Sapog v2 Reference Manual

Zubax Robotics
Akadeemia rd. 21/1, Tallinn 12618, Estonia
info@zubax.com
Q&A: forum.zubax.com

Revision 2020-01-15

Overview

Sapog is an advanced open source sensorless PMSM/BLDC motor controller firmware designed for use in propulsion systems of electric unmanned aircraft and watercraft.

The source repository and the public bug tracker are located at `https://github.com/PX4/sapog`.

This document is applicable to firmware versions 2.x released before 2020-01-15.

This document focuses only on the firmware. Please refer to your product documentation for relevant information about the hardware.

Applications

- Propeller drives of unmanned aerial vehicles.
- Watercraft propulsion systems.
- General purpose sensorless BLDC drives.

Features

- Robust motor control algorithms.
- Fast response (critical for multirotor aircraft).
- Regenerative braking and active freewheeling.
- Optional RPM control loop (RPM governor).
- Current limiting.
- Self diagnostics and extensive real-time status reporting.
- Compatible with most BLDC motors with minimal tuning.
- Highly configurable.
- Automatic firmware update over the CAN bus in the field.
- Supported communication interfaces:
  - UAVCAN interface with optional dual bus redundancy.
  - Command line interface over UART, suitable for M2M communications.
  - RCPWM (analog PWM interface widely used in robotics).

Table of contents

List of figures

List of tables

1 Introduction

1.1 Overview

This document provides detailed information about the basic operating principles and implementation details of the Sapog BLDC motor controller firmware.

This document is focused exclusively on the aspects pertaining to the firmware itself. All questions related to particular hardware implementations are intentionally left out; that information should be gathered from the hardware datasheets distributed by hardware vendors.

1.2 Conventions

Unless specifically stated otherwise, all units of measurement used in this document are assumed to be SI. Units of measurement are typically shown in square brackets: [unit].

Within equations, references to configuration parameter values are made through the symbol Π with the name of the configuration parameter in the subscript; for example: $\Pi_{\mathrm{parametername}}$.

1.3 Definitions

PMSM Permanent magnet synchronous motor.

BLDC Brushless DC motor.

RPM Revolutions per minute.

EMF Electromotive force.

BEMF Back electromotive force.

ADC Analog to digital converter.

VSI Voltage source inverter.

ESC Electronic speed controller, synonymous to PMSM controller in the context of electric vehicles.

2 Principles of operation

This chapter provides a brief overview of the basic operating principles of electric permanent magnet synchronous motors and of the relevant approaches and solutions implemented in Sapog.

2.1 PMSM and BLDC motors

2.1.1 Basic equations

This section provides some of the most important equations that describe a DC electric motor. The principles explained here can be extended to all types of polyphase PMSM motors as well.

In the context of electric drives, BEMF is the voltage induced on the motor windings while the armature of the motor is moving relative to the magnetic field of the rotor. The magnitude of the induced voltage is dependent on the speed of the armature relative to the magnetic field of the rotor and on the magnetic flux linkage of the motor. The dependency can be expressed as follows:

$$E_b = \phi\omega_e \tag{2.1}$$

where E_b [volt] is the induced back electromotive force, ϕ [weber] is the magnetic flux linkage of the motor, and $\omega_e \left[\frac{\text{radian}}{\text{second}}\right]$ is the electrical angular velocity.

Electrical angular velocity is a function of the mechanical angular velocity $\omega_m \left[\frac{\text{radian}}{\text{second}}\right]$ and the number of rotor magnetic poles N_p:

$$\omega_e = \frac{\omega_m N_p}{2} \qquad N_p \geq 2, N_p \text{ is even} \tag{2.2}$$

Both mechanical and electrical angular velocities are related to the mechanical and electrical RPM, respectively, as follows:

$$\text{RPM} = \frac{30\omega}{\pi} \tag{2.3}$$

The armature current I_a [ampere] is proportional to the voltage difference between the induced back EMF and the source voltage E_s [volt]:

$$I_a = \frac{E_s - E_b}{R} \tag{2.4}$$

where R [ohm] is the internal resistance of the stator.

Torque τ [newton · meter] observed on the shaft is dependent on the torque-generating current, magnetic flux linkage, and the number of rotor magnetic poles as follows:

$$\tau = \frac{\sqrt{3}}{2} I_a N_p \phi \tag{2.5}$$

Mechanical power output P [watt] is the product of torque and mechanical angular velocity:

$$P = \tau\omega_m \tag{2.6}$$

Motor velocity constant $K_V \left[\frac{\text{RPM}_m}{\text{volt}}\right]$ is sometimes used in marketing materials and motor specifications instead of magnetic flux linkage.[1] This is not a SI quantity, so its use is generally discouraged. It can be expressed via the magnetic flux linkage ϕ and the number of rotor magnetic poles N_p as follows:

$$K_V = \frac{20\sqrt{3}}{\pi N_p \phi} \tag{2.7}$$

[1] Sometimes K_V is written as KV, although this form is discouraged because it can be confused with kilovolt (kV).

2.1.2 BEMF shape

There are two major types of permanent magnet synchronous motors (PMSM) that differ by the shape of the induced back electromotive force while the rotor is moving at a constant speed.

The first type has a fixed magnetic flux linkage that is independent of the electrical position of the rotor. The induced BEMF per phase therefore has a sinusoidal shape. These motors are typically referred to simply as PMSM, or BLAC. Since the term PMSM is quite ambiguous, we will be using the term BLAC to refer to a PMSM with sine shaped BEMF.

The other type, referred to as brushless DC, or BLDC, has a variable flux linkage that changes with the rotor position in a specific way that results in the trapezoidal shape of the back EMF. The difference is illustrated on figure 2.1.

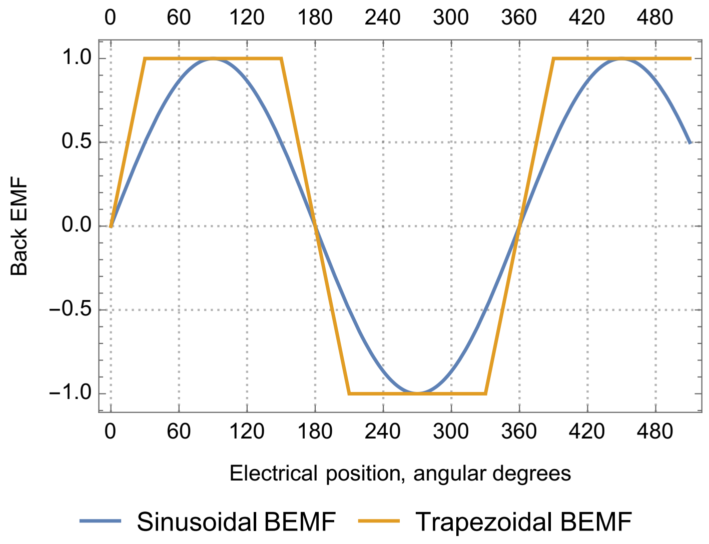

Figure 2.1: Difference between trapezoidal and sinusoidal BEMF

Sapog, being a BLDC drive, drives the motor with trapezoidal modulated voltage. This produces smooth torque and vibration free operation with BLDC motors, since it is expected that the flux linkage will be changing in a specific way that matches the shape of the modulated voltage.

Like any BLDC drive, Sapog can operate with BLAC motors as well, albeit with increased torque ripple. From the model presented in section 2.1.1 we can predict that the motor will exhibit periodic variations of the output torque if the form of the voltage modulated by the controller does not match the form of the back EMF induced in the motor. The resulting periodic torque variations that occur in a BLAC motor driven by a BLDC drive (and, likewise, in a BLDC motor driven with sine modulated voltages) are shown on the figure 2.2. Many applications, however, can tolerate the resulting torque ripple and vibrations.

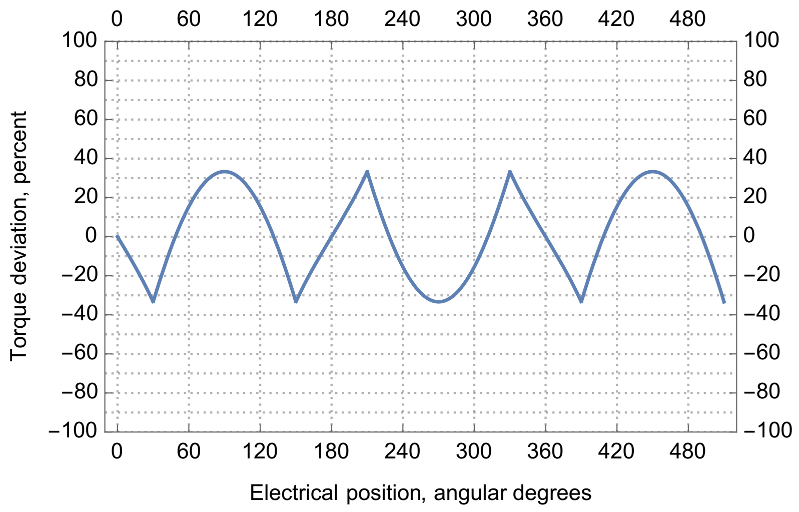

Figure 2.2: Torque ripple caused by mismatching forms of the back EMF and the source voltage

2.2 Motor control

2.2.1 Commutation sequence

Three phase BLDC drives operate by modulating a specific sequence of voltages on the motor phases synchronously with the movement of the rotor. The modulated voltage sequences are shown in the commutation table below (legend: + – positive voltage output; − – negative voltage output; ↑ – the phase is floating, induced BEMF is rising; ↓ – the phase is floating, induced BEMF is falling).

Phase \ Step		0	1	2	3	4	5
Forward	A	−	↑	+	+	↓	−
	B	+	+	↓	−	−	↑
	C	↓	−	−	↑	+	+
Reverse	A	−	↑	+	+	↓	−
	B	↓	−	−	↑	+	+
	C	+	+	↓	−	−	↑

The choice between forward and reverse commutation tables affects the direction of rotation of the motor, and it is specified via the configuration parameter `ctl_dir` (page 31).

The rotor position is deduced from the behavior of the induced BEMF on the floating phase. From figure 2.1 we can deduce that when the induced BEMF of the floating phase crosses the median voltage between the high and the low phase, there are 30 electrical degrees left before the next commutation step begins. The controller leverages this rule to estimate the time when the next commutation should occur. The process is demonstrated on figure 2.3. The voltage waveforms shown on the figure were acquired from the phase voltage signal conditioning circuits, before the ADC inputs (as shown on figure 2.4).

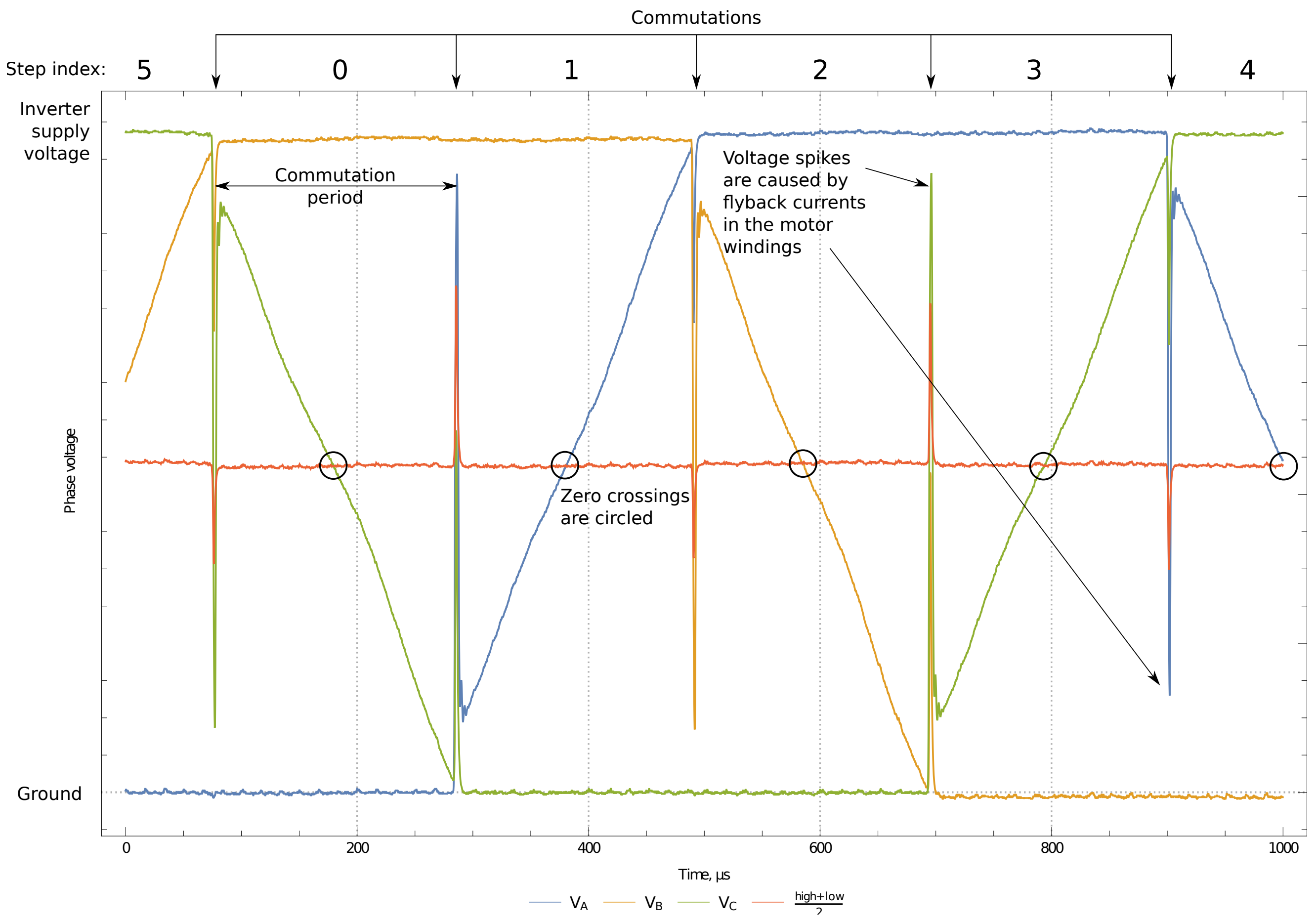

Figure 2.3: Six step commutation sequence observed through phase voltages
PWM modulation is not seen because the shown voltages were acquired while the controller was operating at the full duty cycle.

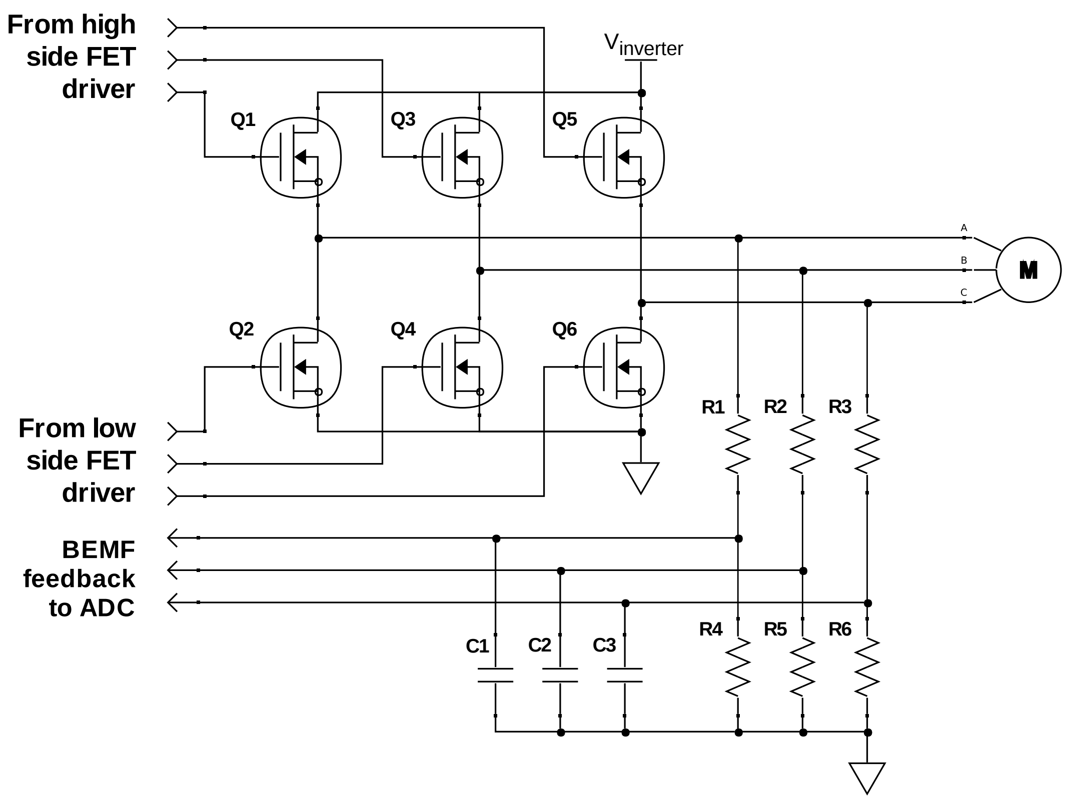

Figure 2.4: Simplified schematic of the VSI and BEMF feedback circuits that Sapog relies on

From section 2.1.1 we know that the magnitude of the induced BEMF is linearly dependent on the speed of the relative motion between the armature and the rotor's magnetic field. Considering also the fact that the controller relies on the induced BEMF signal for rotor position estimation, we can predict that the rotor position estimation may be unreliable if the induced BEMF is not sufficiently strong. In order to avoid issues with low speed operation, Sapog provides several configuration parameters that restrict the minimum operating speed, such as the following:

- `mot_comm_per_max` (page 31) - maximum commutation period, in microseconds. The estimated commutation period will be constrained by the controller to not exceed this value.
- `mot_rpm_min` (page 31) - minimum RPM that can be demanded in the RPM governor mode. If the demanded RPM is lower, the setpoint will be increased by the controller. Reduce this parameter to make the minimum speed lower; increase it to improve stability.
- `mot_v_min` (page 31) - minimum source voltage when operating in the open loop mode, in volts. If the demanded duty cycle amounts to a lower source voltage than this, the setpoint will be increased by the controller. Reduce this parameter to make the minimum speed lower; increase it to improve stability.

These parameters may need adjustment to make Sapog compatible with a specific motor. For example, if the motor cannot maintain stable operation at a low speed, the minimum source voltage should be increased, and/or the maximum commutation period should be made longer. The minimum voltage limit does not apply to the RPM loop mode, where the minimum speed is governed by a dedicated configuration parameter as shown above.

The following equation can be used to convert between commutation period T_{comm} [second] and electrical angular velocity $\omega_e \left[\frac{\text{radian}}{\text{second}}\right]$:

$$\omega_e = \frac{\pi}{3T_{\text{comm}}} \tag{2.8}$$

2.2.2 Source voltage modulation

The source voltage modulation technique employed by Sapog is a fixed carrier frequency, fixed dead time, 4-quadrant, simultaneous complementary switching PWM modulation. This modulation technique supports active freewheeling and regenerative braking.

Reviewing different modulation techniques would be outside of the scope of this document. For more background, the reader is advised to refer to other sources, such as the article "Influence of PWM Schemes and

Commutation Methods for DC and Brushless Motors and Drives” (Dal Y. Ohm, Richard J. Oleksuk).

The following configuration parameters govern the operation of the source voltage modulator:

- `mot_pwm_hz` (page 31) - PWM carrier frequency, in hertz.
- `mot_pwm_dt_ns` (page 31) - PWM switching dead time, in nanoseconds.

2.2.2.1 *PWM dead time considerations*

Generally, the end user should not change the dead time settings from the default values provided by the vendor, because that may have unexpected negative effects on the drive.

However, in certain scenarios where a higher power output is preferred, the application may benefit from reduced dead time intervals. The hardware documentation may contain more useful data on this topic.

2.2.3 Field weakening

From section 2.1.1 we know that the maximum speed that a motor can achieve is dependent on the the load, magnetic flux linkage, and the source voltage. Reduction of the magnetic flux linkage decreases the induced BEMF, which allows the rotor to reach a higher speed, given a constant source voltage and load.

The controller can alter the commutation sequence in a specific way, causing the rotating magnetic field created by the stator to interact with the magnetic field of the rotor in a way that reduces the magnetic field linkage, effectively increasing the maximum speed of the rotor, reducing the output torque, while keeping the maximum mechanical power output constant.

This technique is known as *field weakening*, and it is implemented by means of adjustment of the commutation sequence, shortening the time interval between the point of zero crossing and the moment when the next commutation step begins. In the context of BLDC drives this specific approach is also known as *commutation advancing*.

Observe that since a part of the input power is used for the suppression of the magnetic fields induced by the permanent magnets in the rotor, the overall efficiency of the drive decreases with stronger field weakening.

Sapog defines the commutation advance angle in electrical angular degrees of rotor position, and it can vary from 0° to 29°, inclusively, where 0° corresponds to no advance, and 29° corresponds to the maximum advance angle. The practical effects imposed on the drive are summarized in the following table.

Advance angle	Flux linkage	Maximum torque	Maximum speed	Drive efficiency
Low	High	High	Low	High
High	Low	Low	High	Low

Sapog can modulate a fixed advance angle, or it can be defined as a function of speed. The latter is especially useful, because a strong field weakening at low speeds does not make practical sense, and serves only to reduce the drive efficiency with no apparent benefit for the application. Additionally, a strong field weakening may be responsible for stability issues at low operating speeds, because it effectively reduces the magnitude of the BEMF induced in the motor.

The field weakening feature is configured with the help of the following configuration parameters:

- `mot_tim_adv_max` (page 31) - the maximum commutation advance angle, in electrical rotor position degrees.
- `mot_tim_adv_min` (page 31) - the minimum commutation advance angle, same units.
- `mot_tim_cp_min` (page 31) - when the current commutation period is longer than this value, the minimum advance angle will be used. The units are microseconds.
- `mot_tim_cp_max` (page 31) - when the current commutation period is shorter than this value, the maximum advance angle will be used. Same units.

When the current commutation period is between `mot_tim_cp_max` and `mot_tim_cp_min`, the advance angle will be proportionally linearly interpolated between `mot_tim_adv_min` and `mot_tim_adv_max`. The interpolation logic is demonstrated on figure 2.5.

Sample phase voltage waveforms acquired while the controller was operating with high field weakening settings are shown on figure 2.6.

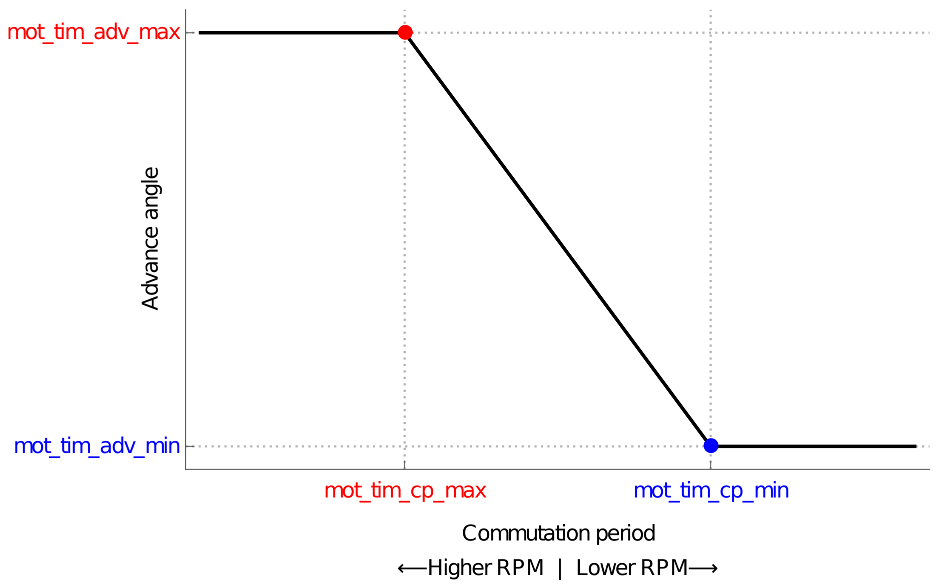

Figure 2.5: Timing advance interpolation logic

2.2.3.1 *Freewheeling considerations*

Upon careful review of the field weakening principle it might become obvious that it poses a certain danger when the motor operates at a high speed.

According to the explanations above, as long as the field weakening is active, the controller actively suppresses the excessive flux of the permanent magnets in the rotor. The suppression allows the controller to lower the induced BEMF, and, therefore, extend the operating speed upwards, while keeping the source voltage constant.

Consider what happens when the inverter (or the controller as a whole) becomes turned off while the motor is running with the rotor field weakened. As the controller ceases to suppress the field, the flux linkage of the motor returns to nominal, which, given that the speed is sufficiently high, may significantly exceed the inverter supply voltage V_{inv}, possibly causing damage to the controller, the power supply, the motor, and so on.

2.2.4 Rotor state observer

2.2.4.1 *BEMF processing*

As mentioned previously, Sapog deduces the current position of the rotor by tracking the induced BEMF of the floating phase. Which phase is floating and the direction in which the BEMF is changing is dependent on the current electrical angle of the rotor.

Measurement of the induced BEMF is done by the ADC, which is connected to the motor phases via resistive voltage dividers which reduce the dynamic range of the motor phase voltages down to a narrower dynamic range acceptable for the ADC, and a buffer capacitor that provides fast recharge of the sample-and-hold capacitor of the ADC, and suppresses high frequency noise. The resulting simplified schematic can be seen on figure 2.4.

BEMF measurements are sensitive to noise. In order to prevent the noise emitted by the VSI from interfering with BEMF measurements, the controller synchronizes the BEMF sampling with PWM switching. However, the temporal resolution provided by the resulting sampling pattern is insufficient for robust rotor state estimation because of the sampling period restrictions imposed by the PWM carrier frequency.

In order to work around this limitation, Sapog collects a number of samples, and, knowing that the shape of the BEMF induced by the floating phase of a BLDC motor is linear, solves a linear regression problem on the collected samples, effectively recovering the most probable shape of the BEMF signal. The resulting approximation is very robust and resilient to noise.

The principle is illustrated on figure 2.7.

The number of BEMF samples N_{samples} considered in the linear regression problem is defined by the config-

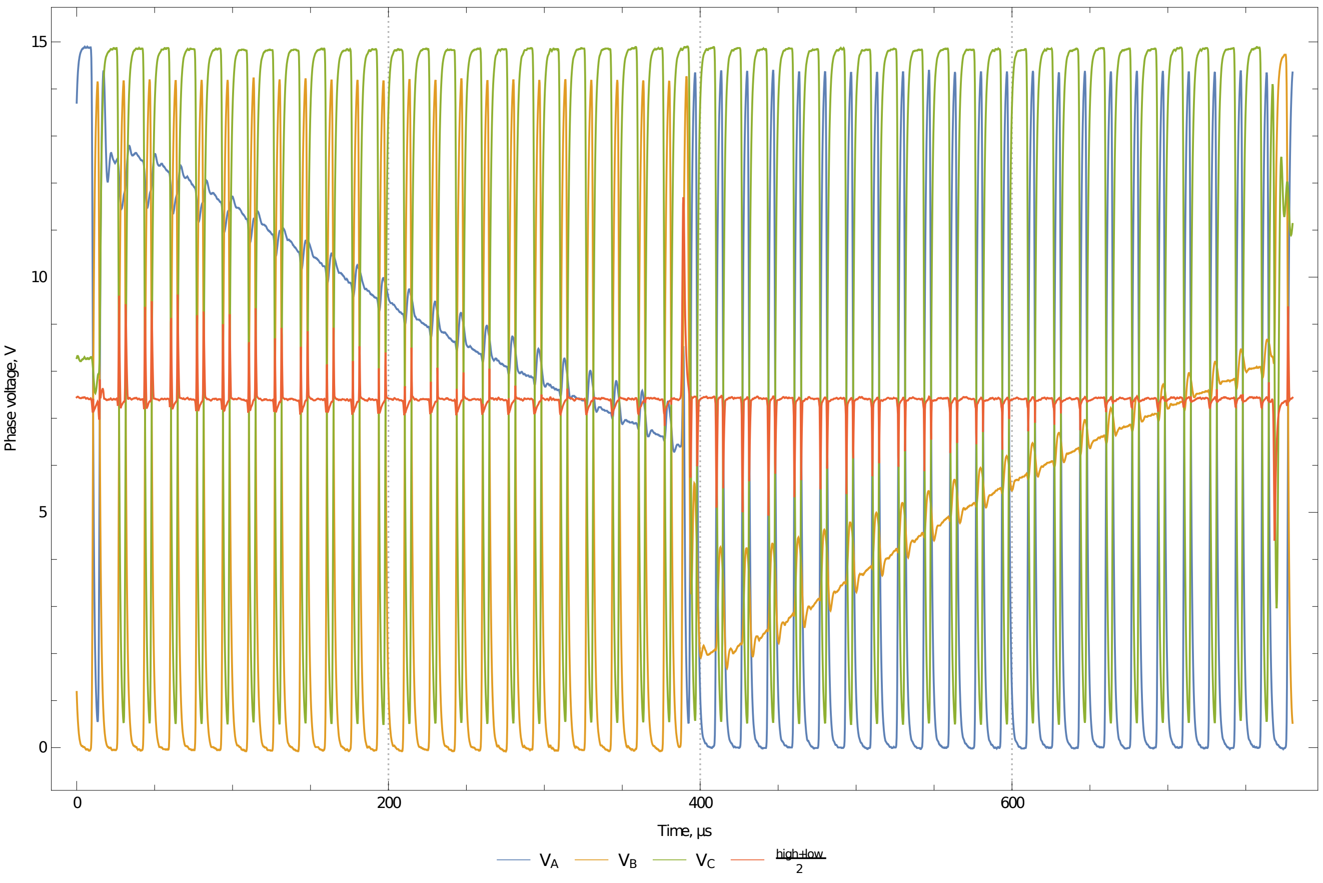

Figure 2.6: Commutation sequence with high advance angle settings
It can be seen that the next commutation begins shortly after the induced BEMF crosses the neutral voltage.

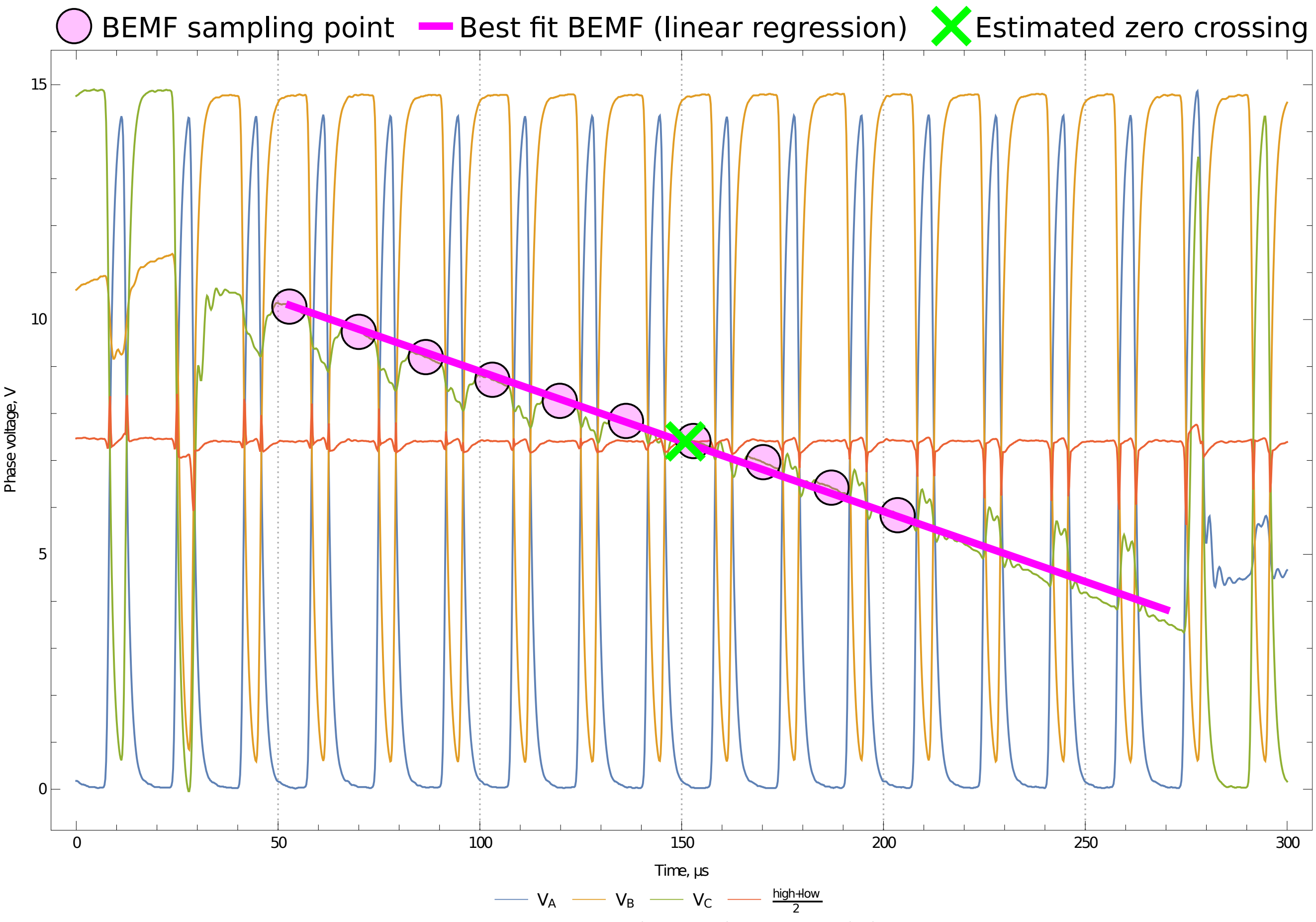

Figure 2.7: BEMF sampling and PWM modulation

uration parameter `mot_bemf_win_den` (page 31) (dimensionless) as follows:

$$N_{\text{samples}} = \frac{T_{\text{comm}} F_{\text{pwm}}}{\frac{\alpha \cdot \Pi_{\text{bemfwinden}}}{15} + \Pi_{\text{bemfwinden}}} + 2 \tag{2.9}$$

where T_{comm} is the current commutation period, $\Pi_{\text{bemfwinden}}$ is the value of the configuration parameter, α is the current commutation advance angle in electrical rotor position degrees, and F_{pwm} [hertz] is the PWM carrier frequency.

An attentive reader might point out that the assumption of linear BEMF that the described method rests upon does not hold for BLAC motors, since the induced BEMF of a floating phase will be sinusoidal rather than linear. While the objection is correct, this deviation does not pose a significant problem, since the sinusoidal BEMF can be effectively approximated to linear near the point of zero crossing. Some issues may arise with very high advance angle settings, where the controller may be forced to resort to the look-ahead extrapolation, so that the linear regression will be applied to a sample set acquired at the point where the linear approximation is not valid. The resulting imprecision, however, should not be of a significant practical importance.

2.2.4.2 *Special cases*

Figure 2.8 provides a closer look at the challenges seen by the controller in certain operating modes, which are reviewed below.

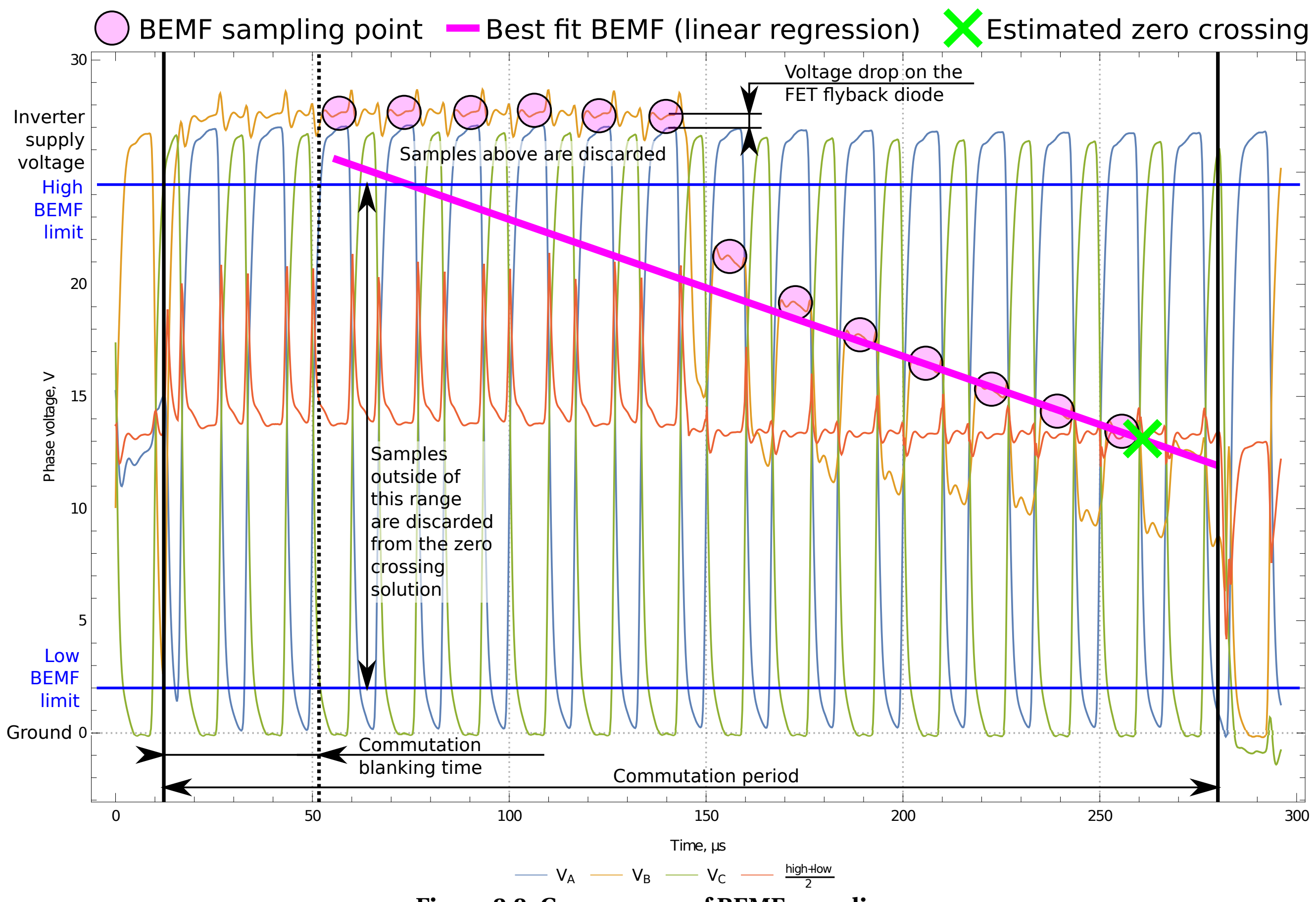

Figure 2.8: Corner cases of BEMF sampling

In certain operating modes the induced BEMF of the floating phase may briefly exceed the inverter supply voltage V_{inv}, or fall below the ground level, causing a voltage drop on either the high-side or low-side flyback diodes, respectively. The BEMF samples acquired at this time are not representative of the true form of the signal, because the signal is altered by external factors; and therefore, such samples cannot be used for the linear regression. Sapog uses a simple heuristic that allows it to detect when the sampled signal is subjected to altering and discard the affected samples. The heuristic checks whether the sampled voltage falls into a specified range near the neutral voltage, and discards the sample if it doesn't.

Additionally, the samples acquired shortly after a commutation may be subjected to transient distortions. Sapog works around that by imposing a fixed *blanking time* after every commutation, during which the BEMF is not sampled.

The following configuration parameters are relevant to the reviewed concepts:

- `mot_bemf_range` (page 31) - the voltage range around the neutral, in percent of the inverter supply voltage V_{inv}, where the BEMF samples will be considered valid and accepted for the regression problem.
- `mot_blank_usec` (page 31) - the duration of the blanking time after the commutation, in microseconds.
- `mot_spup_blnk_pm` (page 31) - the minimum blanking time during spin-up, permill of the current commutation period. This parameter is reviewed in more detail in a dedicated section.

Additional difficulties are posed by operation at very high power levels, and/or when the controlled motor has highly inductive windings. Under these conditions, the motor windings tend to accumulate a significant energy during the duration of the commutation period. When the current commutation period ends and the winding becomes floating, the accumulated energy maintains some current through the winding and through the *flyback diode* of one of the transistors until the energy accumulated while the winding was powered is dissipated. The current caused by the energy release from a disconnected inductive network is known as *flyback current*. As long as the flyback current is flowing though the winding and the flyback diode, the induced BEMF is completely suppressed and, therefore, is unobservable, which does not allow the controller to evaluate the state of the rotor.

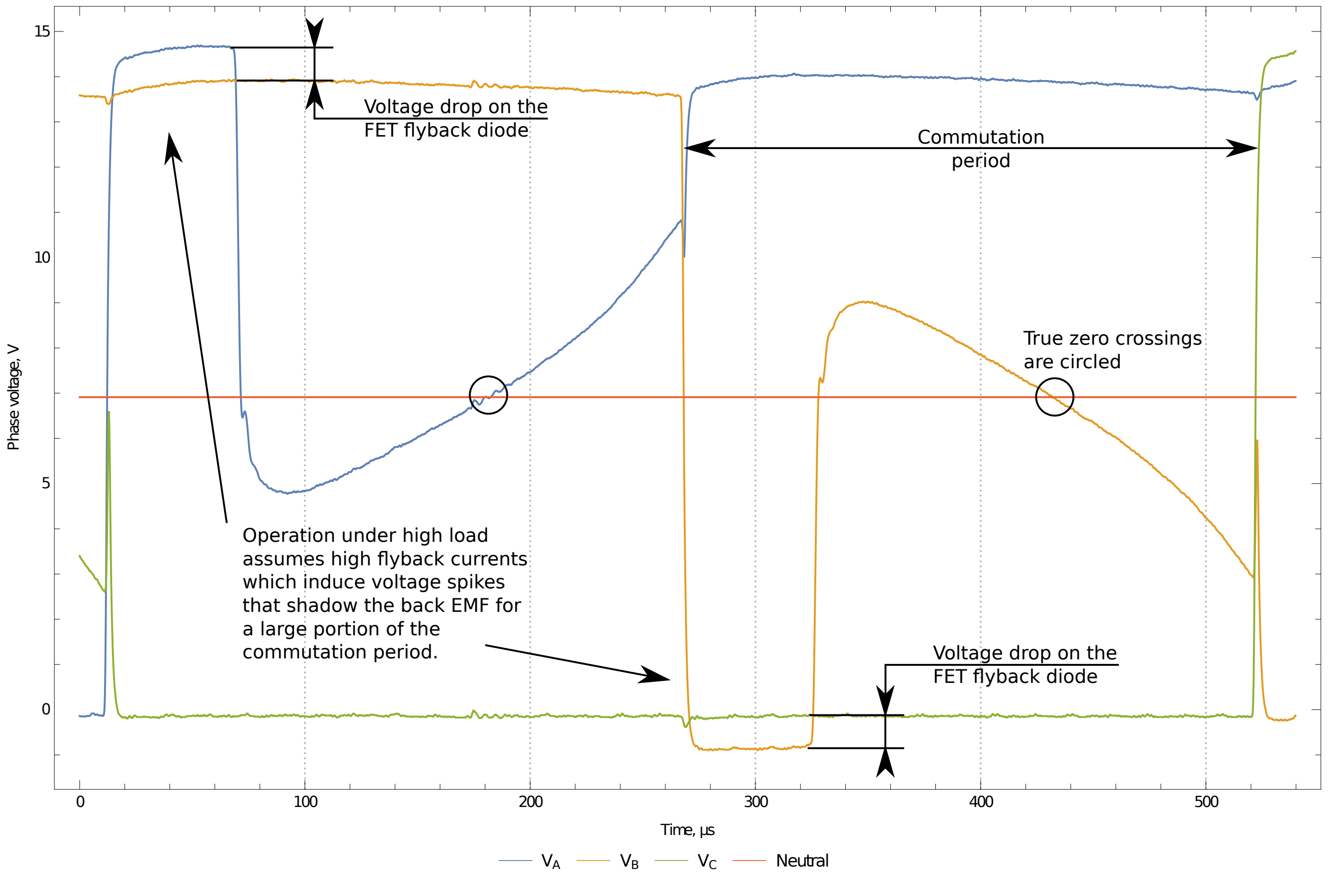

Figure 2.9: Challenges of high power operation

Sapog recognizes this condition and waits for the flyback current to cease. If the current is still flowing past a certain deadline necessary for the controller to evaluate the point of zero crossing, Sapog enters a *desaturation mode* in order to bring the system back into an observable state. In the desaturation mode Sapog turns off the inverter and waits until the next commutation period begins. By the time the next commutation period begins, the flyback currents will have ended, allowing the controller to continue observation of the motor state.

Figure 2.9 shows operation waveforms near the point of activation of the desaturation mode.

2.2.4.3 *Maximum speed restrictions*

It is easy to see that as the speed increases and the commutation period shortens, the number of BEMF samples that can be accommodated in a commutation period decreases. This effect imposes a limit on the maximum achievable speed, and the limit is dependent on the PWM carrier frequency.

When the drive operates in the open loop mode, the controller will limit the maximum duty cycle when the commutation period reaches dangerously low values. The limiting is done by means of a simple P-controller, which overrides the duty cycle setpoint when it exceeds a certain value.

When the drive operates in the RPM control loop mode, the limiting will be performed simply by means of clipping of the RPM setpoint.

The exact thresholds are specified on figure 2.10.

The configuration parameter `mot_pwm_hz` (page 31) can be used to set the desirable PWM frequency in hertz. It is recommended to keep the frequency as low as possible, as that would facilitate lower switching losses and therefore higher efficiency of the drive.

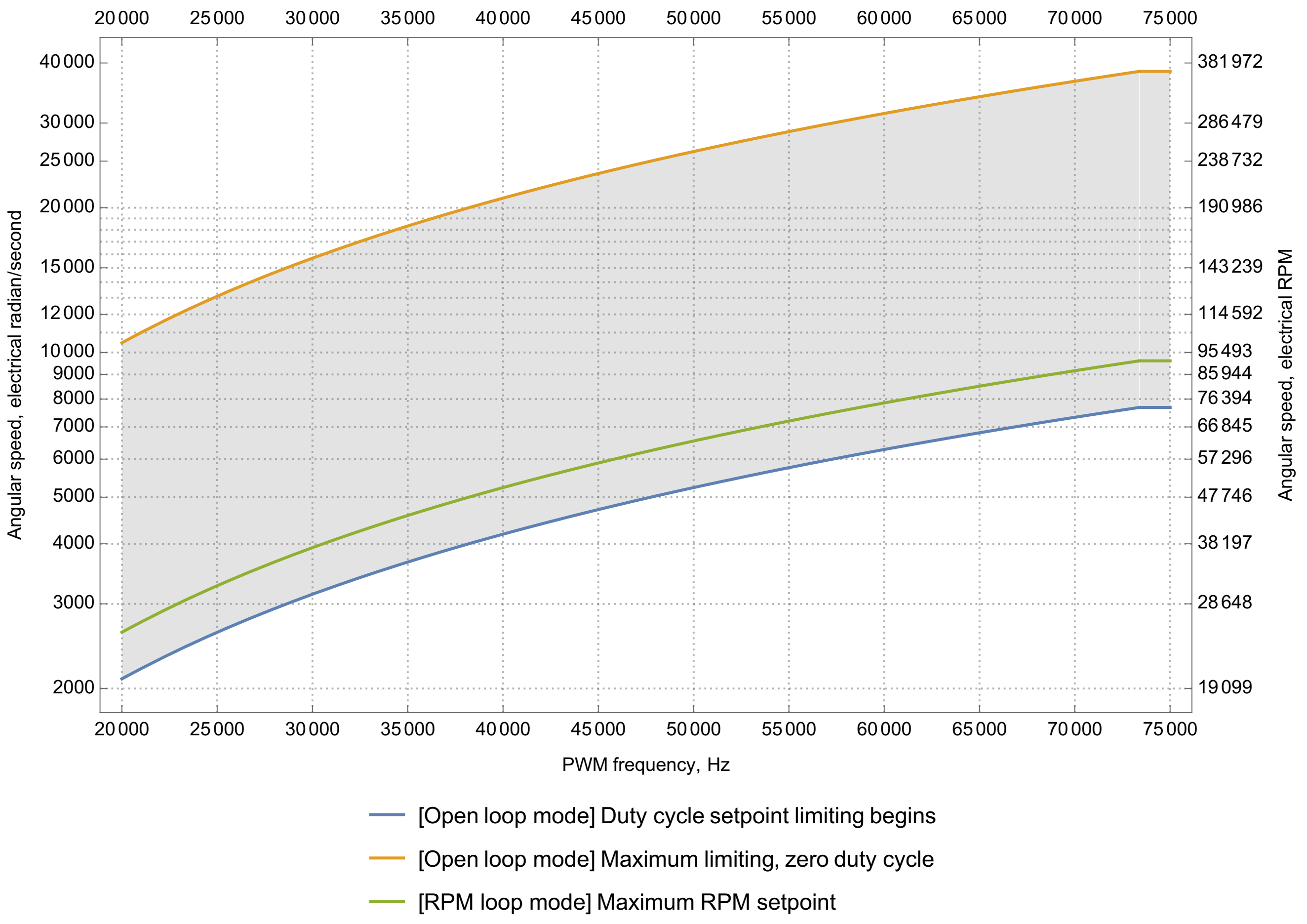

Figure 2.10: Dependency of the maximum speed on PWM frequency

2.2.4.4 *Field weakening restrictions*

The controller requires several microseconds to solve the linear regression problem after the sampling is finished. However, operation at very high advance angles requires the controller to perform commutation virtually immediately after the zero crossing has happened, leaving very little time to perform the computation. The controller strives to work around this contradiction by trying to robustly extrapolate the shape of the BEMF signal, predicting the upcoming zero crossing ahead of time.

Despite that, operation at very high speeds may not provide the time necessary to take a sufficient number of samples and solve the regression problem before the next commutation is due. In this case, Sapog will silently lower the current advance angle setting and keep it at as close to the requested value as the current operating mode permits.

2.2.5 Regenerative braking

During a regenerative braking, the source voltage E_s modulated by the controller is lower than the induced BEMF E_b, which creates a reverse current flow from the motor into the VSI and further into the power supply network. According to the formulas presented in section 2.1.1, reversing the current will also reverse the torque, causing the motor to rapidly decelerate.

This process converts the mechanical energy of the motor into electrical energy that is returned back into the power supply network. If the power supply network cannot absorb the recovered energy (e.g. due to its resistance being too high), the regenerative energy transfer may lead to an increase of the supply voltage beyond the safe operating limits. Typically, the controller's hardware is equipped with some protection circuits that activate and dissipate the excessive energy when V_{inv} increases a certain threshold; however, their energy absorption capabilities are often very limited.

Generally, batteries are capable of absorbing the regenerated energy during braking without issues. Problems may arise if the controller is powered from a source that does not permit reverse currents, such as laboratory power supplies or some types of voltage conversion circuits.

Refer to the documentation provided with your hardware to learn more about the issues concerning the regenerative braking and the excess energy release.

2.2.6 Rotor stall detection

Rotor stall is a condition where the rotor ceases to rotate due to the torque generated by the motor being lower than the mechanical load on the shaft. Sapog detects this condition by observing a specific number of failures to find a valid BEMF zero crossing event.

As explained in one of the earlier sections of this document, Sapog requires the rotor to move in order to be able to operate normally (this applies to virtually all sensorless drives). Once the rotor has stalled, Sapog shuts down the VSI and waits for the next setpoint to arrive. Once the next setpoint has arrived, Sapog restarts the motor normally, unless the number of consecutive stalls has exceeded a specific limit.

In the latter case, Sapog will lock up and refuse to start the motor again, until it has received a zero setpoint. A zero setpoint is treated as a reset command that clears the stall counter and unlocks the controller.

Zero crossing detection failures are counted with the help of a dedicated counter state. When Sapog fails to find a zero crossing solution, the failure counter is incremented by one. When Sapog succeeds to perform six (6) commutations in a row without failures, the failure counter is reset back to zero.

The following configuration parameters configure the behaviors pertaining to stall detection:

- `mot_zc_fails_max` (page 31) - zero crossing detection failure threshold. If the failure counter exceeds this threshold, the rotor will be considered stalled.
- `mot_stop_thres` (page 31) - lock-up threshold. If the rotor has stalled this number of times in a row, the controller will lock up until a zero setpoint is received.

2.2.7 Spin-up

As has been explained before, Sapog requires the rotor to move in order to be able to operate normally, because the state estimation methods that work well when the rotor is moving break apart when the rotor is stationary or nearly so.

From the above follows that a different method of state estimation is needed to start the motor from standstill, especially so if the shaft is attached to a mechanical load.

2.2.7.1 Algorithm

When starting the motor, Sapog sets the source voltage E_s to the value specified in the configuration parameter `mot_v_spinup` (page 31), and activates the commutation step at the index 1. Having done so, Sapog begins to wait for the zero crossing detector to report that it is time to begin the next commutation period, or until the amount of time specified by the configuration parameter `mot_spup_st_cp` (page 31) expires, whichever happens first. Afterwards, Sapog switches to the next commutation step, and the process repeats again.

While in the spin-up mode, Sapog slowly increases the voltage from the initial value specified by `mot_v_spinup` (page 31) until the minimal operating voltage `mot_v_min` (page 31) is reached. The duration of the spin-up voltage ramp is specified by the parameter `mot_spup_vramp_t` (page 31), in seconds. The ramp is graphically illustrated on figure 2.11.

It should be noted that high initial voltages and/or steep voltage ramps need to be avoided, because they may lead to severe instabilities in the beginning of the spin-up sequence. Keep in mind that it is virtually impossible for any synchronous sensorless drive to create a stable torque at low speeds, and especially at

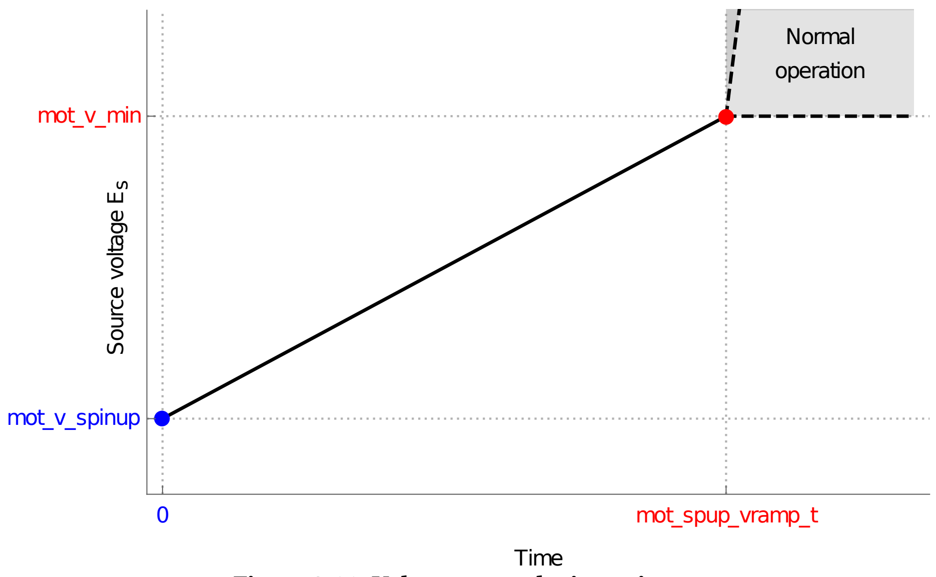

Figure 2.11: Voltage ramp during spin-up

standstill.

While spin-up is in progress, Sapog always operates at the zero advance angle (field weakening disabled) for the reasons explained earlier, regardless of the configured advance angle settings.

2.2.7.2 *Spin-up rotor state observer*

The zero crossing detector mentioned above operates in a very different way compared to the normal rotor state observer used in the normal operating mode.

While operating in the spin-up mode, the observable behavior of the induced BEMF may be very erratic, mainly due to the low magnitude of the signal and also due to the irregular rotation of the rotor caused by imprecise synchronization of the modulated field with the field of the rotor. In order to combat these difficulties, Sapog employs a different kind of observer, which provides more robust operation at low speeds.

Immediately after a commutation Sapog inserts a blanking delay T_{blank} defined by the following equation:

$$T_{\text{blank}} = \max\left(\frac{\Pi_{\text{blankusec}}}{10^6}, \frac{\Pi_{\text{spupblnkpm}} T_{\text{comm}}}{1000}\right) \tag{2.10}$$

where $\Pi_{\text{spupblnkpm}}$ is the value of the configuration parameter `mot_spup_blnk_pm` (page 31) (permill of the commutation period), T_{comm} is the duration of the last commutation period, $\Pi_{\text{blankusec}}$ is the value of the configuration parameter `mot_blank_usec` (page 31) (in microseconds).

Upon expiration of the blanking delay, Sapog watches the BEMF waiting for the flyback current to cease. Afterwards, Sapog begins to integrate the sampled BEMF voltages relative to the neutral voltage. Once the integrated voltage changes the sign, Sapog switches to the next commutation period. A more formal definition of the commutation condition is shown below:

$$\begin{aligned} \sum_n E_{s_n} - E_{\text{neutral}_n} > 0 \qquad & \text{for rising BEMF} \\ \sum_n E_{s_n} - E_{\text{neutral}_n} < 0 \qquad & \text{for falling BEMF} \end{aligned} \tag{2.11}$$

The principle is illustrated on the oscillogram 2.12.

2.2.7.3 *Termination conditions*

Spin-up will be considered finished once the maximum commutation period defined by the parameter `mot_comm_per_max` (page 31) *and* the minimum voltage `mot_v_min` (page 31) are reached. If the target commutation period is reached sooner than the voltage ramp comes to the final voltage level, the controller will significantly accelerate the ramp in order to reach the minimum operating voltage faster. Once the spin-up process is finished, the controller will abandon the spin-up mode and switch into the normal mode.

If the target commutation period could not be reached in the amount of time specified by the parameter

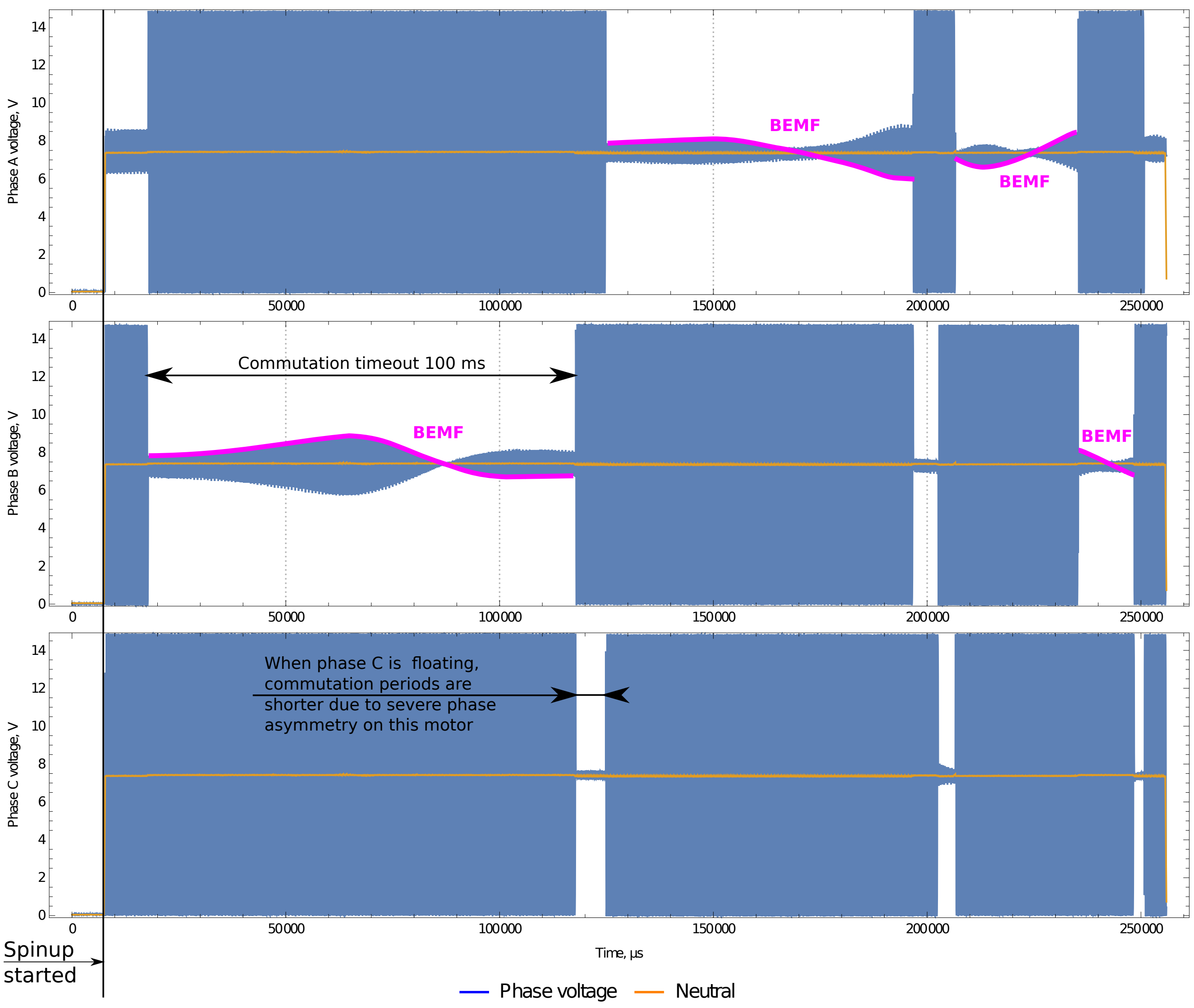

Figure 2.12: Spin-up sequence with a highly inertial load

`mot_spup_to_ms` (page 31), in milliseconds, the controller will abort the spin-up process, shut down the VSI and register a regular rotor stall event.

2.2.7.4 *Spin-up from a non-stationary state*

Sapog assumes that the rotor is (nearly) stationary in the beginning of the spin-up process. If this assumption is violated, there may be brief transient processes involving abnormal currents, high vibrations and high EMI.

The transient currents can be either direct or regenerative, depending on whether the desired and the actual directions of rotation match, and whether the initial guess of the current commutation step was correct.

Typically Sapog can establish an adequate field synchronization in a few commutation steps. If the rotor was rotating in the same direction but at a higher speed, it will be rapidly decelerated until the speed defined by the mechanical load and the source voltage is reached. If it is expected that the motor will be frequently started from a non-stationary state, it is recommended to either shorten the initial voltage ramp, or increase the initial spin-up voltage, in order to reduce the stresses caused by the initial braking.

2.2.7.5 *Configuration parameters*

This section summarizes the information about the configuration parameters pertaining to the spin-up mode. Read the preceding sections for a more detailed review.

- `mot_spup_blnk_pm` (page 31) - the duration of the extended blanking time during spin-up in permill (i.e. one-thousands; 10 permill = 1%) of the current commutation period.
- `mot_spup_to_ms` (page 31) - the overall spin-up timeout, in milliseconds. If a spin-up attempt could not be completed in this amount of time, the rotor will be considered stalled.

- mot_spup_st_cp (page 31) - the maximum commutation period for the spin-up mode, in microseconds. This is also the initial commutation period.
- mot_spup_vramp_t (page 31) - the duration of the initial voltage ramp, in seconds.
- mot_v_spinup (page 31) - the initial source voltage E_s in the beginning of the voltage ramp, in volts.

2.2.8 Open loop operation

While operating in the open loop mode, Sapog accepts setpoints in the range from 0% to 100%, and treats that as the fraction of the inverter supply voltage V_{inv} to deliver to the motor E_s:

$$E_s = V_{\text{inv}} \cdot \text{ramp}\,(\text{limit}\,(S_{\text{DC}})) \qquad \text{for open loop mode}, S_{\text{DC}} \in (0, 1] \tag{2.12}$$

where S_{DC} is the setpoint (DC stands for *duty cycle*), ramp is the setpoint ramp function, and limit is the speed limiting P-controller, reviewed below.

The ramp function replaces abrupt changes of the setpoint with gradual changes. Abrupt changes that are small enough to not be a hazard to the stability of the drive are accepted by the controller directly, bypassing the ramp. The following configuration parameters govern operation of the open loop setpoint ramp:

- mot_dc_slope (page 31) - the slope of the setpoint ramp, in full ranges per second. For example, the value of 1 allows the controller to sweep the setpoint from 0% to 100% in 1 second. The value of 10 allows the controller to sweep the setpoint from 0% to 100% in 0.1 seconds.
- mot_dc_accel (page 31) - the maximum change of setpoint that will be accepted directly, bypassing the ramp. For example, the value of 0.1 allows the controller to apply step changes up to 10% large directly.

Increase the parameters (either one or both) to improve the response characteristics of the drive. Reduce them if rapid setpoint changes cause stability issues.

Note that the minimal acceptable setpoint is constrained by the minimum duty cycle deduced from the parameter mot_v_min (page 31). If the setpoint falls into $\left(0, S_{\text{DC}_{\text{min}}}\right]$, where $S_{\text{DC}_{\text{min}}}$ is the minimum duty cycle deduced from the parameter mot_v_min (page 31), it will be silently overridden to $S_{\text{DC}_{\text{min}}}$.

The maximum acceptable setpoint will be automatically limited by means of a simple P-controller[2] if the motor approaches the maximum operating speed permitted by the current PWM carrier frequency, as shown on figure 2.10.

A zero setpoint is treated as the command to stop the motor. While the motor is stopped, the VSI is disabled, allowing the shaft to rotate freely.

2.2.9 RPM control loop

When operating in the RPM control mode, also known as RPM governor mode, Sapog strives to maintain a specified mechanical speed. The speed control is implemented by means of a parallel PID controller that takes the current mechanical RPM as input and produces the duty cycle setpoint at the output.

The nominal update rate of the PID controller T_{RPMPID} is defined as follows:

$$T_{\text{RPMPID}} \approx \max\left(10^{-3}, T_{\text{comm}}\right) \tag{2.13}$$

where T_{comm} is the current commutation period. Update of the RPM control loop is not a hard real time process, and the software is allowed to deviate.

The RPM control loop relies on the duty cycle slope control function defined in the section 2.2.8, so the related configuration parameters are still applicable. The speed limiting P-controller that is used in the open loop mode, however, is entirely bypassed in the RPM mode. The maximum supported speed is dependent on the PWM carrier frequency; see figure 2.10 for details. Equation 2.2 defines the relation between electrical and mechanical angular rates.

If the commanded setpoint is lower than the value specified by the parameter mot_rpm_min (page 31), the actual setpoint will be overridden to match this value. A zero setpoint is treated as a command to stop the motor. While the motor is stopped, the VSI is disabled, allowing the shaft to rotate freely.

The RPM PID controller does not wind up the integral term if the output is constrained by external factors, such as:

- Duty cycle ramp.

[2] Proportional controller, i.e. only P term of a PID controller is implemented.

- Current limiting controller.
- Maximum RPM limit.

The configuration parameters listed below govern the operation of the RPM control loop. The symbol DC refers to the duty cycle unit ranging in $[-1, 1]$, where negative values correspond to negative inverter source voltages E_s.

- `mot_num_poles` (page 31) - the number of magnetic poles on the rotor, always even, always ≥ 2. This parameter is needed for the controller to convert the electrical speed RPM_e into mechanical RPM_m.
- `rpmctl_p` (page 31) - proportional term of the RPM PID controller, in $\frac{\text{DC}}{\text{RPM}_m}$.
- `rpmctl_i` (page 31) - integral term of the RPM PID controller, in $\frac{\text{DC}}{\text{second}\cdot\text{RPM}_m}$.
- `rpmctl_d` (page 31) - derivative term of the RPM PID controller, in $\frac{\text{second}\cdot\text{DC}}{\text{RPM}_m}$.
- `mot_rpm_min` (page 31) - the minimal mechanical RPM at which stable operation is guaranteed.

The overall RPM control loop output for a time step n can be approximated with the following equations:

$$e(n) = \omega_{m_n} - S_{\omega_{mn}} \tag{2.14}$$

$$E_{s_n} = V_{\text{inv}_n} \cdot \text{ramp}\left[\underbrace{\frac{e(n) - e(n-1)}{T_{\text{RPMPID}}}\Pi_{\text{rpmctld}}}_{\text{derivative}} + \text{clamp}\left(\underbrace{\sum_{i=0}^{n} e(i)\, T_{\text{RPMPID}} \Pi_{\text{rpmctli}}}_{\text{integral}}\right) + \underbrace{e(n)\,\Pi_{\text{rpmctlp}}}_{\text{proportional}}\right] \tag{2.15}$$

where:

- ramp - the slope control function defined in section 2.2.8.
- clamp - the integrator wind-up control function described in this section.
- ω_m - current speed, in mechanical RPM.
- S_{ω_m} - target speed setpoint, in mechanical RPM.
- T_{RPMPID} - instant control loop update period. See equation 2.13.
- Π_{rpmctlp} - proportional term of the PID controller.
- Π_{rpmctli} - integral term of the PID controller.
- Π_{rpmctld} - derivative term of the PID controller.

2.2.10 Command timeout

Every setpoint supplied to the motor control logic, regardless of the operating mode, is accompanied with a time interval that specifies its lifetime – setpoint TTL[3]. If the time interval expires before the next setpoint is received, Sapog will automatically stop the motor.

This safety feature ensures that the motor will automatically stop in the event of a communication failure.

The exact duration of the setpoint TTL depends on the communication interface which provides the command. This is reviewed in detail in the chapter 4.

2.2.11 Current and voltage measurements

2.2.11.1 Measurement

Sapog measures the DC inverter current I_{inv} by means of a current shunt installed on the ground path, the output of which is connected to the ADC via a current amplifier with a fixed gain coefficient of 10.

The inverter supply voltage V_{inv} is measured through a resistor divider with the same division ratio as that of the BEMF feedback circuits. This division ratio is hard-coded per every supported hardware design and cannot be changed by the user.

Measurements of the inverter DC current I_{inv} and voltage V_{inv} are filtered with a first-order low-pass filter. The corner frequency of the filter is specified by the parameter `mot_lpf_freq` (page 31), in hertz.

2.2.11.2 Current limiting

Sapog can limit the maximum DC bus current with the help of a simple P-controller that activates when the instant current exceeds the limit. The gain of the current limiting P-controller is specified via the configu-

[3]Short for time-to-live.

ration parameter `mot_i_max_p` (page 31), in $\frac{\text{DC}}{\text{ampere}}$, where DC is the duty cycle unit ranging in $[-1,1]$. The maximum current is specified via the configuration parameter `mot_i_max` (page 31), in amperes.

2.3 Self diagnostics

Sapog contains a set of extensive diagnostic routines that can detect problems with the hardware. The diagnostic routines are executed automatically at start up. Additionally they can be run by an external request at any time, as long as the motor is not running.

While the tests are being executed, their progress and status information are printed to the CLI. The user can understand the nature of detected failures by reading the printed test reports.

If at least one test fails at start up, the controller will enter the panic mode, where it will remain until power cycled. Learn more about this in the chapter 3.

2.3.1 Power stage test

During the power stage test, Sapog modulates a voltage sequence shown on the table below and then validates the acquired BEMF samples. The following legend applies:

- L - low voltage level. The low side transistor is constantly turned on.
- H - high voltage level. A high duty cycle PWM is modulated on the phase.
- F - the phase is floating. Both high and low transistors are turned off.

Step	Phase A	Phase B	Phase C
1	L	F	F
2	H	F	F
3	F	L	F
4	F	H	F
5	F	F	L
6	F	F	H

The voltage at the output of the BEMF feedback circuits is sampled once for each non-floating phase; which produces six samples $Q_1 \ldots Q_6$. Then the following assertions are verified:

$$\begin{aligned} Q_1 &< L \\ Q_3 &< L \\ Q_5 &< L \\ Q_2 \approx Q_4 &\approx Q_6 \end{aligned} \tag{2.16}$$

where L is a specific lower threshold defined by the hardware implementation, typically at the level corresponding to 5% of the dynamic range of BEMF.

The test is considered passed only if all assertions are true. It is easy to see that this test is invariant to the fact whether the motor is connected or not.

Malfunctions in the following components may cause this test to fail:

- VSI transistors.
- VSI transistor drivers.
- PWM module of the microcontroller.
- BEMF feedback circuits.
- ADC module of the microcontroller.

2.3.2 Phase cross-conductivity test

The cross-conductivity test allows the controller to detect short-circuits between any of the three phases or their BEMF feedback circuits. During this test, the sequence shown in the table below is modulated at the output of the VSI. The following legend applies:

- H - high voltage level. A high duty cycle PWM is modulated on the phase.
- F - the phase is floating. Both high and low transistors are turned off.

Step	Phase A	Phase B	Phase C
1	H	F	F
2	F	H	F
3	F	F	H

At each step, voltages of all three phases are sampled, which yields the set of samples shown in the next table. It is assumed that the phases that are floating will be pulled down by the resistor dividers in their BEMF feedback circuits.

Step	Phase A	Phase B	Phase C
1	Q_{1A}	Q_{1B}	Q_{1C}
2	Q_{2A}	Q_{2B}	Q_{2C}
3	Q_{3A}	Q_{3B}	Q_{3C}

The following assertions are then validated:

$$\begin{array}{ccc} Q_{1A} > H & Q_{1B} < L & Q_{1C} < L \\ Q_{2A} < L & Q_{2B} > H & Q_{2C} < L \\ Q_{3A} < L & Q_{3B} < L & Q_{3C} > H \end{array} \tag{2.17}$$

where H and L are arbitrary hardware-defined thresholds, high and low, respectively.

Failure of at least one assertion against H indicates that the corresponding phase or its BEMF feedback circuits are not functioning correctly. Failure of at least one assertion against L indicates that there is a short circuit between the current high phase and the phase of the column of the failed assertion.

It is easy to see that the test will always be failing as long as the motor is connected. Sapog can recognize that the motor is connected by observing all phases shorted, in which case the results of this test will be discarded and the test will be considered passed.

2.3.3 Inverter voltage and current feedback test

Over the course of this test, Sapog merely verifies that the measurements reported by the current and voltage measurement blocks are sane and fall within the expected ranges, which are hardware-defined.

This test can detect issues with the respective measurement circuits, ADC, and the power supply of the microcontroller.

2.3.4 Temperature monitoring

Sapog periodically queries the digital temperature sensor installed on the power stage. This information is reported outside via communication interfaces, but not used by Sapog itself.

3 Indication

Sapog is equipped with means of visual and audial indication that are reviewed in this chapter.

3.1 RGB LED

The RGB LED is used to indicate the status of the firmware or the status of the bootloader while it is running. It also can be controlled externally via one of the available communication interfaces. In the latter case, the status indication output is overridden in favor of the externally received indication commands.

3.1.1 Firmware status indication

While the firmware is running, the following LED colors can be used to reflect its status.

Table 3.1: Firmware status indication via RGB LED

State		Meaning
Solid white	⬜	Initialization is in progress, not ready to work yet.
Solid green	🟩	Functioning normally.
Blinking cyan	🟦⬛🟦⬛🟦	UAVCAN auto-enumeration is in progress, awaiting user input.
Solid yellow	🟨	Controller is locked (see section 2.2.6), or the temperature sensor is malfunctioning.
Solid red	🟥	Fatal error, or self test failure. Use CLI to obtain detailed info.

3.1.2 Bootloader status indication

The bootloader starts immediately after the device is powered on. It runs for several seconds, and then, unless there is no valid firmware to boot or an update of the firmware is requested, the bootloader starts the firmware. More information on the bootloader and its state indication features is available in section 6.

3.1.3 External control

The RGB LED can be controlled by external commands via the available communication interfaces. External control overrides the state indication capabilities of the LED.

3.2 Sound

Sapog can modulate arbitrary sounds by applying specific low-power voltage patterns to the motor windings. The resulting currents cause the windings to vibrate, which produces audible sounds at the frequency of the applied voltage.

This feature can be invoked by an external command via the supported communication interfaces.

At start up, Sapog generates two short beeps with the following parameters: frequency 1 kHz, duration 100 ms each, spacing 200 ms.

Table 3.2: Sound modulation capabilities

Parameter	Min	Max	Unit
Audio frequency	100	5000	Hz
Duration of modulated sound	1	100	ms

If the requested parameters fall outside of the limits specified in the table, Sapog will automatically constrain the parameters in order to produce the closest achievable results.

Sound modulation requests are only processed if the controller is idling and the motor is stopped. Otherwise, sound modulation requests will be silently ignored.

4 Communication interfaces

Sapog is equipped with multiple communication interfaces. Not all communication interfaces can be disabled, and therefore, more than one interface can be active at the same time.

It is easy to see that this arrangement may lead to issues if more than one interface is used to command the controller. Sapog does not attempt to resolve such ambiguity, simply accepting every command without regard to what interface it was received from. This trait should be kept in mind when using multiple interfaces concurrently. For example, a common cause of confusion is encountered when Sapog is connected to a UAVCAN bus which commands a zero setpoint periodically, and some other interface at the same time is commanding to start the motor. The controller would appear to behave erratically, which is caused by alternating conflicting commands.

4.1 UAVCAN

4.1.1 Overview

This section contains the documentation for the UAVCAN interface implemented in Sapog. A general overview and the specification of the UAVCAN protocol is provided on the official website at `http://uavcan.org`.

The UAVCAN interface enables access to all features of Sapog. It is possible to control the motor in all operating modes via UAVCAN, change the configuration parameters, emit audial and visual indications, upgrade the firmware, and perform automated ESC enumeration.

This section documents the UAVCAN interface as implemented in the firmware, omitting the logic specific to the bootloader, which is documented separately in the chapter 6.

Sapog employs Libuavcan – an open source implementation of the UAVCAN protocol stack in C++ distributed under the MIT software license.

The CAN interface recovers from the bus-off state automatically once the controller has observed 128 occurrences of 11 consecutive recessive bits on the bus, as defined by the CAN specification.

4.1.2 Basic functions

4.1.2.1 Node status reporting

The standard node status message `uavcan.protocol.NodeStatus` is broadcasted at 1 hertz. The mode and health codes are summarized below.

Table 4.1: UAVCAN node health code interpretation

Node health	Meaning
OK	Operating normally.
WARNING	Not used.
ERROR	Not used.
CRITICAL	The controller is locked up (see section 2.2.6), or the temperature sensor is malfunctioning.

Table 4.2: UAVCAN node mode code interpretation

Node mode	Meaning
OPERATIONAL	Operating normally.
INITIALIZATION	The firmware has just started and is not ready to begin normal operation yet.
MAINTENANCE	See the chapter 6 about the bootloader.
SOFTWARE_UPDATE	See the chapter 6 about the bootloader.
OFFLINE	Not applicable.

The vendor-specific status code field is not used by Sapog.

The node uptime is reported from the moment when the firmware is started. The time while the bootloader was running is not included in the reported uptime value.

4.1.2.2 *Node identification*

The service `uavcan.protocol.GetNodeInfo` is responded to as follows.

All fields of the nested structure `uavcan.protocol.SoftwareVersion` are populated, which are `major`, `minor`, `vcs_commit`, and `image_crc`. The VCS commit field is assigned the most significant bits of the git commit hash of the running revision of the firmware. The image CRC is computed for the entire firmware image padded to 8 bytes; the bytes of the image that are occupied by the stored image CRC are assumed to be zero while the image CRC is being computed.

The following fields of the nested structure `uavcan.protocol.HardwareVersion` are always populated: `major`, `minor`, and `unique_id`. The values of the fields `major` and `minor` are specific to the particular hardware model that Sapog is running on. Please refer to the documentation supplied with your hardware for the specific values. The field `certificate_of_authenticity` is populated only if any certificate of authenticity is provided by the vendor.

The field `name` is set to the string `io.px4.sapog`.

4.1.2.3 *Node restarting*

The service `uavcan.protocol.RestartNode`, if the provided magic number is correct, unconditionally stops the motor and reboots the controller. If the provided magic number is incorrect, Sapog returns a response with the field `ok` set to zero (false).

4.1.2.4 *Interface statistics*

The service `uavcan.protocol.GetTransportStats` returns the current statistic counters for both supported CAN interfaces, even if the hardware uses only one of them. All fields of all nested structures are populated.

4.1.2.5 *Data type information*

The service `uavcan.protocol.GetDataTypeInfo` provides an extensive information about the supported UAVCAN data types. No special cases apply.

4.1.3 Initialization

Sapog is a full plug-and-play UAVCAN node that requires no mandatory initial configuration prior to use.

4.1.3.1 *CAN bus bit rate detection*

Once started, Sapog will automatically detect the bit rate of the CAN bus it is connected to (if connected to any at all), and remember the detected bit rate until the next boot up. There is no detection timeout, which means that Sapog can be connected to a CAN bus at any moment after powering up, and it will configure itself immediately.

It is not possible to specify the bit rate manually.

Sapog requires up to approximately 4 seconds to perform the bit rate detection on a properly functioning CAN bus. If the bus is exhibiting erroneous behavior, Sapog may need a longer time to complete the bit rate detection procedure.

The following bit rates can be detected by Sapog automatically:

- 1 Mbit/s
- 500 kbit/s
- 250 kbit/s
- 125 kbit/s

Sapog cannot be interfaced with a CAN bus that operates at a different bit rate.

Note that if the bootloader (section 6) was able to detect the bit rate of the CAN bus before starting the application, it will pass the detected value over to the application while booting it, in which case the application will immediately start using the supplied value rather than performing the bit rate detection again.

4.1.3.2 *Node ID allocation*

The configuration parameter `uavcan_node_id` (page 31), when set to a non-zero value, defines the node ID of the local UAVCAN node.

If this parameter is set to zero, Sapog will request a dynamic UAVCAN node ID from the bus, which is the default behavior.

Note that if the bootloader (section 6) was able to obtain a dynamic UAVCAN node ID from the bus before starting the application, it will pass the detected value over to the application while booting it, in which case the application will use the provided node ID *regardless of the value configured in* `uavcan_node_id` *(page 31)*.

Until there is a valid node ID available for the local UAVCAN node (either specified statically via the configuration parameter, or provided dynamically), no other functions of the UAVCAN interface will work. This implies, for example, that it is not possible to control the motor via UAVCAN until the local node is fully configured.

4.1.4 Motor control

4.1.4.1 *Setpoint input*

Motor control is governed by means of two standard message types that Sapog subscribes to. The message `uavcan.equipment.esc.RawCommand` puts Sapog into the open loop control mode (section 2.2.8). The message `uavcan.equipment.esc.RPMCommand` puts Sapog into the RPM loop control mode (section 2.2.9).

Both messages carry arrays of commands for multiple motors. Sapog selects the single command from the array at the index specified by the configuration parameter `esc_index` (page 31). If the specified index points out of the range of the array, Sapog assumes that the commanded setpoint is zero, and stops the motor.[4]

Sapog does not perform any additional processing of the received command message, and does not prioritize between the supported types of messages and between different broadcasters. This means that the drive will not operate properly if there are multiple types of control messages being received, or if multiple nodes are broadcasting conflicting setpoints. Direct execution of all received commands without additional processing allows Sapog to achieve minimal response latency and high command throughput.

Setpoints received via UAVCAN are assigned a fixed lifetime, which is specified by the configuration parameter `cmd_ttl_ms` (page 31). Read more on the concept of setpoint TTL in section 2.2.10.

As a safety precaution, when the motor is stopped and a non-zero open loop setpoint is received, Sapog will refuse to start the motor if the received non-zero open loop setpoint exceeds the threshold value specified by the configuration parameter `cmd_start_dc` (page 31), in the duty cycle units. This feature helps the user to avoid starting the motor accidentally to a high power level, for example, due to a failure in the system that controls Sapog. This feature is only available for the open loop control mode.

4.1.4.2 *Status reporting*

The standard message type `uavcan.equipment.esc.Status` is used to report the status of the motor control system. The message is broadcasted at 1 hertz if the motor is not running, and at 10 hertz if the motor is starting or running. All fields of the message are populated.

The field `error_count` carries the number of failures to detect a valid zero crossing point since the motor has started. The error counter may overflow if the motor is running without stopping for a very long time, although this will never happen as long as the drive is functioning correctly.

The field `esc_index` contains the value of the configuration parameter `esc_index` (page 31).

4.1.5 Configuration parameter management

The standard UAVCAN configuration parameter management interface is supported by means of the service data types defined in the namespace `uavcan.protocol.param`.

Note that the save action available via the service `uavcan.protocol.param.ExecuteOpcode` is supported, but is redundant, because Sapog commits all the configuration parameters to the non-volatile configuration storage memory automatically after modification.

When obtaining the list of all available configuration parameters using the field `index` of the service

[4] Obviously, multiple ESC can share the same ESC index setting.

uavcan.protocol.param.GetSet, the ordering of the returned configuration parameters is undefined, but it is guaranteed to be consistent within the same firmware build. Updating the firmware to different versions or even different builds of the same version may change the ordering of the configuration parameters.

4.1.6 Audial and visual indication

4.1.6.1 Audial indication

The standard message type uavcan.equipment.indication.BeepCommand can be used to invoke the audial indication feature of Sapog described in section 3.2.

4.1.6.2 RGB LED control

The standard message type uavcan.equipment.indication.LightsCommand can be used to control the RGB LED indicator via UAVCAN, as described in the section 3.1.3.

The configuration parameter light_index (page 31) defines the UAVCAN light ID of the RGB LED. Light control commands will only be accepted if the value of the field light_id matches the value stored in the configuration parameter light_index (page 31).

4.1.7 Firmware upgrade

The service uavcan.protocol.file.BeginFirmwareUpdate reboots the device into the bootloader mode. The bootloader mode is documented in the chapter 6.

If the motor is running at the time of reception of this request, it is stopped unconditionally.

Note that the request argument image_file_remote_path is always ignored. It is therefore required to invoke this service again after the firmware has entered the bootloader (i.e. about 3 ± 1 seconds after the first invocation), otherwise the upgrade process will not begin. This requirement may be removed in the future.

4.1.8 Automated ESC enumeration

Sapog allows the user to configure the ESC index (parameter esc_index (page 31)) and the direction of rotation (parameter ctl_dir (page 31)) intuitively, by manually rotating the rotors in a specific order and direction. This feature is only useful for multi-motor vehicles, such as multirotor UAV, and is entirely optional. The respective configuration parameters can be equally well configured manually by the user, instead of resorting to this more sophisticated method.

The UAVCAN namespace uavcan.protocol.enumeration contains the data type definitions that this feature is based upon.

When invoking the automated enumeration sequence, the field parameter_name of the service type uavcan.protocol.enumeration.Begin should be left empty (which means “parameter name auto-detect” according to the UAVCAN specification).

The user indication input is provided by means of turning the rotor manually in the normal direction of rotation. Sapog will memorize the direction of rotation and reflect it in the corresponding configuration parameter.

The enumeration request will return the error ERROR_INVALID_MODE if the motor is not stopped.

The user can benefit from this feature only if it is adequately supported by the controlling hardware installed on the vehicle (e.g. autopilot). Therefore, the full description of this feature falls outside of the scope of this manual. For reference, the automated ESC enumeration procedure performed with a PX4-based autopilot is demonstrated on the following video: https://youtu.be/4nSa8tvpbgQ.

4.1.9 Data type summary

Table 4.3: UAVCAN broadcast messages

Data type name	Frequency, Hz	Tr. priority	Note
`uavcan.protocol.NodeStatus`	1	16 (medium)	
`uavcan.protocol.dynamic_node_id.Allocation`	Aperiodic	30 (low)	Only during the initialization or while in the bootloader.
`uavcan.protocol.enumeration.Indication`	Aperiodic	16 (medium)	Only during ESC enumeration.
`uavcan.equipment.esc.Status`	1 or 10	16 (medium)	1 Hz while idle, 10 Hz otherwise.
`uavcan.protocol.debug.LogMessage`	Aperiodic	31 (lowest)	

Table 4.4: UAVCAN subscribed messages

Data type name	Note
`uavcan.protocol.dynamic_node_id.Allocation`	Only during the initialization or while in the bootloader.
`uavcan.equipment.esc.RawCommand`	Open loop setpoint.
`uavcan.equipment.esc.RPMCommand`	RPM control loop setpoint.
`uavcan.equipment.indication.BeepCommand`	Ignored if the motor is running.
`uavcan.equipment.indication.LightsCommand`	Overrides the RGB LED state indication.

Table 4.5: UAVCAN provided services

Data type name	Note
`uavcan.protocol.GetNodeInfo`	
`uavcan.protocol.GetDataTypeInfo`	
`uavcan.protocol.GetTransportStats`	
`uavcan.protocol.RestartNode`	
`uavcan.protocol.enumeration.Begin`	Returns an error if the motor is running.
`uavcan.protocol.file.BeginFirmwareUpdate`	Stops the motor and reboots into the bootloader.
`uavcan.protocol.param.ExecuteOpcode`	
`uavcan.protocol.param.GetSet`	

4.2 CLI

4.2.1 Overview

Sapog exposes a plain text command line interface (CLI) on its serial port. The CLI can be used for user interaction directly; also it can be used as a machine-to-machine command interface, as the command syntax is quite simple.

The serial port interface operates at 115200 baud, 8 bit per word, no parity control, one stop bit (115200-8N1).

The CLI line termination sequence is `\r\n` (CR, LF).

Barring any special preferences, it is recommended to use the software listed in table 4.6 for accessing the CLI from the user's PC.

Table 4.6: CLI terminal emulators

PC OS	Application	Note
GNU/Linux	`picocom`	A simple and robust command-line tool. It should be preferred over the popular `minicom` and `screen`.
MS Windows	PuTTY	A simple multi-purpose GUI terminal emulator. The HyperTerminal tool supplied by default with some versions of Windows should be avoided.

4.2.2 Boot logging

While in the process of booting, Sapog prints human-readable diagnostic messages via the serial port. The printed messages, among other things, include the reports of the start-up self-test procedures described in section 2.3.

Normally the boot-up messages should only be of interest for troubleshooting purposes.

4.2.3 Panic reporting

If Sapog encounters a catastrophic failure that renders it dysfunctional (e.g. a memory access violation, stack overflow, failure of the computing hardware, etc), it uses the serial interface as a last resort status reporting option. In this case, Sapog will dump as much status information as possible into the serial port, after which it will lock up and wait for the watchdog timer to reset it.

4.2.4 CLI commands

This section documents the CLI commands that can be of interest to the end user. Some commands that are not intended for use in production are intentionally omitted from this reference.

4.2.4.1 help

Prints the list of all available commands.

4.2.4.2 cfg

This command provides access to the configuration parameter storage. The configuration parameters and their non-volatile storage are described in more detail in the chapter 5.

Syntax:

- `cfg list` - list all configuration parameters, their current values, acceptable value intervals, and default values.
- `cfg save` - save the current configuration into the non-volatile storage. This command is redundant and normally need not be used, because Sapog commits all configuration into the non-volatile storage automatically upon modification.
- `cfg erase` - erase the current configuration and reset the non-volatile memory to the factory defaults.
- `cfg set <name> <value>` - assign the configuration parameter named `<name>` the value `<value>`. For example: `cfg set foo 42`. The non-volatile storage will be updated automatically.

The syntax `cfg list` prints one parameter per line, where the line is formatted as follows:

```
name = value [min, max] (default)
```

The number of whitespaces between tokens may vary.

Floating point parameters are always reported with the point symbol (`.`), which allows one to distinguish them from integer and boolean typed parameters.

Boolean parameters are reported as integers, where 1 represents the logical true and 0 represents the logical false. They can be distinguished from integer parameters by their minimum and maximum values being 0 and 1, respectively.

Whenever a parameter is assigned a new value, the device verifies if the new value is within the acceptable limits. If it is, the new value is assigned. Otherwise, the old value is retained. Afterwards the device prints out the resulting value of the parameter in the following format:

```
name = value
```

The number of whitespaces between tokens may vary.

The response can be used to check whether the new value was accepted by the device or not.

4.2.4.3 reboot

Reboot the firmware. If the motor is running, it will be stopped first.

4.2.4.4 beep

Modulate sound as described in section 3.2.

Syntax:

- `beep` - emit sound of the default duration at the default frequency.
- `beep <freq>` - modulate sound of the default duration at the frequency `<freq>` specified in hertz. For

example: `beep 1000`.
• `beep <freq> <dur>` - modulate sound of the specified duration `<dur>` in milliseconds at the frequency `<freq>` specified in hertz. For example: `beep 5000 200`.

4.2.4.5 *stat*

Print the current state of the motor controller in a human-readable form. At least the following information will be displayed:

- Inverter supply voltage.
- Inverter DC current.
- Mechanical RPM.
- Actual duty cycle (not the setpoint).
- The number of failures since the motor has started.

4.2.4.6 *test*

Execute the self-test routines, described in section 2.3. The output will be printed in a human-readable format.

4.2.4.7 *dc*

Set the open loop control setpoint. See section 2.2.8 for more information about the open loop control mode.

The setpoint TTL when using this command is fixed at 30 seconds. More information on the setpoint TTL feature is provided in section 2.2.10.

By default this command is locked for safety reasons. In order to unlock it, execute the command “`dc arm`” once, and it will stay unlocked until the firmware is rebooted.

Syntax:

- `dc` - assign a zero setpoint, which will stop the motor if it is running.
- `dc arm` - unlock the command.
- `dc <duty cycle>` - set the specified duty cycle setpoint. The setpoint is a real number in the interval $[0,1]$. For example: `dc 0.5` assigns the 50% duty cycle.

4.2.4.8 *rpm*

Set the RPM control loop setpoint. See section 2.2.9 for more information about the RPM control mode.

The setpoint TTL when using this command is fixed at 30 seconds. More information on the setpoint TTL feature is provided in section 2.2.10.

By default this command is locked for safety reasons. In order to unlock it, execute the command “`rpm arm`” once, and it will stay unlocked until the firmware is rebooted.

Syntax:

- `rpm` - assign a zero setpoint, which will stop the motor if it is running.
- `rpm arm` - unlock the command.
- `rpm <RPM>` - set the specified RPM setpoint. The setpoint is an integer number in the interval $[0,65535]$. For example: `rpm 5000` assigns the target setpoint of 5000 mechanical RPM.

4.2.4.9 *md*

Print the internal motor controller status information in a human-readable format. The output of this command is also printed automatically when the rotor stalls.

4.2.4.10 *zubax_id*

This command reports the complete information identifying this particular product: type, version information, make and model. The information is printed in a machine-readable yet human-friendly YAML[5] format.

An example output is shown in the following listing. The meaning of each well-defined field is explained in table 4.7. Fields that are not intended for production use are intentionally left undocumented. Note that the number of fields and their ordering may vary.

[5] https://en.wikipedia.org/wiki/YAML

```
product_id   : 'io.px4.sapog'
product_name : 'PX4 Sapog'
sw_version   : '2.0'
sw_vcs_commit: 7672820
sw_build_date: Apr 29 2017
hw_version   : '1.1'
hw_unique_id : 'M//UBUdHNTZUVxlDAAAAAA=='
hw_signature : 'NX3Y05Y1a/hDNajPyMF5dojuAB9r1HmUxZ3UVU8g39C9L5pjAZMVKdZ7yI1CetvumOJZHIjRYkHOBmMAEKvRU/3mpG\
+KgUND6yGpoHQ5aa+OK+LIcOw2LC+BaUmAPS6Hy4bxOvuARb062QruQlrOXyVc/vs28JtcBzOZo/b/OxY='
hw_info_url  : 'http://device.zubax.com/device_info?uid=33FFD405474735365457194300000000'
```

Table 4.7: Zubax ID fields

Field name	Meaning
`product_id`	Product type identifier string. It is always set to `io.px4.sapog`. The same string is reported via UAVCAN as the node name string.
`sw_version`	Firmware version number in the form "major.minor".
`sw_vcs_commit`	Firmware patch ID as the short git commit hash. The abbreviation VCS stands for "version control system". This number allows one to determine the exact revision of the firmware that is currently running.
`sw_build_date`	Firmware build date in the form "`mmm dd yyyy`".
`hw_version`	Hardware version number in the form "major.minor". Refer to the documentation provided with your hardware to learn what version number it reports to Sapog.
`hw_unique_id`	The 128-bit unique ID of this specific hardware instance. The UID is represented either as a hexadecimal string, or as a Base64 encoded string.
`hw_signature`	The certificate of authenticity (CoA) of this specific hardware instance encoded in a Base64 string. This field will not be present if the current hardware instance does not have any CoA installed. Refer to the documentation provided with your hardware to find out what CoA is used, if any.
`hw_info_url`	Link to the web page that contains the test report, origin information, traceability data, and other important information about this specific hardware instance provided by the vendor. This field may not be populated if the hardware does not have any CoA installed.

Vendors that use Sapog for their own products can use this command to install their own CoA on the hardware. Beware that the CoA can be written only once. The following form is used to write the CoA: `zubax_id <CoA>`, where `<CoA>` is a placeholder for a Base64-encoded certificate of authenticity. More information about using Sapog in custom products is provided in the chapter 8.

Syntax:

- `zubax_id` - print the Zubax ID YAML structure documented above.
- `zubax_id <CoA>` - install the vendor's certificate of authenticity `<CoA>` encoded in Base64. This command can be executed only once.

4.2.4.11 *uavcan*

Print the status information of the local UAVCAN stack in a human-readable format. This command may take several seconds to execute, depending on the state of the UAVCAN stack.

4.3 RCPWM

4.3.1 Overview

RCPWM is a simple analog interface that encodes the command in a series of repeating pulses. The commanded setpoint is proportional to the duration of the pulses, whereas the spacing between the pulses and their interval do not carry any useful information. The principle is illustrated on figure 4.1.

The RCPWM interface is widely used in robotics and in flight control systems of small UAV. Its obvious downsides are the lack of any kind of feedback from the controlled system, very low throughput, high latency (in excess of 2 milliseconds in certain scenarios, and never lower than 1 millisecond), and general physical inefficiency due to the need to use one independent electrical connection per controlled system.

4.3.2 Functional description

When the RCPWM interface is enabled, Sapog demodulates the RCPWM signal into the open loop mode setpoints, and feeds them into the motor controller. Setpoints received via RCPWM are accepted immediately

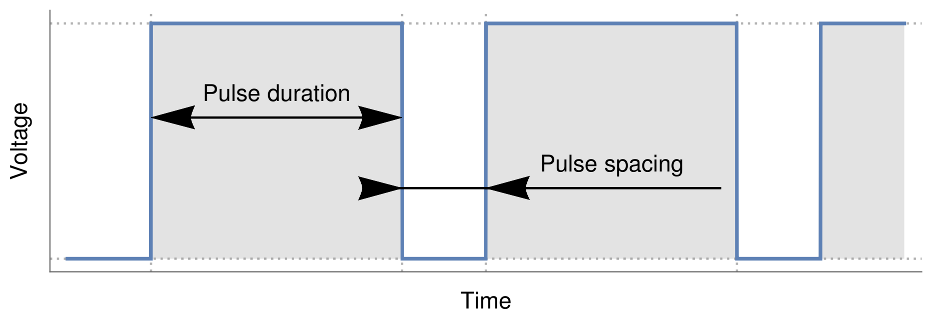

Figure 4.1: RCPWM signal

without pre-filtering or any other preprocessing.

In order to be understood by Sapog, the RCPWM signal should match the requirements listed in table 4.8.

Table 4.8: RCPWM interface constraints

Symbol	Parameter	Min	Max	Unit
$\Delta t_{\mathrm{HL}_{\mathrm{RCPWM}}}$	RCPWM pulse duration	0.5	3	millisecond
$\Delta t_{\mathrm{LH}_{\mathrm{RCPWM}}}$	RCPWM pulse spacing	0.01	50	millisecond

If the RCPWM pulse duration $\Delta t_{\mathrm{HL}_{\mathrm{RCPWM}}}$ is not within the specified limits, Sapog treats it as if there was no modulation at all.

The setpoint TTL for RCPWM commands is fixed at 100 milliseconds. This implies that the motor will be stopped in 100 milliseconds after the RCPWM modulation is gone. More information on the setpoint TTL feature is provided in section 2.2.10.

The configuration parameter `pwm_enable` (page 31) specifies whether the RCPWM interface is active. If the interface is disabled, Sapog will ignore the state of the RCPWM input line. If the interface is enabled, care should be taken to ensure that no conflicting commands are being issued via other interfaces, otherwise the controller may exhibit an erratic behavior.

The configuration parameters `pwm_min_usec` (page 31) and `pwm_max_usec` (page 31) define the pulse duration that corresponds to the minimum (zero) and the maximum (100%) setpoints, respectively, in microseconds. Pulses shorter than `pwm_min_usec` (page 31) are interpreted as a zero setpoint; likewise, pulses that are longer than `pwm_max_usec` (page 31) are interpreted as the maximum setpoint.

Sapog applies a start-stop hysteresis that works as follows: if the motor is stopped, it will not start until the commanded setpoint exceeds 6%. If the motor is running, it will not stop until the commanded setpoint falls below 3%.

A safety feature will not allow the motor to start if the commanded setpoint exceeds 40%. Therefore, combining the effects of the hysteresis feature and the safety threshold, it can be seen that the motor will start only if the commanded setpoint falls into the interval $(6\%, 40\%)$.

The setpoint handling logic is illustrated on figure 4.2.

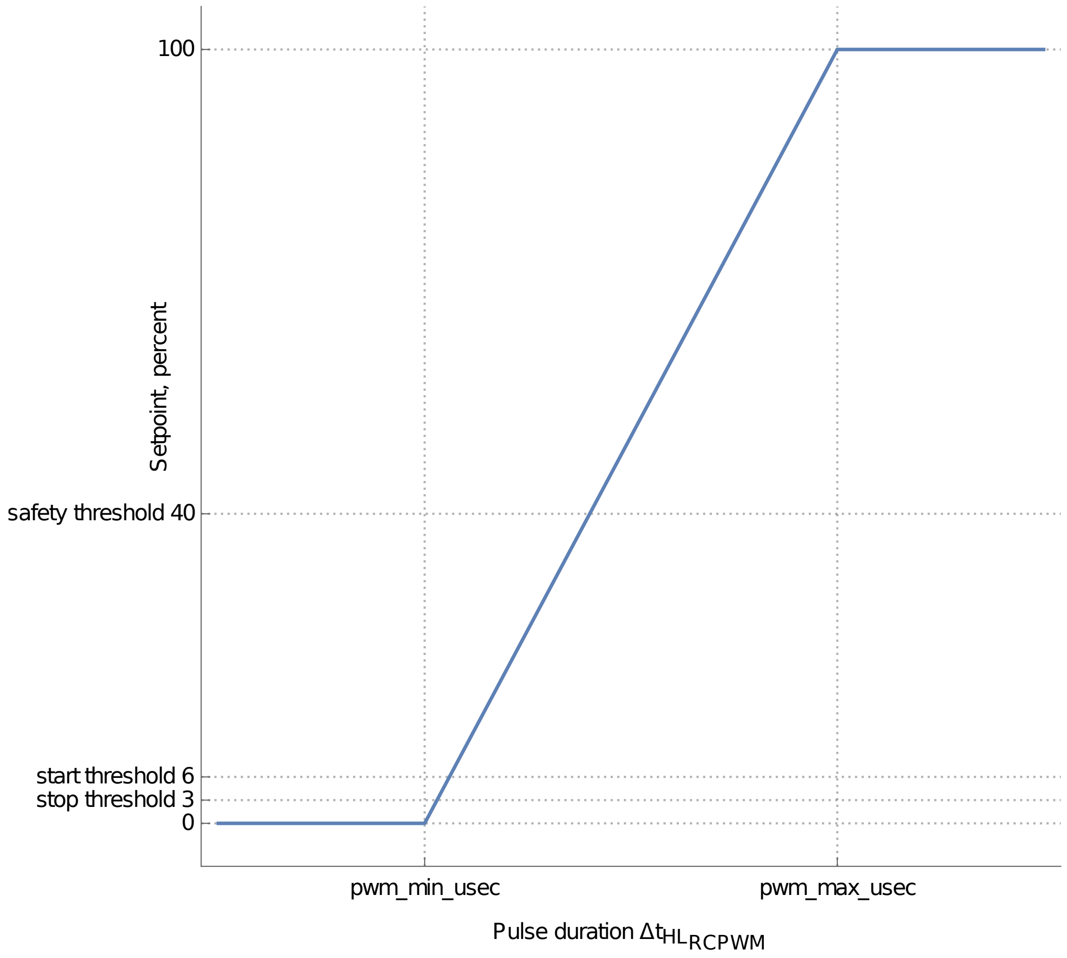

Figure 4.2: RCPWM setpoint handling

5 Configuration parameters

This chapter summarizes all configuration parameters available in Sapog.

The length of configuration parameter names does not exceed 16 characters, which ensures compatibility with the MAVLink protocol version 1.

5.1 Non-volatile configuration storage

All configuration parameters are stored in a non-volatile memory that retains its contents across power cycles.

Modification of any single parameter will trigger Sapog to commit all of them into the non-volatile memory one second after the last modification has taken place. The one second delay ensures that no excessive modifications of the non-volatile memory will be carried out when multiple configuration parameters are changed at the same time.

Sapog will not write or erase the non-volatile memory as long as the motor is running, in order to avoid interference with the hard real time motor control algorithms. If configuration was changed while the motor is running, a write operation will be postponed until the motor is stopped.

If the device is turned off while the configuration storage was being updated, the stored configuration data may get damaged.

The stored configuration is read from the non-volatile memory once upon boot-up. If Sapog detects that the stored configuration data is damaged, it will automatically revert it to the factory default state. Sapog can always reliably detect a damage of the stored configuration data, so it is guaranteed that an invalid configuration can never be loaded.

5.2 Firmware upgrade considerations

Configuration parameters of different revisions of the firmware may be incompatible with each other. For instance, some configuration parameters may be added, removed, or their value intervals may be changed.

Sapog always checks whether the stored configuration data is compatible with the current firmware revision. If it is detected that the stored configuration cannot be applied to the current version of the firmware, Sapog will automatically revert to the factory default configuration.

Keep these considerations in mind when upgrading the firmware.

5.3 Configuration parameter index

The minimum, maximum, and default values provided in the table are shown for exemplary purposes only, and they are *not expected to be valid* for all firmware revisions that this document applies to. The value intervals and default values may change in newer firmware revisions; furthermore, they can also depend on which particular hardware Sapog is running on.

The following abbreviations are adopted for the tables below:

TEA Short for “takes effect at”. This field specifies when the newly specified parameter value will take effect.

Def. Short for “default value”.

5.3.1 Motor control configuration parameters

Table 5.1: Motor control configuration parameters

Name	Unit	Pages	TEA	Min	Max	Def.	Description
`mot_pwm_dt_ns`	nanosecond	6	boot	400	800		PWM dead time
`mot_pwm_hz`	hertz	6, 11	boot	20000	75000	60000	PWM carrier frequency
`mot_spup_blnk_pm`	permill	10, 13, 14	boot	1	300	100	Min blanking time during spin-up
`mot_spup_to_ms`	millisecond	13, 14	boot	100	9000	5000	Spin-up timeout
`mot_spup_st_cp`	microsecond	12, 14	boot	10000	300000	100000	Spin-up max/initial comm. period
`mot_comm_per_max`	microsecond	5, 13	boot	1000	10000	4000	Max comm. period for normal mode
`mot_zc_fails_max`		12	boot	6	300	20	Stall detection threshold
`mot_bemf_range`	percent	10	boot	10	100	90	Valid BEMF magnitude
`mot_bemf_win_den`		7	boot	3	8	4	BEMF sampling window length denominator
`mot_blank_usec`	microsecond	10, 13	boot	10	300	40	Comm. blanking time for normal mode
`mot_tim_cp_min`	microsecond	6	boot	100	50000	600	Advance angle interpolation point X1
`mot_tim_cp_max`	microsecond	6	boot	100	50000	300	Advance angle interpolation point X2
`mot_tim_adv_max`	angular degree	6	boot	0	29	15	Advance angle interpolation point Y1
`mot_tim_adv_min`	angular degree	6	boot	0	20	5	Advance angle interpolation point Y2
`rpmctl_p`	$\frac{\text{dutycycle}}{\text{RPM}_m}$	16	boot	0	1	0.0001	RPM loop PID proportional gain
`rpmctl_i`	$\frac{\text{dutycycle}}{\text{second}\cdot\text{RPM}_m}$	16	boot	0	10	0.001	RPM loop PID integral gain
`rpmctl_d`	$\frac{\text{second}\cdot\text{dutycycle}}{\text{RPM}_m}$	16	boot	0	1	0	RPM loop PID derivative gain
`mot_stop_thres`		12	boot	1	100	7	Lock-up threshold
`mot_lpf_freq`	hertz	16	boot	1	200	20	Current/voltage low-pass filter corner frequency
`mot_i_max`	ampere	16	boot	1	60	20	Maximum inverter current
`mot_i_max_p`	$\frac{\text{dutycycle}}{\text{ampere}}$	16	boot	0.01	2	0.2	Current limiter proportional gain
`mot_rpm_min`	RPM	5, 15, 16	boot	50	5000	1000	Minimum setpoint for RPM control loop
`ctl_dir`		4, 23	boot	0	1	0	Direction of rotation: 0 - forward, 1 - reverse
`mot_num_poles`		16	boot	2	100	14	Number of magnetic poles on the rotor
`mot_dc_slope`	$\frac{\text{dutycycle}}{\text{second}}$	15	boot	0.1	20	5	Duty cycle ramp steepness
`mot_dc_accel`	duty cycle	15	boot	0.001	0.5	0.09	Duty cycle ramp bypass threshold
`mot_spup_vramp_t`	second	12, 15	boot	0	10	3	Duration of spin-up voltage ramp
`mot_v_spinup`	volt	12, 15	boot	0.01	10	0.5	Initial voltage of spin-up voltage ramp
`mot_v_min`	volt	5, 12, 13, 15	boot	0.5	10	2.5	Min inverter source voltage for normal mode

5.3.2 UAVCAN interface configuration parameters

Table 5.2: UAVCAN interface configuration parameters

Name	Unit	Pages	TEA	Min	Max	Def.	Description
`esc_index`		22, 23	boot	0	15	0	UAVCAN index of this ESC
`cmd_ttl_ms`	millisecond	22	boot	100	5000	200	Setpoint timeout
`cmd_start_dc`	duty cycle	22	boot	0.01	1	1	Spin-up safety threshold
`uavcan_node_id`		21, 22	boot	0	125	0	Node ID, zero for dynamic allocation
`light_index`		23	boot	0	255	0	UAVCAN index of the RGB LED

5.3.3 RCPWM interface configuration parameters

Table 5.3: RCPWM interface configuration parameters

Name	Unit	Pages	TEA	Min	Max	Def.	Description
`pwm_max_usec`	microsecond	28	boot	1800	2200	2000	Full throttle pulse duration
`pwm_min_usec`	microsecond	28	boot	800	1200	1000	Zero throttle pulse duration
`pwm_enable`		28	boot	0	1	0	Enable this interface

5.3.4 Forced rotation detection configuration parameters

The configuration parameters listed in this group are deprecated and scheduled for removal in one of the future revisions of the firmware.

Table 5.4: Forced rotation detection configuration parameters

Name	Unit	Pages	TEA	Min	Max	Def.	Description
`enum_max_step`	implementation defined		undefined	2000	100000	50000	Not for production use
`enum_steps`	implementation defined		undefined	6	200	20	Not for production use
`enum_bemf`	implementation defined		undefined	5	500	20	Not for production use

6 Embedded bootloader

6.1 Overview

Sapog employs the PX4 Brickproof Bootloader - an ultra compact open source UAVCAN bootloader designed for ROM-limited embedded applications.

The bootloader starts immediately after the device is powered on. Having started, the bootloader checks if there is a valid application image that can be executed. If there is one, the bootloader measures a 5 second timeout since the point of its initialization, and once the timeout has expired, the bootloader starts the application, unless an external entity has requested the bootloader to download a new application image. If there is no valid application image found, the bootloader will wait forever for a request to download a new application image.

The procedure of firmware update over UAVCAN is defined in the UAVCAN specification at the website `http://uavcan.org`.

6.2 Status indication

While the bootloader is running, the following RGB LED behaviors are defined.

Table 6.1: Bootloader status indication via RGB LED

State		**Meaning**
Solid red	🟥	Error. Possible reasons: could not erase the old firmware image; the file server returned an invalid or unexpected response; file server response has timed out; the CRC of the firmware image is invalid.
Solid white	⬜	Just started, initialization is in progress. This state is very transient.
Solid yellow	🟨	CAN auto bit rate detection.
Solid green	🟩	CAN auto bit rate detection finished. This state is very transient.
Solid magenta	🟪	UAVCAN dynamic node ID allocation started.
Solid blue	🟦	UAVCAN dynamic node ID allocation finished.
Solid cyan	🟦	Firmware image is being downloaded.
Off	⬛	The application is being started.

6.3 UAVCAN interface description

6.3.1 Overview

The bootloader operates on the CAN1 interface only. The CAN2 interface is not used while the bootloader is running. It is therefore mandatory to use CAN1 as the primary communication interface rather than CAN2.

6.3.2 Initialization

Having started, the bootloader attempts to detect the bit rate of the CAN bus it is connected to (if connected at all), unless it was started by a request sent to the application, in which case, if the application already knew the bit rate of the bus, it will pass it to the bootloader, allowing it to skip the bit rate detection procedure.

The bootloader can detect the following bit rates:

- 1 Mbit/s
- 500 kbit/s
- 250 kbit/s
- 125 kbit/s

Having successfully detected the bit rate, the bootloader will attempt to obtain a UAVCAN node ID, unless it was provided by the application. In the latter case, the bootloader will simply re-use the node ID provided by the application.

The initialization can be interrupted at any time due to expiration of the boot timeout. Should that happen, the bootloader will pass the bus parameters it has detected, which are either the bit rate, the node ID, or both, to the application, in order to speed up its initialization.

6.3.3 Node status and info reporting

The message `uavcan.protocol.NodeStatus` is reported by the bootloader with the field `mode` set to `MODE_INITIALIZATION` while the initialization is in progress, or to `MODE_SOFTWARE_UPDATE` while the update is in progress. The field `status` can be set either to `STATUS_OK` or `STATUS_CRITICAL`; the latter is used only if the update process has failed.

The service `uavcan.protocol.GetNodeInfo` is responded to with all the fields of the nested structure `software_version` zeroed, unless there is a valid application image found, in which case the fields will be set according to the information gathered from the firmware image. The fields of the nested structure `hardware_version` are always all set correctly, except for the field `certificate_of_authenticity`, which is always empty, even if there is a valid certificate installed.

6.3.4 Firmware update

6.3.4.1 Initialization

Having received an update request `uavcan.protocol.file.BeginFirmwareUpdate`, the bootloader verifies that the firmware image has an appropriate size by invoking `uavcan.protocol.file.GetInfo`. If the size of the file appears to be incorrect, or if the service call could not be completed after up to three retries, the bootloader aborts the process with an error. Note that if the process fails at this point, the old firmware remains intact.

6.3.4.2 ROM preparation

Next, the bootloader erases the old firmware image. From this point on, there is no valid application that can be booted.

6.3.4.3 Image downloading

Having prepared the ROM for writing the new image, the bootloader downloads the new firmware image by invoking the service `uavcan.protocol.file.Read` repeatedly while advancing the file read offset. If a read request times out in the process, the bootloader will try again up to three times. Shall all the attempts fail, the bootloader will abort the process and report failure.

6.3.4.4 Finalization

When the new image is fully downloaded, the bootloader verifies its integrity and aborts the process if there are issues detected. Otherwise, the application will be started immediately. The bootloader will pass the detected CAN bus bit rate and the node ID to the application in order to accelerate its initialization.

6.3.5 Logging

The bootloader emits the message `uavcan.protocol.debug.LogMessage` at certain moments during the update in order to indicate its progress or report issues.

The text field of the log messages always contains two ASCII letters. The first letter indicates the action, the second letter indicates the result of the action. The log source name is always set to the string “`Boot`”.

Table 6.2: UAVCAN bootloader logging – action codes

Code	Meaning
`I`	Initialization.
`G`	Invoking `uavcan.protocol.file.GetInfo`.
`E`	Erasing ROM.
`R`	Invoking `uavcan.protocol.file.Read`.
`P`	Writing ROM.
`V`	Verification of the downloaded image.
`F`	Finalization.

Table 6.3: UAVCAN bootloader logging – status codes

Code	Meaning
`s`	Started
`f`	Failed
`o`	OK

6.3.6 Data type summary

Table 6.4: UAVCAN bootloader broadcast messages

Data type name	Frequency, Hz	Tr. priority	Note
`uavcan.protocol.NodeStatus`	1	30 (low)	
`uavcan.protocol.dynamic_node_id.Allocation`	Aperiodic	30 (low)	Used only during the initialization.
`uavcan.protocol.debug.LogMessage`	Aperiodic	30 (low)	

Table 6.5: UAVCAN bootloader subscribed messages

Data type name	Note
`uavcan.protocol.dynamic_node_id.Allocation`	Used only during the initialization.

Table 6.6: UAVCAN bootloader provided services

Data type name	Note
`uavcan.protocol.GetNodeInfo`	Software version identification fields will be zeroed if there is no valid application image present.
`uavcan.protocol.file.BeginFirmwareUpdate`	

Table 6.7: UAVCAN bootloader invoked services

Data type name	Tr. priority	Note
`uavcan.protocol.file.GetInfo`	30 (low)	Invoked once before downloading the firmware file.
`uavcan.protocol.file.Read`	30 (low)	Used to download the firmware file.

7 Tuning guide

7.1 Overview

This chapter describes how to configure Sapog for optimal performance with a given motor in a particular application. The reader is assumed to have read the previous parts of the reference manual.

Further help can be gained via `https://forum.zubax.com`.

7.2 Basic configuration

7.2.1 Number of rotor magnetic poles

The number of rotor magnetic poles is required for the closed-loop speed control and measurement of the mechanical speed of the rotor.

The number of magnetic poles can be obtained from the motor specification; alternatively, it can be obtained by counting the number of magnets on the rotor.

The relevant configuration parameter is `mot_num_poles` (page 31).

7.2.2 Maximum current

The parameter `mot_i_max` (page 31) should be configured correctly to avoid overloading the motor. The maximum current allowed for the motor should be provided in the motor specification. For more information, refer to section 2.2.11.2.

7.2.3 Low speed operation

The default settings may be unsuitable for maintaining a stable operation of certain motors at low speeds. This especially affects large, high-power motors.

The parameters `mot_comm_per_max` (page 31) and `mot_v_min` (page 31) should be adjusted in the first order if necessary.

7.2.4 Response characteristics

The settings of the duty cycle ramp should be adjusted for the target application. Please refer to the sections 2.2.8 and 2.2.9 for details.

7.3 Advanced tuning

7.3.1 Field weakening

The field weakening feature (documented in section 2.2.3) is governed by the following set of configuration parameters: `mot_tim_adv_max` (page 31), `mot_tim_adv_min` (page 31), `mot_tim_cp_min` (page 31), `mot_tim_cp_max` (page 31).

The angles are specified in the electrical degrees of rotor position; one electrical degree is equivalent to 3 phase degrees. For example, 90° of phase advance is equivalent to 30° of electrical advance.

It is recommended to follow the algorithm described below to configure the advance angle settings:

1. Find out the idle commutation period of your motor. The commutation period (in seconds) can be derived from RPM using the following formula: 20/(NumPoles × RPM), where **NumPoles** is the number of magnetic poles of the rotor, and **RPM** is the angular velocity of the rotor in revolutions per minute.
2. Assign `mot_tim_cp_min` (page 31) the idle commutation period computed in the step 1.
3. Find out the commutation period when the motor is working at the nominal RPM. Use the formula defined in the step 1 if necessary.
4. Assign `mot_tim_cp_max` (page 31) the commutation period computed in the step 3.
5. Assign `mot_tim_adv_max` (page 31) the desired advance angle at the nominal load.

8 Porting guide

8.1 Overview

This section contains the information necessary for implementation of custom hardware designs compatible with the Sapog firmware.

The reader should be familiar with the firmware codebase in order to be able to implement a successful custom hardware design compatible with Sapog.

Questions concerning custom designs can be asked via `https://forum.zubax.com`.

8.2 Certificate of authenticity

Sapog can store an arbitrary chunk of data up to 256 bytes large in a dedicated ROM area. This data is supposed to be the certificate of authenticity (CoA) provided by the hardware vendor.

The CoA can be written using the command `zubax_id` described in the section 4.2.4.10. Beware that the CoA can be written only once.

8.3 Hardware design guidelines

8.3.1 Requirements

1. The state estimation methods used in Sapog require a high-speed inverter. When choosing the power transistors and their drivers, keep the dynamic characteristics in mind: the total switching time should be under 200 nanoseconds.
2. The RC filters in the phase voltage feedback circuits should be designed so that their cutoff frequency is around 19 kHz and the output impedance does not exceed 8 kΩ.
3. Unless the maximum supply voltage exceeds 25 V (Li-ion $LiCoO_2$ 6S), the DC link voltage measurement divider should not be changed from the default specified in the reference schematic.

8.3.2 Reference schematic

This section contains the schematic of the reference hardware design that can be used as a basis for custom hardware designs compatible with Sapog. The design files and additional information can be gathered from the Github repository at `https://github.com/PX4/Hardware`.

Note that the EEPROM memory chip is not currently used by the firmware, but it is required nevertheless because future versions of the firmware may need it.

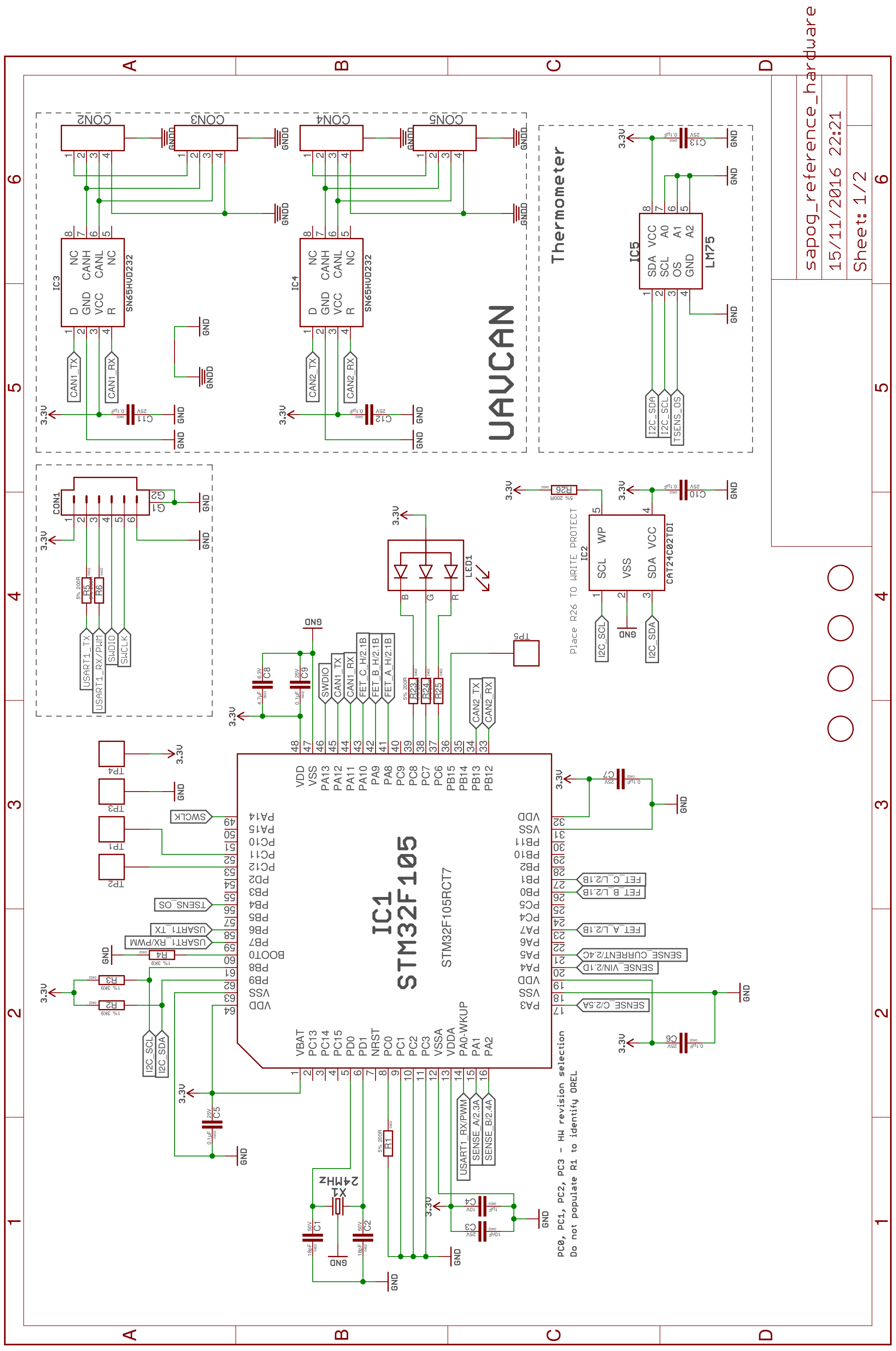

Figure 8.1: Reference schematic, page 1

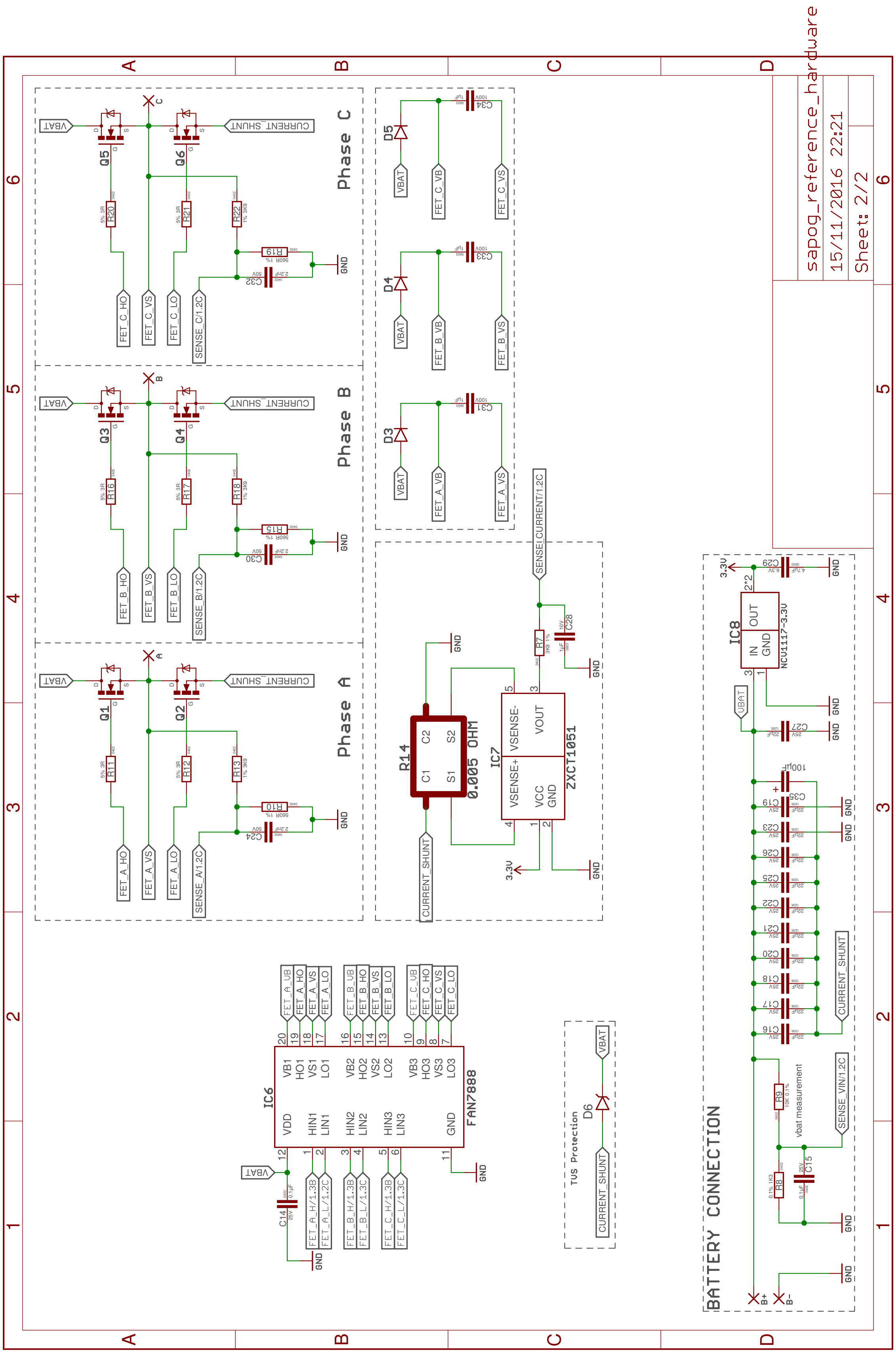

Figure 8.2: Reference schematic, page 2