ELECTRIC MOTOR ROTOR ANGLE

Date: 23rd Mar 2022

Revision: 1.2

Author: Tony Little

Product: Electric Motor

Company: Electric Vehicle System Technology

1 CONTENTS

9-3-2022	V1.1	Added section on Rotor Angle Start/Stop. Added section on Rotor Angle Selection.
22-3-2022	V1.2	Added section 4, Hall Signal Tracking and Thresholds. Octave source code updated.

2 INTRODUCTION

Given the analog hall sensors currently have significant distortion, which varies with rotor speed, it is considered problematic to attempt an analog quadrature rotor angle calculation. When the motor design can accommodate minimal distortion, it would be advantageous to re-consider.

To achieve a function rotor angle for commutation, the approach will be to digitize the analog sensor feedback near the zero-reference point with hysteresis, to provide rotor angle position updates. These form an angle update correction. The rate of updates will then be used to determine the rotation speed to enable faster periodic calculated rotor position. For commutation control, the rotor angle needs to be updated at a faster rate and with significantly more electrical rotation points. Otherwise, the commutation will be severely limited and sector rotation to use all phases will not be possible.

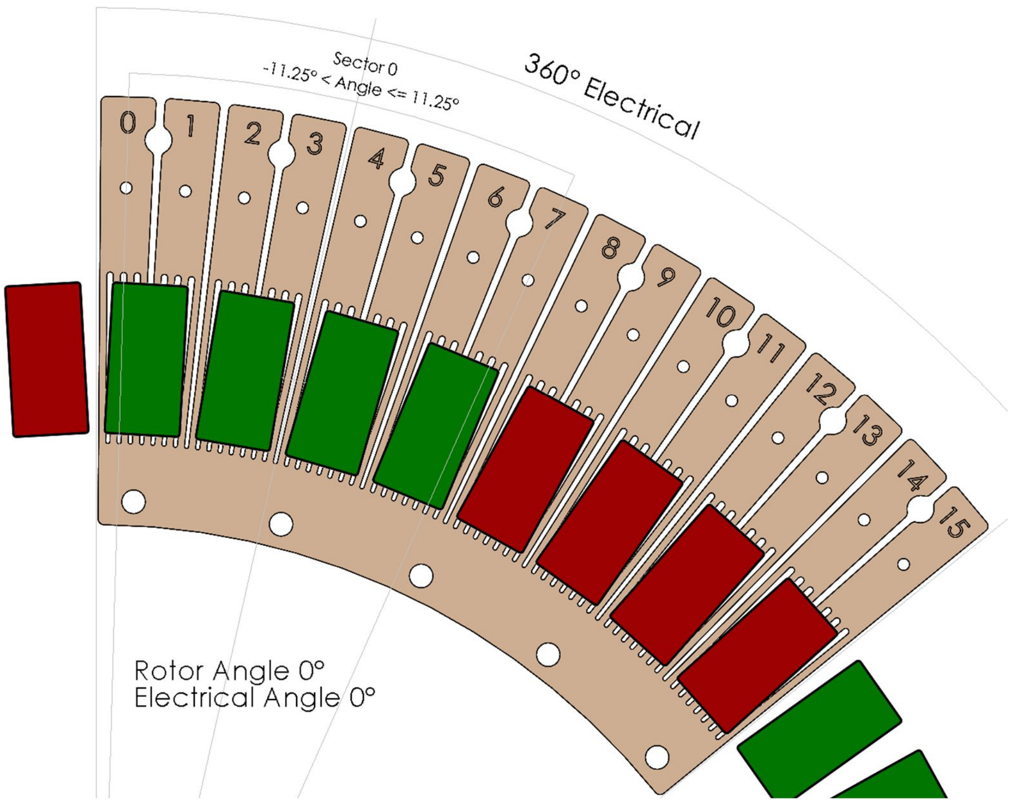
An extension to this is to calculate the acceleration to improve the dynamic performance response.

3 HALL SENSOR QUADRATURE UPDATES

The first stage is to quantize the analog hall sensor data to produce quadrature state changes. These state changes provide the feedback to update the rotor angle with a measured known position at the time of the change. The rate of these state changes will later be used calculate updates at a higher rate to support sector rotation.

3.1 ROTOR ELECTRICAL ANGLE

The previously defined 0 degree electrical angle is shown in the diagram below. The hall sensors are positioned 90 electrical degrees apart, to the midline for bars 7 (Hall Sensor A) and 11 (Hall Sensor B). The bars are 360/16 = 22.5 degrees apart. Given the hall sensors are positioned midline to the bars, they have an offset of ½ a bar, 11.25 degrees. The peak field intensity at 0 degrees occurs between bars 7 to 8 and 0 to 15 on adjacent segments. Therefore, Hall Sensor A will have an 11.25 degree offset to the peak field with the rotor at 0 electrical degrees.



3.2 ROTOR TO HALL SENSOR RELATIONSHIP

The mathematical relationship between electrical angle θ and magnetic field sensed by the hall sensors V_{hall_a}, V_{hall_b} is given below.

$$V_{hall\_a} = \frac{V_{pp}}{2} \cos \left(\frac{2\pi}{360}\left(\theta - \theta_{offset}\right) + V_z\right.$$

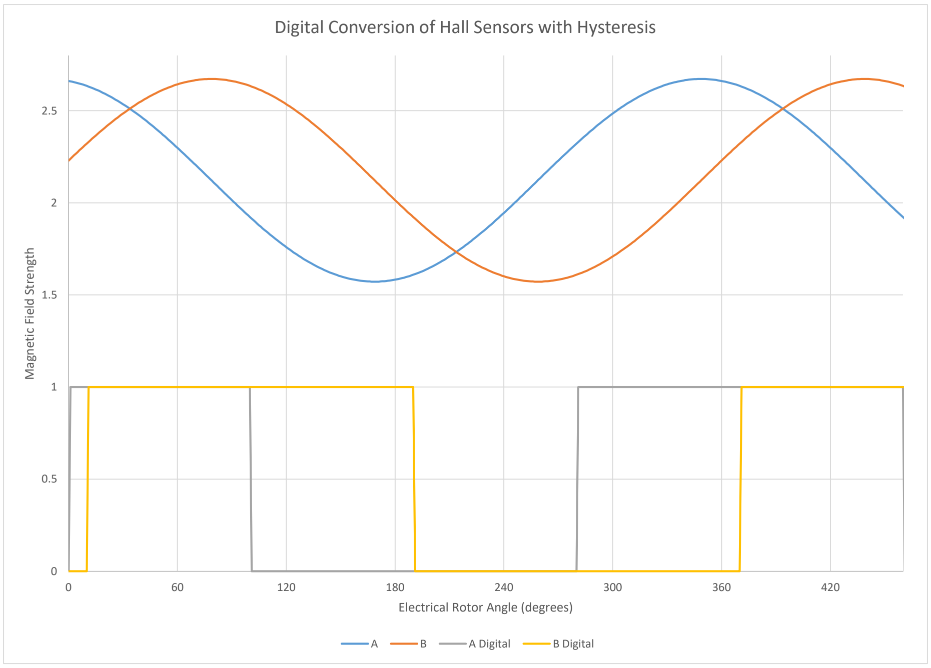
$$V_{hall\_b} = \frac{V_{pp}}{2} \sin \left(\frac{2\pi}{360}\left(\theta - \theta_{offset}\right) + V_z\right.$$

Transposed to solve for θ.

$$\theta = \frac{360}{2\pi} cos^{-1}\left[\frac{2(V_{hall\_a} - V_z)}{V_{pp}}\right] - \theta_{offset}$$

$$\theta = \frac{360}{2\pi} \sin^{-1}\left[\frac{2(V_{hall\_b} - V_z)}{V_{pp}}\right] - \theta_{offset}$$

The graph below shows the analog hall sensors V_{hall_a}, V_{hall_b} with their digitized logic transitions with hysteresis based on the parameters previously defined.



3.3 QUADRATURE TRANSITION STATE TABLE

The transitions are defined in the follow logic table, along with resulting angle updates.

Quadrature Transistion State Table					
A(t-1)	**B(t-1)**	**A**	**B**	**Angle**	**Direction**
0	0	0	0	-	-
0	0	0	1	190.07	Reverse
0	0	1	0	280.07	Forward
0	0	1	1	-	-
0	1	0	0	190.07	Forward
0	1	0	1	-	-
0	1	1	0	-	-
0	1	1	1	100.07	Reverse
1	0	0	0	280.07	Reverse
1	0	0	1	-	-
1	0	1	0	-	-
1	0	1	1	10.07	Forward
1	1	0	0	-	-
1	1	0	1	100.07	Forward
1	1	1	0	10.07	Reverse
1	1	1	1	-	-

Double state changes are considered invalid.

3.4 TRANSITION STATE TABLE FILTERING

A good filter to consider is one that does not allow the state to reverse direction without a minimum period being exceeded. This increases the robustness to noise. This is particularly important for calculating the rotor velocity and acceleration, as errant state changes will have a significant impact on their calculation. An alternative simple filter is to lockout the quadrature quantization for a minimum period after a state change. Given the maximum electrical RPM is 49,000, the maximum quadrature update rate is 60/49k/4 = 306us. A practical generic lockout period of 100us would be realistic.

4 HALL SIGNAL TRACKING AND THRESHOLDS

The following table represents preliminary example for operating parameters and voltage levels previously established from the analysis of the hall sensor performance. In addition, an arbitrary +/-0.2V hysteresis has been specified. Given the current magnetic field distortion, a low hysteresis is desirable. Too low and analog noise may cause a false transition. The level chosen is considered a balance, but also with enough significance to validate the math and identify correct operation.

Parameter	Unit	Value
Phase Offset	Deg	11.25
Hysteresis +/-	V	0.200
Vz Zero Ref	V	2.122
Low	V	1.922
High	V	2.322
Vp-p	V	1.100

The quadrature rotor signals amplitude and DC offset will vary between builds. An offset of 0.1V on one or both of the quadrature signals is sufficient to skew subsequent rotor angle estimations. Therefore, independent tracking of each hall sensor is required to determine the amplitude and subsequent hysteresis thresholds dynamically.

4.1 HALL SIGNAL TRACKING ALGORITHM

To track the AC signal from the hall sensor and subsequent derive the logic switching around the zero reference, a tracking algorithm is required. The aim of the algorithm is to track the minimum and maximum of the signal. This data can then be used to determine the zero reference for each hall signal, which the hysteresis threshold levels can then be dynamically determined.

Both minimum and maximum filter need at least a minimum amount of filtering to ensure any outliers do not adversely impact the desired envelop. A basic 4x IIR filter is used.

These filters must also have an opposing convergence mechanism, otherwise the minimum and maximum values could not recover from the extents of perturbations or initial conditions. This convergence additionally requires fast initial tracking. However, during operation, much slower convergence is required for stable threshold transitions and smooth rotation output data. Therefore, a dual rate convergence mechanism is used.

The follow algorithm equations are for Hall Sensor A. Implementation requires a second set of equations for tracking Hall Sensor B.

4.1.1 OPERATING CONSTANTS

$Hall_{\min\_range} = 0.5V$	*Minimum range the minimum and maximum can converge too.*
$Hall_{\mathrm{over}\_range} = 1.25V$	*When hall tracking range (max – min).*
$Hall_{mid\_init} = 2.0V$	*Mid-voltage of hall sensor oscillation.*
$Hall_{Hysteresis} = 0.2V$	*Hysteresis between analog hall quantization.*
$Hall_{Convergence\_period\_fast} = 1.0ms$	*Maximum period between slow convergence updates.*
$Hall_{Convergence\_period\_slow} = 40.0ms$	*Maximum period between slow convergence updates.*
$Hall_{Convergence\_rate} = 0.005V$	*Convergence step size*

4.1.2 INITIALISATION:

To occur after the first A/D readings are obtained.

$$Hall_{A\_mid} = Hall_{mid\_init}$$

$$Hall_{A\_max} = Hall_{A\_mid} + \frac{Hall_{Hysteresis}}{2}$$

$$Hall_{A\_min} = Hall_{A\_mid} - \frac{Hall_{Hysteresis}}{2}$$

$$Hall_{A\_Convergence\_Timer\_Max} = 0$$

$$Hall_{A\_Convergence\_Timer\_Min} = 0$$

4.1.3 ALOGORITHM LOOP

Convergence Timer Update

Reduce convergence timers (if not zero) with last execution loop period.

$$Hall_{A\_Convergence\_Timer\_Max} = \max(0, Hall_{A\_Convergence\_Timer\_Max} - Loop\ period)$$

$$Hall_{A\_Convergence\_Timer\_Min} = \max(0, Hall_{A\_Convergence\_Timer\_Min} - Loop\ period)$$

Maximum Signal Tracking

4xIIR Peak Value Filter with dual convergence rate.

```
$Hall_{A\_range} = \max(Hall_{A\_max} - Hall_{A\_min}, 0)$
If $(Hall_{A_{ADC}} \geq Hall_{A\_max})$
    $Hall_{A\_max} = \frac{(3*Hall_{A\_max} + Hall_{A_{ADC}})}{4}$        4xIIR Track maximum value
elseif $Hall_{A\_range} \geq Hall_{over\_range}$ && $Hall_{A\_Convergence\_Timer\_Max} == 0$
    $Hall_{A\_max} = Hall_{A\_max} - 0.01V$        *Fast Convergence*
    $Hall_{A\_Convergence\_Timer\_Max} = Hall_{Convergence\_period\_fast}$
elseif $Hall_{A\_range} \geq Hall_{\mathrm{min\_range}}$ && $Hall_{A\_Convergence\_Timer\_Max} == 0$
    $Hall_{A\_max} = Hall_{A\_max} - 0.01V$        *Slow Convergence*
    $Hall_{A\_Convergence\_Timer\_Max} = Hall_{Convergence\_period\_slow}$
endif
```

Minimum Signal Tracking

4xIIR Trough Value Filter with dual convergence rate.

```
$Hall_{A\_range} = \max(Hall_{A\_max} - Hall_{A\_min}, 0)$

If $\left(Hall_{A_{ADC}} \leq Hall_{A\_min}\right)$

    $Hall_{A\_min} = \frac{(3*Hall_{A\_min} + Hall_{A_{ADC}})}{4}$        4xIIR Track maximum value

elseif $Hall_{A\_range} \geq Hall_{over\_range}$ && $Hall_{A\_Convergence\_Timer\_Min} == 0$

    $Hall_{A\_min} = Hall_{A\_min} + 0.01V$        *Fast Convergence*

    $Hall_{A\_Convergence\_Timer\_Min} = Hall_{Convergence\_period\_fast}$

elseif $Hall_{A\_range} \geq Hall_{\min\_range}$ && $Hall_{A\_convergence\_timout} == 0$

    $Hall_{A\_min} = Hall_{A\_min} + 0.01V$        *Slow Convergence*

    $Hall_{A\_Convergence\_Timer\_Min} = Hall_{Convergence\_period\_slow}$

endif
```

Maintain minimum range.

This ensures a minimum distance for realistic and quicker convergence.

```
$Hall_{A\_range} = \max(Hall_{A\_max} - Hall_{A\_min}, 0)$

If $Hall_{A\_range} < Hall_{\min\_range}$

    $Hall_{A\_max} = Hall_{A\_min} + Hall_{\min\_range}$

endif
```

Hall Signal Thresholds

Calculated mid-point of the signal.

$$Hall_{A\_zero\_ref} = \frac{Hall_{A\_max} - Hall_{A\_min}}{2} + Hall_{A\_min}$$

Calculate current thresholds.

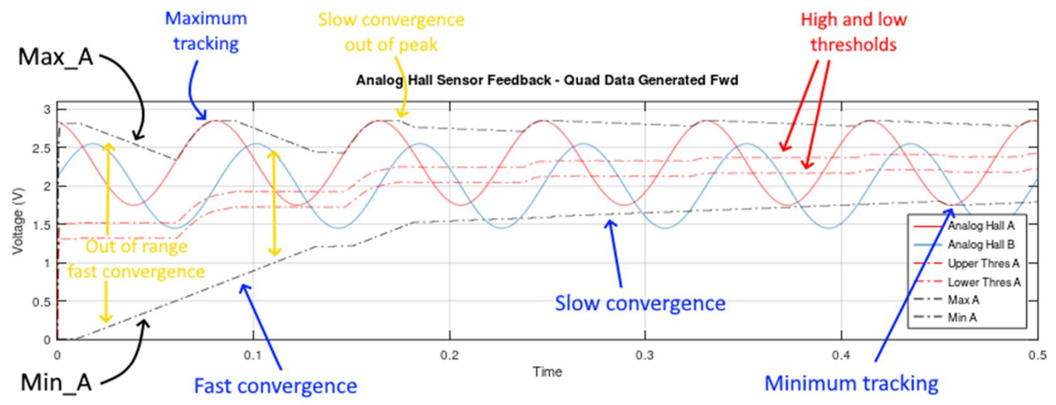
$$Hall_{A\_threshold\_high} = Hall_{A\_zero\_ref} + \frac{Hall_{Hysteresis}}{2}$$

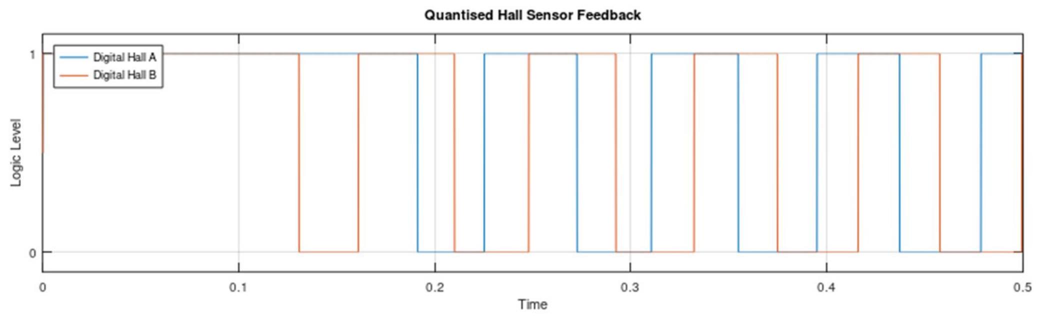
$$Hall_{A\_threshold\_low} = Hall_{A\_zero\_ref} - \frac{Hall_{Hysteresis}}{2}$$

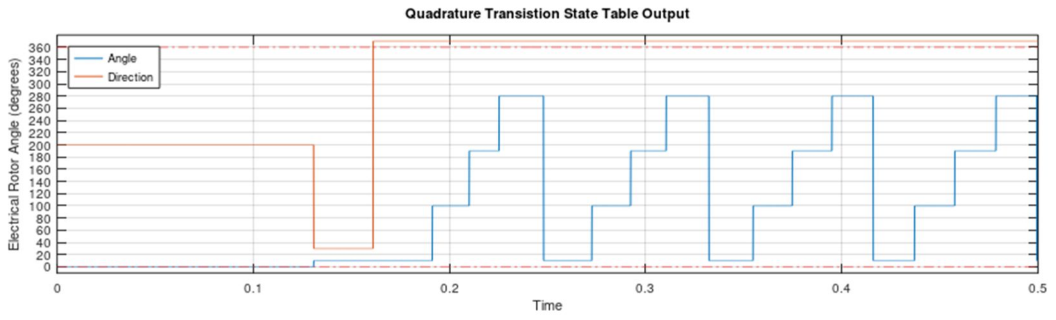
Note: The $Hall_{A\_zero\_ref}$ output can be used by the start/stop rotor angle calculation.

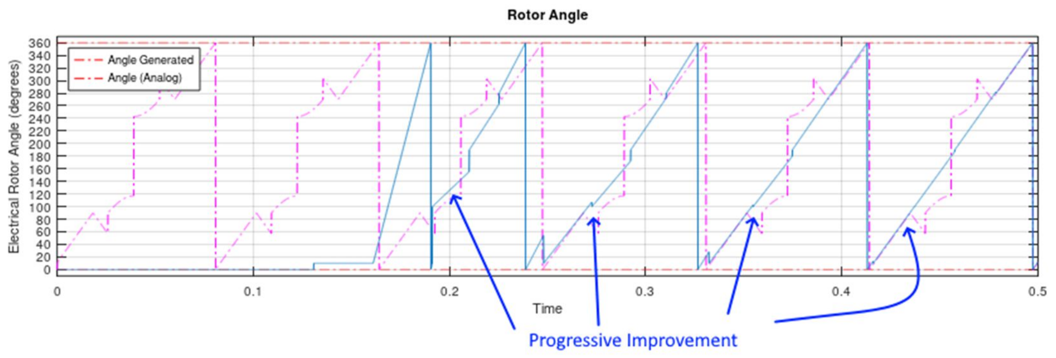
4.2 HALL SIGNAL TRACKING ALGORITHM SIMULATION

The algorithm was implemented in octave. Below shows the quadrature signals with one having an additional 0.3V offset. Convergence rates increased to 1ms/10ms/10mV for demonstration only. In addition, initial conditions were not set to force/test large range adaption. Annotation and dashed plots with respect to Hall A only.

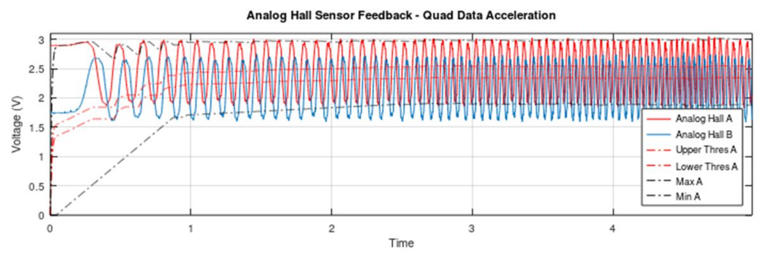


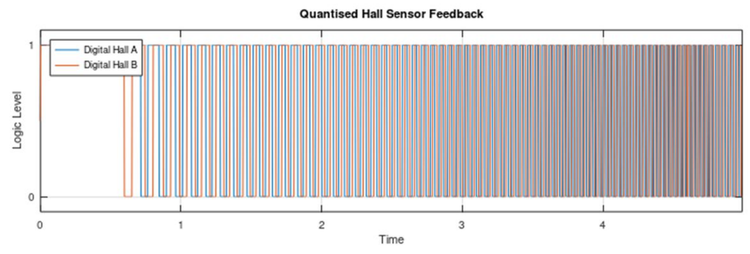


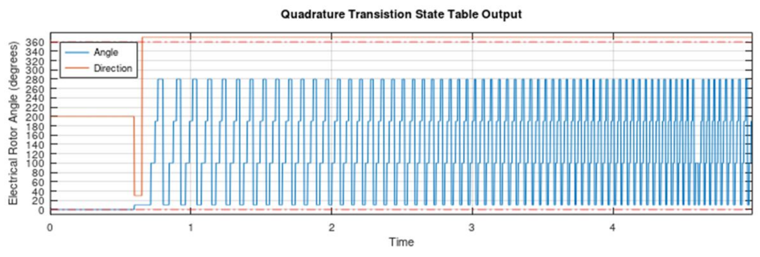


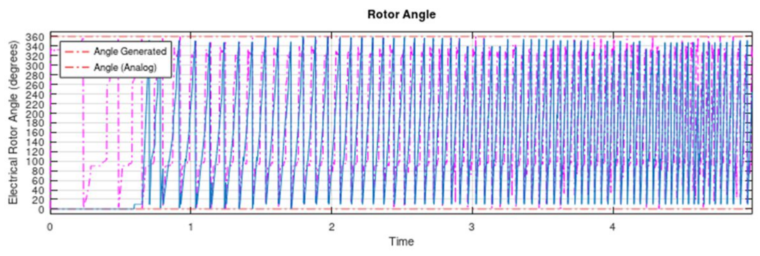


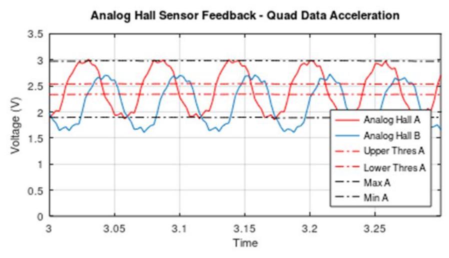
Example using captured date, fast acceleration.

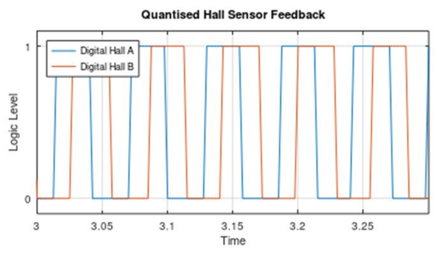


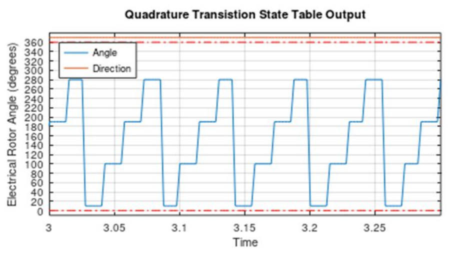


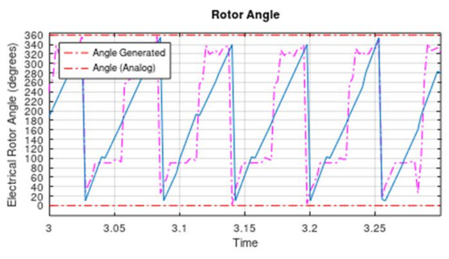












5 ROTOR ANGLE UPDATE

4 rotor angle updates per electrical rotation from the quadrature sensors is not sufficient for commutation. Sector rotation logic has 16 states and the resolution should at least be 10 times this requirement. To accommodate the faster rotor angle updates, the period between the quadrature updates can be measured and the rate of change can be used to periodic update a calculated position. More specifically, the rotor velocity and acceleration can be calculated to applying calculated rotor updates.

Hence, the quadrature state updates are responsible for synchronizing to the hall sensor feedback at 4 points in the electrical rotation. At other times the rotor angle is updated from calculation, based on the rate of quadrature updates.

The requirement is to measure the period between successive quadrature state updates, as well as the delta for rotation. Then calculate the velocity and acceleration and apply at a higher update rate.

Period (seconds)

dRotor_Angle_Period = Current Time - Rotor_Last_Correction_Time

Velocity (angle/second)

Rotor_Angle_Velocity = dRotor_Angle / dRotor_Angle_Period

Acceleration (angle/second/second)

Rotor_Angle_Acceleration = (Rotor_Angle_Velocity - Rotor_Angle_Velocity(t-1)) / dRotor_Angle_Period

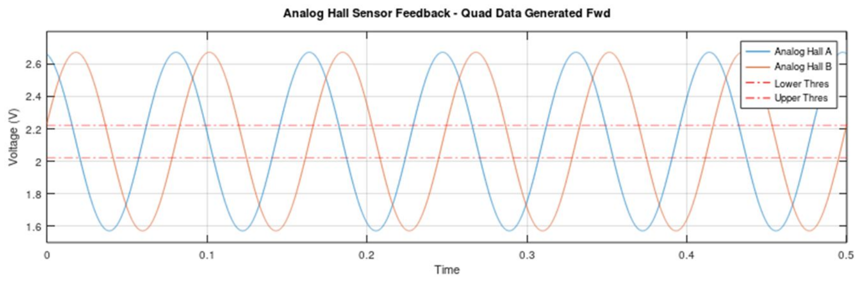
The rate of updates can be applied at the PWM period. These updates can then drive the sector rotation for bar current control.

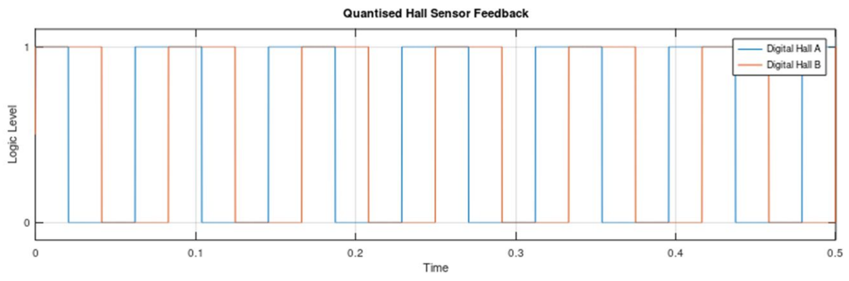
Rotor Update

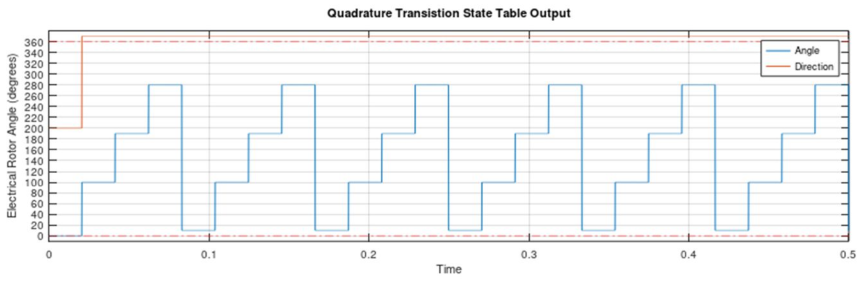
Rotor_Angle = Rotor_Angle + Rotor_Angle_Velocity * PWM_Period + Rotor_Angle_Acceleration * PWM_Period^2

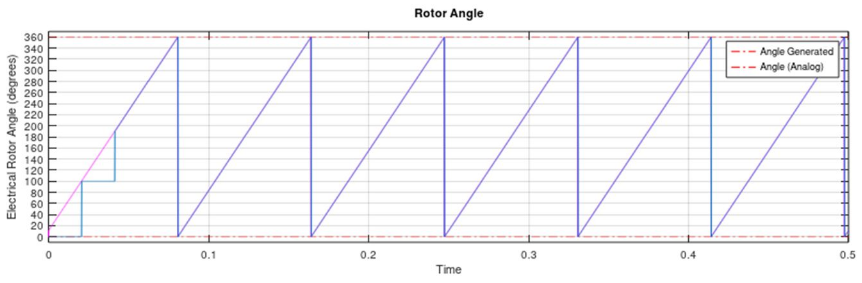
6 SIMULATION

Using simulated hall sensor data, the logic was implemented in Octave. The first plot shows the quadrature analog sensor feedback, with red dashed lines denoting the hysteresis levels. The second plot shows the quantized hall sensor data, based on hysteresis used. The third plot shows the output of the quadrature transition state table, where the 4 angle updates can be see in blue and the direction in orange (high = forward). The last plot shows the resulting rotor angle in blue, with the rate angle calculation applied. The purple line is a reference based on the original analog hall sensor data, generated by a quadrant determination and reverse calculation. It is noticeable the rotor angle in blue begins tracking after the period of state updates can measured.

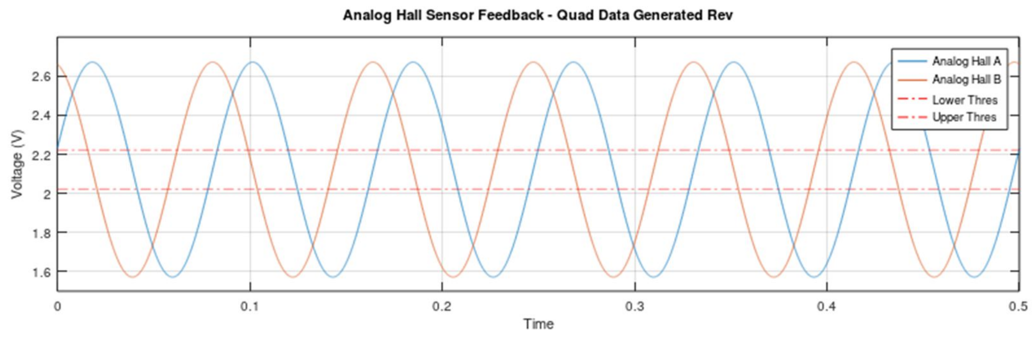


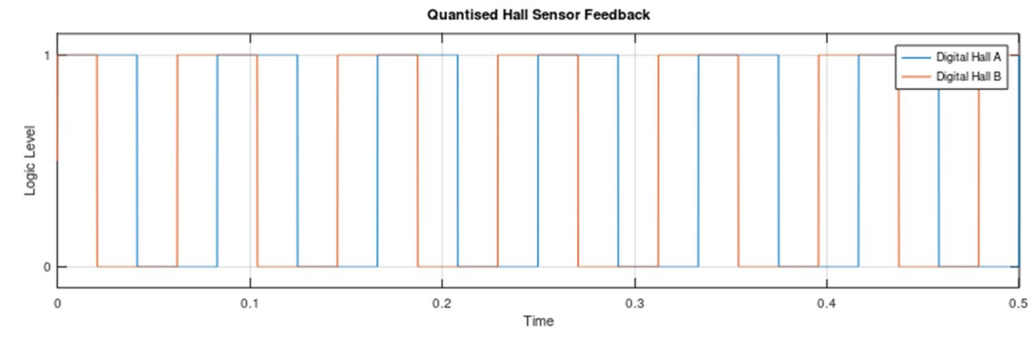


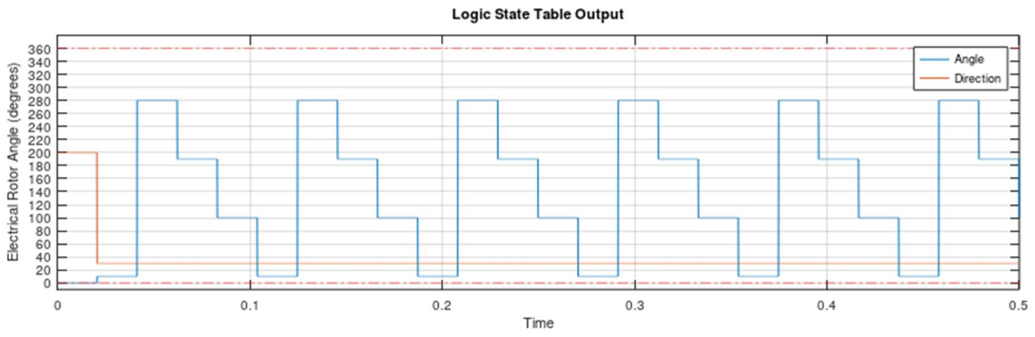


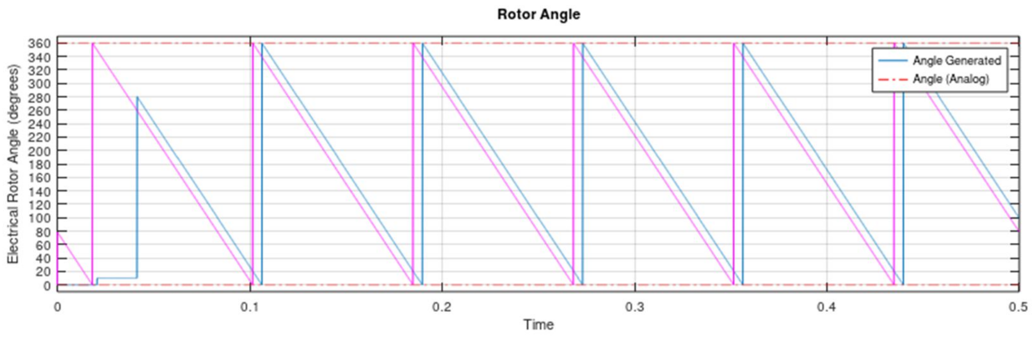


The example below is simulated data, with reverse rotation.

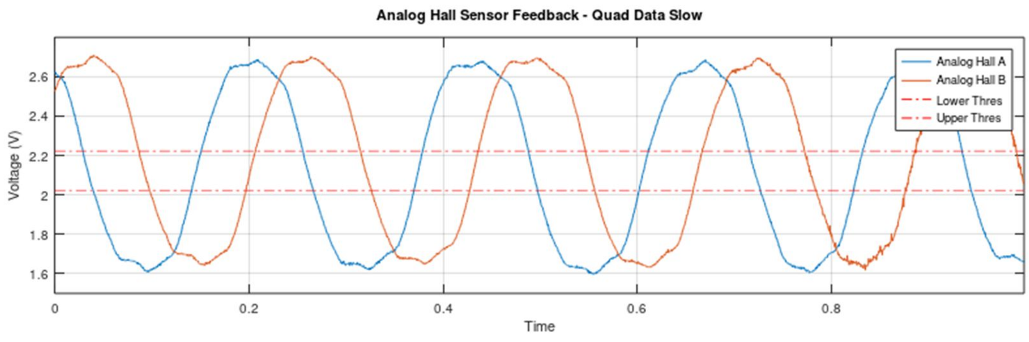


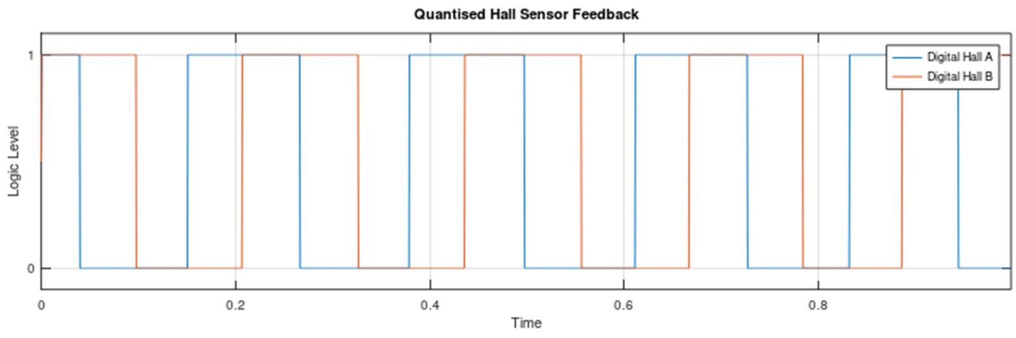


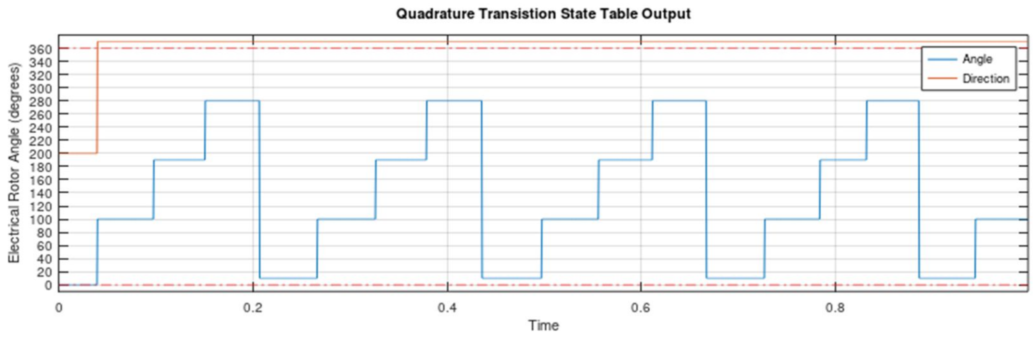


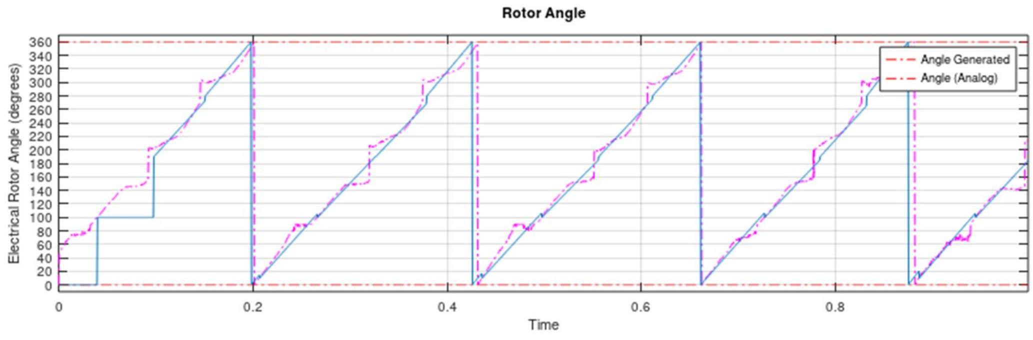


Sample data from the motor rotating slowly. Hall sensor data has low distortion. The blue rotor angle line is quite good. Notice the purple line calculated from the analog data is less ideal, due to the distortion.

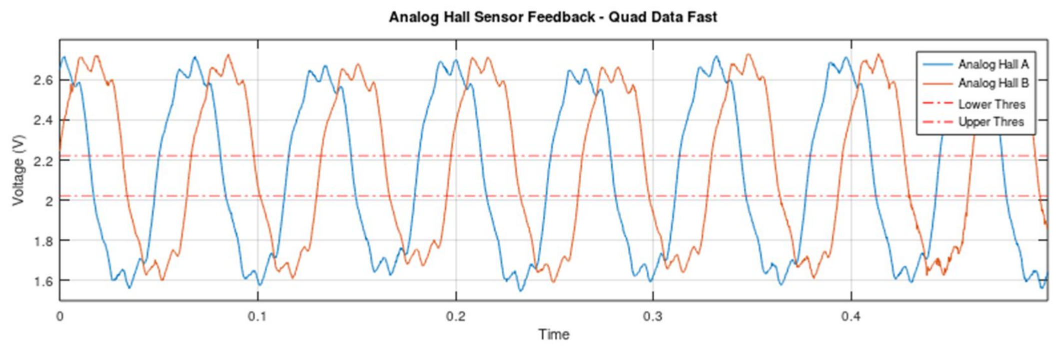


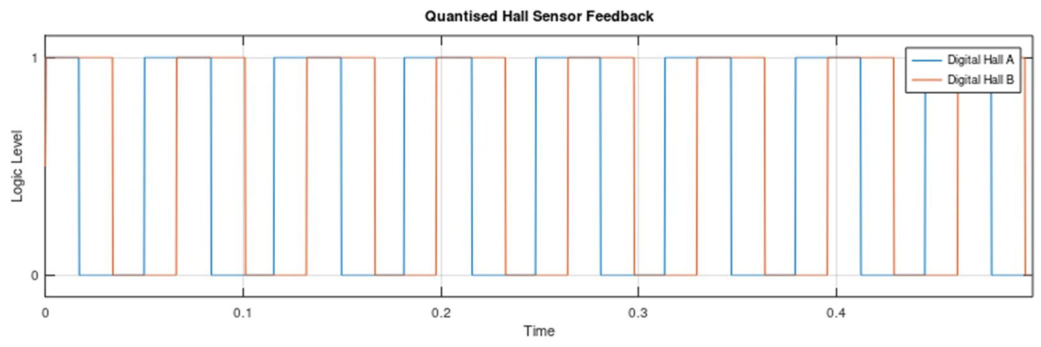


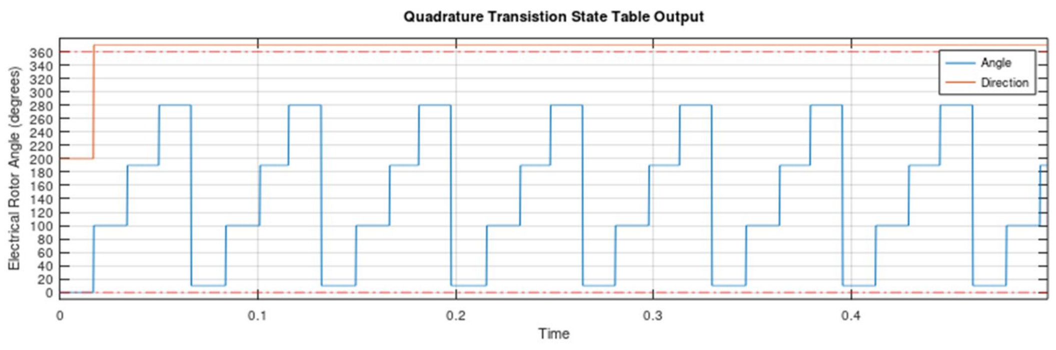


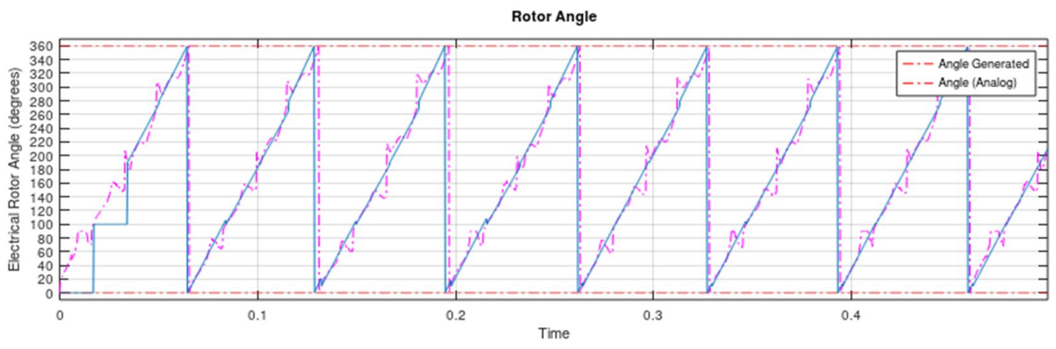


Sampled data from the motor rotating at a fast rate. The rotor angle calculation is looking quite consistent, despite the sensor distortion. The purple calculated angle from the analog data has a lot more noise.

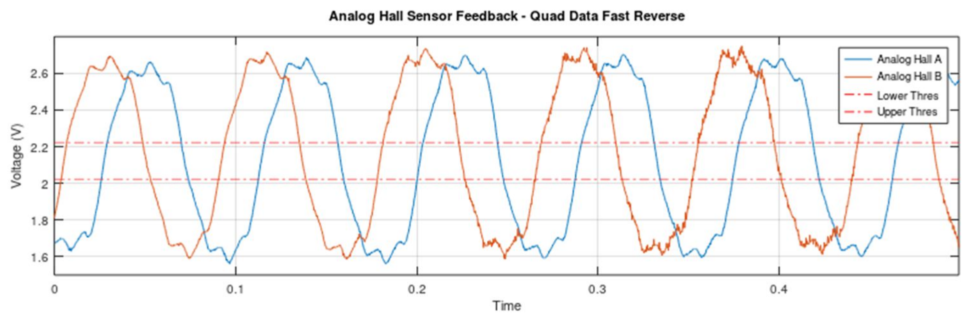


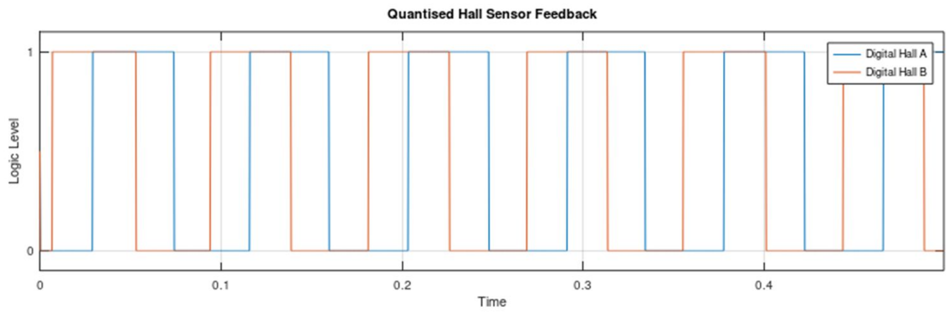


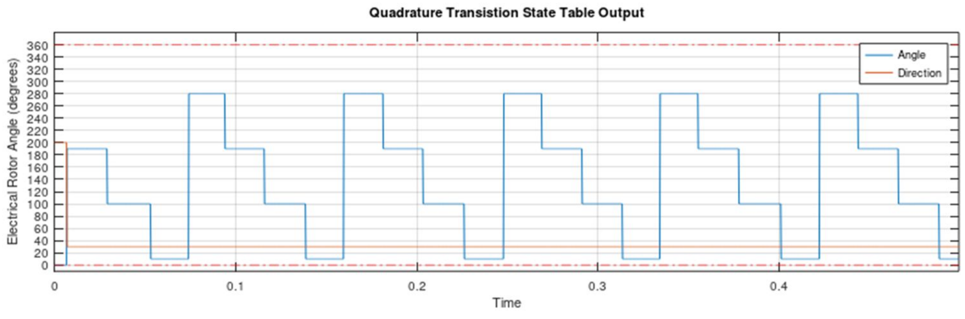


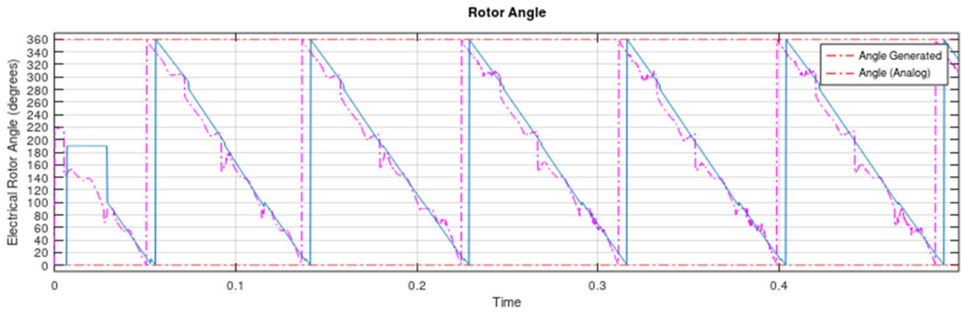


Sampled data from the motor rotating at a relative fast rate, in the reverse direction.

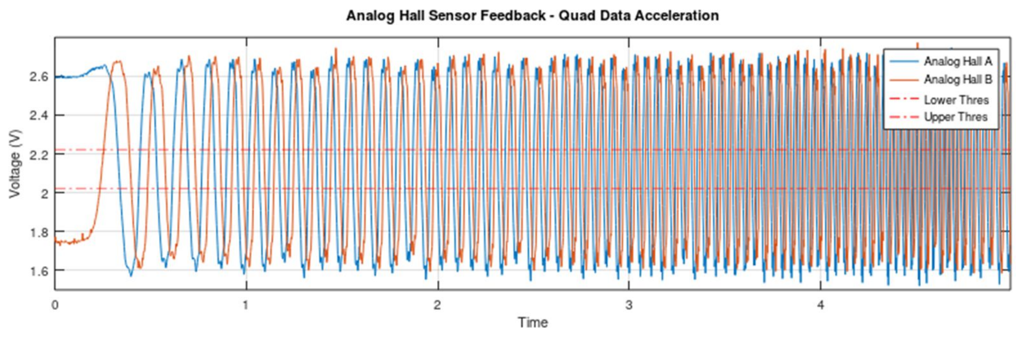


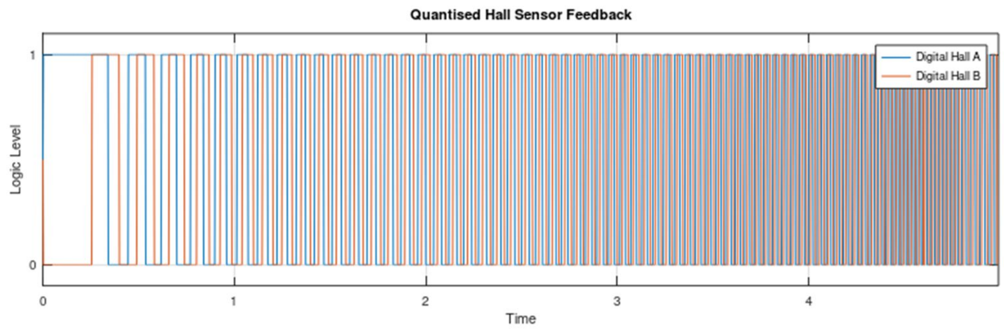


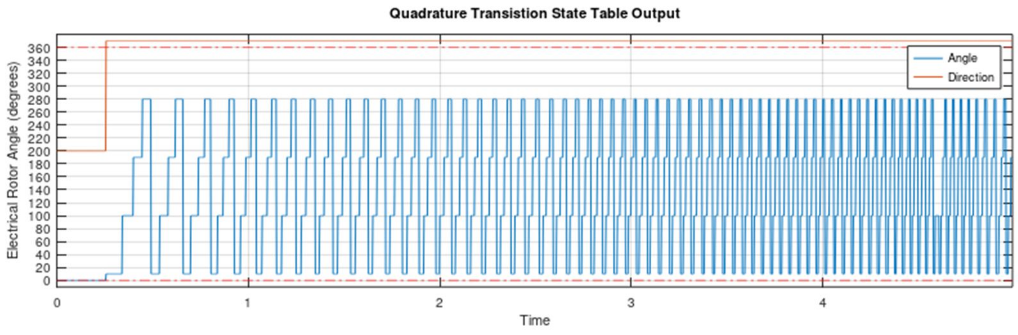


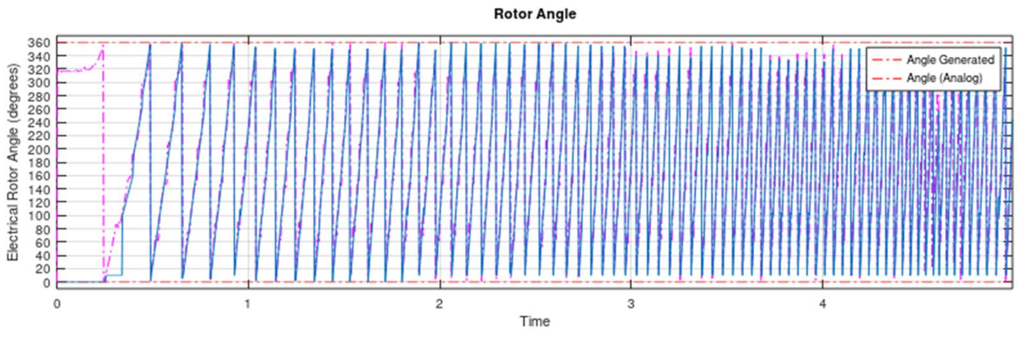


Sampled data from the stationary motor accelerating in rotation, ~5 seconds.

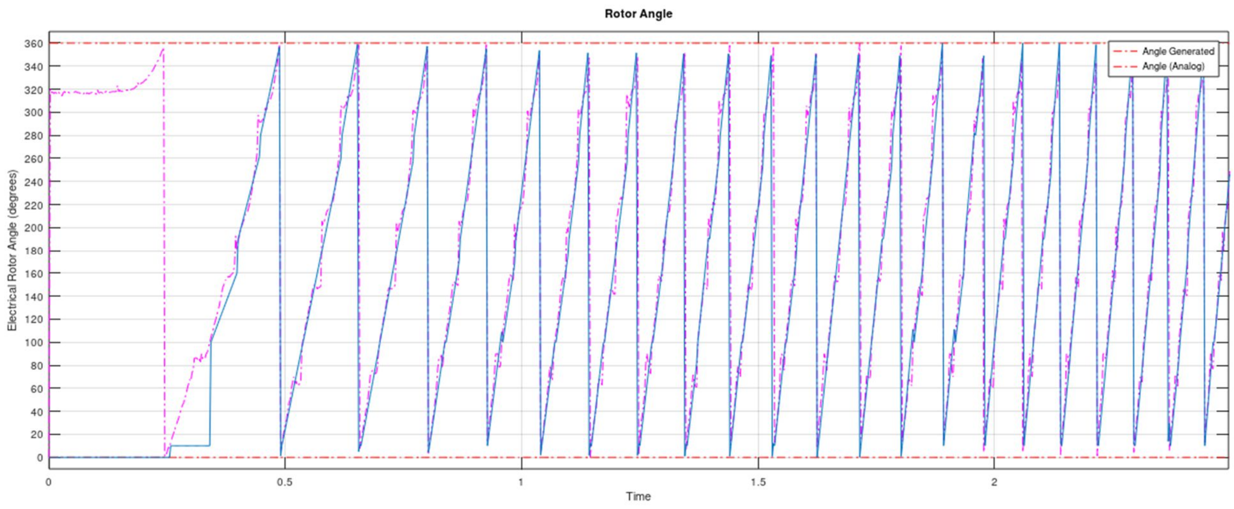




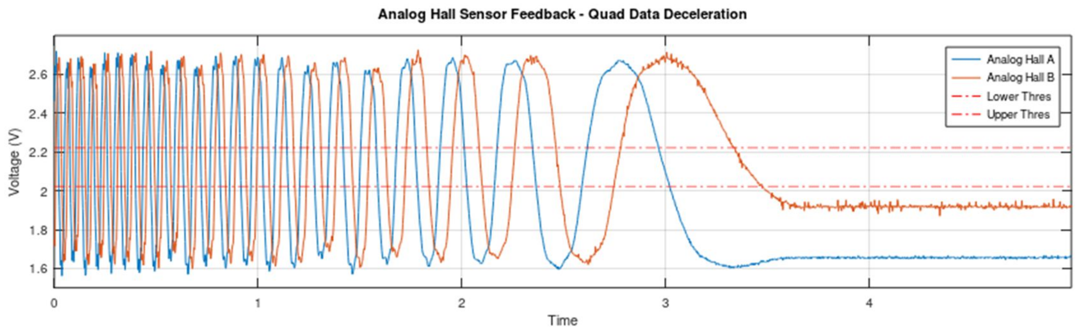


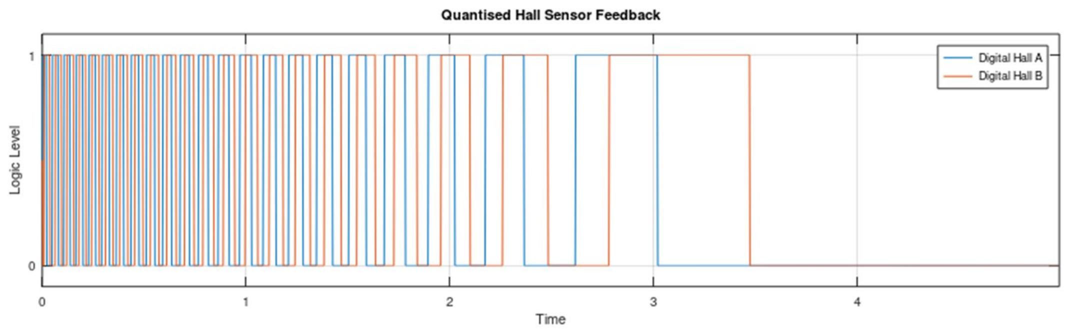


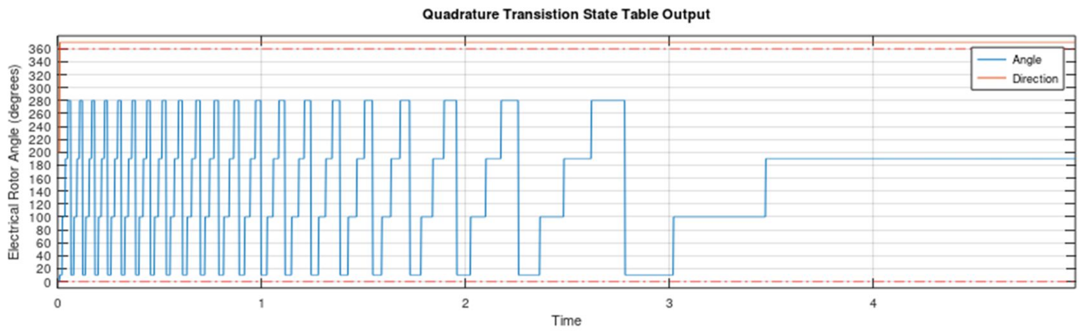
The following plot show a close up of the first ~2.5 seconds. Not noticeable clipping or jagged corrections.

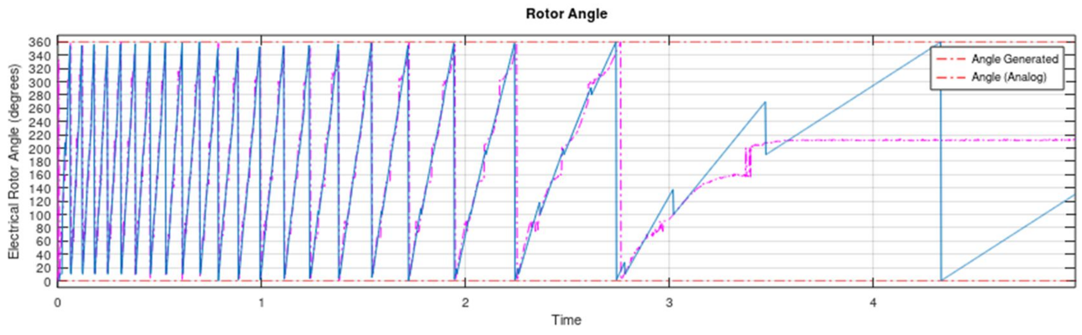


Sampled data from a rotating motor decelerating to a stationary position.

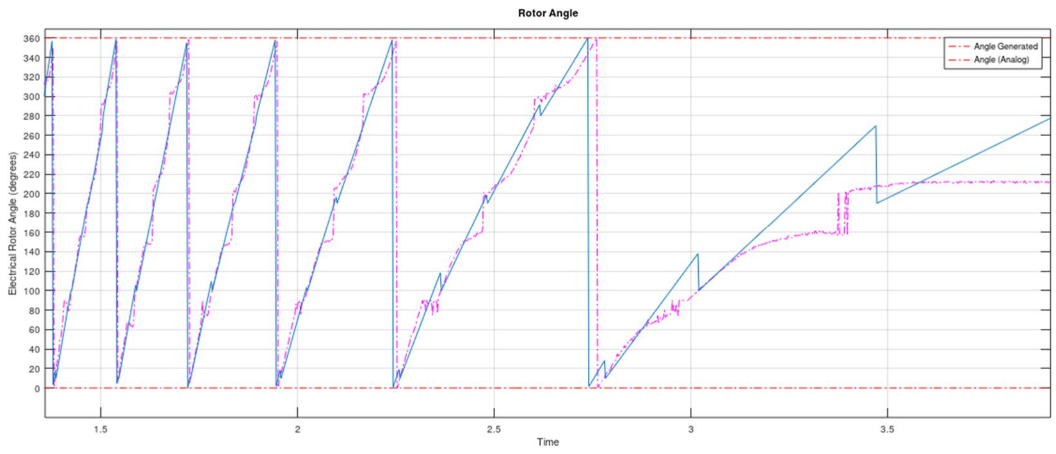




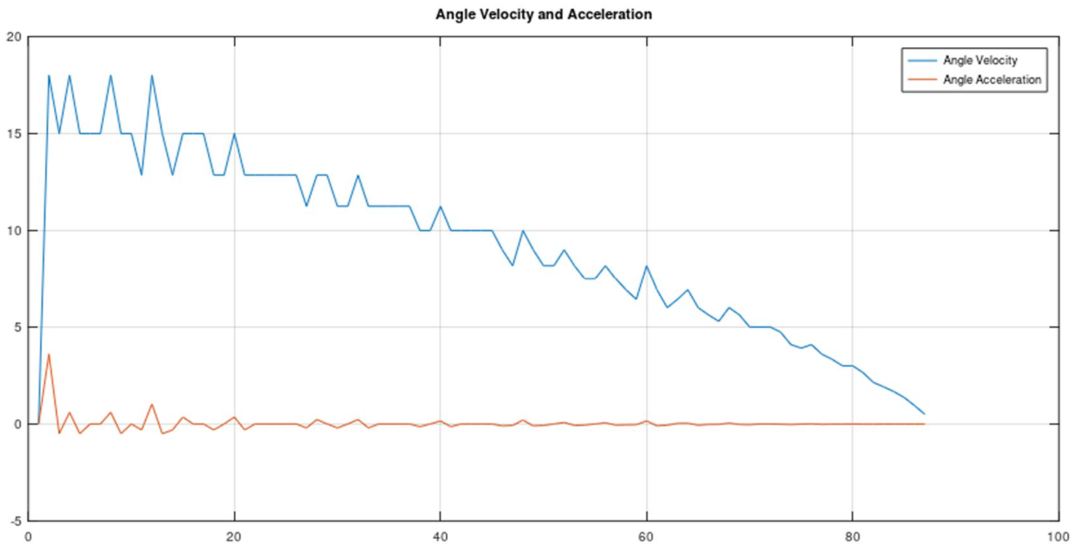




Close up of the deceleration. Minor angle reversals observed. A refinement to the angle update would be to only apply updates in the continuous direction of established rotation. This would result in an increasing or decreasing stepped result, rather than a sharp temporary reversal.



The quadrature updates that generate the angular velocity and acceleration for the deceleration test is shown below. The acceleration parameter (orange) has very little contribution to the rotor angle tracking for this example. It will likely only contribute with significance during higher levels of dynamic performance.



7 COMMENTS

The approach taken to quantized the hall sensor data and generate updates has generated robust results from sampled data. It is expected to accommodate the magnetic field distortion and accommodate higher levels of noise.

This approach will also suit using latched digital hall sensors. The advantage is these do not require ADC inputs that require sampling. Instead, they can be used with digital inputs at higher rates and with less processor overhead. Preferably associated with a quadrature peripheral.

The current implementation uses measured analog hall sensor data as constants. However the machine will have variability. The Vz zero reference and Vpp peak to peak values will vary due to electrical tolerances, magnet strength and mechanical tolerance and property differences. A robust approach would be to have an adaptive Vz and Vpp for each hall sensor.

Sample data from the motor was used to validate the method. This data was collected for the motor rotating slowly, fast, in different directions and with both acceleration and deceleration. The fastest observed operation had ~23 electrical rotations in 1.0 seconds. 23/(7_segments) = 197rpm. Therefore, the term "fast" is based on the limits of a cordless drill system. Testing was not possible through the target 7000rpm range.

The Octave source code has been written with the intent for being a pseudo code reference. The code is in a time indexed sequential format, not matrixed operations. Conditional if-then logic is used and processing separated into functional blocks.

The key blocks to implement the system are:

1. Hall Sensor Logic Translation
2. Quadrature Transition State Machine
3. Rotor Angle Update

8 ROTOR ANGLE START/STOP

The following requirement is to support rotor angle detection/tracking during start/stop conditions, where the slow or non-existent rotor movement is not sufficiently triggering hall effect transition points for the rotor angle estimation to function accurately. This includes establishing an initial system power on start-up angle. In addition, during motor stopping events, the estimator could persist and run-away with old non-zero velocity and acceleration data.

The quadrature analog hall sensors provide feedback that could be used to determine rotors electrical position, however previously this analog data is quantized due to the significant signal distortion. An important distinction is that at zero or near zero rotation speed, the distortion is relatively low. It is anticipated that using the analog data will provide the most accurate tracking of rotor position at this very low or zero speed. This follows the original intent of using analog hall sensors, where absolute rotor position is known for start/stop transitions.

8.1 TRIGONOMETRIC ANGLE CALCULATION

Trigonometry and sector classification can be used to derive the rotor angle from the quadrature hall feedback. This will be referred to as "Quadrature rotor angle tracking". Angle tracking the abbreviated phase. The following Ra and Rb calculations are used conditionally to derive the electrical angle.

$$R_a = \frac{360}{2\pi}\cos^{-1}\left(\frac{2}{1.1}(Hall_A - Hall_{zero\_ref}\right)$$

$$R_b = \frac{360}{2\pi}\sin^{-1}\left(\frac{2}{1.1}(Hall_B - Hall_{zero\_ref}\right)$$

$$Rotor_{Angle} = \begin{cases} R_b & while\ \left(Hall_a \geq Hall_{zero_{ref}}\ AND\ Hall_b \geq Hall_{zero_{ref}}\right) \\ R_b + 360 & while\ \left(Hall_a \geq Hall_{zero_{ref}}\ AND\ Hall_b < Hall_{zero_{ref}}\right) \\ R_a & while\ \left(Hall_a < Hall_{zero_{ref}}\ AND\ Hall_b \geq Hall_{zero_{ref}}\right) \\ 360 - R_a & while\ (Hall_a < Hall_{zero_{ref}}\ AND\ Hall_b < Hall_{zero_{ref}}) \end{cases}$$

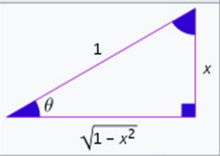
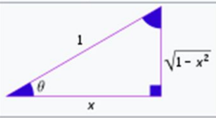
The TMS320F280049 incorporates a Trigonometric Math Unit (TMU), which provides an extended instruction set for the following functions.

Table 6-10. TMU Supported Instructions

INSTRUCTIONS	C EQUIVALENT OPERATION	PIPELINE CYCLES
MPY2PIF32 RaH,RbH	a = b * 2pi	2/3
DIV2PIF32 RaH,RbH	a = b / 2pi	2/3
DIVF32 RaH,RbH,RcH	a = b/c	5
SQRTF32 RaH,RbH	a = sqrt(b)	5
SINPUF32 RaH,RbH	a = sin(b*2pi)	4
COSPUF32 RaH,RbH	a = cos(b*2pi)	4
ATANPUF32 RaH,RbH	a = atan(b)/2pi	4
QUADF32 RaH,RbH,RcH,RdH	Operation to assist in calculating ATANPU2	5

The equations require asin, acos and the square root function. Unfortunately, asin and acos are not supported by the TMU, however it does support atan.

There is a mathematical equivalent for calculating asin and acos as a function of atan and square root (see below, Wiki). However, the acos equivalent is suited for geometric positive inputs, not negative inputs. This can be corrected for the application by negating and adding a positive offset for the specific negative input quadrant.

θ	$\sin(\theta)$	$\cos(\theta)$	$\tan(\theta)$	Diagram
$\arcsin(x)$	$\sin(\arcsin(x)) = x$	$\cos(\arcsin(x)) = \sqrt{1-x^2}$	$\tan(\arcsin(x)) = \frac{x}{\sqrt{1-x^2}}$	
$\arccos(x)$	$\sin(\arccos(x)) = \sqrt{1-x^2}$	$\cos(\arccos(x)) = x$	$\tan(\arccos(x)) = \frac{\sqrt{1-x^2}}{x}$	

An additional intermediate variable is added to the calculations to accommodate the asin & acos functions. Also included is appropriate logic to boundary check the values to avoid erroneous and out of range calculations.

$$X_a = \lim_{-1\ to\ 1}\left[\frac{2}{1.1}(Hall_A - Hall_{zero\_ref}\right]$$

$$X_b = \lim_{-1\ to\ 1}\left[\frac{2}{1.1}(Hall_B - Hall_{zero\_ref}\right]$$

Limits applied to prevent division by zero, as well as very high numbers.

$$V_a = \begin{cases} 10000 & while\ X_a = 0 \\ \dfrac{\sqrt{1 - X_a^2}}{X_a} & while\ X_a > 0 \end{cases}$$

$$V_b = \begin{cases} -10000 & while\ X_b = -1 \\ 10000 & while\ X_b = 1 \\ \dfrac{X_b}{\sqrt{1 - X_b^2}} & while\ X_a > 0 \end{cases}$$

Calculation using atan function and adjustment for acos negative quadrant results…

$$R_a = 180 - \frac{360}{2\pi} tan^{-1}(V_a)$$

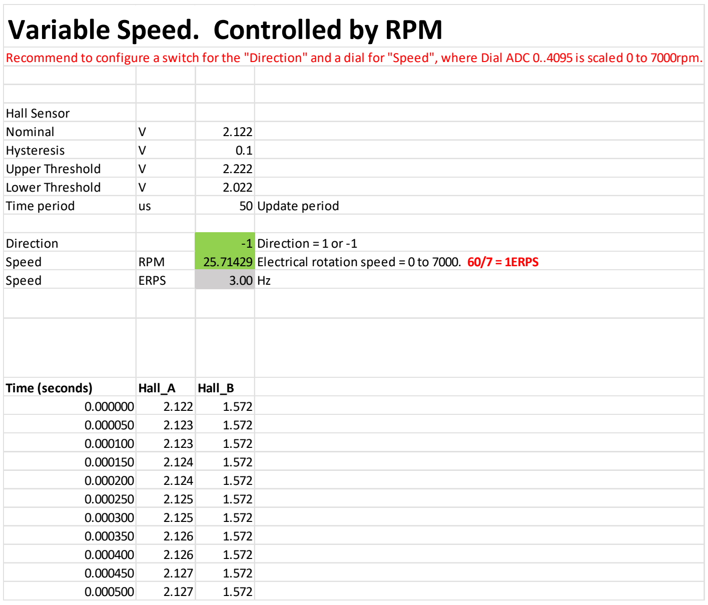
$$R_b = \frac{360}{2\pi} tan^{-1}(V_b)$$

The same conditional output equations apply for determining which quadrant is active to derive the angle.

$$Rotor_{Angle} = \begin{cases} R_b & while\ \left(Hall_a \geq Hall_{zero_{ref}}\ AND\ Hall_b \geq Hall_{zero_{ref}}\right) \\ R_b + 360 & while\ \left(Hall_a \geq Hall_{zero_{ref}}\ AND\ Hall_b < Hall_{zero_{ref}}\right) \\ R_a & while\ \left(Hall_a < Hall_{zero_{ref}}\ AND\ Hall_b \geq Hall_{zero_{ref}}\right) \\ 360 - R_a & while\ (Hall_a < Hall_{zero_{ref}}\ AND\ Hall_b < Hall_{zero_{ref}}) \end{cases}$$

8.2 SIMULATION

The equations and associated logic were implemented in Excel. Also implemented where simulated hall sensor outputs. The motor direction and RPM can be is user controlled by the green fields.

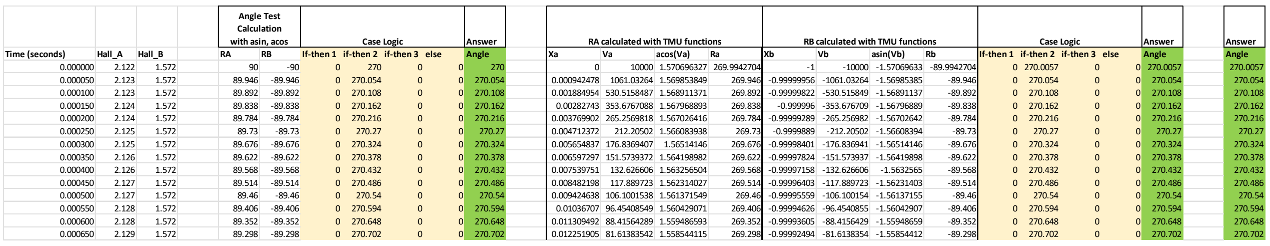


Variable Speed. Controlled by RPM

Recommend to configure a switch for the "Direction" and a dial for "Speed", where Dial ADC 0..4095 is scaled 0 to 7000rpm.

Hall Sensor			
Nominal	V	2.122	
Hysteresis	V	0.1	
Upper Threshold	V	2.222	
Lower Threshold	V	2.022	
Time period	us	50	Update period
Direction		-1	Direction = 1 or -1
Speed	RPM	25.71429	Electrical rotation speed = 0 to 7000. **60/7 = 1ERPS**
Speed	ERPS	3.00	Hz

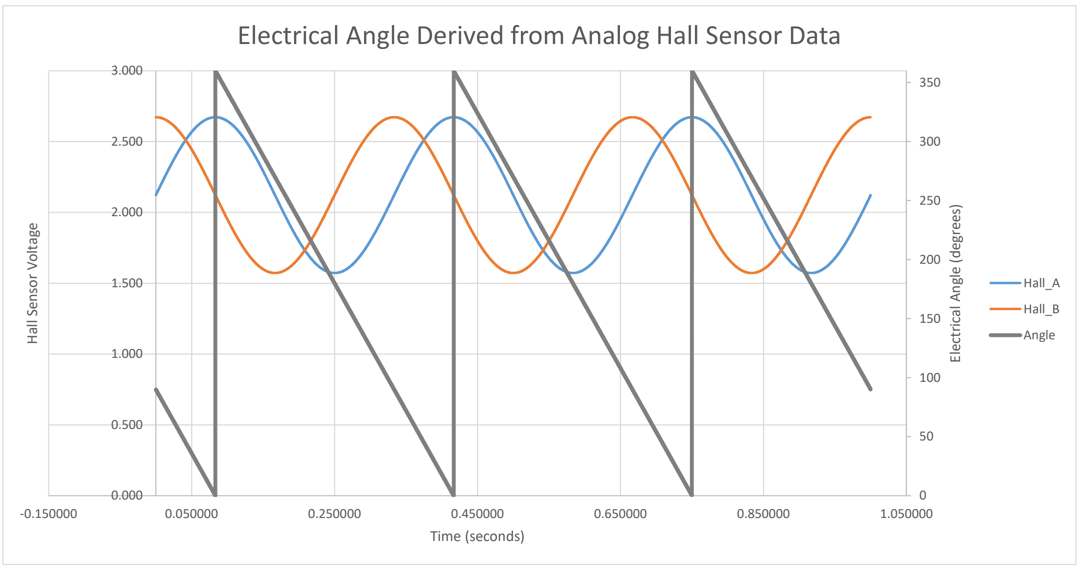
Time (seconds)	**Hall_A**	**Hall_B**
0.000000	2.122	1.572
0.000050	2.123	1.572
0.000100	2.123	1.572
0.000150	2.124	1.572
0.000200	2.124	1.572
0.000250	2.125	1.572
0.000300	2.125	1.572
0.000350	2.126	1.572
0.000400	2.126	1.572
0.000450	2.127	1.572
0.000500	2.127	1.572

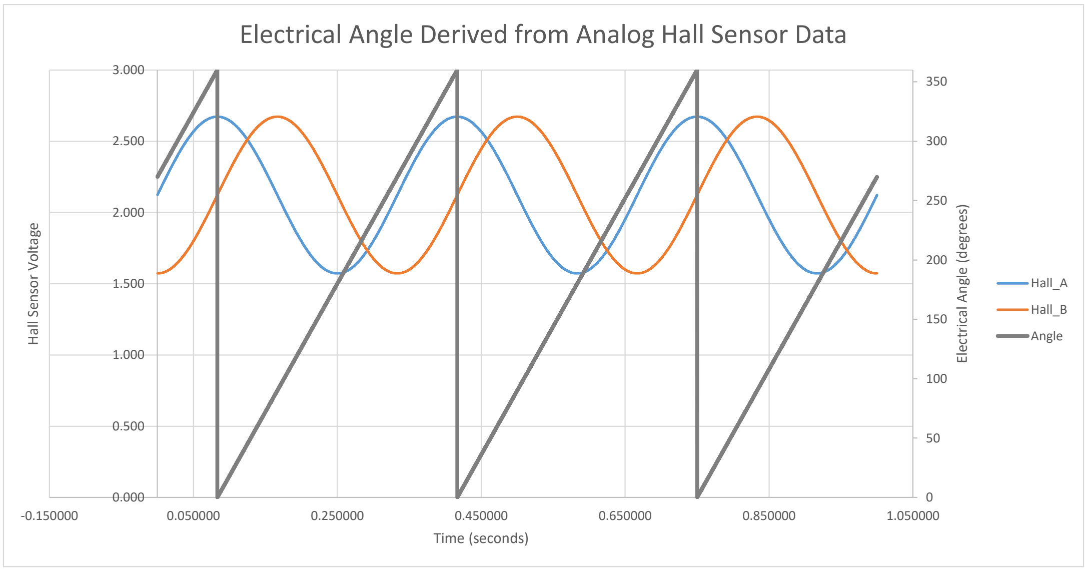


			Angle Test Calculation with asin, acos		Case Logic				Answer
Time (seconds)	**Hall_A**	**Hall_B**	**RA**	**RB**	**If-then 1**	**if-then 2**	**if-then 3**	**else**	**Angle**
0.000000	2.122	1.572	90	-90	0	270	0	0	270
0.000050	2.123	1.572	89.946	-89.946	0	270.054	0	0	270.054
0.000100	2.123	1.572	89.892	-89.892	0	270.108	0	0	270.108
0.000150	2.124	1.572	89.838	-89.838	0	270.162	0	0	270.162
0.000200	2.124	1.572	89.784	-89.784	0	270.216	0	0	270.216
0.000250	2.125	1.572	89.73	-89.73	0	270.27	0	0	270.27
0.000300	2.125	1.572	89.676	-89.676	0	270.324	0	0	270.324
0.000350	2.126	1.572	89.622	-89.622	0	270.378	0	0	270.378
0.000400	2.126	1.572	89.568	-89.568	0	270.432	0	0	270.432
0.000450	2.127	1.572	89.514	-89.514	0	270.486	0	0	270.486
0.000500	2.127	1.572	89.46	-89.46	0	270.54	0	0	270.54
0.000550	2.128	1.572	89.406	-89.406	0	270.594	0	0	270.594
0.000600	2.128	1.572	89.352	-89.352	0	270.648	0	0	270.648
0.000650	2.129	1.572	89.298	-89.298	0	270.702	0	0	270.702

	RA calculated with TMU functions				RB calculated with TMU functions				Case Logic				Answer	Answer
Time (seconds)	**Xa**	**Va**	**acos(Va)**	**Ra**	**Xb**	**Vb**	**asin(Vb)**	**Rb**	**If-then 1**	**if-then 2**	**if-then 3**	**else**	**Angle**	**Angle**
0.000000	0	10000	1.570696327	269.9942704	-1	-10000	-1.57069633	-89.9942704	0	270.0057	0	0	270.0057	270.0057
0.000050	0.000942478	1061.03264	1.569853849	269.946	-0.99999956	-1061.03264	-1.56985385	-89.946	0	270.054	0	0	270.054	270.054
0.000100	0.001884954	530.5158487	1.568911371	269.892	-0.99999822	-530.515849	-1.56891137	-89.892	0	270.108	0	0	270.108	270.108
0.000150	0.00282743	353.6767088	1.567968893	269.838	-0.999996	-353.676709	-1.56796889	-89.838	0	270.162	0	0	270.162	270.162
0.000200	0.003769902	265.2569818	1.567026416	269.784	-0.99999289	-265.256982	-1.56702642	-89.784	0	270.216	0	0	270.216	270.216
0.000250	0.004712372	212.20502	1.566083938	269.73	-0.9999889	-212.20502	-1.56608394	-89.73	0	270.27	0	0	270.27	270.27
0.000300	0.005654837	176.8369407	1.56514146	269.676	-0.99998401	-176.836941	-1.56514146	-89.676	0	270.324	0	0	270.324	270.324
0.000350	0.006597297	151.5739372	1.564198982	269.622	-0.99997824	-151.573937	-1.56419898	-89.622	0	270.378	0	0	270.378	270.378
0.000400	0.007539751	132.626606	1.563256504	269.568	-0.99997158	-132.626606	-1.5632565	-89.568	0	270.432	0	0	270.432	270.432
0.000450	0.008482198	117.889723	1.562314027	269.514	-0.99996403	-117.889723	-1.56231403	-89.514	0	270.486	0	0	270.486	270.486
0.000500	0.009424638	106.1001538	1.561371549	269.46	-0.99995559	-106.100154	-1.56137155	-89.46	0	270.54	0	0	270.54	270.54
0.000550	0.01036707	96.45408549	1.560429071	269.406	-0.99994626	-96.4540855	-1.56042907	-89.406	0	270.594	0	0	270.594	270.594
0.000600	0.011309492	88.41564289	1.559486593	269.352	-0.99993605	-88.4156429	-1.55948659	-89.352	0	270.648	0	0	270.648	270.648
0.000650	0.012251905	81.61383542	1.558544115	269.298	-0.99992494	-81.6138354	-1.55854412	-89.298	0	270.702	0	0	270.702	270.702

The black plot shows the angle operating cleanly between 0 to 360 degrees (secondary scale on right). The math was tested for both forward and reverse rotor directions.





8.3 IMPLEMENATION

At any one time, only Ra or Rb needs to be calculated. Not both. The quadrant classification should be done first to determine whether Ra or Rb needs to be calculated.

$$Rotor_{Angle} = \begin{cases} R_b & while\ \left(Hall_a \geq Hall_{zero_{ref}}\ AND\ Hall_b \geq Hall_{zero_{ref}}\right) \\ R_b + 360 & while\ \left(Hall_a \geq Hall_{zero_{ref}}\ AND\ Hall_b < Hall_{zero_{ref}}\right) \\ R_a & while\ \left(Hall_a < Hall_{zero_{ref}}\ AND\ Hall_b \geq Hall_{zero_{ref}}\right) \\ 360 - R_a & while\ (Hall_a < Hall_{zero_{ref}}\ AND\ Hall_b < Hall_{zero_{ref}}) \end{cases}$$

Reference was found on the TI E2E forums regarding the calculation of asin and acos using the same technique and TMU. The code below was in reference to: *"These should compute in ~60 cycles with --fp_mode = relaxed and optimization off."*

```
volatile float x = 0.6513246f;
volatile float v, a, b;
void main(void)
{
// a = asinf(x)
v = x / sqrtf(1 - (x*x));
a = atanf(v);

// b = acosf(x)
v = sqrtf(1 - (x*x)) / x;
b = atanf(v);
```

Also note, the TMU direct supports instructions for multiplication by 2π and division by 2π.

- MPY2PIF32
- DIV2PIF32

8.4 TESTING

8.4.1 SIMULATED HALL SENSOR DATA

The following equations can be used to generated simulated hall sensor feedback of the motor rotating.

$$Hall_{A\_simulated}(t) = \frac{1.1}{2}\sin(2\pi t * ERPS) + Hall_{zero\_ref}$$

$$Hall_{B_{simulated}}(t) = \frac{1.1}{2}\sin\left(2\pi t * ERPS + \frac{2\pi}{4} * direction\right) + Hall_{zero_{ref}}$$

Where:

$direction$ = -1, 1 (reverse, forwards)

$ERPS$ = electrical rotations per second.

These equations are also implemented in the Excel file.

8.4.2 SIMULATED ROTOR SPEED/DIRECTION

It is recommended to use the user controls on the diagnostic board to control the simulated hall sensor data. Configure a switch for the "Direction" and a dial for "Speed", where Dial ADC 0..4095 is scaled 0 to 49000ERPS.

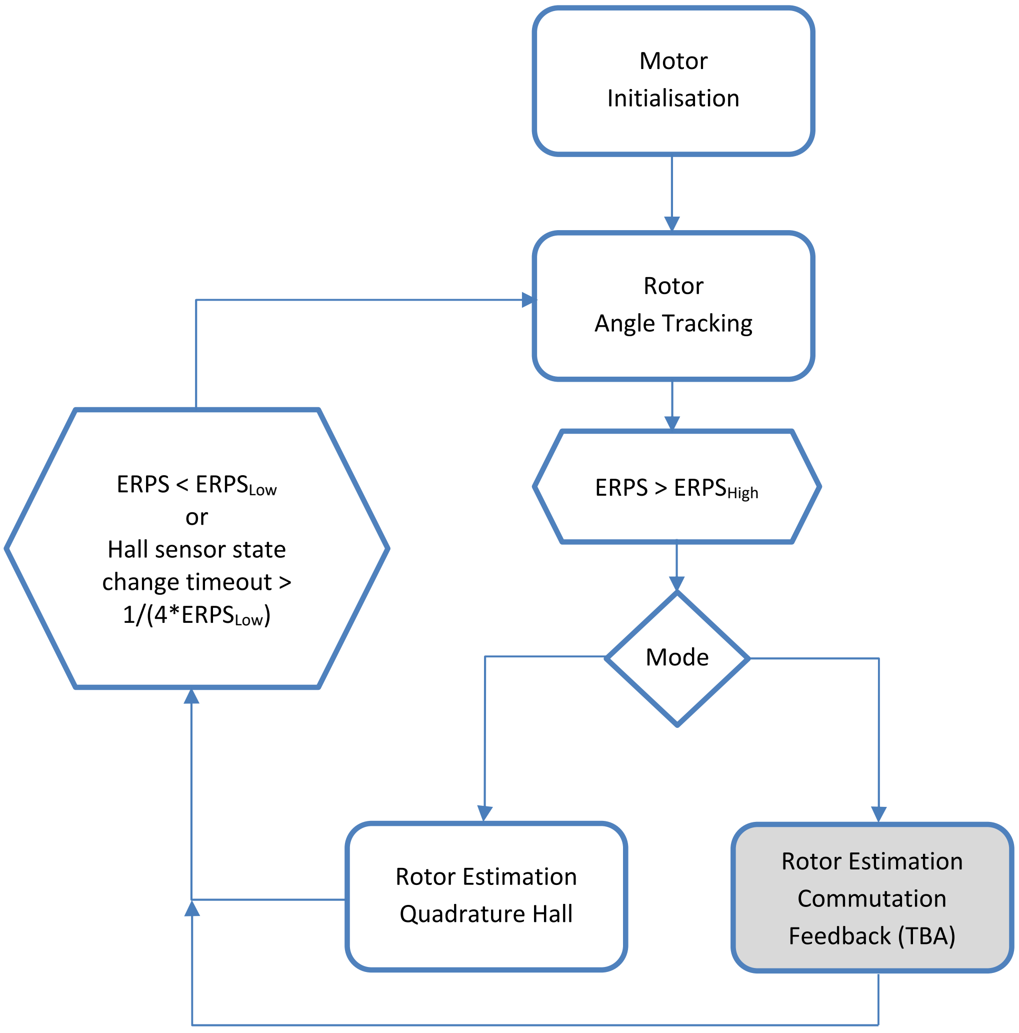
These simulated inputs will be very useful for later testing the sector rotation, which is based on the derived electrical rotor angle.

9 ROTOR ANGLE SELECTION

There are now two methods to determine the electrical rotor angle. The first is rotor angle estimation with quadrature updates. The second is the new quadrature angle tracking for low speed and start/stop operation.

Switching between these two methods requires a state machine and hysteresis. Two parameters are used for controlling the selection between rotor tracking/estimation algorithms, $ERPS_{High}$ and $ERPS_{Low}$. Essentially the mode will change when the rotor speed increases or decreases past these thresholds, with hysteresis in-between. A timeout period is also required for the hall sensor state changes, which is directly linked to the $ERPS_{Low}$ parameter to maintain consistent logic. This will ensure that even a sudden rotor stop event will result in the hall sensor state change timeout triggering and the system will quickly transition to angle tracking.

- $ERPS_{High}$ = 30 ERPS
- $ERPS_{Low}$ = 15 ERPS


Motor Initialisation
Rotor Angle Tracking
ERPS > $ERPS_{High}$
Mode
Rotor Estimation Quadrature Hall
Rotor Estimation Commutation Feedback (TBA)
ERPS < $ERPS_{Low}$ or Hall sensor state change timeout > 1/(4*$ERPS_{Low}$)

Rotor angle tracking is only required to be calculated while in its respective state. Rotor estimation should always be running to facilitate its state machine and timed period between state changes, even though the rotor angle is ignored. However, the rotor angle velocity and acceleration calculation should be zero while $ERPS < ERPS_{low}$ to ensure erroneous calculations are not observed.

10 OCTAVE SOURCE CODE

Updated with Hall Tracking Algorithm.

```
clear all; clf

% Constants
%Hall_Zero_Reference = 2.122;
%Hall_Hysteresis = 0.1;
%Hall_Upper_Threshold = Hall_Zero_Reference + Hall_Hysteresis;
%Hall_Lower_Threshold = Hall_Zero_Reference - Hall_Hysteresis;

Hall_min_range = 0.5;
Hall_over_range = 1.25;
Hall_mid_init = 2.0;
Hall_hysteresis = 0.2;
Hall_convergence_period_fast = 0.001;
Hall_convergence_period_slow = 0.04;
Hall_convergence_rate = 0.005;     % Voltage step size


TRUE = 1;
FALSE = 0;

% Load Sampled Test Data
load Quad_Data.mat

% Math Quadrant Test Data
% This data is only for the simulated testing and not functionally relevant.
Hall_Zero_Reference = Hall_mid_init;
time = 0:0.5/5000:0.5;
angle_offset = 11.25/360*2*pi();
x=1.1/2*sin(2*pi()*6*time./0.5 + angle_offset) + Hall_Zero_Reference;
x2=1.1/2*sin(2*pi()*6*time./0.5 + angle_offset + 2*pi()*(11.5-7.5)*22.5/360) +
Hall_Zero_Reference;
a = 2.*(360/(2*pi())).*acos((2.*(x-Hall_Zero_Reference)./(1.1)));
%a = linspace(0,(6 * 360),size(time,2));
a2 = 360-mod(a,360);
%plot(time,x2); hold on; plot(time, x); plot(time, a2./360+1.6);
%axis([min(Time) max(time) 1.5 2.8]); grid
Quad_Data_Generated_Fwd = [time ; x2 ; x ; 360-a2]';
Quad_Data_Generated_Rev = [time ; x ; x2 ; a2]';
clear time x2 x a a2;

% Set Data to Use
%Quad_Data = Quad_Data_Generated_Fwd;   qd_title = 'Quad Data Generated Fwd';
%Quad_Data = Quad_Data_Generated_Rev;   qd_title = 'Quad Data Generated Rev';
%Quad_Data = Quad_Data_Measured;   qd_title = 'Quad Data Measured';
%Quad_Data = Quad_Data_Slow;   qd_title = 'Quad Data Slow';
Quad_Data = Quad_Data_Fast;   qd_title = 'Quad Data Fast';
%Quad_Data = Quad_Data_Fast_Reverse;   qd_title = 'Quad Data Fast Reverse';
%Quad_Data = Quad_Data_Accel;   qd_title = 'Quad Data Acceleration';
%Quad_Data = Quad_Data_Decel;   qd_title = 'Quad Data Deceleration';

Quad_Data(:,2) = Quad_Data(:,2) +0.3;

% Initialisation
Time = Quad_Data(:,1);
Analog_Hall_A = Quad_Data(:,2);
Analog_Hall_B = Quad_Data(:,3);
Digital_Hall_A = zeros(size(Time));
Digital_Hall_B = zeros(size(Time));
Hall_Upper_Threshold_A = zeros(size(Time));
Hall_Upper_Threshold_B = zeros(size(Time));
Hall_Lower_Threshold_A = zeros(size(Time));
Hall_Lower_Threshold_B = zeros(size(Time));
Hall_Max_A = zeros(size(Time));
```

```
Hall_Mid_A = zeros(size(Time));
Hall_Min_A = zeros(size(Time));
Hall_Max_B = zeros(size(Time));
Hall_Mid_B = zeros(size(Time));
Hall_Min_B = zeros(size(Time));
Direction = zeros(size(Time));
Rotor_Angle = zeros(size(Time));
Rotor_Angle_State_Machine = zeros(size(Time));
Rotor_Correction_Time = Time(1);
Rotor_Last_Correction_Angle = -1;
Rotor_Angle_Velocity = 0;
Rotor_Angle_Acceleration = 0;
State_Change_Lockout_Timer = 0;

% Initial Conditions
Digital_Hall_A(1)=0.5;   % Usually 0 or 1.  Different value to force initial state change.
Digital_Hall_B(1)=0.5;

##Hall_Mid_A(1) = Hall_mid_init;
##Hall_Mid_B(1) = Hall_mid_init;
##
##Hall_Max_A(1) = Hall_Mid_A(1) + Hall_hysteresis/2;
##Hall_Min_A(1) = Hall_Mid_A(1) - Hall_hysteresis/2;
##Hall_Max_B(1) = Hall_Mid_B(1) + Hall_hysteresis/2;
##Hall_Min_B(1) = Hall_Mid_B(1) - Hall_hysteresis/2;

Hall_Convergence_Timer_Max_A = 0;
Hall_Convergence_Timer_Max_B = 0;
Hall_Convergence_Timer_Min_A = 0;
Hall_Convergence_Timer_Min_B = 0;


%----------------------------------------------------------------
%----------------------------------------------------------------
% Run time loop

for i=2:size(Time,1)

  %----------------------------------------------------------------
  % Hall Tracking

  % Timer Update
  Time_Delta = Time(i) - Time(max(i-1,1));
  if Hall_Convergence_Timer_Max_A > 0
    Hall_Convergence_Timer_Max_A = max(Hall_Convergence_Timer_Max_A - Time_Delta,0);
  endif
  if Hall_Convergence_Timer_Min_A > 0
    Hall_Convergence_Timer_Min_A = max(Hall_Convergence_Timer_Min_A - Time_Delta,0);
  endif
  if Hall_Convergence_Timer_Max_B > 0
    Hall_Convergence_Timer_Max_B = max(Hall_Convergence_Timer_Max_B - Time_Delta,0);
  endif
  if Hall_Convergence_Timer_Min_B > 0
    Hall_Convergence_Timer_Min_B = max(Hall_Convergence_Timer_Min_B - Time_Delta,0);
  endif

  % Maintain variable with time indexed
  Hall_Min_A(i) = Hall_Min_A(i-1);
  Hall_Max_A(i) = Hall_Max_A(i-1);
  Hall_Min_B(i) = Hall_Min_B(i-1);
  Hall_Max_B(i) = Hall_Max_B(i-1);

  % Maximum Tracking
  Hall_Range_A = max(Hall_Max_A(i-1) - Hall_Min_A(i-1),0);
  if Analog_Hall_A(i) >= Hall_Max_A(i-1)
    % 4xIIR Track maximum value
    Hall_Max_A(i) = (3*Hall_Max_A(i-1) + Analog_Hall_A(i))/4;
    Hall_Convergence_Timer_Max_A = Hall_convergence_period_slow;
  elseif  Hall_Range_A >= Hall_over_range && Hall_Convergence_Timer_Max_A == 0;
    % Fast Convergence
    Hall_Max_A(i) = Hall_Max_A(i-1) - Hall_convergence_rate;
    Hall_Convergence_Timer_Max_A = Hall_convergence_period_fast;
  elseif  Hall_Range_A >= Hall_min_range && Hall_Convergence_Timer_Max_A == 0;
```

```
    % Slow Convergence
    Hall_Max_A(i) = Hall_Max_A(i-1) - Hall_convergence_rate;
    Hall_Convergence_Timer_Max_A = Hall_convergence_period_slow;
  endif

  Hall_Range_B = max(Hall_Max_B(i-1) - Hall_Min_B(i-1),0);
  if Analog_Hall_B(i) >= Hall_Max_B(i-1)
    % 4xIIR Track maximum value
    Hall_Max_B(i) = (3*Hall_Max_B(i-1) + Analog_Hall_B(i))/4;
    Hall_Convergence_Timer_Max_B = Hall_convergence_period_slow;
  elseif  Hall_Range_B >= Hall_over_range && Hall_Convergence_Timer_Max_B == 0;
    % Fast Convergence
    Hall_Max_B(i) = Hall_Max_B(i-1) - Hall_convergence_rate;
    Hall_Convergence_Timer_Max_B = Hall_convergence_period_fast;
  elseif  Hall_Range_B >= Hall_min_range && Hall_Convergence_Timer_Max_B == 0;
    % Slow Convergence
    Hall_Max_B(i) = Hall_Max_B(i-1) - Hall_convergence_rate;
    Hall_Convergence_Timer_Max_B = Hall_convergence_period_slow;
  endif

  % Minimum Tracking
  Hall_Range_A = max(Hall_Max_A(i) - Hall_Min_A(i-1),0);
  if Analog_Hall_A(i) <= Hall_Min_A(i-1)
    % 4xIIR Track minimum value
    Hall_Min_A(i) = (3*Hall_Min_A(i-1) + Analog_Hall_A(i))/4;
  elseif  Hall_Range_A >= Hall_over_range && Hall_Convergence_Timer_Min_A == 0;
    % Fast Convergence
    Hall_Min_A(i) = Hall_Min_A(i-1) + Hall_convergence_rate;
    Hall_Convergence_Timer_Min_A = Hall_convergence_period_fast;
  elseif  Hall_Range_A >= Hall_min_range && Hall_Convergence_Timer_Min_A == 0;
    % Slow Convergence
    Hall_Min_A(i) = Hall_Min_A(i-1) + Hall_convergence_rate;
    Hall_Convergence_Timer_Min_A = Hall_convergence_period_slow;
  endif

  Hall_Range_B = max(Hall_Max_B(i) - Hall_Min_B(i-1),0);
  if Analog_Hall_B(i) <= Hall_Min_B(i-1)
    % 4xIIR Track minimum value
    Hall_Min_B(i) = (3*Hall_Min_B(i-1) + Analog_Hall_B(i))/4;
  elseif  Hall_Range_B >= Hall_over_range && Hall_Convergence_Timer_Min_B == 0;
    % Fast Convergence
    Hall_Min_B(i) = Hall_Min_B(i-1) + Hall_convergence_rate;
    Hall_Convergence_Timer_Min_B = Hall_convergence_period_fast;
  elseif  Hall_Range_B >= Hall_min_range && Hall_Convergence_Timer_Min_B == 0;
    % Slow Convergence
    Hall_Min_B(i) = Hall_Min_B(i-1) + Hall_convergence_rate;
    Hall_Convergence_Timer_Min_B = Hall_convergence_period_slow;
  endif

  % Maintain Minimum Range
  if Hall_Max_A(i) - Hall_Min_A(i) < Hall_min_range
    % Maintain a minimum range between max and min
    Hall_Max_A(i) = Hall_Min_A(i) + Hall_min_range;
  endif
  if Hall_Max_B(i) - Hall_Min_B(i) < Hall_min_range
    % Maintain a minimum range between max and min
    Hall_Max_B(i) = Hall_Min_B(i) + Hall_min_range;
  endif

  % Update Hall Thresholds
  Hall_Mid_A = (Hall_Max_A(i) - Hall_Min_A(i))/2 + Hall_Min_A(i);
  Hall_Upper_Threshold_A(i) = Hall_Mid_A + Hall_hysteresis/2;
  Hall_Lower_Threshold_A(i) = Hall_Mid_A - Hall_hysteresis/2;

  Hall_Mid_B = (Hall_Max_B(i) - Hall_Min_B(i))/2 + Hall_Min_B(i);
  Hall_Upper_Threshold_B(i) = Hall_Mid_B + Hall_hysteresis/2;
  Hall_Lower_Threshold_B(i) = Hall_Mid_B - Hall_hysteresis/2;



  %----------------------------------------------------------------
  % Hall Sensor Logic Translation
  if State_Change_Lockout_Timer == 0
    % Digital Hall A
    if Analog_Hall_A(i) > Hall_Upper_Threshold_A(i)
      Digital_Hall_A(i) = 1;
    elseif Analog_Hall_A(i) < Hall_Lower_Threshold_A(i)
      Digital_Hall_A(i) = 0;
    else
      Digital_Hall_A(i) = Digital_Hall_A(i-1);    % Hysteresis
    endif

    % Digital Hall B
    if Analog_Hall_B(i) > Hall_Upper_Threshold_B(i)
      Digital_Hall_B(i) = 1;
    elseif Analog_Hall_B(i) < Hall_Lower_Threshold_B(i)
      Digital_Hall_B(i) = 0;
    else
      Digital_Hall_B(i) = Digital_Hall_B(i-1);
    endif
  else
    State_Change_Lockout_Timer = State_Change_Lockout_Timer -1;
    Digital_Hall_A(i) = Digital_Hall_A(i-1);
    Digital_Hall_B(i) = Digital_Hall_B(i-1);
  endif

  %----------------------------------------------------------------
  % Quadrature Transition State Machine
  % Rotor Angle Correction

  if(Digital_Hall_A(i-1)==0 && Digital_Hall_B(i-1)==0 && Digital_Hall_A(i)==0 &&
Digital_Hall_B(i)==1)
    Rotor_Angle(i) = 190.07;
    Direction(i) = -1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==0 && Digital_Hall_B(i-1)==0 && Digital_Hall_A(i)==1 &&
Digital_Hall_B(i)==0)
    Rotor_Angle(i) = 280.07;
    Direction(i) = 1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==0 && Digital_Hall_B(i-1)==1 && Digital_Hall_A(i)==0 &&
Digital_Hall_B(i)==0)
    Rotor_Angle(i) = 190.07;
    Direction(i) = 1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==0 && Digital_Hall_B(i-1)==1 && Digital_Hall_A(i)==1 &&
Digital_Hall_B(i)==1)
    Rotor_Angle(i) = 100.07;
    Direction(i) = -1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==1 && Digital_Hall_B(i-1)==0 && Digital_Hall_A(i)==0 &&
Digital_Hall_B(i)==0)
    Rotor_Angle(i) = 280.07;
    Direction(i) = -1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==1 && Digital_Hall_B(i-1)==0 && Digital_Hall_A(i)==1 &&
Digital_Hall_B(i)==1)
    Rotor_Angle(i) = 10.07;
    Direction(i) = 1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==1 && Digital_Hall_B(i-1)==1 && Digital_Hall_A(i)==0 &&
Digital_Hall_B(i)==1)
    Rotor_Angle(i) = 100.07;
    Direction(i) = 1;
    State_Change = TRUE;
  elseif (Digital_Hall_A(i-1)==1 && Digital_Hall_B(i-1)==1 && Digital_Hall_A(i)==1 &&
Digital_Hall_B(i)==0)
    Rotor_Angle(i) = 10.07;
    Direction(i) = -1;
    State_Change = TRUE;
  else
    Rotor_Angle(i) = Rotor_Angle(i-1);
    Direction(i) = Direction(i-1);
    State_Change = FALSE;
  endif
  % Noise Filter
  % The intent is to lockout the state from changing rapidly due to noise.
  % A better implementation is to check if the state change is a reversal in "Direction".  If so, 
the state change
  % should be blocked from occurring for a minimum time period.  If the motor is acculately 
reversing direction, it would be
  % a relatively slow change and a minor delay would be very insignificant.
  if State_Change == TRUE;
    % Re-start state change lockout timer
    State_Change_Lockout_Timer = 3;         % Time period yet to be determined.  Added logic for 
functional implementation.
  endif


  % Track State Machine Angle
  % This is merely for reference and not a functional requirement.
  % The state changes are recorded to later plot for visual reference.
  if State_Change == TRUE;
    Rotor_Angle_State_Machine(i) = Rotor_Angle(i);
  else
    Rotor_Angle_State_Machine(i) = Rotor_Angle_State_Machine(i-1);
  endif


  %-----------------------------------------------------------------
  % Rotor Update
  % When a quadrature state change occurs, calculate the new angle update rate.
  % When no quadrature update, estimate new angle based on estimated rotor velocity

  if State_Change == TRUE
    % Rotor Angle has already been updated by state machine.
    if(Rotor_Last_Correction_Angle == -1)
      % Update rate cannot be deteremined until second sample
      Rotor_Angle_Velocity = 0;
      Rotor_Angle_Acceleration = 0;
    else
      % Save history
      Rotor_Angle_Velocity_1 = Rotor_Angle_Velocity;       % Used for acceleration calculation
      % Calculate rotation delta since last state change
      if (Direction(i)==1)
        % Forward, Positive Rotation Angle
        if Rotor_Last_Correction_Angle < Rotor_Angle(i)
           dRotor_Angle = Rotor_Angle(i) - Rotor_Last_Correction_Angle;
        else
           dRotor_Angle = Rotor_Angle(i) - (Rotor_Last_Correction_Angle -360);
        end
      else
        % Reverse, Negative Rotation Angle
        if Rotor_Angle(i) < Rotor_Last_Correction_Angle
           dRotor_Angle = Rotor_Angle(i) - Rotor_Last_Correction_Angle;
        else
           dRotor_Angle = (Rotor_Angle(i)-360) - Rotor_Last_Correction_Angle;
        endif
      endif

      % Calculate new rate of angle change
      dRotor_Angle_Period = Time(i) - Rotor_Last_Correction_Time;   % Period difference between 
state machine updates
      Rotor_Angle_Velocity = dRotor_Angle / dRotor_Angle_Period;   % Angle/second
      Rotor_Angle_Acceleration = (Rotor_Angle_Velocity - 
Rotor_Angle_Velocity_1)/dRotor_Angle_Period;  % Angle/second/second
    end
    % Update history for next calculation
    Rotor_Last_Correction_Angle = Rotor_Angle(i);
    Rotor_Last_Correction_Time = Time(i);

  else  % State_Change == FALSE
    % Estimate Rotor Angle Update
    Period = Time(i) - Time(i-1);
    Rotor_Angle(i) = Rotor_Angle(i) + Rotor_Angle_Velocity * Period + Rotor_Angle_Acceleration * 
Period^2 ;
```

```
    % Roll 360 boundary
    if(Rotor_Angle(i) > 360)
      Rotor_Angle(i) = Rotor_Angle(i) - 360;
    elseif (Rotor_Angle(i) < 0)
      Rotor_Angle(i) = Rotor_Angle(i) + 360;
    endif

  endif

  %----------------------------------------------------------------

end


%----------------------------------------------------------------
%----------------------------------------------------------------
% Plot Results

subplot(4,1,1);
plot(Time, Analog_Hall_A,'r'); hold on
plot(Time, Analog_Hall_B),'g';
plot(Time, Hall_Upper_Threshold_A,'-.r')
plot(Time, Hall_Lower_Threshold_A,'-.r')
%plot(Time, Hall_Upper_Threshold_B,'-.g')
%plot(Time, Hall_Lower_Threshold_B,'..g')
plot(Time, Hall_Max_A,'-.k');
plot(Time, Hall_Min_A,'-.k');
axis([min(Time) max(Time) 0 3.1]); grid
title(['Analog Hall Sensor Feedback - ',qd_title]);
legend('Analog Hall A', 'Analog Hall B', 'Upper Thres A','Lower Thres A', 'Max A','Min
A','location','southeast');
xlabel('Time');
ylabel('Voltage (V)');

subplot(4,1,2);
plot(Time, Digital_Hall_A); hold on
plot(Time, Digital_Hall_B)
axis([min(Time) max(Time) -0.1 1.1]); grid
set(gca,'ytick', [0 1]);
title('Quantised Hall Sensor Feedback');
legend('Digital Hall A', 'Digital Hall B','location','northwest');
xlabel('Time');
ylabel('Logic Level');

subplot(4,1,3);
plot(Time, Rotor_Angle_State_Machine); hold on
plot(Time ,Direction.*170+200);
plot(Time, ones(size(Time)).*360,'-.r')
plot(Time, ones(size(Time)).*0,'-.r')
axis([min(Time) max(Time) -10 380]); grid
set(gca,'ytick', [0:20:360]);
title('Quadrature Transistion State Table Output');
legend('Angle', 'Direction','location','northwest');
xlabel('Time');
ylabel('Electrical Rotor Angle (degrees)');

subplot(4,1,4);
plot(Time, ones(size(Time)).*360,'-.r'); hold on;
plot(Time, ones(size(Time)).*0,'-.r')
rotor_angle_analog = zeros(size(Time));
ra = (360/(2*pi())).*acos((2.*(Analog_Hall_A - Hall_Zero_Reference)./(1.1)));
rb = (360/(2*pi())).*asin((2.*(Analog_Hall_B - Hall_Zero_Reference)./(1.1)));
for i=2:size(Time,1)
  if(Analog_Hall_A(i) > Hall_Zero_Reference && Analog_Hall_B(i) > Hall_Zero_Reference)
    rotor_angle_analog(i) = rb(i);
  elseif (Analog_Hall_A(i) > Hall_Zero_Reference && Analog_Hall_B(i) < Hall_Zero_Reference)
    rotor_angle_analog(i) = rb(i)+360;
  elseif (Analog_Hall_A(i) < Hall_Zero_Reference && Analog_Hall_B(i) > Hall_Zero_Reference)
    rotor_angle_analog(i) = ra(i);
  else
    rotor_angle_analog(i) = 360-ra(i);
  endif
```

```
end
plot(Time,rotor_angle_analog,'m-.');
plot(Time, Rotor_Angle);
legend('Angle Generated','Angle (Analog)','location','northwest');
axis([min(Time) max(Time) -10 370]); grid
set(gca,'ytick', [0:20:360]);
title('Rotor Angle');
xlabel('Time');
ylabel('Electrical Rotor Angle (degrees)');


%subplot(4,1,1); v=axis; axis([3.0 3.3 v(3) v(4)])
%subplot(4,1,2); v=axis; axis([3.0 3.3 v(3) v(4)])
%subplot(4,1,3); v=axis; axis([3.0 3.3 v(3) v(4)])
%subplot(4,1,4); v=axis; axis([3.0 3.3 v(3) v(4)])
```

<end of report>