cs}}{5} \approx 163, \tag{10}$$

$$I = \frac{1}{K_i} \approx 0.006. \tag{11}$$

The reference speed of 2000 RPM was selected based on practical and technical considerations relevant to typical BLDC motor applications. This speed lies within the common operating range of 1000–3000 RPM observed in devices such as fans, pumps, and small electric vehicles. Specifically, 2000 RPM serves as a representative midpoint, offering a realistic and steady-state operating condition that minimizes the influence of startup transients and high-speed nonlinearities. Moreover, it provides a balanced tradeoff between torque output and energy efficiency, making it a meaningful benchmark for evaluating the performance of the proposed control strategy.

For a speed reference value of 2000 RPM, a comparative analysis between the simulated and experimentally measured motor speeds was conducted using MIL implementation. The results are illustrated in Figure 12, which shows both the simulation output and hardware response under steady-state operating conditions.

The primary performance metric under consideration is the steady-state error variation, denoted as Δe_{ss}. This error represents the difference between the estimated motor speed and the desired reference value once transient effects have subsided. While small variations may persist due to hardware and sampling limitations, the outer PI controller eliminates the steady-state error in the control-theoretic sense. From the figure, it can be observed that the steady-state error for the hardware implementation is approximately 306 RPM, corresponding to 7.65% of the rated speed. In contrast, the simulation result exhibits a smaller error of 215 RPM, which equates to 5.37% of the rated speed.

This discrepancy highlights the inherent differences between the idealized simulation environment and the physical hardware, where factors such as inverter nonlinearities, sensor inaccuracies, and real-time processing delays contribute to increased deviations. Nevertheless, the close correlation between the two results validates the effectiveness of the proposed control strategy and underscores the utility of MIL as a robust testing methodology for control system development.

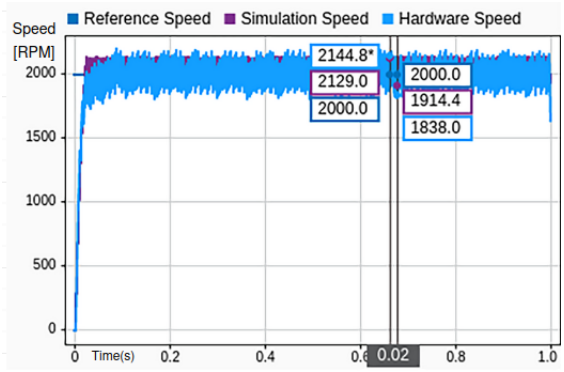

Figure 12. Simulation speed vs. hardware speed steady-state error variation for a 2000 RPM speed reference. ("*" indicating the maximum value).

In terms of dynamic performance, the transient response of the system to a speed reference of 2000 RPM is analyzed by comparing both the simulated and experimental (hardware) speed signals. The comparison is presented in Figure 13, where the rise time of each response is evaluated as an indicator of the system's responsiveness to reference changes.

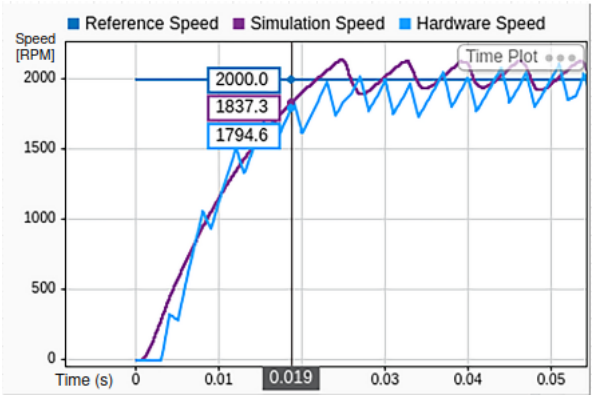

Figure 13. Simulation speed vs. hardware speed response time for a 2000 RPM speed reference.

It can be observed that both the simulation and hardware implementations achieve a response time of approximately 19 milliseconds. However, the simulation exhibits a slightly faster rise, reaching the target speed marginally earlier than the hardware system. This small deviation is primarily attributed to the ideal conditions assumed in the simulation environment, which neglect real-world factors such as the sensor delay, inverter switching transients, and processor execution latency.

Despite this minor difference, the close agreement between the two responses confirms the fidelity of the simulation model and validates its effectiveness in predicting the dynamic behavior of the real system. The result also demonstrates the capability of the designed control structure to achieve fast and stable speed tracking under step input conditions in both simulation and practical implementation.

For the same speed reference value of 2000 RPM, the torque response is analyzed by comparing the simulated and experimental torque signals, as shown in Figure 14.

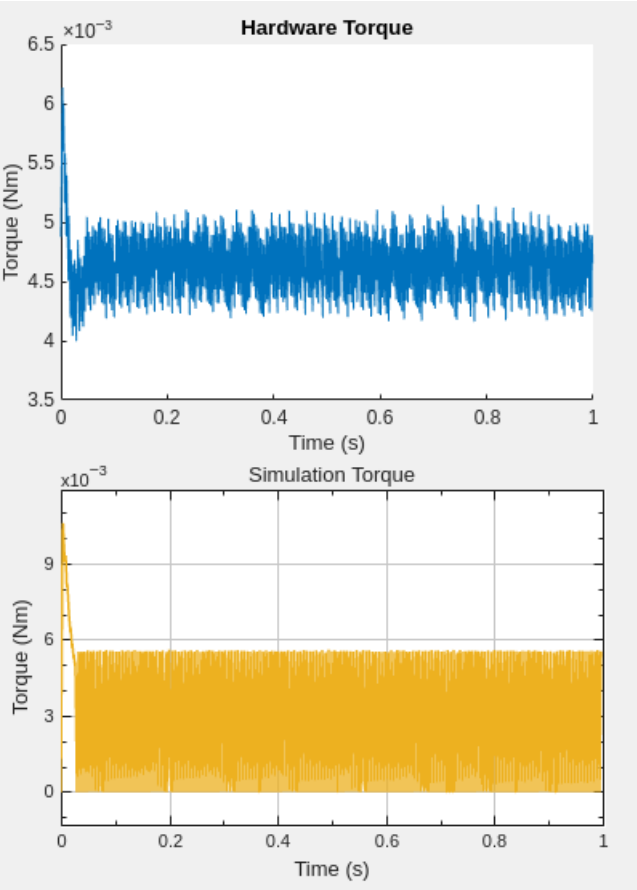

Figure 14. Simulation torque vs. hardware torque for a 2000 RPM speed reference.

Following a response time of approximately 19 milliseconds, consistent with the previously observed speed response, the torque current stabilizes at a steady-state value of approximately 120 mA. During the transient phase, the simulation exhibits a peak torque current of around 280 mA, while the hardware implementation registers a slightly lower peak of 250 mA. This discrepancy can be attributed to system non-idealities.

In contrast to the torque response, the stator flux is intentionally minimized during operation to achieve efficient and smooth control of the BLDC motor. Figure 15 illustrates the hardware flux signal under the same operating condition, confirming that the flux is effectively maintained near zero, as intended.

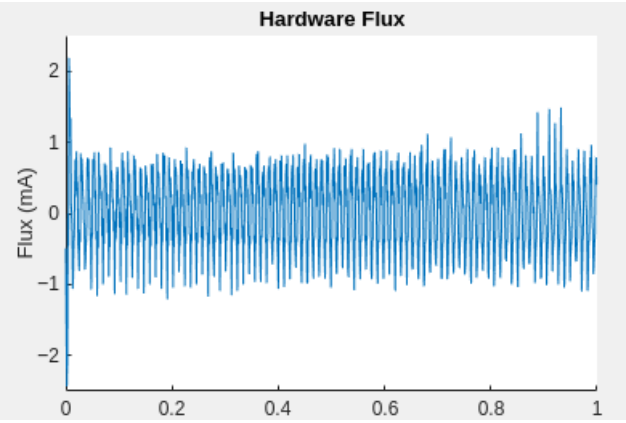

Figure 15. Web application flux results for a 2000 RPM speed reference.

Figures 16 and 17 compare the simulation and hardware responses for a reduced speed reference of 1500 RPM, using the same PI controller parameters as in the 2000 RPM

case. The test was conducted through the Web App interface to maintain consistent control settings across both platforms.

It can be observed that while both the simulation and hardware systems successfully stabilize around the reference speed, the steady-state error variation increases compared to the higher-speed case. Specifically, the fluctuation around the steady-state value reaches approximately 350 RPM, which corresponds to 8.75% of the rated speed. This suggests that the system exhibits greater sensitivity to load disturbances and internal dynamics at lower operating speeds when fixed controller gains are used.

The increased steady-state variation at 1500 RPM indicates that the controller's ability to reject disturbances and maintain tight regulation diminishes as the operating point moves away from its nominal design target. This behavior highlights the importance of adaptive gain tuning or robust control strategies to ensure consistent performance across a broader speed range.

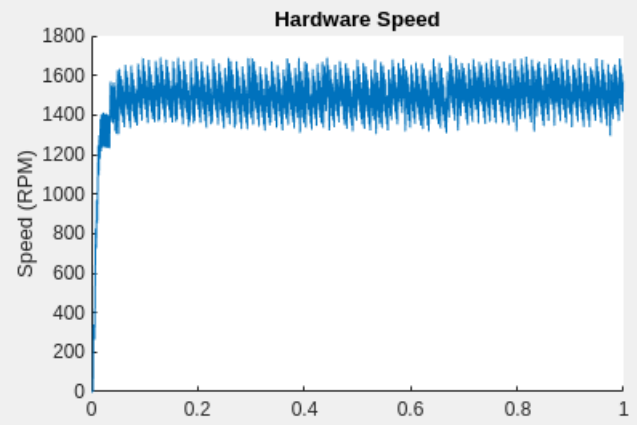

Figure 16. Web application hardware speed results for a 1500 RPM speed reference.

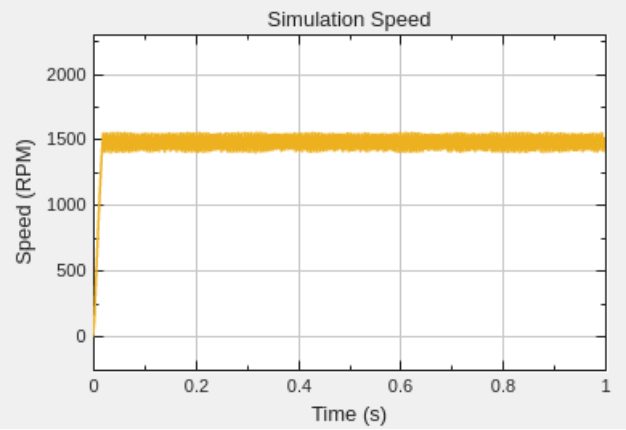

Figure 17. Web application simulation speed results for a 1500 RPM speed reference.

Figures 18 and 19 present a comparative analysis of simulation speed (purple) and hardware speed (light blue) for a system commanded to stabilize at −1500 RPM. This experiment specifically evaluates the controller's performance in achieving reverse rotational motion and maintaining stability under negative speed regulation.

The simulation curve (purple) follows a slightly smoother and monotonic descent toward the target speed of −1500 RPM, representing a system with less noise and disturbance, resulting no observable overshoot and a lesser error.

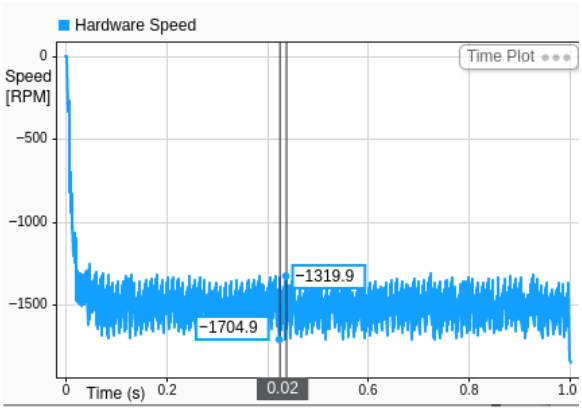

Figure 18. MIL hardware results for a −1500 RPM speed reference.

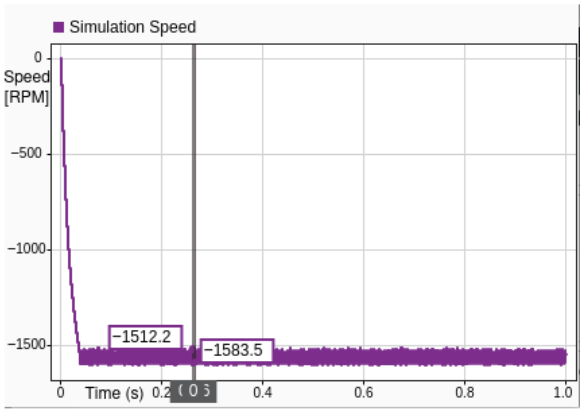

Figure 19. MIL simulation results for a −1500 RPM speed reference.

The hardware response, represented in light blue, closely follows the simulation trend, but includes realistic dynamic effects. Initially, the system does not exhibit a slower rate of deceleration, but adds more oscillation to the system, likely due to motor inductance, measurement noise, or mechanical tolerances in the hardware. Despite these transients, the hardware speed ultimately revolves around the same final value of −1500 RPM, indicating successful regulation.

The close agreement in steady-state values between simulation and hardware validates the accuracy of the simulation model and the robustness of the implemented control strategy. The transient differences highlight the role of real-world effects that are not fully captured in simulation, such as delays, quantization, and external disturbances.

Figure 20 presents a comparative assessment between the simulation and MIL test results under conditions where a 100 g mass object was attached to the rotor in the hardware setup. This additional load introduces realistic inertia, enabling a more representative validation of system dynamics under physically constrained conditions.

The graph illustrates the simulation speed (in purple) and corresponding hardware speed (in blue) over a short time interval. The simulation trace follows a smooth trajectory, reflecting ideal system behavior in the absence of physical perturbations. It reaches a speed of 2056.9 RPMwithin approximately 0.025 s, demonstrating a rapid and uniform acceleration phase.

In contrast, the hardware trace exhibits noticeable oscillations superimposed on the general acceleration trend. The speed lags behind the simulation initially, reaching 1803.3 RPM at 0.023 s, evidencing the inertial delay introduced by the 100 g mass. This physical damping and oscillatory behavior are characteristic of an added moment of iner-

tia in practical electromechanical systems, and are consistent with expectations based on system modeling.

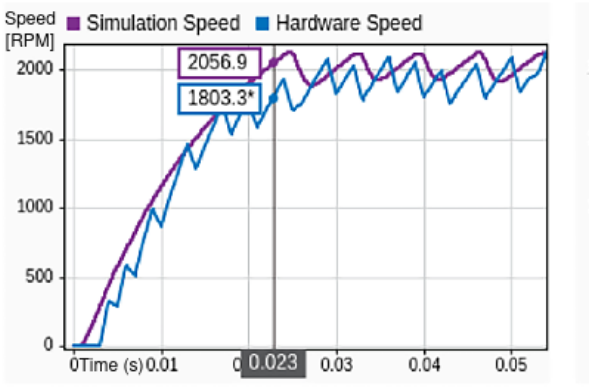

Figure 20. Response time of the hardware speed with an attached object of m = 100 g. "*" means 90% of the reference value.

Despite these dynamic deviations, the hardware response converges toward the simulation curve over time, particularly beyond the 0.04 s mark, where both traces stabilize near the same terminal speed. This convergence confirms that the control algorithm remains effective under load and is robust against disturbances introduced by mass-induced inertia.

Figures 21 and 22 present a comparative analysis of system performance under simulation and MIL conditions, captured through time series plots of execution speeds. Figure 21 illustrates the simulation speed, while Figure 22 depicts the corresponding hardware speed during experimental deployment.

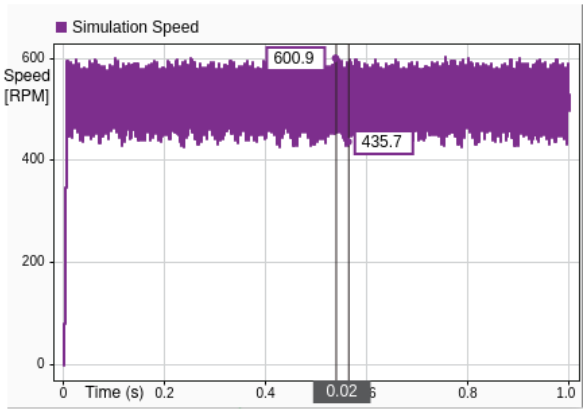

Figure 21. MIL simulation speed for a 500 RPM speed reference.

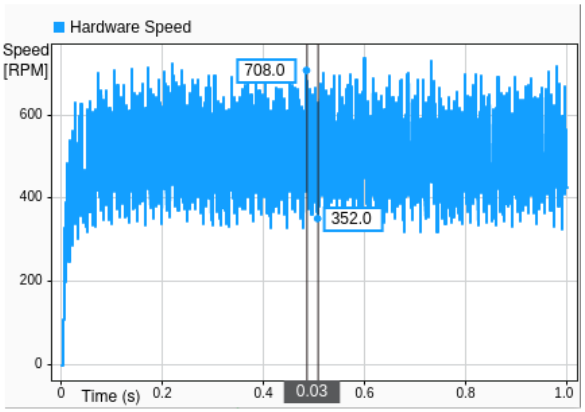

Figure 22. MIL hardware speed for a 500 RPM speed reference.

In the simulation environment, the system achieves a peak execution speed of approximately 600.9 RPM, with a minimum observed dip to around 435.7 RPM. The plot indicates generally stable behavior with speed values concentrated within the upper band, suggesting a high degree of consistency and minimal variance.

Figure 23 presents the transient response of the BLDC motor to a step change in reference speed, demonstrating the system's ability to track dynamic inputs with minimal overshoot and acceptable settling time. This result validates the effectiveness of the proposed control strategy under non-steady-state conditions and highlights the framework's capability for real-time adjustment and evaluation. The test confirms that the system maintains stability and responsiveness during speed transitions, supporting its suitability for dynamic motor control applications.

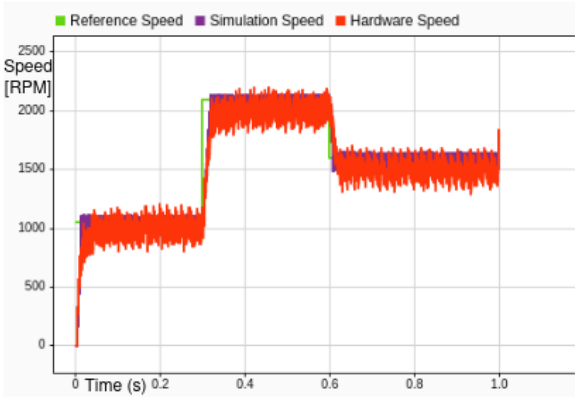

Figure 23. Step-change test for the simulation model and experimental setup.

Figure 24 presents the simulated phase currents corresponding to the step-change in reference speed shown in Figure 23.

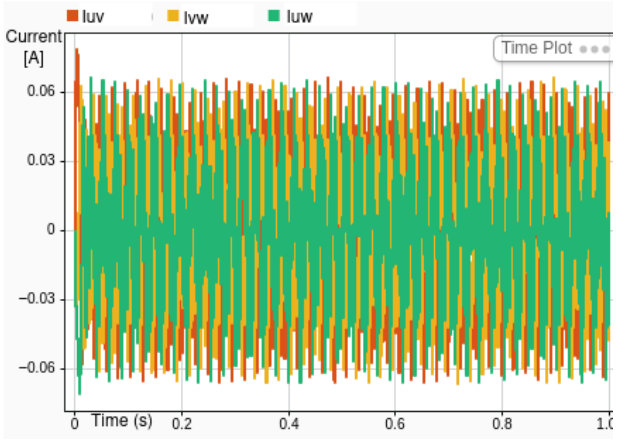

Figure 24. Phase currents for the simulation model step-change test.

7. Conclusions

This paper has presented a practical and cost-effective implementation of Direct Field-Oriented Control (DFOC) for Brushless DC (BLDC) motors using an open source software framework, resulting in an application that is independent of the used hardware. The novelty of this work lies in its integration of a lightweight control framework that operates effectively with a single Hall sensor. Unlike open-source platforms such as SimpleFOC, VESC, and ODrive, which often require dedicated hardware or tuning expertise, the proposed solution enables rapid prototyping and centralized validation within Simulink. This distinction enhances accessibility for academic researchers and industry practitioners alike,

offering a scalable and hardware-agnostic approach to motor control development and experimentation. A single Hall sensor and integrated circuitry are used as hardware components alongside the motor and processing unit. By leveraging Space Vector Pulse Width Modulation (SVPWM), direct and indirect transformation techniques, and a minimal hardware configuration, the proposed approach achieves reliable commutation with minimal positional feedback. The system demonstrates an execution speed of 100 µs per control cycle, enabling real-time operation on resource-constrained embedded platforms.

The proposed system was thoroughly modeled in MATLAB/Simulink and validated through MIL testing, demonstrating good correlation between simulation and real-world results. To facilitate direct comparison, simulation and experimental results are concurrently plotted in Figures 12, 13 and 20. Our experimental findings confirm that the steady-state error diminishes with increasing speed and that the system response remains robust under moderate mechanical load conditions. Moreover, successful bidirectional operation and low-flux behavior emphasize the effectiveness of the torque and flux control loops.

A web-based application was also developed to interface between the simulation and hardware in real time, providing an accessible platform for performance visualization and tuning. This reinforces the system's educational and development utility.

While the proposed framework shows reliable steady-state performance and supports web-based algorithm testing, limitations such as lack of dynamic load variation, MATLAB dependency, and incomplete transient characterization remain to be addressed in future work.

In summary, the proposed software infrastructure and DFOC solution work together to reduce sensor count and system complexity while maintaining reliable control, providing an attractive option for embedded motor control applications where cost, simplicity, and efficiency are critical. Future work might explore further reduction of the steady-state error at low speeds and integration of adaptive or learning-based control strategies.

Author Contributions: Conceptualization, R.B.S. and R.E.; methodology, R.B.S. and R.E.; software, R.B.S.; validation, R.B.S. and R.E.; writing—original draft preparation, R.B.S.; writing—review and editing, R.B.S. and R.E. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Piro grant, ctr. no. 17/01.07.2024 funded by the National Grant Competition—GNaC ARUT 2023.

Data Availability Statement: The firmware and software application supporting the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: We thank the anonymous reviewers for their valuable comments which helped us improve the organization, content, and presentation of this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AC	Alternating Current
ADC	Analog-to-Digital Converter
API	Application Programming Interface
ARM	Advanced RISC Machine
BLDC	Brushless Direct Current Motor
BSP	Board Support Package
D	Direct axis
DC	Direct Current
DFOC	Direct Field-Oriented Control

DMA	Direct Memory Access
DTC	Direct Torque Control
EMF	Electromotive Force
FOC	Field-Oriented Control
GPIO	General-Purpose Input–Output
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IFOC	Indirect Field-Oriented Control
MIL	Model-in-the-Loop
P	Proportional
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PMSM	Permanent Magnet Synchronous Machine
PWM	Pulse Width Modulation
Q	Quadrature axis
RPM	Revolutions Per Minute
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SVPWM	Space Vector Pulse Width Modulation
UART	Universal Asynchronous Receiver–Transmitter
VFD	Variable-Frequency Drive
Web App	Web Application

References

1. Carev, V.; Roháč, J.; Tkachenko, S.; Alloyarov, K. The Electronic Switch of Windings of a Standard BLDC Motor. *Appl. Sci.* **2022**, *12*, 11096. [CrossRef]
2. Fitzgerald, A.; Kingsley, C.; Umans, S. *Electric Machinery*; McGraw-Hill Companies: Columbus, OH, USA, 2003.
3. Tolbert, C.L. Leveraging Variable Frequency Drive Data for Nondestructive Testing and Predictive Maintenance in Industrial Systems. *NDT* **2025**, *3*, 7. [CrossRef]
4. Nicola, M.; Nicola, C.-I.; Selișteanu, D.; Ionete, C.; Șendrescu, D. Improved Performance of the Permanent Magnet Synchronous Motor Sensorless Control System Based on Direct Torque Control Strategy and Sliding Mode Control Using Fractional Order and Fractal Dimension Calculus. *Appl. Sci.* **2024**, *14*, 8816. [CrossRef]
5. Le-Huy, H. Comparison of field-oriented control and direct torque control for induction motor drives. In Proceedings of the Industry Applications Conference. Thirty-Fourth IAS Annual Meeting, Phoenix, AZ, USA, 3–7 October 1999; Conference Record of the 1999 IEEE; pp. 1245–1252.
6. Habetler, T.G.; Profumo, F.; Pastorelli, M.; Tolbert, L.M. Direct torque control of induction machines using space vector modulation. *IEEE Trans. Ind. Appl.* **1992**, *28*, 1045–1053. [CrossRef]
7. Alkurawy, L.E.J.; Khamas, N. Model predictive control for DC motors. In Proceedings of the 2018 1st International Scientific Conference of Engineering Sciences—3rd Scientific Conference of Engineering Science (ISCES), Diyala, Iraq, 10–11 January 2018; pp. 56–61.
8. Lipo, T.A. *Introduction to AC Machine Design*, 2nd ed.; Wisconsin Power Electronics Research Center (WEMPEC): Madison, WI, USA, 2017.
9. Novotny, D.W.; Lipo, T.A. *Vector Control and Dynamics of AC Drives*; Oxford University Press: Oxford, UK, 1996.
10. Blaschke, F. The principle of field orientation as applied to the new transvector closed-loop control system for rotating-field machines. *Siemens Rev.* **1972**, *34*, 217–220.
11. Krishnan, R. *Electric Motor Drives: Modeling, Analysis, and Control*; Prentice Hall: Upper Saddle River, NJ, USA, 2001.
12. Bose, B.K. *Modern Power Electronics and AC Drives*; Prentice Hall: Upper Saddle River, NJ, USA, 2002.
13. Rodríguez, J.; Rivera, M.; Torrez, A. High-performance FOC of BLDC machines with trapezoidal back-EMF. *IEEE Trans. Ind. Appl.* **2017**, *53*, 2142–2150.
14. Furmanik, M.; Gorel, L.; Konvičný, D.; Rafajdus, P. Comparative Study and Overview of Field-Oriented Control Techniques for Six-Phase PMSMs. *Appl. Sci.* **2021**, *11*, 7841. [CrossRef]

15. Venu Gopal, B.T. Comparison Between Direct and Indirect Field Oriented Control of Induction Motor. *Int. J. Eng. Trends Technol.* **2017**, *43*, 364–369. [CrossRef]
16. Amin, F.; Sulaiman, E.; Soomro, H.A. Field Oriented Control Principles for Synchronous Motor. *Int. J. Mech. Eng. Robot. Res.* **2019**, *8*, 284–288. [CrossRef]
17. Basar, M.S.; Bech, M.M.; Andersen, T.O.; Scavenius, P.; Thomas-Basar, T. Comparison of sensorless FOC and SVM-DTFC of PMSM for low-speed applications. In Proceedings of the 4th International Conference on Power Engineering, Energy and Electrical Drives, Istanbul, Turkey, 13–17 May 2013; pp. 864–869.
18. Analog Devices. *QBL4208-x-1k 8 × 8 1k CMOS Analog Crosspoint Switch—Data Sheet*, Rev. 1.40; Analog Devices: Box Hill, VIC, Australia, 2022.