Encoder Feedback

Background

Encoders are used to determine the rotor position and speed, and are the typical method of feedback to the control system in a motor drive. This document explains how to use the AMDC's encoder interface to extract high quality rotor position and speed data.

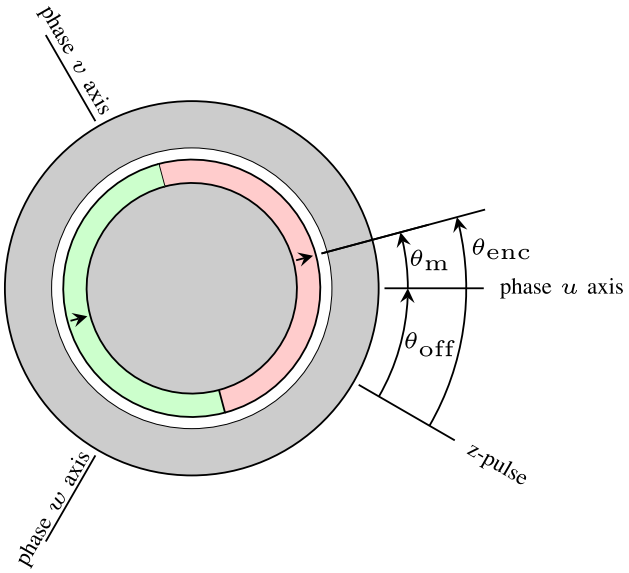
For more information:

- on how encoders work and are interfaced with the AMDC, see the encoder hardware subsystem page;
- on the driver functionality included with the AMDC firmware, see the encoder driver architecture page.

Rotor Position

The AMDC supports incremental encoders with quadrature ABZ outputs and a fixed number of counts per revolution `CPR` (for example, `CPR = 1024`). The user needs to provide code that interfaces to the AMDC's drivers to read the encoder count and convert it into usable angular information that is suitable for use within the control code.

This document assumes the configuration shown to the right, where the control code expects a measurement of the angle of the rotor's north pole relative to the phase u magnetic axis, labeled as a mechanical angle θ_{m}. The encoder provides θ_{enc}, which is the number of counts since the last z-pulse. The user's code needs to convert θ_{enc} (in units of counts) into θ_{m} (likely in units of radians) and handle an offset angle θ_{off} between the encoder's 0 position and the phase u axis.



Configuring the encoder

Upon powerup, the AMDC configures the encoder to a default number of counts per revolution. This is handled in `encoder.c` as part of the standard firmware package. When using an encoder that has a different number counts per revolution, the user must inform the driver by calling `encoder_set_counts_per_rev()`.

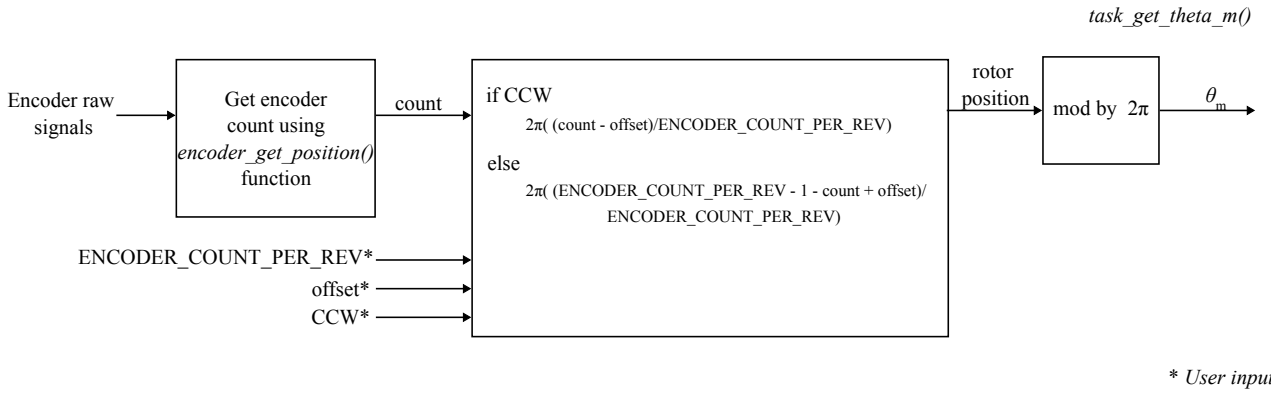
Example code for a 10 bit encoder:

```
#define USER_ENCODER_COUNTS_PER_REV_BITS (10)
#define USER_ENCODER_COUNTS_PER_REV (1 << USER_ENCODER_COUNTS_PER_REV_BITS)

int task_user_app_init(void)
{
    encoder_set_counts_per_rev(USER_ENCODER_COUNTS_PER_REV);

    // other user app one-time initialization code
    // ...
```

> **Tip**
>
> The AMDC provides a convenience function that can be used as an alternate to `encoder_set_counts_per_rev()` when the encoder is specified as a number of bits: `encoder_set_counts_per_rev_bits(USER_ENCODER_COUNTS_PER_REV_BITS).`

Converting the encoder count into rotor position

The recommended approach to reading the shaft position from the encoder is illustrated in the figure below:



First, the AMDC `drv/encoder` driver module function `encoder_get_position()` is used to obtain the the encoder's count θ_{enc} since the last z-pulse.

> **Tip**
>
> The `drv/encoder` driver module also has a function called `encoder_get_steps()` which returns the encoder's count since power-on. One rotation direction increments, the other decrements. This value does not wrap around (it ignores `encoder_set_counts_per_rev()` and the z-pulse). Users are advised to use `encoder_get_position()`, which does wrap around and tracks the z-pulse.

Next, the user should calculate θ_{m} from θ_{enc}. This is done by 1) removing the offset and 2) converting counts into radians. For the angles defined as shown in the image above, this is simply calculated as

$$\theta_{\text{m}} = \frac{2\pi}{\text{COUNTS\_PER\_REV}}(\theta_{\text{enc}} - \theta_{\text{off}}) \tag{1}$$

In this case, a counter-clockwise rotation of the rotor causes the θ_{enc} to increase. However, in some teststands a clockwise rotation causes θ_{enc} to increment. For these encoders, θ_{m} is calculated as

$$\begin{aligned}\theta_{\text{m}} &= \frac{2\pi}{\text{COUNTS\_PER\_REV}}(\text{COUNTS\_PER\_REV} - \theta_{\text{enc}} + \theta_{\text{off}}) \\ &= 2\pi - \theta_{\text{m,CCW}}\end{aligned} \tag{2}$$

> **Tip**
>
> The user can experimentally determine whether the encoder count increases with counter-clockwise rotation of the shaft by rotating the shaft and using logging to observe the trend of θ_{enc}.

Finally, the user must ensure that angle is within the bounds of 0 and 2π by appropriately wrapping the θ_{m}. This can be accomplished in C by using the `mod` function. This is shown in the final block in the diagram.

Here is example code to convert the encoder to angular position in radians (note that this assumes the encoder offset θ_{off} is already know; a procedure to determine this is described in the next subsection):

```
double task_get_theta_m(void)
{
    // User to set encoder offset
    double theta_off = 100;

    // User to set encoder count per revolution
    double ENCODER_COUNT_PER_REV = 1024;

    // User to set 1 if encoder count increases with CCW rotation of shaft, set 0
    int CCW_ROTATION_FLAG = 1;

    // Angular position to be computed
    double theta_m;

    // Get raw encoder position
    uint32_t theta_enc;
    encoder_get_position(&theta_enc);

    // Convert to radians
    theta_m = (double) PI2 * ( ((double)theta_enc - theta_off) / (double) ENCODER

    if (!CCW_ROTATION_FLAG){
        theta_m = PI2 - theta_m;
    }

     // Mod by 2 pi
    theta_m = fmod(theta_m,PI2);
    return theta_m;
}
```

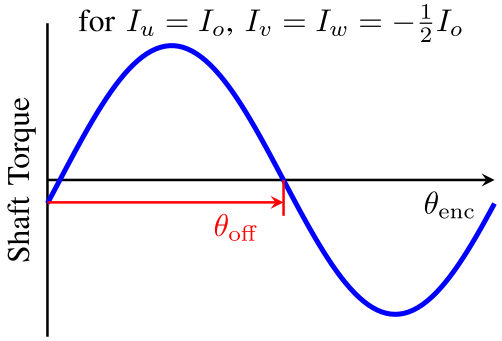
Finding the offset

The example code shown above makes use of an encoder offset value, `theta_off`. For synchronous machines, this offset is the count value measured by the encoder when the d-axis of the rotor is aligned with the phase U winding axis of the stator. This value typically needs to be found experimentally for each motor/encoder pair because it depends on how the encoder was aligned when it was coupled to the motor's shaft. This section provides a procedure to determine `theta_off`.

Determine the approximate offset

The approximate encoder offset can be found by taking advantage of the motor having the torque characteristic shown on the right. This depicts shaft torque as the shaft is rotated counter-clockwise and corresponds to the image at the start of the section; positive torque is in the counter-clockwise direction.

The following simple procedure can be used without any feedback control:

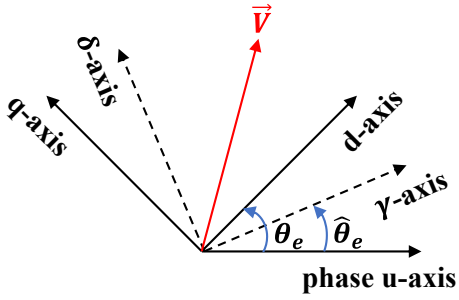


1. Set the `theta_off` to 0 in the control code `task_get_theta_m()`.
2. Eliminate any source of load torque on the shaft.
3. Power on the AMDC and rotate the rotor manually by one revolution (so that the encoder z-pulse is detected).
4. Align the rotor with the phase U winding axis by applying a large current vector at 0 degrees ($I_u = I_0$, $I_v = I_w = -\frac{1}{2}I_0$). This could be accomplished by:
   1. Using a DC power supply, or
   2. Injecting a current command on the d-axis using the AMDC Signal Injection module with `theta_m` fixed to 0.
5. Record the current encoder position and use this as the offset value: `theta_off = encoder_get_position();`.
6. Update the variable `theta_off` to the appropriate value in `task_get_theta_m()`.

Determine precise offset

Friction and cogging torque in the motor can decrease the accuracy of the estimate in Finding the offset. A more precise offset can be found by fine-tuning the `theta_off` value while using closed-loop control to rotate the shaft at different speeds and monitoring the observed d-axis voltage.

The correct offset is determined by considering how errors in the measured rotor angle impact the current controller's understanding of the $\mathrm{d}-\mathrm{q}$ reference frame. This is depicted in the figure on the right, where:



- $\hat{\theta}_{\mathrm{e}}$ is the incorrect eletrical angle (due to error in offset θ_{off}) that the controller is using
- the γ-δ vectors indicate where the controller mistakenly understands the d-q frame to be located based on $\hat{\theta}_{\mathrm{e}}$
- the d-q vectors and θ_{e} angle depict the actual d-q frame of the motor.

Note that $\theta_{\mathrm{e}} = p\theta_{\mathrm{m}}$ is the electrical angle where p is the number of pole-pairs of the motor and $\omega_{\mathrm{e}} = \dot{\theta}_{\mathrm{e}}$ is the electrical angular velocity with units of radians per second.

The voltage vector of the motor terminals $\vec{V}$ is shown in red and can be expressed in complex vector form as follows:

$$\vec{V} = R\vec{i} + L\frac{d\vec{i}}{dt} + j\omega_{\mathrm{e}}\lambda_{\mathrm{pm}}e^{j\theta_{\mathrm{e}}}$$

This voltage vector can be converted into the γ-δ reference frame as follows.

$$v_\gamma + j\, v_\delta = \vec{V}e^{-j\hat{\theta}_{\mathrm{e}}}$$

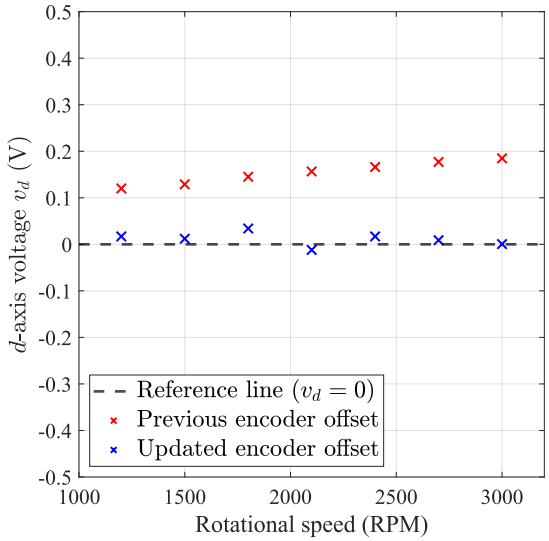
When the current commands are set to $\vec{i} = 0$, v_γ can be expressed as follows.

$$v_\gamma|_{\vec{i}=0} = -\omega_{\mathrm{e}}\lambda_{\mathrm{pm}}\sin(\theta_{\mathrm{e}} - \hat{\theta}_{\mathrm{e}})$$

If there is no estimation error (i.e., $\theta_{\mathrm{e}} - \hat{\theta}_{\mathrm{e}} = 0$), the γ-δ frame aligns with the d-q and the v_{d} value seen by the controller should be zero. Based on this fact, the following procedure describes how to determine the encoder offset by finding the condition where $v_\gamma = v_{\mathrm{d}} = 0$.

1. Configure the AMDC for closed-loop speed and DQ current control, and configure the operating environment to allow for quick edits to `theta_off` and for measuring the d-axis voltage commanded by the current regulator. Consider adding a custom command and using logging to accomplish this.
2. Command the motor to rotate at a steady speed under no-load conditions. Use the estimated `theta_off` obtained in Finding the offset.
3. Sweep `theta_off` over a small range around the initial estimate (e.g., ±5 counts). For each value, monitor the d-axis voltage and find the `theta_off` value that makes the d-axis voltage closest to 0 V. Identify this by observing when the sign of the d-axis voltage changes.
4. Repeat step 3 at multiple rotor speeds. At each speed, record the `theta_off` value that minimizes the d-axis voltage.
5. Take the average of the collected `theta_off` values from step 4 to determine the final encoder offset value.
6. Plot the d-axis voltage with the final offset against the different rotor speeds. The d-axis voltage should be close to zero for all speeds if the offset is tuned correctly.
7. In case there is an error in the offset value, a significant speed-dependent voltage will appear on the d-axis voltage. In this case, the user may have to re-measure the encoder offset.

An example of the results is shown in the plot below. After the calibration process, the updated encoder offset results in the d-axis voltage being closer to 0 across different speeds compared to the previous value.



Computing Speed from Position

The user needs to compute a rotor speed signal from the obtained position signal to be used in the control algorithm. There are several ways to do this.

Difference Equation Approach

A simple, but naive, way to do this would be to compute the discrete time derivative of the position signal in the controller as shown below. This can be referred to as Ω_{raw}.

$$\Omega_{\text{raw}}[k] = \frac{\theta_m[k] - \theta_m[k-1]}{T_s}$$

Unfortunately, using this approach results in noise in Ω_{raw} due to the derivative operation and the digital nature of the incremental encoder.

Low Pass Filter Approach

To solve this, *a low pass filter* may be applied to this signal. This is shown below to obtain a filtered speed, Ω_{lpf}.

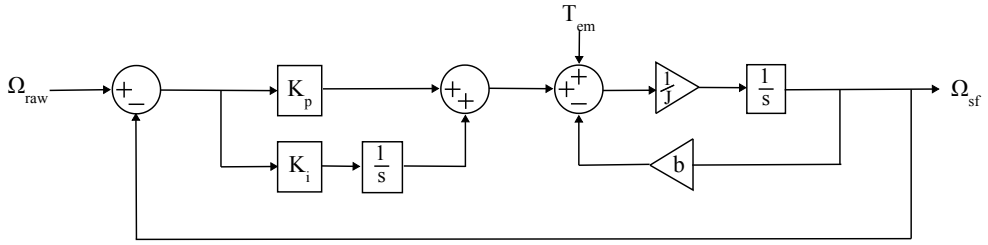
$$\Omega_{\text{lpf}}[k] = \Omega_{\text{raw}}[k](1 - e^{\omega_b T_s}) + \Omega_{\text{lpf}}[k-1]e^{\omega_b T_s}$$

Here, T_s is the control sample rate and ω_b is the low pass filter bandwidth. The user must select this bandwidth to obtain a sufficiently clean speed signal. The optimal bandwidth to use is going to vary based on the motor system. Typically, a bandwidth of 10 Hz is a reasonable starting point. This can be reduced if the speed signal remains too noisy, or increased for higher speed controls.

Note that this low pass filter approach will always produce a lagging speed estimate due to phase delay in the filter transfer function. This may be unacceptable higher performance motor control algorithms.

Observer Approach

To obtain a no-lag estimate of the rotor speed, users may create an observer [1], which implements a mechanical model of the rotor as shown below.



The estimate of rotor speed is denoted by Ω_sf. To implement this observer, the user needs to know the system parameters:

- `J`: the inertia of the rotor
- `b` the damping coefficient of the rotor.

It is also necessary to provide the electromechanical torque, T_{em} as input to the mechanical model.

The `PI` portion of the observer closes the loop on the speed, with Ω_raw being the reference input. The recommended tuning approach is as follows: $
$K_p = \omega_{sf} b, K_i = \omega_{sf} J$$

This tuning ensures a pole zero cancellation in the closed transfer function, resulting in a unity transfer function for speed tracking under ideal parameter estimates of `J` and `b`. An observer bandwidth of 10 Hz is typical of most systems, but similar to the low pass filter approach, users may need to alter this based on the unique aspects of their system.

References

1. R. D. Lorenz and K. W. Van Patten, “High-resolution velocity estimation for all-digital, AC servo drives,” in IEEE Transactions on Industry Applications, vol. 27, no. 4, pp. 701-705, July-Aug. 1991, doi: 10.1109/28.85485. keywords: {Servomechanisms;Optical feedback;Optical signal processing;Transducers;Signal resolution;Velocity measurement;Position control;Feedback loop;Velocity control;Noise reduction}