การประยุกต์ใช้งานระบบควบคุมแบบฟัซซี

APPLICATIONS OF FUZZY LOGIC CONTROL SYSTEMS

โดย

นายธราธร ลิ้มเจริญ

นายนพดล ชินวุธิ

นายสม อภิวัฒน์เจริญกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมแมคคาทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

การประยุกต์ใช้งานระบบควบคุมแบบฟัซซี

APPLICATIONS OF FUZZY LOGIC CONTROL SYSTEMS

โดย

นายธราธร ลิ้มเจริญ

นายนพดล ชินวุธิ

นายสม อภิวัฒน์เจริญกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมแมคคาทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ปีการศึกษา 2550

ภาควิชาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การประยุกต์ใช้งานระบบควบคุมแบบฟัซซี

APPLICATIONS OF FUZZY LOGIC CONTROL SYSTEMS

ผู้จัดทำ	นาย ธราธร	ลิ้มเจริญ	47010328
	นาย นพดล	ชินวุธิ	47010353
	นาย สม	อภิวัฒน์เจริญกุล	47010799

.....อาจารย์ที่ปรึกษา

(ผศ.ดร.ถาวร เบญจนราสุทธิ์)

.....อาจารย์ที่ปรึกษา

(รศ.ดร. จงกล งามวิวิทย์)

การประยุกต์ใช้งานระบบควบคุมแบบฟัซซี

โดย

นาย ธราธร	ลิ้มเจริญ	47010328
นาย นพดล	ชินวุธิ	47010353
นาย สม	อภิวัฒน์เจริญกุล	47010799

อาจารย์ที่ปรึกษา

ผศ.ดร. ถาวร เบญจนราสุทธิ์

รศ.ดร. จงกล งามวิวิทย์

ปีการศึกษา 2550

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอการประยุกต์ใช้งานระบบควบคุมแบบฟัซซี โดยใช้คอมพิวเตอร์ส่วนบุคคลเพื่อควบคุมความเร็วของระบบสองมวล และไมโครคอนโทรลเลอร์ เพื่อควบคุมความเร็ว และทิศทางการเคลื่อนที่ของหุ่นยนต์ ข้อดีประการหนึ่งของการออกแบบระบบควบคุมแบบฟัซซีคือ สามารถออกแบบระบบควบคุมได้โดยไม่จำเป็นต้องทราบแบบจำลองทางคณิตศาสตร์ของระบบที่ต้องการควบคุม

จากการทดลองควบคุมระบบสองมวลด้วยคอมพิวเตอร์ส่วนบุคคล พบว่าระบบควบคุมแบบฟัซซี สามารถควบคุมความเร็วของระบบสองมวลที่ความเร็ว 1,500 รอบต่อนาที ได้โดยสามารถเข้าสู้ค่าอ้างอิงได้เร็ว ไม่มีค่าพุ่งเกิน และไม่มีค่าการแกว่ง และจากการทดลองควบคุมความเร็วและทิศทางการเคลื่อนที่ของหุ่นยนต์โดยใช้ไมโครคอนโทรลเลอร์ พบว่าระบบควบคุม สามารถควบคุมการเคลื่อนที่ของหุ่นยนต์ให้เป็นเส้นตรง ตามที่ต้องการได้ดีกว่าระบบควบคุมแบบ PID โดยมีค่าความผิดพลาดอยู่ในระดับที่ยอมรับได้

APPLICATIONS OF FUZZY LOGIC CONTROL SYSTEMS

By

Mr. Tharathorn Limcharoen

Mr. Nopadol Chinawuti

Mr. Som Apiwatcharoengul

Advisor

Asst. Prof. Dr. Taworn Benjanarasuth

Assoc. Prof. Dr. Jongkol Ngamwiwit

Academic Year 2007

ABSTRACT

This thesis presents the applications of fuzzy logic control systems by using a personal computer (PC) to control the speed of two-mass system and microcontroller to control the speed and movement's direction of robot. One of the advantages in utilizing fuzzy control systems is that the mathematic model is not required.

The experimental results in controlling the speed of two-mass system by PC show that the fuzzy logic controller can control the speed at 1,500 rpm (1.5 Krpm) without overshoot and torsional resonance in the responses. For experimental results in controlling robot's movement, the fuzzy logic controller can control speed and direction of robot's movement better than PID controller with acceptable error.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้ สามารถสำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความช่วยเหลือเป็นอย่างดี จาก ผศ.ดร.ถาวร เบญจนราสุทธิ์ ที่ได้กรุณาให้คำปรึกษาแนะนำที่ดีมาโดยตลอดตั้งแต่ต้น รวมทั้งเอื้อเฟื้ออุปกรณ์ที่จำเป็น และความช่วยเหลืออื่นๆที่เป็นประโยชน์ต่อโครงงาน ผู้จัดทำรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ รศ.ดร.จงกล งามวิวิทย์ ที่คอยถามถึงความคืบหน้าอยู่ตลอดเวลา และให้ความกรุณาเปิดห้องทดลองให้ใช้งานเป็นประจำ

ขอขอบคุณเพื่อนๆในภาควิชาแมคคาทรอนิกส์ และน้องๆที่ชุมนุมโรบอททุกคนที่ให้กำลังใจสนับสนุนอุปกรณ์ที่ขาดเหลือ กระตุ้นเตือน รวมทั้งคอยถามไถ่ความคืบหน้าของโครงงานอยู่เสมอ

สุดท้ายนี้ผู้จัดทำขอกราบขอบพระคุณบิดา มารดา และครอบครัว ที่คอยเป็นกำลังใจที่ดีตลอดมา รวมถึงการสนับสนุนในเรื่องของงบประมาณที่ขาดเหลือ ตลอดจนเป็นแรงบันดาลใจที่ดีที่สุดที่ทำให้โครงงานนี้เสร็จสมบูรณ์ลงได้

ผู้จัดทำ
นาย ธราธร ลิ้มเจริญ
นาย นพดล ชินวุธิ
นาย สม อภิวัฒน์เจริญกุล

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 กล่าวนำ

ในปัจจุบัน ได้มีการพัฒนาวิธีการจัดการสัญญาณ โดยใช้ไมโครคอมพิวเตอร์ที่มีโปรแกรมเป็นตัวจัดการสัญญาณ ซึ่งจะมีความยืดหยุ่น และสะดวกในการใช้งานเป็นอย่างมาก ทั้งนี้ความเร็วในการประมวลผลของไมโครคอมพิวเตอร์ในยุคปัจจุบันมีความเร็วสูงมาก ทำให้เกิดปัญหาน้อยมากในการใช้งานในกระบวนการจริง

ในปี ค.ศ. 1965 Lotfi Zadeh ได้มีการนำเสนอทฤษฎีฟัซซีเซตและฟัซซีลอจิก เพื่อใช้หาข้อสรุปตามหลักการและเหตุผล เมื่อตัวแปรของระบบถูกนิยามในเชิงคุณภาพ และมีความคลุมเครือ ในปี ค.ศ. 1974 Ebrahim Mamdani ได้นำหลักการนี้มาประยุกต์ใช้ในการออกแบบตัวควบคุมแบบฟัซซีโดยแปลงความรู้ที่ได้จากประสบการณ์การควบคุมของผู้ปฏิบัติการที่อยู่ในรูปเงื่อนไข "ถ้า... แล้ว..." ให้เป็นกฎการควบคุมกระบวนการทางอุตสาหกรรม เนื่องจากมีข้อดีที่เป็นจุดเด่นหลายประการดังนี้

- สามารถออกแบบตัวควบคุมได้ โดยไม่ต้องรู้แบบจำลองทางคณิตศาสตร์ของกระบวนการ เนื่องจากการออกแบบด้วยตัวควบคุม ใช้วิธีแปลงความรู้หรือประสบการณ์ของผู้เชี่ยวชาญให้เป็นกฎการควบคุมในรูปเงื่อนไข
- สามารถควบคุมกระบวนการที่ไม่เป็นเชิงเส้นได้ เนื่องจากความไม่เป็นเชิงเส้นนี้ จะถูกควบคุมได้โดยการกำหนดความสัมพันธ์ของกฎการควบคุมแบบไม่เชิงเส้น
- สามารถออกแบบตัวควบคุม สำหรับควบคุมกระบวนการที่มีหลายอินพุท หลายเอาท์พุทได้สะดวก เนื่องจากความซับซ้อนของความสัมพันธ์ของกฎการควบคุมซึ่งอยู่ในรูปแบบที่สามารถทำความเข้าใจและทำการปรับเปลี่ยนได้ง่าย

ในโครงงานนี้จึงเลือกที่จะศึกษาการประยุกต์การใช้งานระบบควบคุมแบบฟัซซี ซึ่งเป็นการควบคุมแบบอินทิลิเจนท์ (Intelligent control) ที่มีจุดเด่นหลายประการดังที่กล่าวในข้างต้น ในระบบการควบคุมแบบอัตโนมัติแบบป้อนกลับ (Feedback control) โดยใช้ภาษา C ในการเขียนโปรแกรมระบบควบคุมแบบฟัซซี ให้ใช้งานได้จริงตามทฤษฎีฟัซซีลอจิก

1.2 วัตถุประสงค์ในการทำปริญญานิพนธ์

1. ศึกษาทฤษฎี และออกแบบระบบควบคุมอัตโนมัติแบบฟัซซี
2. ศึกษาการสร้างระบบควบคุมแบบฟัซซีด้วยเครื่องคอมพิวเตอร์ส่วนบุคคลเพื่อควบคุมความเร็วของระบบสองมวล

3. ศึกษาการสร้างระบบควบคุมแบบฟัซซีด้วยไมโครคอนโทรลเลอร์เพื่อควบคุมความเร็วและการเคลื่อนที่ของหุ่นยนต์
4. ศึกษาและสร้างหุ่นยนต์ ซึ่งประกอบด้วยอุปการณ์ที่ประกอบเป็นโครงสร้างทางกล อุปกรณ์ทางอิเล็กทรอนิกส์ ของระบบควบคุม อุปกรณ์วัดความเร็วของมอเตอร์ไฟฟ้ากระแสตรง รวมถึงส่วนของการควบคุม และประมวลผลโดยในโครงงานนี้ใช้ไมโครคอนโทรลเลอร์ และโปรแกรมภาษา C

1.3 ขั้นตอนการศึกษาและการจัดทำโครงงาน

การศึกษาระบบควบคุมอัตโนมัติแบบฟัซซีนั้น จำเป็นต้องเรียนรู้และเข้าใจทฤษฎีฟัซซี ขั้นพื้นฐานเพื่อให้เกิดความเข้าใจในการตั้งเงื่อนไข "ถ้า...แล้ว..." ในการตั้งกฎเพื่อควบคุมระบบแทนแบบจำลองทางคณิตศาสตร์ จากนั้นแบ่งการทดลองออกเป็นสองส่วนด้วยกันคือ ทดลองสร้างระบบควบคุมแบบฟัซซีด้วยคอมพิวเตอร์ส่วนบุคคล เพื่อควบคุมความเร็วของระบบสองมวลเปรียบเทียบกับระบบควบคุมแบบ PID และทดลองสร้างระบบควบคุมแบบฟัซซีด้วยไมโครคอนโทรลเลอร์ เพื่อควบคุมความเร็ว และการเคลื่อนที่ของหุ่นยนต์ที่ขับเคลื่อนด้วยมอเตอร์ไฟฟ้ากระแสตรงสองตัวซึ่งเป็นอิสระต่อกัน โดยไม่ทราบแบบจำลองทางคณิตศาสตร์ ซึ่งรายละเอียดจะกล่าวในบทต่อไป

1.4 รายละเอียดของปริญญานิพนธ์

เนื้อหาในปริญญานิพนธ์ฉบับนี้ประกอบด้วย

บทที่ 1 บทนำ กล่าวนำถึงวัตถุประสงค์ หลักการ ขั้นตอนการศึกษา และการจัดทำโครงงาน พร้อมทั้งรายละเอียดของปริญญานิพนธ์ของแต่ละบท

บทที่ 2 ทฤษฎีและความรู้ที่เกี่ยวข้อง กล่าวถึงหลักการและทฤษฎีของฟัซซี การวัดความเร็วของมอเตอร์ไฟฟ้ากระแสตรง การใช้งานไมโครคอนโทรลเลอร์ วงจรอิเล็กทรอนิกส์ที่เกี่ยวข้อง และ การนำเอาความรู้ไปประยุกต์ใช้ทำโครงงาน

บทที่ 3 หลักการออกแบบ นำเสนอการประกอบโครงสร้างของระบบ รวมถึงแนวคิดในการออกแบบระบบควบคุม

บทที่ 4 การทดลอง เป็นส่วนการทดสอบองค์ประกอบต่างๆ ในระบบ ตลอดจนการทดลองระบบควบคุมอัตโนมัติแบบฟัซซี

บทที่ 5 บทวิจารณ์และสรุป จะสรุปผลการดำเนินงาน ปัญหาที่เกิดขึ้น และแนวทางการปรับปรุงพัฒนาโครงงานนี้ต่อไป

บทที่ 2
ทฤษฎีและความรู้ที่เกี่ยวข้อง

จากที่ได้กล่าวในบทที่ 1 แล้วว่า ก่อนที่จะมีการออกแบบและประยุกต์การใช้งานระบบควบคุมอัตโนมัติแบบฟัซซีนั้น จำเป็นต้องศึกษาองค์ประกอบต่างๆที่จำเป็นของระบบควบคุมที่สนใจให้เข้าใจเสียก่อน พบว่าระบบควบคุมอัตโนมัติแบบฟัซซีนั้นมีส่วนที่สำคัญหลายส่วน ดังนั้นในบทนี้จะทำการศึกษา และอธิบายถึงองค์ประกอบต่างๆ ที่จะนำไปใช้งานจริงในการประยุกต์ใช้งานระบบควบคุมอัตโนมัติแบบฟัซซี ซึ่งประกอบด้วยทฤษฎีของฟัซซีลอจิก ระบบควบคุมแบบป้อนกลับ มอเตอร์ไฟฟ้ากระแสตรง และหลักการควบคุมความเร็วของมอเตอร์

2.1 ทฤษฎีของฟัซซีลอจิก

ในการออกแบบระบบควบคุมอัตโนมัติแบบฟัซซีจำเป็นต้องใช้เทคนิคของฟัซซีลอจิก ซึ่งอาศัยความรู้ทางคณิตศาสตร์เกี่ยวกับฟัซซี เพื่อเป็นพื้นฐานในการทำความเข้าใจโครงสร้าง และหลักการทำงาน รวมทั้งเพื่อใช้ในการกำหนดโครงสร้าง และวิธีการออกแบบ ให้เหมาะสมกับลักษณะการประยุกต์ใช้งาน ซึ่งประกอบด้วย

2.1.1 ทฤษฎีฟัซซีเซต

นิยามที่ 1 ฟัซซีเซต คือ เซตของคู่อันดับ u และฟังก์ชันการเป็นสมาชิก (Membership function) $\mu_A(u)$ โดยที่ u เป็นสมาชิกใดๆของเอกภพสัมพัทธ์ U เขียนแทนด้วยสัญลักษณ์ดังนี้

$$A = \{(u, \mu_A(u) \mid u \in U)\}$$

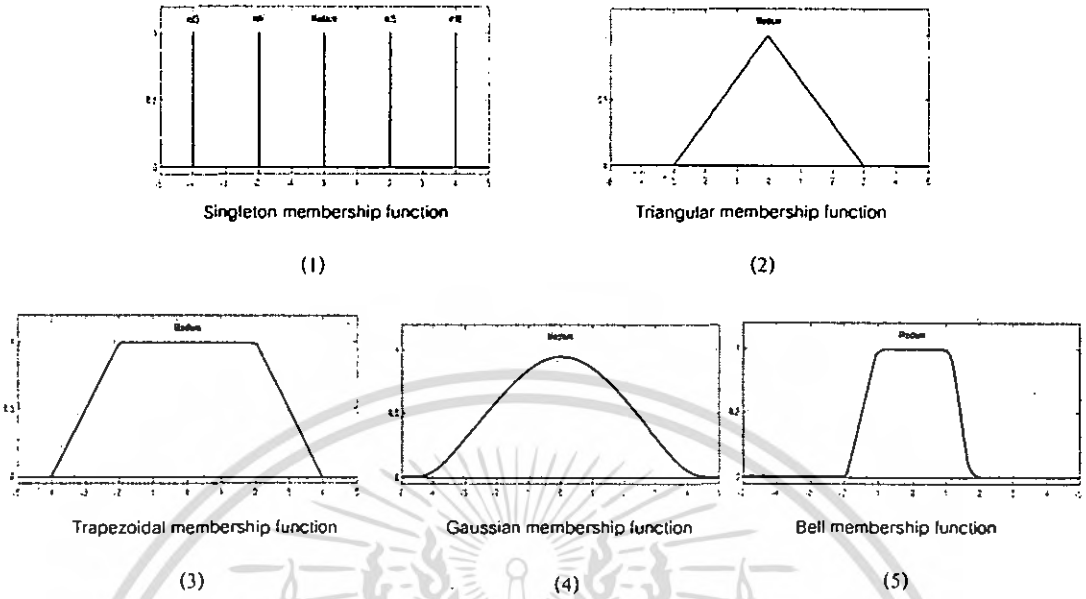
นิยามที่ 2 ฟังก์ชันการเป็นสมาชิกของฟัซซีเซต $\mu_A(u)$ ถูกนิยามให้มีค่าอยู่ภายในช่วง 0 ถึง 1 โดยที่ค่า $\mu_A(u)$ เป็นค่าที่ระบุถึงระดับความเป็นสมาชิกของ u ในฟัซซีเซต

$$\mu_A(u) : U \rightarrow [0,1]$$

โดยปกติจะแบ่งรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตออกเป็น 5 ฟังก์ชันดังนี้

1. ฟังก์ชันการเป็นสมาชิกแบบซิงเกลตัน (Singleton membership function) จะมีสมาชิกเพียงตัวเดียว และมีค่าความเป็นสมาชิก เท่ากับ " 1 "
2. ฟังก์ชันการเป็นสมาชิกรูปสามเหลี่ยม (Triangular membership function)
3. ฟังก์ชันการเป็นสมาชิกรูปสี่เหลี่ยมคางหมู(Trapezoidal membership function)
4. ฟังก์ชันการเป็นสมาชิกแบบเก๊าส์เซียน (Gaussian membership function)
5. ฟังก์ชันการเป็นสมาชิกแบบระฆัง (Bell membership function)

ดังแสดงในรูปที่ 2.1



รูปที่ 2.1 ฟังก์ชันการเป็นสมาชิกของฟัซซีเซตทั้ง 5 แบบ

2.1.2 ฟัซซิฟิเคชัน

ฟัซซิฟิเคชัน (Fuzzification) คือ กระบวนการหาค่าฟัซซีบนเอกภพสัมพัทธ์ U ของอินพุท เพื่อเป็นตัวแทนของอินพุทซึ่งมีค่าเป็นตัวเลข สำหรับใช้เป็นอินพุทของระบบฟัซซี สามารถทำได้ 3 วิธีดังนี้

1. ฟัซซิฟิเคชัน โดยใช้ฟัซซีซิงเกิลตัน จะได้ว่า ฟัซซิฟิเคชัน (u_o) = A โดยที่ A เป็นฟัซซีซิงเกิลตัน

   หรือ $\mu_A(u) = 1 \quad ; \quad u = u_o$

   $\mu_A(u) = 0 \quad ; \quad u \neq u_o$

2. ฟัซซิฟิเคชัน โดยใช้ฟัซซีนัมเบอร์ จะได้ว่าฟัซซิฟิเคชัน (u_o)=A โดยที่ A เป็นฟัซซีนัมเบอร์

   หรือ $\mu_A(u) = 1 \quad ; \quad u = u_o$

   $\mu_A(u)$ ลดลง ; u ห่างออกไปจาก u_o

3. ฟัซซิฟิเคชัน โดยใช้ไฮบริดนัมเบอร์ จะได้ว่า ฟัซซิฟิเคชัน (u_o) = A โดยที่ A เป็นไฮบริดนัมเบอร์

   (ไฮบริดนัมเบอร์ คือ ฟัซซีนัมเบอร์ที่มีสมาชิกในเอกภพสัมพัทธ์เป็นแรนดอมนัมเบอร์)

   หรือ $\mu_A(u) = 1 \quad ; \quad u = u_o$

   $\mu_A(u)$ ลดลง ; u ห่างออกไปจาก u_o

2.1.3 ดีฟัซซิฟิเคชัน

ดีฟัซซิฟิเคชัน (Defuzzification) คือกระบวนการหาค่าเอาท์พุทเพียงค่าเดียว ที่เหมาะสมที่สุด เพื่อเป็นตัวแทนของฟัซซีเอาท์พุทที่มีค่าความเป็นไปได้ กระจายอยู่บนเอกภพสัมพัทธ์ V ของเอาท์พุท ซึ่งมีวิธีการที่หลากหลายดังนี้

1. Max Procedure เป็นวิธีดีฟัซซิฟิเคชัน โดยเลือกค่าเอาท์พุทที่มีค่าฟังก์ชันการเป็นสมาชิกมากที่สุดเพื่อใช้เป็นตัวแทนของฟัซซีเอาท์พุท
2. Mean of Maximum (MOM) เป็นวิธีที่ขยายมาจากวิธี Max Procedure เพื่อใช้ในกรณีที่ค่าเอาท์พุทมีค่าฟังก์ชันการเป็นสมาชิกสูงสุดเท่ากับหลายค่า ให้ทำการหาค่าเฉลี่ยของค่าเอาท์พุทที่มีค่าฟังก์ชันการเป็นสมาชิกสูงสุดเท่านั้น
3. Center of Area (COA) / Center of Gravity (COG) วิธี COA เป็นวิธีหาจุดศูนย์กลางของพื้นที่ใต้กราฟของฟังก์ชันการเป็นสมาชิก เพื่อใช้เป็นตัวแทนของฟัซซีเอาท์พุท ซึ่งทำได้โดยการแบ่งพื้นที่ใต้กราฟของฟังก์ชันการเป็นสมาชิกออกเป็น 2 ส่วนเท่าๆกัน และค่า v_o จะหาได้จากค่า v ที่ตำแหน่งของเส้นแบ่งครึ่ง
สามารถแสดงด้วยสมการดังนี้

$$\int_{-\infty}^{COA\,(B)} \mu_B(v)dv = \int_{COA\,(B)}^{\infty} \mu_B(v)dv$$

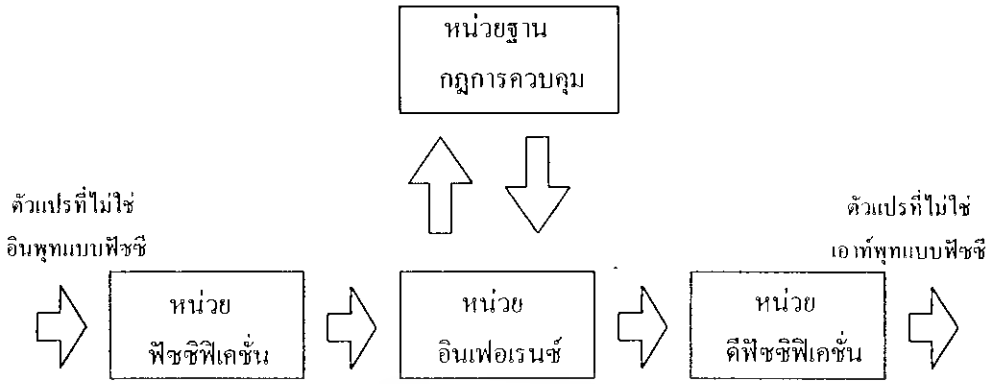
ประมาณได้เป็น $$COG\ (B) = \frac{\int v\mu_B(v)dv}{\int \mu_B(v)dv}$$

ในกรณีเอกภพสัมพัทธ์ V มีสมาชิกเป็นค่าไม่ต่อเนื่อง สามารถหาค่า v_o ได้จากสมการดังนี้

$$COG\ (B) = \frac{\sum_{i=1}^{N_v} v_i \mu_s(v_i)}{\sum_{i=1}^{N_v} \mu_s(v_i)}$$

2.1.4 ตัวควบคุมฟัซซี

จากเทคนิคของฟัซซีลอจิก จำเป็นต้องอาศัยความรู้ทางคณิตศาสตร์เกี่ยวกับฟัซซี จะถูกนำมาใช้ในการออกแบบจำลองฟัซซีโดยโครงสร้างของตัวควบคุมฟัซซีประกอบไปด้วย 4 หน่วยหลัก คือ หน่วยฟัซซิฟิเคชัน (Fuzzification unit) หน่วยฐานกฎการควบคุม (Fuzzy rule base) หน่วยอินเฟอเรนซ์ (Inference unit) และหน่วยดีฟัซซิฟิเคชัน (Defuzzification unit) ซึ่งการทำงานของแต่ละหน่วย สามารถอธิบายดังรูปที่ 2.2



รูปที่ 2.2 โครงสร้างพื้นฐานของตัวควบคุมฟัซซี

2.1.5 หน่วยฐานกฎการควบคุมฟัซซี

หน่วยฐานกฎการควบคุมฟัซซี เป็นหน่วยที่รวบรวมกฎการควบคุมแบบฟัซซีซึ่งอยู่ในรูปแบบ " ถ้า...แล้ว... " หรือ " IF....THEN... " โดยกฎการควบคุมอยู่ในลักษณะดังนี้

$$\text{IF } X_1 \text{ is } X_1^k \text{ and } \ldots \text{ and } X_m \text{ is } X_m^k \text{ THEN } Y \text{ is } Y^k$$

หรือ $$\text{IF } X \text{ is } X^k \text{ THEN } Y \text{ is } Y^k \;;\; k = 1,2,3,\ldots, m$$

โดยที่ X คือ ตัวแปรสถานะของกระบวนการ ซึ่งใช้เป็นอินพุทของตัวควบคุมฟัซซี

โดย $X = [X_1, \ldots, X_m]$

X^k คือ ฟัซซีเซตของตัวแปรอินพุทในส่วนเหตุของกฎการควบคุมที่ k

โดย $X^k = X_1^k \times X_m^k$

m คือ จำนวนตัวแปรอินพุทของตัวควบคุม

Y คือ ตัวแปรเอาท์พุทของตัวแปรควบคุมฟัซซี

Y^k คือ ฟัซซีเซตของตัวแปรเอาท์พุทในส่วนผลของกฎการควบคุมที่ k

M คือ จำนวนกฎการควบคุมทั้งหมดในฐานกฎการควบคุม

ซึ่งจากคำจำกัดความข้างต้น สามารถอธิบายได้ว่ากฎการควบคุม จะถูกกำหนดตามเงื่อนไข " ถ้า...แล้ว... " เช่น

IF X is hot THEN Y is zero

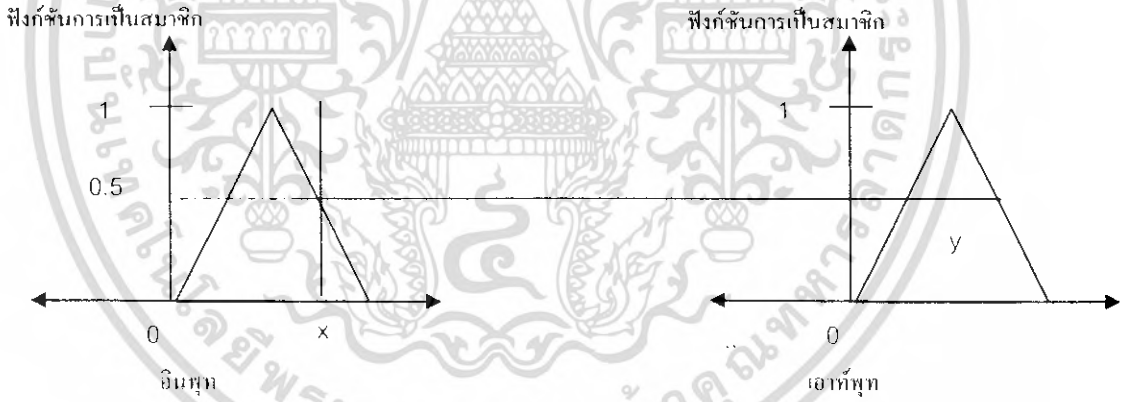
โดยที่ X และ Y เป็นตัวแปรฟัซซีเซต ส่วน hot และ zero เป็นส่วนเหตุและผลที่สอดคล้องกับตัวแปร X และ Y ประพจน์ของฟัซซีเซตคือ X is hot เป็นส่วนเหตุ (Antecedent) ซึ่งก็คือ ส่วนของอินพุทของแบบจำลองฟัซซีลอจิก ส่วนประพจน์ที่ตามหลัง THEN คือ Y is zero เป็นส่วนของผล (Consequent)

สังเกตว่าส่วนเหตุของแบบจำลองฟัซซีลอจิก อาจจะประกอบด้วยจำนวนประพจน์หลายประพจน์ ซึ่งจำนวนของประพจน์จะขึ้นอยู่กับจำนวนตัวแปรอินพุทและจำนวนของเหตุและผลแต่ละตัว เช่น ถ้าอินพุทมีสองตัวแปรในส่วนเหตุ จะมีสองประพจน์เชื่อมต่อกันในกฎหนึ่งกฎ และในการเชื่อมต่อกันของประพจน์จะต้องมีตัวเชื่อมซึ่งในแบบจำลองฟัซซีจะมีอยู่สองตัวด้วยกัน คือ ยูเนียน (OR) และอินเตอร์เซกชัน (AND) เช่น

ในกรณีที่เชื่อมด้วยยูเนียน IF X is hot or M is high THEN Y is zero

และในกรณีที่เชื่อมด้วยอินเตอร์เซกชัน IF X is hot and M is high THEN Y is zero

สำหรับจำนวนกฎในแบบจำลองฟัซซีลอจิกของระบบหนึ่งๆ จะขึ้นอยู่กับตัวแปรอินพุทและจำนวนเทอมเซตของแต่ละตัวแปร เช่นถ้าอินพุทมีสองตัวแปร และตัวแปรแต่ละตัวมี 5 เทอมเซต จำนวนกฎทั้งหมดจะเท่ากับ 25 กฎ ซึ่งจากกฎการควบคุมที่ถูกกำหนดขึ้นมาจากความชำนาญนี้ จะใช้เป็นส่วนหนึ่งของกระบวนการ การอินเฟอเรนซ์แบบฟัซซี (Fuzzy Inference) เพื่อหาระดับความเป็นสมาชิกของแต่ละกฎจากความสำคัญระหว่างตัวแปรอินพุทแบบฟัซซี X กับ ฟัซซีเซตในส่วนเหตุของแต่ละกฎ X^k แล้วจึงคำนวณหาค่าเอาท์พุทแบบฟัซซีจากระดับการเป็นสมาชิกกับฟัซซีเซตในส่วนผลของกฎแต่ละกฎ Y^k

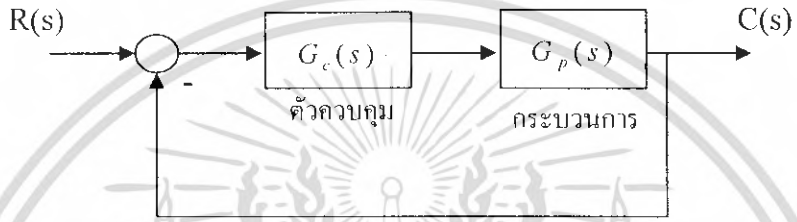


รูปที่ 2.3 การอินเฟอเรนซ์แบบฟัซซี

จากรูปที่ 2.3 ถ้าส่วนกำหนดการควบคุมกำหนดไว้ว่า ถ้ามีข้อมูลอินพุทเข้ามาที่เซต X ค่าเอาท์พุทที่ได้จะต้องเป็นเซต Y ซึ่งหลักฐานนี้เองทำให้หน่วยเอนเฟอเรนซ์สามารถที่จะสรุปหาข้อสรุปออกมาได้โดยให้อินพุท x อยู่ในช่วงสมาชิกของเซต X จึงจะสามารถระบุได้เลยว่าข้อสรุปที่ได้จะต้องเป็นสมาชิกของเซต Y ซึ่งในรูปจะเห็นได้ว่าการที่สมาชิกแต่ละตัวมีระดับการเป็นสมาชิกไม่เท่ากันจึงทำให้ข้อสรุปจะต้องขึ้นอยู่กับระดับการเป็นสมาชิกของเซตด้วย

2.2 ระบบควบคุมแบบป้อนกลับ

ระบบควบคุมแบบป้อนกลับ (Feedback Control System) คือระบบควบคุมที่มีการนำเอาเอาท์พุทที่ได้จากกระบวนการกลับเข้ามาเป็นส่วนหนึ่งของข้อมูล ที่จะนำมาพิจารณา เป็นอินพุทที่จะให้กับกระบวนการนั้นโดยมักมีโครงสร้างดังรูปที่ 2.4 การที่จะทราบค่าเอาท์พุทได้จะต้องมีการวัดข้อมูลของเอาท์พุท เมื่อทราบค่าเอาท์พุทแล้วจะนำค่าเอาท์พุทที่ได้ไปเปรียบเทียบกับค่าอ้างอิงที่ต้องการ เพื่อหาความแตกต่างระหว่างค่าที่ได้จริงกับค่าที่ต้องการ และส่งสัญญาณไปสู่ตัวควบคุม ซึ่งจะประมวลผลเพื่อหาอินพุทที่จะป้อนให้กระบวนการ



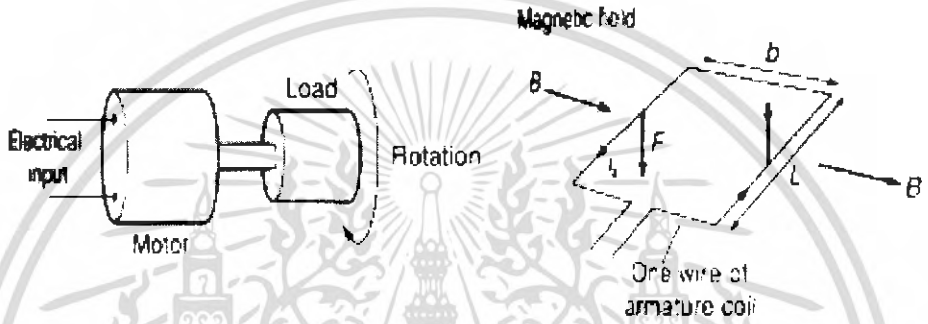
รูปที่ 2.4 บล็อกไดอะแกรมของระบบควบคุมแบบป้อนกลับ

2.3 มอเตอร์ไฟฟ้ากระแสตรง

มอเตอร์ไฟฟ้ากระแสตรง (DC motor) คือ อุปกรณ์ที่ใช้เปลี่ยนพลังงานไฟฟ้าเป็นพลังงานกล เพื่อนำพลังงานกลที่ได้ไปขับเคลื่อนสิ่งต่างๆตามต้องการ มอเตอร์ไฟฟ้ากระแสตรงนั้น จะใช้การขับเคลื่อนในแบบที่มีอัตราเร็วไม่สูงมากนัก เนื่องจากมอเตอร์ไฟฟ้ากระแสตรงนั้นมีแรงบิดเริ่มต้นที่สูง (Starting torque) สามารถควบคุมควบคุมอัตราเร็วได้ค่อนข้างง่าย แต่มีข้อเสียคือมีโครงสร้างที่ค่อนข้างซับซ้อนมาก จึงจำไม่เหมาะที่จะใช้ในงานที่มีอัตราเร็วค่อนสูงมากๆ แรงทางกลที่เกิดขึ้นก็อาศัยหลักการที่ว่าเมื่อมีกระแสไหลในตัวนำซึ่งอยู่ในสนามแม่เหล็กก็จะทำให้เกิดแรงดันไฟฟ้าเหนี่ยวนำเกิดขึ้นในขดลวดตัวนั้น และขดลวดตัวนำจะหมุนไปตามสนามแม่เหล็กที่เกิดขึ้น โดยการดูดของขั้วแม่เหล็กที่ต่างกันและผลักกันระหว่างขั้วแม่เหล็กที่เหมือนกัน ซึ่งแรงดันไฟฟ้าเหนี่ยวนำที่เกิดขึ้นจะทำให้เพลาของมอเตอร์หมุน ก็จะได้พลังงานกลไปใช้งาน แต่ก็จะมีแรงดันไฟฟ้าเหนี่ยวนำเกิดขึ้นในทิศทางตรงกันข้ามกับแรงดันไฟฟ้าที่จ่ายจากภายนอก แรงดันไฟฟ้าเหนี่ยวนำที่เกิดขึ้นนี้เรียกว่า แรงดันไฟฟ้าด้านกลับ (Counter e.m.f หรือ back e.m.f.)

รูปที่ 2.5 การเปลี่ยนพลังงานไฟฟ้าเป็นพลังงานกล

2.3.1 หลักการทำงานของมอเตอร์ไฟฟ้ากระแสตรง

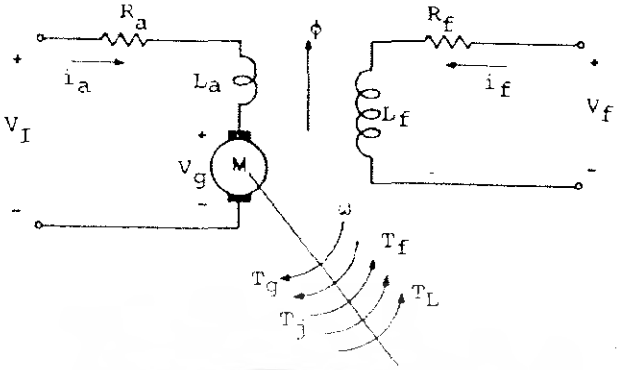


รูปที่ 2.6 การทำงานของมอเตอร์ไฟฟ้ากระแสตรง

รูปที่ 2.6 อธิบายโครงสร้างของมอเตอร์ไฟฟ้ากระแสตรงอย่างง่าย ซึ่งประกอบด้วยขดลวดที่วางอยู่ระหว่างขั้วแม่เหล็ก โดยปลายของขดลวดทั้งสองข้างต่อเข้ากับคอมมิวเตเตอร์ (Commutator) ด้านละซี่ ซึ่งจะมีแปรงถ่านต่อไว้และแปรงถ่านทั้งสองต่อเข้ากับแหล่งจ่ายไฟฟ้ากระแสตรงจากภายนอก ขดลวดตัวนำนั้นจะต้องหมุนอยู่ในสนามแม่เหล็กเมื่อมีการผ่านกระแสไฟฟ้าเข้าไปยังขดลวดในสนามแม่เหล็กจะทำให้เกิดแรงแม่เหล็กซึ่งมีสัดส่วนของแรงขึ้นกับกระแสแรงของสนามแม่เหล็ก โดยแรงจะเกิดขึ้นเป็นมุมฉากกับกระแสและสนามแม่เหล็ก ขณะที่ทิศทางของแรงกลับตรงกันข้ามกัน ถ้าหากกระแสของสนามแม่เหล็กไหลย้อนกลับจะทำให้เกิดการเปลี่ยนแปลงของกระแส และ สนามแม่เหล็กเป็นผลทำให้ทิศทางของแรงเปลี่ยนไปด้วยคุณสมบัตินี้ทำให้มอเตอร์ไฟฟ้ากระแสตรงกลับทิศทางการหมุนได้

สนามแม่เหล็กของมอเตอร์ส่วนหนึ่งเกิดขึ้นจากแม่เหล็กถาวร ซึ่งจะถูกยึดติดกับแผ่นเหล็กหรือ เหล็กกล้า โดยปกติส่วนนี้จะเป็นส่วนที่ยึดอยู่กับที่ และ ขดลวดเหนี่ยวนำจะพันอยู่กับส่วนที่เป็นแกนหมุนของมอเตอร์

2.3.2 คุณสมบัติของมอเตอร์ไฟฟ้ากระแสตรง



รูปที่ 2.7 วงจรภายในของมอเตอร์ไฟฟ้ากระแสตรง

จากรูปที่ 2.7 จะได้ว่าแรงดันไฟฟ้าที่มีผลต่อการใช้งานจริงในอาร์เมเจอร์ของมอเตอร์จึงมีค่าเท่ากับแรงดันไฟฟ้าที่จ่ายให้ลบด้วยแรงดันไฟฟ้าต้านกลับ เขียนเป็นสมการได้คือ

$$I_a R_a = V - E_b$$

หรือ

$$E_b = V - I_a R_a$$

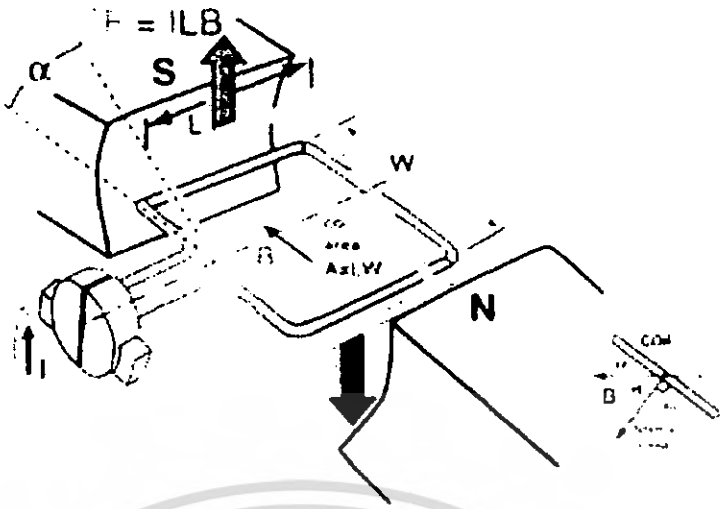
เมื่อ E_b คือ แรงดันไฟฟ้าต้านกลับ (V)

V คือ แรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์ (V)

I_a คือ กระแสที่ไหลในอาร์เมเจอร์ (A)

R_a คือ ความต้านทานของขดลวดอาร์เมเจอร์ (Ω)

ค่าแรงดันไฟฟ้าต้านกลับนี้จะมีค่าไม่เท่ากับแรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์ แรงดันไฟฟ้าต้านกลับที่เกิดขึ้น สามารถเขียนเปรียบเทียบได้เหมือนกับในอาร์เมเจอร์นั้นมีแหล่งจ่ายไฟฟ้ากระแสตรงซ่อนอยู่ และจ่ายไฟออกมาตรงกันข้ามกับแรงดันไฟฟ้าที่จ่ายเข้าไปเมื่อมีกระแสไหลผ่านขดลวดอาร์เมเจอร์ที่วางอยู่ในสนามแม่เหล็ก เกิดแรงไฟฟ้าเหนี่ยวนำ ทำให้มีแรงผลักดันที่อาร์เมเจอร์ เกิดแรงบิดที่สม่ำเสมอราบเรียบ ไม่กระตุกเพราะมีขดลวดหลายๆชุดที่อยู่ติดกันผลัดกันทำงานทีละชุดตามลำดับ



รูปที่ 2.8 แรงที่กระทำกับขดลวด

จากรูปที่ 2.8 แรงที่กระทำบนขดลวดตัวนำต่อหนึ่งขดในขณะที่มีกระแสไหลผ่านตัวนำนั้น จะเป็นสัดส่วนโดยตรงกับจำนวนกระแสที่ไหล ความเข้มของสนามแม่เหล็กและความยาวของตัวนำในส่วนที่ตัดผ่านสนามแม่เหล็ก (หรือความยาวของตัวนำส่วนที่ใช้งานจริง) สามารถเขียนเป็นสมการได้ดังนี้

$$F = B\, I_a\, L$$

เมื่อ F คือ แรงที่กระทำที่ขดลวดตัวนำ (N)

B คือ ความหนาแน่นของเส้นแรงแม่เหล็ก (W_b/m^2)

L คือ ความยาวของตัวนำในส่วนที่ผ่านสนามแม่เหล็ก (m)

I_a คือ กระแสที่ไหลในขดลวดตัวนำ (A)

แรงที่กระทำ F นี้จะยังส่งผลให้เกิดแรงบิดซึ่ง แรงบิด (Torque) คือการหมุนหรือการบิดของโมเมนต์ของแรงๆหนึ่ง ที่กระทำรอบๆแกนของมอเตอร์ มีขนาดเท่ากับผลคูณของแรงกันรัศมีเขียนเป็นสมการได้ดังนี้

$$T = F\, r$$

เมื่อ T คือ แรงบิดที่เกิดขึ้น (N–m)

r คือ ระยะห่างระหว่างจุดศูนย์กลางของเพลาถึงตัวนำ (M)

คุณลักษณะการทำงานของมอเตอร์ไฟฟ้ากระแสตรงเมื่อพิจารณาในกรณีความสัมพันธ์ของแรงบิดกันกระแสที่ไหลในอาร์เมเจอร์ จะได้ว่าแรงบิดที่เกิดขึ้นจากมอเตอร์ไฟฟ้ากระแสตรงนั้นขึ้นอยู่กับกระแสที่ไหลในอาร์เมเจอร์และเส้นแรงแม่เหล็กที่เกิดขึ้นจากฟิลด์คอยล์หรือจากขั้วแม่เหล็กถาวรสามารถเขียนเป็นความสัมพันธ์ได้ คือ

$$T = K_e \Phi I_a$$

เมื่อ K_e คือ ค่าคงที่ของมอเตอร์

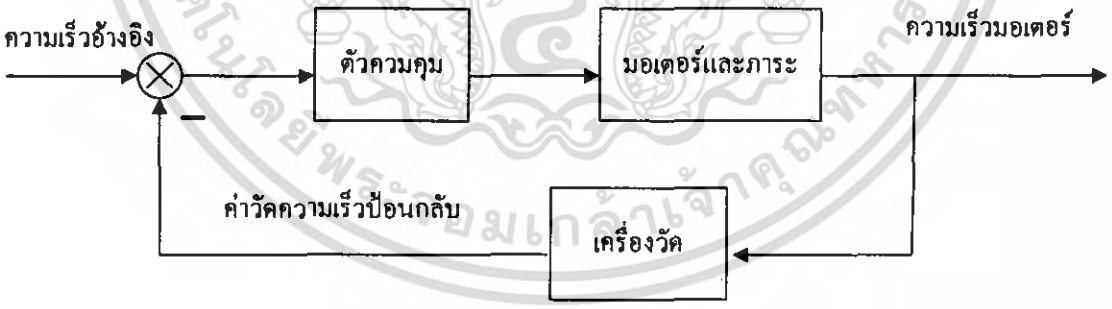
Φ คือ เส้นแรงแม่เหล็ก (W_b)

2.4 การควบคุม ความเร็วของมอเตอร์ไฟฟ้ากระแสตรง

2.4.1 หลักการควบคุมความเร็ว

รูปที่ 2.9 แสดงบล็อกไดอะแกรมของระบบควบคุมความเร็วของ มอเตอร์แบบเซอร์โว (Servo motor) เครื่องวัดรอบในลูปแบบป้อนกลับจะวัดความเร็วของมอเตอร์แบบเซอร์โว และส่งป้อนกลับมาในรูปของสัญญาณไฟฟ้า (แรงดัน หรือ กระแส) ซึ่งแปรตามความเร็วเพลาของมอเตอร์ ในที่นี้ลูปป้อนกลับจะทำให้ความเร็วเอาท์พุทของมอเตอร์มีค่าคงที่มากขึ้น

ระบบควบคุมแบบป้อนกลับ ถูกใช้รวมอยู่ในระบบเครื่องมือ เครื่องจักร หรือ สว่านกำลัง เพื่อชดเชยความแตกต่างของโหลด (วัสดุที่ถูกตัดหรือถูกเจาะ) วัสดุเนื้อแข็งจะหน่วงความเร็วของสว่าน หรือ เครื่องมือที่ใช้ในการตัด และ สว่านจะมีความเร็วเพิ่มขึ้นในวัสดุเนื้ออ่อนในลูปป้อนกลับความเร็วของมอเตอร์ ที่ประกอบด้วยเครื่องวัดความเร็ว ความเร็วของเครื่องมือจะยังคงมีค่าคงที่ เนื่องจากเมื่อเครื่องมือที่ใช้ในการตัดมีความเร็วลดลง สัญญาณป้อนกลับจะควบคุมมอเตอร์ให้เพิ่มความเร็วขึ้น ในขณะเดียวกัน เมื่อเครื่องมือตัดชิ้นงานที่เป็นวัสดุอ่อน ลูปป้อนกลับจะป้องกันไม่ให้มอเตอร์เร่งความเร็วเกินขนาด



รูปที่ 2.9 ระบบควบคุมความเร็ว ที่ประกอบด้วยลูปการควบคุมป้อนกลับเพียงลูปเดียว

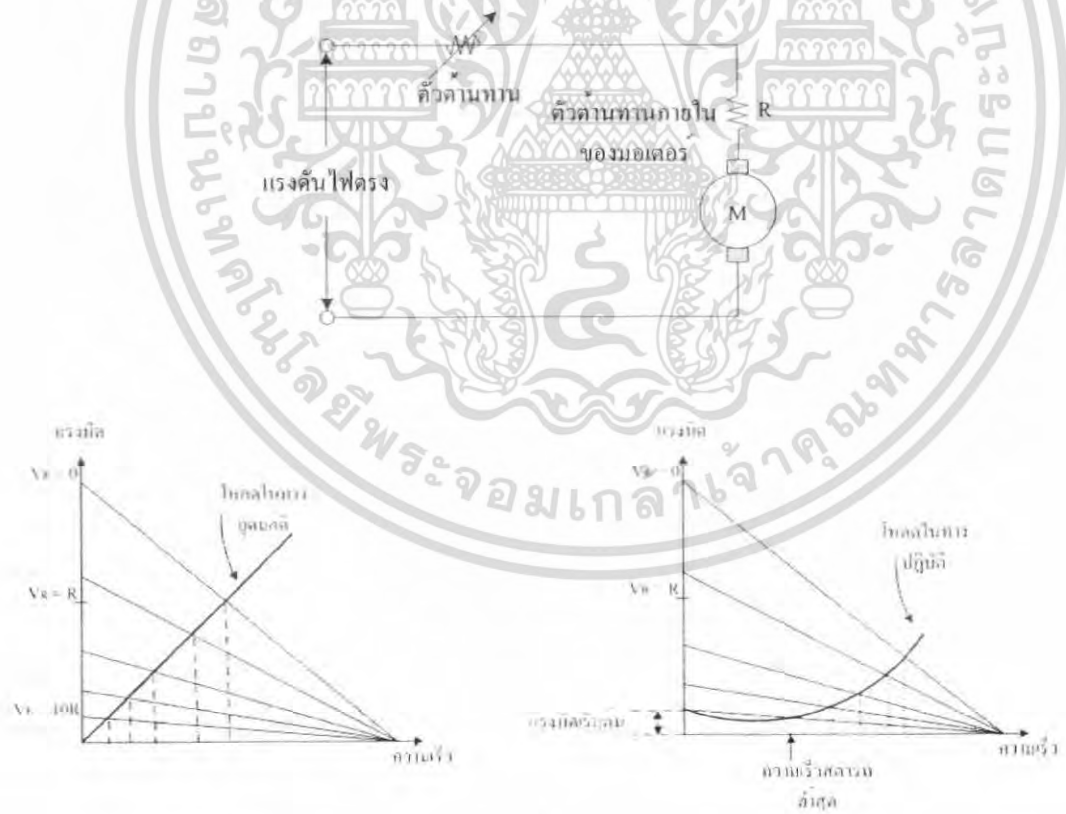
2.4.2 ประเภทของการควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง

แบ่งออกเป็น 3 ประเภทดังนี้

1. การควบคุมด้วยตัวต้านทานที่ปรับค่าได้
2. การควบคุมด้วยวิธีเปลี่ยนค่าแรงดัน
3. การควบคุมความกว้างของพัลส์

2.4.2.1 การควบคุมด้วยตัวต้านทานที่ปรับค่าได้

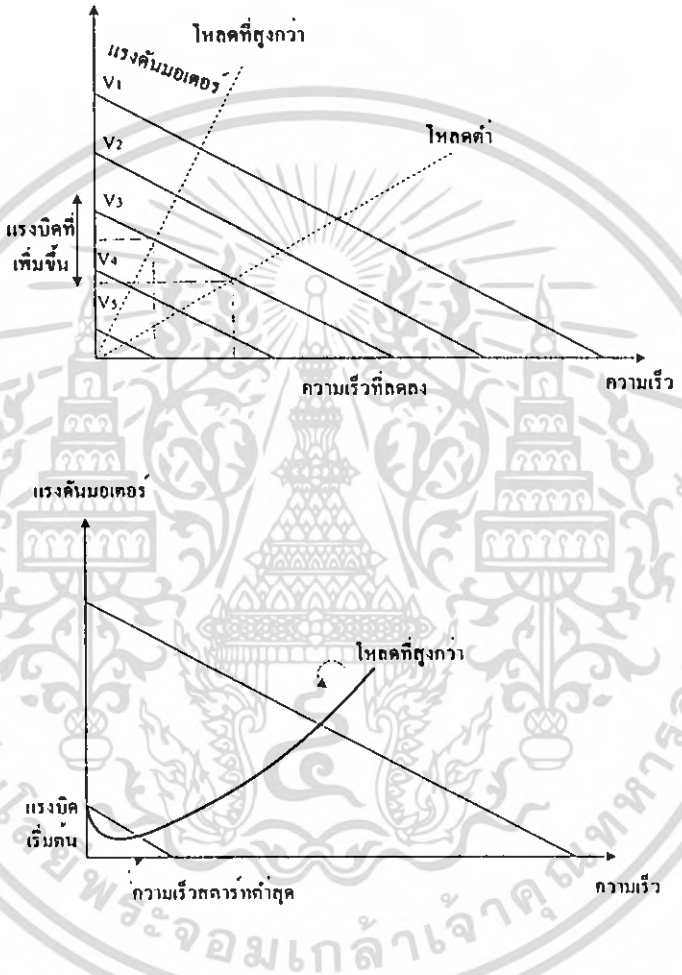
เป็นรูปแบบพื้นฐานที่สุดของการควบคุมมอเตอร์คือ ใช้ตัวต้านทานปรับค่าได้อนุกรมกับมอเตอร์ โดยตัวต้านทานที่ปรับค่าได้จะเป็นตัวกำหนดความเร็วในการหมุนของมอเตอร์ การบังคับแบบนี้ไม่มีประสิทธิภาพเพราะกำลังไฟสูญเสียไปในตัวความต้านทาน มักนิยมใช้กับมอเตอร์ตัวเล็กๆ การบังคับแบบนี้ให้คุณสมบัติการสตาร์ทดี (ให้แรงบิดสูงที่ความเร็วต่ำ) แต่จะให้ความเร็วสูงมากเมื่อมอเตอร์อยู่ในภาวะที่มีโหลดน้อยๆ ดังนั้นการบังคับแบบนี้มีประโยชน์เฉพาะภาวะที่แรงต้านคงที่ เช่น การบังคับความเร็วของเครื่องจักรเย็บผ้า เป็นต้น



รูปที่ 2.10 วงจรควบคุมความเร็วของมอเตอร์กระแสตรง
แบบใช้ตัวต้านทานอนุกรม และกราฟแสดงคุณสมบัติ

2.4.2.2 การควบคุมด้วยวิธีเปลี่ยนค่าแรงดัน

วิธีการนี้ดีกว่าวิธีการแรกแต่จะซับซ้อนกว่าต้องใช้อุปกรณ์อิเล็กทรอนิคส์ที่ทีอัตราขยายกำลังสูง และ มอเตอร์จะถูกป้อนด้วยแรงดันที่เปลี่ยนแปลงค่าได้ จากแหล่งจ่ายที่มีอิมพีแดนซ์ต่ำ ข้อดีของการควบคุมวิธีนี้คือ ถ้าความเร็วลดลงจากผลของแรงบิด แรงดันที่ป้อนให้กับมอเตอร์จะเพิ่มขึ้นเพื่อรักษาระดับความเร็ว ส่วนข้อเสียจากการควบคุมวิธีนี้คือ เมื่อมอเตอร์มีความเร็วต่ำ แรงดันที่ป้อนให้กับมอเตอร์จะมีค่าต่ำเช่นกัน



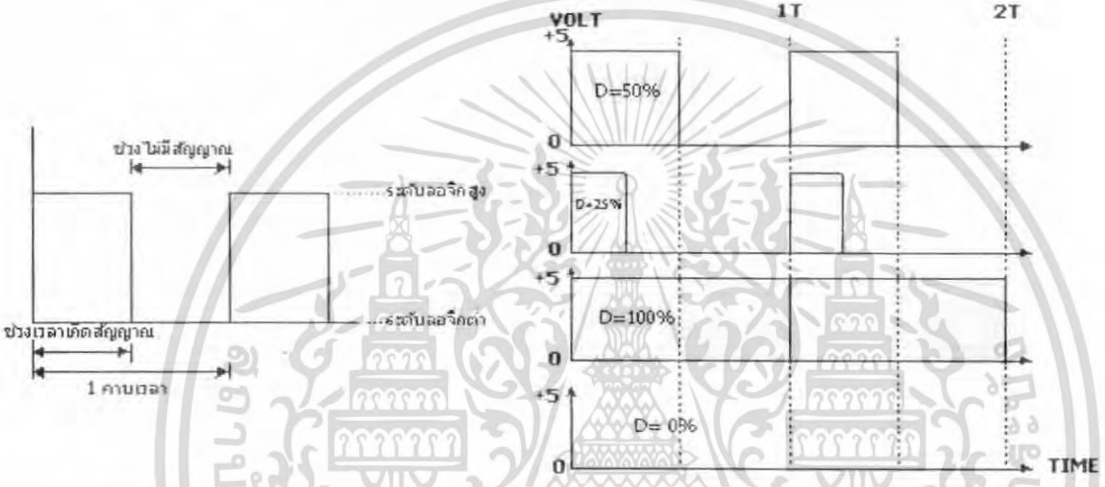
รูปที่ 2.11 การควบคุมความเร็วโดยเปลี่ยนค่าแรงดัน

2.4.2.3 การควบคุมความเร็วของมอเตอร์แบบการควบคุมความกว้างของสัญญาณพัลส์

การควบคุมความกว้างของพัลส์ (Pulse Width Modulation: PWM) คือ การปรับเปลี่ยนที่สัดส่วน และความกว้างของสัญญาณพัลส์ โดยความถี่ของสัญญาณพัลส์จะไม่มีการเปลี่ยนแปลง หรือเป็นการเปลี่ยนแปลงที่ค่าของดิวตี้ไซเคิล (Duty cycle) นั่นเอง ซึ่งค่าของดิวตี้ไซเคิล คือช่วงความกว้างของพัลส์ที่มีสถานะลอจิกสูง โดยคิดสัดส่วนเป็นเปอร์เซนต์จากความกว้างของพัลส์ทั้งหมด ยกตัวอย่างเช่น ถ้าหากค่าดิวตี้ไซเคิลมีค่าเท่ากับเท่ากับ 50% ก็หมายถึงในแต่ละรูปสัญญาณ

พัลส์จะมีช่วงของสัญญาณที่เป็นสถานะลอจิกสูงอยู่ครึ่งหนึ่ง และสถานะลอจิกต่ำอยู่อีกครึ่งหนึ่ง ดังรูปที่ 2.12 และในทำนองเดียวกันถ้าหากค่าดิวตี้ไซเคิลมีค่ามาก หมายความว่าความกว้างของพัลส์ที่เป็นสถานะลอจิกสูงจะมีความกว้างมากขึ้น หากค่าดิวตี้ไซเคิลมีค่าเท่ากับ 100% ก็หมายความว่าจะไม่มีสถานะลอจิกต่ำเลย ซึ่งค่าดิวตี้ไซเคิล สามารถหาได้จากค่าความสัมพันธ์ดังนี้

$$\text{ค่าดิวตี้ไซเคิล} = \frac{\text{ช่วงของสัญญาณพัลส์}}{\text{คาบเวลาทั้งหมดของสัญญาณ}} \times 100\%$$



รูปที่ 2.12 สัญญาณ PWM ซึ่งแสดงค่าดิวตี้ไซเคิล ที่ต่างๆกัน

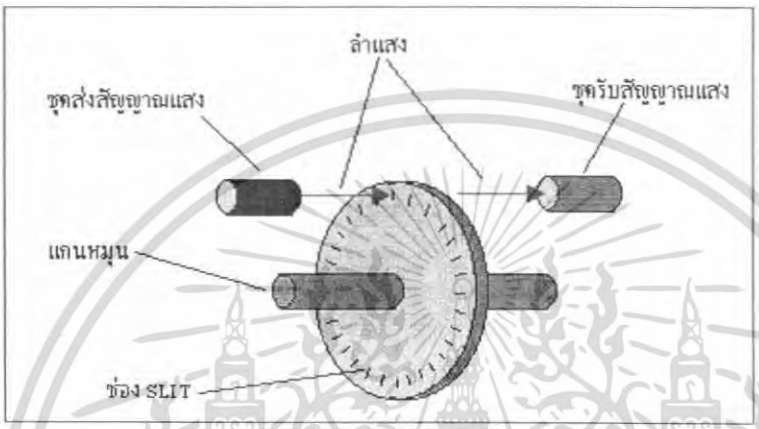
จะเห็นได้ว่ามีข้อสำคัญ 3 ประการในการเลือกใช้การควบคุมความเร็วของมอเตอร์แบบการควบคุมความกว้างของพัลส์ คือ

1. ง่ายในการอินเตอร์เฟสกับไมโครคอนโทรลเลอร์ และ ใช้เพียงแค่เอาท์พุทสัญญาณเดียวในการควบคุมความเร็ว
2. มีประสิทธิภาพ คือ แหล่งจ่ายพลังงาน (Power supply) จะจ่ายกำลังได้เต็มที่ทั้ง สถานะลอจิกสูง (Full on) และสถานะลอจิกต่ำ (Full off)
3. ทำให้ได้ค่า แรงบิดและ ความเร็ว สูงสุดของมอเตอร์ เป็นเพราะแหล่งจ่ายพลังงานจะจ่ายกำลังได้เต็มที่ทั้งสถานะลอจิกสูงและสถานะลอจิกต่ำ

ดังนั้นในโครงงานนี้ จึงเลือกใช้การควบคุมความเร็วมอเตอร์แบบการควบคุมความกว้างของพัลส์

2.5 เอ็นโค้ดเดอร์

เอ็นโค้ดเดอร์ (Encoder) มีลักษณะเป็นแผ่นกลมมีแกนอยู่ตรงกลาง และที่แผ่นกลม จะมีช่องเล็ก ที่แสงสามารถ ส่องผ่านได้ เป็นจำนวนมาก เรียกช่องนี้ว่า ช่องสลิท (Slit) ซึ่งที่ด้านหนึ่งของแผ่นกลม นี้ จะมีตัวส่งแสงอินฟาเรด (Infrared) ไปยังตัวรับสัญญาณแสงอินฟาเรด ซึ่งจะอยู่ในด้านตรงกันข้าม ดังรูปที่ 2.13



รูปที่ 2.13 โครงสร้างของเอ็นโค้ดเดอร์

เมื่อหมุนแกนหมุนทำให้แผ่นกลมหมุนไปตัดลำแสงอินฟาเรด ดังนั้นชุดรับแสงอินฟาเรดจึงมีแสงมากระทบเป็นช่วงๆ ตามจังหวะที่แสงผ่านช่องสลิท จึงทำให้ สัญญาณเอาท์พุทของ ชุดรับแสงอินฟาเรด มีลักษณะเป็นสัญญาณพัลส์ ดังรูปที่ 2.14

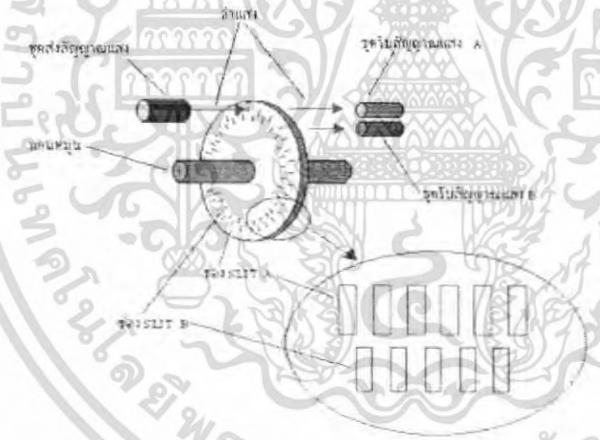
รูปที่ 2.14 การสร้างพัลส์ ของเอ็นโค้ดเดอร์

จำนวนพัลส์ที่ได้ออกมานี้ จะเป็นตัวที่ชี้บ่งว่ามอเตอร์หมุนไปกี่องศา หรือกี่รอบ ซึ่งสามารถคำนวณได้จากความสัมพันธ์

$$\text{จำนวนรอบที่มอเตอร์หมุนไป} = \frac{\text{จำนวนพัลส์}}{\text{ความละเอียดของเอ็นโค้ดเดอร์}}$$

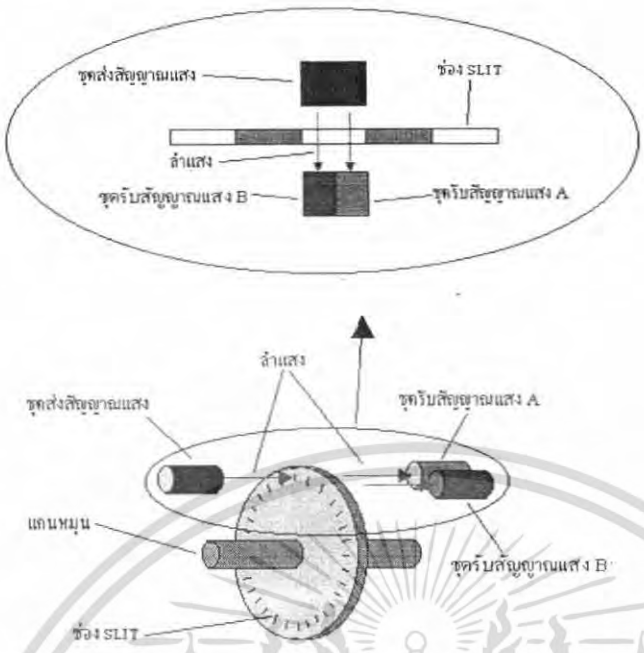
โดยที่ค่าความละเอียดของเอ็นโค้ดเดอร์ใช้หน่วยเป็นจำนวนพัลส์ต่อรอบ (pulse/round: ppr)เช่น 1000 ppr ก็หมายถึงว่าเมื่อมอเตอร์หมุนไป 1 รอบจะมีสัญญาณพัลส์ออกมา 1000 พัลส์ เป็นต้น

ส่วนการตรวจสอบทิศทางการหมุนของมอเตอร์ สามารถทราบได้จากวิธีสร้างช่องสลิทเป็น 2 ชุดเหลื่อมกัน 90 องศา ดังรูปที่ 2.15



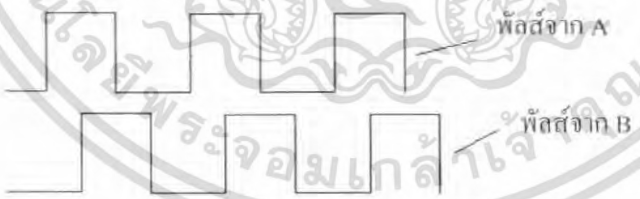
รูปที่ 2.15 เอ็นโค้ดเดอร์ที่มีช่องสลิทสองชุดเหลื่อมกัน 90 องศา

หรืออาจจะใช้ช่องสลิทเพียง 1 ชุดแต่มีการจัดวางชุดรับสัญญาณแสงดังรูปที่ 2.16 แต่ข้อสำคัญคือ ต้องมีเฟสต่างกัน 90 องศา



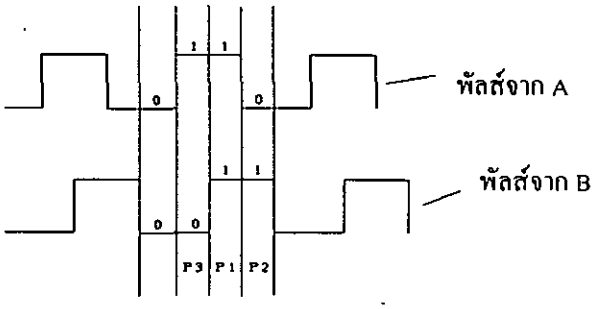
รูปที่ 2.16 เอ็นโค้ดเดอร์ที่มีชุดรับสัญญาณแสงที่มีเฟสต่างกัน 90 องศา

ดังนั้นสัญญาณเอาท์พุทจากมอเตอร์จึงมี 2 ชุด คือ A และ B โดยที่สัญญาณพัลส์จาก A และ B จะเหลื่อมกัน 90 องศาด้วย อาจกล่าวได้ว่าสัญญาณเอาท์พุทจากมอเตอร์มีค่าเท่ากับ 2 บิท คือ หนึ่งบิท มาจาก A และอีกหนึ่งบิท มาจาก B ดังรูปที่ 2.17



รูปที่ 2.17 ลักษณะพัลส์ของเอ็นโค้ดเดอร์ทั้งสองเฟส

ถ้าให้พัลส์ในช่วงลอจิกสูงมีค่าเป็น "1" และพัลส์ในช่วงลอจิกต่ำมีค่าเป็น "0" จะสามารถใช้ค่าดังกล่าว มาคำนวณหาทิศทางการหมุนของมอเตอร์ได้โดยใช้วิธีการทางดิจิตอล (Digital) คือ การนำค่าที่อ่านได้มาดำเนินการทางตรรกะ แบบค่าต้องแตกต่างกันจึงจะเป็นจริง (Exclusive OR: XOR) โดยการนำบิททางขวาของค่าเก่ามาดำเนินการทางตรรกะค่าต้องแตกต่างกันจึงจะเป็นจริง กับบิททางซ้ายของค่าใหม่ที่อ่านได้ เช่นตัวอย่างดังต่อไปนี้



รูปที่ 2.18 ความละเอียดของพัลส์

จากรูปที่ 2.18 ตอนแรกมอเตอร์อยู่ที่ตำแหน่ง P1 เพราะฉะนั้นค่าของตัวเลข ฐานสอง (binary) ที่อ่านได้จากเอ็นโค้ดเดอร์พัลส์A และ B จะมีค่าเป็น "1" และ "1" ตามลำดับ ถ้ามอเตอร์หมุนไปที่ตำแหน่ง P3 จะทำให้ค่าของตัวเลขฐานสองที่อ่านได้จากเอ็นโค้ดเดอร์พัลส์ A และ B จะมีค่าเป็น "1" และ "0" ตามลำดับ ดังนั้นถ้านำเอาค่าทางขวา จากค่าที่อ่านได้ในครั้งแรก (11) ซึ่งก็คือ 1 มาโอเปอเรเดอร์ทางตรรกะด้วยค่าต้องแตกต่างกันจึงจะเป็นจริง กับค่าทางซ้ายที่อ่านได้ในครั้งต่อมา (10) ซึ่งก็คือ 1 จะได้ผลลัพธ์เท่ากับ 0 เพราะฉะนั้นค่า 0 จึงเป็นค่าที่บอกว่ามอเตอร์หมุนไปทางขวา ในทางกลับกัน ถ้าในตอนแรกมอเตอร์อยู่ที่ตำแหน่ง P1และต่อมามอเตอร์หมุนไปที่ตำแหน่ง P2 ถ้านำเอาค่าทางขวาจากค่าที่อ่านได้ในครั้งแรก (11) ซึ่งก็คือ 1 มาดำเนินการทางตรรกะด้วยค่าต้องแตกต่างกันจึงจะเป็นจริง กับค่าทางซ้ายที่อ่านได้ในครั้งต่อมา (01) ซึ่งก็คือ 0 จะได้ผลลัพธ์เท่ากับ 1 เพราะฉะนั้นค่า 1 จึงเป็นค่าที่บอกว่ามอเตอร์หมุนไปทางซ้าย และการดำเนินการทางตรรกะด้วยค่าต้องแตกต่างกันจึงจะเป็นจริง แสดงไว้ดังตารางที่ 2.1

ตารางที่ 2.1 ผลทางตรรกะด้วยค่าต้องแตกต่างกันจึงจะเป็นจริง

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

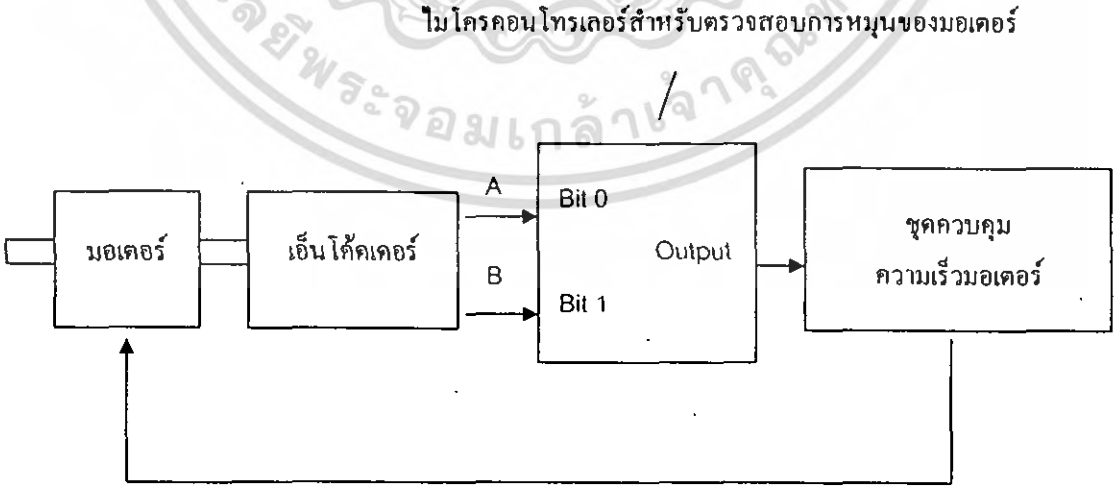
พัลส์ที่ได้จากเอ็นโค้ดเดอร์จะถูกนำไปเชื่อมต่อเข้ากับวงจรนับสัญญาณพัลส์ หรือวงจรคำนวณ เพื่อที่จะตรวจสอบว่ามอเตอร์หมุนไปทิศทางใด และอยู่ที่ตำแหน่งใด ตามวิธีที่ได้กล่าวไว้ข้างต้น โดยส่วนมากแล้ว มอเตอร์ที่ใช้เอ็นโค้ดเดอร์แบบนี้จะมีจำนวนรอบของการหมุน ที่ไม่จำกัด โดยจะหมุนไปกี่รอบก็ได้ขึ้นอยู่กับความสามารถของวงจรนับพัลส์ และวงจรคำนวณ

ดังนั้นจึงกล่าวได้ว่าเอ็นโค้ดเดอร์ ที่ใช้กันอยู่โดยทั่วไปนั้นมีอยู่ 2 แบบใหญ่ๆ คือ แบบมีค่าเอาท์พุทที่ไม่จำกัด (Incremental encoder) และแบบที่มีค่าเอาท์พุทที่จำกัด (Absolute encoder)

เอ็นโค้ดเดอร์แบบมีค่าเอาท์พุทที่ไม่จำกัด มักจะเป็นเอ็นโค้ดเดอร์แบบโรตารี่เอ็นโค้ดเดอร์ (Rotary encoder) ซึ่งขนาดของเอาท์พุทของเอ็นโค้ดเดอร์แบบนี้ ขึ้นอยู่กับการออกแบบวงจรนับและวงจรคำนวณ

เอ็นโค้ดเดอร์แบบที่มีค่าเอาท์พุทที่จำกัดที่พบเห็นส่วนใหญ่ จะเป็นแบบตัวต้านทานปรับค่าได้ แต่ในบางครั้งก็อาจเป็นแบบโรตารี่เอ็นโค้ดเดอร์ก็ได้ แต่จะมีการออกแบบลักษณะของช่องสลิทที่ต่างไปจากโรตารี่เอ็นโค้ดเดอร์ที่ได้ไว้ในตอนต้น โดย อาจจะมีค่าของเอาท์พุทตั้งแต่ 4 บิท ถึง 16 บิท

ในแง่ของการใช้งานแล้ว มอเตอร์ที่มีเอ็นโค้ดเดอร์แบบมีค่าเอาท์พุทที่ไม่จำกัดนั้น จะใช้งานยุ่งยากกว่ามอเตอร์ที่มีเอ็นโค้ดเดอร์แบบที่มีค่าเอาท์พุทที่จำกัด เพราะเอาท์พุทของเอ็นโค้ดเดอร์แบบมีค่าเอาท์พุทที่ไม่จำกัดนั้น ไม่สามารถเชื่อมต่อกับชุดควบคุมมอเตอร์ได้ทันที ต้องต่อผ่านวงจรนับสัญญาณพัลส์และวงจรคำนวณที่ยุ่งยากซับซ้อนก่อน ถึงจะต่อเข้ากับชุดควบคุมมอเตอร์ได้ แต่อย่างไรก็ตาม ในปัจจุบันมีไมโครโปรเซสเซอร์ที่มีอุปกรณ์ช่วยในการนับสัญญาณเอ็นโค้ดเดอร์อยู่ภายในตัว เช่น ไมโครคอนโทรลเลอร์ตระกูล PIC เป็นต้น จึงสามารถนำมาใช้แทนวงจรนับสัญญาณพัลส์และวงจรคำนวณได้จึงทำให้การใช้งานมีความง่ายขึ้น ดังรูปที่ 2.19



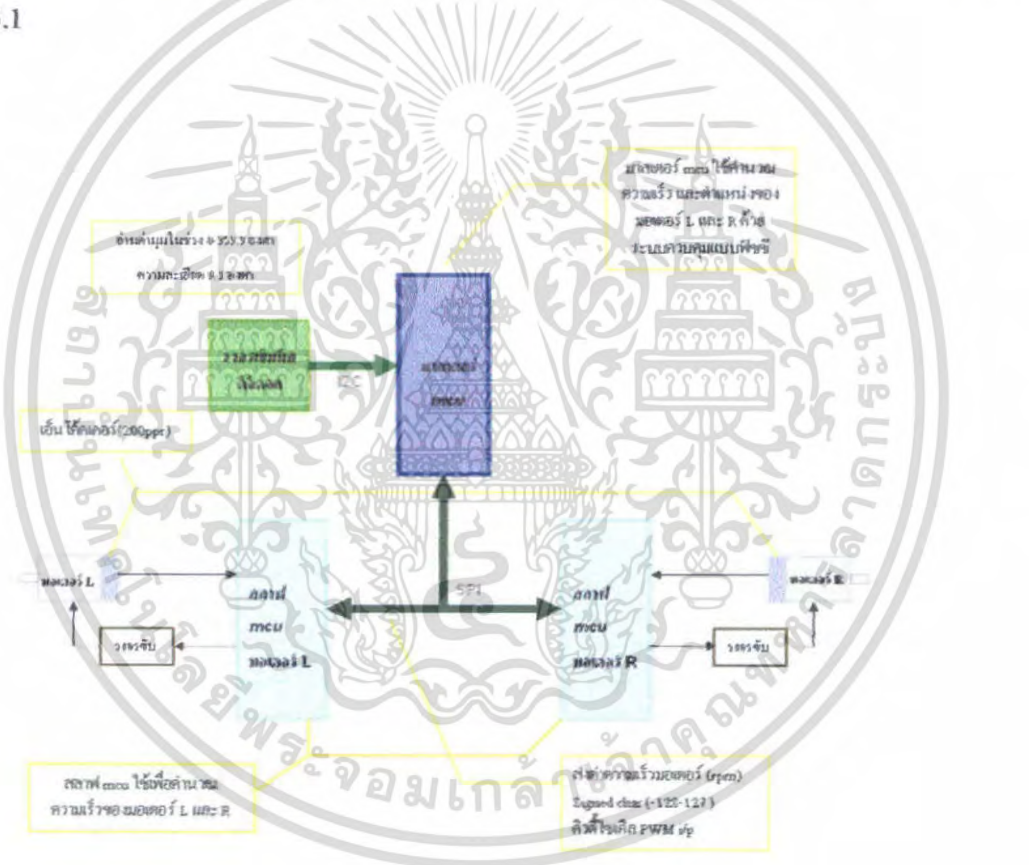
รูปที่ 2.19 บล็อคไดอะแกรมของระบบควบคุมความเร็วมอเตอร์กระแสตรง

บทที่ 3

หลักการออกแบบ

การออกแบบระบบควบคุมอัตโนมัติแบบฟัซซี ในเบื้องต้นนั้นจำเป็นต้องทำความเข้าใจกับภาพรวมของระบบตวบคุมก่อนว่ามีหลักการทำงานอย่างไร แล้วจึงพิจารณาส่วนย่อยต่างๆ อันประกอบไปด้วยส่วนที่เป็นโครงสร้างระบบควบคุม ส่วนที่เป็นวงจรอิเล็กทรอนิกส์ และส่วนของโปรแกรมที่ใช้ในการประมวลผลแบบฟัซซีเพื่อควบคุมระบบ

หลักการทำงานในภาพรวมของระบบควบคุมอัตโนมัติแบบฟัซซี สามารถแสดงดังแผนผังในรูปที่ 3.1



รูปที่ 3.1 แผนผังรวมของระบบระบบควบคุมอัตโนมัติแบบฟัซซี

จากรูปที่ 3.1 เห็นว่าระบบควบคุมอัตโนมัติแบบฟัซซี ประกอบด้วยส่วนหลักๆ ดังนี้คือ

- ไมโครคอนโทรลเลอร์
- วงจรขับมอเตอร์ไฟฟ้ากระแสตรง
- วงจรเข็มทิศดิจิตอล

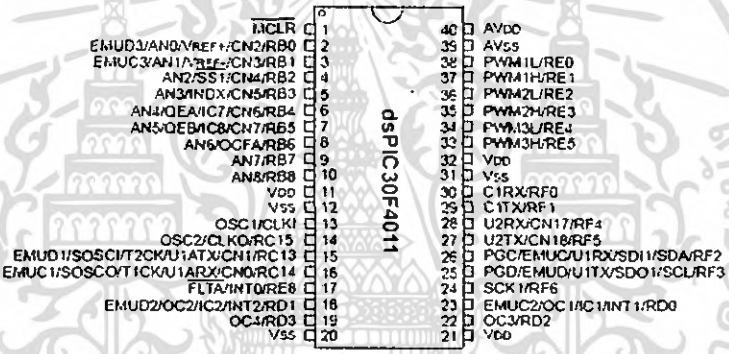
- โครงสร้างทางกายภาพของหุ่นยนต์
- โครงสร้างของระบบควบคุม

3.1 ไมโครคอนโทรลเลอร์

ในโครงงานนี้เลือกใช้ใช้ไมโครคอนโทรลเลอร์ในการควบคุมระบบควบคุมอัตโนมัติแบบฟัซซี 2 ส่วนหลักซึ่งประกอบด้วยส่วนที่เป็นมาสเตอร์ (Master) และ สลาฟ (Slave)

3.1.1 มาสเตอร์ไมโครคอนโทรเลอร์

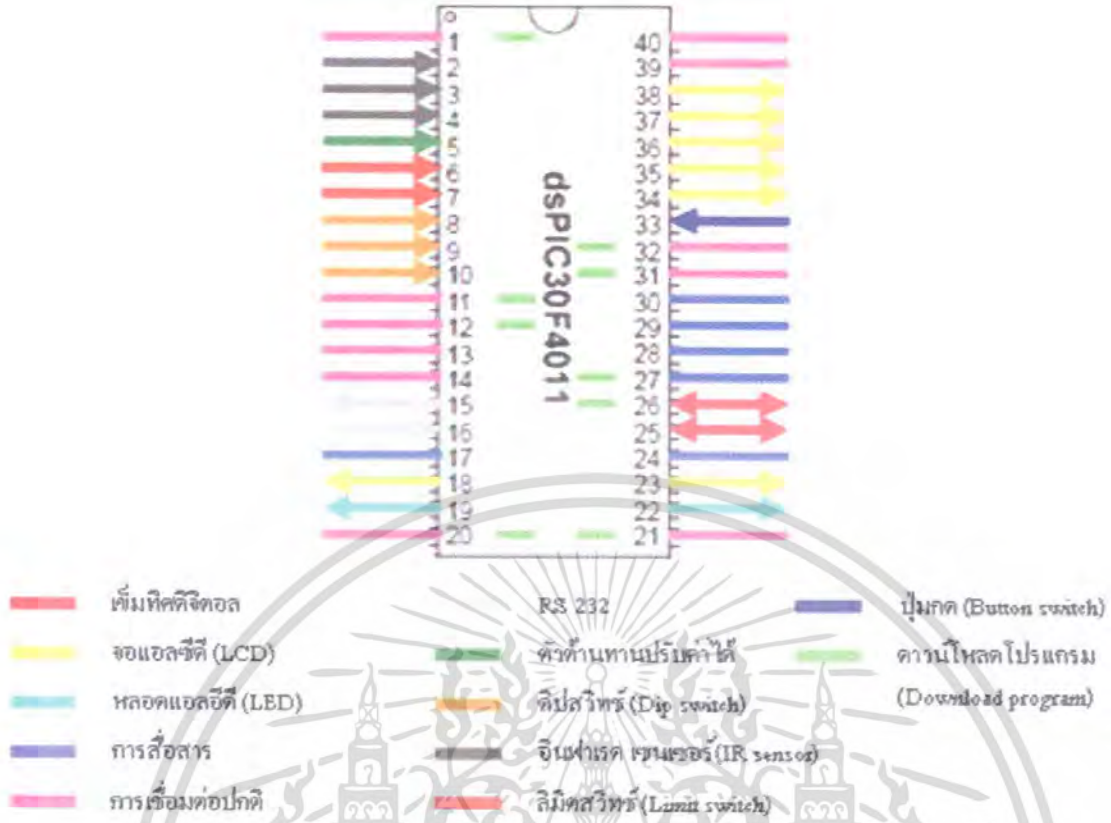
ในโครงงานนี้เลือกใช้ dsPIC30F4011 ทำหน้าที่ในการควบคุมระบบหลักในการควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง โดยจะประมวลผลชุดคำสั่งหลักก่อน จึงส่งคำสั่งไปควบคุมตัวสลาฟ ซึ่งมีคุณสมบัติดังรูปที่ 3.2 และตารางที่ 3.1



รูปที่ 3.2 มาสเตอร์ไมโครคอนโทรลเลอร์เบอร์ 30F4011

ตารางที่ 3.1 คุณสมบัติของมาสเตอร์ไมโครคอนโทรลเลอร์

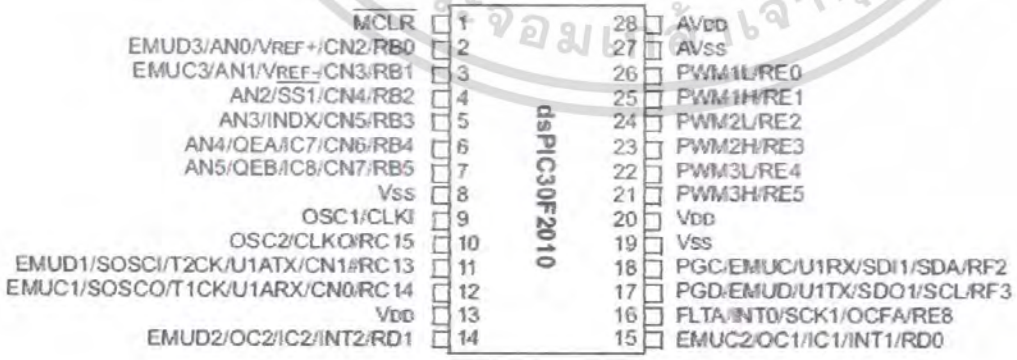
Device	Pins	Program Mem. Bytes/ Instructions	SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Motor Control PWM	10-Bit A/D 1 Msps	Quad Enc	UART	SPI	I²C™	CAN
dsPIC30F2010	28	12K/4K	512	1024	3	4	2	6 ch	6 ch	Yes	1	1	1	--
dsPIC30F3010	28	24K/8K	1024	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	--
dsPIC30F4012	28	48K/16K	2048	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	1
dsPIC30F3011	40/44	24K/8K	1024	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	--
dsPIC30F4011	40/44	48K/16K	2048	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66K/22K	2048	1024	5	4	4	8 ch	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144K/48K	8192	4096	5	8	8	8 ch	16 ch	Yes	2	2	1	2



รูปที่ 3.3 การเชื่อมต่อมาสเตอร์ไมโครคอนโทรลเลอร์

3.1.2 สลาฟไมโครคอนโทรเลอร์

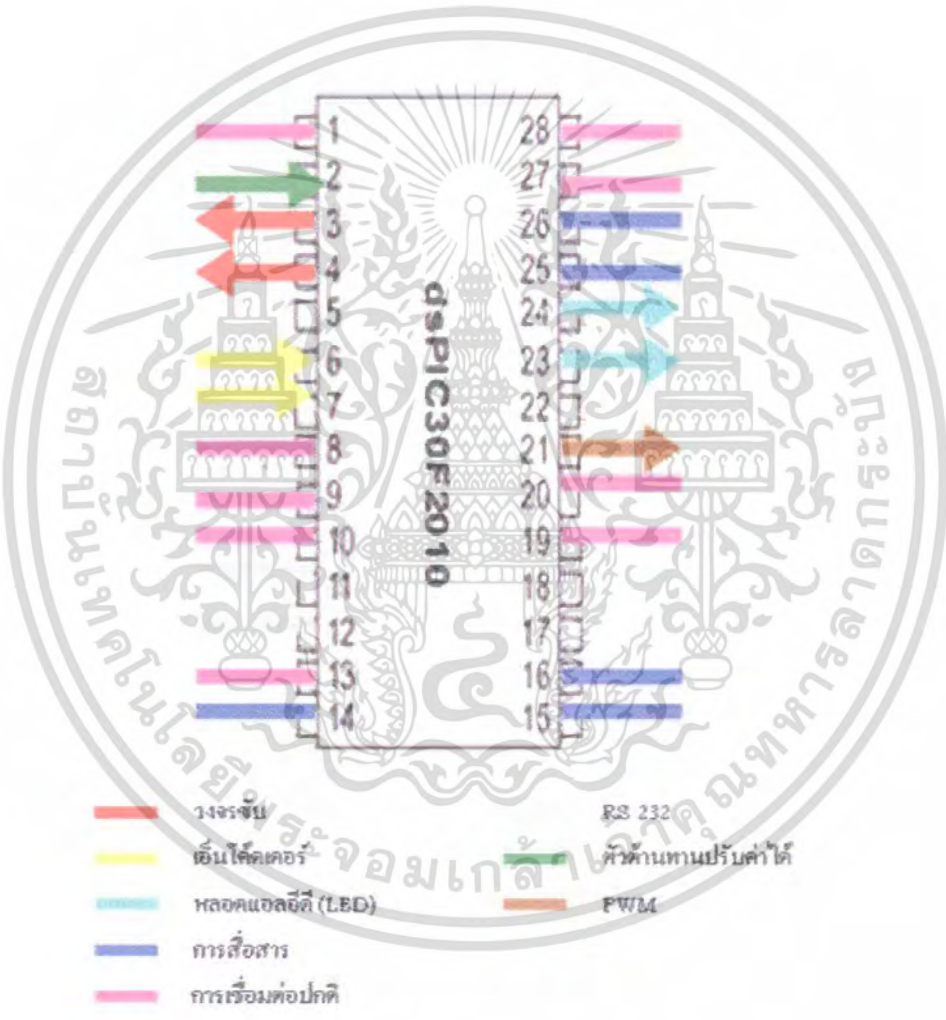
ในโครงงานนี้เลือกใช้ dsPIC30F2010 จำนวน 2 ตัว ด้วยกันเพื่อแยกกันทำหน้าที่ในการควบคุมวงจรขับสำหรับควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง โดยรับคำสั่งมาจากตัวมาสเตอร์ไมโครคอนโทรลเลอร์ ซึ่งจำเป็นมีคุณสมบัติดังรูปที่ 3.4 และ ตารางที่ 3.2



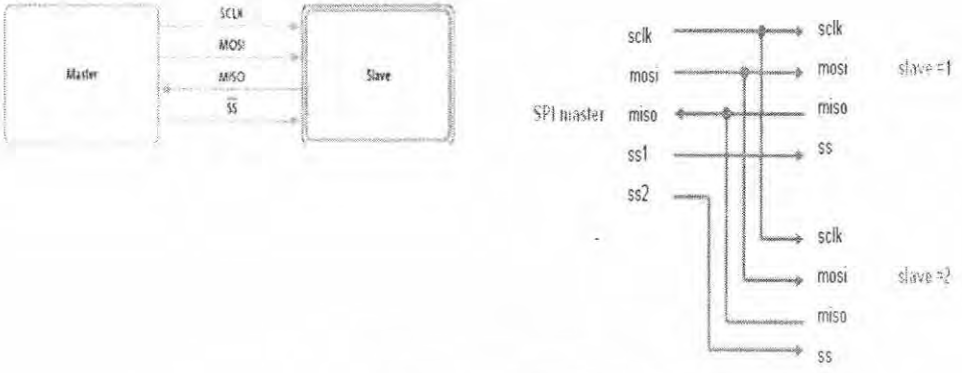
รูปที่ 3.4 สลาฟไมโครคอนโทรลเลอร์เบอร์ 30F2010

ตารางที่ 3.2 คุณสมบัติของสลาฟไมโครคอนโทรลเลอร์

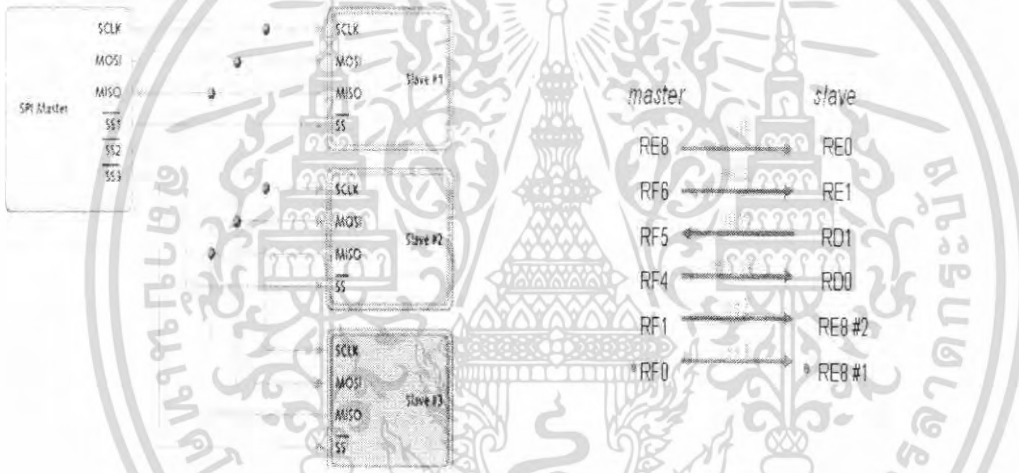
Device	Pins	Program Mem. Bytes/ Instructions	SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Motor Control PWM	10-Bit A/D 1 Msps	Quad Enc	UART	SPI	I²C™	CAN
dsPIC30F2010	28	12K/4K	512	1024	3	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F3010	28	24K/8K	1024	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F4012	28	48K/16K	2048	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	1
dsPIC30F3011	40/44	24K/8K	1024	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	-
dsPIC30F4011	40/44	48K/16K	2048	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66K/22K	2048	1024	5	4	4	8 ch	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144K/48K	8192	4096	5	8	8	8 ch	16 ch	Yes	2	2	1	2



รูปที่ 3.5 การเชื่อมต่อสลาฟไมโครคอนโทรลเลอร์



(ก) การเชื่อต่อมาสเตอร์กับสลาฟเพียงตัวเดียว



(ข) การเชื่อต่อมาสเตอร์กับสลาฟหลายตัวเดียว

รูปที่ 3.6 การเชื่อต่อระหว่างมาสเตอร์ และ สลาฟไมโครคอนโทรลเลอร์

3.2 วงจรขับมอเตอร์ไฟฟ้ากระแสตรง

ใช้คำสั่งจาก MCPWM ของสลาฟไมโครคอนโทรลเลอร์ ในการขับมอเตอร์ไฟฟ้ากระแสตรง เพื่อสั่งการให้มอเตอร์ขับเคลื่อนได้ดังที่ได้อธิบายไว้ในบทที่ 2

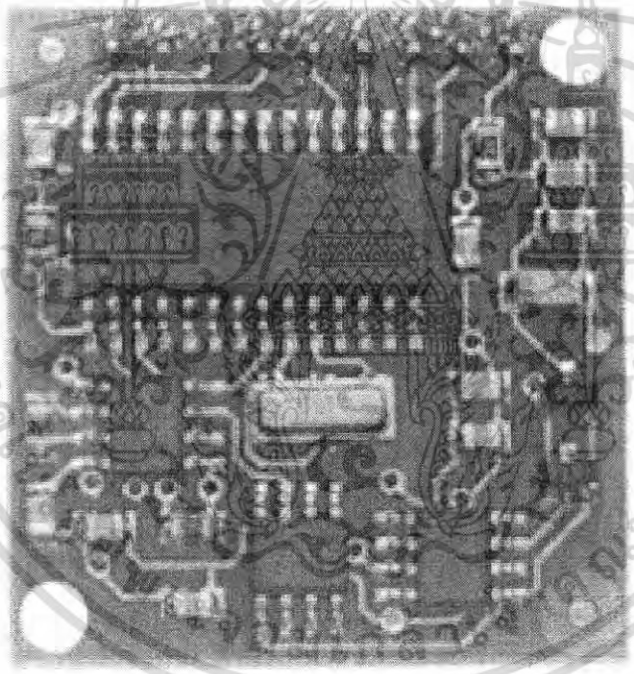
```
//PWM
void mcpwm_init() {
  unsigned int period=512,sptime,config1,config2,config3;

  CloseMCPWM();

  sptime=0x00;              // SEVTCMP = Special Time(Not Used)

  config1=PWM_EN &          // Enable PWM Function
      PWM_IDLE_STOP &       // Disable PWM in IDLE Mode
      PWM_OP_SCALE1 &       // PWM Post Scale = 1
      PWM_IPCLK_SCALE1 &    // PWM Input Clock Prescale = 1
      PWM_MOD_FREE ;        // PWM = Free Running

  config2=PWM_MOD1_IND &    // PWM1 = Free Mode
      PWM_MOD2_IND &        // PWM2 = Free Mode
      PWM_MOD3_IND &        // PWM3 = Free Mode
      PWM_PEN3H &           // H of channel 3 work as PWM
      PWM_PDIS3L &          // L of channel 3 work as IO
      PWM_PDIS2H &          // H of channel 2 work as IO
      PWM_PDIS2L &          // L of channel 2 work as IO
      PWM_PDIS1H &          // H of channel 1 work as IO
      PWM_PDIS1L;           // L of channel 1 work as IO

  config3=PWM_SEVOPS1 &     // Special Even Post Scaler = 1:1
      PWM_OSYNC_PWM &       // Overide Sync. With PWM Clock
      PWM_UEN ;             // Enable PWM Update
  OpenMCPWM(512,sptime,config1,config2,config3);
}

void speed_output(signed char ch,signed int duty) {
  if(duty>0) {
    dir1=backward;  dir2=forward;
    if(duty>1023) {
      duty=1023;
    }
    SetDCMCPWM(ch,duty,0);
  } else if(duty<0) {
    dir1=forward;  dir2=backward;
    if(duty<-1023) {
      duty=-1023;
    }
    duty=abs(duty);
    SetDCMCPWM(ch,duty,0);
  } else {
    dir1=0;  dir2=0;
    SetDCMCPWM(ch,0,0);
  }
```

รูปที่ 3.7 ตัวอย่างคำสั่ง MCPWM ของสลาฟไมโครคอนโทรลเลอร์

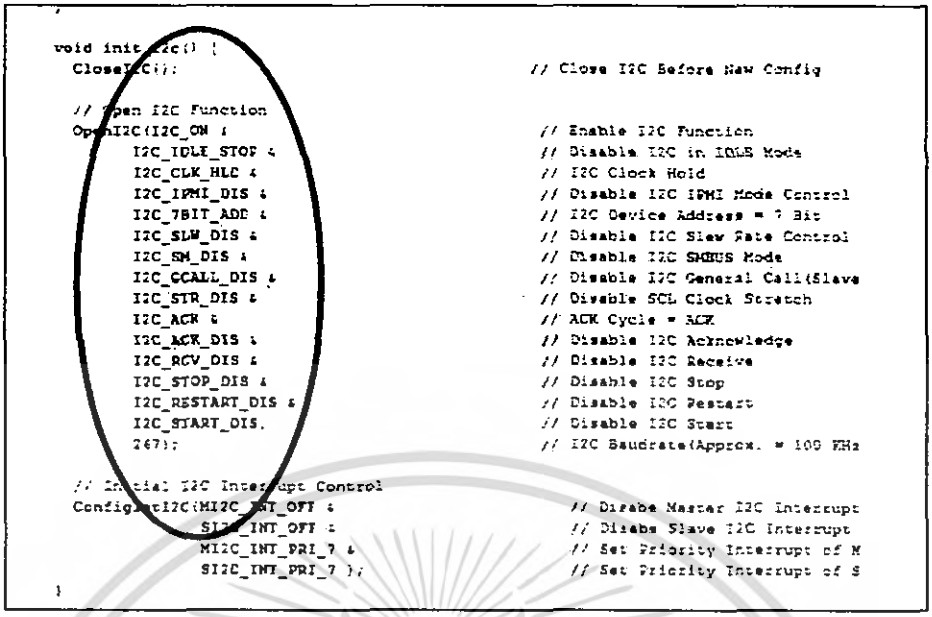
3.3 วงจรเข็มทิศดิจิตอล

ใช้คำสั่งจาก I^2C ของมาสเตอร์ไมโครคอนโทรลเลอร์ในการสื่อสารกับวงจรเข็มทิศดิจิตอล เพื่อใช้ในการตรวจเช็คค่ามุมของตัวหุ่นยนต์ว่า เคลื่อนที่เป็นเส้นตรงหรือไม่ ในโครงงานนี้เลือกใช้วงจรเข็มทิศดิจิตอลรุ่น CMPS03 เป็นผลงานของ Devantoch ซึ่งมีคุณสมบัติดังนี้

- ใช้ไฟเลี้ยง +5 V และ กระแสไฟฟ้า 20 mA
- สามารถตรวจจับสนามแม่เหล็กโลกได้
- ความละเอียดของมุม 0.1 องศา
- ขนาด 32 x 35 มิลลิเมตร
- สามารถสื่อสารกับไมโครคอนโทรลเลอร์ได้



รูปที่ 3.8 วงจรเข็มทิศดิจิตอล



```
void init_i2c() {
  CloseI2C();                                  // Close I2C Before New Config

  // Open I2C Function
  OpenI2C(I2C_ON &                             // Enable I2C Function
          I2C_IDLE_STOP &                      // Disable I2C in IDLE Mode
          I2C_CLK_HLD &                        // I2C Clock Hold
          I2C_IPMI_DIS &                       // Disable I2C IPMI Mode Control
          I2C_7BIT_ADD &                       // I2C Device Address = 7 Bit
          I2C_SLW_DIS &                        // Disable I2C Slew Rate Control
          I2C_SM_DIS &                         // Disable I2C SMBUS Mode
          I2C_GCALL_DIS &                      // Disable I2C General Call(Slave
          I2C_STR_DIS &                        // Disable SCL Clock Stretch
          I2C_ACK &                            // ACK Cycle = ACK
          I2C_ACK_DIS &                        // Disable I2C Acknowledge
          I2C_RCV_DIS &                        // Disable I2C Receive
          I2C_STOP_DIS &                       // Disable I2C Stop
          I2C_RESTART_DIS &                    // Disable I2C Restart
          I2C_START_DIS,                       // Disable I2C Start
          267);                                // I2C Baudrate(Approx. = 100 KHz

  // Initial I2C Interrupt Control
  ConfigIntI2C(MI2C_INT_OFF &                  // Disabe Master I2C Interrupt
               SI2C_INT_OFF &                  // Disabe Slave I2C Interrupt
               MI2C_INT_PRI_7 &                // Set Priority Interrupt of M
               SI2C_INT_PRI_7 );               // Set Priority Interrupt of S
}
```

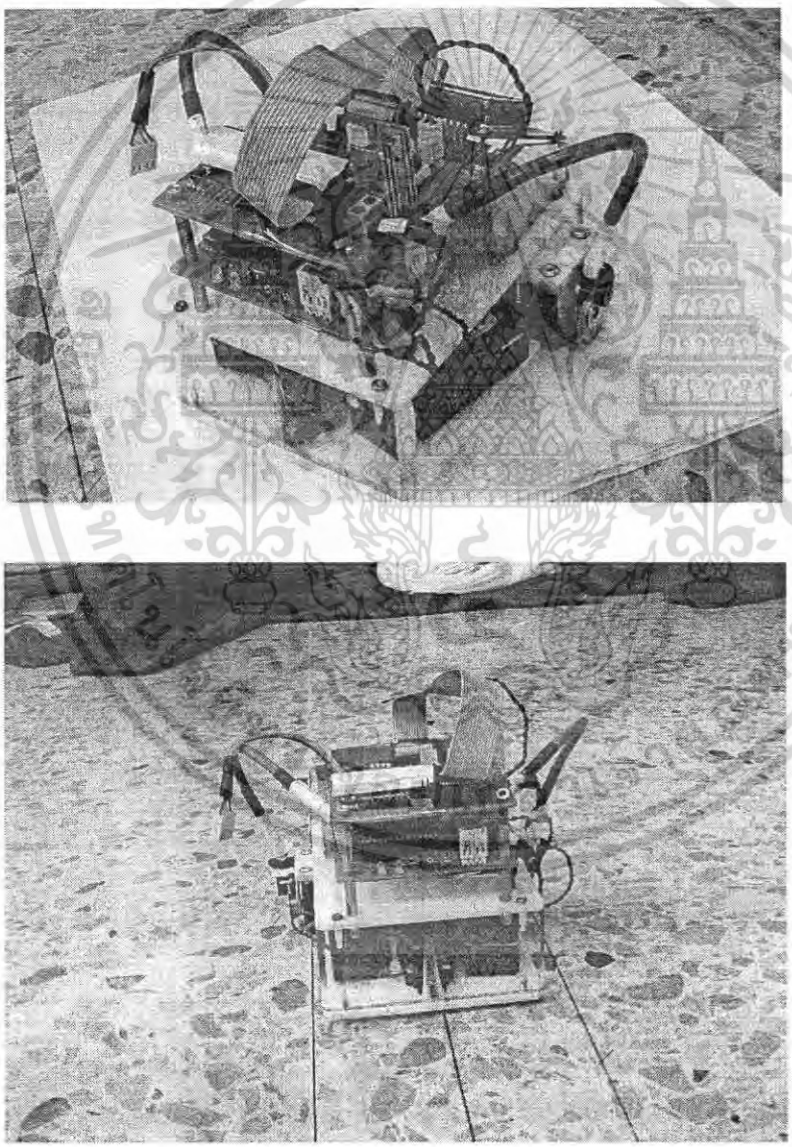
รูปที่ 3.9 ตัวอย่างคำสั่ง I^2C ของมาสเตอร์ไมโครคอนโทรลเลอร์

```
//#include"C:\dsPIC\dspic_o00olib\i2c_cmp.h"
//Pin Hardware
//cmp sensor on portF   RF2 = SDA   RF3 = SCL
#include<i2c.h>                                  // Used I2C Function Library
#define cmp_wd 0xC0
#define cmp_rd 0xC1

void init_i2c();
void rd_angle_cmp0_3599(unsigned int *ptemp);
unsigned char rd_angle_cmp0_255();

unsigned int angle() {
  unsigned char temp[2]={0,0};
  unsigned int ang_now=0;
  rd_angle_cmp0_3599(&temp);
  ang_now=temp[0];
  ang_now<<=8;
  ang_now|=temp[1];
  return(ang_now);
}
```

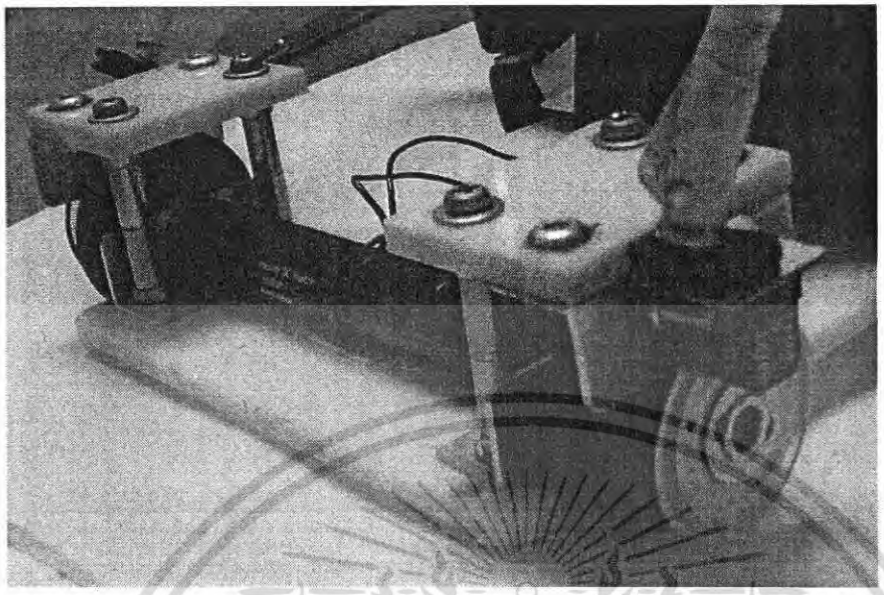
รูปที่ 3.10 ตัวอย่างคำสั่ง อ่านค่าจากโมดูลเข็มทิศสนามแม่เหล็กโลก

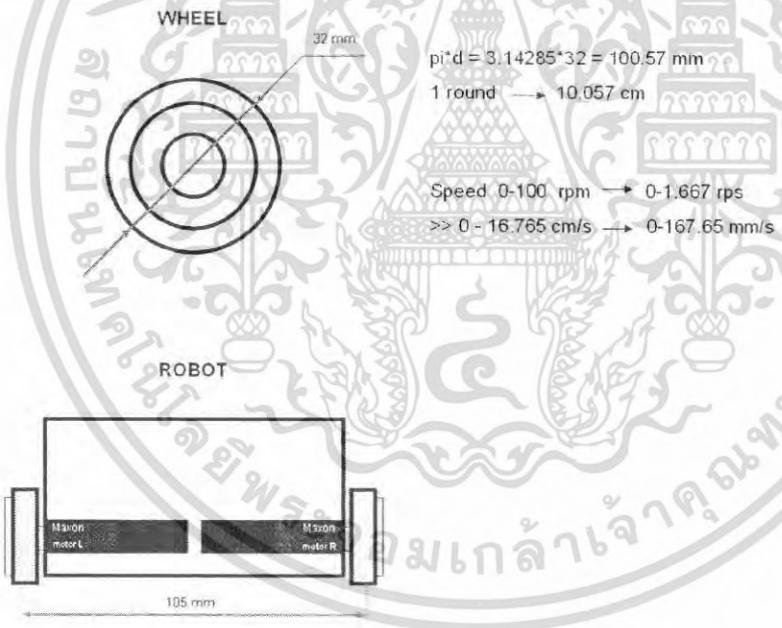
3.4 โครงสร้างทางกายภาพของหุ่นยนต์

หลักจากได้ผังการทำงานโดยรวมของระบบควบคุมแล้ว จึงทำการออกแบบรูปทรงที่เหมาะสมกับวงจรที่เกี่ยวข้องในการขับเคลื่อนหุ่นยนต์ โดยยึดตำแหน่งของมอเตอร์ไฟฟ้ากระแสตรงที่เป็นอิสระต่อกันทั้งสองข้างเป็นหลักในการออกแบบกลไกในการเคลื่อนที่ของหุ่นยนต์ โดยจะยึดมอเตอร์ไว้ในแนวเส้นตรงเดียวกัน มีโรตารี่เอ็นโค้ดเดอร์ความละเอียด 200 พัลส์ เพื่อตรวจสอบความเร็ว ใช้แรงดัน 6 โวลต์ ในการควบคุมหุ่นยนต์ มีโมดูลจอแสดงผลแอลซีดีขนาด 16 ตัวอักษร 4 บรรทัด เป็นจอแสดงผล และมีโมดูลเข็มทิศดิจิตอลช่วยในการควบคุมทิศทางของหุ่นยนต์ ดังแสงในรูปที่ 3.12



รูปที่ 3.11 ภาพรวมทางกายภาพของหุ่นยนต์ในโครงงาน

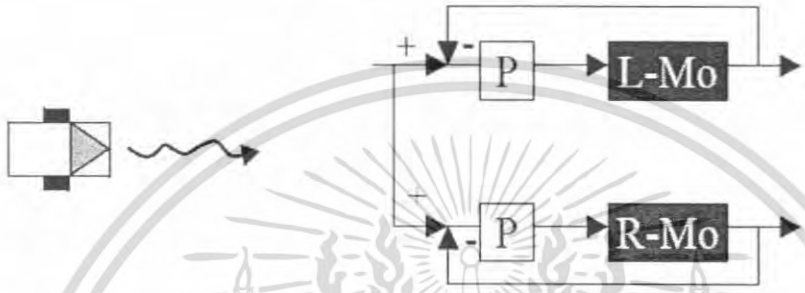




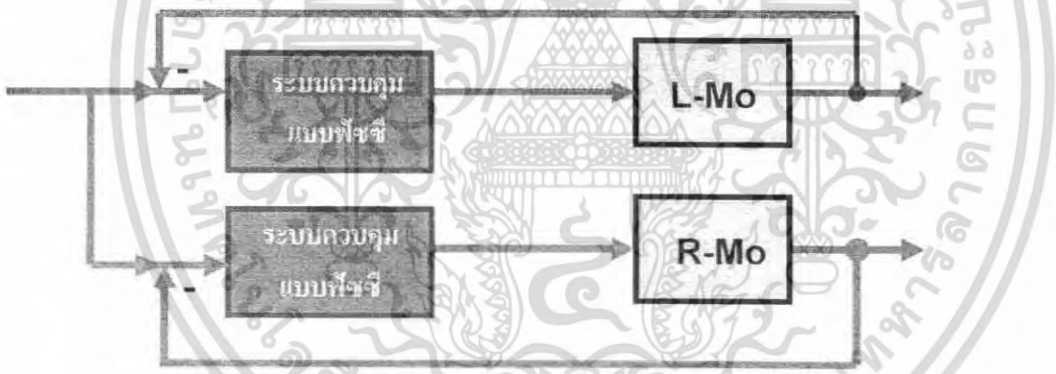
รูปที่ 3.12 การยึดมอเตอร์ในแนวเส้นตรงเดียวกัน

3.5 โครงสร้างของระบบควบคุมหุ่นยนต์

เริ่มพัฒนาจากระบบควบคุมป้อนกลับแยกตัวควบคุมระหว่างมอเตอร์ด้านซ้ายและขวา ดังรูปที่ 3.13 เพื่อศึกษาคุณสมบัติของระบบขั้นต้น ซึ่งมอเตอร์ทั้งสองข้างเป็นอิสระต่อกันไม่สามารถสื่อสารกันได้ โดยใช้สลาฟไมโครคอนโทรลเลอร์เป็นระบบควบคุมแบบฟัซซีแทนในส่วนของตัวควบคุมแบบ P



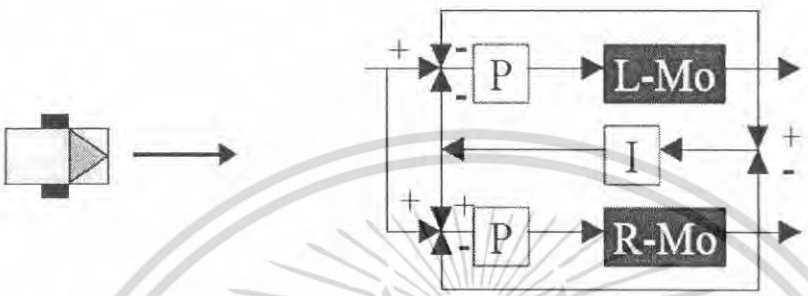
(ก) ระบบควบคุมแบบป้อนกลับ ด้วยระบบควบคุมแบบ P



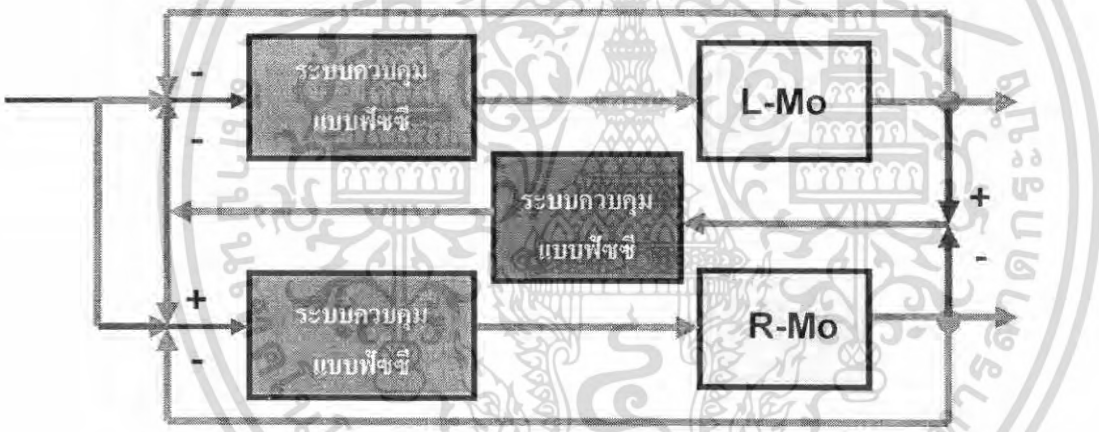
(ข) ระบบควบคุมแบบป้อนกลับ ด้วยระบบควบคุมแบบฟัซซี

รูปที่ 3.13 ระบบควบคุมป้อนกลับแยกตัวควบคุมระหว่างมอเตอร์ด้านซ้ายและขวา

เมื่อทำการควบคุมแบบแยกตัวควบคุมมอเตอร์ด้านซ้ายและด้านขวาซึ่งไม่มีการเปรียบเทียบความเร็วของล้อทั้งซ้ายและขวา เพื่อต้องการควบคุมความเร็วของมอเตอร์ให้ดีขึ้น ต่อมาได้พัฒนาระบบควบคุมเป็น ระบบควบคุมป้อนกลับที่เพิ่มตัวควบคุมเพื่อชดเชยค่าที่ได้จากตัวควบคุมระหว่างมอเตอร์ด้านซ้ายและขวา ดังรูปที่ 3.14



(ก) ระบบควบคุมแบบป้อนกลับ ด้วยระบบควบคุม P และ I

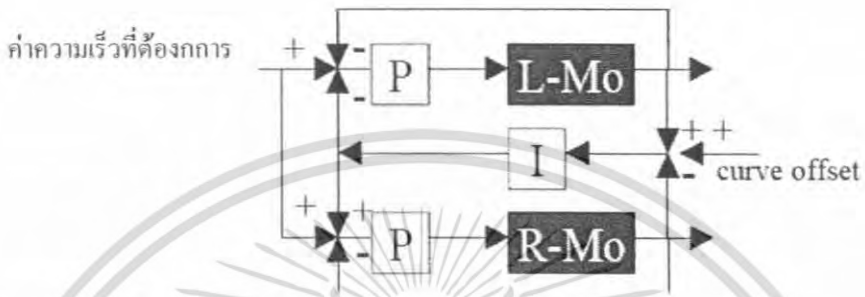


(ข) ระบบควบคุมแบบป้อนกลับ ด้วยระบบควบคุมแบบฟัซซี

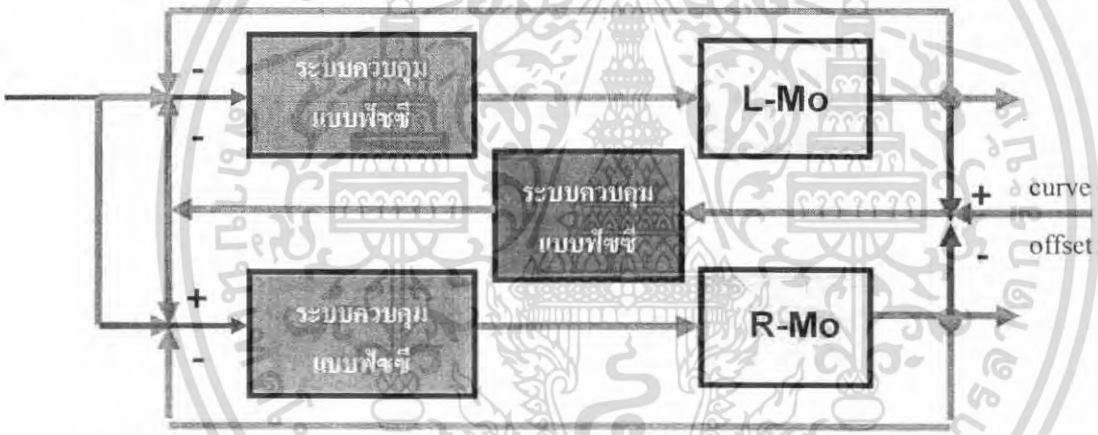
รูปที่ 3.14 ระบบควบคุมป้อนกลับซึ่งเพิ่มตัวควบคุมเพื่อชดเชย
ค่าที่ได้จากตัวควบคุมระหว่างมอเตอร์ด้านซ้ายและขวา

โดยในการพัฒนาในโครงงาน จะใช้มาสเตอร์คอนโทรลเลอร์ทำหน้าที่เป็นตัวควบคุมในส่วนการชดเชยความเร็วระหว่างมอเตอร์ซ้ายและขวา ซึ่งถูกวัดและส่งมาจากสลาฟไมโครคอนโทรลเลอร์

ส่วนขั้นตอนสุดท้ายของการพัฒนาระบบควบคุม เมื่อได้เพิ่มตัวควบคุมเพื่อชดเชยค่าที่ได้จากตัวควบคุมระหว่างมอเตอร์ด้านซ้ายและขวา ซึ่งไม่มีการควบคุมตำแหน่งการเคลื่อนที่ของหุ่นยนต์ ต่อมาได้เพิ่มการสื่อสารจากวงจรเข็มทิศดิจิตอลเข้ามาในระบบ เพื่อช่วยในการชดเชยค่ามุมที่เปลี่ยนไปจากการเคลื่อนที่ของหุ่นยนต์ ดังรูปที่ 3.15



(ก) ระบบควบคุมแบบป้อนกลับ ด้วยระบบควบคุม P และ I และ curve offset



(ข) ระบบควบคุมแบบป้อนกลับ ด้วยระบบควบคุมแบบฟัซซี และ curve offset

รูปที่ 3.15 ระบบควบคุมป้อนกลับ โดยเพิ่มตัวควบคุมเพื่อชดเชย ค่าที่ได้จากตัวควบคุมระหว่างระหว่างมอเตอร์ซ้าย และ ขวาและ ค่ามุมที่เปลี่ยนจากมุมเริ่มต้นการวิ่ง

บทที่ 4

การทดลอง

ในบทนี้ จะกล่าวถึงการทดลองและผลการทดลอง การปรับแต่งกฎของฟัซซีด้วย MATLAB การเขียนโปรแกรมควบคุมด้วยภาษา C และการทำงานของระบบควบคุมอัตโนมัติแบบฟัซซีเพื่อควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง โดยมีรายละเอียดของการทดลองดังนี้

4.1 การทดลองการปรับแต่งกฎของฟัซซีด้วย MATLAB

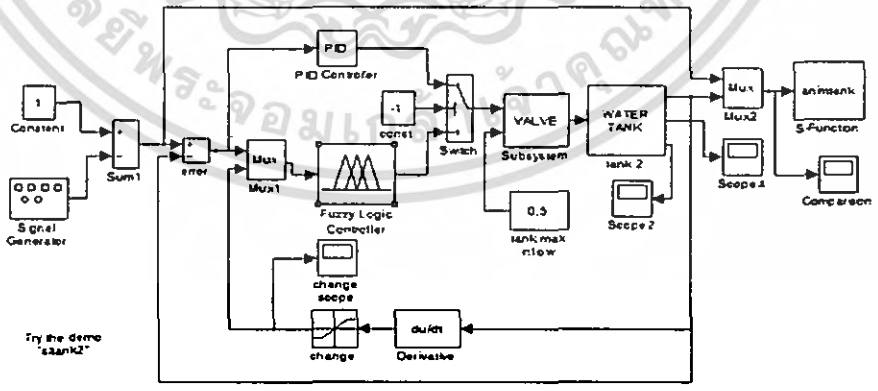
ในส่วนนี้ เป็นการศึกษาเฉพาะส่วนการปรับแต่งกฎของฟัซซีด้วย MATLAB ซึ่งใช้ฟังก์ชัน Simulink ใน โปรแกรม MATLAB ช่วยในการจำลองตัวควบคุมแบบฟัซซี เพื่อศึกษาแนวโน้มในการปรับแต่งในรูปแบบต่างๆ ตามทฤษฎีมอเตอร์ เปรียบเทียบกับตัวควบคุมแบบ PID ซึ่งเป็นระบบควบคุมที่นิยมใช้กันมากในยุคปัจจุบัน ได้ผลการทดลองดังนี้

4.1.1 ผลกระทบจากการเลือกฟังก์ชันการเป็นสมาชิกของฟัซซีเซต

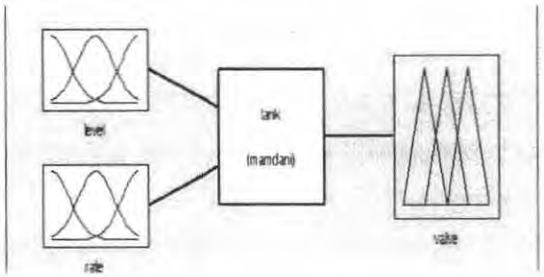
ที่มาของการทดลอง เนื่องจากระบบควบคุมแบบฟัซซีมีรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตถึง 5 แบบแต่ มีเพียง 3 รูปแบบที่ง่ายต่อการเขียนโปรแกรมจึงทำการจำลองเพื่อเปรียบเทียบ

ขั้นตอนการทดลอง

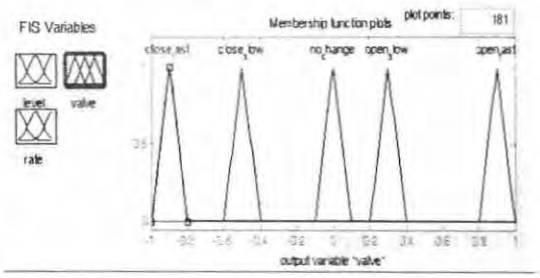
เลือกแบบจำลองระบบควบคุมฟัซซีจาก แบบจำลองระบบควบคุมระดับน้ำ ของโปรแกรม Simulink ในโปรแกรม MATLAB



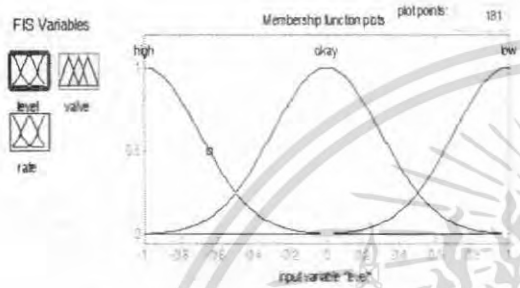
รูปที่ 4.1 แบบจำลองระบบควบคุมระดับน้ำจาก แบบจำลอง ของ Simulink ใน MATLAB



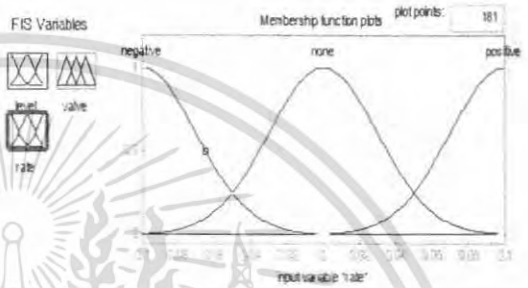
รูปที่ 4.2 แบบจำลองของตัวควบคุมฟัซซีที่ใช้ในการควบคุม



รูปที่ 4.3 รูปแบบฟังก์ชันการเป็นสมาชิกเอาท์พุท valve



รูปที่ 4.4 รูปแบบฟังก์ชันการเป็นสมาชิกอินพุท level



รูปที่ 4.5 รูปแบบฟังก์ชันการเป็นสมาชิกอินพุท rate

If (level is okey) then (valve is no_change)

If (level is low) then (valve is open_fast)

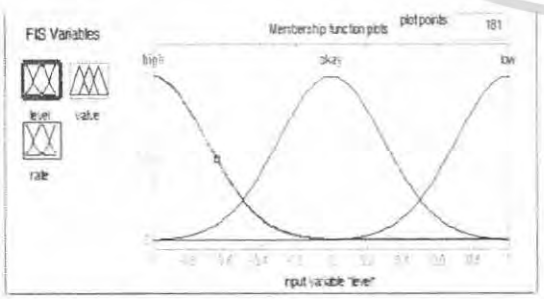
If (level is high) then (valve is close_fast)

If (level is okey) and (rate is positive) then (valve is close_slow)

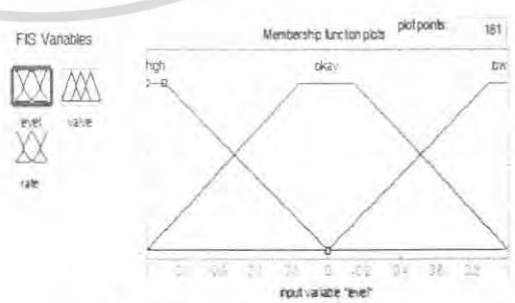
If (level is okey) and (rate is negative) then (valve is open_slow)

รูปที่ 4.6 กฎที่ใช้ในการออกแบบตัวควบคุม

ทดลองปรับรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตรูปแบบต่างๆและบันทึกผล



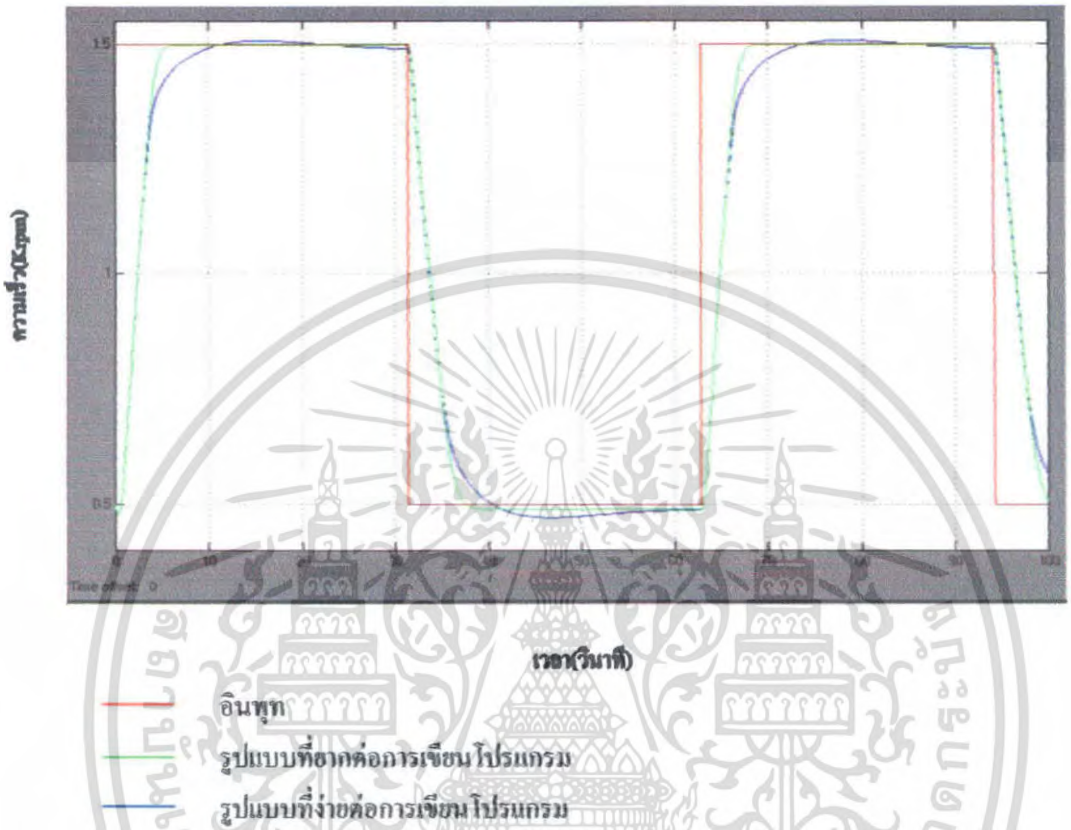
รูปที่ 4.7 รูปแบบฟังก์ชันการเป็นสมาชิกที่ยากต่อการเขียนโปรแกรม



รูปที่ 4.8 รูปแบบฟังก์ชันการเป็นสมาชิกที่ง่ายต่อการเขียนโปรแกรม

เปรียบเทียบผลที่ได้จากการเปลี่ยนแปลงฟังก์ชันการเป็นสมาชิกของฟัซซีเซต

สรุป และ วิจารณ์ ผลการทดลอง



รูปที่ 4.9 ผลการทดลองรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซต ที่สะดวกต่อการเขียนโปรแกรม

จากผลที่ได้รูปที่ 4.9 ด้านซ้าย เป็นผลจากรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตที่ง่ายต่อการเขียนโปรแกรม เมื่อเทียบกับกราฟสีเขียวที่เป็นผลจากรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตที่ยากต่อการเขียนโปรแกรม จะเห็นได้ว่าผลที่ได้ กราฟสีเขียว เมื่อเทียบกับ ค่าอินพุท กราฟสีแดง จากรูปกราฟสีเขียวจะมี ค่าพุ่งเกิน มากกว่า กราฟสีน้ำเงิน ซึ่งไม่มีค่า ออฟเซ็ต และมีค่าเวลาเข้า t_s (Settling time) เร็วกว่า จึงสรุปว่าควรใช้รูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตที่ง่ายต่อการเขียนโปรแกรม ในการตั้งรูปแบบฟังก์ชันการเป็นสมาชิกของฟัซซีเซตที่จะนำไปใช้ต่อไปในการเขียนโปรแกรม

4.1.2 เปรียบเทียบการทำงานระหว่างระบบควบคุมฟัซซีแบบ I กับ ระบบควบคุมแบบ I ของระบบสองมวล

ที่มาของการทดลอง เนื่องจากระบบควบคุมแบบ I จะใช้ในการแก้ปัญหา ออฟเซ็ต ของระบบ โดยจะเข้าไปกำจัดค่า ออฟเซ็ต ที่ยังคงมีอยู่ให้ระบบเข้าสู่ ค่าอ้างอิง ตามค่าเวลาสะสม (Integral Time) ที่กำหนด ซึ่งค่าที่กำหนดให้เวลาสะสมนั้นหากมีค่าน้อยเกินไปจะเข้าสู่ ค่าอ้างอิง อย่างรวดเร็ว แต่จะเกิดการกระเพื่อม (Hinting) ของกระบวนการ (Process) มากขึ้นตามมาด้วย และหากกำหนดค่าเวลาสะสมมากเกินไปจะเกิดการกระเพื่อมของกระบวนการน้อย แต่จะใช้เวลานานกว่าระบบจะเข้าสู่ค่าอ้างอิง ด้วยเหตุนี้จึงสมควรใช้ระบบควบคุมฟัซซีแบบ I แทนเพื่อสะดวกแก่การเปลี่ยนแปลงค่า I ให้เหมาะสมกับระบบตามสถานการณ์ และให้ผลลัพธ์ที่ดีกว่าเดิม ซึ่งในการทดลองนี้ได้ใช้สำหรับควบคุมระบบสองมวล ให้มีเสถียรภาพ และเปรียบเทียบการทำงานของระบบควบคุมแบบฟัซซีแบบ I กับระบบควบคมแบบ I

$$G_p(s) = \frac{3648s^2 + 3.038s + 5941000}{s^4 + 250s^3 + 17150s^2 + 814300s + 22630000} \qquad G_c(s) = \frac{1}{T_i s}$$

รูปที่ 4.10 แบบจำลองของระบบสองมวล

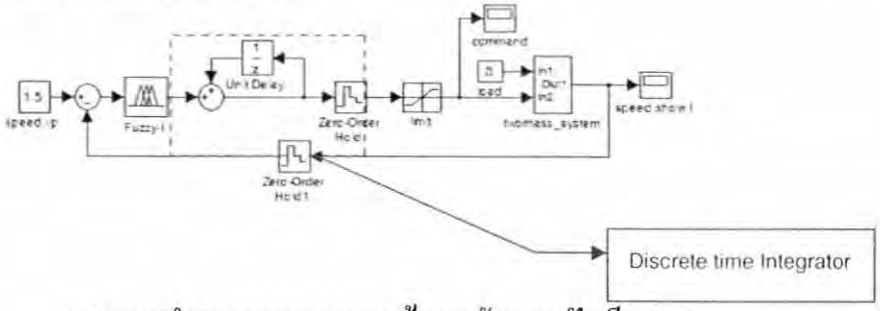
โดยที่ G_p คือแบบจำลองทางคณิตศาสตร์ของระบบสองมวล

$G_c(s)$ คือแบบจำลองทางคณิตศาสตร์ของระบบควบคุมแบบ I

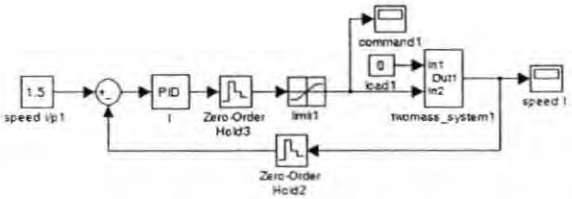
ค่าอ้างอิงคือ 1,500 รอบต่อนาที

ขั้นตอนการทดลอง

สร้างระบบควบคุมแบบป้อนกลับด้วยระบบควบคุมแบบ ฟัซซีแบบ I และระบบควบคุมแบบ I จาก Simulink ในโปรแกรม MATLAB



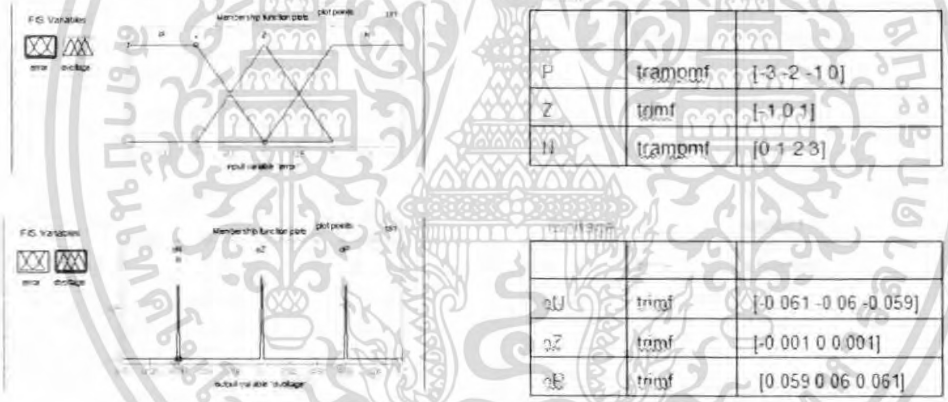
(ก) แบบจำลองระบบควบคุมป้อนกลับแบบฟัซซีแบบ I



(ข) แบบจำลองการควบคุมแบบ I

รูปที่ 4.11 แบบจำลองระบบควบคุมแบบป้อนกลับด้วย ฟัซซีแบบ I และระบบควบคุมแบบ I ของระบบสองมวล

ปรับแต่งค่าเริ่มต้นของระบบควบคุมแบบ ฟัซซีแบบ I และระบบควบคุมแบบ I เพื่อให้ได้ผลดีที่สุดของระบบควบคุมทั้งสองแบบ ในที่นี้ระบบควบคุมแบบ I ใช้ค่า $K_i = \frac{1}{T} = 5.5$ และในส่วนของระบบควบคุมแบบ ฟัซซีแบบ I ปรับค่าฟังก์ชันการเป็นสมาชิก และกฎการควบคุม ดังรูปที่ 4.12



(ก) รูปแสดงการปรับแต่งฟังก์ชันการเป็นสมาชิกฟัซซี I

If (error is Z) then (dvoltage is oZ) (1)

If (error is P) then (dvoltage is oN) (1)

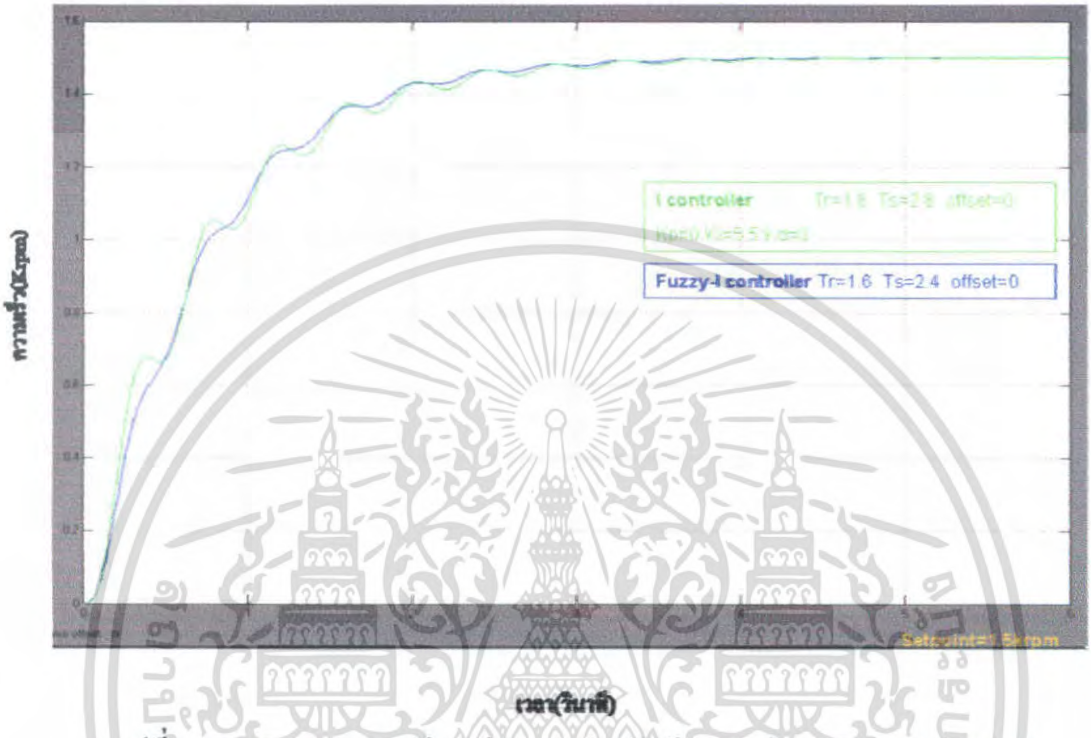
If (error is N) then (dvoltage is oP) (1)

(ข) กฎการควบคุมฟัซซี I

รูปที่ 4.12 ค่าการปรับแต่งฟังก์ชันการเป็นสมาชิก และกฎการควบคุมของฟัซซีแบบ I ของระบบสองมวล

บันทึกค่าเอาท์พุท ของระบบทั้งสอง เพื่อนำมาเปรียบเทียบกัน

สรุป และ วิจารณ์ ผลการทดลอง



รูปที่ 4.13 ผลการทดลองเปรียบเทียบระหว่าง ฟัซซีแบบ I กับ ระบบควบคุมแบบ I ของระบบสองมวล

จากรูปที่ 4.13 แสดงระบบควบคุมแบบ ฟัซซีแบบ I ให้เอาท์พุทใกล้เคียงกับระบบควบคุมแบบ I โดยมีการกระเพื่อมน้อยกว่า สนับสนุนจุดประสงค์ของการทดลองตามที่ได้กล่าวไว้ข้างต้น

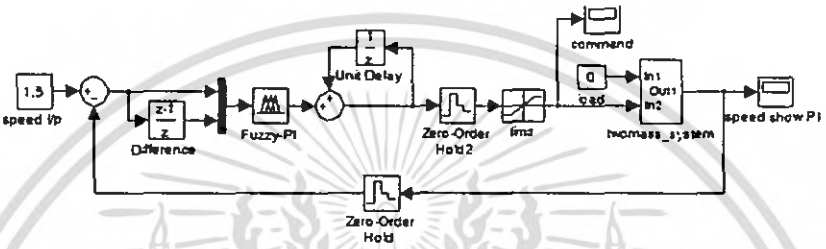
4.1.3 เปรียบเทียบการทำงานระหว่างระบบควบคุมฟัซซีแบบ PI กับระบบควบคุมแบบ PI ของระบบสองมวล

ที่มาของการทดลอง คล้ายกับการทดลองที่ 4.1.2 แต่เพิ่มพจน์ P เข้าไปซึ่งสัญญาณควบคุมจะมีขนานเป็นสัดส่วนกับขนาดของค่า $e(t)$ จึงสามาถช่วยเร่งให้ระบบเข้าสู่ ค่าอ้างอิง ได้รวดเร็วขึ้น แต่สัญญาณวัดของตัวแปรแทรกซ้อน จะมีค่าเท่ากับค่าเป้าหมายที่สภาวะการทำงาน และสภาพแวดล้อมค่าใดค่าหนึ่งเท่านั้น ถ้าสภาวะการทำงาน และสภาพแวดล้อมเปลี่ยนแปลงไปจากค่าเดิม จะเกิดสัญญาณรบกวน (Disturbance) ขึ้น แต่เมื่อทำงานร่วมกับพจน์ I ซึ่งมีการดำเนินการอินทิกรัล (Integral Action) เพื่อทำหน้าที่กำจัด ออฟเซ็ต โดยความไวในการกำจัด ออฟเซ็ต จะขึ้นกับค่า T_i แต่หากเพิ่มอินทิกรัลแอคชัน มากจะมีคาพาซิตี้แลค (Capacity Lag) เข้าไปในระบบ

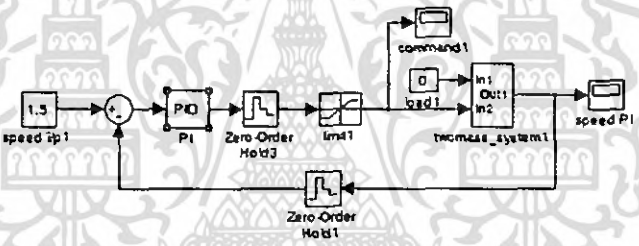
ควบคุม ทำให้ตัวแปรกระบวนการ เปลี่ยนแปลงเข้าสู่ ค่าอ้างอิง ได้ช้าลงจึงใช้ระบบควบคุมแบบฟัซซี PI ทดแทนในส่วนการปรับค่า K_p และ T_i ให้เหมาะสมกับระบบตามสถานการณ์ และให้ผลลัพธ์ที่ดีกว่าเดิม โดยมีแบบจำลองทางคณิตศาสตร์ของระบบควบคุมแบบ PI คือ $G_c(s) = K_p\left(1+\frac{1}{T_i s}\right)$ และมีค่าอ้างอิง คือ 1,500 รอบต่อนาที

ขั้นตอนการทดลอง

สร้างระบบควบคุมแบบป้อนกลับด้วยระบบควบคุมแบบฟัซซีแบบ PI และระบบควบคุมแบบ PI จาก Simulink ในโปรแกรม MATLAB



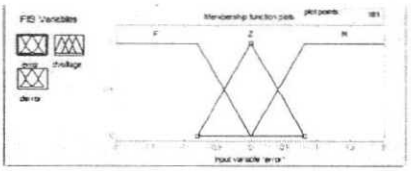
(ก) แบบจำลองระบบควบคุมป้อนกลับแบบฟัซซีแบบ PI



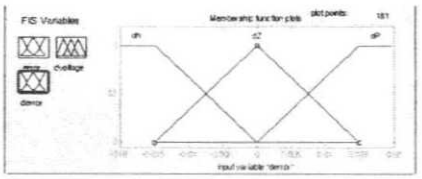
(ข) แบบจำลองระบบควบคุมแบบ PI

รูปที่ 4.14 แบบจำลองระบบควบคุมแบบป้อนกลับด้วยฟัซซีแบบ PI และระบบควบคุมแบบ PI ของระบบสองมวล

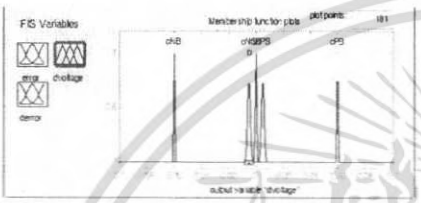
ในส่วนของระบบควบคุมแบบ ฟัซซีแบบ PI จะมี อินพุท 2 ตัว เป็น error และ Difference error ซึ่งเมื่อเจอกับ อินทิกรัลแอคชันจะถูกแปลงให้เสมือนเป็น K_p และ K_i ในระบบควบคุมแบบ PI จากนั้นปรับแต่งค่าเริ่มต้นของระบบควบคุมแบบ ฟัซซีแบบ PIและระบบควบคุมแบบ PI เพื่อให้ได้ผลดีที่สุดของระบบควบคุมทั้งสองแบบ ในที่นี้ระบบควบคุมแบบ PI ใช้ค่า $K_p = 0.1$ และ $K_i = \frac{K_p}{T_i} = 5.5$ สำหรับในส่วนของระบบควบคุมแบบฟัซซีแบบ PI ปรับค่า ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม ดังรูปที่ 4.15



P	trampmf	[-3 -2 -0.8 0]
Z	trimf	[-0.8 0 0.8]
N	trampmf	[0 0.8 2 3]



dN	trampmf	[-0.8 -0.4 -0.015 0]
dZ	trimf	[-0.015 0 0.015]
dP	trampmf	[0 0.015 0.4 0.8]



oNB	trimf	[-0.061 -0.06 -0.059]
oNS	trimf	[-0.007 -0.005 -0.003]
oZ	trimf	[-0.0001 0 0.0001]
oPS	trimf	[0.003 0.005 0.007]
oPB	trimf	[0.059 0.06 0.061]

(ก) การปรับแต่งฟังก์ชันการเป็นสมาชิกฟัซซี PI

If (error is Z) then (dvoltage is oZ) (1)

If (error is P) then (dvoltage is oNB) (1)

If (error is N) then (dvoltage is oPB) (1)

If (error is P) and (derror is dZ) then (dvoltage is oNS) (1)

If (error is P) and (derror is dN) then (dvoltage is oNS) (1)

If (error is N) and (derror is dZ) then (dvoltage is oPS) (1)

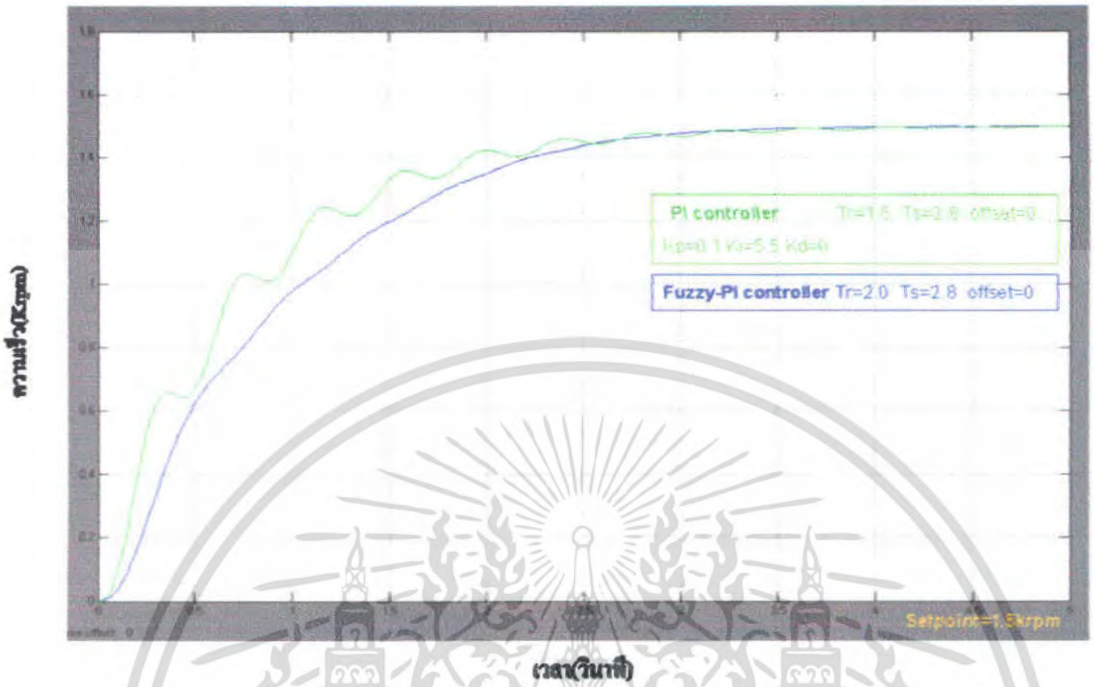
If (error is N) and (derror is dP) then (dvoltage is oPS) (1)

(ข) กฎการควบคุมฟัซซี PI

รูปที่ 4.15 ค่าการปรับแต่ง ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม ของระบบควบคุมแบบ ฟัซซีแบบ PI ของระบบสองมวล

บันทึกค่าเอาท์พุท ของระบบทั้งสอง เพื่อนำมาเปรียบเทียบกัน

สรุป และ วิจารณ์ ผลการทดลอง



รูปที่ 4.16 ผลการทดลองเปรียบเทียบระหว่าง ฟัซซีแบบ PI กับระบบควบคุมแบบ PI ของระบบสองมวล

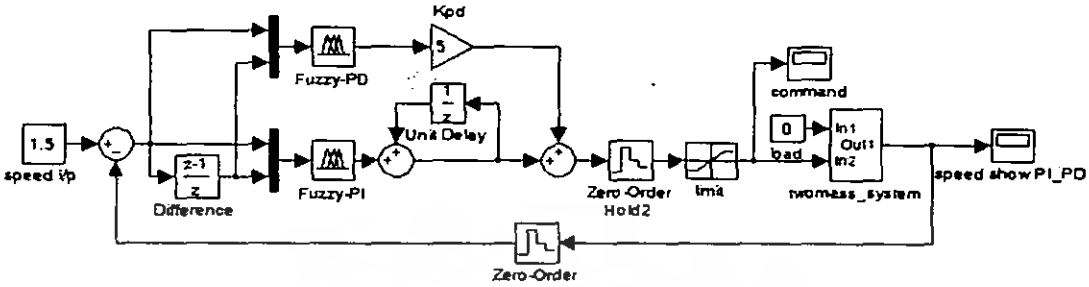
จากรูปที่ 4.16 แสดงว่าระบบควบคุมแบบ ฟัซซีแบบ PI ให้เอาท์พุทดีกว่าระบบควบคุมแบบ PI โดยพบว่าไม่มีการแกว่งของกราฟ ดังนั้นจึงยังคงสนับสนุนจุดประสงค์ของการทดลองตามที่ได้กล่าวไว้ข้างต้น

4.1.4 เปรียบเทียบการทำงานระหว่างระบบควบคุมฟัซซีแบบ PI+PD กับระบบควบคุมแบบ PID ของระบบสองมวล

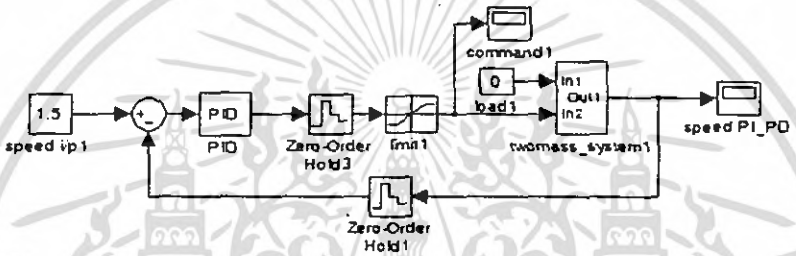
ที่มาของการทดลอง คล้ายกับการทดลองที่ 2 และ 3 แต่เพิ่มพจน์ D เข้าไปซึ่งมีเดริเวทีฟแอคชัน (Derivative Action) เพื่อทำให้ผลตอบสนองของระบบต่อสิ่งรบกวนไวขึ้น โดยขึ้นอยู่กับค่า T_d แต่ระบบจะมีความไวต่อสัญญาณรบกวนเป็นอย่างมากหากมีการรบกวนจากตัวรบกวน อาจจะทำให้เกิดความผิดพลาดของผลตอบสนองอย่างมาก ด้วยเหตุนี้จึงสมควรใช้ระบบควบคุมฟัซซีแบบ PI+PD ทดแทนในส่วนการปรับค่า K_p T_i และ T_d ให้เหมาะสมกับระบบตามสถานการณ์ และให้ผลลัพธ์ที่ดีกว่าเดิม ซึ่งประกอบด้วยแบบจำลองทางคณิตศาสตร์ของระบบควบคุมแบบ PID คือ $G_c = K_p\left(1+\frac{1}{T_i s}+T_d s\right)$ และค่าอ้างอิง คือ 1,500 รอบต่อนาที

ขั้นตอนการทดลอง

สร้างระบบควบคุมแบบป้อนกลับด้วยระบบควบคุมแบบฟัซซีแบบ PI+PD และระบบควบคุมแบบ PID จาก Simulink ในโปรแกรม MATLAB



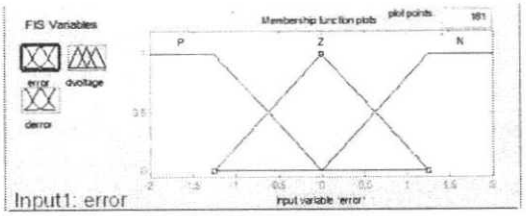
(ก) แบบจำลองระบบควบคุมป้อนกลับแบบฟัซซีแบบ PI +PD



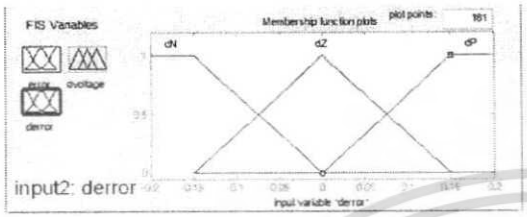
(ข) แบบจำลองระบบควบคุมป้อนกลับแบบฟัซซีแบบ PI + PD

รูปที่ 4.17 แบบจำลองระบบควบคุมแบบป้อนกลับด้วย ฟัซซีแบบ PI+PD และระบบควบคุมแบบ PID ของระบบสองมวล

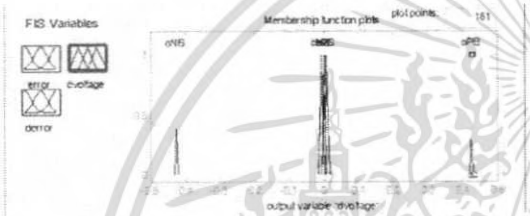
ในส่วนของระบบควบคุมแบบฟัซซีแบบ PI+PD จะอาศัยอินพุทแบบค่าผิดพลาด (error) และผลต่างของค่าผิดพลาด (Difference error) จากระบบควบคุมแบบ ฟัซซีแบบ PI อีก 1 ชุดเมื่อไม่มีอินทิกรัลแอคชันมาหักล้าง Difference error จะทำหน้าที่เสมือน เดริเวทีฟแอคชัน และจะถูกแปลงให้เสมือนเป็น K_p K_i และ K_d ในระบบควบคุมแบบ PID จากนั้นปรับแต่งค่าเริ่มต้นของระบบควบคุมแบบฟัซซีแบบ PI+PD และระบบควบคุมแบบ PID เพื่อให้ได้ผลตอบสนองที่ดีที่สุดของระบบควบคุมทั้งสองแบบ ในที่นี้ระบบควบคุมแบบ PID ใช้ค่า $K_p = 20$, $K_i = \frac{K_p}{T_i} = 50$ และ $K_d = K_p T_d = 0.8$ สำหรับในส่วนของระบบควบคุมแบบฟัซซีแบบ PI+PD ปรับค่าฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม ดังรูปที่ 4.18



P	trampmf	[-3 -2 -1.25 0]
Z	trimf	[-1.25 0 1.25]
N	trampmf	[0 1.25 2 3]



oN	trampmf	[-8 -4 -0.15 0]
oZ	trimf	[-0.15 0 0.15]
oP	trampmf	[0 0.15 4 8]



oNB	trapmf	[-0.433 -0.431 -0.429 -0.427]
oNS	trapmf	[-0.015 -0.006 -0.004 -0.0005]
oZ	trimf	[-0.0001 0 0.0001]
oPS	trapmf	[-0.005 0.004 0.006 0.015]
oPB	trapmf	[0.427 0.429 0.431 0.433]

(ก) รูปแสดงการปรับแต่งฟังก์ชันการเป็นสมาชิกฟัซซี PI

If (error is Z) then (dvoltage is oZ) (1)

If (error is P) then (dvoltage is oNB) (1)

If (error is N) then (dvoltage is oPB) (1)

If (error is P) and (derror is dZ) then (dvoltage is oNS) (1)

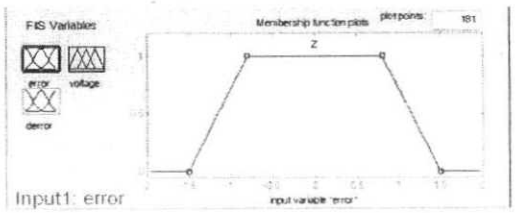
If (error is P) and (derror is dN) then (dvoltage is oNS) (1)

If (error is N) and (derror is dZ) then (dvoltage is oPS) (1)

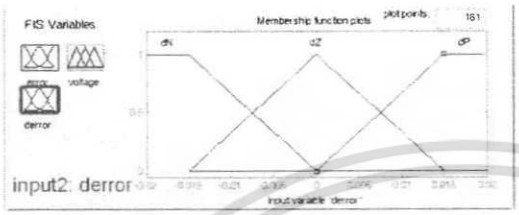
If (error is N) and (derror is dP) then (dvoltage is oPS) (1)

(ข) กฎการควบคุมฟัซซี PI

รูปที่ 4.18 ค่าการปรับแต่ง ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม ของระบบควบคุมแบบ ฟัซซีแบบ PI ของระบบสองมวล



Z	trapmf	[-1.5 -0.8 0.8 1.5]



dN	trampmf	[-0.8 -0.4 -0.015 0]
dZ	trimf	[-0.015 0 0.015]
dP	trampmf	[0 0.015 0.4 0.8]

oN	trapmf	[-0.117 -0.116 -0.115 -0.114]
oZ	trapmf	[-0.00013 -0.00012 0.00012 0.00013]
oP	trapmf	[0.114 0.115 0.116 0.117]

(ก) รูปแสดงการปรับแต่งฟังก์ชันการเป็นสมาชิกฟัซซี PD

If (error is Z) and (derror is dZ) then (dvoltage is oZ) (1)

If (error is Z) and (derror is dN) then (dvoltage is oNB) (1)

If (error is Z) and (derror is dP) then (dvoltage is oPB) (1)

(ข) กฎการควบคุมฟัซซี PD

รูปที่ 4.19 ค่าการปรับแต่ง ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม ของระบบควบคุมแบบ ฟัซซีแบบ PD ของระบบสองมวล

บันทึกค่าเอาท์พุท ของระบบทั้งสอง เพื่อนำมาเปรียบเทียบกัน

สรุป และ วิจารณ์ ผลการทดลอง

รูปที่ 4.20 ผลการทดลองเปรียบเทียบระหว่าง ฟัซซีแบบ PI+PD กับระบบควบคุมแบบ PID ของระบบสองมวล

จากรูปที่ 4.20 แสดงว่าระบบควบคุมแบบ ฟัซซีแบบ PI+PD ให้เอาท์พุทด้อยกว่าระบบควบคุมแบบ PID เนื่องจากการเขียนระบบให้ ฟัซซีเป็น PID เลยนั้น มีความยากต่อการเขียนโปรแกรมมาก จึงได้ปรับเปลี่ยนเป็น PI+PD แทน และจากการที่มี อินพุท ในระบบ มากขึ้นจึงทำให้ยากต่อการกำหนดค่า ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม ที่เหมาะสมเนื่องจากการขาดประสบการณ์ของผู้ทดลอง ดังนั้นผลที่ออกมาจึงด้อยกว่าระบบควบคุมแบบ PID จึงสนับสนุนแนวคิดที่ว่า หากผู้ปรับไม่มีความเชี่ยวชาญในระบบเพียงพอ จะทำให้ปรับแต่งตัวควบคุมแบบฟัซซีได้ยาก

4.2 การทดลองเขียนโปรแกรมด้วยภาษา C เพื่อควบคุมระบบสองมวลตามที่ได้ออกแบบไว้

ที่มาของการทดลอง เพื่อทดสอบว่าระบบควบคุมแบบฟัซซี สามารถใช้งานได้จริงตามที่ได้ออกแบบไว้ในการทดลองที่ 4.1 จริง จึงใช้ภาษา C เขียนโปรแกรมเพื่อควบคุมระบบสองมวลจากเครื่องคอมพิวเตอร์ และดูผลลัพธ์ที่ได้ว่ามีความใกล้เคียงกับที่ได้ออกแบบไว้ หรือไม่

ขั้นตอนการทดลอง

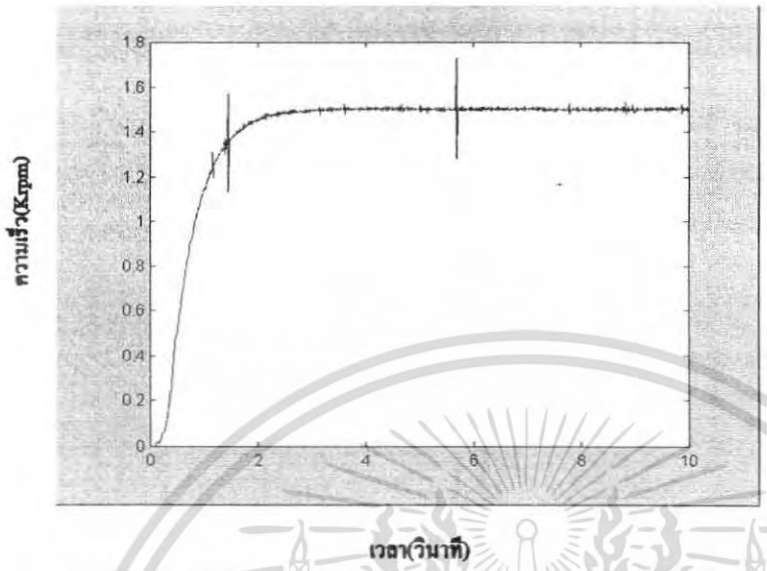
เขียนโปรแกรมด้วยภาษา C เพื่อสร้างระบบควบคุมแบบป้อนกลับด้วยระบบควบคุมแบบฟัซซีแบบ I ฟัซซีแบบ PI และระบบควบคุมแบบ ฟัซซีแบบ PI+PD ตามที่ได้ออกแบบไว้จาก Simulink ในโปรแกรม MATLAB

```
//membership function of output of fuzzy_PI
#define cN3_pi  -0.43
#define cN5_pi  -0.005
#define cZ_pi    0
#define cP3_pi   0.005
#define cP3_pi   0.43
//membership function of output of fuzzy_PD
#define cN_pd  -0.115
#define cZ_pd   0
#define cP_pd   0.115

float value_error=0,value_derror=0;
//fuzzy-PI
float uP3_pi=0,uP_pi=0,uZ_pi=0,uN_pi=0,uN3_pi=0;  //degree of membership function of error of fuzz
float udN_pi=0,udZ_pi=0,udP_pi=0;                 //degree of membership function of derror of fu
float uoN3_pi=0,uoN5_pi=0,uoZ_pi=0,uoP5_pi=0,uoP3_pi=0;    //degree of membership function of anal
float temp_uoN5[2]={0,0},temp_uoP5[2]={0,0};               //temp variable in rule base
//fuzzy-PD
float uP_pd=0,uZ_pd=0,uN_pd=0;                //degree of membership function of error of fuzzy-PD
float udN_pd=0,udZ_pd=0,udP_pd=0;             //degree of membership function of derror of fuzzy-P
float uoN_pd=0,uoZ_pd=0,uoP_pd=0;             //degree of membership function of dvoltage of fuzzy

void fuzzifer(unsigned char case_con,float value);
void rule_base();
void inference();
float defuzzifier();
float min(float value1,float value2);
float max(float value1,float value2);
void  clear_uMF();
float u_MF(unsigned char case_con,unsigned char mf_con,float value);
```

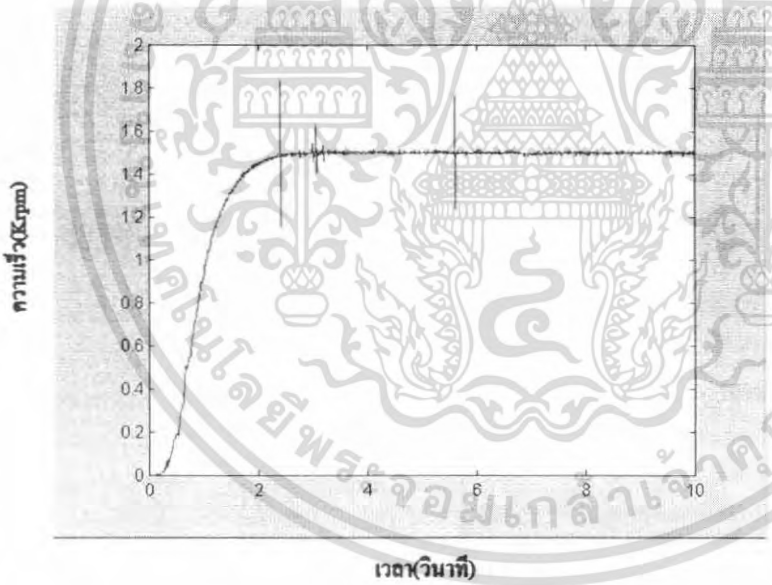
รูปที่ 4.21 ตัวอย่างการเขียนโปรแกรมด้วยภาษา C

บันทึกค่าเอาท์พุท ของระบบทั้งสาม เพื่อดูผลเปรียบเทียบ กับ ผลการจำลองที่ผ่านมา

สรุป และ วิจารณ์ ผลการทดลอง



(ก) ผลจากการทดลองการควบคุมระบบสองมวลแบบ ฟัซซีแบบ I



(ข) ผลจากการทดลองการควบคุมระบบสองมวลแบบ ฟัซซีแบบ PI

รูปที่ 4.22 ผลจากการเขียนโปรแกรมภาษา C เพื่อควบคุมระบบสองมวล

จากการทดลองสรุปได้ว่าผลที่ได้จากการเขียนโปรแกรม มีค่าใกล้เคียงกับที่จำลองไว้ แม้จะเกิดสัญญาณรบกวน ซึ่งทำให้ค่าการอ่านสัญญาณเพี้ยนไป

4.3 การทดลองควบคุมหุ่นยนต์โดยไมโครคอนโทรลเลอร์

ในการทดลองนี้มีจุดประสงค์เพื่อประยุกต์การใช้งานของระบบควบคุมอัตโนมัติแบบฟัซซี ซึ่งเป็นอินทิลิเจนท์คอนโทรลที่มีจุดเด่นหลายประการ โดยประยุกต์ใช้กับระบบควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง เพื่อใช้ในระบบควบคุมทิศทางการเคลื่อนที่แบบเส้นตรงของหุ่นยนต์ให้สามารถใช้งานได้จริง ซึ่งทำการทดลองดังนี้

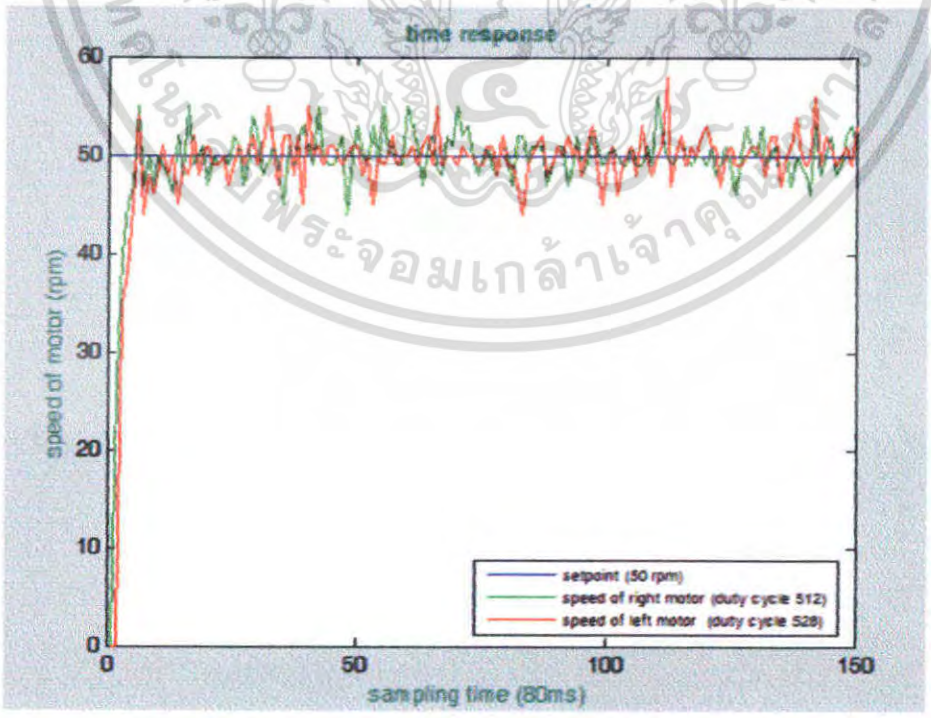
4.3.1 เพื่อทดสอบระบบควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบเปิด

ที่มาของการทดลอง เนื่องจากระบบควบคุมแบบเปิด เป็นระบบพื้นฐานก่อนจะพัฒนาไปสู่ระบบควบคุมแบบอื่น จึงทำการทดลองเพื่อบันทึกผลไว้สำหรับเปรียบเทียบกับระบบควบคุมในลำดับต่อไป

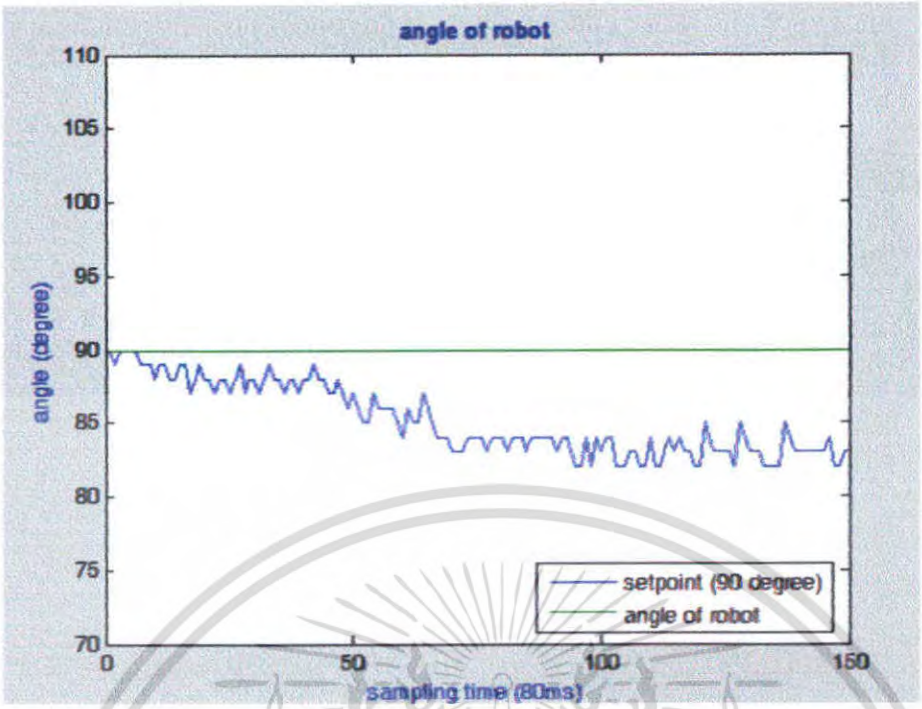
ขั้นตอนการทดลอง

เขียนโปรแกรมระบบควบคุมแบบเปิดควบคุมระบบผ่านไมโครคอนโทรลเลอร์ด้วยภาษา C จากนั้นทดลองให้หุ่นยนต์วิ่งบนสนามที่ได้จัดไว้ เลือกใช้ค่าดิวตี้ไซเคิล PWM 525 และ 510 เพื่อควบคุมความเร็วที่ 50 รอบต่อนาที ของมอเตอร์ด้านซ้ายและขวาตามลำดับ จากนั้นบันทึกค่าผลลัพธ์ที่ได้ไว้ และนำค่าที่ได้มาวาดกราฟเปรียบเทียบความเร็วที่ได้จากการทดลอง กับ ความเร็วที่ 50 รอบต่อนาที

สรุป และ วิจารณ์ ผลการทดลอง



(ก) ความเร็วของหุ่นยนต์



(ข) ทิศทางการเคลื่อนที่ของหุ่นยนต์

รูปที่ 4.23 ผลการทดลองควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบเปิด

จากรูปที่ 4.23 ผลที่ได้จะเป็นกราฟแสดงผลตอบสนองในการเข้าสู่เสถียรภาพของความเร็วของมอเตอร์ไฟฟ้ากระแสตรงทั้ง 2 ข้างที่วัดได้ (กราฟสีเขียว : ด้านขวา – กราฟสีแดง : ด้านซ้าย) เทียบกับ ค่าอ้างอิง (กราฟสีน้ำเงิน) พบว่าเมื่อใช้ระบบควบคุมแบบเปิดนั้นไม่สามารถเข้าสู่ ค่าอ้างอิง ที่ต้องการได้ และยังเกิดการแกว่ง แม้ว่าจะมีค่าเวลาเข้า t_s น้อยก็ตาม จึงสรุปได้ว่าไม่สมควรที่จะใช้ระบบควบคุมแบบเปิดในระบบควบคุม

4.3.2 เพื่อทดสอบระบบควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบ PI

ที่มาของการทดลอง เนื่องจากผลการทดลองที่ 4.3.1 สามารถสรุปได้ว่าไม่สามารถใช้ระบบควบคุมแบบเปิดได้ จึงได้ทดลองพัฒนาระบบควบคุมแบบ PI ขึ้นมาเพื่อทดลองใช้ควบคุมแทนระบบควบคุมแบบเปิด และบันทึกไว้เปรียบเทียบกับระบบควบคุมในลำดับต่อไป

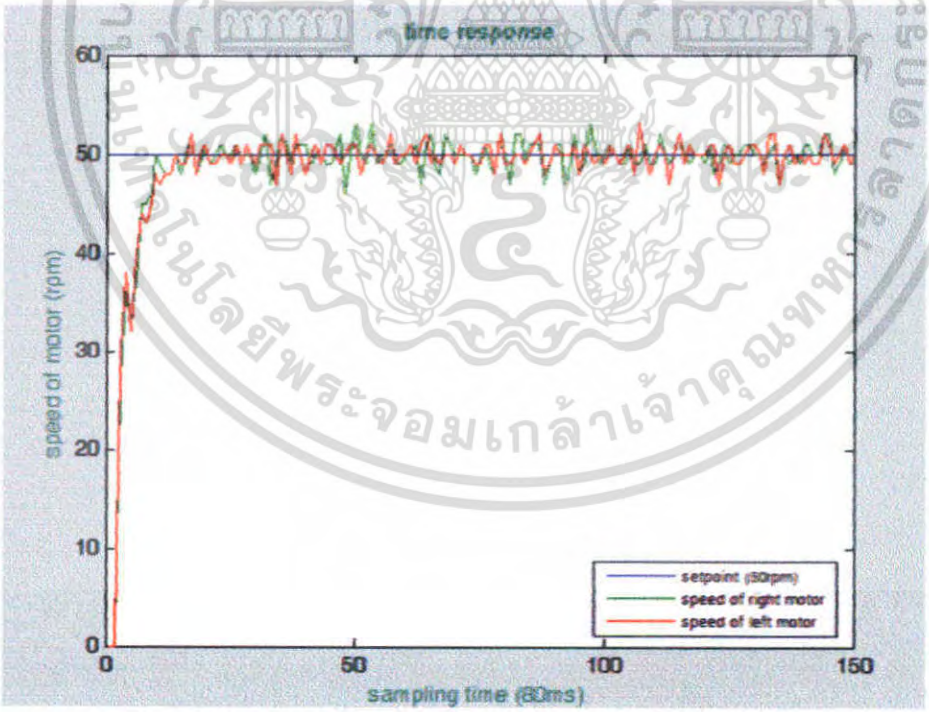
ขั้นตอนการทดลอง

เขียนโปรแกรมระบบควบคุมแบบ PI ควบคุมระบบผ่านไมโครคอนโทรลเลอร์ด้วยภาษา C ทดลองให้หุ่นยนต์วิ่งบนสนามที่ได้จัดไว้ และบันทึกค่าผลลัพธ์ที่ได้ไว้ และนำค่าที่ได้มาวาดกราฟเปรียบเทียบความเร็วที่ได้จากการทดลอง กับ ความเร็วที่ต้องการ หากยังไม่ได้ผลที่น่าพอใจให้ทดลองปรับแต่งอัตราขยาย K_p และ K_i และกลับไปทำซ้ำจนกว่าจะได้ผลที่น่าพอใจ

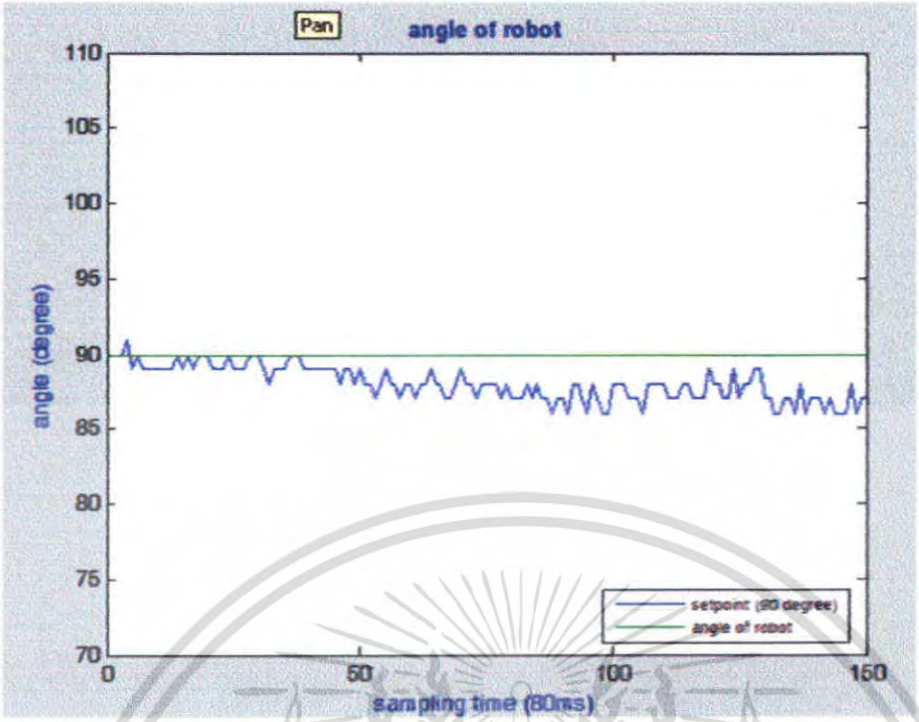
โปรแกรมส่วนที่ใช้ควบคุมแบบ PI

```
void PI_controller(signed int setpoint) {
  signed int error_speed,duty_output=0;
  error_speed=setpoint-real_speed;
  if((Kp*error_speed)>30000) {
    duty_output=30000;
  } else if((Kp*error_speed)<-30000) {
    duty_output=-30000;
  } else {
    duty_output=Kp*error_speed;
  }
  duty_acc+=Ki*error_speed;
  if(duty_acc>30000) {
    duty_acc=30000;
  } else if(duty_acc<-30000) {
    duty_acc=-30000;
  }
  duty_output+=duty_acc;
  if(duty_output>30000) {
    duty_output=30000;
  } else if(duty_output<-30000) {
    duty_output=-30000;
  }
  speed_output(3,duty_output);      //set pwm duty
}
```

สรุป และ วิจารณ์ ผลการทดลอง



(ก) ความเร็วของหุ่นยนต์



(ข) ทิศทางการเคลื่อนที่ของหุ่นยนต์

รูปที่ 4.24 ผลการทดลองควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบ PI

จากรูปที่ 4.24 ผลที่ได้จะเป็นกราฟแสดงผลตอบสนองในการเข้าสู่เสถียรภาพของความเร็วของมอเตอร์ไฟฟ้ากระแสตรงทั้ง 2 ข้างที่วัดได้ (กราฟสีเขียว : ด้านขวา – กราฟสีแดง : ด้านซ้าย) เทียบกับ ค่าอ้างอิง (กราฟสีน้ำเงิน) พบว่าเมื่อใช้ระบบควบคุมแบบ PI นั้นสามารถเข้าสู่ ค่าอ้างอิง ที่ต้องการได้ แม้ว่าจะยังเกิดค่าการแกว่งบ้างก็ตาม จึงสรุปได้ว่าระบบควบคุมแบบ PI สามารถใช้ในระบบควบคุมได้จริงแต่ยังมีค่าค่าเวลาเข้า t_s สูงอยู่มาก และทิศทางในการเคลื่อนที่ยังมีความผิดพลาดอยู่บ้าง

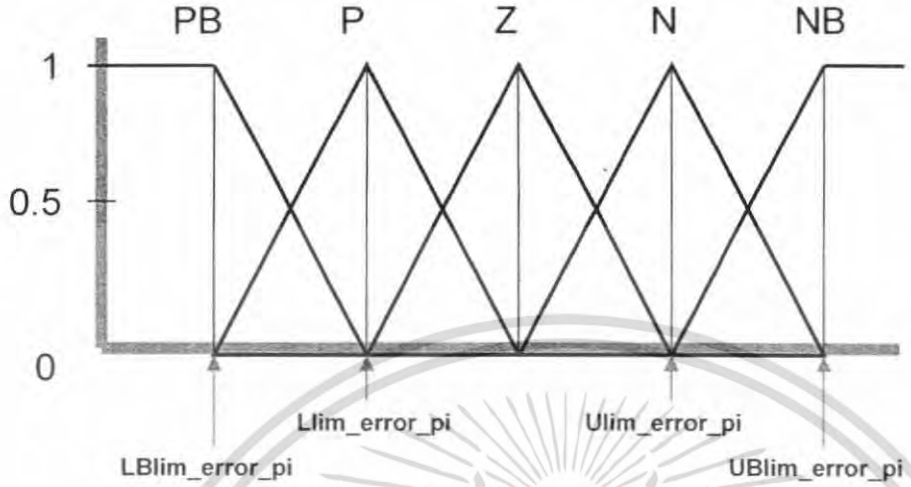
4.3.3 เพื่อทดสอบระบบควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบ ฟัซซี PI

ที่มาของการทดลอง เนื่องจากผลการทดลองที่ 2 ยังมีค่าเวลาเข้า t_s สูงอยู่มาก จึงได้ทดลองพัฒนาระบบควบคุมแบบ ฟัซซี PI ขึ้นมาเพื่อทดลองใช้ควบคุมแทนระบบควบคุมแบบ PI และบันทึกไว้เปรียบเทียบกับระบบควบคุมในลำดับต่อไป

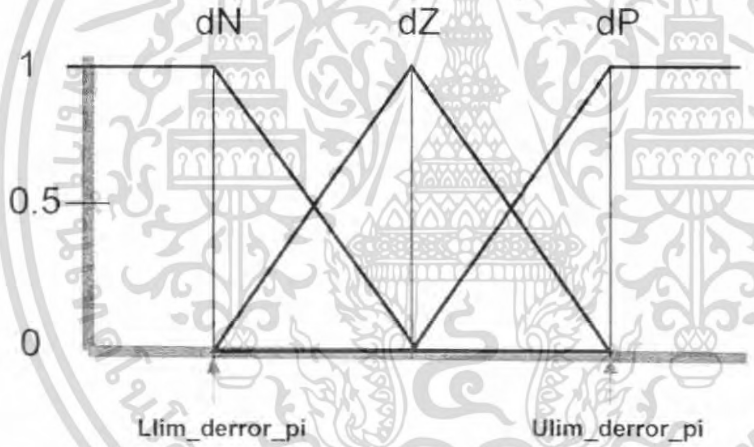
ขั้นตอนการทดลอง

เขียนโปรแกรม ระบบควบคุมแบบฟัซซี PI ควบคุมผ่านไมโครคอนโทรลเลอร์ ด้วยภาษา C ทดลองให้หุ่นยนต์วิ่งบนสนามที่ได้จัดไว้ และบันทึกค่าผลลัพธ์ที่ได้ไว้ และนำค่าที่ได้มาวาด กราฟเปรียบเทียบความเร็วที่ได้จากการทดลอง กับ ความเร็วที่ต้องการหากยังไม่ได้ผลที่น่าพอใจ ให้

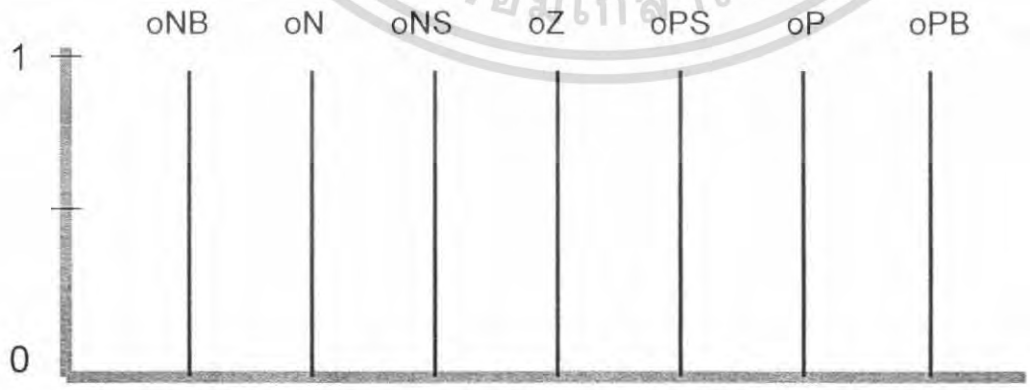
ทดลองปรับแต่งค่า ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม จากนั้นกลับไปทำซ้ำจนกว่าจะได้ผลที่น่าพอใจ



รูปที่ 4.25 ฟังก์ชันการเป็นสมาชิกของอินพุท error PI



รูปที่ 4.26 ฟังก์ชันการเป็นสมาชิกของอินพุท derror PI



รูปที่ 4.27 ฟังก์ชันความเป็นสมาชิกของเอาท์พุท ดิวตี้ไซเคิลจาก PWM PI

โปรแกรมส่วนที่ใช้ควบคุมแบบฟัซซี PI

```
//fuzzy PI controller
void FLC_PI(signed int setpoint,signed char speed_R,signed char speed_L) {
  signed int error_R=0,error_L=0;
  error_R=setpoint-speed_R;
  error_L=setpoint-speed_L;
  op_PI_R+=(signed int)FLC(error_R,(error_R-last_error_R),F_PI);
  op_PI_L+=(signed int)FLC(error_L,(error_L-last_error_L),F_PI);
  //limit
  if(op_PI_R>30000) {           //upper limit
    op_PI_R=30000;
  } else if(op_PI_R<-30000) {   //lower limit
    op_PI_R=-30000;
  }

  if(op_PI_L>30000) {           //upper limit
    op_PI_L=30000;
  } else if(op_PI_L<-30000) {   //lower limit
    op_PI_L=-30000;
  }

  spi_send16_s(1,change_duty16,op_PI_R);    //R
  spi_send16_s(2,change_duty16,op_PI_L);    //L

  last_error_R=error_R;
  last_error_L=error_L;
```

ค่าที่เลือกใช้ในการเลือกค่าความเป็นสมาชิกฟัซซี PI

```
  //limit of graph degree of membership
  //error
  #define Llim_error  -9
  #define Ulim_error   9
  #define m_error      0.125
  #define LBlim_error -35
  #define UBlim_error  35
  #define mB_error     0.0370 //slope error  = 1/(UBlim_error-Ulim_error) =1/12
  #define c1          -0.2963 //c=-(-mB_error*Llim_error)  PB,NB line
  #define c2           1.2963 //c=-(mB_error*LBlim_error)  P,N   line
  //derror
  #define Llim_derror -20
  #define Ulim_derror  20
  #define m_derror     0.05

  //membership function of output   dduty >> (-100 - 100)
  #define oNB -225
  #define oN  -13
  #define oNS -10
  #define oZ   0
  #define oPS  10
  #define oP   13
  #define oPB  225
```

กฎที่ใช้

rule 1: if(error is Z) then (dvoltage is oZ)

rule 2: if(error is PB) then (dvoltage is oNB)

rule 3: if(error is NB) then (dvoltage is oPB)

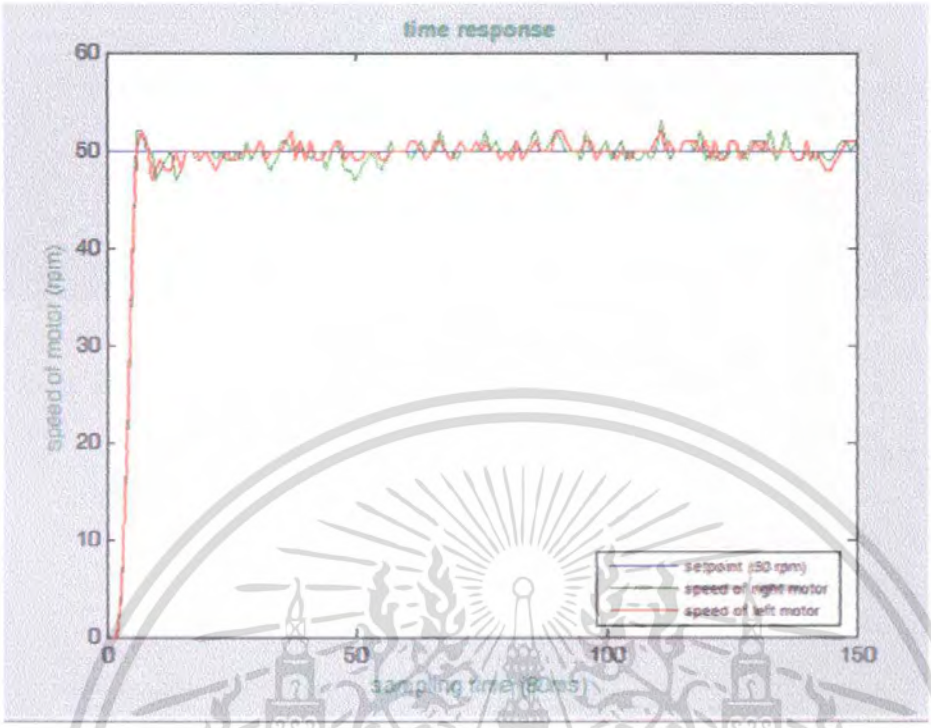
rule 4: if(error is P) then (dvoltage is oN)

rule 5: if(error is N) then (dvoltage is oP)

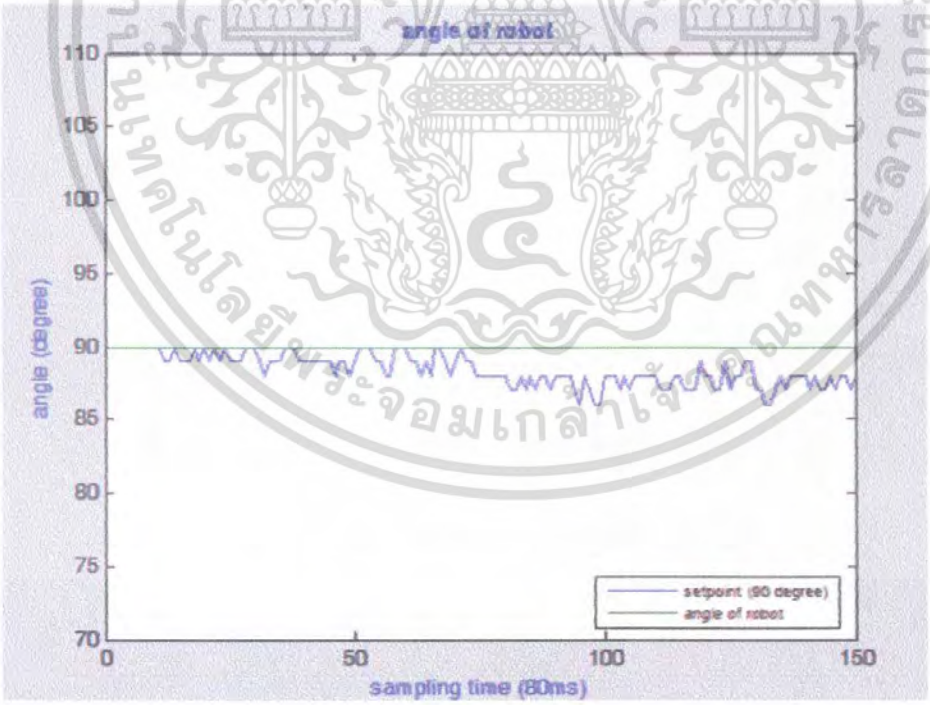
rule 6: if(error is P) and (derror is dN) then (dvoltage is oNS)

rule 7: if(error is N) and (derror is dP) then (dvoltage is oPS)

สรุป และ วิจารณ์ ผลการทดลอง



(ก) ความเร็วของหุ่นยนต์



(ข) ทิศทางการเคลื่อนที่ของหุ่นยนต์

รูปที่ 4.28 ผลการทดลองควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบฟัซซี PI

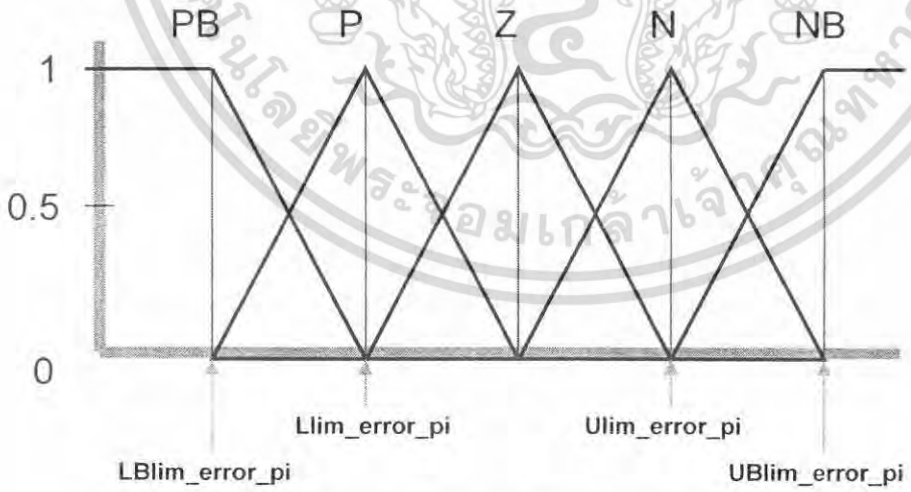
จากผลที่ได้จะเป็นกราฟแสดงผลตอบสนอง ในการเข้าสู่เสถียรภาพของความเร็วของมอเตอร์ไฟฟ้ากระแสตรงทั้ง 2 ข้างที่วัดได้ (กราฟสีเขียว : ด้านขวา – กราฟสีแดง : ด้านซ้าย) เทียบกับ ค่าอ้างอิง (กราฟสีน้ำเงิน) พบว่าเมื่อใช้ระบบควบคุมแบบ ฟัซซี PI นั้นสามารถเข้าสู่ ค่าอ้างอิง ที่ต้องการได้ แม้ว่าจะยังเกิดการสั่นของกราฟ บ้างก็ตาม จึงสรุปได้ว่าระบบควบคุมแบบ ฟัซซี PI สามารถใช้ในการควบคุมได้จริง แม้จะมีค่าค่าเวลาเข้า t_s น้อยกว่า ระบบควบคุมแบบ PI และยังมีค่าพุ่งเกิน เกิดขึ้นตามมาด้วย และทิศทางในการเคลื่อนที่ยังมีความผิดพลาดอยู่บ้างแต่ก็น้อยกว่าการทดลองที่ 4.3.2

4.3.4 เพื่อทดสอบการควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบ ฟัซซี PI – I

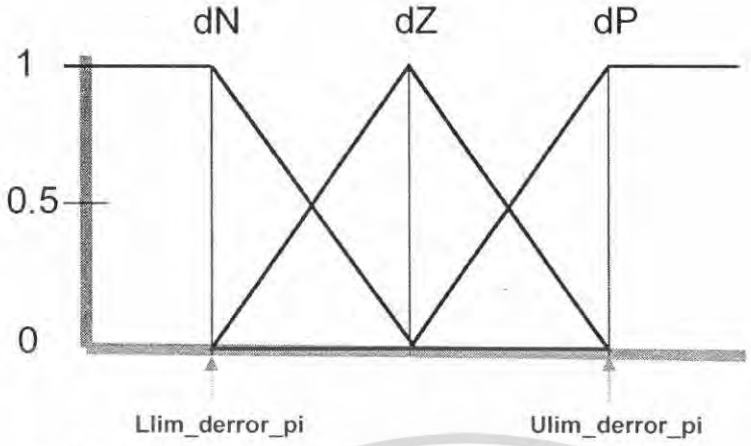
ที่มาของการทดลอง เนื่องจากผลการทดลองที่ 3 ยังมี ค่าพุ่งเกิน เกิดขึ้นจึงได้ทดลองพัฒนาระบบควบคุมแบบฟัซซี PI - I ขึ้นมาเพื่อทดลองใช้ควบคุมแทนระบบควบคุมแบบ ฟัซซี PI และบันทึกไว้เปรียบเทียบกับระบบควบคุมในลำดับต่อไป

ขั้นตอนการทดลอง

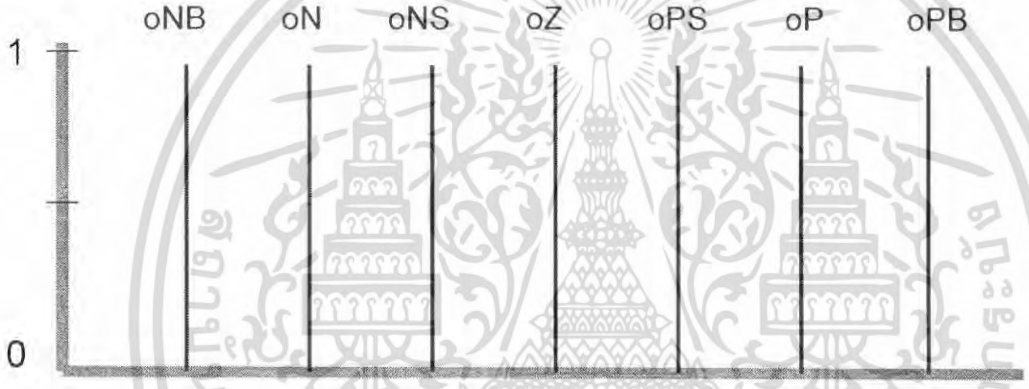
เขียนโปรแกรมระบบควบคุมแบบ ฟัซซี PI - I ควบคุมผ่านไมโครคอนโทรลเลอร์ด้วยภาษา C ทดลองให้หุ่นยนต์วิ่งบนสนามที่ได้จัดไว้ และบันทึกค่าผลลัพธ์ที่ได้ไว้ และนำค่าที่ได้มาวาดกราฟเปรียบเทียบความเร็วที่ได้จากการทดลอง กับ ความเร็วที่ต้องการ หากยังไม่ได้ผลที่น่าพอใจ ให้ทดลองปรับแต่งค่า ฟังก์ชันการเป็นสมาชิก และ กฎการควบคุม จากนั้นกลับไปทำซ้ำจนกว่าจะได้ผลที่น่าพอใจ



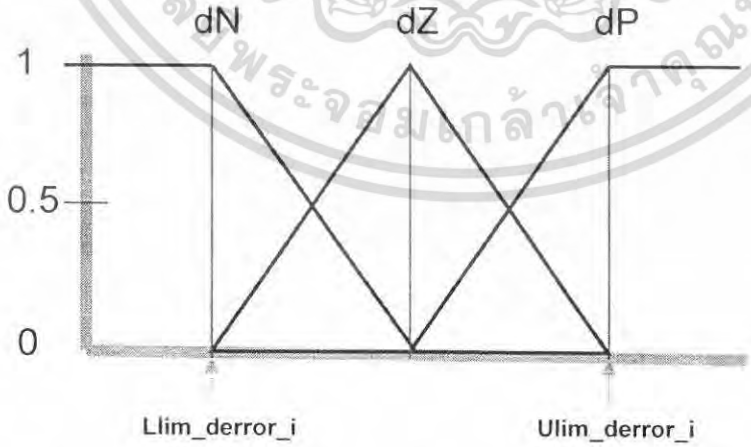
รูปที่ 4.29 ฟังก์ชันความเป็นสมาชิกของอินพุท error ฟัซซี PI



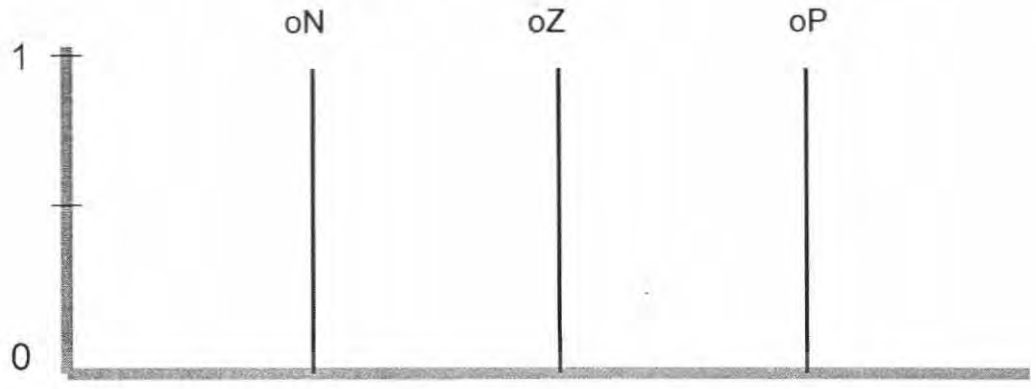
รูปที่ 4.30 ฟังก์ชันความเป็นสมาชิกของอินพุท derror ฟัซซี PI



รูปที่ 4.31 ฟังก์ชันความเป็นสมาชิกของเอาท์พุท ดิวตี้ไซเคิลจาก PWM ฟัซซี PI



รูปที่ 4.32 ฟังก์ชันความเป็นสมาชิกของอินพุท error ฟัซซี I



รูปที่ 4.33 ฟังก์ชันความเป็นสมาชิกของเอาท์พุทดิวตี้ไซเคิลจาก PWM ฟัซซี I

โปรแกรมส่วนที่ใช้ควบคุมแบบฟัซซี PI-I

```
//fuzzy pi-i controller
void FLC_PI_I signed int setpoint signed char speed_R signed char speed_L
  signed int fw_path_L fw_path_R
  op_I = signed int FLC speed_L speed_R F_I
  if op_I
    op_I
    else if op_I
    op_I

  //Right motor path
  fw_path_R setpoint op_I speed_R
  if fw_path_R
    fw_path_R
    else if fw_path_R
    fw_path_R
  //Left motor path
  fw_path_L setpoint op_I speed_L
  if fw_path_L
    fw_path_L
    else if fw_path_L
    fw_path_L

  op_PI_R = signed int FLC fw_path_R fw_path_R last_fw_path_R F_PI
  op_PI_I = signed int FLC fw_path_L fw_path_L last_fw_path_L F_PI
  spi_send16_s change_duty16 op_PI_R
  spi_send16_s change_duty16 op_PI_I

  last_fw_path_R fw_path_R
  last_fw_path_L fw_path_L
```

ค่าที่เลือกใช้ในการเลือกค่าความเป็นสมาชิกฟัซซี PI-I

```
//error
#define Llim_error_pi  -10
#define Ulim_error_pi   10
#define m_error_pi   0.1    //slope error  = 1/10
#define LBlim_error_pi -60
#define UBlim_error_pi  60
#define mB_error_pi      0.02   //slope error  = 1/(UBlim_error-Ulim_error) =1/50
#define c1_pi           -0.2    //c=-(-mB_error*Llim_error)  PB,NB line
#define c2_pi            1.2    //c=-(mB_error*LBlim_error)  P,N   line
//derror
#define Llim_derror_pi -20
#define Ulim_derror_pi  20
#define m_derror_pi  0.05   //slope derror = 1/20
//membership function of output   dduty >> (-100 - 100)
#define oNB_pi -235
#define oN_pi  -15
#define oNS_pi -12
#define oZ_pi   0
#define oPS_pi  12
#define oP_pi   15
#define oPB_pi  235

//Fuzzy I limit of graph degree of membership
#define Llim_error_i  -22
#define Ulim_error_i   22
#define m_error_i    0.045455   //slope error  = 1/22
//membership function of output
#define oN_i  -3
#define oZ_i   0
#define oP_i   3
```

กฎที่ใช้

ในส่วนของฟัซซี PI

rule 1: if(error is Z) then (dvoltage is oZ)

rule 2: if(error is PB) then (dvoltage is oNB)

rule 3: if(error is NB) then (dvoltage is oPB)

rule 4: if(error is P) then (dvoltage is oN)

rule 5: if(error is N) then (dvoltage is oP)

rule 6: if(error is P) and (derror is dN) then (dvoltage is oNS)

rule 7: if(error is N) and (derror is dP) then (dvoltage is oPS)

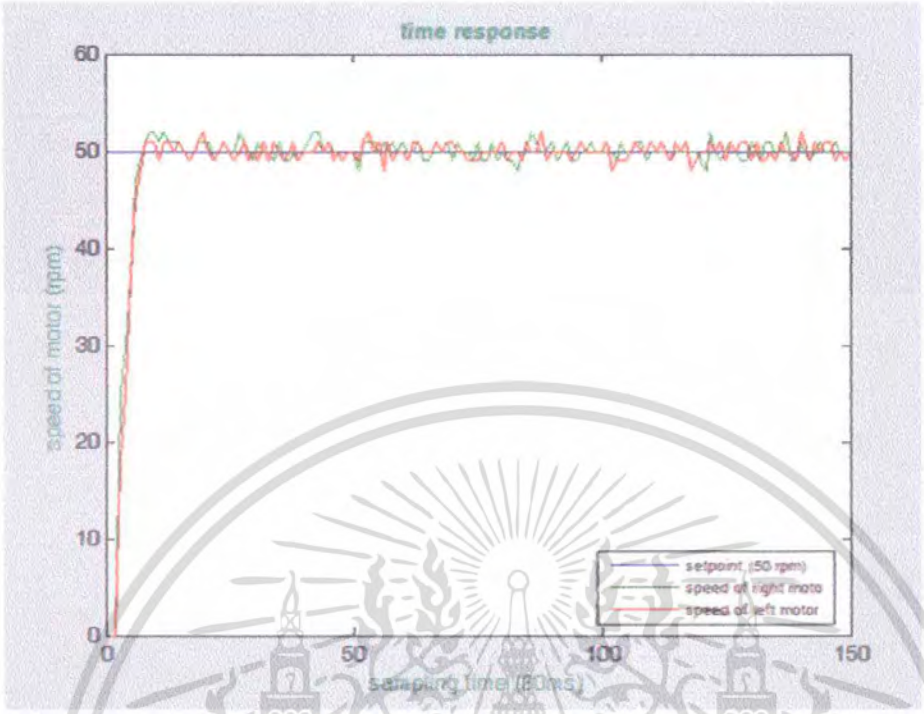
ในส่วนของฟัซซี I

rule 1: if(error is Z) then (voltage is oZ)

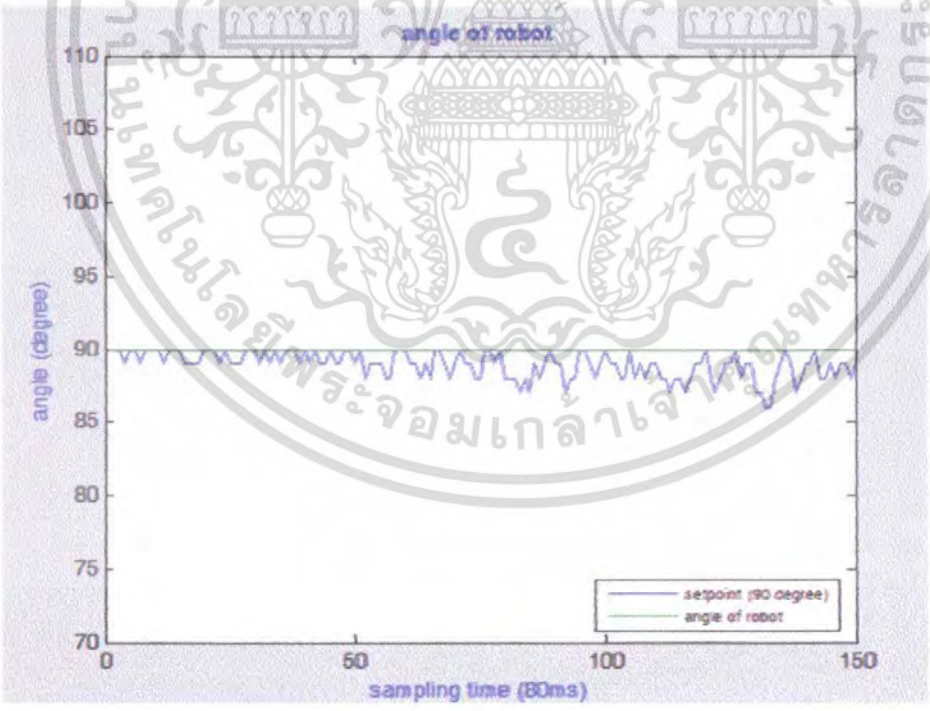
rule 2: if(error is P) then (dvoltage is oN)

rule 3: if(error is N) then (dvoltage is oP)

สรุป และ วิจารณ์ ผลการทดลอง



(ก) ความเร็วของหุ่นยนต์



(ข) ทิศทางการเคลื่อนที่ของหุ่นยนต์

รูปที่ 4.34 ผลการทดลองควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบฟัซซี PI - I

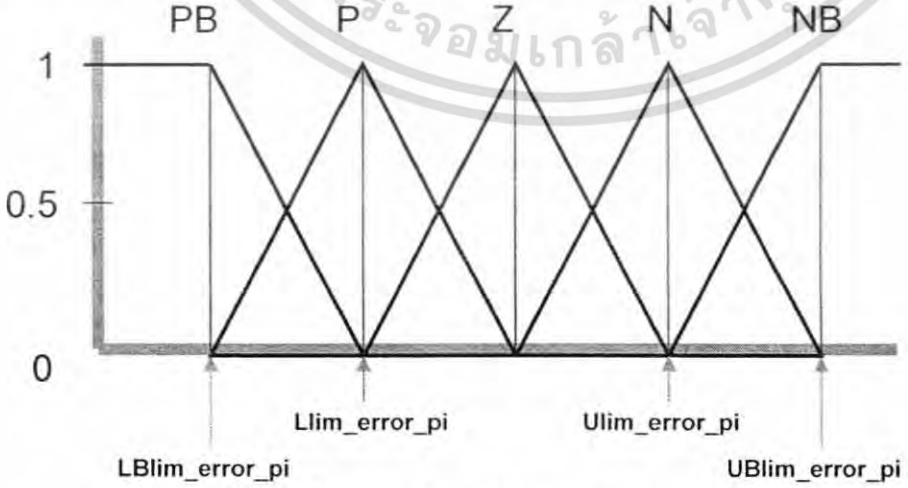
จากผลที่ได้จะเป็นกราฟแสดงผลตอบสนอง ในการเข้าสู่เสถียรภาพของความเร็วของมอเตอร์ไฟฟ้ากระแสตรงทั้ง 2 ข้างที่วัดได้ (กราฟสีเขียว : ด้านขวา – กราฟสีแดง : ด้านซ้าย) เทียบกับ ค่าอ้างอิง (กราฟสีน้ำเงิน) พบว่าเมื่อใช้ระบบควบคุมแบบ ฟัซซี PI - I นั้นสามารถเข้าสู่ ค่าอ้างอิง ที่ต้องการได้ แม้ว่าจะยังเกิดค่าการแกว่งบ้างก็ตาม จึงสรุปได้ว่าระบบควบคุมแบบฟัซซี PI - I สามารถใช้ในการควบคุมได้จริง แม้ในส่วนของการคุมทิศทางยังมีปัญหาจากการออกตัวของหุ่นยนต์เกิดแรงกระชาก ทำให้ทิศทางคลาดเคลื่อนบ้างก็ตาม แต่เมื่อนำผลที่ได้มาเทียบกับการทดลองที่ผ่านๆมา พบว่ามีค่าความผิดพลาดลดลง

4.3.5 เพื่อทดสอบการควบคุมความเร็วของหุ่นยนต์เมื่อใช้ระบบควบคุมแบบฟัซซี PI – I ร่วมกับ วงจรเข็มทิศดิจิตอล

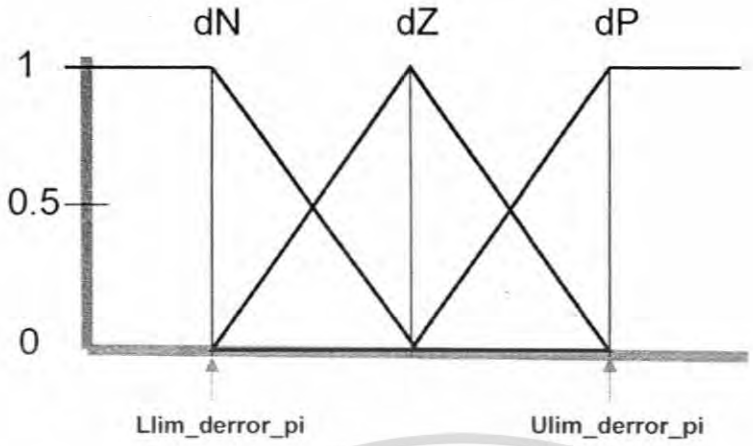
ที่มาของการทดลอง เนื่องจากผลการทดลองที่ 4.3.4 ยังมี ปัญหาเกิดขึ้นเนื่องจากเกิดแรงกระชากขึ้นในขณะที่เริ่มปล่อยตัวออกจากจุดเริ่มต้น ทำให้ทิศทางในการเคลื่อนที่คลาดเคลื่อนไปจากแนวเดิมที่ตั้งก่อนออกตัว ในการทดลองนี้จึงได้เพิ่มเติมในส่วนของ วงจรเข็มทิศดิจิตอล เข้าไปเพื่อช่วยในการปรับทิศทางการเคลื่อนที่ของหุ่นยนต์

ขั้นตอนการทดลอง

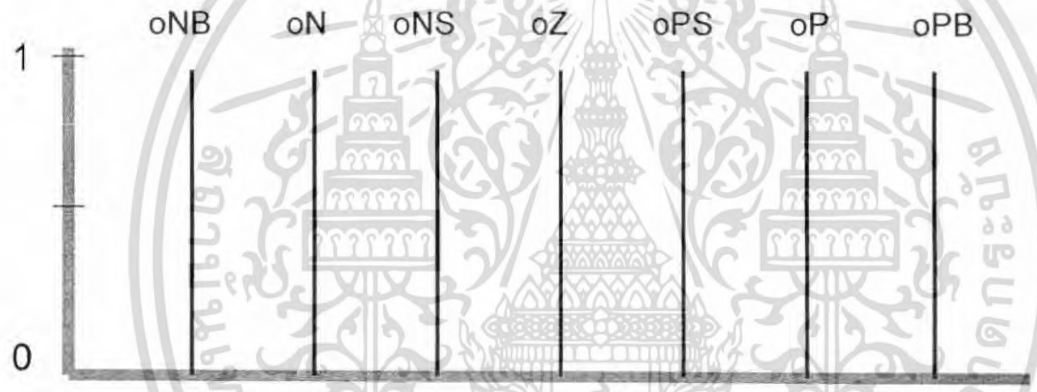
เขียนโปรแกรมระบบควบคุมแบบฟัซซี PI - I และวงจรเข็มทิศแบบดิจิตอล ควบคุมผ่านไมโครคอนโทรลเลอร์ด้วยภาษา C ทดลองให้หุ่นยนต์วิ่งบนสนามที่ได้จัดไว้ และบันทึกค่าผลลัพธ์ที่ได้ไว้ และนำค่าที่ได้มาวาดกราฟเปรียบเทียบความเร็วที่ได้จากการทดลอง กับความเร็วที่ต้องการ หากยังไม่ได้ผลที่น่าพอใจ ให้ทดลองปรับแต่งค่าฟังก์ชันการเป็นสมาชิก และกฎการควบคุม จากนั้นกลับไปทำซ้ำจนกว่าจะได้ผลที่น่าพอใจ



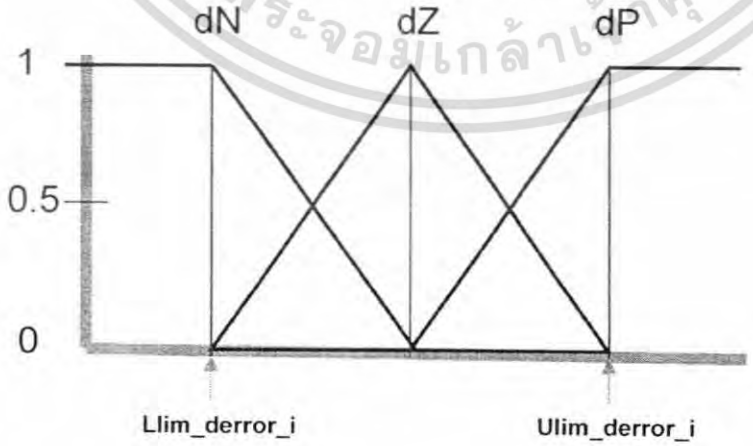
รูปที่ 4.35 ฟังก์ชันความเป็นสมาชิกของอินพุท error ฟัซซี PI



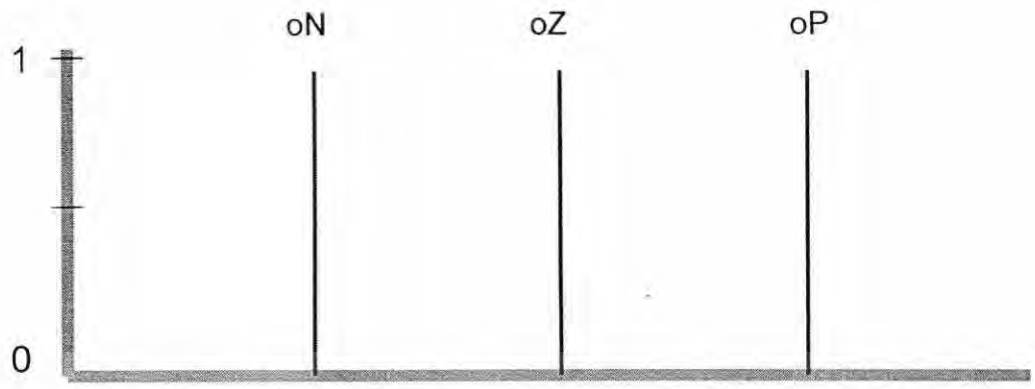
รูปที่ 4.36 ฟังก์ชันความเป็นสมาชิกของอินพุท derror ฟัซซี PI



รูปที่ 4.37 ฟังก์ชันความเป็นสมาชิกของเอาท์พุทดิวตี้ไซเคิลจาก PWM ฟัซซี PI



รูปที่ 4.38 ฟังก์ชันความเป็นสมาชิกของอินพุท error ฟัซซี I



รูปที่ 4.39 ฟังก์ชันความเป็นสมาชิกของเอาท์พุทดิวตี้ไซเคิลจาก PWM ฟัซซี I

โปรแกรมส่วนที่ใช้ควบคุมแบบฟัซซี PI-I ร่วมกับวงจรเข็มทิศดิจิตอล

```
//fuzzy PI-I controller with cmp sensor
void FLC_PI_I_cmp signed int setpoint signed char speed_R signed char speed_I unsigned int ang_st unsigned int ang_now
 signed int fw_path_I=0,fw_path_R=0
 signed int curve_offset=0
 unsigned char flag_ang=0
 if ang_now>ang_st                    //calculate curve offset
   flag_ang=0;
   curve_offset=ang_now-ang_st
  else
   flag_ang=1;
   curve_offset=ang_st-ang_now;
 }
 if curve_offset>180
   curve_offset=curve_offset-360;
 if flag_ang==1
   curve_offset*=-1;
 op_I+= signed int FLC speed_I-speed_R-curve_offset,F_I
 if op_I>30000                        //limit output I controller
   op_I=30000;
   else if op_I <-30000
   op_I=-30000;

 fw_path_R=setpoint-op_I-speed_R      //right motor path
 if fw_path_R>30000
   fw_path_R=30000;
   else if fw_path_R<-30000
   fw_path_R=-30000;

 fw_path_I=setpoint-op_I+speed_I      //left motor path
 if fw_path_I>30000
   fw_path_I=30000;
  else if fw_path_I<-30000
   fw_path_I=-30000;
 }
 op_PI_R+= signed int FLC fw_path_R,fw_path_R-last_fw_path_R,F_PI
 op_PI_I+= signed int FLC fw_path_I,fw_path_I-last_fw_path_I,F_PI
 spi_send16_s1,change_duty16,op_PI_R ;    //R
 spi_send16_s2,change_duty16,op_PI_I ;    //L
 last_fw_path_R=fw_path_R;
 last_fw_path_I=fw_path_I;
```

ค่าที่เลือกใช้ในการเลือกค่าความเป็นสมาชิกฟัซซี PI-I ร่วมกับวงจรเข็มทิศดิจิตอล

```
//error
#define Llim_error_pi  -10
#define Ulim_error_pi   10
#define m_error_pi    0.1    //slope error  = 1/10
#define LBlim_error_pi -60
#define UBlim_error_pi  60
#define mB_error_pi      0.02   //slope error  = 1/(UBlim_error-Ulim_error) =1/50
#define c1_pi           -0.2    //c=-(-mB_error*Llim_error)  PS,NS line
#define c2_pi            1.2    //c=-(mB_error*LBlim_error)  P,N   line
//derror
#define Llim_derror_pi -20
#define Ulim_derror_pi  20
#define m_derror_pi  0.05   //slope derror = 1/20
//membership function of output   dduty >> (-100 - 100)
#define oNB_pi -235
#define oN_pi  -15
#define oNS_pi -12
#define oZ_pi   0
#define oPS_pi  12
#define oP_pi   15
#define oPB_pi  235

//Fuzzy I limit of graph degree of membership
#define Llim_error_i  -22
#define Ulim_error_i   22
#define m_error_i    0.045455   //slope error  = 1/22
//membership function of output
#define oN_i  -6
#define oZ_i   0
#define oP_i   3
```

กฎที่ใช้

ในส่วนของฟัซซี PI

rule 1: if(error is Z) then (dvoltage is oZ)

rule 2: if(error is PB) then (dvoltage is oNB)

rule 3: if(error is NB) then (dvoltage is oPB)

rule 4: if(error is P) then (dvoltage is oN)

rule 5: if(error is N) then (dvoltage is oP)

rule 6: if(error is P) and (derror is dN) then (dvoltage is oNS)

rule 7: if(error is N) and (derror is dP) then (dvoltage is oPS)

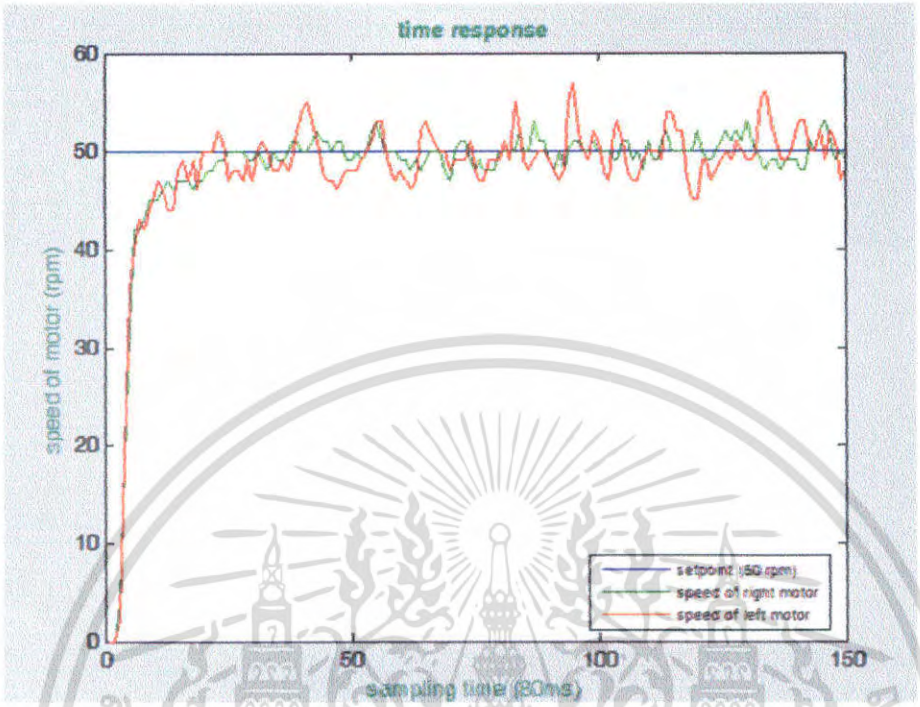
ในส่วนของฟัซซี I

rule 1: if(error is Z) then (voltage is oZ)

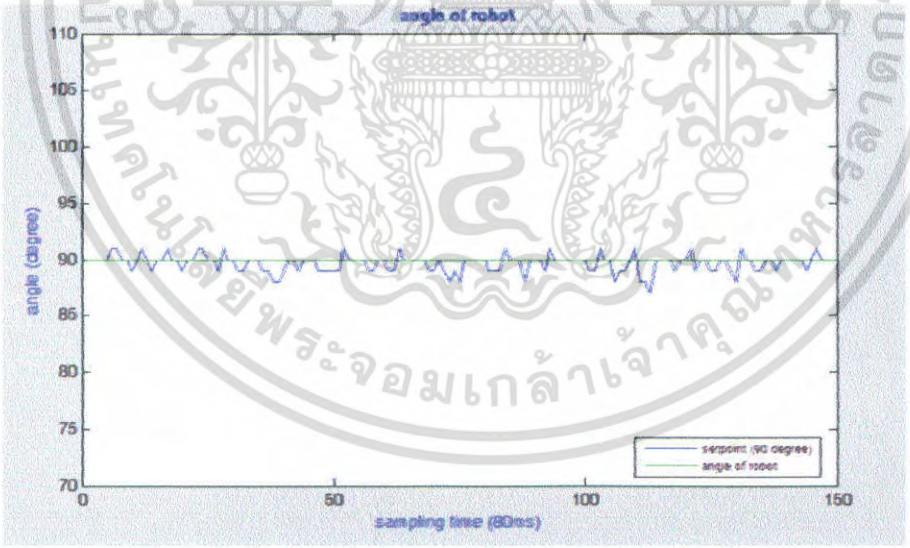
rule 2: if(error is P) then (dvoltage is oN)

rule 3: if(error is N) then (dvoltage is oP)

สรุป และ วิจารณ์ ผลการทดลอง



(ก) ความเร็วของหุ่นยนต์



(ข) ทิศทางการเคลื่อนที่ของหุ่นยนต์

รูปที่ 4.40 ผลการทดลองควบคุมความเร็วและทิศทางของหุ่นยนต์
เมื่อใช้ระบบควบคุมแบบ ฟัซซี PI – I ร่วมกับวงจรเข็มทิศดิจิตอล

จากผลที่ได้จะเป็นกราฟแสดงผลตอบสนองในการเข้าสู่เสถียรภาพของความเร็วของมอเตอร์ไฟฟ้ากระแสตรงทั้ง 2 ข้างที่วัดได้ (กราฟสีเขียว : ด้านขวา – กราฟสีแดง : ด้านซ้าย) เทียบกับ ค่าอ้างอิง (กราฟสีน้ำเงิน) พบว่าเมื่อใช้ระบบควบคุมแบบ ฟัซซี PI - I ร่วมกับ วงจรเข็มดิจิตอล นั้นสามารถเข้าสู่ ค่าอ้างอิง ที่ต้องการได้ แม้ว่าจะยังเกิดค่า การแกว่ง บ้างก็ตาม จึงสรุปได้ว่าระบบควบคุมแบบ ฟัซซี PI - I ร่วมกับ วงจรเข็มทิศดิจิตอล สามารถใช้ในการควบคุมได้จริง แม้ในส่วนกราฟผลตอบสนองที่ได้จะดูแย่กว่าระบบควบคุมแบบ ฟัซซี PI ก็ตาม แต่เมื่อดูผลความแม่นยำของทิศทางการเคลื่อนที่เป็นเส้นตรงแล้วพบว่า วงจรเข็มทิศดิจิตอล มีส่วนช่วยในการปรับทิศทางอย่างมาก ทำให้ผลที่ได้มีค่าความผิดพลาดน้อยมากเมื่อเทียบกับระบบควบคุมในการทดลองที่ผ่านมา และค่ามุมของหุ่นยนต์ที่ได้นั้นจะเห็นได้ว่า เกิดการปรับมุมให้กลับสู่ค่าแรกเริ่มอยู่เสมอๆ และในตอนสุดท้ายของการเคลื่อนที่ค่ามุมได้กลับมาสู่ค่าเดิมเหมือนตอนแรกเริ่ม ซึ่งแสดงให้เห็นว่าหุ่นยนต์สามารถเดินเป็นเส้นตรงได้

บทที่ 5
บทวิจารณ์และสรุป

5.1 สรุปผลการทดลอง

จากการทดลองออกแบบระบบควบคุมแบบฟัซซีด้วยคอมพิวเตอร์ส่วนบุคคล พบว่าสามารถควบคุมความเร็วของระบบสองมวลที่ความเร็ว 1,500 รอบต่อนาที ได้ดีโดยสามารถเข้าสู้ค่าอ้างอิงได้เร็ว ไม่มีค่าพุ่งเกิน และไม่มีค่าการแกว่ง และในส่วนการทดลองออกแบบระบบควบคุมแบบฟัซซีด้วยไมโครคอนโทรลเลอร์ พบว่าสามารถควบคุมความเร็ว และทิศทางการเคลื่อนที่ของหุ่นยนต์ได้ดีกว่าระบบควบคุมแบบ PID แม้จะเกิดความผิดพลาดขึ้นบ้างเล็กน้อย จึงสรุปได้ว่าระบบควบคุมแบบฟัซซีสามารถประยุกต์ใช้กับระบบที่ไม่ทราบแบบจำลองทางคณิตศาสตร์ได้ทั้งสองระบบ

5.2 ปัญหาที่พบและแนวทางแก้ไข

ในการทดลองออกแบบระบบควบคุมแบบฟัซซีเพื่อควบคุมความเร็วของระบบสองมวลด้วยคอมพิวเตอร์ส่วนบุคคล พบปัญหาเกี่ยวกับการปรับแต่งค่าฟังก์ชันการเป็นสมาชิก และกฎการควบคุม ซึ่งมีความละเอียดอ่อนและซับซ้อนมาก จำเป็นต้องมีความรู้ และประสบการณ์ของผู้เชี่ยวชาญในระบบนั้นๆ ทำให้เสียเวลาในการปรับแต่งค่าฟังก์ชันการเป็นสมาชิก และกฎการควบคุมเป็นอย่างมาก และจากการทดลองออกแบบระบบควบคุมแบบฟัซซีเพื่อควบคุมความเร็วและทิศทางการเคลื่อนที่ของหุ่นยนต์ด้วยไมโครคอนโทรลเลอร์ พบปัญหาเกี่ยวกับแรงกระชากเมื่อปล่อยหุ่นยนต์ออกจากจุดเริ่มต้น ทำให้เกิดการเอียงก่อนออกตัว จึงจำเป็นต้องใช้วงจรเข็มทิศดิจิตอลเข้ามาช่วยในการควบคุมทิศทางของหุ่นยนต์

5.3 ข้อเสนอแนะและแนวทางในการค้นคว้าพัฒนา

ข้อเสนอแนะสำหรับการพัฒนาโครงงานในอนาคต ควรประยุกต์ใช้งานระบบควมคุมแบบฟัซซีกับระบบอื่นๆ เช่น การควบคุมเส้นทางที่ไม่ใช้เส้นทางตรง หรือเปรียบเทียบกับการควบคุมแบบโครงข่ายประสาทเทียม

เอกสารอ้างอิง

[1] Thomas Bräunl. **EMBEDDED ROBOTICS: Mobile Robot Design and Applications with Embedded Systems.** Second Edition. Springer Berlin Heidelberg. New York 2003.

[2] **ปราโมทย์ แพทย์พันธ์. "เครื่องให้อาหารปลาอัตโนมัติ."** ปริญญานิพนธ์วิศวกรรมศาตรบันฑิต สาขาวิชาระบบควบคุม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2546.

[3] นคร ภักดีชาติ และคณะ. **คู่มือการทดลองเบื้องต้น dsPIC Microcontroller.** กรุงเทพ. บริษัท อินโนเวดีฟ เอ็กเพอริเมนต์ จำกัด. 2550.

[4] วิชาญ ฉัตรรัตนวุฒิ. **"ตัวควบคุมแมมดานิฟัซซีไอ-พีดีสำหรับควบคุมตำแหน่งเชิงมุมของแขนกลสองข้อต่อ."** สาขาวิชาระบบควบคุม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2550.

[5] ณภัทรพงศ์ สุขเสรีวัฒนากุล **"การควบคุมของระบบสองมวลโดยตัวควบคุม แมมดานิฟัซซีไอ-พี."** สาขาวิชาระบบควบคุม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2549.

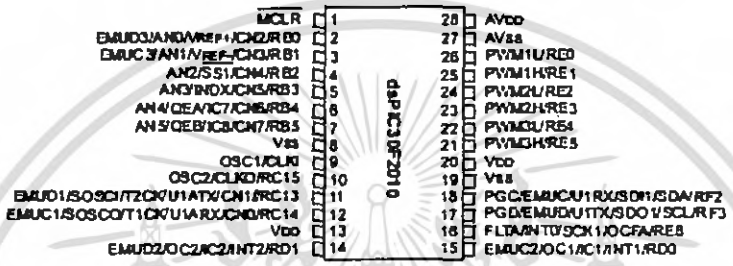
ภาคผนวก

ภาคผนวก ก

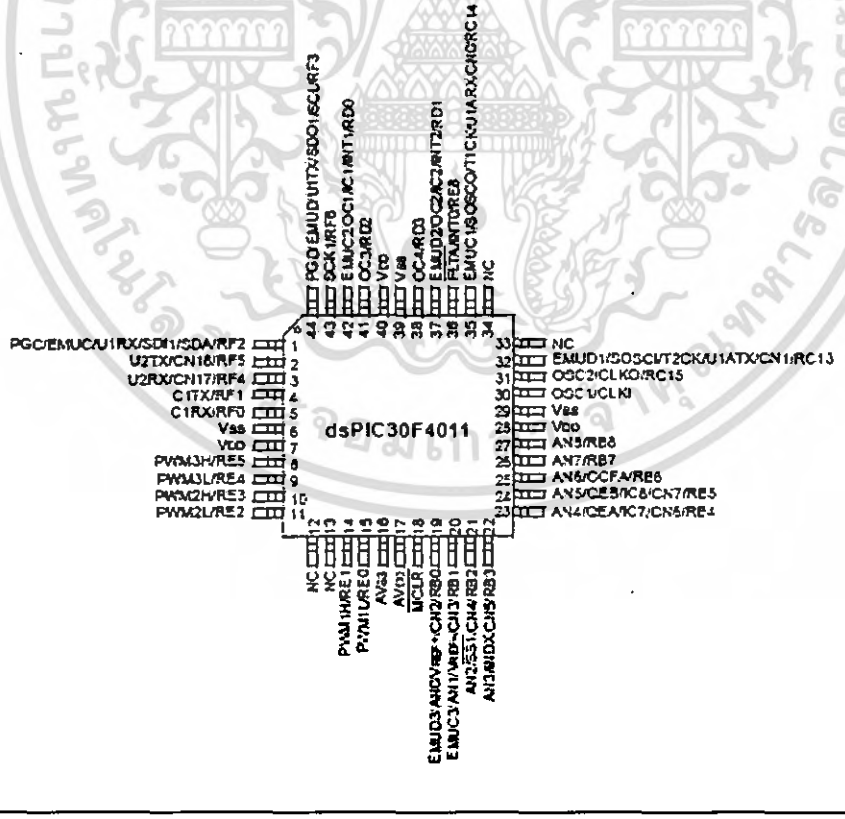
ไมโครคอนโทรลเลอร์

Pin Diagrams

28-Pin SDIP and SOIC

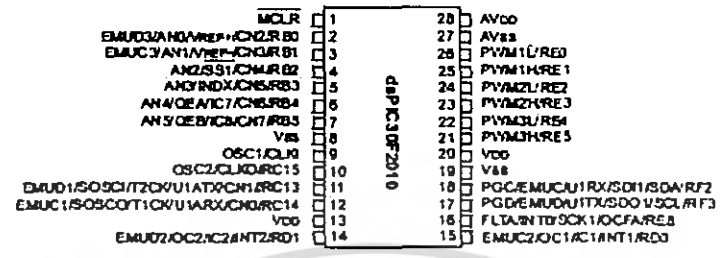


44-Pin TQFP

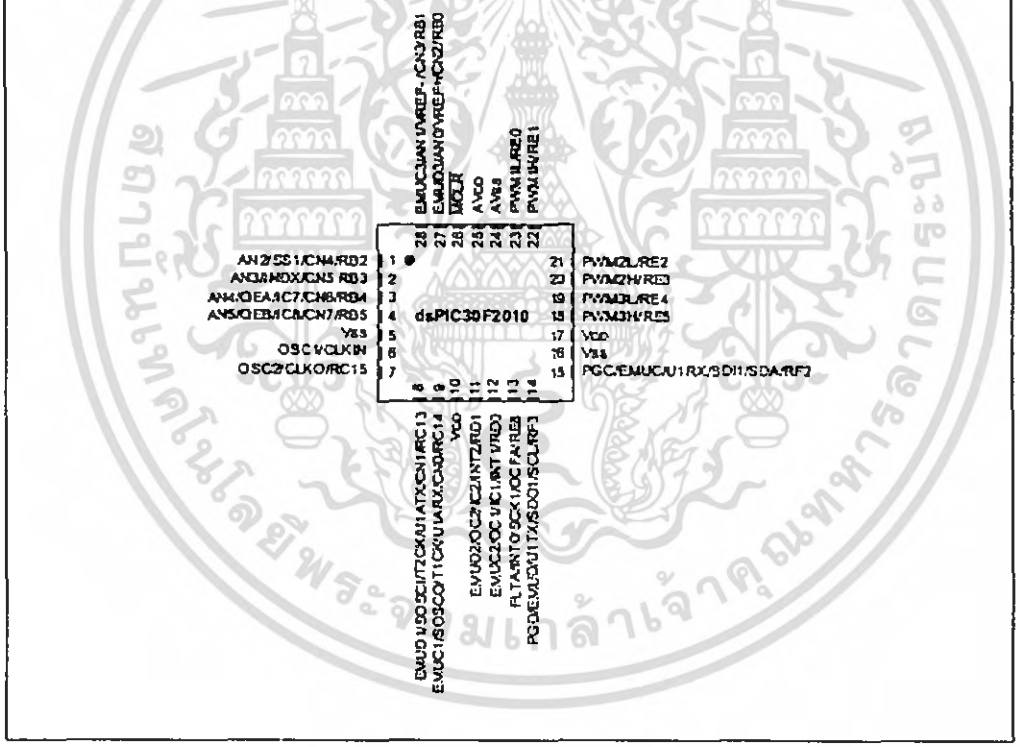


Pin Diagrams

28-Pin SDIP and SOIC



28-Pin QFN



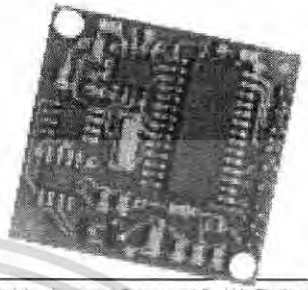
ภาคผนวก ข

โมดูลเข็มทิศดิจิตอล

CMPS03

Digital Compass Module

โมดูลเข็มทิศดิจิตอล

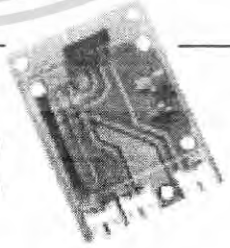


Distributed by Innovative Experiment Co.,Ltd., Thailand

คุณสมบัติ

- ใช้ไฟเลี้ยง +5V ต้องการกระแสไฟฟ้า 20mA
- ใช้ตัวตรวจจับสนามแม่เหล็กเบอร์ KMZ51 ของ Philips จำนวน 2 ตัว เพื่อให้สามารถตรวจจับสนามแม่เหล็กโลกได้อย่างสมบูรณ์และมีความละเอียดมากเพียงพอ
- ความละเอียดของมุม 0.1 องศา
- ค่าความผิดพลาด 3-4 องศา โดยประมาณ หลังจากการปรับเทียบ
- เอาต์พุตแบบสัญญาณพัลส์ ความกว้าง 1 ถึง 37 มิลลิวินาที โดยมีอัตราเพิ่มครั้งละ 0.1 มิลลิวินาที
- เอาต์พุตข้อมูลดิจิตอลผ่านการติดต่อระบบบัส I²C รองรับสัญญาณนาฬิกาความถี่สูงถึง 1MHz โดยให้ข้อมูล 2 รูปแบบคือ 0-255 และ 0-3599
- ขนาดเล็กเพียง 32 x 35 มิลลิเมตร
- สื่อสารกับไมโครคอนโทรลเลอร์แอดนิยมใช้ทุกตระกูล อาทิ เบสิกแสตมป์ 2SX/2P, PIC, MCS-51, PSoC, 68HC11 ทั้งผ่านระบบบัส I²C และด้วยการรับสัญญาณพัลส์

อุปกรณ์เสริม



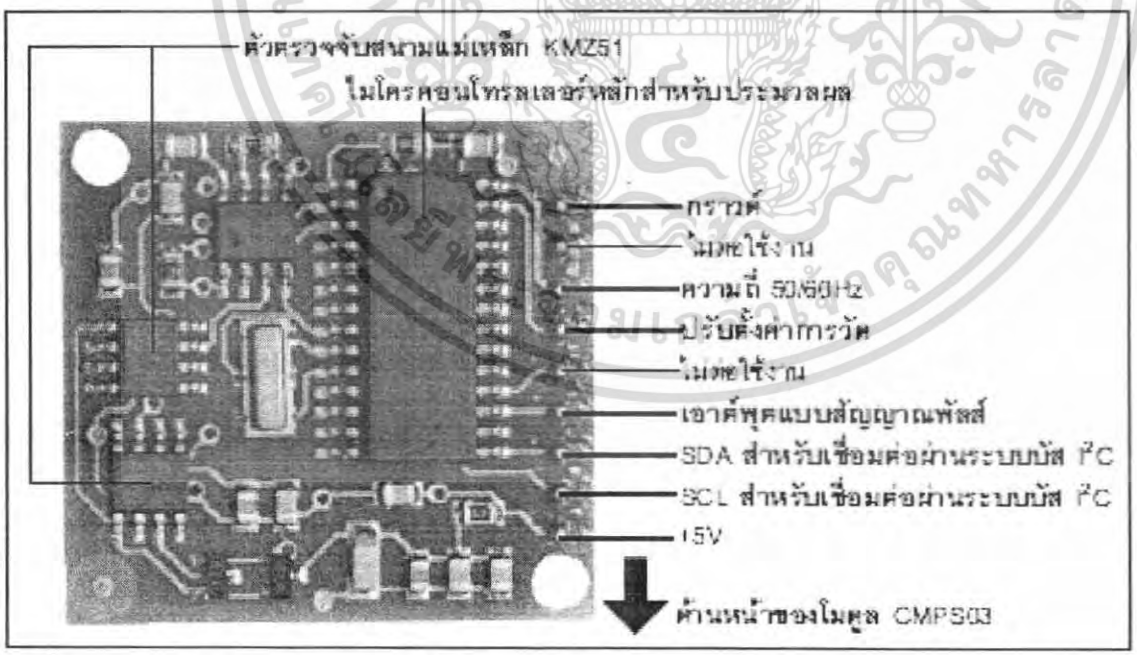
- บอร์ด ADX-CMPS03 ซึ่งเป็นบอร์ดขยายสำหรับช่วยอำนวยความสะดวกในการเชื่อมต่อกับบอร์ดไมโครคอนโทรลเลอร์และแผงต่อวงจรเพื่อเบรดบอร์ด
- สาย PCB3A สำหรับเชื่อมต่อกับบอร์ดควบคุมยนต์ของ Hex

โมดูลเข็มทิศดิจิตอล CMPS03 เป็นผลงานของ Devantech (www.radio-electronics.co.uk) ออกแบบมาเพื่อช่วยในการกำหนดทิศทางการเคลื่อนที่ของหุ่นยนต์อัตโนมัติ และ นำมาใช้ในการสร้างเครื่องมือวัดและตรวจสอบทิศระบบอิเล็กทรอนิกส์ โดยหัวใจสำคัญของโมดูล CMPS03 คือตัวตรวจจับสนามแม่เหล็กเบอร์ KMZ51 ของ Philips จำนวน 2 ตัว เพื่อให้มีความไวเพียงพอในการตรวจจับสนามแม่เหล็กโลก (Earth magnetic field) และไมโครคอนโทรลเลอร์เพื่อรับสัญญาณจากตัวตรวจจับมาประมวลผลเป็นข้อมูลดิจิตอลและสัญญาณพัลส์สำหรับแจ้งผลการวัดทิศทาง

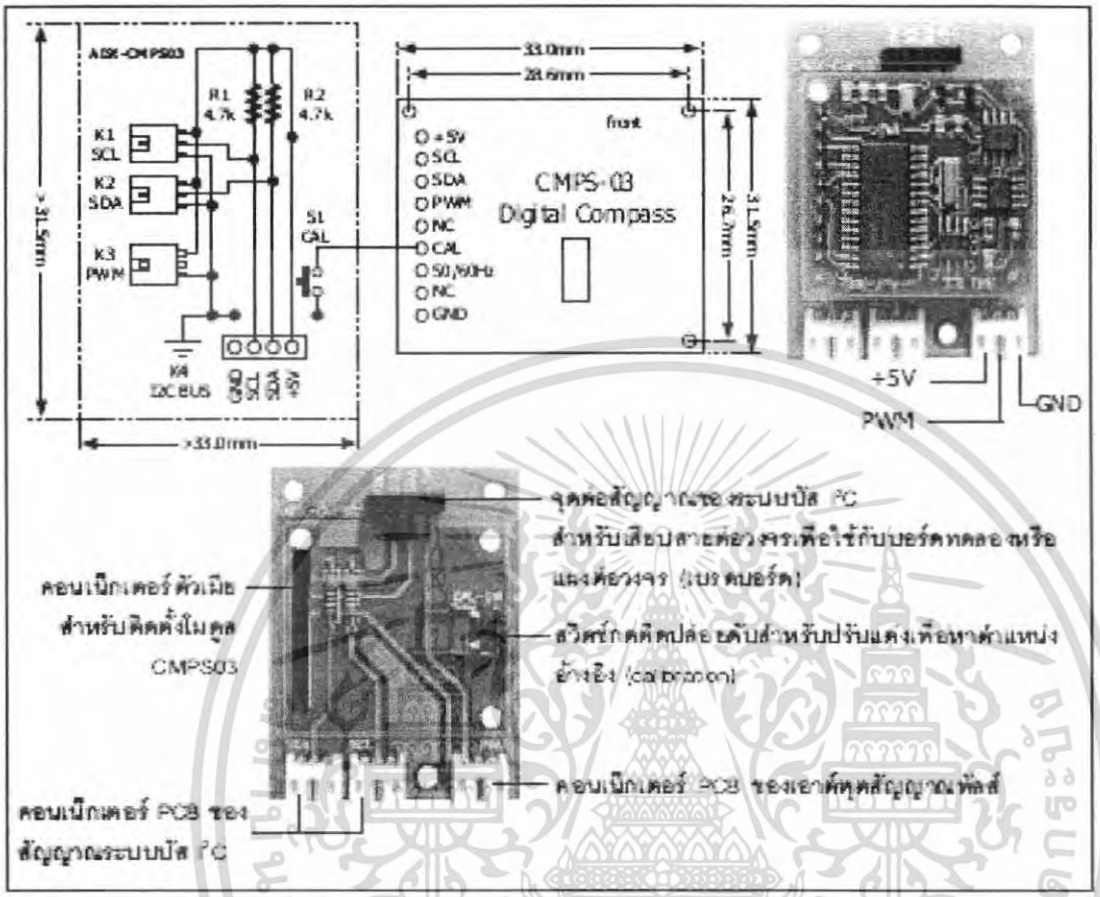
1. ตำแหน่งขาและการต่อใช้งาน

ในรูปที่ 1 แสดงรูปร่างหน้าตาและการจัดขาของ CMPS03 โมดูลเข็มทิศดิจิตอล จะเห็นว่าเป็นแผงวงจรที่มีคอนเน็กเตอร์ต่อออกมาเพื่อให้เชื่อมต่อไปใช้งาน อย่างไรก็ตามเพื่ออำนวยความสะดวกแก่ผู้ใช้งานกับบอร์ดควบคุมหุ่นยนต์ของบริษัท อินโนวลีฟ เอ็กเซอริเมนต์ จำกัด (i-nex : เป็นตัวแทนจำหน่ายสินค้าของ Devantech ในประเทศไทยอย่างเป็นทางการ) จึงได้พัฒนาบอร์ดอะแดปเตอร์รุ่น ADX-CMPS03 เพื่อให้นำโมดูล CMPS03 มาติดตั้ง (โดยบอร์ด ADX-CMPS03 ต้องซื้อแยก)

บนบอร์ด ADX-CMPS03 ได้จัดเตรียมคอนเน็กเตอร์ PCB 5 ขาตัวผู้สำหรับเชื่อมต่อกับบอร์ดควบคุมหุ่นยนต์ และคอนเน็กเตอร์ IDC ตัวเมียแถวเดี่ยว 4 ขาสำหรับเสียบสายต่อวงจรเบอร์ AWG#22 เพื่อต่อกับแผงต่อวงจรหรือบอร์ด นอกจากนั้นยังมีสวิตช์กดสำหรับปรับตั้งค่า(calibration) เพื่อกำหนดตำแหน่งทิศอ้างอิง โดยวงจรของบอร์ด ADX-CMPS03 แสดงในรูปที่ 2



รูปที่ 1 แสดงรูปร่างและตำแหน่งขาสำหรับการต่อใช้งาน



รูปที่ 2 แสดงวงจรของบอร์ด ADX-CMPS03 และการเชื่อมต่อกับโมดูล CMPS03

2. การปรับแต่งค่าทิศทางอ้างอิงแก่โมดูล CMPS03

เพื่อให้การวัดทิศทางของโมดูล CMPS03 มีความแม่นยำมากที่สุด จึงมีอินพุตสำหรับปรับแต่งค่าทิศทางอ้างอิง ทั้งนี้เพื่อประโยชน์ในการกำหนดทิศทางอ้างอิงเฉพาะสำหรับผู้ใช้งาน โดยต้องป้อนสัญญาณลอจิก "0" เข้าที่ขาอินพุตสำหรับปรับแต่งโมดูล CMPS03 ซึ่งก็คือขา CAL หากใช้บอร์ด ADX-CMPS03 กับโมดูล CMPS03 จะมีสวิตช์กดติดปล่อยดับติดตั้งไว้ให้แล้ว การปรับแต่งมีขั้นตอนดังนี้

(1) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศเหนือ กดสวิตช์ 1 ครั้ง

(2) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันออก กดสวิตช์

(3) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศใต้ กดสวิตช์ 1 ครั้ง

(4) วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันตก กดสวิตช์

เป็นอันสิ้นสุดการปรับตั้งค่าทิศทางอ้างอิงของโมดูล CMPS03 โดยโมดูลจะเก็บค่าอ้างอิงนี้ไว้ในหน่วยความจำอีอีพรอมและไม่ต้องปรับตั้งค่าใหม่อีกเมื่อจ่ายไฟเลี้ยงครั้งใหม่

3. การอ่านค่าสัญญาณเอาต์พุตของโมดูล CMPS03

3.1 การอ่านค่าทิศทางจากเอาต์พุตสัญญาณพัลส์

การอ่านค่าสัญญาณในโหมดนี้ เป็นการนำค่าความกว้างพัลส์ที่ได้จากเอาต์พุตสัญญาณพัลส์ของโมดูล CMPS03 มาระบุค่าเป็นหน่วยองศา จาก 0 ถึง 359.9 องศา โดยมีช่วงของค่าความกว้างสัญญาณพัลส์จาก 1 มิลลิวินาทีไปจนถึง 36.99 มิลลิวินาที มีความละเอียด 0.1 มิลลิวินาทีต่อองศา ในสัญญาณพัลส์แต่ละไซเกิล มีช่วงลอจิก "0" กว้าง 65 มิลลิวินาที

ดังนั้นในการนำสัญญาณพัลส์มาประมวลผลเป็นค่ามุม จึงต้องใช้การนับความกว้างของสัญญาณพัลส์เป็นหลักในการคำนวณหาค่ามุมที่โมดูล CMPS03 วัดได้

3.1.1 ตัวอย่างโปรแกรมที่ใช้ควบคุมสำหรับบอร์ดแสดงผล 2SX และ i-Stamp

การใช้งานร่วมกับบอร์ดแสดงผล 2SX และ i-Stamp นั้น จะใช้คำสั่ง PULSIN ในการนับสัญญาณพัลส์ โดยจะเพิ่มค่าการนับขึ้นทุกๆ 0.8 ไมโครวินาทีดังนั้นที่ความกว้างของพัลส์ที่ 1 มิลลิวินาทีสำหรับค่ามุม 0 องศา บอร์ดแสดงผล 2SX และ i-Stamp จะนับค่าได้เท่ากับ 1,250 จึงสามารถใช้ค่านี้เป็นจุดอ้างอิงที่ 0 องศา *เมื่อต้องการหาค่ามุมที่แท้จริง ให้นำค่ามุมที่นับได้ลบด้วย 1,250 แล้วหารด้วย 125 ก็จะได้ค่ามุมในหน่วยองศาที่ต้องการ* รายละเอียดของโปรแกรมแสดงในโปรแกรมที่ 1

ที่ความกว้างพัลส์สูงสุดคือ 36.99 มิลลิวินาที ค่าที่นับได้จากคำสั่ง PULSIN เท่ากับ 46,237 เมื่อลบด้วย 1,250 แล้วหารด้วย 125 เพื่อแปลงเป็นองศา ค่าสูงสุดที่แสดงเป็นผลลัพธ์ได้คือ 359 เป็นค่าหน่วยของศาสูงสุดนั่นเอง

หมายเหตุ ในบางกรณี ค่าที่อ่านได้สูงสุดจากคำสั่ง PULSIN อาจไม่ถึง 46,237 ผู้ใช้งานสามารถปรับเปลี่ยนการคำนวณใหม่ให้ได้มุมเป็น 359.9 องศาได้

```
'{$STAMP BS2sx}
'{$PBASIC 2.5}
bearing  VAR  WORD
main:
     PULSIN 4, 1, bearing                ' Get reading
     bearing = (bearing-1250)/125        ' BS2sx  - Calculate Bearing
                                         ' in degrees
     DEBUG "Compass Bearing  ", DEC3 bearing ,CR
                                         ' Display Compass Bearing
     GOTO main
```

โปรแกรมที่ 1 แสดงการอ่านค่าสัญญาณพัลส์จากโมดูล CMPS03

ตำแหน่งรีจิสเตอร์	รายละเอียด
0	ตัวเลขแสดงรุ่นของบอร์ด CMPS03
1	ส่งค่าตำแหน่งแบบหยาบ (0-255)
2,3	ส่งค่าตำแหน่งแบบละเอียดด้วยตัวเลข 16 บิต (0-3599) สามารถแปลงค่าที่ได้แสดงของค่า 0-359.9 องศาได้โดยตรง
4,5	สำหรับตรวจสอบค่ากราฟิน โดยจะแสดงค่าความต่างของ Sensor1 เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
6,7	สำหรับตรวจสอบค่ากราฟิน โดยจะแสดงค่าความต่างของ Sensor2 เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
8,9	แสดงค่าตัวเลขการปรับแต่งภายใน (calibration value1) เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
10,11	แสดงค่าตัวเลขการปรับแต่งภายใน (calibration value2) เป็นตัวเลข 16 บิตแบบคิดเครื่องหมาย
12,13	ไม่ใช้งาน ค่าที่ได้เป็น 0
14	ไม่ใช้งาน ไม่ให้ค่าหรือค่าใด
15	คำสั่งสำหรับการปรับแต่งค่า โดยนี้จะต้องการปรับแต่งค่า ต้องเขียนข้อมูล 255 เข้าที่รีจิสเตอร์ตำแหน่งนี้

ตารางที่ 1 แสดงตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03

3.2 การอ่านค่าทิศทางเป็นข้อมูลดิจิตอลผ่านระบบบัส I²C

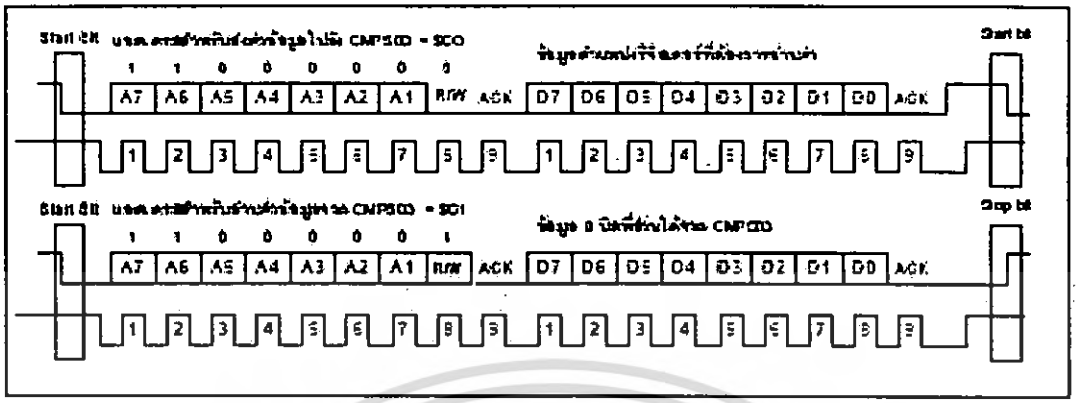
การอ่านค่าจากโมดูล CMPS03 ให้ได้ค่าที่มีความแม่นยำสูงควรเลือกอ่านข้อมูลดิจิตอลผ่านระบบบัส I²C โดยโมดูล CMPS03 สามารถส่งข้อมูลของตำแหน่งออกมาที่ความละเอียดสูงสุด 0.1 องศาโดยไม่จำเป็นต้องมีการคำนวณหรือแปลงค่าใด ๆ อีก

3.2.1 รูปแบบการสื่อสารข้อมูลบัส I²C

บัส I²C เชื่อมต่อกับไมโครคอนโทรลเลอร์โดยใช้สายสัญญาณ 2 เส้น ได้แก่ขา SDA (ข้อมูลและส่งข้อมูล) และ SCL (ขาสัญญาณนาฬิกา) โดยขาสัญญาณทั้งสองจะต้องต่อตัวต้านทานพูลอัปต่อไว้เพื่อกำหนดสถานะลอจิก "1" ให้กับระบบบัส

3.2.2 ลำดับขั้นการติดต่อ

ค่แอดเดรสของโมดูล CMPS03 คือ 3C0 สำหรับการส่งข้อมูล และ 3C1 สำหรับการอ่านค่าข้อมูล โดยขั้นตอนการติดต่อกับโมดูล CMPS03 เพื่ออ่านข้อมูลมีดังนี้



รูปที่ 3 แสดงไทมิ่งไดอะแกรมของการติดต่อสื่อสารกับโมดูล CMPS03 ผ่านระบบบัส I²C

1. ส่งบิตเริ่มต้นหรือ Start bit เพื่อแจ้งให้ระบบบัส I²C เตรียมพร้อมรับข้อมูล

2. ส่งค่าแอดเดรส SC0 เพื่อระบุว่าต้องการติดต่อเพื่อเขียนข้อมูลไปยังกับโมดูล CMPS03

3. ส่งค่าตำแหน่งรีจิสเตอร์ภายในโมดูล CMPS03 ที่ต้องการอ่านค่า ซึ่งมีรายละเอียดแสดงในตารางที่ 1

4. ส่งค่าแอดเดรส SC1 เพื่อระบุว่าต้องการอ่านค่าข้อมูลจากโมดูล CMPS03

5. อ่านค่าข้อมูลจากโมดูล CMPS03 มาเก็บไว้ในหน่วยความจำ

6. ส่งบิตหยุดหรือ Stop เพื่อหยุดการสื่อสารข้อมูลและกำหนดให้บัสอยู่ในสภาวะบัสว่าง

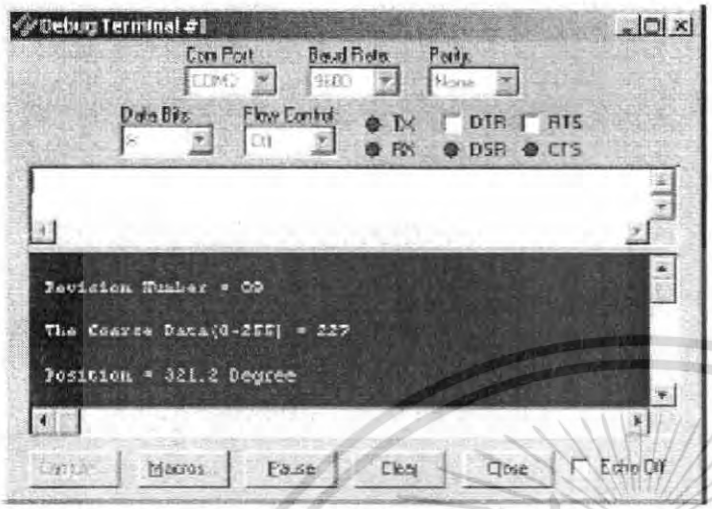
จากลำดับขั้นการติดต่อสื่อสารข้างต้น สามารถนำมาเขียนเป็นโปรแกรมตัวอย่างเพื่ออ่านข้อมูลจากโมดูล CMPS03 โดยใช้ฟังก์ชันแสดงในที่ 2SX หรือ i-Stamp ให้ดังแสดงในโปรแกรมที่ 2

```
'{$STAMP BS2sx}
'{$PBASIC 2.5}
SDA             CON     6           ' I2C serial data line
SCL             CON     7           ' I2C serial clock line
WrCMPS03        CON     $C0         ' write to compass
RdCMPS03        CON     $C1         ' read from compass
Ack             CON     0           ' acknowledge bit
Nak             CON     1           ' no ack bit
i2cSDA          VAR     NIB         ' I2C serial data pin
i2cData         VAR     WORD        ' data to/from device
REGISTER        VAR     BYTE        ' register address
i2cWork         VAR     BYTE        ' work byte for TX routine
i2cAck          VAR     BIT         ' Ack bit from device
temp            VAR     WORD        ' for rj printing
digits          VAR     NIB
width           VAR     NIB

Init:
   PAUSE 250
   i2cSDA = SDA                                 ' define SDA pin
   REGISTER = 0                                 ' compass revision number
   GOSUB Read_Byte
   DEBUG 2,1,1, "Revision Number = ",DEC2 i2cData
Main:
   REGISTER = 1                                 ' Show Data 0-255 for 0-360 degree
   GOSUB Read_Byte                              ' Read Byte From I2C
   DEBUG 2,1,3, "The Coarse Data(0-255) = ",DEC i2cData
   REGISTER = 2                                 ' get Data in degrees, 0.0 - 359.9 Degree
   GOSUB Read_Word
   DEBUG 2,1,5, "Position = ",DEC i2cData/10,".",DEC1 i2cData," Degree"
   PAUSE 250
   GOTO Main

' Compass Access Subroutines
' Writes low byte of i2cData to REGISTER
Write_Byte:
   GOSUB I2C_Start
   i2cWork = WrCMPS03
   GOSUB I2C_TX_Byte                            ' send device address
   i2cWork = REGISTER
   GOSUB I2C_TX_Byte                            ' send register number
   i2cWork = i2cData.LOWBYTE
   GOSUB I2C_TX_Byte                            ' send the data
   GOSUB I2C_Stop
   RETURN

' Writes i2cData to REGISTER
Write_Word:
   GOSUB I2C_Start
   i2cWork = WrCMPS03
   GOSUB I2C_TX_Byte                            ' send device address
   i2cWork = REGISTER
   GOSUB I2C_TX_Byte                            ' send register number
   i2cWork = i2cData.HIGHBYTE
   GOSUB I2C_TX_Byte                            ' send the data - high byte
   i2cWork = i2cData.LOWBYTE
   GOSUB I2C_TX_Byte                            ' send the data - low byte
   GOSUB I2C_Stop
   RETURN
```

โปรแกรมที่ 2 โปรแกรมอ่านค่าจากโมดูล CMPS03 ผ่านระบบบัส I^2C ของ i-Stamp (มีต่อ)

```
' Read i2cData (8 bits) from REGISTER
Read_Byte: GOSUB I2C_Start
           i2cWork = WrCMPS03
           GOSUB I2C_TX_Byte                    ' send compass address
           i2cWork = REGISTER
           GOSUB I2C_TX_Byte                    ' send register number
           GOSUB I2C_Start                      ' repeat start (sets register)
           i2cWork = RdCMPS03
           GOSUB I2C_TX_Byte                    ' send read command
           GOSUB I2C_RX_Byte_Nak
           GOSUB I2C_Stop
           i2cData = i2cWork                    ' return the data
           RETURN

' Read i2cData (16 bits) from REGISTER
Read_Word: GOSUB I2C_Start
           i2cWork = WrCMPS03
           GOSUB I2C_TX_Byte                    ' send compass address
           i2cWork = REGISTER
           GOSUB I2C_TX_Byte                    ' send register number
           GOSUB I2C_Start                      ' repeat start (sets register)
           i2cWork = RdCMPS03
           GOSUB I2C_TX_Byte                    ' send read command
           GOSUB I2C_RX_Byte
           i2cData.HIGHBYTE = i2cWork           ' read high byte of data
           GOSUB I2C_RX_Byte_Nak
           GOSUB I2C_Stop
           i2cData.LOWBYTE = i2cWork            ' read low byte of data
           RETURN
' Low Level I2C Subroutines
'Start
I2C_Start: INPUT i2cSDA                         ' I2C start bit sequence
           INPUT SCL
           LOW i2cSDA                           ' SDA --> low while SCL high
Clock_Hold:
   IF (INS.LOWBIT(SCL) = 0) THEN Clock_Hold     ' device ready?
   RETURN

'Transmit
I2C_TX_Byte:
   SHIFTOUT i2cSDA,SCL,MSBFIRST,[i2cWork\8]     ' send byte to device
   SHIFTIN i2cSDA,SCL,MSBPRE,[i2cAck\1]         ' get acknowledge bit
   RETURN

'Receive
I2C_RX_Byte_Nak: i2cAck = Nak                   ' no Ack = high
                 GOTO I2C_RX
I2C_RX_Byte:     i2cAck = Ack                   ' Ack = low
I2C_RX:
   SHIFTIN i2cSDA,SCL,MSBPRE,[i2cWork\8]        ' get byte from device
   SHIFTOUT i2cSDA,SCL,LSBFIRST,[i2cAck\1]      ' send ack or nak
   RETURN

'Stop
I2C_Stop:  LOW i2cSDA                           ' I2C stop bit sequence
           INPUT SCL
           INPUT i2cSDA                         ' SDA --> high while SCL high
           RETURN
```

โปรแกรมที่ 2 โปรแกรมอ่านค่าจากโมดูล CMPS03 ผ่านระบบบัส I²C ของ i-Stamp (มีต่อ)



รูปที่ 4 แสดงหน้าจอของ Debug terminal แสดงข้อมูลที่อ่านได้จากโมดูล CMPS03

3.2.3 การทำงานของโปรแกรมที่ 2 ติดต่อกับโมดูล CMPS03 ผ่านระบบบัส I^2C ด้วย i-Stamp

เนื่องจากชุดสถานสอนปี 2SX และ i-Stamp ไม่มีคำสั่งติดต่อกับระบบบัส I^2C ดังนั้น โปรแกรมติดต่อกับโมดูล CMPS03 ผ่านระบบบัส I^2C จึงค่อนข้างยาว เนื่องจากต้องสร้างโปรแกรมย่อยสำหรับการสื่อสารข้อมูลกับระบบบัส I^2C หลังจากนี้ผู้ใช้งานสามารถนำโปรแกรมย่อยนี้ไปประยุกต์ใช้งานกับอุปกรณ์ตัวอื่นๆ ที่ได้ใช้การติดต่อระบบบัส I^2C ได้ทันที สำหรับขั้นตอนการทำงานหลักๆ ของโปรแกรมที่ 2 มีดังนี้

1. กำหนดรีจิสเตอร์เป็น 0 เพื่อติดต่ออ่านค่าเวอร์ชันของโมดูล CMPS03 จากนั้นอ่านค่ามาแสดงที่หน้าต่าง Debug terminal

2. ให้โปรแกรมวนลูปที่โปรแกรมหลัก จากนั้นส่งค่ารีจิสเตอร์เท่ากับ 1 เพื่ออ่านค่าข้อมูลแบบหยาบออกมา แล้วนำมาแสดงผลที่หน้าต่าง Debug terminal

3. ส่งค่ารีจิสเตอร์เท่ากับ 2 เพื่ออ่านค่าข้อมูลแบบละเอียด จากนั้นสั่งอ่านค่าข้อมูลจากบัส I^2C แบบเรียล (อ่านข้อมูลออกมา 16 บิต)

4. นำค่าข้อมูลที่ได้หารด้วย 10 ก่อนเพื่อแปลงค่าที่ได้เป็นองศาแสดงที่หน้าต่าง Debug terminal จากนั้นนำค่าหลักสุดท้ายมาแสดง ซึ่งตำแหน่งสุดท้ายเป็นตำแหน่งของจุดทศนิยม

3.3.4 การปรับแต่งค่าของโมดูล CMPS03 ผ่านทางระบบบัส I²C

การปรับแต่งค่าทำได้โดยการส่งค่า 0xFF ไปยังรีจิสเตอร์ 15 ของโมดูล CMPS03 โดยจะต้องส่งค่า 4 ครั้งและระบุทิศทางหลักๆ 4 ทิศทางเช่นเดียวกับการกำหนดค่าด้วยสวิตช์โดยตรง มีขั้นตอนดังนี้

1. วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศเหนือ จากนั้นเขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

2. วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันออก เขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

3. วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศใต้ กดสวิตช์ เขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

4. วางโมดูล CMPS03 ขนานกับพื้น หันด้านหน้าของโมดูลไปทางทิศตะวันตก กดสวิตช์ เขียนค่า 255 (0xFF) ไปยังรีจิสเตอร์ 15

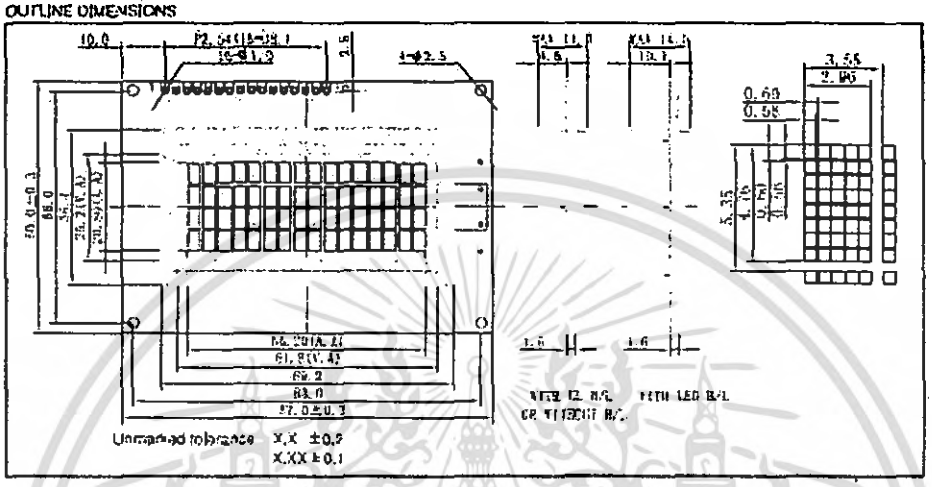
หลังจากปรับแต่งค่าแล้วค่าที่ปรับแต่งจะถูกบันทึกไว้ที่หน่วยความจำอีอีพรอม ดังนั้นแม้ไม่จ่ายไฟให้กับบอร์ด ข้อมูลที่ปรับแต่งแล้วจะยังคงอยู่ต่อไป

ภาคผนวก ค

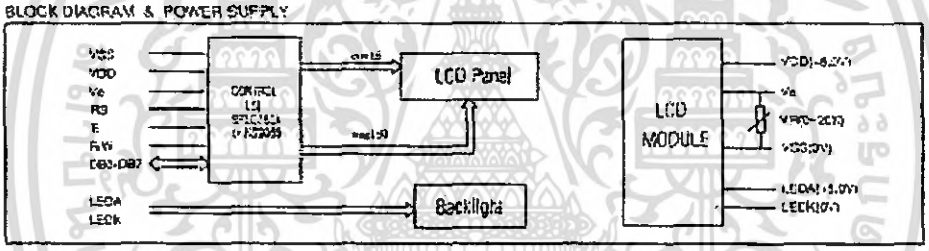
โมดูลจอแสดงผลแอลซีดี

AV-DISPLAY **ABC016004A Series** 16x4 CHARACTERS 1/16 DUTY, 1/5 BIAS

OUTLINE DIMENSIONS



BLOCK DIAGRAM & POWER SUPPLY



MECHANICAL SPECIFICATIONS & FEATURE

Item	Nominal Dimensions/Value	Item	Value
Module Size (W*H*T)	87.0x60.0x11.0/14.5	LCD Type	STN, FSTN
View Area (W*H)	61.8x25.2	LCD Color	Black/Yellow-Green, Gray, Other Gray
Dot Size (W*H)	0.55x0.75	View Angle	6 O'clock, 12 O'clock
Character Size (W*H)	2.95x4.75	Display Type	Positive type, Negative type
Character Font	5x8	Rear Color	Transflective, Reflective, Transmissive
Dot Pitch (W*H)	0.60x0.80	Operating Temperature	0°C ~ 50°C, -20°C ~ +70°C
Dot Size (W*H)	0.55x0.75	Backlight	LED(White, Blue-Green, Blue, Amber)

ELECTRICAL CHARACTERISTICS

Item	Symbol	Test Condition	Min.	Typ.	Max.	Unit
Operating Voltage	Vdd	Ta=25°C	—	5.0	—	V
Operating Voltage for LCD	Vlcd	Ta=25°C	—	4.5	—	V
Supply Current	IDD	Ta=25°C, Vdd=5.0V	—	2.0	5.0	mA
Supply Current for Backlight(LED)	IL	Ta=25°C, VL=4.0V	—	180	—	mA

INTERFACE PIN CONNECTIONS

Pin No	Symbol	Level	Description
1	VSS	—	Ground for Logic
2	VDD	—	Power supply for Logic
3	Vo	—	Power supply for LCD drive
4	RS	H/L	Register Selection (H:Data register, L:Instruction register)
5	R/W	H/L	Read/write selection (H:Read, L:Write)
6	E	H/H→L	Enable signal for LCM
7-14	DB0-DB7	H/L	Data bus lines
15	LEDA	—	Power supply for Backlight(+)
16	LEDK	—	Power supply for Backlight(-)

ภาคผนวก ง
โปรแกรมภาษา C ที่ใช้ควบคุม

Program of master controller

```
//program:sam_80ms_master.c
//case of dip switch 2&3 on->"0" off->"1"
//off off : setting speed setpoint
//off on :      changing number of sampling data
// on off : send duty cycle of PWM to slave controller
// on  on : read angle from cmp sensor module
#include<p30f4011.h>
unsigned char uart_buf[25];                 // "sprintf" Buffer
#include<uart.h>                    // Used UART Library Function
#include<stdio.h>                                       // Used "sprintf" Function
#include<timer.h>
#include <adc10.h>
#include"C:\dsPIC\dspic_o00olib\led.h"
#include"C:\dsPIC\dspic_o00olib\lcd_mcdspic.h"
#include"C:\dsPIC\dspic_o00olib\spi_master_project.h"
#include"C:\dsPIC\dspic_o00olib\adcdspic.h"
#include "C:\dsPIC\dspic_o00olib\i2c_cmp.h"
#include"C:\dsPIC\dspic_o00olib\motor\fuzzy_PI_1.h"             //control loop 2 PI-1
_FOSC(CSW_FSCM_ON & XT_PLL16);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_ON & BORV_45 & PWRT_64 & MCLR_EN);
_FGS(CODE_PROT_OFF);
#define match_value   14746             //PS 1:8 interrupt every (0.5*8)=4ms  14745.6
#define num_dat                         250             //number of sampling data recoded for plot the time response graph
#define init_num_sam    150             //initial sampling number
//I/p of master
#define sw1  PORTBbits.RB4
#define sw2  PORTBbits.RB5
#define dip3  PORTBbits.RB6
#define dip2  PORTBbits.RB7
#define dip1  PORTBbits.RB8
#define bt1     PORTEbits.RE4
#define on      0
#define dly_key 100
unsigned char count_4ms=0;
unsigned int pos=0;
signed char data_show[2][num_dat];
unsigned char flag_complete=0,flag_sampling=0,flag_st_duty=0; //sampling data has completed
unsigned int num_sam=init_num_sam,last_num_sam=0;
unsigned int duty_pwm=0,last_duty_pwm=0;
unsigned int ang_start=0;
signed char present_setpoint=0,last_setpoint=0;                 //considered data
void init();
void init_dip();
void init_timer();
void init_adc();
void init_uart();
void show_uart();
void delay(unsigned long dly);
void _ISR _T1Interrupt() {//interrupt every 4ms
 if(flag_sampling==1) {
  WriteTimer1(0);
  if(flag_st_duty==1) {
   spi_send16_s(1,change_duty16,duty_pwm);
   spi_send16_s(2,change_duty16,duty_pwm);
  }
  if(count_4ms++>=20) { //in loop every  80 ms
   if(pos++<num_sam) {
                //Right
    spi_send8(1,cal_speed,0);
    data_show[0][pos]=spi_receive8_s(1,read_speed8);            //R

    //Left
    spi_send8(2,cal_speed,0);
    data_show[1][pos]=spi_receive8_s(2,read_speed8);            //L

FLC_PI_1_cmp(present_setpoint,data_show[0][pos],data_show[1][pos],ang_start,(angle)/10);
    led_show(r_blink);
    count_4ms=0;
   } else {
    ConfigIntTimer1(T1_INT_OFF & T1_INT_PRIOR_1);
    spi_send8(1,stop_motor,0);          //send data to stop motor R
    spi_send8(2,stop_motor,0);          //send data to stop motor L
    led_show(g_on);
    lcd_text(line3,"finish");
    delay(1000000);
    flag_complete=1;
   }
  }
 }
 IFS0bits.T1IF = 0;
}
int main() {
 init();
 while(1) {
  if(flag_complete) {
   show_uart();
  }
 }
}
void init() {
 signed int temp_adc=0,last_temp_adc=0;                 //adc value
 unsigned int ang_now=0,last_ang=0;
 unsigned char flag_first[4]={0,0,0,0};
```

```
lcd_init();                          //init lib
init_led();
init_spi_master();
 init_dip();                         //init function
init_timer();                        //start sampling
init_adc();
init_uart();
show_uart();
init_i2c();
spi_send8_s(1,0,0);       //R
spi_send8_s(2,0,0);       //L
 lcd_text(line1,"sampling 80 ms");   //init display
sprintf(uart_buf,"test run multi motor\r\n");          //init show terminal
putsUART1((unsigned int *)uart_buf);
while(BusyUART1());
sprintf(uart_buf,"fuzzy_PI_I_master.b\r\n");          //init show terminal
putsUART1((unsigned int *)uart_buf);
while(BusyUART1());
sprintf(uart_buf,"set speed\r\n");    //init show terminal
putsUART1((unsigned int *)uart_buf);
while(BusyUART1());
lcd_text(line4,"press button");
while(bt1==1);
delay(100);
while(bt1==0);
delay(100);
lcd_text(line4,"          ");

while(dip1==on) {
 if((dip2==1)&&(dip3==1)) {                          //set speed run          temp_adc=setpoint_value
  if(flag_first[0]==0) {
   flag_first[1]=0;
   flag_first[2]=0;
   flag_first[3]=0;
   flag_first[0]=1;
   led_show(r_on);
   lcd_text(line1,"change sp:     ");
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,present_setpoint,10);
   lcd_text(line4,"              ");
   temp_adc=0;
  }
  temp_adc= -50+((adc_read(3))/10);
  if(temp_adc>50) {
   temp_adc=50;
  }
  if(last_temp_adc!=temp_adc) {
   lcd_text(line1+10,"    ");
   sinttolcd(line1+10,temp_adc,10);
   last_temp_adc=temp_adc;
  }
  if(last_setpoint!=present_setpoint) { //change setpoint of left motor
   lcd_text(line2+2 ,"     ");
   sinttolcd(line2+2,present_setpoint,10);
   last_setpoint=present_setpoint;
  }
  if(bt1==0) {                                                          //change setpoint value
   delay(dly_key);
   if(bt1==0) {
    present_setpoint=temp_adc;
    spi_send8_s(1,change_setpoint8,present_setpoint);          //R
    spi_send8_s(2,change_setpoint8,present_setpoint); //L
    led_show(off);
    delay(1000000);
    led_show(r_on);
    delay(1000000);
    led_show(off);
    delay(1000000);
    led_show(r_on);
   }
  }
  sinttolcd(line4+11,spi_receive8_s(1,read_setpoint8),10);      //R
  sinttolcd(line4+2 ,spi_receive8_s(2,read_setpoint8),10);                //L
 } else if(dip2==1&&dip3==0) {                        //change times of sampling time
  if(flag_first[1]==0) {
   flag_first[0]=0;
   flag_first[2]=0;
   flag_first[3]=0;
   flag_first[1]=1;
   led_show(g_on);
   lcd_text(line1,"sampling :     ");
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,num_sam,10);
   lcd_text(line4,"              ");
   temp_adc=0;
  }
  temp_adc=adc_read(3)/4;
  if(temp_adc>250) {
   temp_adc=250;
  }
  if(last_temp_adc!=temp_adc) {
   lcd_text(line1+10,"    ");
   inttolcd(line1+10,temp_adc,10);
   last_temp_adc=temp_adc;
  }
  if(last_num_sam!=num_sam) {
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,num_sam,10);
   last_num_sam=num_sam;
  }
  if(bt1==0) {
   delay(dly_key);
   if(bt1==0) {
```

```
    num_sam=temp_adc;
    lcd_show(off);
    delay(1000000);
    lcd_show(g_on);
    delay(1000000);
    lcd_show(off);
    delay(1000000);
    lcd_show(g_on);
   }
  }
      } else if(dip2==0&&dip3==1) {                    //change duty cycle of sampling
  if(flag_first[2]==0) {
   flag_first[0]=0;
   flag_first[1]=0;
   flag_first[3]=0;
   flag_first[2]=1;
   lcd_show(g_on);
   lcd_text(line1,"duty PWM :     ");
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,duty_pwm,10);
   lcd_text(line4,"              ");
   temp_adc=0;
  }
  temp_adc=adc_read(3);
  if(last_temp_adc!=temp_adc) {
   lcd_text(line1+10,"     ");
   inttolcd(line1+10,temp_adc,10);
   last_temp_adc=temp_adc;
  }
  if(last_duty_pwm!=duty_pwm) {
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,duty_pwm,10);
   last_duty_pwm=duty_pwm;
  }
  if(bt1==0) {
   delay(dly_key);
   if(bt1==0) {
    duty_pwm=temp_adc;
    lcd_show(off);
    delay(1000000);
    lcd_show(r_on);
    delay(1000000);
    lcd_show(off);
    delay(1000000);
    lcd_show(g_on);
   }
  }
 } else if(dip2==0&&dip3==0) {
  if(flag_first[3]==0) {
   flag_first[0]=0;
   flag_first[1]=0;
   flag_first[2]=0;
   flag_first[3]=1;
   lcd_show(r_on);
   lcd_text(line1,"cmp angle:     ");
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,ang_now,10);
   lcd_text(line4,"              ");
  }
  temp_adc=angle();
  if(last_temp_adc!=temp_adc) {
   lcd_text(line1+10,"     ");
   sinttolcd(line1+10,temp_adc,10);
   last_temp_adc=temp_adc;
  }
  if(last_ang!=ang_now) {
   lcd_text(line2+2 ,"     ");
   inttolcd(line2+2,ang_now,10);
   last_ang=ang_now;
  }
  if(bt1==0) {
   delay(dly_key);
   if(bt1==0) {
    ang_now=temp_adc/10;
    //ang_now=temp_adc;
    lcd_show(off);
    delay(1000000);
    lcd_show(r_on);
    delay(1000000);
    lcd_show(off);
    delay(1000000);
    lcd_show(g_on);
   }
  }
 }
 delay(100000);
}

sprintf(uart_buf,"setpoint consider R:%d  L:%d  \r\n",spi_receive8_s(1,read_setpoint8),spi_receive8_s(2,read_setpoint8));          //init show terminal
putsUART1((unsigned int *)uart_buf);
while(BusyUART1());

if(dip2==0&&dip3==1) {
 flag_st_duty=1;
} else if(dip2==0&&dip3==0) {
 ang_start=ang_now;
 lcd_text(line2+2 ,"     ");
 delay(1000000);
 inttolcd(line2+2,ang_start,10);
}
delay(5000000);
lcd_text(line1," go go  robot  ");
flag_sampling=1;
```

```
}
void init_dip() {
 TRISBbits.TRISB4=1;   //limit sw1
 TRISBbits.TRISB5=1;   //limit sw2
 TRISBbits.TRISB6=1;   //dip sw1
 TRISBbits.TRISB7=1;   //dip sw2
 TRISBbits.TRISB8=1;   //dip sw3
 TRISEbits.TRISE4=1;   //dip sw3
}
void init_timer() {
 //init timer1
 CloseTimer1();
 ConfigIntTimer1(T1_INT_ON & T1_INT_PRIOR_1);
 WriteTimer1(0);
 OpenTimer1(T1_ON &
                                T1_IDLE_STOP &
                                T1_GATE_OFF &
                                T1_PS_1_8 &
        T1_SYNC_EXT_OFF &
                                T1_SOURCE_INT,match_value);
}
void init_adc() {
 ADCON1bits.ADON = 0;                  // Turn-OFF ADC Before Change Config
 // Initial ADC Channel
 SetChanADC10(ADC_CH0_POS_SAMPLEA_AN0 &        // ADC0 = Input(+)
        ADC_CH0_POS_SAMPLEA_AN1 &              // ADC1 = Input(+)
        ADC_CH0_POS_SAMPLEA_AN2 &              // ADC2 = Input(+)
        ADC_CH0_POS_SAMPLEA_AN3 &              // ADC3 = Input(+)
        ADC_CH0_NEG_SAMPLEA_NVREF);            // GND  = Input(-)
 // Initial ADC Interrupt
 ConfigIntADC10(ADC_INT_DISABLE);              // Disable ADC Interrupt
 // Open ADC Function & Turn ON ADC
 OpenADC10(ADC_MODULE_ON &                     // Turn-ON ADC Function
            ADC_IDLE_STOP &                    // Stop ADC in IDLE Mode
            ADC_FORMAT_INTG &                  // Result Format = Unsigned Integer
            ADC_CLK_MANUAL &                   // ADC Clock = Manual
            ADC_AUTO_SAMPLING_ON &                    // Enable ADC Sampling
        ADC_SAMPLE_SIMULTANEOUS,               // Sample Style = Simultaneous
        // ADC Config2
            ADC_VREF_AVDD_AVSS &                      // VDD=Vref(H),VSS=Vref(L)
            ADC_SCAN_ON &                             // Enable Scan
            ADC_CONVERT_CH0 &                  // Used ADC0 to Convert Result
            ADC_SAMPLES_PER_INT_4 &            // Number of Sample Between Interrupt
            ADC_ALT_BUF_OFF &                  // Disable Alternate Buffer
            ADC_ALT_INPUT_OFF ,                // Disable Alternate Input
            // ADC Config3
            ADC_SAMPLE_TIME_1 &                // Samplink Time = Fast
            ADC_CONV_CLK_INTERNAL_RC &         // Used Internal RC Clock Sampling
            //ADC_CONV_CLK_SYSTEM &                   // Used System Clock Sampling
            ADC_CONV_CLK_Tcy,                         // Conversion Clock = Fast
            // ADC Config Port = Enable RB[0..3] = ADC[0..3]
            ENABLE_AN0_ANA &                          // Enable RB0 = ADC0
            ENABLE_AN1_ANA &                          // Enable RB1 = ADC1
            ENABLE_AN2_ANA &                          // Enable RB2 = ADC2
        ENABLE_AN3_ANA,                        // Enable RB3 = ADC3
            // ADC Config Scan = ON ADC[0..3] , OFF ADC[4..8]
            SKIP_SCAN_AN4 &                    // Disable Scan ADC4
            SKIP_SCAN_AN5 &                    // Disable Scan ADC5
            SKIP_SCAN_AN6 &                    // Disable Scan ADC6
        SKIP_SCAN_AN7 &                        // Disable Scan ADC7
            SKIP_SCAN_AN8);                    // Disable Scan ADC8
}
void init_uart() {
 CloseUART1();                         // Disable UART1 Before New Config
 // Config UART1 Interrupt Control
 ConfigIntUART1(UART_RX_INT_DIS &      // Disable RX Interrupt
          UART_RX_INT_PR2 &            // RX Interrupt Priority = 2
          UART_TX_INT_DIS &            // Disable TX Interrupt
          UART_TX_INT_PR3);            // TX Interrupt Priority = 3
 // Open UART1 = Mode,Status,Baudrate
 OpenUART1(UART_EN  &                  // Enable UART(UART Mode)
        UART_IDLE_STOP &               // Disable UART in IDLE Mode
            UART_ALTRX_ALTTX &         // Select U1TX=RC13,U1RX=RC14
        UART_DIS_WAKE &                // Disable Wake-Up
            UART_DIS_LOOPBACK &        // Disable Loop Back
            UART_DIS_ABAUD &           // Disable Auto Baudrate
            UART_NO_PAR_8BIT &         // UART = 8 Bit, No Parity
            UART_1STOPBIT,                    // UART = 1 Stop Bit
       // Config UART1 Status
            UART_INT_TX &                     // Select Interrupt After TX Complete
            UART_TX_PIN_NORMAL &              // Normal U1TX Mode
            UART_TX_ENABLE &           // Enable U1TX
            UART_INT_RX_BUF_FUL &      // Flasg Set After RX Complete
            UART_ADR_DETECT_DIS &             // Disable Check Address
            UART_RX_OVERRUN_CLEAR,     // Clear Overrun Flag
        32);                                                  // ET-dsPIC30F2010 UART Baudrate = 57600 BPS
}
void show_uart() {
 unsigned int j=0,k=0;
 unsigned int temp_ang=0,last_temp_ang=0;
 if(bt1==0) {
  delay(100);
  if(bt1==0) {
   delay(700000);
   while(bt1) {
    led_show(alternate_blink);          //send data to terminal
    delay(500000);
   }
   delay(1000);
   while(!bt1);                                       //send data to terminal
   delay(1000);
   for(j=0;j<2;j++) {
    if(j==0) {
```

```
      sprintf(uart_buf,"\r\n %d Right\n\r",j);                    // Print Message String
      putsUART1((unsigned int *)uart_buf);        // Print uart_buff to UART1
      while(BusyUART1());
     } else if(j==1) {
      sprintf(uart_buf,"\r\n %d Left\n\r",j);                     // Print Message String
      putsUART1((unsigned int *)uart_buf);        // Print uart_buff to UART1
      while(BusyUART1());
     }
     for(k=0;k<num_sam;k++) {
      sprintf(uart_buf,"%d \n\r",data_show[j][k]);                // Print Message String
      putsUART1((unsigned int *)uart_buf);        // Print uart_buff to UART1
      while(BusyUART1());
     }
     sprintf(uart_buf,"--%d\r\n",num_sam);                       // Print Message String
     putsUART1((unsigned int *)uart_buf);         // Print uart_buff to UART1
     while(BusyUART1());
     k=0;
    }
    lcd_text(line4,"end");
    sprintf(uart_buf,"..end..\n\r");                          // Print Message String
    putsUART1((unsigned int *)uart_buf);          // Print uart_buff to UART1
    while(BusyUART1());
    while(bt1) {
     led_show(o_on);
    }
    delay(1000);
    while(!bt1);
    delay(1000000);
   }
  }
  led_show(r_blink);
  temp_ang=angle()/10;
  //temp_ang=angle();
  if(last_temp_ang!=temp_ang) {
   lcd_text(line4,"      ");
   inttolcd(line4,temp_ang,10);
   last_temp_ang=temp_ang;
  }
  delay(500000);
}

void delay(unsigned long dly) {
 while(dly--);
}
```

Program of slave controller

```
//PROGRAM: sam_80ms_slave.c <sampling every 80ms>
//PIN HARDWARE:
//QEI                          RB4 = QEA   RB5 = QEB
//DIR motor              RB1                    RB2
//MCPWM                RE5 = PWM3H
//LED                          RE2 - RE3 (-,+)>green  (+,-)>>red
#include<p30f2010.h>
#include<ports.h>                                 // Used Port & Interrupt Function Library
#include"C:\dsPIC\dspic_o00olib\spi_slave_project.h"
#include"C:\dsPIC\dspic_o00olib\lcd_slave.h"
//motor control2
#include"C:\dsPIC\dspic_o00olib\motor\R_motor_control.h"
//#include"C:\dsPIC\dspic_o00olib\motor\L_motor_control.h"
#include"C:\dsPIC\dspic_o00olib\motor\fuzzy_PI.h"
#include"C:\dsPIC\dspic_o00olib\motor\PID_controller.h"
_FOSC(CSW_FSCM_ON & XT_PLL16);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_ON & BORV_45 & PWRT_64 & MCLR_EN);
_FGS(CODE_PROT_OFF);
//Function Prototype
void init();                                              //init data
void init_int0();
void delay(unsigned long dly);
void sorting_type_dat();
//INT0 Interrupt Service Active by Port Pin RE8
void _ISR _INT0Interrupt(void) {
 sorting_type_dat();
 IFS0bits.INT0IF = 0;                    // Reset INT0 Interrupt Flag
}
int main() {
 setpoint=0;                                              //speed>>rpm
 init();
 while(1);
}
void init() {
 motor_control_init();
 init_spi_slave();
 init_led_slave();
 init_int0();
}
void init_int0() {
 ConfigINT0(FALLING_EDGE_INT &     // Falling Edge Trigger Interrupt
        EXT_INT_ENABLE &       // Enable INT0 Interrupt
        EXT_INT_PRI_6);       // INT0 Interrupt Priority = 6
}
void delay(unsigned long dly) {
 while(dly--);
}
void sorting_type_dat() {
 switch(spi_in_slave()) {
  //slave receive 8 bits data
  case(change_setpoint8): {
   setpoint=spi_in_slave_s();
```

```
  } break;
  case(stop_motor): {
   if(spi_in_slave()==0) {
    dir1=0; dir2=0;
    setpoint=0;
   }
  } break;
  case(cal_speed): {
   if(spi_in_slave()==0) {
    cal_speed80ms();
   }
  } break;
  case(FLC_sp): {                                  //FLC-PI 5 o/p
   if(spi_in_slave()==0) {
    FLC_PI(setpoint);
   }
  } break;
  case(P_sp): {
   if(spi_in_slave()==0) {
    P_controller(setpoint);
   }
  } break;
  case(PI_sp): {
   if(spi_in_slave()==0) {
    PI_controller(setpoint);
   }
  } break;
  case(FLC_P_sp): {
   if(spi_in_slave()==0) {
    FLC_P(setpoint);
   }
  } break;
  case(FLC_I_sp): {
   if(spi_in_slave()==0) {
    FLC_I(setpoint);
   }
  } break;
  //slave send 8 bits data
  case(read_speed8): {
   spi_out_slave_s((signed char)real_speed);
  } break;
  case(read_setpoint8): {
   spi_out_slave_s((signed char)setpoint);
  } break;
  //slave receive 16 bits data
  case(change_setpoint16): {
   setpoint=spi_in_slave16_s();
  } break;
  case(change_duty16): {
   speed_output(3,spi_in_slave16_s());
  } break;
  //slave send 16 bits data
  case(read_speed16): {
   spi_out_slave16_s((signed int)real_speed);
  } break;
  case(read_setpoint16): {
   spi_out_slave16_s((signed int)setpoint);
  } break;
 }
}
//credit by o00o
//robot club kmitl
```

Library Program of lcd display module (Master)

```
//#include"C:\dsPIC\dspic_o00olib\lcd_medspic.h"
//reference lcd display 16*4
//Pin Hardware
//LCD on portE   RD0 = RS   RD1 = E   GND = RW   RE0-RE3 = D4-D7
#define line1 0x80
#define line2 0xC0
#define line3 0x90
#define line4 0xD0
#define lcd0; lcd_text(line4+15," ");
#define lcd1; lcd_text(line4+15,".");
#define num_of_digit 32
#define lcd_clear()  lcd_command(0x01)
#define lcd_origin() lcd_command(0x02)
#define lcd_on()          lcd_command(0x0F)
#define lcd_off()         lcd_command(0x0B)
/* Display ON/OFF Control */
#define DON                0x0F     // Display on
#define DOFF                        0x0B      // Display off
#define CURSOR_ON     0x0F          // Cursor on
#define CURSOR_OFF    0x0D          // Cursor off
#define BLINK_ON   0x0F             // Cursor Blink
#define BLINK_OFF  0x0E             // Cursor No Blink
/* Cursor or Display Shift */
#define SHIFT_CUR_LEFT  0x13 // Cursor shifts to the left
#define SHIFT_CUR_RIGHT 0x17 // Cursor shifts to the right
#define SHIFT_DISP_LEFT 0x1B // Display shifts to the left
#define SHIFT_DISP_RIGHT 0x1F // Display shifts to the right
/*Pin Hardware*/                                // Data Pin = RE0-RE3
#define rs    LATDbits.LATD0        // RD0 = RS
#define e     LATDbits.LATD1        // RD1 = E
#define dly_cmd  5500
#define dly_init 25000
unsigned char flag_lcd=0;           //flag blink status
/*Function delay*/
```

```
void lcd_delay(unsigned int dly) {
 while(dly--);
}
/*Function send command to LCD format 4 Bit*/
void lcd_command(unsigned char cmd) {
 unsigned char temp;
 temp=(cmd&0xF0)>>4;
 rs=0;
 e=1;
 LATE=(LATE&0xF0)|temp;
 lcd_delay(dly_cmd);
 e=0;
 lcd_delay(dly_cmd);

 temp=cmd&0x0F;
 e=1;
 LATE=(LATE&0xF0)|temp;
 lcd_delay(dly_cmd);
 e=0;
 lcd_delay(dly_cmd);
}
/*Function send data to LCD format 4 Bit*/
void lcd_data(unsigned char dat) {
 unsigned char temp;
 temp=(dat&0xF0)>>4;
 rs=1;
 e=1;
 LATE=(LATE&0xF0)|temp;
 lcd_delay(dly_cmd);
 e=0;
 lcd_delay(dly_cmd);
 temp=dat&0x0F;
 e=1;
 LATE=(LATE&0xF0)|temp;
 lcd_delay(dly_cmd);
 e=0;
 lcd_delay(dly_cmd);
}
/*Function show string message*/
void lcd_text(unsigned char line,char *p) {
 lcd_origin();
 lcd_command(line);
 while(*p) {
  lcd_data(*p);
  p++;
 }
}
/*Function set intial format 4 Bit*/
void lcd_init() {              //Intial LCD 4bit
 TRISEbits.TRISE0=0;
 TRISEbits.TRISE1=0;
 TRISEbits.TRISE2=0;
 TRISEbits.TRISE3=0;
 TRISDbits.TRISD0=0;
 TRISDbits.TRISD1=0;
 lcd_delay(dly_init);
 lcd_command(0x33);
 lcd_command(0x32);
 lcd_command(0x28);
 lcd_command(0x0C);
 lcd_command(0x01);
}
void ultoa(unsigned long value,char *string,unsigned char radix) {
 unsigned char index;
 char buffer[num_of_digit];
  index=num_of_digit;
 do
 {
  buffer[--index] = '0' + (value % radix);
  if(buffer[index] > '9')
   buffer[index] += 'A'-'9'-1;
  value /= radix;
 } while(value != 0);
 do
 {
  *string++ = buffer[index++];
 } while(index < num_of_digit);
 *string=0;
}

void ltoa(long value_l,char *string_l,unsigned char radix_l) {
 if(value_l < 0 && radix_l == 10) {
  *string_l++ = '-';
  ultoa(-value_l,string_l,radix_l);
 }
 else {
  ultoa(value_l,string_l,radix_l);
 }
}
void intolcd(unsigned char posi,long value,int format) {
 char buff[12];
 ultoa(value,&buff[0],format);
 lcd_text(posi,&buff[0]);
}
void sintolcd(unsigned char posi,long value,int format) {
 char buff[12];
 ltoa(value,&buff[0],format);
 lcd_text(posi,&buff[0]);
}
void lcd_blink() {
 if(flag_lcd==0) {
  flag_lcd=1;
  lcd0;
```

```
 } else {
  flag_lcd=0;
  lcd1;
 }
}
```

Library Program of communication via SPI bus interface (Master)

```
//#include"C:\dsPIC\dspic_o00olib\spi_master_project.h"
//Pin Hardware
//communicate spi bus
#define cs1   LATFbits.LATF0          //o/p
#define cs2   LATFbits.LATF1          //o/p
#define en_spi              LATFbits.LATF4          //o/p         not use
#define miso PORTFbits.RF5   //i/p
#define mosi LATFbits.LATF6           //o/p
#define sclk  LATEbits.LATE8          //o/p
//type of data
//master send 8 bits 0xAX
#define change_setpoint8  0xA0
#define stop_motor                    0xA1
#define cal_speed                     0xA2          //every 20ms
#define FLC_sp                                      0xA3          //fuzzy PI
#define P_sp                          0xA4
#define PI_sp                                       0xA5
#define FLC_P_sp                      0xA6
#define FLC_I_sp                      0xA7
//master receive 8 bits 0xBX
#define read_speed8                   0xB0
#define read_setpoint8          0xB1
//master send 16 bits 0xCX
#define change_setpoint16 0xC0
#define change_duty16           0xC1
//master receive 16 bits 0xDX
#define read_speed16            0xD0
#define read_setpoint16         0xD1
//endtype of data
//delay for waiting something
#define dly_clk   15
#define dly_ready  25
#define dly_process_complete 65
//void spi_send8(unsigned char ch,unsigned char type_dat,unsigned char dat);
//unsigned char spi_receive8(unsigned char ch,unsigned char type_dat);
//void spi_send16(unsigned char ch,unsigned char type_dat,unsigned int dat);
//unsigned int spi_receive16(unsigned char ch,unsigned char type_dat);
//
void delay_spi(unsigned long dly) {
 while(dly--);
}
//
void init_spi_master() {
 TRISFbits.TRISF0=0;    //cs1
 TRISFbits.TRISF1=0;    //cs2
 TRISFbits.TRISF4=0;    //en_spi
 TRISFbits.TRISF5=1;    //miso
 TRISFbits.TRISF6=0;    //mosi
 TRISEbits.TRISE8=0;    //sclk
 cs1=1;
 cs2=1;
 en_spi=1;
 mosi=1;
 sclk=1;
}
//
void cs_pluse(unsigned char cs_ch) {
 if(cs_ch==1) {
  cs1=0;
  delay_spi(dly_clk);
  cs1=1;
  delay_spi(dly_clk);
 } else if(cs_ch==2) {
  cs2=0;
  delay_spi(dly_clk);
  cs2=1;
  delay_spi(dly_clk);
 }
}
//
//MASTER send-recieve 8 bits with SPI BUS
void spi_out_master(unsigned char dat) {
 unsigned char i=0;
 for(i=0;i<8;i++) {
  delay_spi(dly_clk);
  if((dat&0x80)!=0) {
   mosi=1;
  } else {
   mosi=0;
  }
  sclk=1;
  delay_spi(dly_clk);
  sclk=0;
  dat<<=1;
 }
}
//check have completed
void spi_out_master_s(signed char dat) {
 unsigned char i=0;
 for(i=0;i<8;i++) {
  delay_spi(dly_clk);
```

```
  if((dat&0x80)!=0) {
   mosi=1;
  } else {
   mosi=0;
  }
  sclk=1;
  delay_spi(dly_clk);
  sclk=0;
  dat<<=1;
 }
}
//
unsigned char spi_in_master() {
 unsigned char dat=0,i=0;
 for(i=0;i<8;i++) {
  delay_spi(dly_clk);
  dat<<=1;
  sclk=1;
  delay_spi(dly_clk);
  dat|=miso;
  sclk=0;
 }
 return(dat);
}
//check have completed
signed char spi_in_master_s() {
 signed char dat=0;
 unsigned char i=0;
 for(i=0;i<8;i++) {
  delay_spi(dly_clk);
  dat<<=1;
  sclk=1;
  delay_spi(dly_clk);
  dat|=miso;
  sclk=0;
 }
 return(dat);
}
//end MASTER send-recieve 8 bits with SPI BUS
//
//MASTER send-recieve 16 bits with SPI BUS
void spi_out_master16(unsigned int dat) {
 unsigned char i=0;
 for(i=0;i<16;i++) {
  delay_spi(dly_clk);
  if((dat&0x8000)!=0) {
   mosi=1;
  } else {
   mosi=0;
  }
  sclk=1;
  delay_spi(dly_clk);
  sclk=0;
  dat<<=1;
 }
}
//check have completed
void spi_out_master16_s(signed int dat) {
 unsigned char i=0;
 for(i=0;i<16;i++) {
  delay_spi(dly_clk);
  if((dat&0x8000)!=0) {
   mosi=1;
  } else {
   mosi=0;
  }
  sclk=1;
  delay_spi(dly_clk);
  sclk=0;
  dat<<=1;
 }
}
//
unsigned int spi_in_master16() {
 unsigned int dat=0;
 unsigned char i=0;
 for(i=0;i<16;i++) {
  delay_spi(dly_clk);
  dat<<=1;
  sclk=1;
  delay_spi(dly_clk);
  dat|=miso;
  sclk=0;
 }
 return(dat);
}
//check have completed
signed int spi_in_master16_s() {
 signed int dat=0;
 unsigned char i=0;
 for(i=0;i<16;i++) {
  delay_spi(dly_clk);
  dat<<=1;
  sclk=1;
  delay_spi(dly_clk);
  dat|=miso;
  sclk=0;
 }
 return(dat);
}
//end MASTER send-recieve 16 bits with SPI BUS
//
//functions for send-receive the packet data
```

```
void spi_send8(unsigned char ch,unsigned char type_dat,unsigned char dat) {//send 8 bits
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 spi_out_master(dat);
 delay_spi(dly_process_complete);
}
//check have completed
void spi_send8_s(unsigned char ch,unsigned char type_dat,signed char dat) {//send 8 bits
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 spi_out_master_s(dat);
 delay_spi(dly_process_complete);
}
//
unsigned char spi_receive8(unsigned char ch,unsigned char type_dat) {                    //receive 8 bits
 unsigned char temp=0;
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 temp=spi_in_master();
 delay_spi(dly_process_complete);
 return(temp);
}
//check have completed
signed char spi_receive8_s(unsigned char ch,unsigned char type_dat) {                    //receive 8 bits
 signed char temp=0;
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 temp=spi_in_master_s();
 delay_spi(dly_process_complete);
 return(temp);
}
//
void spi_send16(unsigned char ch,unsigned char type_dat,unsigned int dat) {      //send 16 bits
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 spi_out_master16(dat);
 delay_spi(dly_process_complete);
}
//check have completed
void spi_send16_s(unsigned char ch,unsigned char type_dat,signed int dat) {      //send 16 bits
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 spi_out_master16_s(dat);
 delay_spi(dly_process_complete);
}
//
unsigned int spi_receive16(unsigned char ch,unsigned char type_dat) {                    //receive 16 bits
 unsigned int temp=0;
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 temp=spi_in_master16();
 delay_spi(dly_process_complete);
 return(temp);
}
//check have completed
signed int spi_receive16_s(unsigned char ch,unsigned char type_dat) {                    //receive 16 bits
 signed int temp=0;
 if(ch==1) {
  cs_pluse(1);
 } else if(ch==2) {
  cs_pluse(2);
```

```
 }
 delay_spi(dly_ready);
 spi_out_master(type_dat);
 delay_spi(dly_ready);
 temp=spi_in_master16_s();
 delay_spi(dly_process_complete);
 return(temp);}
```

Library Program of calculation motor speed (Slave)

```
//#include"C:\dsPIC\dspic_o00olib\motor\cal_speed_80ms.h"
//want variable (signed int count_step=0,count_roumnd=0;) from X_motor_control.h
signed int last_count_step=0,last_count_round=0;      //use for calculate speed of motor <in function cal_speed()>
signed int real_speed=0;
void cal_speed80ms() {                    //calculate every 80ms for encoder 200 ppr
 count_step=POSCNT;
 real_speed=((count_step-last_count_step)+((count_round-last_count_round)*num_ppr))*0.9375;          //200ppr,Ts=80ms
speed(rpm)=12.5(s)*60(min)/800(ppr) = 0.9375rpm
 last_count_round=count_round;
 last_count_step=count_step;
}
```

Library Program of PID controller (Master)

```
//#include"C:\dsPIC\dspic_o00olib\motor\PID_controller.h"
//P & PI controller
#define Kp 10
#define Ki 5.5
#define Kd 0
signed int duty_acc=0;
void P_controller(signed int setpoint) {
 signed int error_speed;
 signed long duty_output=0;
 error_speed=setpoint-real_speed;
 duty_output=Kp*error_speed;
 if(duty_output>30000) {
  duty_output=30000;
 } else if(duty_output<-30000) {
  duty_output=-30000;
 }
 speed_output(3,(signed int)duty_output);                    //set pwm duty
}
void PI_controller(signed int setpoint) {
 signed int error_speed,duty_output=0;
 error_speed=setpoint-real_speed;
 if((Kp*error_speed)>30000) {
  duty_output=30000;
 } else if((Kp*error_speed)<-30000) {
  duty_output=-30000;
 } else {
  duty_output=Kp*error_speed;
 }
 duty_acc+=Ki*error_speed;
 if(duty_acc>30000) {
  duty_acc=30000;
 } else if(duty_acc<-30000) {
  duty_acc=-30000;
 }
 duty_output+=duty_acc;
 if(duty_output>30000) {
  duty_output=30000;
 } else if(duty_output<-30000) {
  duty_output=-30000;
 }
 speed_output(3,duty_output);                    //set pwm duty
}
```

Library Program of slave controller which control left motor

```
//#include"C:\dsPIC\dspic_o00olib\motor\L_motor_control.h"
//QEI PWM(dir motor & mcpwm)
//PIN HARDWARE:
//QEI                                   RB4 = QEA   RB5 = QEB
//DIR motor                RB1                        RB2
//MCPWM                      RE5 = PWM3H
#define dir1 LATBbits.LATB1
#define dir2 LATBbits.LATB2
//Left motor
#define forward  1
#define backward 0
/*//Right motor
#define forward  0
#define backward 1
*/
#define maxcnt_value  799//for motor test (encoder 200 ppr) 4n-1=4(200)-1=799
#define num_ppr (maxcnt_value+1)

signed char flag_count_round;                                                    //consider count round >> negative first round =
-1 ,positive first round = 0 ,normal case = 1
signed int  count_step=0,count_round=0;                                  //use for calculate speed of motor <in function cal_speed()>
signed int  setpoint=0;
#include<pwm.h>
```

```
#include<qei.h>
#include"C:\dsPIC\dspic_o00olib\motor\cal_speed_80ms.h"  //for cal speed of motor
void mcpwm_init();
void dir_init();
void speed_output(signed char ch,signed int duty);
void qei_init();
void motor_control_init();
void _ISR _QEIInterrupt() {
 if(QEICONbits.UPDN) {                  //case step count up
  POSCNT=0;
  if(flag_count_round==-1) {
   flag_count_round=0;
  } else {
   flag_count_round=1;
   led_show(r_blink);
   if(++count_round>=30000) {
    count_round=30000;
   }
  }
 }
 else {
  POSCNT=maxcnt_value;
  if(flag_count_round==0) {
   flag_count_round=-1;
  } else if(flag_count_round==-1||flag_count_round==1) {
   flag_count_round=1;
   led_show(g_blink);
   if(--count_round<=-30000) {
    count_round=-30000;
   }
  }
 }
 IFS2bits.QEIIF=0;
}
//PWM
void mcpwm_init() {
 unsigned int period=512,sptime,config1,config2,config3;
 CloseMCPWM();
 sptime=0x00;          // SEVTCMP = Special Time(Not Used)
 config1=PWM_EN &                       // Enable PWM Function
   PWM_IDLE_STOP &                      // Disable PWM in IDLE Mode
   PWM_OP_SCALE1 &                      // PWM Post Scale = 1
   PWM_IPCLK_SCALE1 &                   // PWM Input Clock Prescale = 1
   PWM_MOD_FREE ;                       // PWM = Free Running
  config2=PWM_MOD1_IND &                // PWM1 = Free Mode
   PWM_MOD2_IND &                       // PWM2 = Free Mode
   PWM_MOD3_IND &                       // PWM3 = Free Mode
   PWM_PEN3H &                          // H of channel 3 work as PWM
   PWM_PDIS3L &         // L of channel 3 work as IO
   PWM_PDIS2H &         // H of channel 2 work as IO
   PWM_PDIS2L &         // L of channel 2 work as IO
   PWM_PDIS1H &         // H of channel 1 work as IO
   PWM_PDIS1L;          // L of channel 1 work as IO
 config3=PWM_SEVOPS1 &     // Special Even Post Scaler = 1:1
   PWM_OSYNC_PWM &     // Overide Sync. With PWM Clock
    PWM_UEN ;       // Enable PWM Update
 OpenMCPWM(512,sptime,config1,config2,config3);
}
void dir_init() {
 TRISBbits.TRISB1 = 0;    // Config RB1 = Output
 TRISBbits.TRISB2 = 0;    // Config RB2 = Output
 dir1=0;
 dir2=0;
}

void speed_output(signed char ch,signed int duty) {
 if(duty>0) {
  dir1=backward; dir2=forward;
  if(duty>1023) {
   duty=1023;
  }
  SetDCMCPWM(ch,duty,0);
 } else if(duty<0) {
  dir1=forward; dir2=backward;
  if(duty<-1023) {
   duty=-1023;
  }
  duty=abs(duty);
  SetDCMCPWM(ch,duty,0);
 } else {
  dir1=0; dir2=0;
  SetDCMCPWM(ch,0,0);
 }
}
//QEI
void qei_init() {
 unsigned int config1;
 ADPCFG=0xFFFF;
 ConfigIntQEI(QEI_INT_PRI_7 &
                                        QEI_INT_ENABLE);
 POSCNT=0;
 MAXCNT=maxcnt_value;
  config1=(QEI_DIR_SEL_QEB &
                          QEI_INT_CLK &
                          QEI_INDEX_RESET_DISABLE &
                          QEI_CLK_PRESCALE_1 &
                          QEI_GATED_ACC_DISABLE &
                          QEI_NORMAL_IO &
    //QEI_INPUTS_NOSWAP &                  //R
                          QEI_INPUTS_SWAP &                 //L
                          QEI_MODE_x4_MATCH &
                          QEI_UP_COUNT &
                          QEI_IDLE_CON);
```

```
 OpenQEI(config1,0);
}
void motor_control_init() {
 mcpwm_init();
 dir_init();
 qei_init();
}
```

Library Program of slave controller which control right motor

```
//#include"C:\dsPIC\dspic_o00olib\motor\R_motor_control.h"
//QEI PWM(dir motor & mcpwm)
//PIN HARDWARE:
//QEI                                RB4 = QEA   RB5 = QEB
//DIR motor                RB1                        RB2
//MCPWM                      RE5 = PWM3H
#define dir1 LATBbits.LATB1
#define dir2 LATBbits.LATB2
/*//Left motor
#define forward  1
#define backward 0
*/
//Right motor
#define forward  0
#define backward 1
#define maxcnt_value  799//for motor test (encoder 200 ppr) 4n-1=4(200)-1=799
#define num_ppr (maxcnt_value+1)
signed char flag_count_round;                                          //consider count round >> negative first round =
-1 ,positive first round = 0 ,normal case = 1
signed int  count_step=0,count_round=0;                                //use for calculate speed of motor <in function cal_speed()>
signed int  setpoint=0;
#include<pwm.h>
#include<qei.h>
#include"C:\dsPIC\dspic_o00olib\motor\cal_speed_80ms.h"  //for cal speed of motor
void mcpwm_init();
void dir_init();
void speed_output(signed char ch,signed int duty);
void qei_init();
void motor_control_init();
void _ISR _QEIInterrupt() {
 if(QEICONbits.UPDN) {                 //case step count up
  POSCNT=0;
  if(flag_count_round==-1) {
   flag_count_round=0;
  } else {
   flag_count_round=1;
   led_show(r_blink);
   if(++count_round>=30000) {
    count_round=30000;
   }
  }
 }
 else {
  POSCNT=maxcnt_value;
  if(flag_count_round==0) {
   flag_count_round=-1;
  } else if(flag_count_round==-1||flag_count_round==1) {
   flag_count_round=1;
   led_show(g_blink);
   if(--count_round<=-30000) {
    count_round=-30000;
   }
  }
 }
 IFS2bits.QEIIF=0;
}
//PWM
void mcpwm_init() {
 unsigned int period=512,sptime,config1,config2,config3;
 CloseMCPWM();
 sptime=0x00;          // SEVTCMP = Special Time(Not Used)
 config1=PWM_EN &                  // Enable PWM Function
    PWM_IDLE_STOP &                // Disable PWM in IDLE Mode
    PWM_OP_SCALE1 &                // PWM Post Scale = 1
    PWM_IPCLK_SCALE1 &             // PWM Input Clock Prescale = 1
    PWM_MOD_FREE ;                 // PWM = Free Running
 config2=PWM_MOD1_IND &            // PWM1 = Free Mode
     PWM_MOD2_IND &                // PWM2 = Free Mode
     PWM_MOD3_IND &                // PWM3 = Free Mode
     PWM_PEN3H &                   // H of channel 3 work as PWM
     PWM_PDIS3L &        // L of channel 3 work as IO
     PWM_PDIS2H &        // H of channel 2 work as IO
     PWM_PDIS2L &        // L of channel 2 work as IO
     PWM_PDIS1H &        // H of channel 1 work as IO
     PWM_PDIS1L;         // L of channel 1 work as IO
 config3=PWM_SEVOPS1 &     // Special Even Post Scaler = 1:1
    PWM_OSYNC_PWM &     // Overide Sync. With PWM Clock
      PWM_UEN ;       // Enable PWM Update
 OpenMCPWM(512,sptime,config1,config2,config3);
}
void dir_init() {
 TRISBbits.TRISB1 = 0;    // Config RB1 = Output
 TRISBbits.TRISB2 = 0;    // Config RB2 = Output
 dir1=0;
 dir2=0;
}
void speed_output(signed char ch,signed int duty) {
 if(duty>0) {
  dir1=backward; dir2=forward;
  if(duty>1023) {
   duty=1023;
```

```
  }
  SetDCMCPWM(ch,duty,0);
 } else if(duty<0) {
  dir1=forward; dir2=backward;
  if(duty<-1023) {
   duty=-1023;
  }
  duty=abs(duty);
  SetDCMCPWM(ch,duty,0);
 } else {
  dir1=0; dir2=0;
  SetDCMCPWM(ch,0,0);
 }
}
//QEI
void qei_init() {
 unsigned int config1;
 ADPCFG=0xFFFF;
 ConfigIntQEI(QEI_INT_PRI_7 &
                                  QEI_INT_ENABLE);
 POSCNT=0;
 MAXCNT=maxcnt_value;
  config1=(QEI_DIR_SEL_QEB &
                        QEI_INT_CLK &
                        QEI_INDEX_RESET_DISABLE &
                        QEI_CLK_PRESCALE_1 &
                        QEI_GATED_ACC_DISABLE &
                        QEI_NORMAL_IO &
     QEI_INPUTS_NOSWAP &                //R
                        //QEI_INPUTS_SWAP &              //L
                        QEI_MODE_x4_MATCH &
                        QEI_UP_COUNT &
                        QEI_IDLE_CON);
 OpenQEI(config1,0);
}
void motor_control_init() {
 mcpwm_init();
 dir_init();
 qei_init();
}
```

Library Program of Fuzzy I controller (Master)

```
//#include"C:\dsPIC\dspic_o00olib\motor\fuzzy_1.h"
//Fuzzy Control Speed module
//Fuzzy-I controller (sampling every 20ms)
//I/P error and derror        >> speed(rpm)
//O/P duty                              >> -1023 - 1023
//min-max speed motor     >> -100 - 100 rpm
//want >> real_speed grobal variable in main program
#define error  0
#define P    -1
#define Z    0
#define N    1
//limit of graph degree of membership
//error
#define Llim_error  -55
#define Ulim_error   55
#define m_error              0.0182        //slope error  = 1/55
//membership function of output   dduty >> (-100 - 100)
#define oN -250
#define oZ   0
#define oP  250
float uP=0,uZ=0,uN=0;    //degree of membership function of error
float uoN=0,uoZ=0,uoP=0;              //degree of membership function of dvoltage
signed int duty=0;
void fuzzifer(float value);
void rule_base(void);
//void inference(void);
float defuzzifier(void);
float min(float value1,float value2);
float max(float value1,float value2);
void clear_uMF(void);
float u_MF(signed char mf_con,float value);
float FLC(signed int error_value);
void FLC_I(signed int set_point);
//fuzzy-I controller
void FLC_I(signed int setpoint) {
 signed int duty_output=0;
 duty+=FLC(setpoint-real_speed);       //FLC(error)=duty          >> (-1023 - 1023)
 //limit
 if(duty>30000) {                                  //upper limit
  duty=30000;
 } else if(duty<-30000) {  //lower limit
  duty=-30000;
 }
 //output limit
 if(duty>1023) {                                   //upper output limit
  duty_output=1023;
 } else if(duty<-1023) {    //lower output limit
  duty_output=-1023;
 } else {
  duty_output=duty;
 }
 speed_output(3,duty_output);                      //set pwm duty
}
void fuzzifer(float value) {
 uP=u_MF(P,value);              //error is positive
 uZ=u_MF(Z,value);              //error is zero
 uN=u_MF(N,value);               //error is negative
}
```

```
void rule_base() {            //FLC have 5 rules.
 //rule 1: if( error is Z ) then ( voltage is oZ )
 if(uZ>0) {
  uoZ=uZ;
 } else {
  uoZ=0;
 }
 //rule 2: if( error is P ) then ( dvoltage is oN )
 if(uP>0) {
  uoN=uP;
 } else {
  uoN=0;
 }
  //rule 3: if( error is N ) then ( dvoltage is oP )
 if(uN>0) {
  uoP=uN;
 } else {
  uoP=0;
 }
}
/*void inference() {
 //not use
}*/
float defuzzifier() {    //COA
 float output=0,sum_uoutput=0;
 output+=(oP*uoP);
 output+=(oZ*uoZ);
 output+=(oN*uoN);
 sum_uoutput=uoP+uoZ+uoN;
 return(output/sum_uoutput);
}
float min(float value1,float value2) {
 float value=0;
 if(value1<=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
float max(float value1,float value2) {
 float value=0;
 if(value1>=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
void clear_uMF() {
 //clear all previous data
  uP=0; uZ=0; uN=0;
  uoN=0;  uoZ=0; uoP=0;
}
float u_MF(signed char mf_con,float value) {
 float temp_u_mf=0;
 //input is error
 switch(mf_con) {
  case(P): {  //error is positive
   if(value<=Llim_error) {
    temp_u_mf=1;
   } else if((value>Llim_error)&&(value<0)) {
    temp_u_mf=(-m_error)*value;
   } else {
    temp_u_mf=0;
   }
  } break;
  case(Z): {  //error is zero
   if((value>Llim_error)&&(value<0)) {
    temp_u_mf=((m_error)*value)+1;
   } else if(value==0){
    temp_u_mf=1;
   } else if((value>0)&&(value<Ulim_error)) {
    temp_u_mf=((-m_error)*value)+1;
   } else {
    temp_u_mf=0;
   }
  } break;
  case(N): {  //error is negative
   if((value>0)&&(value<Ulim_error)) {
    temp_u_mf=(m_error)*value;
   } else if(value>=Ulim_error){
    temp_u_mf=1;
   } else {
    temp_u_mf=0;
   }
  } break;
 }
 return(temp_u_mf);
}
float FLC(signed int error_value) {
 clear_uMF();
 fuzzifier(error_value);
 rule_base();
 //inference();
 return(defuzzifier());
}
```

Library Program of Fuzzy P controller (Master)

```
//#include"C:\dsPIC\dspic_o00olib\motor\fuzzy_P.h"
//Fuzzy Control Speed module
//Fuzzy-P controller (sampling every 20ms)
//I/P error and derror          >> speed(rpm)
//O/P duty                                  >> -1023 - 1023
//min-max speed motor     >> -100 - 100 rpm
//want >> real_speed grobal variable in main program
#define error  0
#define P     -1
#define Z     0
#define N     1
//limit of graph degree of membership
//error
#define Llim_error  -80
#define Ulim_error   80
#define m_error              0.0125        //slope error  = 1/80
//membership function of output   dduty >> (-100 - 100)
#define oN -900
#define oZ  0
#define oP  900
float uP=0,uZ=0,uN=0;     //degree of membership function of error
float uoN=0,uoZ=0,uoP=0;                  //degree of membership function of dvoltage
void fuzzifer(float value);
void rule_base(void);
//void inference(void);
float defuzzifier(void);
float min(float value1,float value2);
float max(float value1,float value2);
void clear_uMF(void);
float u_MF(signed char mf_con,float value);
float FLC(signed int error_value);
void FLC_P(signed int set_point);
//fuzzy-P controller
void FLC_P(signed int setpoint) {
 signed int duty_output=0;
 duty_output=(signed int)FLC(setpoint-real_speed);  //FLC(error)=duty          >> (-1023 - 1023)
 //output limit
 if(duty_output>1023) {                             //upper output limit
  duty_output=1023;
 } else if(duty_output<-1023) {             //lower output limit
  duty_output=-1023;
 }
 speed_output(3,duty_output);                       //set pwm duty
}
void fuzzifer(float value) {                //input1 is  error
 uP=u_MF(P,value);              //error is positive
 uZ=u_MF(Z,value);               //error is zero
 uN=u_MF(N,value);               //error is negative
}
void rule_base() {            //FLC have 5 rules.
 //rule 1: if( error is Z ) then ( voltage is oZ )
 if(uZ>0) {
  uoZ=uZ;
 } else {
  uoZ=0;
 }
 //rule 2: if( error is P ) then ( dvoltage is oN )
 if(uP>0) {
  uoN=uP;
 } else {
  uoN=0;
 }
  //rule 3: if( error is N ) then ( dvoltage is oP )
 if(uN>0) {
  uoP=uN;
 } else {
  uoP=0;
 }
}
/*void inference() {
 //not use
}*/
float defuzzifier() {    //COA
 float output=0,sum_uoutput=0;
 output+=(oP*uoP);
 output+=(oZ*uoZ);
 output+=(oN*uoN);
 sum_uoutput=uoP+uoZ+uoN;
 return(output/sum_uoutput);
}
float min(float value1,float value2) {
 float value=0;
 if(value1<=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
float max(float value1,float value2) {
 float value=0;
 if(value1>=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
void clear_uMF() {
 //clear all previous data
```

```
 uP=0; uZ=0; uN=0;
 uoN=0; uoZ=0; uoP=0;
}
float u_MF(signed char mf_con,float value) {
 float temp_u_mf=0;
 //input is error
 switch(mf_con) {
  case(P): {  //error is positive
   if(value<=Llim_error) {
    temp_u_mf=1;
   } else if((value>Llim_error)&&(value<0)) {
    temp_u_mf=(-m_error)*value;
   } else {
    temp_u_mf=0;
   }
  } break;
  case(Z): {  //error is zero
   if((value>Llim_error)&&(value<0)) {
    temp_u_mf=((m_error)*value)+1;
   } else if(value==0){
    temp_u_mf=1;
   } else if((value>0)&&(value<Ulim_error)) {
    temp_u_mf=((-m_error)*value)+1;
   } else {
    temp_u_mf=0;
   }
  } break;
  case(N): {  //error is negative
   if((value>0)&&(value<Ulim_error)) {
    temp_u_mf=(m_error)*value;
   } else if(value>=Ulim_error){
    temp_u_mf=1;
   } else {
    temp_u_mf=0;
   }
  } break;
 }
 return(temp_u_mf);
}
float FLC(signed int error_value) {
 clear_uMF();
 fuzzifer(error_value);
 rule_base();
 //inference();
 return(defuzzifier()); }
```

Library Program of Fuzzy P-I controller (Master)

```
///#include"C:\dsPIC\dspic_o00olib\motor\fuzzy_P_I.h"
//Fuzzy P-I controller (sampling every 20ms)
//control speed muti-motor
#define P    -1
#define Z    0
#define N    1
#define F_P   0
#define F_I   1
//P controller  limit of graph degree of membership
#define Llim_error_p  -80
#define Ulim_error_p  80
#define m_error_p          0.0125       //slope error  = 1/80
//membership function of output
#define oN_p -1000
#define oZ_p  0
#define oP_p  1000
//I controller  limit of graph degree of membership
#define Llim_error_i  -15
#define Ulim_error_i  15
#define m_error_i          0.0667       //slope error  = 1/15
//membership function of output
#define oN_i  -4
#define oZ_i  0
#define oP_i  4
float uP=0 ,uZ=0 ,uN=0;   //degree of membership function of error
float uoN=0,uoZ=0,uoP=0;                 //degree of membership function of dvoltage
signed int op_I=0;
void fuzzifer(float value,unsigned char type_fuzzy);
void rule_base(void);
//void inference(void);
float defuzzifier(unsigned char type_fuzzy);
float min(float value1,float value2);
float max(float value1,float value2);
void clear_uMF(void);
float u_MF(signed char mf_con,float value,unsigned char type_fuzzy);
float FLC(float error_value,unsigned char type_fuzzy);
//this function want 3 variable concluded setpoint_con,speed_L and speed_R
//cal speed L-R
//mul_FLC_P_I
//--step--
//                        sum speed before I block
//                        I controller
//                        sum (setpoint,unity feedback,I path)
void FLC_P_I(signed int setpoint_con,signed char speed_R,signed char speed_L) {
 signed int fw_path_L=0,fw_path_R=0;
 op_I+=(signed int)FLC((speed_L-speed_R),F_I);
 if(op_I>30000) {                                        //limit output I controller
  op_I=30000;
 } else if(op_I<-30000) {
  op_I=-30000;
```

```
 }
 fw_path_L=setpoint_con-op_I-speed_L;
 if(fw_path_L>30000) {
  fw_path_L=30000;
 } else if(fw_path_L<-30000) {
  fw_path_L=-30000;
 }
 fw_path_R=setpoint_con+op_I-speed_R;
 if(fw_path_R>30000) {
  fw_path_R=30000;
 } else if(fw_path_R<-30000) {
  fw_path_R=-30000;
 }
 spi_send16_s(1,change_duty16,FLC(fw_path_R,F_P));                    //R
 spi_send16_s(2,change_duty16,FLC(fw_path_L,F_P));                    //L
}
void fuzzifer(float value,unsigned char type_fuzzy) {  //input is  error
 uP=u_MF(P,value,type_fuzzy);                          //error is positive
 uZ=u_MF(Z,value,type_fuzzy);        //error is zero
 uN=u_MF(N,value,type_fuzzy);        //error is negative
}
void rule_base() {            //FLC have 5 rules.
 //rule 1: if( error is Z ) then ( voltage is oZ )
 if(uZ>0) {
  uoZ=uZ;
 } else {
  uoZ=0;
 }
 //rule 2: if( error is P ) then ( dvoltage is oN )
 if(uP>0) {
  uoN=uP;
 } else {
  uoN=0;
 }
 //rule 3: if( error is N ) then ( dvoltage is oP )
 if(uN>0) {
  uoP=uN;
 } else {
  uoP=0;
 }
}
/*void inference() {
 //not use
}*/
float defuzzifier(unsigned char type_fuzzy) {    //COA
 float output=0,sum_uoutput=0;
 float oP=0,oZ=0,oN=0;
 if(type_fuzzy==F_P) {
  oP=oP_p;
  oZ=oZ_p;
  oN=oN_p;
 } else if(type_fuzzy==F_I) {
  oP=oP_i;
  oZ=oZ_i;
  oN=oN_i;
 }
  output+=(oP*uoP);
 output+=(oZ*uoZ);
 output+=(oN*uoN);
 sum_uoutput=uoP+uoZ+uoN;
 return(output/sum_uoutput);
}
float min(float value1,float value2) {
 float value=0;
 if(value1<=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
float max(float value1,float value2) {
 float value=0;
 if(value1>=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
void clear_uMF() { //clear all previous data
  uP=0;  uZ=0;  uN=0;
  uoN=0;  uoZ=0; uoP=0;
}
float u_MF(signed char mf_con,float value,unsigned char type_fuzzy) {
 float temp_u_mf=0;
 float Llim_error=0,Ulim_error=0,m_error=0;

 if(type_fuzzy==F_P) {
  Llim_error=Llim_error_p;
  Ulim_error=Ulim_error_p;
  m_error=m_error_p;
 } else if(type_fuzzy==F_I) {
  Llim_error=Llim_error_i;
  Ulim_error=Ulim_error_i;
  m_error=m_error_i;
 }
 switch(mf_con) {
  case(P): { //error is positive
    if(value<=Llim_error) {
     temp_u_mf=1;
    } else if((value>Llim_error)&&(value<0)) {
     temp_u_mf=(-m_error)*value;
```

```
    } else {
     temp_u_mf=0;
    }
   } break;
    case(Z): {  //error is zero
    if((value>Llim_error)&&(value<0)) {
     temp_u_mf=((m_error)*value)+1;
    } else if(value==0){
     temp_u_mf=1;
    } else if((value>0)&&(value<Ulim_error)) {
     temp_u_mf=((-m_error)*value)+1;
    } else {
     temp_u_mf=0;
    }
   } break;
   case(N): {  //error is negative
    if((value>0)&&(value<Ulim_error)) {
     temp_u_mf=(m_error)*value;
    } else if(value>=Ulim_error){
     temp_u_mf=1;
    } else {
     temp_u_mf=0;
    }
   } break;
  }
  return(temp_u_mf);
}
float FLC(float error_value,unsigned char type_fuzzy) {
 clear_uMF();
 fuzzifer(error_value,type_fuzzy);
 rule_base();
 //inference();
 return(defuzzifier(type_fuzzy));
}
```

Library Program of Fuzzy PI controller

```
//C:\dsPIC\dspic_a00olib\motor\fuzzy_PI.h
//Fuzzy Control Speed module
//Fuzzy-PI controller (sampling every 10ms)
//I/P error and derror          >> speed(rpm)
//O/P duty                                  >> -1023 - 1023
//min-max speed motor     >> -100 - 100 rpm
#define error  0
#define derror 1
#define PB     -2
#define P      -1
#define Z      0
#define N      1
#define NB     2
#define dN     0
#define dZ     1
#define dP     2
//limit of graph degree of membership
//error
#define Llim_error  -8
#define Ulim_error   8
#define m_error              0.125
#define LBlim_error -35
#define UBlim_error  35
#define mB_error             0.0370        //slope error  = 1/(UBlim_error-Ulim_error) = 1/12
#define c1                                 -0.2963       //c=-(-mB_error*Llim_error)  PB,NB line
#define c2                                  1.2963       //c=-(mB_error*LBlim_error)            P,N   line
//derror
#define Llim_derror -20
#define Ulim_derror  20
#define m_derror             0.05
//membership function of output   dduty >> (-100 - 100)
#define oNB -235
#define oN  -13.3
#define oNS -11.5
#define oZ   0
#define oPS  11.5
#define oP   13.3
#define oPB  235
float uPB=0,uP=0,uZ=0,uN=0,uNB=0;  //degree of membership function of error
float udN=0,udZ=0,udP=0;                                        //degree of membership function of derror
float uoNB=0,uoN=0,uoNS=0,uoZ=0,uoPS=0,uoP=0,uoPB=0;      //degree of membership function of dvoltage
signed int error_speed=0,last_error_speed=0;
signed int duty=0;
void fuzzifer(signed char case_con,float value);
void rule_base(void);
//void inference(void);
float defuzzifier(void);
float min(float value1,float value2);
float max(float value1,float value2);
void clear_uMF(void);
float u_MF(signed char case_con,signed char mf_con,float value);
float FLC(signed int temp_error,signed int temp_derror);
void FLC_PI(signed int set_point);
//fuzzy-PI controller
void FLC_PI(signed int setpoint) {
 signed int duty_output=0;
 error_speed=setpoint-real_speed;
 duty=duty+(signed int)FLC(error_speed,(error_speed-last_error_speed));        //FLC(error,error-last_error)=dduty      >> (-100 - 100)
 //limit
 if(duty>30000) {                                  //upper limit
  duty=30000;
 } else if(duty<-30000) {  //lower limit
  duty=-30000;
 }
```

```
 //output limit
 if(duty>1023) {                                    . //upper output limit
  duty_output=1023;
 } else if(duty<-1023) {    //lower output limit
  duty_output=-1023;
 } else {
  duty_output=duty;
 }
 last_error_speed=error_speed;
 speed_output(3,duty_output);                       //set pwm duty
}
void fuzzifer(signed char case_con,float value) {
 switch(case_con) {
  case(error): {                    //input1 is  error
    uPB=u_MF(error,PB,value);           //error is positive big
    uP=u_MF(error,P,value);            //error is positive
    uZ=u_MF(error,Z,value);            //error is zero
    uN=u_MF(error,N,value);            //error is negative
    uNB=u_MF(error,NB,value);           //error is negative big
   } break;
  case(derror): {                   //input2 is derror
    udN=u_MF(derror,dN,value);          //derror is negative
    udZ=u_MF(derror,dZ,value);          //derror is zero
    udP=u_MF(derror,dP,value);          //derror is positive
   } break;
 }
}
void rule_base() {          //FLC have 7 rules.
 //rule 1: if( error is Z ) then ( dvoltage is oZ )
 if(uZ>0) {
  uoZ=uZ;
 } else {
  uoZ=0;
 }
 //rule 2: if( error is PB ) then ( dvoltage is oNB )
 if(uPB>0) {
  uoNB=uPB;
 } else {
  uoNB=0;
 }
  //rule 3: if( error is NB ) then ( dvoltage is oPB )
 if(uNB>0) {
  uoPB=uNB;
 } else {
  uoPB=0;
 }
 //rule 4: if( error is P ) then ( dvoltage is oN )
 if(uP>0) {
  uoN=uP;
 } else {
  uoN=0;
 }
  //rule 5: if( error is N ) then ( dvoltage is oP )
 if(uN>0) {
  uoP=uN;
 } else {
  uoP=0;
 }
 //rule 6: if( error is P ) and ( derror is dN ) then ( dvoltage is oNS )
 if((uP>0)&&(udN>0)) {
  uoNS=min(uP,udN);
 } else {
  uoNS=0;
 }
//rule 7: if( error is N ) and ( derror is dP ) then ( dvoltage is oPS )
 if((uN>0)&&(udP>0)) {
  uoPS=min(uN,udP);
 } else {
  uoPS=0;
 }
}
/*void inference() {
 //not use
}*/
float defuzzifier() {    //COA
 float output=0,sum_uoutput=0;
 output+=(oPB*uoPB);
 output+=(oP*uoP);
 output+=(oPS*uoPS);
 output+=(oZ*uoZ);
 output+=(oNS*uoNS);
 output+=(oN*uoN);
 output+=(oNB*uoNB);
 sum_uoutput=uoPB+uoP+uoPS+uoZ+uoNS+uoN+uoNB;
 return(output/sum_uoutput);
}
float min(float value1,float value2) {
 float value=0;
 if(value1<=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
float max(float value1,float value2) {
 float value=0;
 if(value1>=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
```

```
}
void clear_uMF() {
 //clear all previous data
 uPB=0; uP=0; uZ=0; uN=0; uNB=0;
 udN=0; udZ=0; udP=0;
 uoNB=0; uoN=0; uoNS=0; uoZ=0; uoPS=0; uoP=0; uoPB=0;
}
float u_MF(signed char case_con,signed char mf_con,float value) {
 float temp_u_mf=0;
 switch(case_con) {
  //input1 is error
  case(error): {
   switch(mf_con) {
    case(PB): {  //error is positive big
     if(value<=LBlim_error) {
      temp_u_mf=1;
     } else if((value>LBlim_error)&&(value<Llim_error)) {
      temp_u_mf=((-mB_error)*value)+c1;
     } else {
      temp_u_mf=0;
     }
    } break;
    case(P): {  //error is positive
     if((value>LBlim_error)&&(value<LBlim_error)) {
      temp_u_mf=((mB_error)*value)+c2;
     } else if(value==Llim_error){
      temp_u_mf=1;
     } else if((value>Llim_error)&&(value<0)) {
      temp_u_mf=(-m_error)*value;
     } else {
      temp_u_mf=0;
     }
    } break;
   case(Z): {  //error is zero
     if((value>Llim_error)&&(value<0)) {
      temp_u_mf=((m_error)*value)+1;
     } else if(value==0){
      temp_u_mf=1;
     } else if((value>0)&&(value<Ulim_error)) {
      temp_u_mf=((-m_error)*value)+1;
     } else {
      temp_u_mf=0;
     }
    } break;
    case(N): {  //error is negative
     if((value>0)&&(value<Ulim_error)) {
      temp_u_mf=(m_error)*value;
     } else if(value==Ulim_error){
      temp_u_mf=1;
     } else if((value>Ulim_error)&&(value<UBlim_error)) {
      temp_u_mf=((-mB_error)*value)+c2;
     } else {
      temp_u_mf=0;
     }
    } break;

    case(NB): {   //error is negative big
     if((value>Ulim_error)&&(value<UBlim_error)) {
      temp_u_mf=((mB_error)*value)+c1;
     } else if(value>=UBlim_error) {
      temp_u_mf=1;
     } else {
      temp_u_mf=0;
     }
    } break;
   }
  } break;
    //input2 is derror
  case(derror): {
   switch(mf_con) {
    case(dN): {   //derror is negative
     if(value<=-Llim_derror) {
      temp_u_mf=1;
     } else if((value>-Llim_derror)&&(value<0)) {
      temp_u_mf=(-m_derror)*value;
     } else {
      temp_u_mf=0;
     }
    } break;
    case(dZ): { //derror is zero
     if((value>-Llim_derror)&&(value<0)) {
      temp_u_mf=((m_derror)*value)+1;
     } else if(value==0){
      temp_u_mf=1;
     } else if((value>0)&&(value<Ulim_derror)) {
      temp_u_mf=((-m_derror)*value)+1;
     } else {
      temp_u_mf=0;
     }
    } break;
        case(dP): {  //derror is positive
     if(value>0)&&(value<Ulim_derror)) {
      temp_u_mf=(m_derror)*value;
     } else if(value>=Ulim_derror) {
      temp_u_mf=1;
     } else {
      temp_u_mf=0;   }
    } break;
   }
  } break;
 }
 return(temp_u_mf);
}
```

```
float FLC(signed int temp_error,signed int temp_derror) {
 clear_uMF();
 fuzzifer(error,temp_error);
 fuzzifer(derror,temp_derror);
 rule_base();
 //inference();
 return(defuzzifier());
}
```

Library Program of Fuzzy PI-I controller (Master)

```
//C:\dsPIC\dspic_o00olib\fuzzy_PI_I.h
//Fuzzy Control Speed module
//Fuzzy PI-I controller with compass (sampling every 20ms)
//I/P error and derror        >> speed(rpm)
//O/P duty                          >> -1023 - 1023
//min-max speed motor     >> -100 - 100 rpm
#define error  0
#define derror 1
#define PB    -2
#define P     -1
#define Z     0
#define N     1
#define NB    2
#define dN    0
#define dZ    1
#define dP    2
#define F_PI  0
#define F_I   1
//Fuzzy PI limit of graph degree of membership
//error
#define Llim_error_pi  -10
#define Ulim_error_pi   10
#define m_error_pi          0.1         //slope error  = 1/9.5
#define LBlim_error_pi -60
#define UBlim_error_pi  60
#define mB_error_pi         0.02        //slope error  = 1/(UBlim_error-Ulim_error) = 1/12
#define c1_pi                           -0.2        //c=-(-mB_error*Llim_error)  PB,NB line
#define c2_pi                           1.2         //c=-(mB_error*LBlim_error)          P,N   line
//derror
#define Llim_derror_pi -20
#define Ulim_derror_pi  20
#define m_derror_pi         0.05        //slope derror = 1/20
//membership function of output   dduty >> (-100 - 100)
#define oNB_pi -235
#define oN_pi  -15
#define oNS_pi -12
#define oZ_pi   0
#define oPS_pi  12
#define oP_pi   15
#define oPB_pi  235
//Fuzzy I limit of graph degree of membership
#define Llim_error_i  -30
#define Ulim_error_i   30
#define m_error_i           0.03333     //slope error  = 1/22
//membership function of output
#define oN_i  -3
#define oZ_i   0
#define oP_i   3

//variable of Fuzzy PI
float uPB_pi=0,uP_pi=0,uZ_pi=0,uN_pi=0,uNB_pi=0;                //degree of membership function of error
float udN_pi=0,udZ_pi=0,udP_pi=0;                               //degree of membership function of derror
float uoNB_pi=0,uoN_pi=0,uoNS_pi=0,uoZ_pi=0,uoPS_pi=0,uoP_pi=0,uoPB_pi=0;       //degree of membership function of dvoltage
//variable of Fuzzy I
float uP_i=0 ,uZ_i=0  ,uN_i=0;
float uoN_i=0 ,uoZ_i=0 ,uoP_i=0;
signed int last_fw_path_R=0,last_fw_path_L=0;       //fuzzy PI-I
signed int last_error_R=0,last_error_L=0;                       //fuzzy PI
signed int op_I=0,op_PI_R=0,op_PI_L=0;
void fuzzifer(signed char case_con,float value,unsigned char type_fuzzy);
void rule_base(unsigned char type_fuzzy);
//void inference(void);
float defuzzifier(unsigned char type_fuzzy);
float min(float value1,float value2);
float max(float value1,float value2);
void clear_uMF(unsigned char type_fuzzy);
float u_MF(signed char case_con,signed char mf_con,float value,unsigned char type_fuzzy);
float FLC(signed int temp_error,signed int temp_derror,unsigned char type_fuzzy);
void FLC_PI_I_cmp(signed int setpoint,signed char speed_R,signed char speed_L,unsigned int ang_st,unsigned int ang_now);
void FLC_PI_I(signed int setpoint,signed char speed_R,signed char speed_L);
void FLC_PI(signed int setpoint,signed char speed_R,signed char speed_L);
//fuzzy PI-I controller with cmp sensor
void FLC_PI_I_cmp(signed int setpoint,signed char speed_R,signed char speed_L,unsigned int ang_st,unsigned int ang_now) {
 signed int fw_path_L=0,fw_path_R=0;
 signed int curve_offset=0;
 unsigned char flag_ang=0;
 //calculate curve offset
 if(ang_now>ang_st) {
  flag_ang=0;
  curve_offset=ang_now-ang_st;
 } else {
  flag_ang=1;
  curve_offset=ang_st-ang_now;
 }
 if(curve_offset>180) {
  curve_offset=curve_offset-360;
 }
 if(flag_ang==1) {
```

```
 curve_offset*=-1;
}
op_I+=(signed int)FLC(speed_L-speed_R+curve_offset,0,F_I);
if(op_I>30000) {                                        //limit output I controller
 op_I=30000;
} else if(op_I<-30000) {
 op_I=-30000;
}

//Right motor path
fw_path_R=setpoint+op_I-speed_R;
if(fw_path_R>30000) {
 fw_path_R=30000;
} else if(fw_path_R<-30000) {
 fw_path_R=-30000;
}
//Left motor path
fw_path_L=setpoint-op_I-speed_L;
if(fw_path_L>30000) {
 fw_path_L=30000;
} else if(fw_path_L<-30000) {
 fw_path_L=-30000;
}
op_PI_R+=(signed int)FLC(fw_path_R,fw_path_R-last_fw_path_R,F_PI);
op_PI_L+=(signed int)FLC(fw_path_L,fw_path_L-last_fw_path_L,F_PI);
/*sprintf(uart_buf,"%d %d\r\n",op_PI_R,op_PI_L); //init show terminal
putsUART1((unsigned int *)uart_buf);
while(BusyUART1());*/
spi_send16_s(1,change_duty16,op_PI_R);          //R
spi_send16_s(2,change_duty16,op_PI_L);          //L
last_fw_path_R=fw_path_R;
last_fw_path_L=fw_path_L;
}
//fuzzy PI-I controller
void FLC_PI_I(signed int setpoint,signed char speed_R,signed char speed_L) {
 signed int fw_path_L=0,fw_path_R=0;
 op_I+=(signed int)FLC(speed_L-speed_R,0,F_I);
 if(op_I>30000) {                                       //limit output I controller
  op_I=30000;
 } else if(op_I<-30000) {
  op_I=-30000;
 }
 //Right motor path
 fw_path_R=setpoint+op_I-speed_R;
 if(fw_path_R>30000) {
  fw_path_R=30000;
 } else if(fw_path_R<-30000) {
  fw_path_R=-30000;
 }
 //Left motor path
 fw_path_L=setpoint-op_I-speed_L;
 if(fw_path_L>30000) {
  fw_path_L=30000;
 } else if(fw_path_L<-30000) {
  fw_path_L=-30000;
 }
 op_PI_R+=(signed int)FLC(fw_path_R,fw_path_R-last_fw_path_R,F_PI);
 op_PI_L+=(signed int)FLC(fw_path_L,fw_path_L-last_fw_path_L,F_PI);
 /*sprintf(uart_buf,"%d %d\r\n",op_PI_R,op_PI_L); //init show terminal
 putsUART1((unsigned int *)uart_buf);
 while(BusyUART1());*/
 spi_send16_s(1,change_duty16,op_PI_R);         //R
 spi_send16_s(2,change_duty16,op_PI_L);         //L

 last_fw_path_R=fw_path_R;
 last_fw_path_L=fw_path_L;
}
//fuzzy PI controller
void FLC_PI(signed int setpoint,signed char speed_R,signed char speed_L) {
 signed int error_R=0,error_L=0;
 error_R=setpoint-speed_R;
 error_L=setpoint-speed_L;
 op_PI_R+=(signed int)FLC(error_R,(error_R-last_error_R),F_PI);
 op_PI_L+=(signed int)FLC(error_L,(error_L-last_error_L),F_PI);
 /*sprintf(uart_buf,"%d %d\r\n",op_PI_R,op_PI_L); //init show terminal
 putsUART1((unsigned int *)uart_buf);
 while(BusyUART1());*/
 //limit
 if(op_PI_R>30000) {                                    //upper limit
  op_PI_R=30000;
 } else if(op_PI_R<-30000) {            //lower limit
  op_PI_R=-30000;
 }
 if(op_PI_L>30000) {                                    //upper limit
  op_PI_L=30000;
 } else if(op_PI_L<-30000) {            //lower limit
  op_PI_L=-30000;
 }
 spi_send16_s(1,change_duty16,op_PI_R);         //R
 spi_send16_s(2,change_duty16,op_PI_L);         //L
 last_error_R=error_R;
 last_error_L=error_L;
}
void fuzzifer(signed char case_con,float value,unsigned char type_fuzzy) {
 if(type_fuzzy==F_PI) {
  switch(case_con) {
   case(error): {                       //input1 is error
    uPB_pi=u_MF(error,PB,value,type_fuzzy);    //error is positive big
    uP_pi =u_MF(error,P,value ,type_fuzzy);    //error is positive
    uZ_pi =u_MF(error,Z,value ,type_fuzzy);    //error is zero
    uN_pi =u_MF(error,N,value ,type_fuzzy);    //error is negative
    uNB_pi=u_MF(error,NB,value,type_fuzzy);    //error is negative big
   } break;
```

```
    case(derror): {                         //input2 is derror
     udN_pi=u_MF(derror,dN,value,type_fuzzy);   //derror is negative
     udZ_pi=u_MF(derror,dZ,value,type_fuzzy);   //derror is zero
     udP_pi=u_MF(derror,dP,value,type_fuzzy);   //derror is positive
    } break;
   }
 } else if(type_fuzzy==F_I) {
   uP_i = u_MF(error,P ,value,F_I);     //error is positive
   uZ_i = u_MF(error,Z ,value,F_I);                 //error is zero
   uN_i = u_MF(error,N ,value,F_I);     //error is negative
 }
}

void rule_base(unsigned char type_fuzzy) {              //FLC have 7 rules.
 if(type_fuzzy==F_PI) {
  //rule 1: if( error is Z ) then ( dvoltage is oZ )
  if(uZ_pi>0) {
   uoZ_pi=uZ_pi;
  } else {
   uoZ_pi=0;
  }
  //rule 2: if( error is PB ) then ( dvoltage is oNB )
  if(uPB_pi>0) {
   uoNB_pi=uPB_pi;
  } else {
   uoNB_pi=0;
  }
  //rule 3: if( error is NB ) then ( dvoltage is oPB )
  if(uNB_pi>0) {
   uoPB_pi=uNB_pi;
  } else {
   uoPB_pi=0;
  }
  //rule 4: if( error is P ) then ( dvoltage is oN )
  if(uP_pi>0) {
   uoN_pi=uP_pi;
  } else {
   uoN_pi=0;
  }
  //rule 5: if( error is N ) then ( dvoltage is oP )
  if(uN_pi>0) {
   uoP_pi=uN_pi;
  } else {
   uoP_pi=0;
  }

  //rule 6: if( error is P ) and ( derror is dN ) then ( dvoltage is oNS )
  if((uP_pi>0)&&(udN_pi>0)) {
   uoNS_pi=min(uP_pi,udN_pi);
  } else {
   uoNS_pi=0;
  }
  //rule 7: if( error is N ) and ( derror is dP ) then ( dvoltage is oPS )
  if((uN_pi>0)&&(udP_pi>0)) {
   uoPS_pi=min(uN_pi,udP_pi);
  } else {
   uoPS_pi=0;
  }
 } else if(type_fuzzy==F_I) {
  //rule 1: if( error is Z ) then ( voltage is oZ )
  if(uZ_i>0) {
   uoZ_i=uZ_i;
  } else {
   uoZ_i=0;
  }
  //rule 2: if( error is P ) then ( dvoltage is oN )
  if(uP_i>0) {
   uoN_i=uP_i;
  } else {
   uoN_i=0;
  }
  //rule 3: if( error is N ) then ( dvoltage is oP )
  if(uN_i>0) {
   uoP_i=uN_i;
  } else {
   uoP_i=0;
  }
 }
}
/*void inference() {
 //not use
}*/
float defuzzifier(unsigned char type_fuzzy) {    //COA
 float output=0,sum_uoutput=0;
 if(type_fuzzy==F_PI) {
  output+=(oPB_pi*uoPB_pi);
  output+=(oP_pi*uoP_pi);
  output+=(oPS_pi*uoPS_pi);
  output+=(oZ_pi*uoZ_pi);
  output+=(oNS_pi*uoNS_pi);
  output+=(oN_pi*uoN_pi);
  output+=(oNB_pi*uoNB_pi);
  sum_uoutput=uoPB_pi+uoP_pi+uoPS_pi+uoZ_pi+uoNS_pi+uoN_pi+uoNB_pi;
 } else if(type_fuzzy==F_I) {
  output+=(oP_i*uoP_i);
  output+=(oZ_i*uoZ_i);
  output+=(oN_i*uoN_i);
  sum_uoutput=uoP_i+uoZ_i+uoN_i;
 }
 if(sum_uoutput==0) {
  return(0);                                        //protect error in math rule
 } else {
  return(output/sum_uoutput);
```

```
 }
}
float min(float value1,float value2) {
 float value=0;
 if(value1<=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
float max(float value1,float value2) {
 float value=0;
 if(value1>=value2) {
  value=value1;
 } else {
  value=value2;
 }
 return(value);
}
void clear_uMF(unsigned char type_fuzzy) {
 //clear all previous data
 if(type_fuzzy==F_PI) {
  uPB_pi=0; uP_pi=0; uZ_pi=0; uN_pi=0; uNB_pi=0;
  udN_pi=0; udZ_pi=0; udP_pi=0;
  uoNB_pi=0; uoN_pi=0; uoNS_pi=0; uoZ_pi=0; uoPS_pi=0; uoP_pi=0; uoPB_pi=0;
 } else if(type_fuzzy==F_I) {
  uP_i=0;  uZ_i=0;  uN_i=0;
  uoN_i=0;  uoZ_i=0; uoP_i=0;
 }
}
float u_MF(signed char case_con,signed char mf_con,float value,unsigned char type_fuzzy) {
 float temp_u_mf=0;
 if(type_fuzzy==F_PI) {
  switch(case_con) {
   //input1 is error
   case(error): {
    switch(mf_con) {
     case(PB): {  //error is positive big
      if(value<=LBlim_error_pi) {
       temp_u_mf=1;
      } else if((value>LBlim_error_pi)&&(value<Llim_error_pi)) {
       temp_u_mf=((-mB_error_pi)*value)+c1_pi;
      } else {
       temp_u_mf=0;
      }
     } break;
     case(P): {  //error is positive
      if((value>LBlim_error_pi)&&(value<LBlim_error_pi)) {
       temp_u_mf=((mB_error_pi)*value)+c2_pi;
      } else if(value==Llim_error_pi){
       temp_u_mf=1;
      } else if((value>Llim_error_pi)&&(value<0)) {
       temp_u_mf=(-m_error_pi)*value;
      } else {
       temp_u_mf=0;
      }
     } break;
         case(Z): {  //error is zero
      if((value>Llim_error_pi)&&(value<0)) {
       temp_u_mf=((m_error_pi)*value)+1;
      } else if(value==0){
       temp_u_mf=1;
      } else if((value>0)&&(value<Ulim_error_pi)) {
       temp_u_mf=((-m_error_pi)*value)+1;
      } else {
       temp_u_mf=0;
      }
     } break;
     case(N): {  //error is negative
      if((value>0)&&(value<Ulim_error_pi)) {
       temp_u_mf=(m_error_pi)*value;
      } else if(value==Ulim_error_pi){
       temp_u_mf=1;
      } else if((value>Ulim_error_pi)&&(value<UBlim_error_pi)) {
       temp_u_mf=((-mB_error_pi)*value)+c2_pi;
      } else {
       temp_u_mf=0;
      }
     } break;
     case(NB): {  //error is negative big
      if((value>Ulim_error_pi)&&(value<UBlim_error_pi)) {
       temp_u_mf=((mB_error_pi)*value)+c1_pi;
      } else if(value>=UBlim_error_pi) {
       temp_u_mf=1;
      } else {
       temp_u_mf=0;
      }
     } break;
    }
   } break;
    //input2 is deror
   case(derror): {
    switch(mf_con) {
     case(dN): {  //derror is negative
      if(value<=-Llim_derror_pi) {
       temp_u_mf=1;
      } else if((value>-Llim_derror_pi)&&(value<0)) {
       temp_u_mf=(-m_derror_pi)*value;
      } else {
       temp_u_mf=0;
      }
     } break;
```

```
      case(dZ): { //derror is zero
       if((value>-Llim_derror_pi)&&(value<0)) {
        temp_u_mf=((m_derror_pi)*value)+1;
       } else if(value==0){
        temp_u_mf=1;
       } else if((value>0)&&(value<Ulim_derror_pi)) {
        temp_u_mf=((-m_derror_pi)*value)+1;
       } else {
        temp_u_mf=0;
       }
      } break;
          case(dP): {  //derror is positive
       if((value>0)&&(value<Ulim_derror_pi)) {
        temp_u_mf=(m_derror_pi)*value;
       } else if(value>=Ulim_derror_pi) {
        temp_u_mf=1;
       } else {
        temp_u_mf=0;
       }
      } break;
     }
    } break;
   }
  } else if(type_fuzzy==F_I) {
   switch(mf_con) {
    case(P): {                                  //error is positive
     if(value<=Llim_error_i) {
      temp_u_mf=1;
     } else if((value>Llim_error_i)&&(value<0)) {
      temp_u_mf=(-m_error_i)*value;
     } else {
      temp_u_mf=0;
     }
    } break;
        case(Z): {                              //error is zero
     if((value>Llim_error_i)&&(value<0)) {
      temp_u_mf=((m_error_i)*value)+1;
     } else if(value==0){
      temp_u_mf=1;
     } else if((value>0)&&(value<Ulim_error_i)) {
      temp_u_mf=((-m_error_i)*value)+1;
     } else {
      temp_u_mf=0;
     }
    } break;

    case(N): {                                  //error is negative
     if((value>0)&&(value<Ulim_error_i)) {
      temp_u_mf=(m_error_i)*value;
     } else if(value>=Ulim_error_i){
      temp_u_mf=1;
     } else {
      temp_u_mf=0;
     }
    } break;
   }
  }

  return(temp_u_mf);
}

float FLC(signed int temp_error,signed int temp_derror,unsigned char type_fuzzy) {
 clear_uMF(type_fuzzy);
 if(type_fuzzy==F_PI) {
  fuzzifer(error,temp_error,F_PI);
  fuzzifer(derror,temp_derror,F_PI);
 } else if(type_fuzzy==F_I) {
  fuzzifer(error,temp_error,F_I);
 }
 rule_base(type_fuzzy);
 //inference();
 return(defuzzifier(type_fuzzy));
}
```