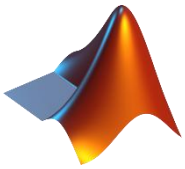
RMXplorer



LAB 2: Actuators

Name

- นายนันทิวัฒน์ จันทรเทพ 67340500054
- นายปกรณ์ บัวงาม 67340500055
- นายสมานสิน เจตนาเจริญชัย 67340500075

Objectives

- เพื่อเรียนรู้การออกแบบการทดลองตามหลักวิทยาศาสตร์ในการสำรวจพฤติกรรมและปรากฏการณ์ของ Actuators ทั้ง 3 ชนิด ได้แก่ DC Motor, Stepper Motor และ BLDC MOTOR
- เพื่อเรียนรู้และทำความเข้าใจหลักการทำงานของ Actuators โดยศึกษากระบวนการทำงานของมอเตอร์
- เพื่อเรียนรู้การประยุกต์ใช้โปรแกรม MATLAB SIMULINK STM32CUBEMX และบอร์ดทดลอง ในการใช้บันทึกข้อมูล และนำมาวิเคราะห์ อธิบาย และสรุปผลการทดลอง
- เพื่อเรียนรู้การออกแบบการทดลอง โดยการตั้งตัวแปร สมมติฐาน และการอ้างอิงจากทฤษฎี ให้สอดคล้องกับการทดลอง
- เพื่อเรียนรู้การเขียนรายงานทางวิทยาศาสตร์

Lab 2.1 DC Motor

การทดลองที่ 1 การทดลองผลของ Motor Frequency ที่ส่งผลต่อการควบคุม Motor

จุดประสงค์

1. เพื่อศึกษาผลกระทบของ Motor Frequency เมื่อ Motor Frequency เปลี่ยนไป

สมมติฐาน

เมื่อความถี่ของสัญญาณ PWM สูงขึ้น มอเตอร์จะสูญเสียประสิทธิภาพมากขึ้นในช่วง Duty Cycle ต่ำ

ตัวแปร

1. ตัวแปรต้น :
   - Motor Frequency
2. ตัวแปรตาม :
   - Motor Speed (No load and Stall load)
   - Motor Current (No load and Stall load)
   - Stall Torque
3. ตัวแปรควบคุม :
   - Ramping Motor PWM
   - ไฟเลี้ยงมอเตอร์ 12V
   - NUCLEO G474RE
   - Incremental Encoder AMT103-V
   - WCS1600 Hall Current Sensor
   - MD20A DC Motor Driver

เอกสารและงานวิจัยที่เกี่ยวข้อง

Pulse Width Modulation หรือ PWM เป็นวิธีการควบคุมปริมาณแรงดันไฟฟ้าที่จ่ายให้กับอุปกรณ์ โดยการสลับระหว่างการเปิดและปิด แหล่งจ่ายไฟด้วยความถี่สูง แทนการลดระดับแรงดันไฟฟ้า ซึ่งช่วยลดการสูญเสียพลังงานในรูปความร้อนได้อย่างมีประสิทธิภาพ หลักการทำงานสำคัญอยู่ที่ค่า Duty Cycle ซึ่งหมายถึงสัดส่วนของเวลาที่เปิด ต่อเวลาทั้งหมดในหนึ่งคาบเวลา โดยค่าแรงดันไฟฟ้าเฉลี่ยที่มอเตอร์ได้รับจะแปรผันตรงกับค่า Duty Cycle ตามสมการ:

$$D = \frac{T_{on}}{Period} \times 100\%$$

$$V_{avg} = \frac{D}{100} \times V_{max}$$

โดยที่:

D	คือ	Duty Cycle	$[\%]$
T_{on}	คือ	ระยะเวลาที่สัญญาณเปิด	$[s]$
$Period$	คือ	เวลารวมที่ใช้ในหนึ่งคาบเวลา	$[s]$
V_{avg}	คือ	แรงดันไฟฟ้าเฉลี่ยของสัญญาณ	$[V]$
V_{max}	คือ	แรงดันไฟฟ้าสูงสุดของสัญญาณ	$[V]$

วิธีการคำนวณประสิทธิภาพ Motor

ประสิทธิภาพของมอเตอร์ (Motor Efficiency) คือ อัตราส่วนระหว่างกำลังงานทางกลที่มอเตอร์สามารถส่งออกได้ เทียบกับกำลังงานไฟฟ้าที่จ่ายเข้าไปในมอเตอร์ โดยสามารถคำนวณได้จากสมการพื้นฐานดังนี้

$$\eta = \frac{P_{out}}{P_{in}} \times 100\%$$

โดยที่:

η	คือ	ประสิทธิภาพของมอเตอร์	$[\%]$
P_{out}	คือ	กำลังงานขาออกของมอเตอร์	$[W]$
P_{in}	คือ	กำลังงานขาเข้าของมอเตอร์	$[W]$

หลักการทำงานของ DC Motor

มอเตอร์กระแสตรง (DC Motor) เป็นอุปกรณ์ที่ทำหน้าที่แปลงพลังงานไฟฟ้ากระแสตรงให้เป็นพลังงานกล โดยอาศัยหลักการแรงแม่เหล็กไฟฟ้า เมื่อมีกระแสไฟฟ้าไหลผ่านตัวนำซึ่งอยู่ในสนามแม่เหล็ก จะเกิดแรงกระทำต่อตัวนำ ทำให้เกิดการหมุนของโรเตอร์ภายในมอเตอร์ โดยทิศทางการเกิดแรงสามารถอธิบายได้ด้วยกฎมือซ้ายของเฟลมมิง ซึ่งมอเตอร์กระแสตรงประกอบด้วยขดลวด สนามแม่เหล็ก และแปรงถ่าน เมื่อจ่ายแรงดันไฟฟ้ากระแสตรงเข้าสู่มอเตอร์ กระแสจะไหลผ่านขดลวด ทำให้เกิดแรงบิดและทำให้เพลามอเตอร์หมุนอย่างต่อเนื่อง แรงบิดที่เกิดขึ้นจะแปรผันตรงกับฟลักซ์แม่เหล็กและกระแส ตามสมการ

$$T = k\phi I_a$$

และความเร็วรอบของมอเตอร์มีความสัมพันธ์กับแรงดันไฟฟ้าและฟลักซ์แม่เหล็กตามสมการ

$$N = \frac{V - I_a R_a}{k\phi}$$

โดยที่:

T	คือ	แรงบิดของมอเตอร์	$[N \cdot m]$
k	คือ	ค่าคงที่ของมอเตอร์	
ϕ	คือ	ฟลักซ์แม่เหล็ก	$[Wb]$
I_a	คือ	กระแสไฟฟ้า	$[A]$
N	คือ	ความเร็วรอบของมอเตอร์	$[Round/Min]$
V	คือ	แรงดันไฟฟ้าที่จ่ายให้มอเตอร์	$[V]$
R_a	คือ	ความต้านทานของขดลวด	$[\Omega]$

สมการ Motor Characteristic Curve

Motor Characteristic Curve แสดงความสัมพันธ์ระหว่าง ความเร็วรอบ และ แรงบิด ของมอเตอร์ DC โดยทั่วไปมอเตอร์แบบ Shunt หรือ Constant Flux Motor จะมีความสัมพันธ์พื้นฐานดังนี้:

- ความเร็วรอบของมอเตอร์ลดลงเมื่อแรงบิดเพิ่มขึ้น (ความสัมพันธ์เชิงเส้นแบบลบ)
- ความเร็วรอบแปรผันตามแรงดันที่ป้อนให้มอเตอร์
- แรงบิดแปรผันตรงกับกระแสแขนงอาร์เมเจอร์ I_a

สมการพื้นฐานของมอเตอร์ DC คือ:

$$N \approx \frac{V - I_a R_a}{k\phi}$$

$$T = k\phi I_a$$

มอเตอร์ DC สามารถคำนวณกำลังได้สองแบบคือ กำลังไฟฟ้า และ กำลังทางกล

กำลังไฟฟ้าพิจารณาจากแรงดันและกระแสที่ป้อนมอเตอร์:

$$P_{electrical} = VI$$

โดยที่:

V	คือ	แรงดันไฟฟ้าขาเข้า	$[V]$
I	คือ	กระแสที่มอเตอร์ดึง	$[A]$

กำลังทางกลพิจารณาจากแรงบิดและความเร็วเชิงมุม:

$$P_{mechanical} = T\omega$$

โดยที่:

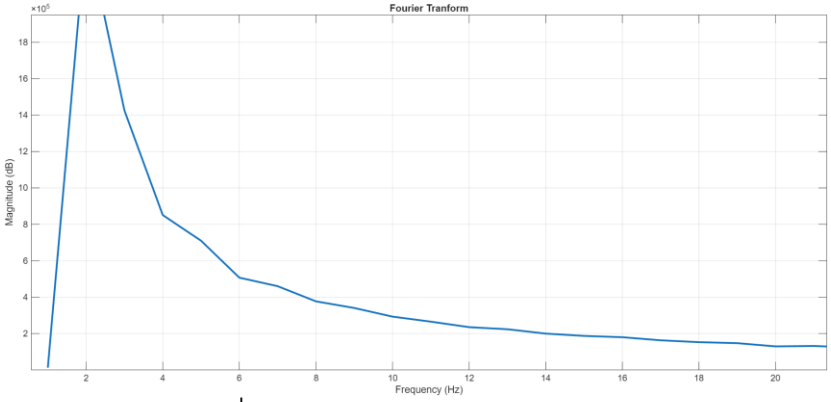
T	คือ	แรงบิดของมอเตอร์	$[N \cdot m]$
ω	คือ	ความเร็วเชิงมุม	$[rad/s]$

ความสัมพันธ์ต่อประสิทธิภาพ

$$\eta = \frac{P_{mechanical}}{P_{electrical}}$$

ขั้นตอนการดำเนินงาน

1. ทำการ Signal Conditioning ของอุปกรณ์ทดลองได้แก่ Load Cell, Encoder, Hall Current Sensor
2. ปรับค่า gain ของ load cell ให้เหมาะสมกับการทดลอง โดยที่ Motor จะมีแรงดึงอยู่ที่ประมาณ 200 กรัม ดังนั้น range ที่ load cell อ่านได้ควรปรับอยู่ในช่วง 0-300 กรัม จากนั้นทำการเก็บตัวอย่างข้อมูลในแต่ละน้ำหนัก เพื่อสร้างสมการของ load cell
3. Encoder ในสัญญาณเริ่มต้นของ Encoder จะมีความไม่เสถียรเป็นอย่างมากดังรูปที่... โดยวิธีที่จะใช้ในการจัดการข้อมูลนี้คือ การทำ low pass filter ผ่าน function filtfilt() ใน Matlab โดยจะมีวิธีการทำดังนี้
4. ขั้นตอนแรกให้นำค่าสัญญาณที่อ่านได้ผ่าน fourier transform โดยจะได้ค่าสัญญาณออกมาดังรูปที่ 1 โดยจากค่าสัญญาณจะพบว่า Cut-Off Frequency ที่เหมาะสมอยู่ที่ประมาณ 10 Hz



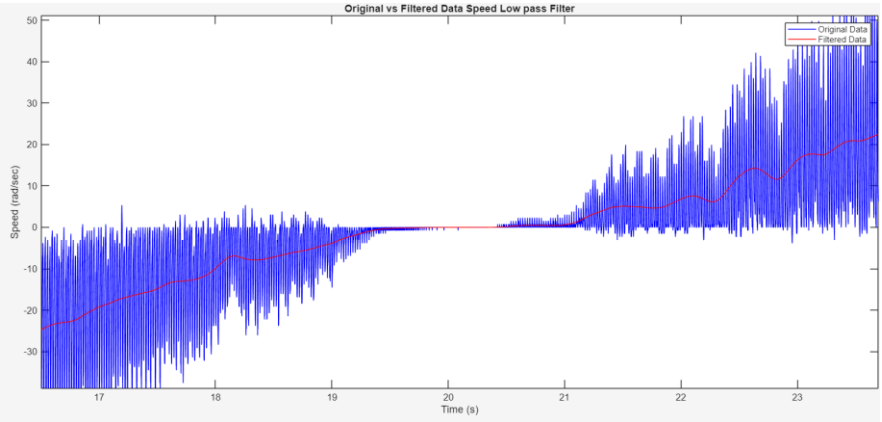
รูปที่ 1 รูปแสดง ผลของ Fourier Tranform

5. ให้นำค่าที่อ่านได้ลง Function filtfilt() เพื่อทำการกรองสัญญาณโดยที่ตั้งค่าดังภาพ

```
[b,a] = butter(2,fc/fs/2,"low");
data_current_filt = filtfilt(b,a,data_current);
```

รูปที่ 2 รูปแสดงคำสั่งสำหรับการกรองสัญญาณ

โดยผลของการกรองสัญญาณจะได้รูปดังนี้ โดยที่สีแดงคือ สัญญาณที่กรองแล้ว และสีน้ำเงินคือสัญญาณที่อ่านค่าได้จาก Encoder



รูปที่ 3 รูปแสดงสัญญาณก่อน และหลังผ่านการกรองสัญญาณ

6. Signal Conditioning ของ Current Sensor เนื่องจาก Current Sensor ใช้หลักการเดียวกับ Hall Sensor ดังนั้นวิธีการอ่านค่าสัญญานคือนำค่าที่อ่านได้ที่เป็น digital แปลงเป็น Voltage จากนั้น ลบกับ Voltage ที่ 0 A จากนั้น คูณด้วย Sensitivityจะได้ออกมาเป็นกระแสไฟฟ้าที่อ่านได้ ดังสมการต่อไปนี้

$$I_{current} = (V_{signal} - V_{offset}) * Sensitivity$$

โดยการหาค่า Sensitivity ของ Hall sensor ให้ทำการใช้ Multimeter ในการอ่านสัญญานกระแสจากนั้นทำการหา Sensitivity ที่ตรวกับค่าที่ multimeter อ่านได้ สุดท้ายจะได้ Sensitivity อยู่ที่ 27 mV/A

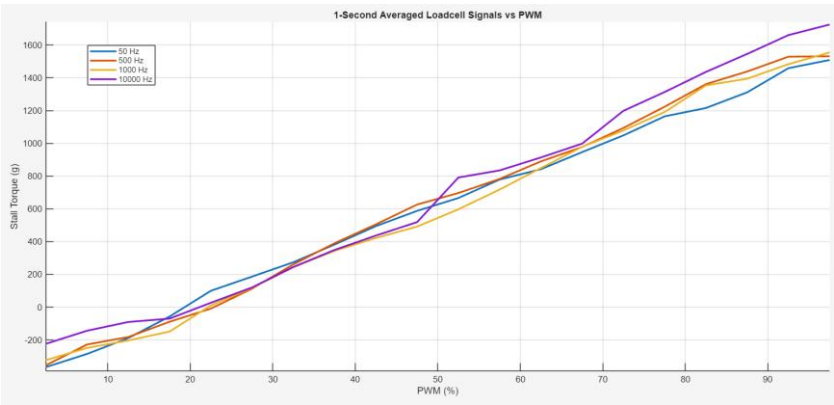
7. ทำการเขียน Code Control Motor โดยที่ Code จะสั่งให้ Motor ทำงานตั้งแต่ PWM -100% ถึง PWM 100% โดยที่มี Ramping ที่ละ +5%/sec
8. ถอดไม้ตีของบอร์ดทดลองโดยที่ให้เก็บค่าสัญญาน เมื่อ Motor นั้น No Load ที่ Motor Frequency 50 Hz, 100 Hz, 1,000 Hz และ 10,000 Hz โดยเก็บผลการทดลอง Motor Frequency ละ 3 ครั้ง

ติดตั้งไม้ตีและติดตั้ง load cell จากนั้นทำการเก็บผลการทดลองดังข้อที่ 3

ในการคำนวณ Stall torque ให้นำค่าที่อ่านได้จาก Load Cell คูณด้วยความยาวของไม้ที่ประมาณ 11.12 เซนติเมตร

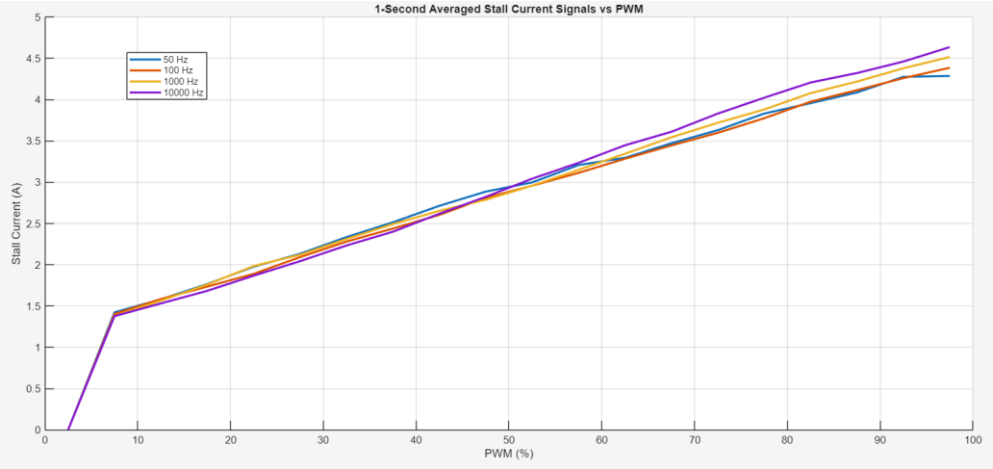
ผลการทดลอง

Stall Torque



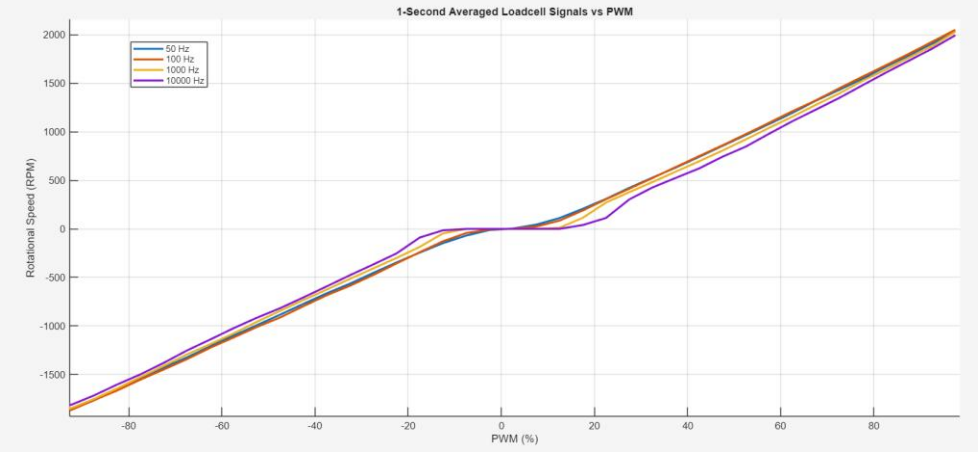
รูปที่ 4 รูปแสดงสัญญาณของ Stall torque เปรียบเทียบกับ PWM

Stall Current

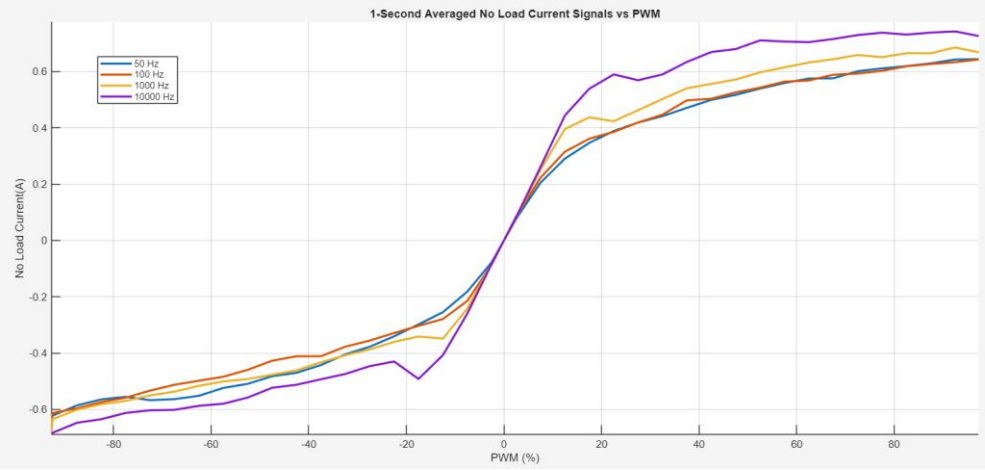


รูปที่ 5 รูปแสดงสัญญาณของ Stall current เปรียบเทียบกับ PWM

RPM

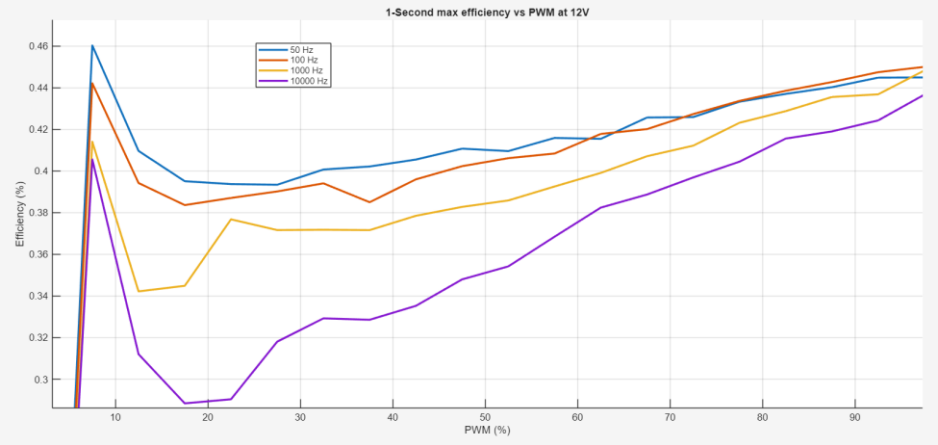


รูปที่ 6 รูปแสดงสัญญาณของ RPM เปรียบเทียบกับ PWM



รูปที่ 7 รูปแสดงสัญญาณของ No load current เปรียบเทียบกับ PWM

ประสิทธิภาพ



รูปที่ 8 รูปแสดงสัญญาณของ Efficiency เปรียบเทียบกับ PWM ที่แรงดันไฟ 12V

สรุปผลการทดลอง

จากการทดลองพบว่า ในการควบคุม PWM Frequency ของ Motor จะมีผลอย่างเห็นได้ชัดเจนใน 2 ประเด็นคือ 1. ช่วง Deadtime ของ Motor 2. Efficiency ของ Motor โดย

1. ช่วง Deadtime พบว่าเมื่อ Motor เพิ่มความถี่ในการควบคุม ช่วงDeadtime จะมีค่ามากขึ้นเรื่อยๆ

2. พบว่าเมื่อเพิ่ม PWM Frequency จะส่งผลให้ประสิทธิภาพของ Motor ตกลงอย่างรวดเร็วในช่วง PWM ต่ำ และจะเพิ่มขึ้นจนกระทั่งใกล้เคียงกันที่ PWM 100%

อภิปรายผล

Deadtime เมื่อ PWM Frequency สูงขึ้นจะส่งผลให้ Motor มีเวลาในการ Charge กระแสที่น้อยลง ส่งผลให้กำลัง ที่ Motor สามารถจ่ายได้นั้นลดน้อยลง ดังสมการ $P = IV$ ซึ่งสามารถสังเกตได้จากกราฟ No load Current ที่เมื่อ ปรับ PWM Frequency สูงขึ้น Motor กลับพยายามที่จะดึงกระแสมากยิ่งขึ้น ในทางกลับกันความเร็วในการหมุนนั้นกลับไม่เพิ่มขึ้น หมายความว่าประสิทธิ์ภาพการใช้พลังงานของ Motor นั้นต่ำลง ซึ่งส่งผลให้ Power Output ของระบบนั้นไม่เพียงพอที่จะทำให้ Motor เกิดการหมุนได้

แต่ข้อดีของการปรับให้ PWM Frequency สูงคือเมื่อ PWM Frequency นั้นมีค่าที่สูงยิ่งขึ้นจะทำให้เสียงขณะทำงานของ Motor มีเสียงที่เบาลงอย่างเห็นได้ชัดโดยในช่วงที่ Frequency ประมาณ 50-100 Hz Motor จะมีเสียงหวีดแหลมออกมา ซึ่งสามารถก่อความรบกวนได้

ข้อเสนอแนะ

-

อ้างอิง

https://www.circuitbread.com/ee-faq/what-is-a-pwm-signal?.com

https://www.engineeringtoolbox.com/electrical-motor-efficiency-d_655.html?.com

https://ecampusontario.pressbooks.pub/electrotechnology/chapter/part-5-dc-motor-operation/?.com

https://uomus.edu.iq/img/lectures21/MUCLecture_2024_102231579.pdf?.com

https://www.g-tech.ac.th/vdo/moterdoc/เครื่องกลไฟฟ้ากระแสตรง/บทที่%204%20มอเตอร์ไฟฟ้ากระแสตรง.pdf?.com

การทดลองที่ 2 การทดลองผลของ PWM ที่ส่งผลต่อ Motor

จุดประสงค์

1. เพื่อศึกษาความสัมพันธ์ระหว่างการปรับค่า Duty Cycle ของสัญญาณ PWM กับการเปลี่ยนแปลงของความเร็วรอบมอเตอร์ และวิเคราะห์ผลของแรงดันเฉลี่ยที่เพิ่มขึ้นต่อพฤติกรรมการหมุนของมอเตอร์
2. เพื่อวิเคราะห์การตอบสนองของมอเตอร์เมื่ออยู่ภายใต้ภาระโหลดสูงสุด โดยศึกษาความสัมพันธ์ระหว่างค่า Duty Cycle กับแรงบิดที่มอเตอร์สามารถสร้างได้ รวมถึงปริมาณกระแสที่มอเตอร์ดึง เพื่อประเมินสมรรถนะของมอเตอร์ในสภาวะที่ต้องใช้แรงบิดสูง

สมมติฐาน

เมื่อเพิ่มค่า Duty Cycle ของสัญญาณ PWM แรงดันไฟฟ้าเฉลี่ยที่จ่ายให้มอเตอร์จะเพิ่มขึ้น ส่งผลให้ความเร็วรอบของมอเตอร์เพิ่มขึ้นอย่างมีแนวโน้มเป็นสัดส่วนกับค่า Duty Cycle ภายใต้เงื่อนไขอื่นคงที่

เมื่อมอเตอร์อยู่ในสภาวะภาระโหลดสูงสุด การเพิ่มค่า Duty Cycle ของสัญญาณ PWM จะทำให้แรงบิดที่มอเตอร์สามารถสร้างได้ และกระแสไฟฟ้าที่มอเตอร์ดึง เพิ่มสูงขึ้นตามลำดับ แสดงให้เห็นถึงความสัมพันธ์เชิงเพิ่มขึ้นระหว่าง Duty Cycle กับความสามารถในการสร้างแรงบิดของมอเตอร์

ตัวแปร

1. ตัวแปรต้น :
   - Motor PWM
2. ตัวแปรตาม :
   - Motor Speed (No load and Stall load)
   - Motor Current (No load and Stall load)
   - Stall load
3. ตัวแปรควบคุม :
   - ตั้ง Motor Frequency 1000Hz
   - ไฟเลี้ยงมอเตอร์ 12V
   - Incremental Encoder AMT103-V
   - WCS1600 Hall Current Sensor
   - MD20A DC Motor Driver

เอกสารและงานวิจัยที่เกี่ยวข้อง

การคำนวณ Motor Characteristic ในการคำนวณกราฟ Motor Characteristic อย่างง่ายสามารถฟาได้ด้วยค่าทั้งหมด 4 อย่างประกอบไปด้วย 1.No Load Speed 2.No Load Current 3.Stall Current 4.Stall Torque โดยแต่ละค่าจะถูกนำไปคำนวนสมการ Characteristic ประกอบไปด้วย 1.Current เทียบ Torque 2. RPM เทียบ Torque 3. Power Output 4. Efficiency โดยที่จะสามารถคำนวณค่าได้ดังนี้

1. Current หาได้จากการนำ No Load Current และ stall Current มาลากเป็นสมการเส้นตรง
2. RPMหาได้จาก No Load Speed และ ความเร็วในการหมุนที่ Stall Torque มาลากเป็นสมการเส้นตรง
3. Power สามารถหาได้จาก $T \cdot N(rad/s)$
4. Efficiency สามารถหาได้จาก Pout/Pin*100 $\frac{P_{out}}{P_{in}} \times 100\%$

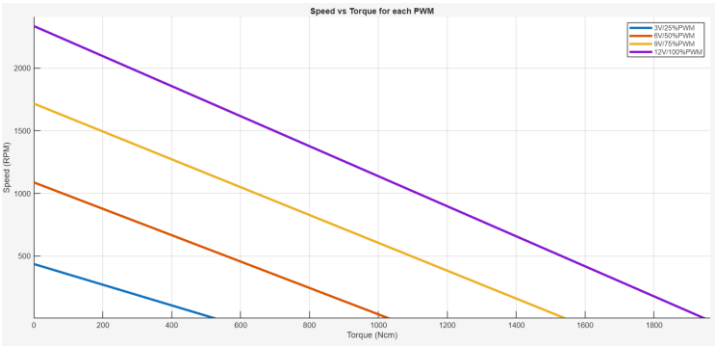
Motor Constant คือค่าคงที่ระหว่างการเปลี่ยน Torque และกระแสที่จ่ายให้กับ Motor ดังสมการ $\tau = K_m i$ โดยที่สามารถหาได้จาก $\frac{R\tau_{st}}{V_{in}} = K_m$ และ *R* หรือความต้านทานของ *Motor* สามารถหาได้จาก

$$R = \frac{V_{in}}{i_{st}}$$

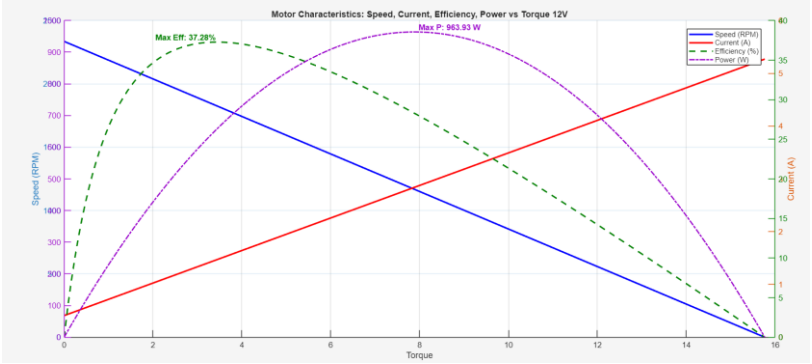
ขั้นตอนการดำเนินงาน

1. ทำ Signal Conditioning ของ Sensor ทุกตัว
2. ตั้ง PWM Frequency ไว้ที่ 1000 Hz และถอดไม้ตีของชุดการทดลอง
3. เปิด Motor ให้หมุนเป็นเวลา 5 วินาที โดยที่ตั้ง PWM ไว้ที่ 25% ทำซ้ำจำนวน 3 ครั้ง
4. ทำซ้ำข้อ 3 โดยเปลี่ยน PWM เป็น 50%, 75%, 100% ตามลำดับ
5. ติดตั้งไม้ตีและ Load cell ของชุดการทดลอง จากนั้นทำซ้ำข้อ 3,4

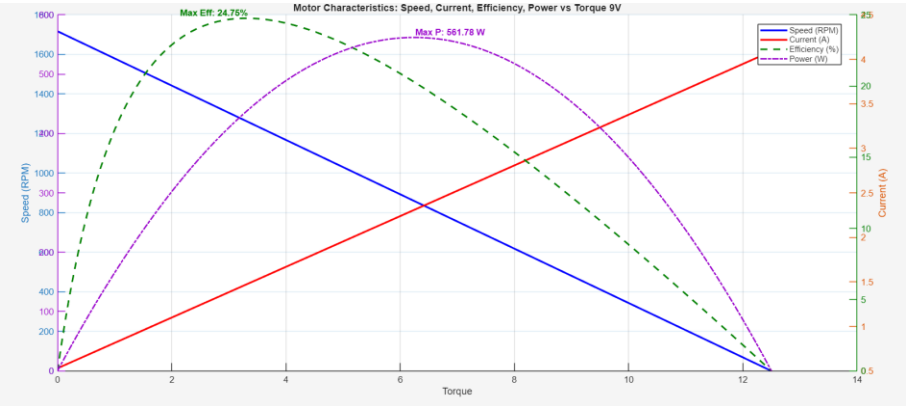
ผลการทดลอง



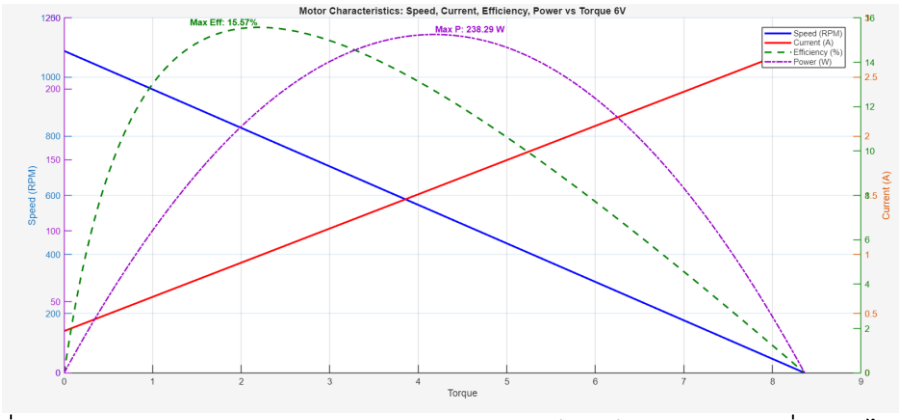
รูปที่ 9 รูปแสดงสัญญาณของ RPM เปรียบเทียบกับ Torque



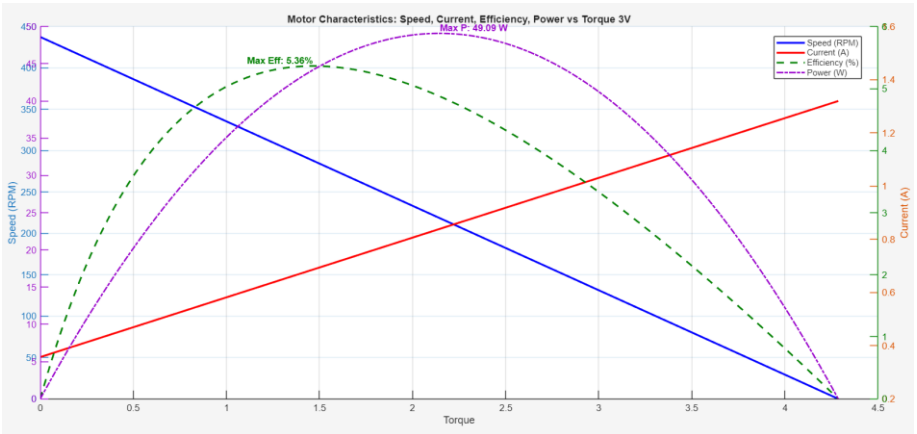
รูปที่ 10 รูปแสดงสัญญาณของ Motor Characteristics เปรียบเทียบกับ Torque ที่แรงดันไฟ 12V



รูปที่ 11 รูปแสดงสัญญาณของ Motor Characteristics เปรียบเทียบกับ Torque ที่แรงดันไฟ 9V



รูปที่ 12 รูปแสดงสัญญาณของ Motor Characteristics เปรียบเทียบกับ Torque ที่แรงดันไฟ 6V



รูปที่ 13 รูปแสดงสัญญาณของ Motor Characteristics เปรียบเทียบกับ Torque ที่แรงดันไฟ 3V

สรุปผลการทดลอง

จากการทดลองพบว่าเมื่อปรับค่า PWM]fลงจะทำให้ Motor ทำงานมีประสิทธิภาพที่ต่ำลงซึ่งรวมถึง Power ที่ Motor สามารถก็ลดลงเช่นเดียวกัน

อภิปรายผล

จากการทดลองสามารถคำนวณได้ว่า Motor มีค่า R หรือค่าความต้านทานภายในอยู่ที่ R=5.1/12 =0.425 โอห์ม โดยสามารถคำนวณ $\frac{R\tau_{st}}{V_{in}} = K_m$ ซึ่งจะได้ค่า Km คือ 0.531

ข้อเสนอแนะ

ทั้งนี้ค่า R สามารถเปลี่ยนได้จากหลายปัจจัยแต่ปัจจัยที่ทำให้เปลี่ยนค่าได้ง่ายมากที่สุดคือ ค่าความร้อนของ Motor ซึ่งสามารถเกิดได้ง่ายเมื่อ Motor เริ่มทำงาน หรือขณะทำการทดลองวัดค่า Stall Torque

อ้างอิง

https://solution.mabuchi-motor.com/blog/en/motor-performance-curves?.com

การทดลองที่ 3 การทดลองความแตกต่างระหว่าง Motor driver Mode Lock Anti-Phase และ Sign-Magnitude

จุดประสงค์

1. เพื่อศึกษาและเปรียบเทียบ ความสัมพันธ์ระหว่างค่า Duty Cycle กับความเร็วรอบของมอเตอร์ ระหว่างการควบคุมแบบ Sign-Magnitude และ Locked Anti-Phase
2. เพื่อวิเคราะห์ ความแตกต่างของการใช้กระแสไฟฟ้า โดยเฉพาะในช่วงจุดหยุดนิ่ง และช่วงที่มีภาระโหลด

สมมติฐาน

โหมด Locked Anti-Phase มีความเป็นเชิงเส้นดีกว่าและไม่มีจุดบอดในช่วงความเร็วต่ำ เมื่อเทียบกับ Sign-Magnitude

ขณะสั่งหยุดนิ่ง โหมด Locked Anti-Phase จะยังคงมีกระแสไฟฟ้าไหลผ่านและเกิดความร้อน สูงกว่าโหมด Sign-Magnitude ที่ไม่มีกระแสไหล

ที่ระดับ Duty Cycle สูงสุด ทั้งสองโหมดจะให้ค่าความเร็วรอบและแรงบิดที่เท่ากัน

ตัวแปร

1. ตัวแปรต้น:
   - Motor PWM
2. ตัวแปรตาม:
   - Motor Speed (No load and Stall load)
   - Motor Current (No load and Stall load)
   - Stall load
3. ตัวแปรควบคุม:
   - ตั้ง Motor driver Mode เป็น Lock Anti-Phase
   - ตั้ง Motor Frequency 1000Hz
   - ไฟเลี้ยงมอเตอร์ 12V
   - NUCLEO G474RE
   - Incremental Encoder AMT103-V
   - WCS1600 Hall Current Sensor
   - MD20A DC Motor Driver

เอกสารและงานวิจัยที่เกี่ยวข้อง

การควบคุมมอเตอร์แบบ Sign-Magnitude และ Locked Anti-Phase มีความแตกต่างสำคัญที่มีผลต่อความเร็วและการใช้กระแสไฟฟ้า โดย Sign-Magnitude ใช้สัญญาณ PWM เพื่อกำหนดขนาดแรงดันเฉลี่ย และ

ใช้สัญญาณทิศทางแยกต่างหาก เมื่อ PWM ต่ำหรือเป็นศูนย์ มอเตอร์จะหยุดและแทบไม่มีกระแสไหล ทำให้สูญเสียพลังงานน้อย เหมาะกับกรณีที่ต้องการให้มอเตอร์หยุดนิ่งจริง ขณะที่ Locked Anti-Phase ใช้ PWM เพียงเส้นเดียวในการควบคุมทั้งทิศทางและแรงดัน โดยจุด duty cycle ที่ 50% คือสถานะหยุด มอเตอร์แม้จะหยุดแต่ยังมีการสลับขั้วภายในตลอดเวลา ทำให้มี ripple current อยู่บ้าง ส่งผลให้มีการใช้กระแสมากกว่าในช่วงหยุดนิ่ง เมื่อวิเคราะห์ความเร็ว พบว่า Sign-Magnitude มีความสัมพันธ์ระหว่าง duty กับความเร็วแบบตรงไปตรงมา ส่วน Locked Anti-Phase duty cycle จะกำหนดทั้งทิศทางและขนาด ทำให้ความสัมพันธ์เป็นแบบสมมาตรรอบ 50% ความแตกต่างเหล่านี้ทำให้ Sign-Magnitude ประหยัดพลังงานกว่าในช่วงไม่มีโหลด ขณะที่ Locked Anti-Phase ให้การควบคุมที่ต่อเนื่องและตอบสนองเร็วกว่า แต่มีการสูญเสียด้านกระแสสูงกว่าเล็กน้อย

ขั้นตอนการดำเนินงาน

1. ทำ signal Conditioning ตามการทดลองที่ผ่านมา
2. เก็บผลการทดลอง โดยตั้ง Stepper Motor โดยการย้ายสาย PWM ไปที่ขา DIR และนำ 3.3V เข้าค่า PWM
3. เก็บผลการทดลองโดยการ ตั้ง PWM ไปที่ 0% จากนั้นถอด ไม้ตีของชุดการทดลอง เพื่อทดลองเก็บค่า No Load จากนั้นทำซ้ำและบันทึกผล Avg Speed Avg No Load Current ที่วัดได้ ทำซ้ำจำนวน 3 ครั้ง
4. ทำซ้ำข้อ 3 โดยเปลี่ยน PWM เป็น 12.5%,25%,37.5%... 87.5%,100%
5. ติดตั้งไม้ตีและทำซ้ำข้อ 3 และ 4 เพื่อเก็บค่า Stall Current, Stall Torque

ผลการทดลอง

signed magnitude		No Load Speed rad/sec				No Load Speed(rpm)			
PWM	Voltage	1	2	3	MEAN	1	2	3	MEAN
0	0	0	0	0	0	0	0	0	0
12.5	1.5	17.97	stall	stall	17.97	171.6008596	#VALUE!	#VALUE!	#VALUE!
25	3	44.03	46.61	46.78	45.80666667	420.4555287	445.0927139	446.7160943	437.4214456
37.5	4.5	79.23	81.44	81.64	80.77	756.5907685	777.6947139	779.6045732	771.2966852
50	6	112.7	114.51	114.71	113.9733333	1076.205725	1093.489952	1095.399811	1088.365163
62.5	7.5	145.81	147.29	147.61	146.9033333	1392.382935	1406.515894	1409.571669	1402.823499
75	9	178.17	180.48	180.67	179.7733333	1701.398173	1723.457048	1725.271414	1716.708878
87.5	10.5	210.18	213.49	212.84	212.17	2007.071156	2038.679328	2032.472285	2026.074257
100	12	242.59	245.47	245.57	244.5433333	2316.563859	2344.065833	2345.020763	2335.216818
locked antiphase		No Load Speed rad/sec				No Load Speed(rpm)			
PWM	Voltage	1	2	3	MEAN	1	2	3	MEAN
	0	242.41	242.88	243.28	242.8566667	2314.844985	2319.333155	2323.152873	2319.110338
	12.5	177.08	177.19	177.27	177.18	1690.989439	1692.039862	1692.803806	1691.944369
	25	114.78	112.19	112.54	113.17	1096.068262	1071.335584	1074.677838	1080.693895
	37.5	49.17	46.72	47.59	47.82666667	469.5389131	446.1431365	454.4510245	456.7110247
	50	0	0	0	0	0	0	0	0
	62.5	-34.46	-35.43	-35.7	-35.19666667	-329.0687603	-338.331578	-340.9096881	-336.1034088
	75	-100.33	-103.1	-104.39	-102.6066667	-958.0809264	-984.532478	-996.8510706	-979.8214917
	87.5	-166.64	-168.61	-168.87	-168.04	-1591.294783	-1610.106897	-1612.589714	-1604.663798
	100	-230.49	-233.71	-234.02	-232.74	-2201.01737	-2231.766105	-2234.726387	-2222.503287

No load Current				Stall Load				Stall Current(A)			
1	2	3	MEAN	1	2	3	MEAN	1	2	3	avg
0	0	0	0	0	0	0	0	0	0	0	0
0.26	stall	stall	0.26	19	18	19	18.66666667	0.45	0.45	0.45	0.45
0.37	0.37	0.33	0.3566666667	47	48	48	47.66666667	1.3	1.34	1.33	1.323333333
0.34	0.33	0.33	0.3333333333	76	76	72	74.66666667	2.03	2.1	2.07	2.066666667
0.35	0.36	0.35	0.3533333333	95	93	91	93	2.78	2.78	2.74	2.766666667
0.36	0.37	0.36	0.3633333333	116	114	113	114.3333333	3.48	3.45	3.42	3.45
0.37	0.35	0.34	0.3533333333	142	137	138	139	4.12	4.07	4.07	4.086666667
0.43	0.4	0.4	0.41	161	162	159	160.6666667	4.71	4.7	4.63	4.68
0.4	0.4	0.43	0.41	176	177	171	175.3333333	5.32	5.31	5.19	5.273333333
No Load Current				Stall Load				Stall Current(A)			
1	2	3	MEAN	1	2	3	MEAN	1	2	3	avg
0.43	0.41	0.41	0.4166666667	198	192	192	194	5.73	5.6	5.65	5.66
0.39	0.39	0.37	0.3833333333	147	143	144	144.6666667	4.07	4.12	4.13	4.106666667
0.34	0.35	0.35	0.3466666667	96	97	96	96.33333333	2.69	2.75	2.69	2.71
0.28	0.3	0.29	0.29	48	45	48	47	1.28	1.23	1.24	1.25
-0.11	-0.12	-0.13	-0.12	0	0	0	0	0.09	0.09	0.11	0.09666666667
-0.63	-0.65	-0.65	-0.6433333333	#	#	#	#	#	#	#	#
-0.65	-0.64	-0.61	-0.6333333333	#	#	#	#	#	#	#	#
-0.65	-0.63	-0.61	-0.63	#	#	#	#	#	#	#	#
-0.67	-0.64	-0.64	-0.65	#	#	#	#	#	#	#	#

สรุปผลการทดลอง

จากการทดลองแสดงให้เห็นว่า Motor ทั้งสองโหมดให้ลักษณะการทำงานออกมาใกล้เคียงกันแต่ว่า แบบ lock anti phase สามารถทำงานได้ที่PWM ต่ำกว่า

อภิปรายผล

ข้อดีของ Lock Anti Phase

- ใช้สายไฟในการ Control น้อยกว่า แค่ PWM ในการควบคุม
- สามารถควบคุมได้ละเอียดกว่าการควบคุม PWM ปกติ

ข้อเสีย

- สูญเสียพลังงานมากเนื่องจาก เป็นการ OPEN ตลอดเวลาซึ่งจะทำให้ Motor สูญเสียพลังงานตลอดเวลา

ข้อเสนอแนะ

อ้างอิง (ใส่แค่ Link)

https://www.pcb-3d.com/wordpress/tutorials/what-is-a-h-bridge-sign-magnitude-and-locked-anti-phase-control-of-a-dc-motor/?.com

Lab 2.2 Stepper Motor

การทดลองที่ 1 การทดลองผลของ ความถี่ของสัญญาณที่ส่งผลต่อการทำงานของ Stepper แต่ละโหมด

จุดประสงค์ (การทดลองนี้สร้างขึ้นเพื่ออะไร)

1. เพื่อทดลองควบคุมการทำงานของ Stepper Motor ในแต่ละโหมด

สมมติฐาน (ข้อสันนิษฐานเบื้องต้น)

ในแต่ละ Mode ของStepper การปรับแบบ Full Step จะใช้ค่า Frequency ในการสั่งการต่ำที่สุด ตามด้วย Half-Step และ Micro-Step

ตัวแปร

1. ตัวแปรต้น:
   - Mode ในการควบคุม Stepper Motor
2. ตัวแปรตาม:
   - ค่า RPM ที่วัดได้
3. ตัวแปรควบคุม:
   - บอร์ดทดลอง
   - การทำ signal Conditioning
   - PWM ที่ใช้ในการทดลอง

เอกสารและงานวิจัยที่เกี่ยวข้อง

หลักการทำงานของ Stepper Motor

หลักการทำงานของ Stepper Motor สามารถอธิบายได้จากการควบคุมสนามแม่เหล็กภายในมอเตอร์ที่เปลี่ยนไปตามลำดับการกระตุ้นขดลวด (Excitation Sequence) ทำให้โรเตอร์เคลื่อนที่ทีละ “สเต็ป” อย่างแม่นยำ การหมุนแบบเป็นขั้นนี้เกิดจากการที่ขดลวดสเตเตอร์ถูกกระตุ้นให้สร้างขั้วแม่เหล็กสลับตำแหน่ง ทำให้โรเตอร์ถูกดึงไปยังตำแหน่งถัดไปของสนามแม่เหล็ก การทำงานลักษณะนี้ทำให้สเต็ปเปอร์มอเตอร์สามารถควบคุมตำแหน่งได้อย่างละเอียดโดยไม่ต้องใช้เซนเซอร์วัดตำแหน่ง ในเอกสารจาก Oriental Motor ได้อธิบายว่ามอเตอร์ชนิด Hybrid Stepper เป็นแบบที่นิยมที่สุด เนื่องจากรวมข้อดีของ Permanent Magnet และ Variable Reluctance Motor เข้าด้วยกัน ทำให้ให้แรงบิดสูงในความเร็วต่ำ และตอบสนองได้ดีเมื่อใช้งานกับการควบคุมแบบ Microstepping นอกจากนี้เอกสารยังชี้ให้เห็นว่าแรงบิดของ Stepper Motor จะลดลงเมื่อความเร็วเพิ่มขึ้น เนื่องจากกระแสไม่สามารถสร้างขึ้นได้เต็มที่ในแต่ละสเต็ป ซึ่งเป็นคุณสมบัติเชิงพลวัตที่สำคัญต่อการออกแบบระบบควบคุม

หลักการขับเคลื่อน Stepper Motor ในแต่ละโหมด

Stepper Motor สามารถขับเคลื่อนได้หลายโหมด โดยหลักการทำงานขึ้นอยู่กับรูปแบบการจ่ายกระแสให้ขดลวดแต่ละเฟสของมอเตอร์

- Wave Drive (One-Phase-On)

กระตุ้นทีละเฟสในแต่ละสเต็ป ทำให้โรเตอร์ขยับตามลำดับสนามแม่เหล็กทีละตำแหน่ง ให้ความร้อนต่ำและใช้พลังงานน้อยที่สุด แต่แรงบิดต่ำกว่าโหมดอื่น

- Full-Step Mode (Two-Phase-On)

กระตุ้นสองเฟสพร้อมกัน ทำให้โรเตอร์อยู่ในตำแหน่งสมดุลระหว่างสองขั้วแม่เหล็ก ให้แรงบิดสูงกว่า Wave Drive แต่ความละเอียดของสเต็ปเท่าเดิม

- Half-Step Mode

สลับการกระตุ้นระหว่างหนึ่งเฟสและสองเฟส ทำให้จำนวนสเต็ปเพิ่มขึ้นเป็นสองเท่า เช่น 1.8° ⟶ 0.9° ให้ความละเอียดมากขึ้น แต่แรงบิดในสเต็ปที่ใช้เฟสเดียวจะลดลง

- Microstepping Mode

ควบคุมกระแสในแต่ละเฟสให้เพิ่ม–ลดแบบต่อเนื่องคล้ายคลื่นไซน์ ทำให้การเคลื่อนที่นุ่มนวลที่สุดและมีความละเอียดสูงสุด แม้แรงบิดสูงสุดจะลดลงเล็กน้อยเมื่อเทียบกับ Full-Step

Table 1. Stepping Format

MODE2	MODE1	MODE0	STEP MODE
0	0	0	Full step (2-phase excitation) with 71% current
0	0	1	1/2 step (1-2 phase excitation)
0	1	0	1/4 step (W1-2 phase excitation)
0	1	1	8 microsteps/step
1	0	0	16 microsteps/step
1	0	1	32 microsteps/step
1	1	0	32 microsteps/step
1	1	1	32 microsteps/step

รูปที่ 14 รูปแสดงตารางการปรับโหมดของ Stepper Moter

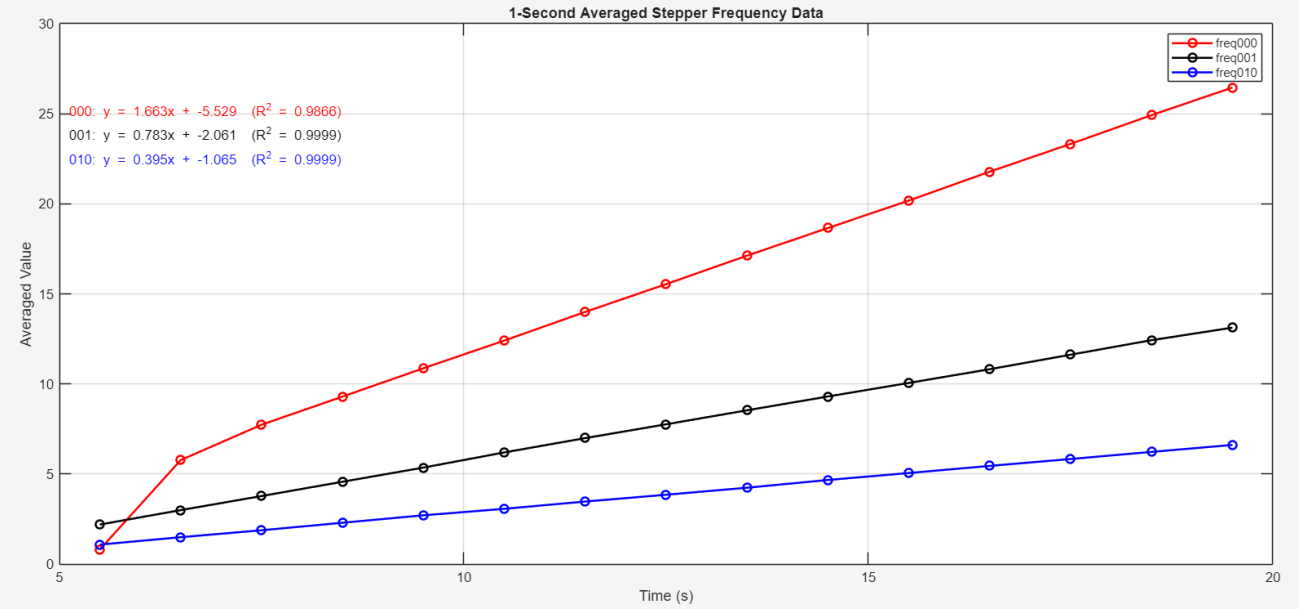
ขั้นตอนการดำเนินงาน

1. ทำ signal Conditioning ของทุกเซ็นเซอร์

2.ทำการเขียนโค้ดโดยจะทำการเก็บค่าจาก เริ่มต้นคือที่ Frequency = 0 Ramp-up ที่ 50 จนกระทั่ง Motor หยุดการทำงาน จากนั้นบันทึกผลการทดลอง และทำซ้ำทั้งหมด 3 ครั้ง โดยในรอบแรกให้ตั้งไปที่ Mode Full Step(M0 =0,M1= 0,M2=0)

3. ทำซ้ำข้อที่ 2 โดยที่ให้เปลี่ยนเป็น Half Step (M0 = 1, M1 = 0, M2 = 0) และ Micro Step ¼ (M0 = 0, M1 = 1, M2 = 0) ตามลำดับ

ผลการทดลอง



รูปที่ 15 รูปแสดงสัญญาณของ Stepper Frequency แต่ละโหมด เปรียบเทียบกับ เวลา

สรุปผลการทดลอง

จากการทดลองพบว่า เมื่อตั้งแบบ Full Step จะใช้ค่า PWM Frequency ต่ำที่สุดในการสั่งงาน เพื่อให้ได้ความเร็วในการหมุนสูง ซึ่งจะลดหลั่นตามลงมาตามแต่ละโหมดโดยที่ micro-Step จะมีค่าต่ำที่สุดซึ่งเป็นไปตามสมมติฐาน

อภิปรายผล

สาเหตุที่ Stepper Motor ยิ่งปรับ Mode ให้เล็กลงต้องอาศัย Stepper Frequency ที่สูงขึ้นเพราะเมื่อ ปรับลงมาเป็น Micro step และ Half Step ในทุกๆครั้งที่ทำการสั่ง PWM แต่สาเหตุที่สมการ R-square ของ Motor ความชันนั้นไม่เป็นไปตามการเปลี่ยนแปลงของ Full Step, Half Step และ Micro-Step ¼ เพราะว่า เกิดการ loss Step ขึ้นระหว่างการทดลองซึ่งส่งผลให้ค่าที่อ่านได้เกิดความคลาดเคลื่อน

ข้อเสนอแนะ

-

อ้างอิง (ใส่แค่ Link)

https://media.monolithicpower.com/mps_cms_document/2/0/2020-stepper-motors-basics-types-uses-and-working-principles_r1.0_1.pdf?.com

การทดลองที่ 2 การทดลองการ Loss Step ของ Stepper Motor

จุดประสงค์

เพื่อศึกษาสาเหตุและปัจจัยที่ทำให้เกิดการ Loss Step ของ Stepper Motor

สมมติฐาน

Stepper Motor จะเกิดการ Loss Stepได้เมื่อสั่ง Stepper Frequency มากเกินไป

ตัวแปร

1. ตัวแปรต้น
   - Stepper Frequency
2. ตัวแปรตาม
   - RPM ที่อ่านได้
   - ค่า Acceleration ที่คำนวณได้
3. ตัวแปรควบคุม
   - ชุดการทดลองที่ใช้ในการทำการทดลอง
   - Stepper Frequency

เอกสารและงานวิจัยที่เกี่ยวข้อง

อาการ Loss Step (การสูญเสียตำแหน่ง) เกิดจากความไม่สัมพันธ์กันระหว่าง สัญญาณไฟฟ้าที่สั่ง กับการเคลื่อนที่เชิงกลจริง เนื่องจาก Stepper Motor ทำงานแบบ Open Loop Control (สั่งงานโดยไม่มีการตรวจสอบกลับ) โดยมีหลักการทางฟิสิกส์ที่เกี่ยวข้องดังนี้

1. ข้อจำกัดด้านแรงบิด (Torque Limitation)

- Pull-out Torque: คือค่าแรงบิดสูงสุดที่มอเตอร์สามารถทำได้ที่ความเร็วรอบหนึ่งๆ หากโหลด (Load) มีแรงต้านมากกว่าค่านี้ สนามแม่เหล็กในขดลวดจะไม่สามารถดึงโรเตอร์ให้หมุนตามทันได้ ทำให้เกิดการ "รูด" หรือข้าม Step

- Torque-Speed Relationship: ธรรมชาติของ Stepper Motor คือ "ยิ่งหมุนเร็ว แรงบิดยิ่งตก" ดังนั้น อาการ Loss Step จึงมักเกิดขึ้นในช่วงที่มอเตอร์ใช้ความเร็วสูง

2. ความเฉื่อยและการเปลี่ยนความเร็ว (Inertia & Dynamics)

- ตามกฎของนิวตัน τ = I x α แรงบิดที่ใช้ในการหมุนขึ้นอยู่กับค่าความเฉื่อยของโหลด I และความเร่ง α
- หากเราเขียนโปรแกรมสั่งให้มอเตอร์ออกตัวกระชาก (High Acceleration) หรือหยุดกะทันหัน แรงบิดที่มอเตอร์มีอาจจะไม่พอที่จะเอาชนะความเฉื่อยของวัตถุ ทำให้โรเตอร์หมุนไม่ไปตามสั่ง หรือหยุดไม่ทันตามตำแหน่งที่ต้องการ

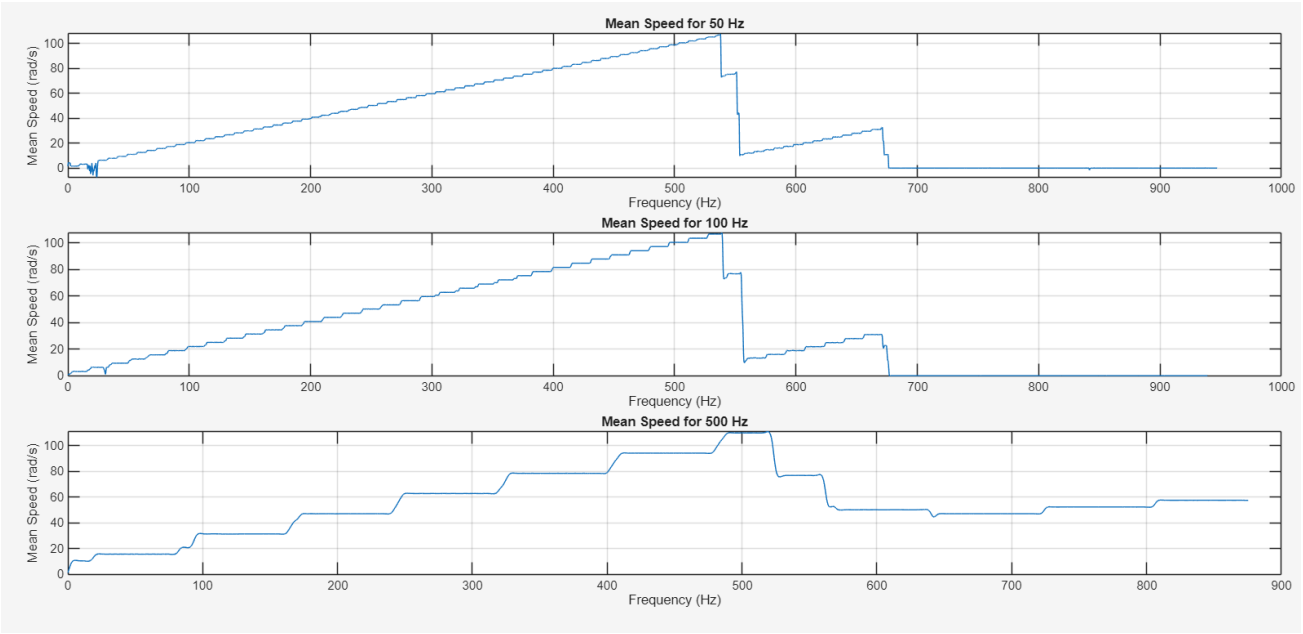
3. การสั่นพ้อง (Resonance)

- Stepper Motor มีการเคลื่อนที่แบบเป็นขั้นๆ (Discrete Steps) ซึ่งสร้างแรงสั่นสะเทือน หากความถี่ในการส่งสัญญาณตรงกับ ความถี่ธรรมชาติ (Natural Frequency) ของระบบกลไก จะเกิดการสั่นรุนแรงจนสนามแม่เหล็กจับโรเตอร์ไม่อยู่ ส่งผลให้เสียตำแหน่ง

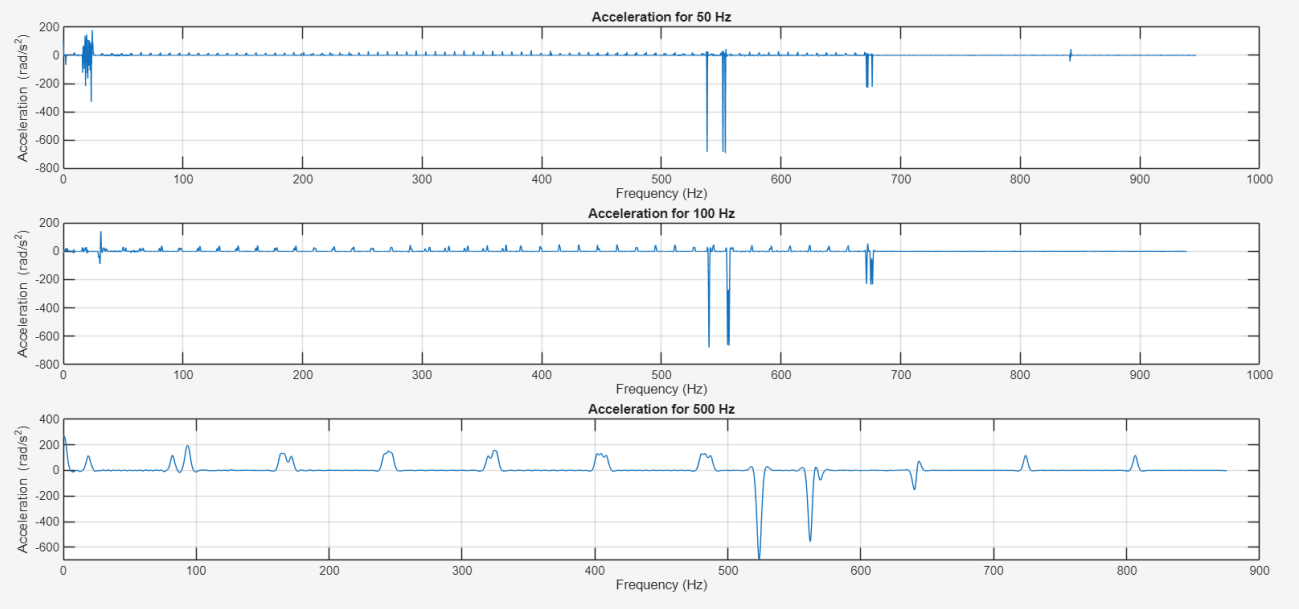
ขั้นตอนการดำเนินงาน

1. PWM ไว้ที่ 100% และตั้งโหมดของ Stepper Motor ไปที่ Full Step
2. ถอดไม้ตีและปลด belt ที่เชื่อมต่อกับ Motor
3. ทำการเก็บผลการทดลองโดยที่ตั้ง Ramping ไปที่ 50 Hz/sec และทำงานไปเรื่อยๆจนกระทั่ง Frequency 1000 Hz โดยทำซ้ำทั้งหมด 3 ครั้ง
4. ทำซ้ำข้อ 3 โดยเปลี่ยน Ramping ไปที่ 100 Hz/sec และที่ 500 Hz/sec

ผลการทดลอง



รูปที่ 16 รูปแสดงสัญญาณของ ความเร็ว ที่แต่ละความถี่



รูปที่ 17 รูปแสดงสัญญาณของ ความเร่ง ที่แต่ละความถี่

สรุปผลการทดลอง

จากการทดลองพบว่า Ramping ของ Motor ส่งผลต่อลักษณะการเกิด Loss Step แตกต่างกัน โดยจะพบว่ายิ่งปรับ Ramping ที่สูงขึ้น การเกิด loss Step ในช่วงเริ่มต้นนั้นจะหายไป และ Stepper Motor จะมักจะเกิดการ loss Step ที่ ประมาณ frequency ที่ 500Hz

อภิปรายผล

ในช่วงต้นที่ Stepper Motor เริ่มหมุน พบว่า มักจะเกิดการ loss Step โดยมักจะเกิดที่ Ramping ต่ำ ๆ ซึ่งสามารถอนุมานได้ว่า เมื่อ ที่ Ramping ต่ำ ๆ ความเร่งของ Motor นั้นมีค่าที่ไม่เพียงพอที่จะMotor จะหลุดจาก Frictionของระบบได้ ซึ่งสามารถตีความได้ว่า Stepper Frequency นั้นส่งผลต่อแรงบิดของระบบเช่นเดียวกัน

ในช่วงที่ Motor เริ่มหมุนไปถึงประมาณ 500-600 Hz ไม่ว่าที่ Ramping เท่าไรก็ตามสามารถตีความได้ว่า Motor นั้นอาจเกิด resonance ได้ซึ่งทำให้ Motor ไม่หมุนตามที่ต้องการหรือแสดงออกมาในรูปที่ว่า ค่า Acceleration จะ drop ลงทันที แต่จะไม่หยุดหมุนยังคงหมุนต่อไปได้

Motor ที่ Full Step จะเริ่มหยุดการทำงานเมื่อ Frequency ขึ้นไปถึงประมาณ 700 Hz ซึ่งโดยสาเหตุเป็นไปได้เกิดจากการที่ เมื่อ Motor หมุนด้วยความเร็วที่มากขึ้น Motor เริ่มสูญเสียแรงบิดซึ่งจะทำให้ Stepper เกิดการ loss Step ได้

ข้อเสนอแนะ

-

อ้างอิง

https://www.orientalmotor.com/stepper-motors/technology/speed-torque-curves-for-stepper-motors.html
https://cdn.faulhaber.com/media/DAM/Documents/Tutorials/faulhaber-tutorial-stepper-motor-step-loss-prevention.pdf
https://www.ti.com/lit/an/slvaff1/slvaff1.pdf

Lab 2.3 Brushless DC Motor

การทดลองที่ 1 การศึกษาและเปรียบเทียบเทคนิคการขับเคลื่อนมอเตอร์ BLDC: 6-StepCommutation และ Field Oriented Control (FOC)

จุดประสงค์ เพื่อศึกษาและเปรียบเทียบหลักการควบคุมมอเตอร์ BLDC

1. เพื่อมุ่งเน้นความเข้าใจในหลักการขับเคลื่อนแบบ 6-Step Commutation และ Field Oriented Control (FOC)
2. เพื่อเปรียบเทียบความแตกต่างระหว่างการระบุตำแหน่งโรเตอร์แบบใช้เซนเซอร์ (Hall Effect Sensor) และแบบไม่ใช้เซนเซอร์ (Back EMF Sensing / Sensor less)

สมมติฐาน

เมื่อทำการปรับค่าความเร็วรอบ (RPM) ในโปรแกรมควบคุม จะส่งผลให้ชุดควบคุมจ่ายสัญญาณแบบ 6-Step Commutation ไปยังมอเตอร์ ทำให้มอเตอร์มีความเร็วเพิ่มขึ้นแบบเชิงเส้น (Linear) และสามารถทำงานได้ต่อเนื่องในช่วงความเร็ว 1 - 10,000 RPM

ตัวแปร

ตัวแปรต้น:

ความเร็วรอบ (RPM) ที่กำหนด

ตัวแปรตาม:

ความถี่ BEMF

รูปคลื่นสัญญาณ

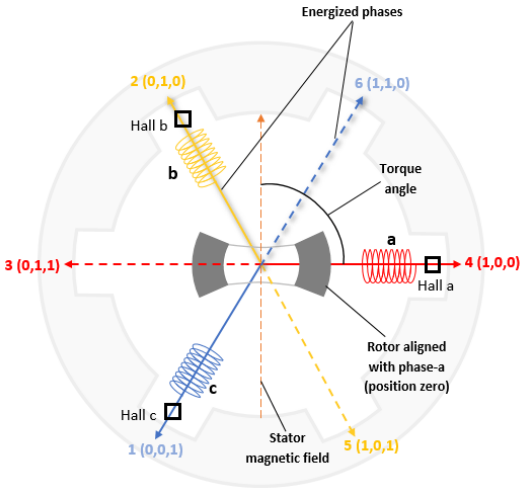
ตัวแปรควบคุม:

แหล่งจ่ายไฟ

โหลดมอเตอร์

เอกสารและงานวิจัยที่เกี่ยวข้อง

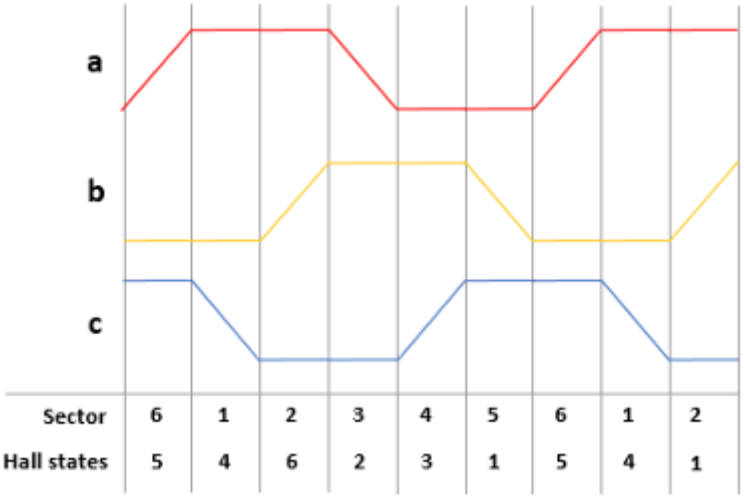
หลักการขับเคลื่อนแบบ 6-Step Commutation (Trapezoidal Control)



รูปที่ 18 โครงสร้างมอเตอร์ BLDC

Hall State (Hall a, Hall b, Hall c)	Switching Sequence (AA' BB' CC')		
	AA'	BB'	CC'
4 (100)	00	10	01
6 (110)	01	10	00
2 (010)	01	00	10
3 (011)	00	01	10
1 (001)	10	01	00
5 (101)	10	00	01

รูปที่ 19 ตาราง logic ในการขับมอเตอร์



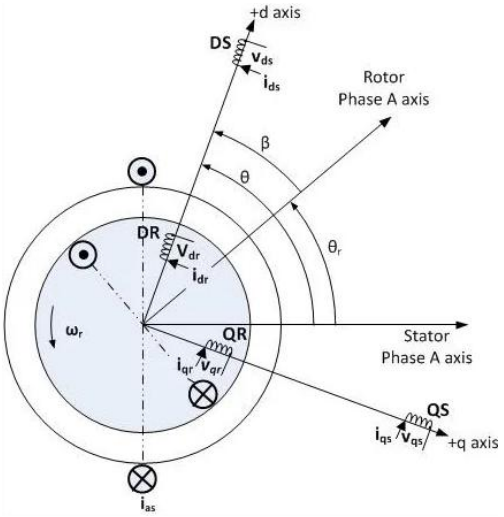
รูปที่ 20 คลื่นสัญญาณจากการควบคุมแบบ 6-Step Commutation

การควบคุมแบบ 6-Step หรือ Trapezoidal Control เป็นเทคนิคพื้นฐานในการขับเคลื่อนมอเตอร์ BLDC โดยมีหลักการทำงานคือ การจ่ายกระแสไฟฟ้าเข้าสู่ขดลวดสเตเตอร์ (Stator) เพียง 2 เฟสในเวลาเดียวกัน ในขณะที่อีก 1 เฟสจะถูกปล่อยลอย (Floating) เพื่อรอจังหวะการสลับเฟสถัดไป

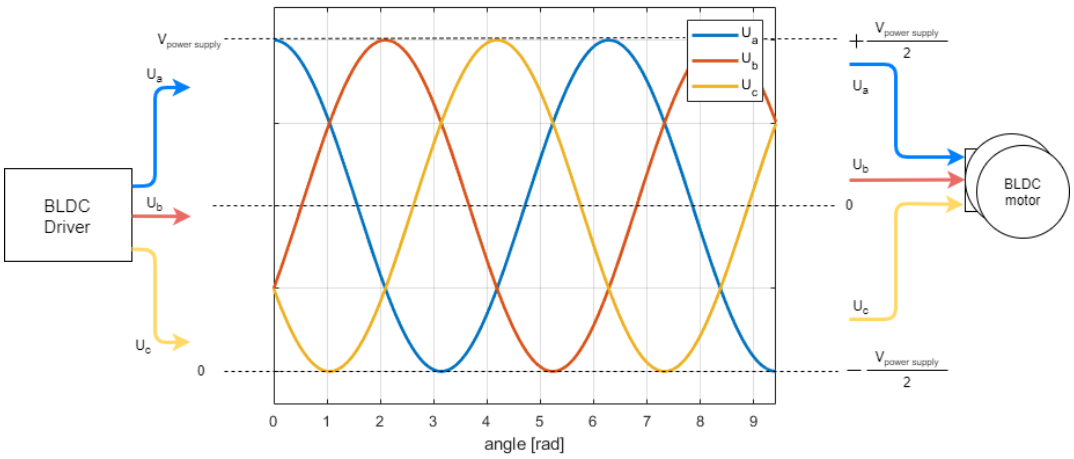
- กระบวนการทำงาน: วงจรควบคุมจะแบ่งรอบการหมุนทางไฟฟ้าออกเป็น 6 ส่วน (Sectors) โดยแต่ละส่วนมีมุมต่างกัน 60 องศาทางไฟฟ้า รวมเป็น 360 องศาต่อหนึ่งรอบไฟฟ้า
- ลักษณะสัญญาณ: รูปคลื่นกระแสที่ไหลผ่านขดลวดจะมีลักษณะเป็นรูปสี่เหลี่ยมคางหมู (Trapezoidal) ซึ่งเกิดจากการสลับสวิตช์ (Commutation) ทุกๆ 60 องศา เพื่อสร้างสนามแม่เหล็กหมุน (Rotating Magnetic Field) ให้ดึงดูดกับแม่เหล็กถาวรที่โรเตอร์

- ข้อจำกัด: เนื่องจากมีการเปลี่ยนเฟสแบบขั้นบันได (Step) ทำให้เกิดแรงบิดกระเพื่อม (Torque Ripple) ในจังหวะสลับเฟส ส่งผลให้มอเตอร์มีการสั่นสะเทือนและเสียงรบกวนมากกว่าการควบคุมแบบคลื่นไซน์

หลักการควบคุมแบบ Field Oriented Control (FOC)



รูปที่ 21 ความสัมพันธ์ระหว่างแกนอ้างอิง d - q กับแกนขดลวดสเตเตอร์และโรเตอร์



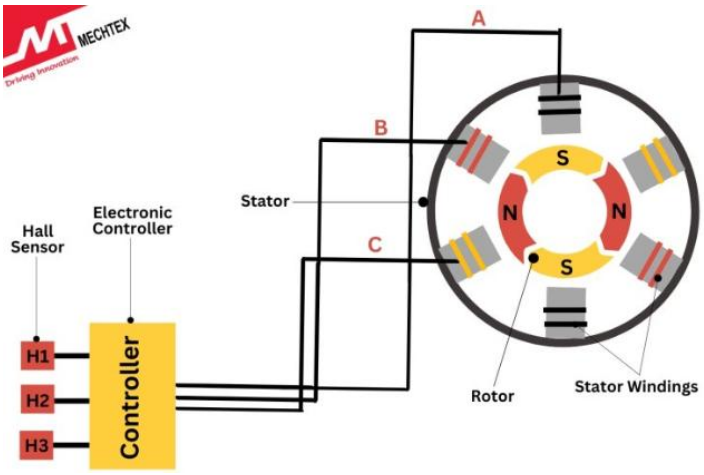
รูปที่ 22 สัญญานแบบ Field-Oriented Control

Field Oriented Control (FOC) หรือ Vector Control เป็นเทคนิคการควบคุมขั้นสูงที่มุ่งเน้นการควบคุมเวกเตอร์ของกระแสไฟฟ้าเพื่อให้ได้ประสิทธิภาพและแรงบิดสูงสุด

- หลักการทำงาน: FOC จะทำการแปลงกระแสไฟฟ้า 3 เฟส (I_a, I_b, I_c) ที่วัดได้ ให้กลายเป็นกระแสในแกนหมุน 2 แกน (I_d , I_q) ผ่านกระบวนการทางคณิตศาสตร์ที่เรียกว่า Clarke และ Park Transformation
    - แกน I_d (Flux Component): ควบคุมให้เป็น 0 เพื่อลดการสูญเสียพลังงาน (ในกรณีมอเตอร์แม่เหล็กถาวรทั่วไป)
    - แกน I_q (Torque Component): ควบคุมให้ตั้งฉากกับสนามแม่เหล็กโรเตอร์ตลอดเวลา (90 องศา) เพื่อสร้างแรงบิดสูงสุด

- ลักษณะสัญญาณ: การควบคุมนี้จะสร้างสัญญาณ PWM แบบ Space Vector (SVPWM) ทำให้กระแสที่จ่ายเข้ามอเตอร์เป็นรูปคลื่นไซน์ (Sinusoidal) ที่สมบูรณ์ ส่งผลให้มอเตอร์หมุนเรียบ ไม่มีแรงบิดกระเพื่อม และเสียงเงียบ

หลักการทำงานของ Hall Effect Sensor

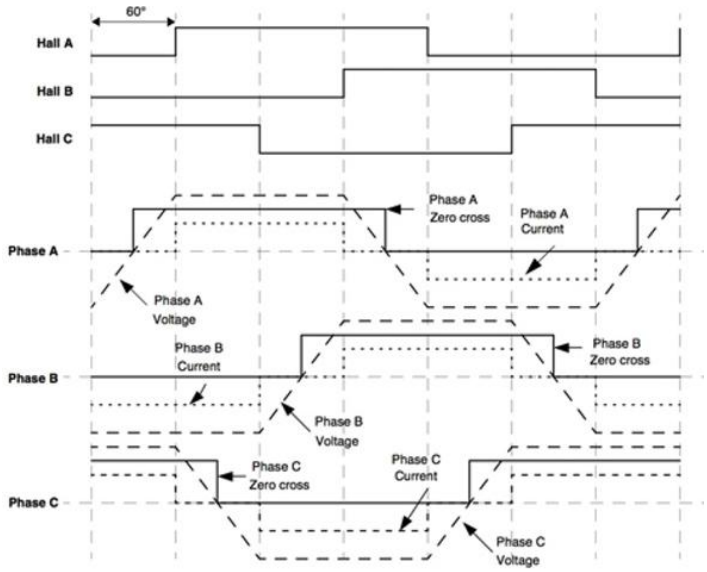


รูปที่ 23 Hall sensor in BLDC Motor

Hall Effect Sensor เป็นอุปกรณ์ตรวจจับตำแหน่งโรเตอร์ที่นิยมใช้ในมอเตอร์ BLDC

- หลักการทำงาน: ทำงานโดยอาศัยปรากฏการณ์ Hall Effect เมื่อมีสนามแม่เหล็กจากโรเตอร์เคลื่อนที่ผ่านตัวเซนเซอร์ จะเกิดแรงดันไฟฟ้าตกคร่อมภายในตัวเซนเซอร์ ซึ่งจะถูกแปลงเป็นสัญญาณดิจิทัล (High/Low)
- การติดตั้ง: โดยทั่วไปจะติดตั้งเซนเซอร์ 3 ตัววางทำมุมห่างกัน 120 องศา หรือ 60 องศาทางไฟฟ้ารอบสเตเตอร์ เพื่อระบุตำแหน่งของขั้วแม่เหล็กโรเตอร์
- หน้าที่: สัญญาณจาก Hall Sensor จะถูกส่งไปยังคอนโทรลเลอร์เพื่อบอกจังหวะในการสลับเฟส (Commutation Timing) ซึ่งวิธีนี้มีความแม่นยำสูงและทำงานได้ดีแม้ในขณะที่มอเตอร์ยังไม่หมุนหรือหมุนที่รอบต่ำ

หลักการตรวจจับแบบ Back EMF Sensing (sensor less)



รูปที่ 24 สัญญานเอาต์พุตของ Hall sensor เทียบกับค่าแรงดัน BEMF

Back Electromotive Force (Back EMF) หรือแรงดันต้านกลับ เป็นเทคนิคการระบุตำแหน่งโรเตอร์โดยไม่ต้องใช้เซนเซอร์ภายนอก (sensor less Control)

- หลักการทำงาน: เมื่อมอเตอร์หมุน ขดลวดจะตัดผ่านสนามแม่เหล็กและเหนี่ยวนำให้เกิดแรงดันไฟฟ้า (Back EMF) ในเฟสที่ไม่ได้จ่ายไฟ (Floating Phase) แรงดันนี้จะแปรผันตรงกับความเร็วรอบของมอเตอร์ $E = K_e \times \omega$
- Zero Crossing Detection (ZCD): วงจรควบคุมจะทำการตรวจสอบจุดที่สัญญาณ Back EMF ตัดผ่านระดับแรงดันอ้างอิงกึ่งกลาง (Zero Crossing Point) เพื่อใช้เป็นจุดอ้างอิงในการสั่งสลับเฟสถัดไป
- ข้อจำกัด: เทคนิคนี้ไม่สามารถใช้งานได้ดีในช่วงเริ่มต้นหมุน (Startup) หรือที่ความเร็วรอบต่ำ เนื่องจากแรงดัน Back EMF จะมีค่าน้อยเกินกว่าที่วงจรจะตรวจจับได้แม่นยำ (Low Signal-to-Noise Ratio)

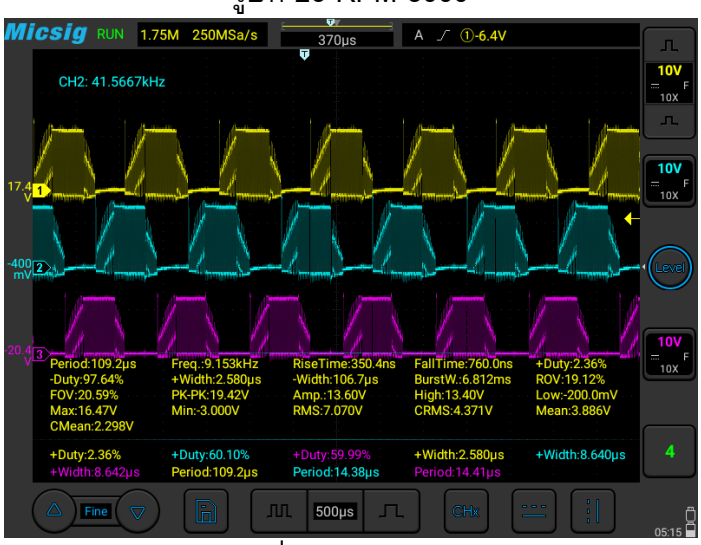
ขั้นตอนการดำเนินงาน

เอาสาย Oscilloscope Probe ต่อเข้าแต่ละขาของ Motor ได้แก่ขา GND,L1,L2,L3
ติดตั้งและตั้งค่าโปรแกรมสำหรับบอร์ดควบคุมมอเตอร์
เริ่มทดสอบโดยปรับค่า PWM ทีละ 500 ตั้งแต่ -10000 จนถึง 10,000
บันทึกผลการทดลอง รูปคลื่นสัญญาณ และค่าความเร็วรอบที่วัดได้จริง

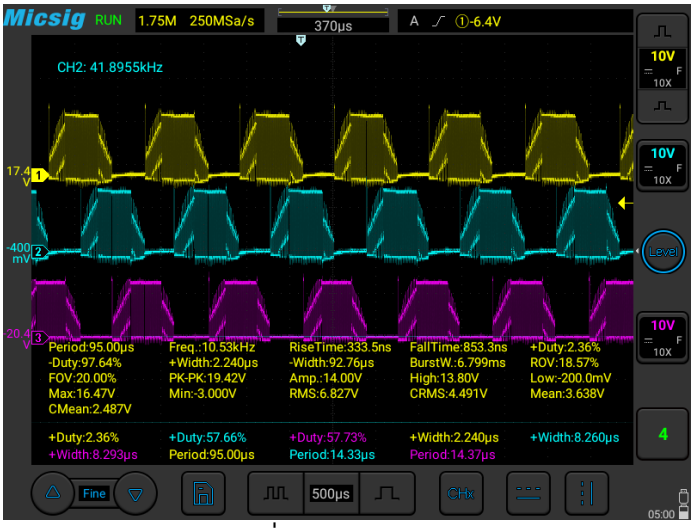
ผลการทดลอง

รูปที่ 25 RPM 2000

รูปที่ 26 RPM 5000



รูปที่ 27 RPM 8000



รูปที่ 28 RPM -8000

สรุปผลการทดลอง

จากการทดลองนี้พบว่ามอเตอร์เริ่มหมุนได้อย่างเสถียรที่ RPM |2000| จนถึง |8800| ถ้าหากน้อยกว่า |2000| จะไม่หมุน และถ้ามากกว่า |8800| โปรแกรมหยุดการทำงานและขึ้น Over voltage

จากรูปที่ x RPM จะได้ค่าสัญญานที่อ่านมาจาก Oscilloscope ในช่วง 2000 RPM จะมีคาบสัญญานที่กว้าง และจากรูปที่ x RPM 8000 ยิ่ง RPM มากคาบของสัญญานจะเล็กลง

จากรูปที่ x RPM 8000 สัญญานจะมีลำดับการไล่จาก L3 -> L2 -> L1 ส่วนจากรูปที่ x RPM -8000 สัญญานไล่ลำดับตรงข้ามจาก L1 -> L2 -> L3

อภิปรายผล (วิเคราะห์สิ่งที่เกิดขึ้นในการทดลอง)

มอเตอร์หยุดหมุนในช่วงน้อยกว่า RPM |2000| เนื่องจากว่ามอเตอร์ตัวนี้ใช้การตรวจจับสัญญานแบบ Zero crossing detection ซึ่งเป็นการอ่านค่าจาก Back EMF เพื่อจับจังหวะในการจ่ายสัญญาน จากสมการ $E = K_e \times \omega$ ด้วยมอเตอร์ที่หมุนเร็วไม่พอทำให้ค่าสัญญาน Back EMF น้อยโปรแกรมจึงตรวจจับค่าสัญญานไม่เจอ ทำให้เกิดอาการ Loss of synchronism หรืออาการที่ความที่ไม่สม่ำเสมอทำให้โปรแกรมรวน

และในช่วงที่มากกว่า RPM |8800| มอเตอร์หยุดหมุนและแจ้งเตือน Over Voltage เนื่องจากเมื่อมอเตอร์หมุนด้วยความเร็วสูงมาก ค่าแรงดัน Back EMF จะสูงขึ้นตามไปด้วย หากแรงดันนี้สูงเกินกว่าแรงดันของแหล่งจ่าย (DC Bus Voltage) หรือเกินขีดจำกัดความปลอดภัยของวงจร ระบบจะตัดการทำงานเพื่อป้องกันความเสียหายต่ออุปกรณ์อิเล็กทรอนิกส์จากการย้อนกลับของพลังงาน

ความแตกต่างของลำดับสัญญาณระหว่างค่าบวกและลบ ยืนยันได้ว่าเราสามารถใช้ลำดับการเกิดสัญญาณ (Phase Sequence) ในการระบุและควบคุมทิศทางการหมุนของมอเตอร์ได้

นอกจากนี้ จากการสังเกตภาพสัญญาณบนออสซิลโลสโคป จะพบความสัมพันธ์ระหว่างความถี่ของสัญญาณและความเร็วของมอเตอร์ที่เป็นไปในทิศทางเดียวกัน (แปรผันตรง) กล่าวคือ เมื่อความเร็วรอบ (RPM) สูงขึ้น ความถี่ของสัญญาณ Back EMF จะสูงขึ้นตามไปด้วย ซึ่งสอดคล้องกับทฤษฎี

ความสัมพันธ์ระหว่างความเร็วทางกลและความถี่ทางไฟฟ้า เราสามารถยืนยันความสัมพันธ์นี้ได้โดยการคำนวณความเร็วของมอเตอร์จากความถี่ของสัญญาณ BEMF ด้วยสมการ

$$N = \frac{120 \times f}{P}$$

(โดยที่ N คือความเร็วรอบ, f คือความถี่สัญญาณที่อ่านได้, และ P คือจำนวนขั้วแม่เหล็กของมอเตอร์) ซึ่งเมื่อนำค่าความถี่จากผลการทดลองมาแทนค่าในสมการ จะได้ค่าความเร็วรอบที่ใกล้เคียงกับค่า RPM ที่ตั้งไว้ในโปรแกรม เป็นการยืนยันว่าระบบควบคุมทำงานได้ถูกต้องตามหลักการ 6-Step Commutation

ข้อเสนอแนะ

เนื่องจากที่ความเร็วรอบสูง ค่า RPM ที่อ่านได้จะมีการแกว่งตัว เช่น ตั้งค่า 8,000 แต่อ่านได้ 7,800-8,200 ในการทดลองครั้งถัดไป ควรรอให้มอเตอร์ทำงานจนเข้าสู่สภาวะคงตัว (Steady State) ประมาณ 1 นาที ก่อนทำการบันทึกผลเพื่อความแม่นยำ

อ้างอิง

https://www.mathworks.com/help/mcb/ref/sixstepcommutation.html

https://wiki.st.com/stm32mcu/wiki/STM32MotorControl:6-step_Firmware_Algorithm

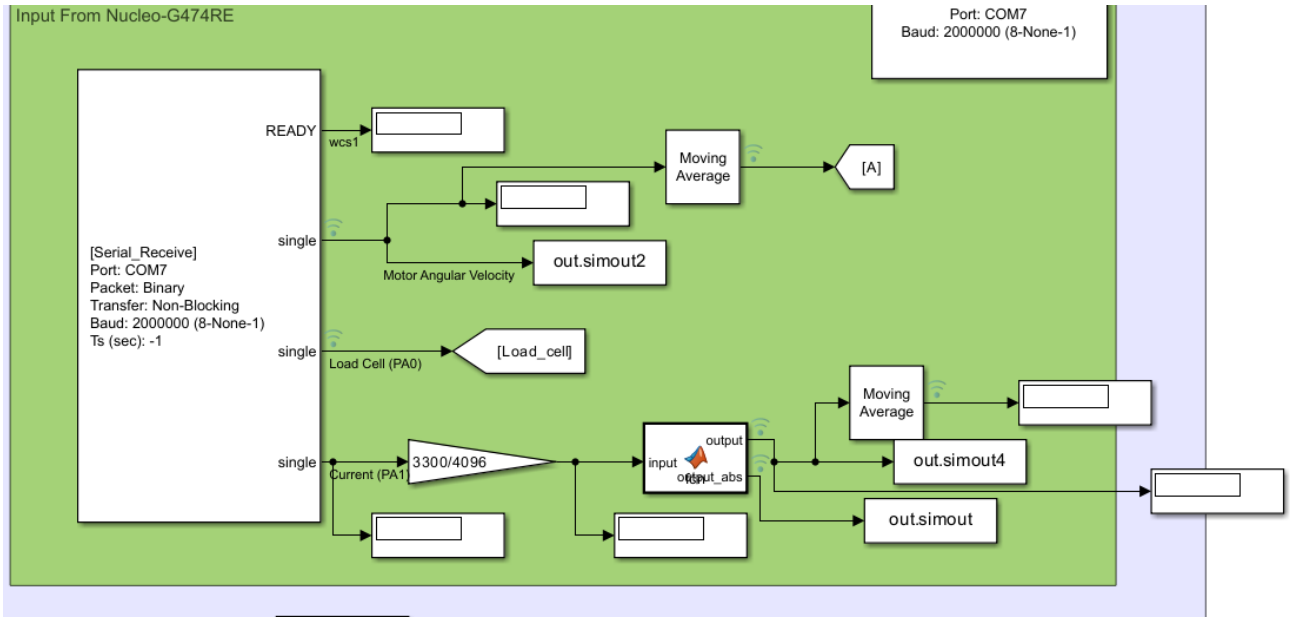
https://br.mouser.com/blog/basics-of-motor-field-oriented-control

https://mechtex.com/blog/understanding-the-hall-effect-sensors-in-bldc-motors

https://www.digikey.co.th/th/articles/controlling-sensorless-bldc-motors-via-back-emf

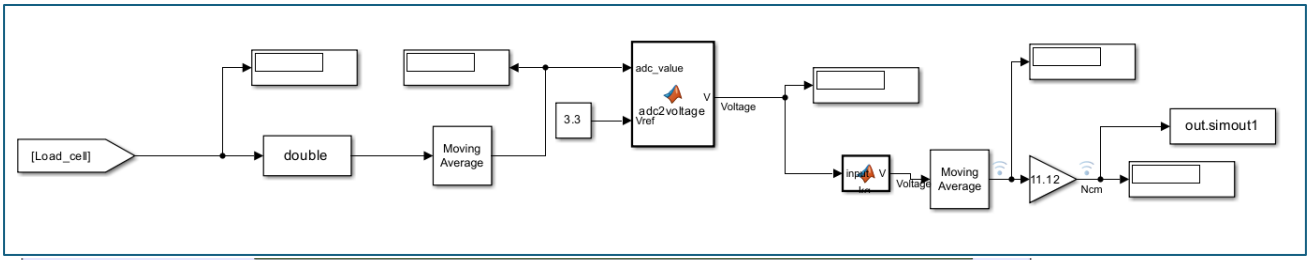
https://www.millenniumsemi.com/blog/how-hall-effect-sensors-are-used-to-control-bldc-motors/

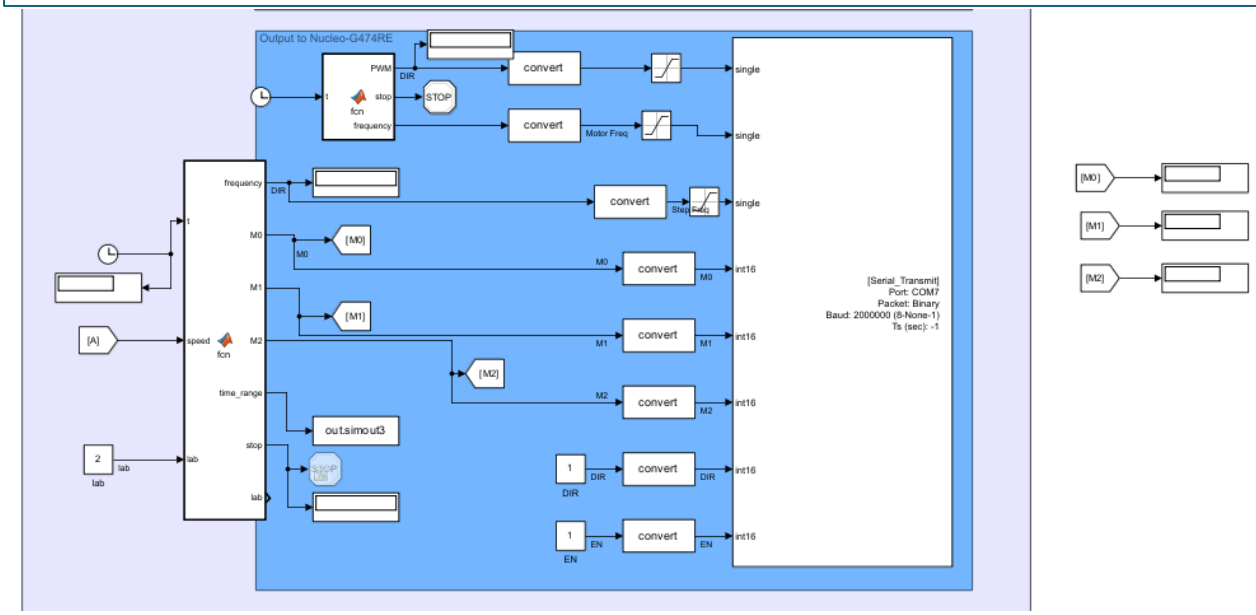
ภาคผนวก ก Simulink



เป็นหน้าสำหรับการแสดงค่า Sensor ต่างๆที่อ่านค่าได้ โดยประกอบไปด้วย 3 ส่วนคือ 1. RPM response 2.Load Cell 3. Current Sensor

1. RPM response จะทำการใส่ Moving Average ไว้เพื่อป้องกันไม่ให้ Motor





```
function [frequency, M0, M1, M2, time_range, stop, lab] = fcn(t, speed, lab)
%#codegen

persistent cycleStartTime
detectDelay = 40;        % detect 0-speed AFTER this time (case 1)
frequency_stop = 6000;   % target stop frequency for case 2

% Initialize
if isempty(cycleStartTime)
    cycleStartTime = 0;
end

% Defaults
M0 = 0; M1 = 0; M2 = 0;
frequency = 0;
stop = 0;

dt = t - cycleStartTime;
time_range = [cycleStartTime t];

switch lab

    % ============================================================
    % CASE 1 : Your original logic
    % ============================================================
    case 1

        if dt < 3
            frequency = 900;
            return;
        end

        block = floor(dt - 3) + 1;
        if block < 1
            block = 1;
        end

        frequency = block * 50;

        if dt < detectDelay
            return;
        end

        if speed <= 1
            frequency = 0;
            stop = 1;
        end
```

```
NCER\Documents\GitHub\FRA271_Lab2_Actuators\DC Motor and Stepper\DC_load_cell_average.m
%% --- Load All Files ---
load('DC50HzLoadcell_1.mat');    d50_1 = out;
load('DC50HzLoadcell_2.mat');    d50_2 = out;
load('DC50HzLoadcell_3.mat');    d50_3 = out;

load('DC500HzLoadcell_1.mat');   d500_1 = out;
load('DC500HzLoadcell_2.mat');   d500_2 = out;
load('DC500HzLoadcell_3.mat');   d500_3 = out;

load('DC1000HzLoadcell_1.mat');  d1k_1 = out;
load('DC1000HzLoadcell_2.mat');  d1k_2 = out;
load('DC1000HzLoadcell_3.mat');  d1k_3 = out;

load('DC10000HzLoadcell_1.mat'); d10k_1 = out;
load('DC10000HzLoadcell_2.mat'); d10k_2 = out;
load('DC10000HzLoadcell_3.mat'); d10k_3 = out;


%% --- Function to average and filter ---
process = @(d1,d2,d3) ...
    filter_and_avg( ...
        squeeze(d1.simout1.signals.values), ...
        squeeze(d2.simout1.signals.values), ...
        squeeze(d3.simout1.signals.values), ...
        d1.tout );


%% --- Process all groups ---
[t50, stall_torque50]     = deal([]);
[PWM50,  stall_torque50]  = process_and_bin(squeeze(d50_1.simout1.s
[PWM500, stall_torque500] = process_and_bin(squeeze(d500_1.simout1.
[PWM1k,  stall_torque1k]  = process_and_bin(squeeze(d1k_1.simout1.s
[PWM10k, stall_torque10k] = process_and_bin(squeeze(d10k_1.simout1.


%% --- Plot All Together ---
figure;
hold on;
plot(PWM50,  stall_torque50,  'LineWidth', 2);
plot(PWM500, stall_torque500, 'LineWidth', 2);
plot(PWM1k,  stall_torque1k,  'LineWidth', 2);
plot(PWM10k, stall_torque10k, 'LineWidth', 2);
hold off;

title('1-Second Averaged Loadcell Signals vs PWM');
xlabel('PWM (%)');
ylabel('Stall Torque (g)');
legend('50 Hz', '100 Hz', '1000 Hz', '10000 Hz');
grid on;


%% --- Helper Function ---
function [PWM_out, y_out] = process_and_bin(sig1, sig2, sig3, t)

    % 1) Average 3 signals
    avg = (sig1 + sig2 + sig3) / 3;

    % 2) Low-pass filter
    fs = 1 / mean(diff(t));
    fc = 10;
    [b,a] = butter(2, fc/(fs/2), 'low');
    avg_filt = filtfilt(b,a,avg);

    % 3) Convert time → PWM
```

rs\ACER\Documents\GitHub\FRA271_Lab2_Actuators\DC Motor and Stepper\DC_Stall_Current_average.m

```
figure;
hold on;
plot(PWM50,  stall_current50,  'LineWidth', 2);
plot(PWM500, stall_current500, 'LineWidth', 2);
plot(PWM1k,  stall_current1k,  'LineWidth', 2);
plot(PWM10k, stall_current10k, 'LineWidth', 2);
hold off;

title('1-Second Averaged Stall Current Signals vs PWM');
xlabel('PWM (%)');
ylabel('Stall Current (A)');
legend('50 Hz', '100 Hz', '1000 Hz', '10000 Hz');
grid on;


%% --- Helper Function ---
function [PWM_out, y_out] = process_and_bin(sig1, sig2, sig3, t)

    % 1) Average 3 signals
    avg = (sig1 + sig2 + sig3) / 3;

    % 2) Low-pass filter
    fs = 1 / mean(diff(t));
    fc = 10;
    [b,a] = butter(2, fc/(fs/2), 'low');
    avg_filt = filtfilt(b,a,avg);

    % 3) Convert time → PWM
    PWM = 5 * t;

    % 4) Bin into 1-second intervals
    bin_edges = 0:1:max(t);
    y_out   = zeros(length(bin_edges)-1,1);
    PWM_out = zeros(length(bin_edges)-1,1);

    for i = 1:length(bin_edges)-1
        idx = t >= bin_edges(i) & t < bin_edges(i+1);
        y_out(i)   = mean(avg_filt(idx));      % torque avg in that 1-sec window
        PWM_out(i) = mean(PWM(idx));           % PWM avg for that window
    end
    % 5) Remove offset → make first value = 0
    y_out = y_out - y_out(1);
end
```

```
[t50, noloadcurrent50]     = deal([]);
[PWM50,  noloadcurrent50]  = process_and_bin(squeeze(d50_1.simout4.signals.values),  squeeze(d50_2.simout4.signals.values),  squeeze(d50_3.simout4.signals.values),  d50_1.tout);
[PWM100, noloadcurrent100] = process_and_bin(squeeze(d100_1.simout4.signals.values), squeeze(d100_2.simout4.signals.values), squeeze(d100_3.simout4.signals.values), d100_1.tout);
[PWM1k,  noloadcurrent1k]  = process_and_bin(squeeze(d1k_1.simout4.signals.values),  squeeze(d1k_2.simout4.signals.values),  squeeze(d1k_3.simout4.signals.values),  d1k_1.tout);
[PWM10k, noloadcurrent10k] = process_and_bin(squeeze(d10k_1.simout4.signals.values), squeeze(d10k_2.simout4.signals.values), squeeze(d10k_3.simout4.signals.values), d10k_1.tout);
```

```
%% --- Plot All Together ---
figure;
hold on;
plot(PWM50,  noloadcurrent50,  'LineWidth', 2);
plot(PWM100, noloadcurrent100, 'LineWidth', 2);
plot(PWM1k,  noloadcurrent1k,  'LineWidth', 2);
plot(PWM10k, noloadcurrent10k, 'LineWidth', 2);
hold off;

title('1-Second Averaged No load Current Signals vs PWM');
xlabel('PWM (%)');
ylabel('No Load Current (A)');
legend('50 Hz', '100 Hz', '1000 Hz', '10000 Hz');
grid on;
```

```
%% --- Helper Function ---
function [PWM_out, y_out] = process_and_bin(sig1, sig2, sig3, t)

    % 1) Average 3 signals
    avg = (sig1 + sig2 + sig3) / 3;

    % 2) Low-pass filter
    fs = 1 / mean(diff(t));
    fc = 10;
    [b,a] = butter(2, fc/(fs/2), 'low');
    avg_filt = filtfilt(b,a,avg);

    % 3) Convert time → PWM
    PWM = 5 * t-100;

    % 4) Bin into 1-second intervals
    bin_edges = 0:1:max(t);
    y_out   = zeros(length(bin_edges)-1,1);
    PWM_out = zeros(length(bin_edges)-1,1);

    for i = 1:length(bin_edges)-1
        idx = t >= bin_edges(i) & t < bin_edges(i+1);
        y_out(i)   = mean(avg_filt(idx));      % torque avg in that 1-sec window
        PWM_out(i) = mean(PWM(idx));           % PWM avg for that window
    end
    % 5) Remove offset → make first value = 0
    midIndex = round(length(y_out) / 2);
    y_out = y_out - (y_out(midIndex)+y_out(midIndex+1))/2;

end
```

```
function modified_signal = delete_first_half(signal)
    signal_length = length(signal);
    half_index = ceil(signal_length / 2);
    modified_signal = signal(half_index + 1:end);
end

function maximum_efficiency = calculate_efficiency(noloadcurrent, stallcurrent)

    if stallcurrent == 0
        maximum_efficiency = 0;
        return;
    end

    ratio = noloadcurrent ./ stallcurrent;   % element-wise division
    ratio(ratio < 0) = 0;

    maximum_efficiency = (1 - sqrt(ratio)) ./ (1 + sqrt(ratio));
end


% --- Only cut CURRENT, do NOT cut PWM ---

half_noloadcurrent50  = delete_first_half(noloadcurrent50);
half_noloadcurrent100 = delete_first_half(noloadcurrent100);
half_noloadcurrent1k  = delete_first_half(noloadcurrent1k);
half_noloadcurrent10k = delete_first_half(noloadcurrent10k);

stall_current50_adj  = stall_current50  + half_noloadcurrent50(1);
stall_current100_adj = stall_current500 + half_noloadcurrent100(1);
stall_current1k_adj  = stall_current1k  + half_noloadcurrent1k(1);
stall_current10k_adj = stall_current10k + half_noloadcurrent10k(1);


% Compute efficiency (vector)
maxeff50  = calculate_efficiency(half_noloadcurrent50,  stall_current50_adj);
maxeff100 = calculate_efficiency(half_noloadcurrent100, stall_current100_adj);
maxeff1k  = calculate_efficiency(half_noloadcurrent1k,  stall_current1k_adj);
maxeff10k = calculate_efficiency(half_noloadcurrent10k, stall_current10k_adj);



% --- PWM must match length, so trim PWM to last half ---
PWM50_trim  = PWM50(end-length(maxeff50)+1:end);
PWM100_trim = PWM100(end-length(maxeff100)+1:end);
PWM1k_trim  = PWM1k(end-length(maxeff1k)+1:end);
PWM10k_trim = PWM10k(end-length(maxeff10k)+1:end);

% --- Plot stall current adjusted with noload current ---
% figure;
% hold on;
% plot(PWM50_trim, stall_current50_adj, 'LineWidth', 2, 'DisplayName', '50 Hz');
% plot(PWM50_trim, half_noloadcurrent50, 'LineWidth', 2, 'DisplayName', '1000 Hz');
% hold off;
%
% title('Adjusted Stall Current vs PWM');
% xlabel('PWM (%)');
% ylabel('Adjusted Stall Current');
% legend('show');
% grid on;
% % % --- Plot ---
figure;
hold on;
plot(PWM50_trim,  maxeff50,  'LineWidth', 2);
plot(PWM100_trim, maxeff100, 'LineWidth', 2);
plot(PWM1k_trim,  maxeff1k,  'LineWidth', 2);
plot(PWM10k_trim, maxeff10k, 'LineWidth', 2);
hold off;
```

```
torque_values = linspace(0, max([PWM3V.stalltorque, PWM6V.stalltorque, PWM9V.stalltorque, PWM12V.stalltorque]), 100);

% Calculate speed for each motor using the speed equation
speed_3V = speedConstant3V * torque_values + offset3V;
speed_6V = speedConstant6V * torque_values + offset6V;
speed_9V = speedConstant9V * torque_values + offset9V;
speed_12V = speedConstant12V * torque_values + offset12V;

% Plotting the results
figure;
hold on;
plot(torque_values, speed_3V, 'DisplayName', '3V/25%PWM', 'LineWidth', 3);
plot(torque_values, speed_6V, 'DisplayName', '6V/50%PWM', 'LineWidth', 3);
plot(torque_values, speed_9V, 'DisplayName', '9V/75%PWM', 'LineWidth', 3);
plot(torque_values, speed_12V, 'DisplayName', '12V/100%PWM', 'LineWidth', 3);
hold off;

% Adding labels and legend
xlabel('Torque (Nm)');
ylabel('Speed (RPM)');
title('Speed vs Torque for Different Motors');
legend show;
grid on;
```

ers\ACER\Documents\GitHub\FRA271_Lab2_Actuators\DC Motor and Stepper\DC_rpm_average.m

```
%% --- Load All Files ---
load('DC50HzPWM_1.mat');     d50_1  = out;
load('DC50HzPWM_2.mat');     d50_2  = out;
load('DC50HzPWM_3.mat');     d50_3  = out;

load('DC100HzPWM_1.mat');    d100_1 = out;
load('DC100HzPWM_2.mat');    d100_2 = out;
load('DC100HzPWM_3.mat');    d100_3 = out;

load('DC1000HzPWM_1.mat');   d1k_1  = out;
load('DC1000HzPWM_2.mat');   d1k_2  = out;
load('DC1000HzPWM_3.mat');   d1k_3  = out;

load('DC10000HzPWM_1.mat');  d10k_1 = out;
load('DC10000HzPWM_2.mat');  d10k_2 = out;
load('DC10000HzPWM_3.mat');  d10k_3 = out;
```

```
%% --- Function to average and filter ---
process = @(d1,d2,d3) ...
    filter_and_avg( ...
        squeeze(d1.simout2.signals.values), ...
        squeeze(d2.simout2.signals.values), ...
        squeeze(d3.simout2.signals.values), ...
        d1.tout );
```

```
%% --- Process all groups ---
[t50, rpm50]     = deal([]);
[PWM50,  rpm50]  = process_and_bin(squeeze(d50_1.simout2.signals.values),  squeeze(d50_2.simout2.signals.values),  squeeze(d50_3.simout2.signals.values),  d50_1.tout);
[PWM100, rpm100] = process_and_bin(squeeze(d100_1.simout2.signals.values), squeeze(d100_2.simout2.signals.values), squeeze(d100_3.simout2.signals.values), d100_1.tout);
[PWM1k,  rpm1k]  = process_and_bin(squeeze(d1k_1.simout2.signals.values),  squeeze(d1k_2.simout2.signals.values),  squeeze(d1k_3.simout2.signals.values),  d1k_1.tout);
[PWM10k, rpm10k] = process_and_bin(squeeze(d10k_1.simout2.signals.values), squeeze(d10k_2.simout2.signals.values), squeeze(d10k_3.simout2.signals.values), d10k_1.tout);
```

```
%% --- Plot All Together ---
figure;
hold on;
plot(PWM50,  rpm50,  'LineWidth', 2);
plot(PWM100, rpm100, 'LineWidth', 2);
plot(PWM1k,  rpm1k,  'LineWidth', 2);
plot(PWM10k, rpm10k, 'LineWidth', 2);
hold off;

title('1-Second Averaged Loadcell Signals vs PWM');
xlabel('PWM (%)');
ylabel('Rotational Speed (RPM)');
legend('50 Hz', '100 Hz', '1000 Hz', '10000 Hz');
grid on;
```

```
%% --- Helper Function ---
function [PWM_out, y_out] = process_and_bin(sig1, sig2, sig3, t)

    % 1) Average 3 signals
    avg = (sig1 + sig2 + sig3) / 3*60 / (2*pi);

    % 2) Low-pass filter
    fs = 1 / mean(diff(t));
    fc = 10;
    [b,a] = butter(2, fc/(fs/2), 'low');
    avg_filt = filtfilt(b,a,avg);
```

```
function motorSpec = createMotorSpec(noloadspeed, noloadcurrent, stalltorque, stallcurrent)
    motorSpec = struct('noloadspeed', noloadspeed, ...
                       'noloadcurrent', noloadcurrent, ...
                       'stalltorque', stalltorque, ...
                       'stallcurrent', stallcurrent);
end

PWM3V = createMotorSpec(437.42,0.357,530.05,1.32);
PWM6V = createMotorSpec(1088.37,0.353,1034.16,2.77);
PWM9V = createMotorSpec(1716.71,0.53,1545.68,4.09);
PMW12V = createMotorSpec(2335.22,0.41,1949.71,5.27);

% Create arrays for plotting
noloadspeeds = [PWM3V.noloadspeed, PWM6V.noloadspeed, PWM9V.noloadspeed, PMW12V.noloadspeed];
stalltorques = [PWM3V.stalltorque, PWM6V.stalltorque, PWM9V.stalltorque, PMW12V.stalltorque];

plotMotorCharacteristics(437.42,0.357,530.05,1.32);
function plotMotorCharacteristics(noloadspeed, noloadcurrent, stalltorque, stallcurrent)
stalltorque = stalltorque/11.12/11.12
% plotMotorCharacteristics(noloadspeed, noloadcurrent, stalltorque, stallcurrent)
% Plots Speed (rpm), Current (A), Efficiency (%) and Power (W) vs Torque on
% a single figure using multiple overlaid axes.
%
% NOTE: This code assumes `stalltorque` units are mNm (millinewton-meter).
% If your torque is already in N*m change the line "torque_Nm = torque/1000;"
% to "torque_Nm = torque;"

    %% create torque vector (same units as stalltorque input)
    torque = linspace(0, stalltorque, 200);

    %% motor linear model
    speed = noloadspeed * (1 - torque / stalltorque);                    % rpm
    current = noloadcurrent + (stallcurrent - noloadcurrent) * (torque / stalltorque);

    %% Efficiency (simple estimate)
    % Here we compute mechanical power below more properly and compute efficiency
    % as Pout / (I * Vscale). Because we do not have V, we normalize efficiency
    % to percent by scaling with max(Pout) if desired. Keep as relative.
    % (You can replace Vscale if you know the supply voltage).
    omega_rad = speed * 2*pi/60;    % convert rpm -> rad/s

    % Torque units: convert to N*m if given in mNm
    torque_Nm = torque;      % <-- change/remove if input is in N*m

    % Mechanical output power (W)
    Pout = torque_Nm .* omega_rad;  % W

    % Simple input power proxy: use current scaled by a constant (no V known)
    Pin_proxy = current .* 12;   % arbitrary scaling so shapes sensible
    efficiency = zeros(size(Pout));
    valid = (Pin_proxy > 0);
    efficiency(valid) = (Pout(valid) ./ Pin_proxy(valid));  % % (relative)
    efficiency(~valid) = 0;
    efficiency(isnan(efficiency)) = 0;
    efficiency(efficiency < 0) = 0;

    %% create main axes
    fig = figure('Name','Motor Characteristics (multi-axis)','NumberTitle','off');
    ax1 = axes('Parent', fig);
    hold(ax1, 'on');

    % left y-axis -> Speed
    yyaxis(ax1, 'left');
    hSpeed = plot(torque, speed, 'b', 'LineWidth', 2);
    ylabel('Speed (RPM)');

    % right y-axis -> Current
```

```
grid on;
xlabel('Time (s)');
ylabel('Averaged Value');
title('1-Second Averaged Stepper Frequency Data');

% Display R-squared text
text(5.1, max([f000_bin; f001_bin; f010_bin]) * 0.95, eq000, 'Color','r');
text(5.1, max([f000_bin; f001_bin; f010_bin]) * 0.90, eq001, 'Color','k');  % ← changed to black
text(5.1, max([f000_bin; f001_bin; f010_bin]) * 0.85, eq010, 'Color','b');

legend show;
```

```
%% --- R-SQUARE helper ---
function [m, b, R2] = linearFitWithR2(x, y)
    p = polyfit(x, y, 1);   % linear fit
    m = p(1);
    b = p(2);

    yfit = polyval(p, x);
    SSres = sum((y - yfit).^2);
    SStot = sum((y - mean(y)).^2);

    R2 = 1 - SSres/SStot;
end
```

No load Current			Stall Load				Stall Current(A)				Stall Load			
2	3	MEAN	1	2	3	MEAN	1	2	3	avg	1	2	3	MEAN
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
stall	stall	0.26	19	18	19	18.66666667	0.45	0.45	0.45	0.45	1.708633094	1.618705036	1.708633094	1.678657074
0.37	0.33	0.3566666667	47	48	48	47.66666667	1.3	1.34	1.33	1.323333333	4.226618705	4.316546763	4.316546763	4.286570743
0.33	0.33	0.3333333333	76	76	72	74.66666667	2.03	2.1	2.07	2.066666667	6.834532374	6.834532374	6.474820144	6.714628297
0.36	0.35	0.3533333333	95	93	91	93	2.78	2.78	2.74	2.766666667	8.543165468	8.363309353	8.183453237	8.363309353
0.37	0.36	0.3633333333	116	114	113	114.3333333	3.48	3.45	3.42	3.45	10.43165468	10.25179856	10.1618705	10.28177458
0.35	0.34	0.3533333333	142	137	138	139	4.12	4.07	4.07	4.086666667	12.76978417	12.32014388	12.41007194	12.5
0.4	0.4	0.41	161	162	159	160.6666667	4.71	4.7	4.63	4.68	14.47841727	14.56834532	14.29856115	14.44844125
0.4	0.43	0.41	178	177	171	175.3333333	5.32	5.31	5.19	5.273333333	16.00719424	15.91726619	15.37769784	15.76738609

ACER\Documents\GitHub\FRA271_Lab2_Actuators\DC Motor and Stepper\stepper_lab2.m

```
% ---------------- Load Data ----------------
load('stepperramp50_1.mat'); ramp50_1_data = squeeze(out.simout2.signals.values);
load('stepperramp50_2.mat'); ramp50_2_data = squeeze(out.simout2.signals.values);
load('stepperramp50_3.mat'); ramp50_3_data = squeeze(out.simout2.signals.values);
tramp50  = out.tout;
load('stepperramp100_1.mat'); ramp100_1_data = squeeze(out.simout2.signals.values);
load('stepperramp100_2.mat'); ramp100_2_data = squeeze(out.simout2.signals.values);
load('stepperramp100_3.mat'); ramp100_3_data = squeeze(out.simout2.signals.values);
tramp100  = out.tout;
load('stepperramp500_1.mat'); ramp500_1_data = squeeze(out.simout2.signals.values);
load('stepperramp500_2.mat'); ramp500_2_data = squeeze(out.simout2.signals.values);
load('stepperramp500_3.mat'); ramp500_3_data = squeeze(out.simout2.signals.values);
tramp500  = out.tout;

% ---------------- Filtering ----------------
fs = 1000;
fc = 10;
[b,a] = butter(2, fc/(fs/2));

% 50 Hz
f50_1  = filtfilt(b,a,ramp50_1_data);
f50_2  = filtfilt(b,a,ramp50_2_data);
f50_3  = filtfilt(b,a,ramp50_3_data);

% 100 Hz
f100_1 = filtfilt(b,a,ramp100_1_data);
f100_2 = filtfilt(b,a,ramp100_2_data);
f100_3 = filtfilt(b,a,ramp100_3_data);

% 500 Hz
f500_1 = filtfilt(b,a,ramp500_1_data);
f500_2 = filtfilt(b,a,ramp500_2_data);
f500_3 = filtfilt(b,a,ramp500_3_data);

% ---------------- Compute Mean RPM ----------------
mean50  = mean([f50_1 f50_2 f50_3],2);
mean100 = mean([f100_1 f100_2 f100_3],2);
mean500 = mean([f500_1 f500_2 f500_3],2);

accel50 = gradient(mean50, tramp50);
accel100 = gradient(mean100, tramp100);
accel500 = gradient(mean500, tramp500);



% % ---------------- Plot Mean Speed Graphs ----------------
% figure;
%
% % Define frequency variables in Hz
% freq50 = tramp50 * 50 / (2 * pi);
% freq100 = tramp100 * 100 / (2 * pi);
% freq500 = tramp500 * 500 / (2 * pi);
%
% % Plot for 50 Hz
% subplot(3, 1, 1);
% plot(freq50, mean50);
% title('Mean Speed for 50 Hz');
% xlabel('Frequency (Hz)');
% ylabel('Mean Speed (rad/s)');
% grid on;
%
% % Plot for 100 Hz
% subplot(3, 1, 2);
% plot(freq100, mean100);
% title('Mean Speed for 100 Hz');
% xlabel('Frequency (Hz)');
```