Mobile DCU Inverter Background and Simulink Simulation

UofT Formula Racing Team

Michael Strojny

October 2025

1 Background

The inverter applies a 3-phase AC voltage to three windings in the stator. This creates a magnetic pole pair every 120 degrees, creating a rotating magnetic field vector with speed ω_e; the poles of the rotor (rotating shaft) follow this vector with ω_m. We can transform the 3-phase flux abc into 2 dimensions dq: a d-axis pointing down a rotor north pole and a q-axis perpendicular to d, which creates torque.

Since each stator current produces a magnetic field in one of the three directions abc, we project the current vector on d, q (based on each current's influence on flux in these directions). The inductances L_d and L_q represent how easy it is to add flux in either direction. Since i_d has no effect on torque, we usually just set it to 0 (unless dq inductances are not equal, in which case it has influence).

If DC voltage is too high, we decrease i_d to reduce back EMF in q and thus require a lower voltage to yield the same torque. This is called Field Weakening. To control our motor's speed or torque, we demand a certain torque value; either directly through torque mode or indirectly by modifying demanded torque using a PI loop until we get a desired speed.

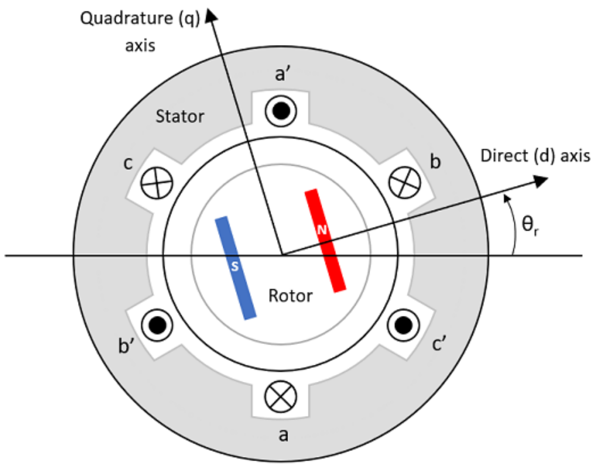

Figure 1: PMSM Diagram Showing dq Currents, Rotor and Stator

1.1 Park and Clarke Transforms

The Clarke transform converts the abc basis into α, β where α is aligned with one phase and β is perpendicular to it. The Park transform then goes from α, β to dq, and it requires the electrical position θ_e to know how much of α and β aligns with the rotor flux (d direction). In other words, Park rotates α, β to align with dq.

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}, \quad \begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} \cos\theta_e & \sin\theta_e \\ -\sin\theta_e & \cos\theta_e \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix}$$

References: Clarke Transform, Park Transform

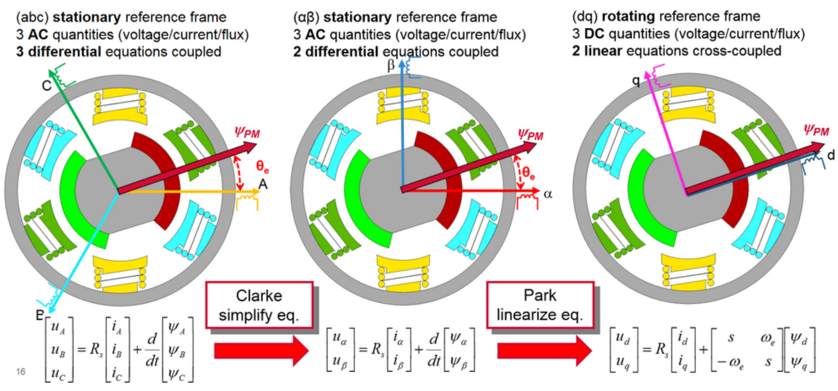
(abc) **stationary** reference frame
3 **AC** quantities (voltage/current/flux)
3 differential equations coupled
(αβ) **stationary** reference frame
3 **AC** quantities (voltage/current/flux)
2 differential equations coupled
(dq) **rotating** reference frame
3 **DC** quantities (voltage/current/flux)
2 linear equations cross-coupled
C
A
B
ψ_{PM}
θ_e
β
α
q
d
Clarke simplify eq.
Park linearize eq.
$\begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix} = R_s \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix}$
$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = R_s \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_\alpha \\ \psi_\beta \end{bmatrix}$
$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} s & \omega_e \\ -\omega_e & s \end{bmatrix} \begin{bmatrix} \psi_d \\ \psi_q \end{bmatrix}$
16

Figure 2: Shows abc currents being converted into the dq frame using Clarke and then Park

1.2 PI Controller

PI controllers are used a lot throughout, so here is a quick summary with an example. Say we have a current source connected to some unknown resistance, and e_{dc} is the deviation between desired and actual resistor voltage. We compute a correction i_{corr} to the initial current, $i_{\text{fixed}} = i_{\text{initial}} + i_{\text{corr}}$, to achieve our desired voltage:

$$i_{\text{corr}} = K_p\, e_{dc} + K_i \int e_{dc}\, dt.$$

The proportional term is the instantaneous response; K_p is how sensitively we modify i based on e_{dc}. The integral term sums past error. $K_i = \dfrac{K_p}{T_n}$; a larger T_n slows integral buildup, leading to slower response.

1.2.1 PI Tuning

Higher K_p gives faster instant response to a change but can cause overshoot / oscillation. A good strategy is to look at our governing equation, like for instance, what is $\frac{di}{dt}$ multiplied by, and then set K_p to be a fraction of this constant. Smaller T_n fixes steady state error faster can cause overshoot and windup. We should at least set it to be larger than the time constant of our system (time constant, described in 1.2.4, is how fast your system settles in a new state after disturbance). For nested loops, we also want to make each outer loop 2-10x slower than it's next inner loop. For instance, if one PI controls speed command, and another PI, based on that PI's speed command, controls current, if the current loop is 2× slower than speed PIs, the current PIs will double apply the correction to speed error since they will not sense their first correction applied. To tune PI gains, we usually start by only tuning proportional gain, and then tuning the integral gain.

1.2.2 Anti-Windup

What happens if we hit maximum current yet the PI keeps commanding more current? The integral term will just grow. Now, say we want to decrease current afterwards. The accumulated integral will make this slow. We use anti-windup to prevent this. When $i = i_{\max}$, we use:

$$i_{\text{corr}} = K_p\, e_{dc} + K_i \int \left[e_{dc} + k_{aw}(i_{\max} - i)\right] dt.$$

The anti-windup gain k_{aw} is typically set to $k_{aw} = \frac{1}{K_i}$, since when e is small, we get the ODE $\dot{i} = k_{aw}(i_{\max} - i)$ which has time constant $\frac{1}{k_{aw}}$. The integrator, which does not depend on i, does not have a time constant. We simply use $\frac{1}{k_{aw}}$ to get a response on a similar scale and speed to the integral wind up.

1.2.3 Current and Voltage Limits

Enforcing limits on voltage, current and the rate of change of these values is simple. We simply clamp these values to the limit. For variable S we would choose $S = \text{Sat}(S, S_{max}, S_{min})$ and $\dot{S} = \text{Sat}(\dot{S}, \dot{S}_{max}, \dot{S}_{min})$. We formally call rate change **slew**.

1.2.4 Time Constants

$G(s)$, where k is gain and s is a point on the complex plane, is a transfer function by which we multiply our input signal in the complex domain to get our output signal. It has **time constant** τ (how fast output reacts to input).

$$G(s) = \frac{k}{\tau s + 1}$$

Using the theory surrounding it, we can derive the PI bandwidth ω_c implicitly. Read More Here. Using Bandwidth is very overkill however for our purposes, and its better just to manually tune the PI gains. Bandwidth is the maximum frequency of change that the PI loop can effectively respond to (aka Track).

1.3 Inverter Switching

How do we go from DC to AC? Each coil switches between $V_{\text{dc}}/2$ and $-V_{\text{dc}}/2$. There are 8 combinations and 6 non-zero states (if all coils have same voltage, the vector sum is 0). To induce sinusoidal phase voltages 120° apart, we use space vector modulation (PWM). Duty D is the amount of the square wave period during which $V_{\text{dc}}/2$ is on. The average voltage over a period is:

$$v_{\text{avg}} = (2D - 1)\,\frac{V_{\text{dc}}}{2}.$$

The PWM frequency is how quickly we can change D. Due to the very short duration each duty cycle compared to the time constants of the rest of the system, the rest of the system sees only the average voltage. Usually, we don't model the switching behavior and instead just apply sine waves with max amplitude $V_{\text{dc}}/\sqrt{3}$. Where does $V_{\text{dc}}/\sqrt{3}$ come from? If we transform abc to α, β and plot the six possible vector combinations, we get a hexagon and the magnitude of each vector is $\frac{1}{3}V_{dc}$ (a 2/3 scaling factor is applied to $|V_{\text{dc}}/2|$ since the 3 abc vectors overlap equally in all directions, leading to double counting that must be fixed when transforming to $\alpha\beta$). The radius of the largest circle within the hexagon is $V_{\text{dc}}/\sqrt{3}$ (the circle in which we will rotate $\alpha\beta$ to induce sine waves). The $\frac{1}{\sqrt{3}}$ scaling factor also applies to the dq frame since $\alpha\beta$ to dq has no vector overlap and thus no correction factor needed.

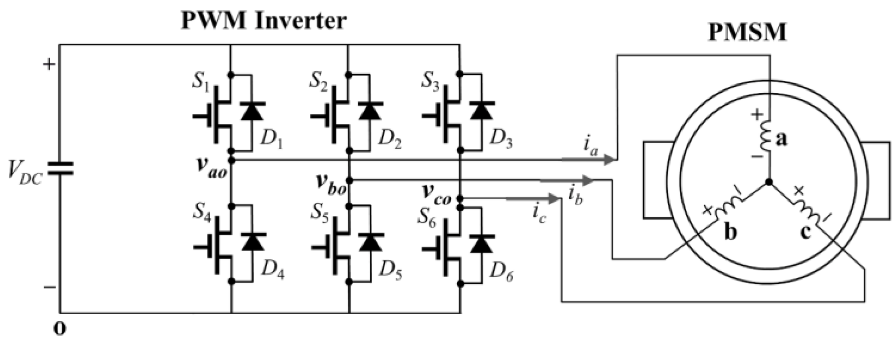

Figure 3: MOSFET Switching Converts DC Into 3 Phase AC

1.4 DC Link

The DC link pre-charges the capacitors between the source and the inverter before start, and discharges them when inverter is off.

$$P = \frac{3}{2}(v_d i_d + v_q i_q)$$

$$C\,\frac{dV_{DC}}{dt} = \frac{V_{\text{DC,battery}} - V_{DC}}{R} - \frac{P}{V_{dc}}, \qquad \text{(DC)}$$

This is an ODE. Our initial condition is $V_{DC}(0) = V_{DC,\text{battery}}$ since we precharge the capacitor. The power P of the motor is scaled by $\frac{3}{2}$ since we are converting back from dq to abc. Remember, dq is just a projection, and when we adjust dq, we must rescale to abc before applying. DC link capacitor

2 Regenerative Braking

When the rotor spins, a back-EMF is created in the stator windings due to changing flux linkage (Lenz's law). The i_q created by the car rolling forward is negative; we must inject positive i_q to get zero torque. As negative i_q becomes more negative, the negative torque T_e also becomes more negative and more mechanical power is needed to overcome it. Electrically, as the magnitude of negative i_q goes higher, applying the same back EMF (by rotating the rotor) requires more work (since $P = IV$). PMSM regen braking, Regen

3 PMSM Equations

ψ is flux linkage (flux times turns), ψ_f is the rotor flux, and stator voltage is $v = Ri + \frac{d\psi}{dt}$. Projecting onto dq (Park transform), we get:

$$\dot{\psi}_d = v_d - R_s\, i_d + \omega_e L_q i_q, \qquad (1)$$
$$\dot{\psi}_q = v_q - R_s\, i_q - \omega_e (L_d i_d + \psi_f). \qquad (2)$$

where, using current = flux/inductance:

$$i_d = \frac{\psi_d - \psi_f}{L_d}, \qquad i_q = \frac{\psi_q}{L_q}, \qquad \omega_e = p\,\omega_m, \qquad \theta_e = p\,\theta_m. \qquad (3)$$

The electrical angle θ_e and speed ω_e are scaled by the number of pole pairs p because each mechanical revolution produces p complete electrical cycles. For a motor with $p = 3$ pole pairs, one mechanical revolution (360°) produces three complete electrical cycles. The two fluxes are coupled due to back EMF.

3.1 Torque

Torque (again, the 3/2 comes from the Clarke transform):

$$T_e = \tfrac{3}{2}\,p\big(\psi_f i_q + (L_d - L_q)\,i_d i_q\big), \tag{4}$$

Mechanical dynamics:

$$J\dot{\omega}_m = T_e - T_L - B\omega_m, \qquad \dot{\theta}_m = \omega_m. \tag{5}$$

The torque equation comes from energy balance. The extra flux added by i_d does not influence torque since it does not increase the angle between stator and rotor flux; it creates a flux parallel to the ψ_f which does not pull on it at all. If the L's are not equal (L_d smaller for instance), then it's easier for the stator to point in a direction with more i_d and it does so; our dq frame is technically wrong now. The true "d" basis becomes a vector tilted closer to q in our old dq frame. Now, pushing in what used to be "d" actually pushes a bit into "q" also. Torque Dynamics

4 Field Weakening

When we demand more V_{DC} than is allowed, we can reduce the back EMF in q so that less voltage is needed to maintain the same i_q. However, this isn't free energy since to reduce q's back EMF, we need more current and must make i_d more negative (see equation 2). When $L_d = L_q$, decreasing i_d does not change torque. In the unequal case, the resulting slight error in torque will be fixed by the speed PI controller or torque command. Although the torque command is not a PI loop (rather, a relation between torque and i_q), it will still fix this error since the change in torque will change i_q.

We set our V_{DC} limit to $\varsigma\frac{1}{\sqrt{3}}$ times the true maximum where ς is fraction of the maximum DC voltage that we can use. We also cap the magnitude and slew (rate of change) of i_d. Here is the field weakening PI controller:

$$i_{\text{d, final}} = i_{\text{d, initial}} - i_{\text{d, corr}}$$

$$i_{\text{d, corr}} = K_p(V - V_{\max}) + K_i\int (V - V_{\max})\,dt$$

Field Weakening

5 Current Controller

This PI modifies v_{dq} so that i_d and i_q equal the desired $i_d^\star$ and $i_q^\star$ set by the field-weakening and the outer torque/speed/generator loops (section 6) respectively. $v_{dq} = v_{dq,\text{old}} + v_{\text{corr}}$:

$$\begin{aligned} v_{d,\text{corr}} &= K_{p,d}\,(i_d^\star - i_d) + K_{i,d}\int (i_d^\star - i_d)\,dt, \\ v_{q,\text{corr}} &= K_{p,q}\,(i_q^\star - i_q) + K_{i,q}\int (i_q^\star - i_q)\,dt, \end{aligned} \tag{8}$$

After each computation of these correction voltages, we can "fix" them to account for coupling (back EMF) between d and q. The correction is multiplied by a strength constant between 0 and 1. We should apply the coupling correction only if we trust in our estimates of L_d, L_q. The correction will simply make the optimizer's job easier.

$$\begin{aligned} v_d &= v_{d,\text{corr}} - \omega_e L_q i_q, \\ v_q &= v_{q,\text{corr}} + \omega_e (L_d i_d + \psi_f), \end{aligned} \tag{11}$$

Vector Control, Current Controller

6 Control Modes

You choose an "inner" control mode (section 6.2); you either obtain your mechanical speed and electrical position using encoder/resolver or an observer (sensorless). You then choose an outer mode (section 6.1) to define what you control.

6.1 Full DCU Workflow

1. Measure 3-phase currents and Clarke Transform into i_α, i_β
2. Get θ_e and ω_e from either Resolver or Observer (sensorless).
3. Use θ_e to apply Park Transform and get i_d, i_q
4. Outer loop (Speed/Torque/Generator mode): Compute desired $i_q^\star$
5. Field Weakening: Compute desired $i_d^\star$
6. Apply current limits and current rate of change (slew) limits
7. Use current PI controller to compute desired v_d, v_q
8. Apply voltage limits and voltage rate of change (slew) limits
9. Inverse Park Transform and then Inverse Clarke Transform to get v_a, v_b, v_c
10. Generate duty cycles to apply each voltage

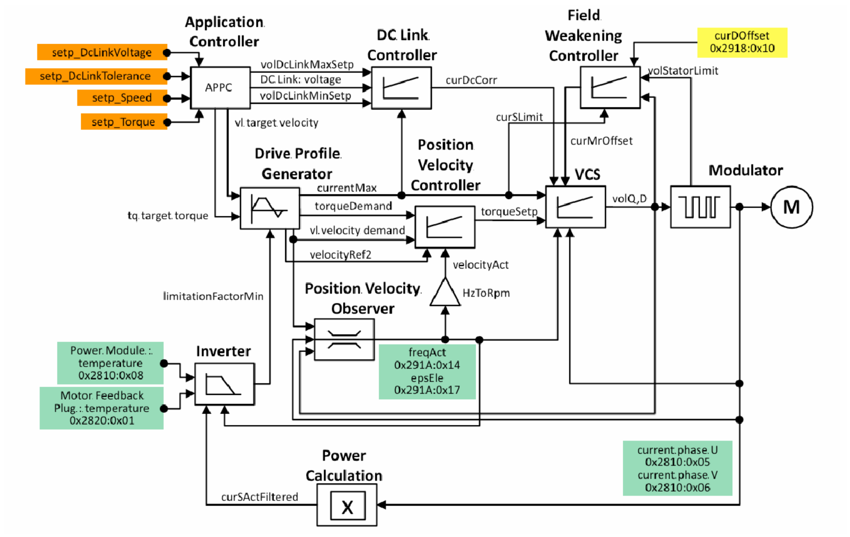

Figure 4: Full DCU Workflow Logic

6.2 Outer Control Modes (CiA-402)

These are the controllers that get us our target $i_q^\star$. We want these outer loops to remain slower than the current loop so we don't overshoot/undershoot current (see 1.2.1 to understand why nested loops must become progressively faster).

6.2.1 Velocity Mode ($m = 2$)

DCU uses a PI controller for this. We keep notation brief, since we know how PI controllers work now:

$$e_\omega = \omega_{\text{Cmd}} - \omega_m, \qquad i_q^\star = k_{p\omega} e_\omega + \xi_\omega, \qquad \dot{\xi}_\omega = k_{i\omega} e_\omega \tag{13}$$

Generates desired $i_q^\star$ (torque) from the speed error e_ω. ω_{Cmd} is desired speed, ω_m is measured mechanical speed. ξ_ω is the integrator value. $i_d^\star$ is provided by the field weakening PI controller. The raw speed command is passed through a rate limiter which clamps it at a maximum acceleration or maximum negative acceleration.

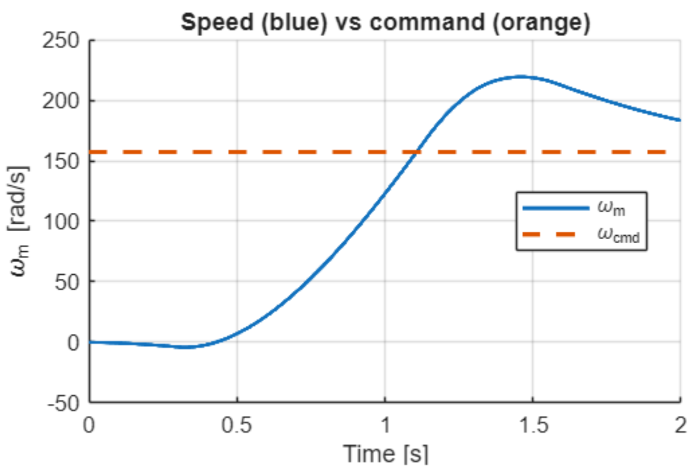

Figure 5: Example Velocity Response with Poorly Set Velocity PI Gains

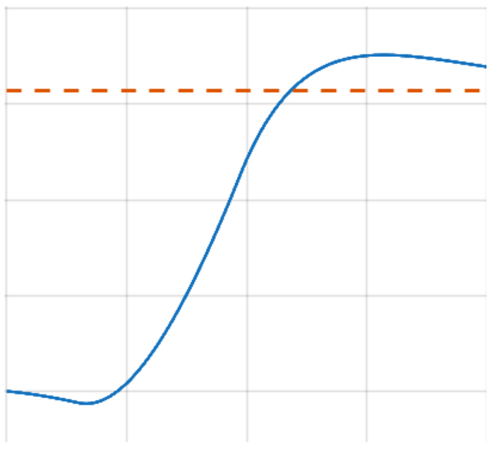

Figure 6: Example Velocity Response with Better Velocity PI Gains. Compared to Figure 5, we decreased speed loop gains to prevent overshoot.

6.2.2 Torque Mode ($m = 4$)

This does not use a PI controller. T_{Cmd} is desired torque. We assume $L_q = L_d$ and, and by equation 4:

$$i_q^\star = \frac{2}{3p\,\psi_f} T_{\text{Cmd}},$$

Torque slew (rate of change) and magnitude is clamped. Read mode: Torque estimate

6.2.3 Generator Mode ($m = -5$)

Field weakening (section 4) prevents over voltage by reducing d-flux. Generator mode controls torque instead and also addresses under voltage. If $V_{dc} > V_{\max}$, we increase i_q to reduce power generation. If $V_{dc} < V_{\max}$, we decrease the magnitude of i_q (torque) to prevent undervoltage. Here is the PI Controller:

$$T_{\text{corr}} = K_p e + K_i \int e\,dt$$

The deadband Δ is a range of voltage outside the limits where the controller doesn't react. This prevents the PI from constantly adjusting for small errors. We only turn on the PI when $\Delta > \max(V_{dc} - V_{max}, V_{min} - V_{dc})$.

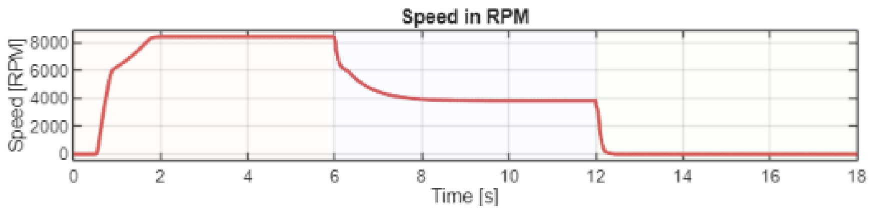

Figure 7: Torque mode, then speed mode, then Regen. The third Section (green) is when regenerative braking was activated

6.2.4 Cyclic Synchronous Position Mode ($m = 8$)

Converts position error into a desired speed at each time step. Basically moves our motor to desired position. For example, it would be used for the motor in a CNC machine. Not useful for our purposes.

6.3 Inner Control Modes

We must somehow measure/estimate ω_e which is used in the speed control loop, and most importantly, θ_e, which lets us apply Park Transform to obtain i_{qd}.

6.3.1 Sensorless Vector Control ($m = 5$)

θ_e and ω_e are estimated by an observer. We operate in the stationary frame (α, β) instead of dq since converting to dq requires knowing θ_e.

$$\begin{aligned} v_\alpha &= R_s i_\alpha + L_s \frac{di_\alpha}{dt} + e_\alpha, \\ v_\beta &= R_s i_\beta + L_s \frac{di_\beta}{dt} + e_\beta, \end{aligned} \tag{18}$$

The $L_s = \frac{L_d + L_q}{2}$ approximation is fine as the difference between dq inductance does not influence back EMF e, which depends only on ψ_f, ω_e and θ_e.

$$e_\alpha = -\psi_f \omega_e \sin\theta_e, \qquad e_\beta = \psi_f \omega_e \cos\theta_e, \tag{19}$$

The algorithm below is how we estimate the back EMFs. Hat means estimate.

$$\begin{aligned} \frac{d\hat{e}_\alpha}{dt} &= k\big[(v_\alpha - R_s i_\alpha - L_s \tfrac{di_\alpha}{dt}) - \hat{e}_\alpha\big], \\ \frac{d\hat{e}_\beta}{dt} &= k\big[(v_\beta - R_s i_\beta - L_s \tfrac{di_\beta}{dt}) - \hat{e}_\beta\big], \end{aligned} \tag{20}$$

These equations come from equation (18). Ideally, the part inside [ ] should equal zero. The deviation from 0 within the [ ] is proportional to $\frac{d\hat{e}}{dt}$ (by proportionality constant k). We increment $\hat{e}$ by this derivative until the derivative is 0 (and equation 18 is satisfied). From here, plugging our estimate for back EMF into (19) we obtain ω_e and θ_e.

Observer Gain k determines how quickly the observer converges to the true back-EMF value. High k is more sensitive to measurement noise and parameter uncertainties. However, low k can lag behind quick speed changes and cause oscillation. A reasonable starting point for the gain, just like for the other PIs, is the reciprocal of the time constant. Back EMF observer

6.3.2 Resolver ($m = 6$)

Electrical angle θ_e and speed ω_e are measured directly from the encoder or resolver. We will skip encoder and just discuss **resolver**. The resolver rotor is connected to our motor's rotor rotates with our unknown θ_e. The rotor flux is time varying, created by:

$$V_{\text{coil}}(t) = V_0 \sin(\omega_c t),$$

where ω_c is the carrier frequency, typically in the range of 2-10 kHz. Common values are 5 kHz or 10 kHz - high enough to avoid interference with motor currents but low enough for reliable signal processing.

The coils in the stator are 90 degrees apart, meaning that straight lines through each coil's area would intersect at a 90 degree angle. Call the flux of the coil at 0 degrees SIN and the 90 degree flux COS. The mutual inductance between the rotor and the SIN coils is $M \sin\theta_e$; for COS, the mutual inductance is $M \cos\theta_e$, where M is the transformer ratio between the AC in the rotor and the stator coils.

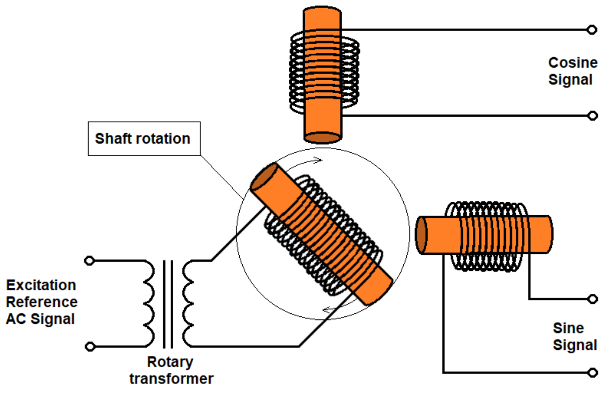

Figure 8: Resolver Circuit Diagram

For $V = M(\theta_e)\frac{di_{\text{rotor}}}{dt}$, substitute in the θ_e based mutual inductances to get the EMF in the SIN and COS directions.

$$V_{\sin}(t) = MV_0 \sin(\omega_c t) \sin(\theta_m + \theta_{\text{off}}),$$
$$V_{\cos}(t) = MV_0 \sin(\omega_c t) \cos(\theta_m + \theta_{\text{off}}).$$

Due to manufacturing issues (ex: error in SIN vs COS angle, different $L_{\text{SIN}}, L_{\text{COS}}$), θ_e may not equal θ_m. We measure the error θ_{off} during calibration by slowly rotating the motor and comparing electrical position to mechanical position. Our pole pairs ratio (poles on motor / 2) converts from mechanical to electrical angle. This ratio can change depending on the gear ratio by which we connect the motor with the resolver.

Resolver Low Pass Filter: Some resolvers have low pass filters on θ with time constant less than current loop but greater than the noise in θ. Keeps estimates from being influenced by noise. $\theta_t = \theta_{t-1} + \alpha(\theta_t - \theta_{t-1})$.
References: Resolver datasheet, Resolver

7 Mode Switching

When changing control modes, we perform a **bumpless transfer**: we back-calculate and carry integrator states so the new controller's output matches the current operating point. Also, accel/decel and torque/current slopes ensure continuity as modes switch.

Why mode switches cause spikes Imagine we switch from torque mode to speed mode. The math that generates $i_q^\star$ changes suddenly and we get a spike in $e_q = i_q^\star - i_q$. Likewise, the old integrator value may result in different behavior than what we want it to be as the integrator constant changes as we switch modes. To mitigate this latter issue, we modify the old integrator value such that the new PI controller will yield the same dynamics from a timestep ago. For instance, entering speed mode, we would set:

$$\xi(t_0^+) \;=\; \frac{i_q^\star(t_0^-) \;-\; K_{p,\omega}\big(\omega_{\text{cmd}} - \omega_m\big)}{\max(K_{i,\omega}, \varepsilon)},$$

This is literally re-arranging the new PI controller equation and solving for ξ such that we yield the i_q command at t_0^-. If the new integrator uses an anti-windup PI, we also apply the anti windup correction $k_{aw}(v_{max} - v)$ to $\xi(t_0^+)$ before plugging it in. See section 1.2.2 for more details about Anti Windup.

8 Simulink Simulation Parameters and Quick Summary

Table 1 shows a list of parameters our simulation uses and which part of the DCU they relate to. We ignore thermal de-rating as our car operates significantly below the temperature limit above which I_{max} begins to decrease (120 C^o). The simulation can be found using this link: MATLAB Code. We implement the DCU as a *Matlab Function Block*. All parameters are doubles. The load torque, torque command and speed command are double time series (2xN matrix) and hv_ok is a boolean double. **Units are: seconds, amps, volts, ohms, henries, farads, rad/s, Nm and kg m^2**. Please copy the DCU Implementation (excluding the demonstration runner at the top) within the Bucher_DCU_MLX_Demo.mlx file into a Matlab Function block in your Simulink Simulation. You can use the demonstration runner within Bucher_DCU_MLX_Demo.mlx to play around with the DCU and potentially debug it if you choose to modify the script.

Parameter	**Definition**
Simulation	
Ts	sample time
Tfinal	sim duration
PMSM Electrical	
p	pole pairs
Rs	stator resistance
Ld	d-axis inductance
Lq	q-axis inductance
psi_f	PM flux linkage
Current, Torque, Speed Limits	
Imax	current limit
diq_slew	i_q slew limit
did_slew	i_d slew limit
Tmax_reg	regen torque limit
T_rated	rated torque
T_fac	torque factor
w_max	speed limit
acc_max	accel limit
dec_max	decel limit
DC Link	
Vdc_nom	nominal DC-link
Vdc_max	max DC-link
Vdc_min	min DC-link
Cdc	DC-link capacitance
Rsrc	source resistance
dv_max	voltage slew limit
Current, Speed PI Gains	
Kp_d	i_d PI K_p
Ki_d	i_d PI K_i
Kp_q	i_q PI K_p
Ki_q	i_q PI K_i
decouple_k	dq decoupling gain
Kp_w	speed PI K_p
Ki_w	speed PI K_i
Field Weakening	
FW_Kp	FW PI K_p
FW_Ti	FW PI T_i
vfac	voltage fraction
id_fac	$\lvert i_d \rvert / I_{\max}$ limit
FW_on	FW on threshold
FW_off	FW off threshold
Regen Controller	
omega_regen_min	regen stops below this speed
Vp_vdc	regen PI K_p
Tn_vdc	regen PI T_i
Mechanical	
J	rotor inertia
B	viscous friction
T_coulomb	Coulomb friction
Resolver / Inner Mode	
mode_inner	inner angle mode
pos_offset	angle offset
pole_pairs_ratio	pole pair ratio
alpha_res	resolver low pass filter gain
hv_ok	HV enable (safety switch)

Table 1: Simulink Parameters