การควบคุมแฮลิคอปเตอร์ 4 ใบพัด

Quad copter control

พงศ์ภัค ทองวงศ์ตระกูล
PONGPAK TONGWONGTRAKUL

พงษ์มาศ ดีอันกอง
PONGMASH DEUNKONG

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2556

การควบคุมเฮลิคอปเตอร์ 4 ใบพัด

Quad Copter control

โดย

พงศ์ภัค ทองวงศ์ตระกูล

พงษ์มาศ ดีอันกอง

อาจารย์ที่ปรึกษา

อาจารย์ โกศล ชวนขยัน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2556

ปริญญานิพนธ์ ปีการศึกษา 2556

สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมเฮลิคอปเตอร์ 4 ใบพัด

Quad Copter control

ผู้จัดทำ นาย พงศ์ภัค ทองวงศ์ตระกูล รหัสนักศึกษา 53011033

นาย พงษ์มาศ ดีอันกอง รหัสนักศึกษา 53011039

ปริญญานิพนธ์นี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

(.....)

อาจารย์ที่ปรึกษา

หัวข้อปริญญานิพนธ์ การควบคุมเฮลิคอปเตอร์ 4 ใบพัด
นักศึกษา นาย พงศ์ภัค ทองวงศ์ตระกูล รหัสนักศึกษา 53011033
นาย พงษ์มาศ ดีอันกอง รหัสนักศึกษา 53011039
ปริญญา วิศวกรรมศาสตรบัณฑิต
สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์
ปีการศึกษา 2556
อาจารย์ที่ปรึกษาปริญญานิพนธ์ อาจารย์ โกศล ชวนขยัน

บทคัดย่อ

โครงการนี้เป็นการประยุกต์ใช้ระบบไมโครคอนโทรลเลอร์เพื่อควบคุมเครื่องบินสี่ใบพัดโดยอาศัยอินพุตจากเซนเซอร์ได้แก่ไจโรสโคป(Gyroscope) เป็นเซนเซอร์ในการวัดความเร็วเชิงมุมและตัววัดความเร่ง (Accelerometer) เป็นเซนเซอร์ในการวัดความเร่งเพื่อให้ไมโครคอนโทรลเลอร์ทำหน้าที่รักษาระดับการบินและควบคุมทิศทางการบินจากการประมวลผลข้อมูลที่ได้จากเซนเซอร์ไจโรสโคปตัววัดความเร่ง

Thesis Title	Quad Copter control	
Student	Mr. Pongpak Tongwongtrakul	53011033
	Mr. Pongmash Deeunkong	53011039
Degree	Bachelor of Engineering	
Program	Electronics Engineering	
Year	2013	
Thesis Advisor	Mr. Kosol chuankayun	

Abstract

This project applied the microcontroller system to control the Quad Copter by using input from sensors which are Gyroscope sensor and Accelerometer. Gyroscope sensor is the sensor used to measure angular velocity and accelerometer sensor is the sensor that measure acceleration. Microcontroller keeps balancing and control Quad Copter by calculating data from gyroscope and accelerometer.

กิตติกรรมประกาศ

ปริญญานิพนธ์ เรื่องการควบคุมเฮลิคอปเตอร์ 4 ใบพัด สำเร็จได้เป็นอย่างดีด้วยคำแนะนำและคำปรึกษาจากอาจารย์ โกศล ชวนขยัน (อาจารย์ที่ปรึกษา) และ อาจารย์พลผดุง ผดุงกุล ที่ชี้แนะแนวทางที่เป็นประโยชน์ต่อการดำเนินการทำปริญญานิพนธ์ กลุ่มของข้าพเจ้ารู้สึกทราบซึ้งในความกรุณาเป็นอย่างยิ่ง และขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุกท่านที่ได้รับประสิทธิ์ประสาทวิชาความรู้ต่างๆให้คำแนะนำและคำปรึกษาอย่างดีมาโดยตลอด

พงศ์ภัค ทองวงศ์ตระกูล

พงษ์มาศ ดีอันกอง

สารบัญ

สารบัญรูป

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญ

ปัจจุบันการพัฒนาเครื่องบินที่สามารถบินนิ่งในอากาศถูกพัฒนาขึ้นอย่างแพร่หลายซึ่งมีรูปแบบในการพัฒนาที่แตกต่างกัน โดยใช้จำนวนใบพัดตั้งแต่หนึ่งใบพัดไปจนถึงแปดใบพัด รูปแบบของเครื่องบินมีผลต่อการควบคุมและการเคลื่อนที่ในอากาศ ในการบินจำเป็นต้องอาศัยระบบไมโครคอนโทรลเลอร์ประมวลผลเพื่อควบคุมการทำงานจึงเป็นการประยุกต์ใช้ระบบไมโครคอนโทรลเลอร์ออกแบบและสร้างเครื่องบินเพื่อให้ทราบถึงหลักการทำงานของเครื่องบินและทำการศึกษาปัจจัยต่าง ๆ ที่มีผลต่อการบิน เพื่อนำไปพัฒนาต่อยอดเป็นเครื่องบินที่มีความสามารถต่าง ๆ เช่น UAV (Unmanned Aerial Vehicle) ที่สามารถบินสำรวจพื้นที่ หรือเครื่องบินที่สามารถถ่ายภาพทางอากาศได้ต่อไป

1.2 วัตถุประสงค์

1.เพื่อออกแบบและสร้าง Quad copter ที่มีความสมดุล โดยควบคุมผ่านระบบไมโครคอนโทรลเลอร์

2.เพื่อประยุกต์ใช้งานเซนเซอร์ ได้แก่ ไจโรสโคป (Gyroscope) และ ตัววัดความเร่ง (Accelerometer)

3. เพื่อศึกษาและออกแบบระบบควบคุม PID (Proportional-Integral-Derivative)

4. เพื่อศึกษาและใช้งาน Kalman filter

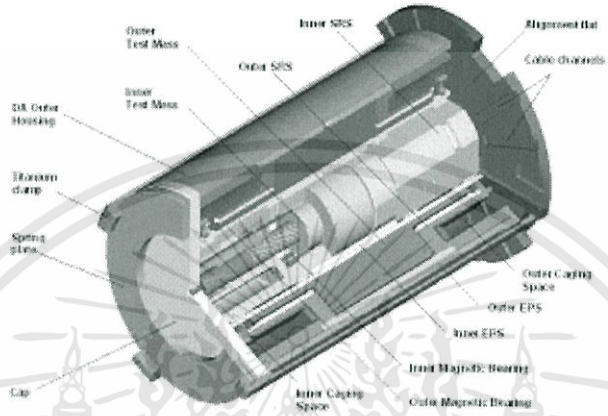
5.เพื่อศึกษาการทำงานของ Microcontroller

1.3 ขอบเขตของโครงงาน

ออกแบบและสร้างอากาศยานเฮลิคอปเตอร์สี่ใบพัดเพื่อให้สามารถบินและทรงตัวอยู่ได้กลางอากาศ โดยใช้ Kalman filter มาเพื่อกรองข้อมูลที่ได้จากเซ็นเซอร์ แล้วนำไปเข้าระบบ PID เพื่อควบคุมความเร็วของมอเตอร์Blush less การควบคุมมอเตอร์Blush lessนั้นควบคุมโดยการสร้างสัญญาณ PWM (*Pulse-width modulation*)ส่งไปยัง ESC(*Electronic speed control*) เพื่อไปหมุนมอเตอร์Blush less

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 Accelerometer



รูปที่ 2.1 โครงสร้าง Accelerometer

Accelerometer คือ เครื่องตัววัดความเร่ง ของการเคลื่อนที่ของวัตถุ

โครงสร้างของ accelerometer จะประกอบด้วยสปริงและลูกตุ้มน้ำหนัก เมื่อมีการเคลื่อนที่ด้วยความเร่งลูกตุ้มน้ำหนักจะถูกกดไปอีกฝั่งตรงข้ามกับการเคลื่อนที่ สปริงก็ทำหน้าที่ดึงกลับเข้าที่อีกครั้งเมื่อหยุดการเคลื่อนที่ การเคลื่อนที่ด้วยความเร็วคงที่คือความเร่งเท่ากับศูนย์ ค่าที่วัดได้ก็จะไม่เปลี่ยนแปลง

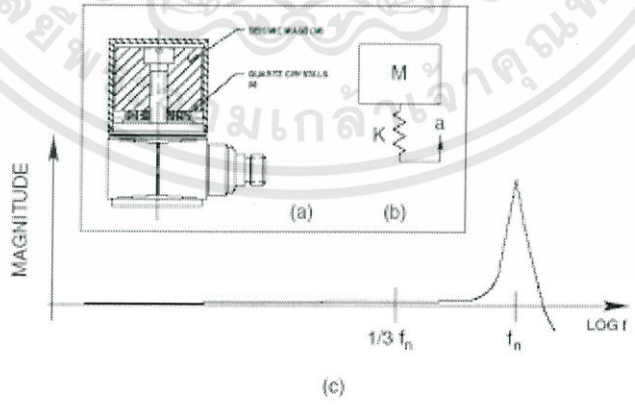


Figure 2: The accelerometer as a spring-mass system

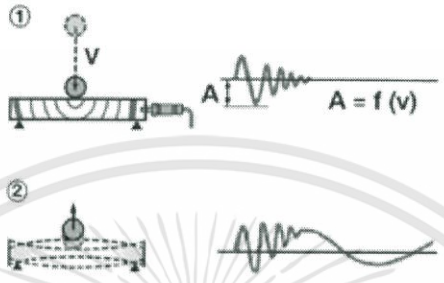
รูปที่ 2.2 ระบบภายใน Accelerometer

ส่วนตัวเซ็นเซอร์ภายใน ที่จะใช้ในการตรวจวัดความเร่งของลูกตุ้มที่อยู่ในระบบนั้นมีหลายชนิด เช่น เพียโซอิเล็กทริค, สเตรนเกจ, ชนิดใช้แสงตรวจวัด, วัดแรงเฉือน เป็นต้น โดยที่สามารถแบ่งลักษณะการตรวจวัดได้ 2 ลักษณะ

1. การตรวจวัดการช็อก (shock) และการสั่นสะเทือน (vibration) ซึ่ง

*การช็อก คือ อัตราเร่งขนาดมหาศาลที่เกิดขึ้นในช่วงเวลาสั้นๆ

*การสั่นสะเทือน คือ อัตราเร่งขนาดเล็กที่เกิดขึ้นซ้ำกันไปเรื่อยๆ



รูปที่ 2.3 การทำงานของ Accelerometer

2. การตรวจวัดอัตราเร่งของวัตถุ เพื่อนำข้อมูลไปใช้ในการระบุตำแหน่ง ความเร็ว และระยะทางที่ได้จากการเคลื่อนที่

มิเตอร์วัดความเร่งนี้โดยหลักๆแล้วจะแบ่งเป็น 2 ชนิด

- มิเตอร์วัดอัตราเร่งแบบไซซมิกแมส (seismic mass accelerometer)มิเตอร์ชนิดนี้อาศัยหลักการตรวจวัดระยะขจัดเชิงเส้นแล้วนำไปคำนวณหาอัตราเร่งที่เกิดขึ้นโดยเทคนิคดังกล่าวสามารถอธิบายง่ายๆ ได้ก็คือ วัตถุชิ้นหนึ่งจะมีความเร่งได้ ก็จะต้องมีแรงมากระทำยิ่งมีแรงมากระทำมาก ก็จะยิ่งมีความเร่งมาก ในขณะเดียวกันแรงต้านการเคลื่อนที่ก็จะมากด้วย นอกจากนี้เมื่อมีแรงมาทำให้วัตถุเกิดการเคลื่อนที่ ก็จะมีระยะขจัด ซึ่งก็จะแปรผันตรงกับแรงที่มากระทำที่วัตถุ ยิ่งแรงมากระยะขจัดยิ่งมาก จากความสัมพันธ์ดังกล่าวได้นำไปใช้เป็นหลักการพื้นฐานของมิเตอร์วัดอัตราเร่งแบบไซซมิกแมสในการตรวจวัดอัตราเร่งของวัตถุในเทอมของระยะขจัดที่เกิดขึ้น

$$F = ma = k\,dx$$

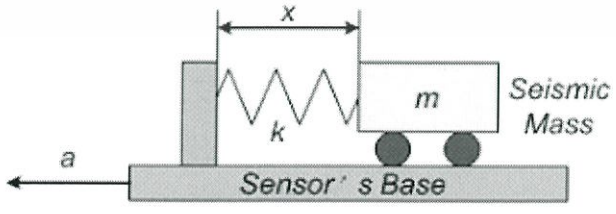


Fig-1 A spring-mass system

รูปที่ 2.4 โครงสร้างพื้นฐานของมิเตอร์วัดอัตราเร่งแบบไซซมิกแมส

จากรูป แสดงโครงสร้างพื้นฐานของมิเตอร์วัดอัตราเร่งแบบไซซมิกแมส โครงสร้างนี้มีมวล m ที่เรียกว่ามวลตรวจการสั่นไหว (seismic mass) ยึดติดอยู่กับสปริงที่มีค่า spring constant เท่ากับ k และมวลนี้สามารถเคลื่อนที่ในแนวระดับได้
ซึ่งหลักการทำงานก็ง่ายๆ ไม่ได้ซับซ้อนอะไร

เมื่อตัวเซนเซอร์ตัวนี้ถูกทำให้มีอัตราเร่งเกิดขึ้นจะส่งผลให้มวล m เคลื่อนที่ ซึ่งระยะที่เคลื่อนที่ออกไปจะเป็นระยะขจัดเท่ากับ x และมีทิศทางตรงกันข้ามกับการเคลื่อนที่ของตัวมิเตอร์ ดังนั้นอัตราเร่ง a ของวัตถุสามารถคำนวณหาค่าได้จากความสัมพันธ์ต่อไปนี้

$a = xk/m$ โดยที่

a คือ อัตราเร่งของวัตถุ หน่วย เมตร/วินาที

x คือ ระยะขจัดของมวล m หน่วย เมตร

k คือ ค่าคงที่ของสปริง หน่วย นิวตัน/เมตร

m คือ น้ำหนักของมวล m หน่วย กิโลกรัม

จากสมการดังกล่าวจะแสดงให้เห็นว่า

-เมื่ออัตราเร่งของวัตถุมีค่าเพิ่มขึ้น ทำให้ระยะขจัดของมวล m มีค่าเพิ่มขึ้นตามไปด้วย

-เมื่ออัตราเร่งของวัตถุม่ค่าลดลง ทำให้มวล m เคลื่อนที่ไปดันสปริง

-เมื่ออัตราเร่งของวัตถุหยุดลง ก็จะทำให้มวล m เคลื่อนที่กลับมาอยู่ตำแหน่งเดิม (ตำแหน่งอ้างอิง)

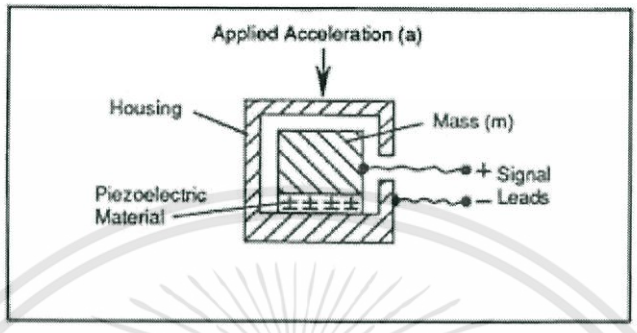
แต่ในทางปฏิบัติเราสามารถวัดระยะขจัดของมวล m ได้โดยอาศัยมิเตอร์อีกชนิดหนึ่ง คือมิเตอร์วัดระยะขจัดเชิงเส้น (LVDT, potentiometer)

ส่วนการวิเคราะห์หาค่าอัตราเร่งที่เกิดขึ้นเราสามารถคำนวณหาได้โดยใช้คอมพิวเตอร์ มิเตอร์วัดอัตราเร่งแบบไซซมิกแมสนี้จะนิยมใช้ในการตรวจวัดลักษณะการช็อกและลักษณะการสั่นสะเทือนที่มีความถี่ต่ำมากๆ เช่น ในเครื่องมือตรวจวัดแผ่นดินไหว หรือในเครื่องมือตรวจวัดการปะทุใต้ดินของภูเขาไฟ ฯลฯ

- มิเตอร์วัดอัตราเร่งแบบเพียโซอิเล็กทริก (piezoelectric accelerometer)

คุณสมบัติพื้นฐานทางไฟฟ้าของผลึกเพียโซอิเล็กทริก (piezoelectric crystal) ถูกค้นพบโดย Pierre และ Jacques Curie ในราวปี ค.ศ.1880 ซึ่งเจ้า piezoelectric crystal นี้มันมีคุณสมบัติพิเศษ คือ เมื่อมันถูกแรงทางกลมากระทำ มันจะสร้างประจุไฟฟ้าขึ้นมา โดยเป็นสัดส่วนกับแรงกระทำนั้น ซึ่งจากคุณสมบัติพิเศษนี้ได้ถูกดัดแปลงนำไปใช้สร้างอุปกรณ์ต่างๆมากมาย เช่น ใช้เป็นแบตเตอรี่จ่ายพลังงานไฟฟ้าให้กับนาฬิกาข้อมือดิจิตอลที่เราใช้ทั่วไป และยังใช้สร้างมิเตอร์วัดอัตราเร่งแบบเพียโซอิเล็กทริกอีกด้วย

โครงสร้างของมอเตอร์วัดอัตราเร่งแบบเพียโซอิเล็กทริกจะประกอบด้วย seismic mass ยึดติดกับ piezoelectric crystal และบรรจุอยู่ในตัวถังป้องกัน โดย piezoelectric crystal ที่นิยมนำมาใช้งาน ได้แก่ ผลึกควอตซ์ และผลึกโซเดียมโปตัสเซียมตาเตรต (sodium potassium tartrate) เพราะมีความทนทานต่อแรงกระทำ และราคาไม่แพงมากนัก



รูปที่ 2.5 โครงสร้างพื้นฐานของมิเตอร์วัดอัตราเร่งแบบเพียโซอิเล็กทริก

จากรูปแสดงโครงสร้างพื้นฐานของมิเตอร์วัดอัตราเร่งแบบเพียโซอิเล็กทริก (piezoelectric accelerometer) สามารถอธิบายการทำงานง่ายๆ ได้ดังนี้

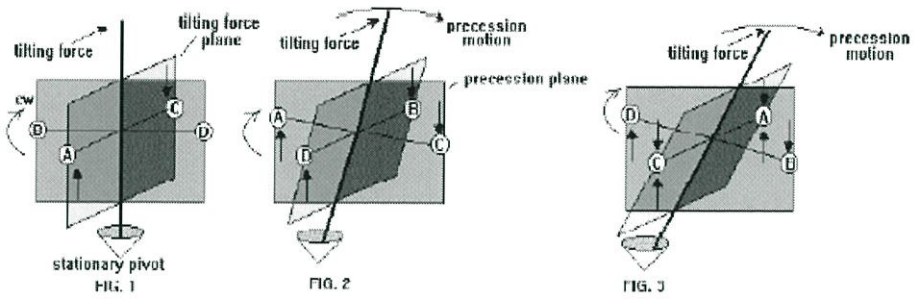
เมื่อ seismic mass (m) ถูกทำให้เกิดอัตราเร่งขึ้น (ถูกกด) มันจะส่งผ่านแรงกดไปกระทำกับ piezoelectric crystal ที่ถูกยึดติดอยู่ด้วยกัน ด้วยคุณสมบัติพิเศษของมันจะทำให้ประจุไฟฟ้าถูกสร้างขึ้น และถูกสายนำสัญญาณออกไปยังเอาต์พุตของวงจร โดยที่ด้านเอาต์พุตจะต้องมีวงจรขยายประจุไฟฟ้า (charge amplifier) เพื่อขยายค่าประจุไฟฟ้าที่ได้ให้เป็นแรงดันเอาต์พุตตามสัดส่วนของอัตราเร่งที่เกิด จะได้สามารถแสดงผลได้ด้วยโวลต์มิเตอร์

2.2 Gyroscope

ไจโรสโคปเป็นเซนเซอร์วัดอัตราการเปลี่ยนแปลงของมุม ซึ่งในงานนี้จะใช้ไจโรสโคปเพื่อวัดอัตราการเปลี่ยนแปลงของมุม ทั้งหมด 3 แกน ได้แก่แกน X Y และ Z

หลักการทำงานของGyroscope

จากรูปเราจะแทนที่ขอบด้วย A,B,C,D ทั้งแถบในการดูว่า Gyroscope ทำงานอย่างไร จุดล่างสุดเป็นแกนคงที่แต่สามารถหมุนได้รอบทิศทาง



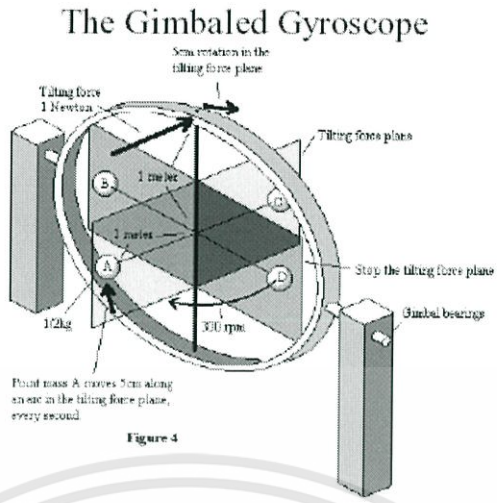
รูปที่ 2.6 หลักการทำงานของ Gyroscope

เมื่อแรงกระทำ(tilting force)ที่ส่วนบนของแกน GyroscopeจุดAจะเคลื่อนที่ขึ้นตามแนวตั้ง จุดCเลื่อนลงตามแนวนอนพร้อมกัน A และ B หมุนไป90^0 เช่นเดียวกับที่เกิดกับ C และ D โดยที่ A นั้นยังคงเลื่อนขึ้นในตำแหน่ง90^0 และC เลื่อนลง ผลของการเลื่อนของ A และ C ทำให้แกนของ Gyro หมุนตามการกระทำของ precession plane เรียกการเกิดขึ้นของลักษณะนี้ว่า precession แกนของ Gyroจะหมุนไปทางมุมขวาเนื่องจากการหมุน ถ้าGyroถูกทำให้หมุนทวนเข็มนาฬิกามันก็จะไปทางมุมซ้าย หมายความว่าแรงกระทำตอนต้นเป็นการดึง

เมื่อGyroหมุนไปอีก90^0ตาม Fig.3 จุด C จะมาแทนจุด A ในจุดที่แรงกระทำไปแล้วครั้งแรก การเคลื่อนที่ลงของจุด C จะถูกต้านโดย tilting force ทำให้แกนของGyro ไม่เปลี่ยนแปลง ยิ่งมีแรง tilting กระทำมากขึ้นแกนของ Gyro ก็จะมีแรงดีดกลับมามากเมื่อขอบของ precession plane อยู่ที่ 180^0

จากข้างต้นทำให้ทราบว่าการหมุนของแกน Gyro เนื่องจากแรงจุด A และ C เคลื่อนที่ขึ้นลง ฉะนั้นเมื่อหมุน Gyro ในทิศทางตรงกันข้ามกับข้างต้นจะเกิดแรงเคลื่อนที่ขึ้นลงมากขึ้น

บางครั้งการเกิดขึ้นของ precession เป็นสิ่งที่ไม่ต้องการจึงมักมีการสร้าง Gyro แบบมีแกนที่เรียกว่า Gimbaled Gyroscope ดังรูป ซึ่งเป็น Gyro พื้นฐานถูกติดตั้งไว้ในระนาบที่ตั้งฉากกับแนวแรง เมื่อหมุนขอบไปทางระนาบ Gimbal พลังงานจะถ่ายเทไปยังขอบโดย tilting force เพื่อหยุดการกระทำนั้น ขอบก็จะหมุนกลับในทิศทางระนาบของแรง tilting แต่ละครั้งที่ Gyro ถูกกระตุ้นแกนจะหมุนไปตามส่วนโค้งในระนาบของแรง tilting โดยที่ไม่มีการเปลี่ยนความเร็วรอบการหมุนของขอบรอบ ๆ แกน

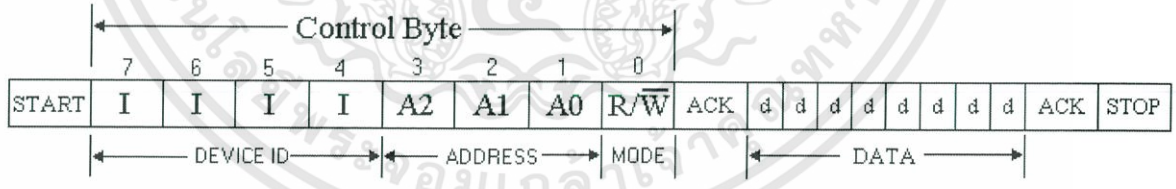


รูปที่ 2.7 Gyroscope

2.3 การเชื่อมต่ออุปกรณ์แบบ I^2C

I^2C ย่อมาจาก Inter Integrate Circuit Bus (IIC) นิยมเรียกสั้นๆว่า I^2C เป็นการสื่อสารอนุกรม แบบซิงโครนัส (Synchronous) เพื่อใช้ ติดต่อสื่อสาร ระหว่าง ไมโครคอนโทรลเลอร์ (MCU) กับอุปกรณ์ภายนอก ซึ่งถูกพัฒนาขึ้นโดยบริษัท Philips Semiconductors โดยใช้สายสัญญาณเพียง 2 เส้นเท่านั้น คือ serial data (SDA) และสาย serial clock (SCL) ซึ่งสามารถ เชื่อมต่ออุปกรณ์จำนวนหลายๆ ตัว เข้าด้วยกันได้ ทำให้ MCU ใช้พอร์ตเพียง 2 พอร์ตเท่านั้น

การเขียน-อ่านข้อมูลกับอุปกรณ์แบบ I^2C Bus



รูปที่2.8 รูปแบบการเขียน/อ่านข้อมูลแบบ I^2C BUS

การรับ-ส่งข้อมูลแบบ I^2C BUS MCU จะเริ่มต้นการส่งข้อมูลด้วยการ

-ส่งสถานะเริ่มต้น (START Conditions) เพื่อแสดงการขอใช้บัส

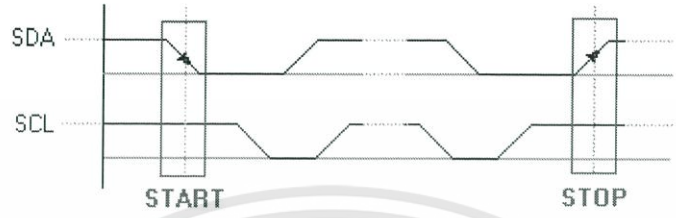
-แล้วตามด้วย รหัสควบคุม (Control Byte) ซึ่งประกอบ ด้วยรหัส ประจำตัวอุปกรณ์ Device ID, Device Address และ Mode ในการเขียนหรืออ่านข้อมูล

- เมื่ออุปกรณ์ รับทราบว่า MCU ต้องการ จะติดต่อด้วยก็ต้องส่งสถานะรับรู้ (Acknowledge) หรือแจ้งให้ MCU รับรู้ว่าข้อมูลที่ได้ส่งมามีความถูกต้อง

- และเมื่อสิ้นสุดการส่งข้อมูล MCU จะต้องส่ง สถานะสิ้นสุด (STOP Conditions) เพื่อบอกกับอุปกรณ์ว่า สิ้นสุดการใช้บัส

สถานะบัสว่าง คือเมื่อบัสไม่ได้ถูกใช้งาน ทั้ง SCL และ SDA จะเป็น 1 ทั้งคู่

การกำหนดสถานะเริ่มต้นและสถานะสิ้นสุดของ I^2C BUS (START and STOP Conditions)

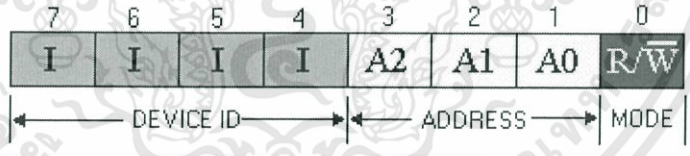


รูปที่ 2.9 I^2C BUS START and STOP Conditions

- เมื่อต้องการส่งข้อมูล MCU จะต้องส่งสถานะเริ่มต้น (START Conditions) คือให้ SDA เปลี่ยนจาก 1 มาเป็น 0 ในขณะที่ SCL มีค่าเป็น 1

- เมื่อสิ้นสุดการการใช้บัส MCU จะต้องส่งสถานะสิ้นสุด (STOP Conditions) คือให้ SDA เปลี่ยนจาก 0 มาเป็น 1ในขณะที่ SCL มีค่าเป็น 1

รหัสควบคุมของ I^2C BUS (Control Byte)



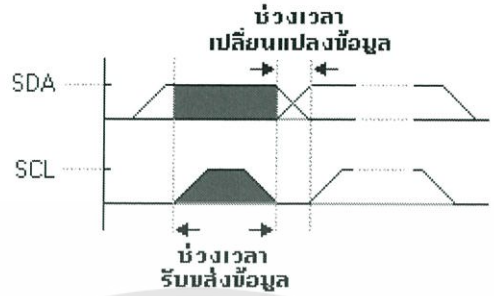
รูปที่ 2.10 I^2C BUS (Control Byte)

รหัสควบคุมของ I^2C BUS ประกอบด้วยรหัสประจำตัวของอุปกรณ์ (Device ID) ประกอบด้วยบิต 1-7 และบิต 0 เป็นบิตควบคุมการเขียนอ่าน

- รหัสประจำตัวของอุปกรณ์ ประกอบด้วยรหัสประจำตัวจากผู้ผลิต Product ID 4 บิต (บิต 4-7) ที่เปลี่ยนแปลงแก้ไขไม่ได้ และ Device Address 3 บิต (บิต 1-3) ซึ่งผู้ใช้ สามารถ กำหนดเองได้ รวมแล้วเป็นรหัส 7 บิท ใช้ระบุตัวอุปกรณ์ ที่ต่ออยู่บนบัส จะมีค่าซ้ำกันไม่ได้

- บิตควบคุมการเขียนอ่าน (Mode) บิต 0 เมื่อ MCU ต้องการเขียนข้อมูลไปยังอุปกรณ์ก็กำหนดให้บิตนี้เป็น 0 และเมื่อต้องการ อ่านข้อมูล จากอุปกรณ์ ก็กำหนดให้บิตนี้เป็น 1

ช่วงเวลารับส่งบิตข้อมูลของ I^2C BUS



รูปที่ 2.11 การรับส่งบิตข้อมูลของ I^2C BUS

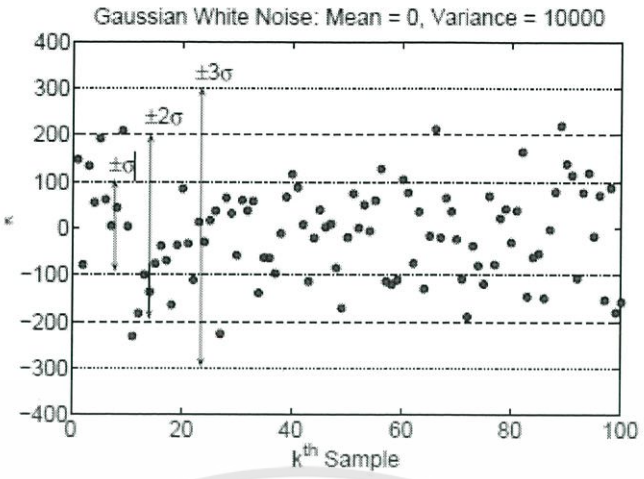
- สะภาวะการรับ-ส่งข้อมูล จะกระทำในขณะที่ขา SCL เป็น 1
- สะภาวะการเปลี่ยนแปลงข้อมูล จะกระทำในขณะที่ขา SCL เป็น 0

2.4 KALMAN FILTER ALGORITHM

ให้ x_k เป็นตัวแปรใดๆ ณ เวลา t_k ที่เราไม่รู้ค่าแต่ต้องการประมาณค่า ณ เวลา $t_0, t_1, \ldots, t_k$ ไปเรื่อยๆ สมมุติว่า x_k เปลี่ยนแปลงตามเวลาตามสมการ

$$x_k = \Phi x_{k-1} + w_k \qquad (1)$$

สมการที่ (1) คือแบบจำลองทางคณิตศาสตร์ของระบบเรียกว่า Plantหรือ Process Model เพราะแสดงความสัมพันธ์ว่า *x* เปลี่ยนแปลงตามเวลาอย่างไร ให้ w_k เป็น Gaussian White Noise มีค่าเฉลี่ย(Mean) เท่ากับ μ และ Variance เท่ากับ *Q* (ดังนั้น StandardDeviation ของ w_k คือ $\sigma = \sqrt{Q}$) โดยปกติเราจะไม่สามารถบอกด้วยความมั่นใจ 100% ได้ว่าตัวแปรสุ่มจะมีค่าเป็นเท่าไร แต่เนื่องจากw_k เป็นตัวแปรสุ่มแบบ Gaussian เราจึงสามารถใช้ค่า Mean และ Variance อธิบายเกี่ยวกับ w_k ในเชิงความน่าจะเป็นได้อย่างสมบูรณ์ว่าถ้าสุ่มหลายๆ ครั้ง *โดยเฉลี่ย* w_k จะมีค่าเท่ากับ μ นอกจากนี้ประมาณ 68%, 95% และ 99% ของค่าของ w_k จากการสุ่มทั้งหมดจะอยู่ในช่วง $\mu \pm \sigma$, $\mu \pm 2\sigma$ และ $\mu \pm 3\sigma$ ตามลำ ดับ สมมุติให้μ = 0 และ *Q* =10000 ภาพที่ 1 แสดงผลของการสุ่มตัวอย่าง w_k 100 ครั้ง



รูปที่ 2.12 ตัวอย่างการสุ่มตัวแปร w_k

ภาพที่ 1 ตัวอย่างการสุ่มตัวแปร w_k จะสังเกตได้ว่าประมาณ 68%, 95%และ 99% ของ w_k จะอยู่ในช่วง $\pm\sigma$, $\pm2\sigma$ และ $\pm3\sigma$ ตามลำดับ(μ = 0)White Noise หมายความว่า w_k *ไม่* เกี่ยวข้องกับตัวแปรอื่นใดหรือตัวมันเองโดยสิ้นเชิง(Uncorrelated) ดังนั้นแม้ว่าเราจะรู้ค่าของ *w* ในอดีตหรือค่าของตัวแปรอื่นใด ก็ไม่สามารถใช้ข้อมูลนั้นทำนายค่าของ w_k ในอนาคต (2) ถึง (4) สรุปคุณสมบัติของ w_k ด้วยสูตรทางคณิตศาสตร์ $E[\cdot]$ หมายถึงค่าที่เราคาดว่าตัวแปรจะเป็น (Expected Value)

โดย w_k และ v_k คือ Gaussian White Noise

Mean ของ w_k $$E[w_k] = 0 \tag{2}$$

Variance ของ w_k $$E[w_k w_j] = \begin{cases} Q & for\ i = j \\ 0 & for\ i \neq j \end{cases} \tag{3}$$

Covariance ของ w_k และ v_k $$E[w_k, v_k] = 0 \tag{4}$$

เนื่องจาก w_k เป็นตัวแปรสุ่มแบบ Gaussian และ (1) เป็นสมการเชิงเส้น (Linear) จึงสรุปได้ว่า x_kเป็นตัวแปรสุ่มแบบ Gaussianก่อนใช้งาน Kalman Filter เราจำเป็นต้องให้ค่าประมาณของ *x* และค่าความคลาดเคลื่อนของการประมาณกับ Kalman Filter เพื่อเริ่มต้นกำหนดให้ความคลาดเคลื่อน $e = x - \hat{x}$ (x และ $\hat{x} = E[x]$ คือค่าที่แท้จริงและค่าประมาณของ *x* ตามลำดับ) เนื่องจาก *x* เป็นตัวแปรสุ่มแบบ Gaussian e จึงเป็นตัวแปรสุ่มแบบ Gaussian (ดังนั้นเพื่ออธิบาย *e* ในเชิงความน่าจะเป็นอย่างสมบูรณ์เราต้องรู้ค่า Mean และVariance ของ e)

สำหรับตัวอย่างนี้ สมมุติให้ Φ = 0.8 และสมมุติว่าจากแหล่งข้อมูลที่ดีที่สุดที่มี ทำให้เชื่อได้ว่า ณ เวลาเริ่มต้น *x* มีค่าเท่ากับ 100 และความคลาดเคลื่อนของการประมาณนี้มีค่า Mean $e_0 = 0$ และVariance (P_0) เท่ากับ 40000 ดังนั้น เราสามารถเขียนได้ว่า

$$\hat{x}_0 = E[x_0] = 100 \tag{5}$$

$$P_0 = \mathrm{E}[(x - \hat{x}_0)^2] = \mathrm{E}[(e_0)^2] = 40000 \tag{6}$$

ถ้าเรารู้ว่า w_1 มีค่าเป็นเท่าไรอย่างแน่นอน ก็จะสามารถคาดการณ์ล่วงหน้า (Predict) จาก (1) ได้ว่า X ณ เวลา t_1 จะมีค่าเป็นเท่าไรจาก

$$x_1 = \Phi x_0 + w_1 \tag{7}$$

แต่เนื่องจาก w_k เป็นตัวแปรสุ่มเราจึงไม่มีทางรู้ว่า w_1 จะมีค่าเป็นเท่าไรอย่างแน่นอน Kalman Filter Algorithm บอกเราว่าสามารถหาค่าประมาณที่ดีที่สุดของ x_1 ได้จาก

$$\hat{x}_1^- = \mathrm{E}[x_1] = \mathrm{E}[\Phi x_0 + w_1] = \Phi \mathrm{E}[x_0] + \mathrm{E}[w_1] = \Phi \hat{x}_0 \tag{8}$$

เครื่องหมายลบใช้ระบุว่า $\hat{x}_1^-$ เป็นการคาดการณ์ล่วงหน้าถึงค่าของ x_1 โดย*ไม่มี*ค่าที่เซนเซอร์วัดได้มาประกอบการพิจารณา เราสามารถหาค่า Mean และ Variance ของความคลาดเคลื่อน (Error Variance)ในการคาดการณ์นี้ได้จาก

$$\begin{aligned} \mathrm{E}[e_1^-] &= \mathrm{E}[x_1 - \hat{x}_1^-] = \mathrm{E}[\Phi x_0 + w_1 - \Phi \hat{x}_0] \\ &= \mathrm{E}[\Phi(x_0 - \hat{x}_0) + w_1] = \Phi E[e_0] + E[w_1] = 0 \end{aligned} \tag{9}$$

$$\begin{aligned} P_1^- &= \mathrm{E}[(x_1 - \hat{x}_1^-)^2] = \mathrm{E}[(e_1^-)^2] = \mathrm{E}[(\Phi e_0 + w_1)^2] \\ &= \mathrm{E}[(\Phi e_0)^2 + 2\Phi e_0 w_1 + w_1^2] \\ &= \Phi^2 \mathrm{E}[(e_0)^2] + 2\Phi \mathrm{E}[e_0 w_1] + \mathrm{E}[(w_1)^2] \\ &= \Phi^2 P_0 + Q \end{aligned} \tag{10}$$

ดังนั้น ในตัวอย่างนี้ เราจะได้

$$\hat{x}_1^- = 0.8 \times 100 = 80 \tag{11}$$

$$P_1^- = 0.8^2 \times 40000 + 10000 = 35600 \tag{12}$$

สมมุติว่าเรามีเซ็นเซอร์ชนิดหนึ่ง เซ็นเซอร์ของเรามีข้อจำกัดคือไม่สามารถวัดค่า x_k ได้โดยตรง แต่สามารถวัดค่า z_k ได้ และเรารู้ล่วงหน้าว่า z_k กับ x_k มีความสัมพันธ์กันตามสมการ

$$z_k = Hx_k + v_k \tag{13}$$

สมการที่ (13) คือแบบจำลองทางคณิตศาสตร์ของระบบเรียกว่าMeasurement Model เพราะอธิบายความสัมพันธ์ระหว่างค่าที่เซ็นเซอร์วัดได้ z_k กับสถานะของระบบ x_k (1) และ (13) รวมกันเรียกว่าแบบจำลองทางคณิตศาสตร์ของระบบ (System Model) ที่สมบูรณ์ ให้ v_kเป็น Gaussian White Noise มีค่า Mean เท่ากับศูนย์และ Variance เท่ากับ *R* สมมุติว่า ณ เวลา t_1 เซ็นเซอร์ของเราวัดค่าz_1ได้เท่ากับ 30 มีคำถามว่าเราจะสามารถนำค่า $z_1 = 30$ มาช่วยหาค่าประมาณที่ดีที่สุดของ x_1ได้หรือไม่อย่างไร คำตอบต่อคำถามนี้คือหัวใจของ Kalman Filter Algorithm

Kalman Filter บอกว่าค่าประมาณที่ดีที่สุดของ x_1คือ

$$\hat{x}_1^+ = \hat{x}_1^- + K_1[z_1 - H\hat{x}_1^-] \tag{14}$$

K_1 คือ Kalman Gain ณ เวลา t_1เครื่องหมายบวกใช้ระบุว่า $\hat{x}_1^+$เป็นการประมาณค่าของ x_1โดย*นำ*ค่าที่เซนเซอร์วัดได้มาประกอบการพิจารณาด้วย เราสามารถพิสูจน์ได้ว่า $H\hat{x}_1^-$ คือค่าประมาณ $\hat{z}_1$ของค่าที่เซ็นเซอร์ควรจะวัดได้ ณ เวลา t_1จาก

$$\hat{z}_1 = \mathrm{E}[z_1] = \mathrm{E}[Hx_1 + v_k] = H\mathrm{E}[x_1] + \mathrm{E}[v_k] = H\hat{x}_1^- \tag{15}$$

เทอม $z_1 - H\hat{x}_1^-$ (หรือ $z_1 - \hat{z}_1$) เรียกว่า *Residual* จะเห็นได้ว่า (14) คือสมการที่ใช้แก้ไข (Update) ค่า $\hat{x}_1^-$ที่เราคาดการณ์ไว้ล่วงหน้า(ก่อนที่เซ็นเซอร์จะวัดค่าได้) โดยนำผลต่างระหว่างค่าที่เซ็นเซอร์ควรวัดได้กับค่าที่วัดได้จริง มาให้น้ำหนัก (คูณด้วย Kalman Gain *K*)แล้วนำผลที่ได้มาใช้แก้ไขค่า $\hat{x}_1^-$

จะสังเกตได้ว่าขนาดของค่าที่นำมาใช้แก้ไข $\hat{x}_1^-$ ส่วนหนึ่งขึ้นอยู่กับขนาดของ Residual $z_1 - H\hat{x}_1^-$

ถ้าค่าที่คาดการณ์ไว้กับค่าที่วัดได้จริงต่างกันมาก (Residual มีค่าสูง)ทำให้จะต้องแก้ไข $\hat{x}_1^-$ มาก ในทางกลับกันถ้าค่าที่คาดการณ์ไว้ต่างจากค่าที่วัดได้จริงเพียงเล็กน้อย(Residual มีค่าต่ำ) ก็ไม่จำเป็นต้องแก้ไข $\hat{x}_1^-$ มากสมมุติว่าเราหาค่าประมาณ $\hat{x}_1^+$ โดยใช้ (14) (ยังไม่ต้องกังวลว่าKalman Gain K_kจะมีค่าเป็เท่าไร) เราสามารถหาค่า Mean และVariance ของความคลาดเคลื่อนในการประมาณ ได้จาก

$$\begin{aligned}
\mathrm{E}[e_1^+] &= \mathrm{E}[x_1 - \hat{x}_1^+] = \mathrm{E}[x_1 - \{\hat{x}_1^- + K_1(z_1 - H\hat{x}_1^-)\}] \\
&= \mathrm{E}[e_1^- - K_1(He_1^- + v_1)] \\
&= \mathrm{E}[(1 - K_1H)e_1^- + K_1v_1] \\
&= (1 - K_1H)\mathrm{E}[e_1^-] + K_1\mathrm{E}[v_1] = 0
\end{aligned} \tag{16}$$

$$\begin{aligned}
P_1^+ &= \mathrm{E}[(x_1 - \hat{x}_1^-)^2] = \mathrm{E}[(e_1^-)^2] \\
&= \mathrm{E}[\{(1 - K_1H)e_1^- + K_1v_1\}^2] \\
&= \mathrm{E}[\{(1 - K_1H)e_1^-\}^2 + \{2K_1(1 - K_1H)e_1^-v_1\} + \{K_1v_1\}^2] \\
&= (1 - K_1H)^2\mathrm{E}[(e_1^-)^2] + K_1^2\mathrm{E}[(v_1)^2] \\
&= (1 - K_1H)^2P_1^- + K_1^2R
\end{aligned} \tag{17}$$

คำถามต่อไปก็คือจะใช้ค่า Kalman Gain K_1 เป็นเท่าไรจึงจะดีที่สุดสำหรับ Kalman Filter Algorithm ค่า K_1 *K ที่ดีที่สุดในเชิงของความน่าจะเป็น* คือค่าที่ทำให้ความคลาดเคลื่อนในการประมาณ $\hat{x}_1^+$ มีค่าต่ำที่สุด ซึ่งก็คือค่า K_1 ที่ทำให้ (18) เป็นจริง

$$\frac{dP_1^+}{dK_1} = 0 \tag{18}$$

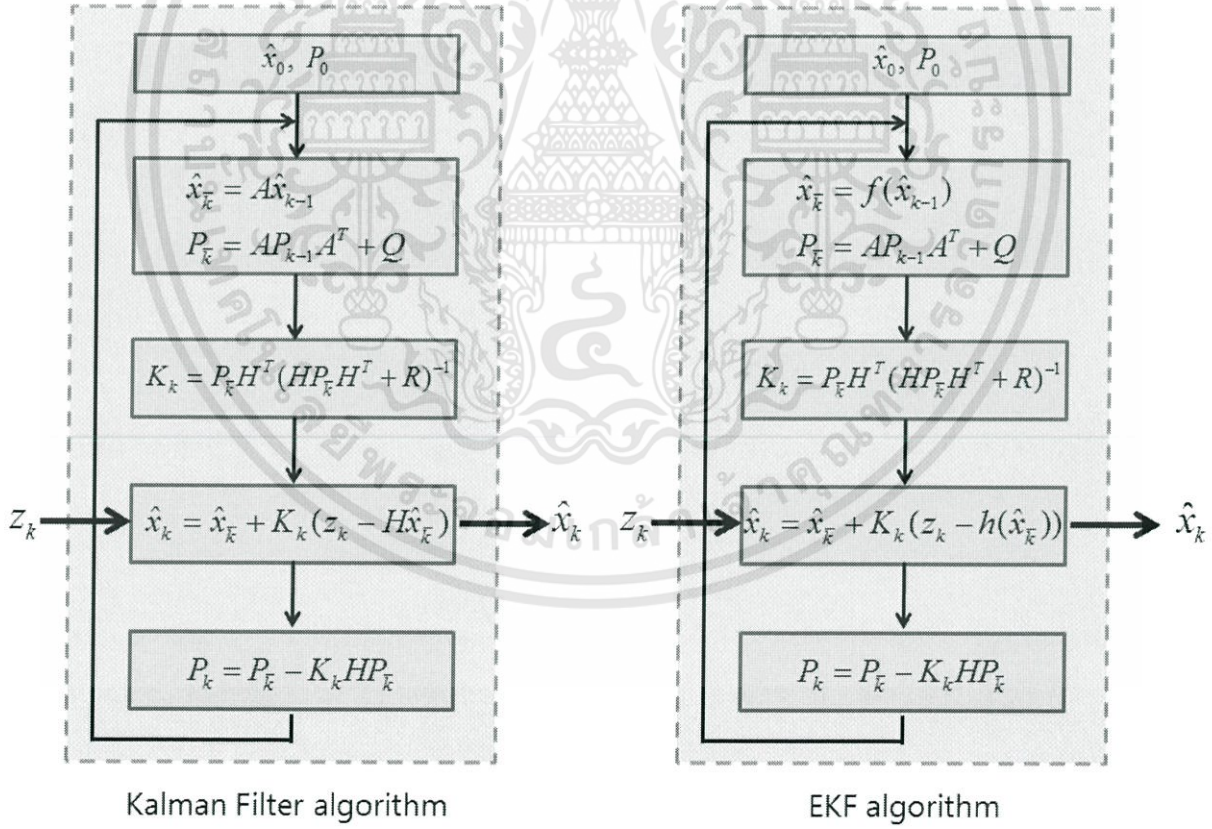
เมื่อ Differentiate (17) เทียบกับ K_1 และให้มีค่าเป็นศูนย์ จะได้

$$\frac{dP_1^+}{dK_1} = -2(1 - K_1 H)P_1^- H + 2K_1 R = 0 \tag{19}$$

จาก (19) เราสามารถหาค่า K_1 ที่ดีที่สุดคือ

$$K_1 = P_1^- H(H^2 P_1^- + R)^{-1} \tag{20}$$

เราสามารถนำ Kalman filter algorithm มาสรุปได้ดังรูปด้านล่าง โดยรูปทางซ้ายเป็น อัลกอลิทึมของ Linear Kalman ทางซ้ายเป็น Extended Kalman filter สำหรับในกรณีที่เป็น non-linear



รูปที่ 2.13 Kalman Filter algorithm

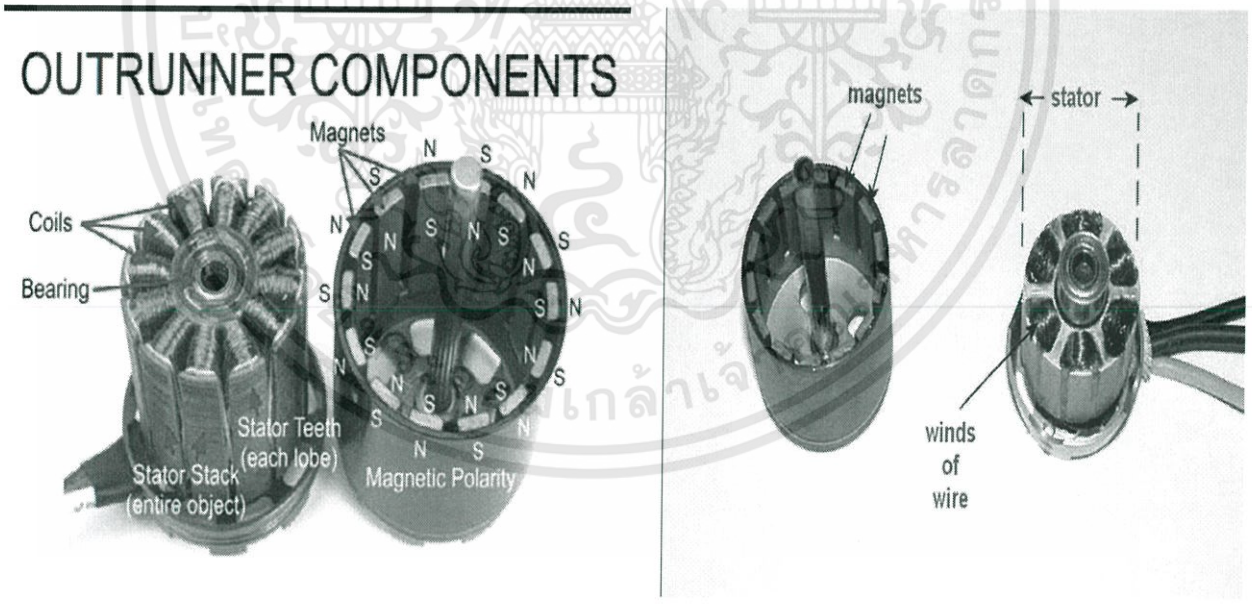
2.5 มอเตอร์ Blushless

Blushless คือมอเตอร์ชนิดที่ไม่แปรงถ่าน หรือมอเตอร์ซิงโครนัส 3 เฟส ที่ทำงานโดยอาศัยอุปกรณ์อิเล็กทรอนิกส์กำลังเป็นสวิตในการตัดต่อกระแสที่จ่ายให้กับขดลวดมอเตอร์ โดยที่ชนิดของมอเตอร์ จะพิจารณาตามลักษณะรูปคลื่นกระแสและคุณสมบัติของแรงบิดหรือทอร์ค โดยจะนิยมเรียกว่า

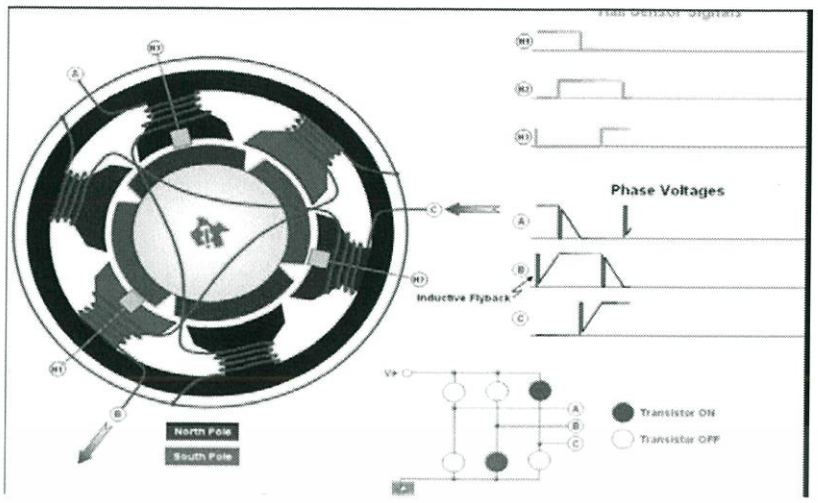
brushless dc motor ในกรณีที่รูปแบบของกระแสและทอร์คของมอเตอร์ที่ใช้มีลักษณะเป็นแบบสี่เหลี่ยม (trapezoidal current/torque) และจะเรียกว่า brushless ac motor หรือเรียกง่ายๆ ว่า brushless เมื่อลักษณะกระแสและทอร์ค เป็นรูปคลื่นซายน์ (sinusoidal current/torque format)

ข้อดีของบัสเลส คือ ไม่มีส่วนสัมผัสที่สึกหรอง่ายทำให้มีอายุการใช้งานที่ยาวนานมากและดูแลรักษาได้ง่าย ไม่มีไฟสปาร์คอีกอย่างคือไม่ได้จ่ายไฟตลอดเวลาเหมือน blushed ทำให้มีประสิทธิภาพสูงกว่าและกินไฟน้อยกว่า

ข้อเสียคือต้องใช้วงจรขับสัญญาณไฟฟ้าให้แก่มอเตอร์ซึ่งยุ่งยากกว่า Blushed ที่เพียงต่อไฟ DC เข้าไปก็หมุนได้ แต่ Blushless ต้องใช้สัญญาณ Pulse DC ที่พอเหมาะ



รูปที่ 2.14 โครงสร้างของ มอเตอร์ Blushless(ซ้าย) และโครงสร้างของ มอเตอร์ Blushless(ขวา)



รูปที่ 2.15 การทำงานของมอเตอร์ Blushless

มอเตอร์กระแสตรงแบบไร้แปรงถ่าน(Brushless DC motor)ที่เลือกใช้มีคุณสมบัติ ดังนี้

1.เป็นมอเตอร์กระแสตรงที่ทำงานโดยอาศัยชุดอุปกรณ์อิเล็กทรอนิกส์ตัดต่อกระแสที่จ่ายให้กับ
ขดลวดมอเตอร์โดยมีความเร็วสูงสุดที่ 1000KV สำมารถคำนวณความเร็วได้โดย
ค่า KV คือ ค่ำคงที่ของมอเตอร์ในหน่วยรอบต่อวินาทีต่อโวลต์ ดังนั้นความเร็วที่ได้จะขึ้นอยู่กับการจ่ายความต่างศักย์ของไฟฟ้าให้ในกรณีนี้จะสมมุติให้ความต่างศักย์ของแบตเตอรี่เท่ากับ 12 โวลต์

จะได้ความเร็วคือ 1000 × 12 = 12000 รอบต่อนาทีซึ่งรอบที่ได้เพียงพอต่อการสร้างแรงบิดในการยกตัวของเครื่องบินได้(ทั้งนี้การคำนวณ ขึ้นอยู่กับการเลือกขนาดใบพัดด้วย)

2. มีการระบายความร้อนที่ดีเพราะมีขดลวด 3 ชุดอยู่ตรงกลาง และมีตัวหมุน (rotor) ที่เป็นแม่เหล็กถาวรเป็นเปลือกวงกลมอยู่รอบนอก (out runner) ทำให้ระบายความร้อนได้อย่างดี

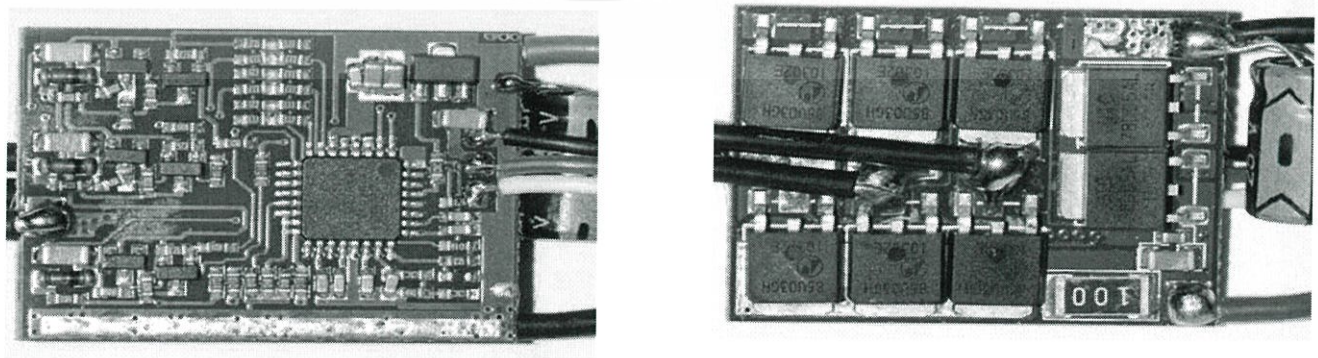
2.6 เครื่องมือควบคุมความเร็วมอเตอร์แบบอิเล็กทรอนิกส์ (Electronic Speed Controller: ESC)

รูปที่ 2.16 Electronic Speed Controller: ESC

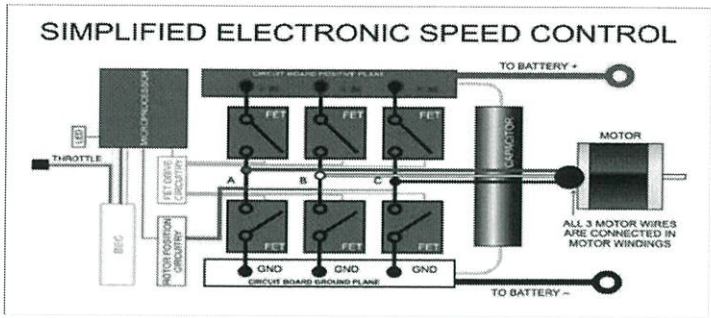
ESC ที่เลือกใช้มีคุณสมบัติดังนี้

1.เป็นอุปกรณ์ที่ออกแบบมาสำหรับใช้ควบคุมมอเตอร์ที่ใช้ในเครื่องบินโดยเฉพาะหรือมีชื่อเรียกอีก
อย่างหนึ่งว่าBEC (Battery Eliminate Circuit) โดยภายในประกอบด้วย วงจรVoltage Regulator สำหรับลดระดับแรงดันของแบตเตอรี่เป็น 5-6 โวลต์ เพื่อใช้ในอุปกรณ์ไฟฟ้าทั้งหมด ของเครื่องบินได้เพียงพอ(สามารถจ่ายให้บอร์ดไมโครคอนโทรลเลอร์ที่ต้องการระดับแรงดัน 5 โวลต์ อื่น ๆ)ได้และใช้ในวงจรของ Speed Control ที่อยู่ภายใน มีกำรออกแบบโดยใช้ ทรานซิสเตอร์ชนิดFET (Field Effect Transistor) ซึ่งเบอร์ที่ใช้งานนี้สำมารถจ่ายกระแสได้สูงสุด ถึง20 แอมแปร์ เป็นวงจรสำหรับควบคุมมอเตอร์3 เฟส โดยการป้อนอินพุตเป็นสัญญาณ PWM และควบคุมมอเตอร์ที่มีรอบสูง ๆ เช่น มอเตอร์ของเครื่องบินบังคับได้

2 .สามารถควบคุมความเร็วของมอเตอร์โดยจ่ายสัญญาณพัลส์วิดมอดดเลชัน (Pulse width modulation) โดยความเร็วที่ได้ขึ้นอยู่กับความกว้างของสัญญาณ PWM ตั้งแต่1-2 ms ที่สั่งจ่าย โดยไมโครคอนโทรลเลอร์

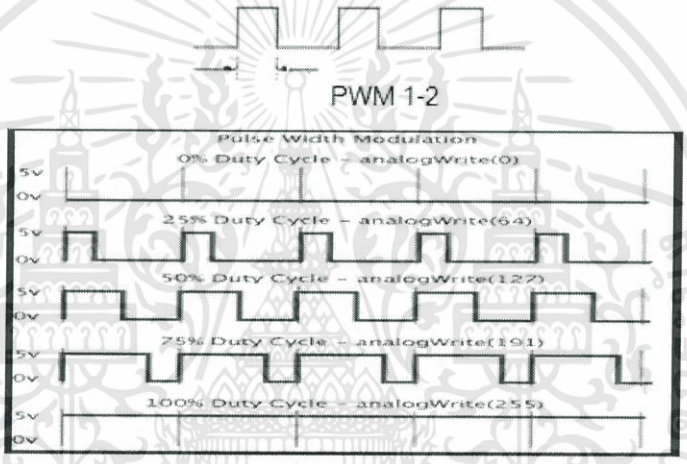


รูปที่ 2.17 รูปอุปกรณ์ภายใน Electronic Speed Controller



รูปที่ 2.18 วงจรภายใน Electronic Speed Controller

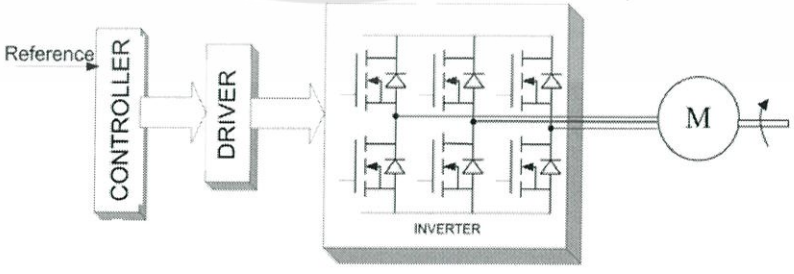
สัญญาณที่ Input ของ Electronic Speed Controller จะเป็นสัญญาณ PWM ตั้งแต่1-2 ms โดยความเร็วของมอเตอร์จะขึ้นอยู่กับ Duty Cycle ของ PWM



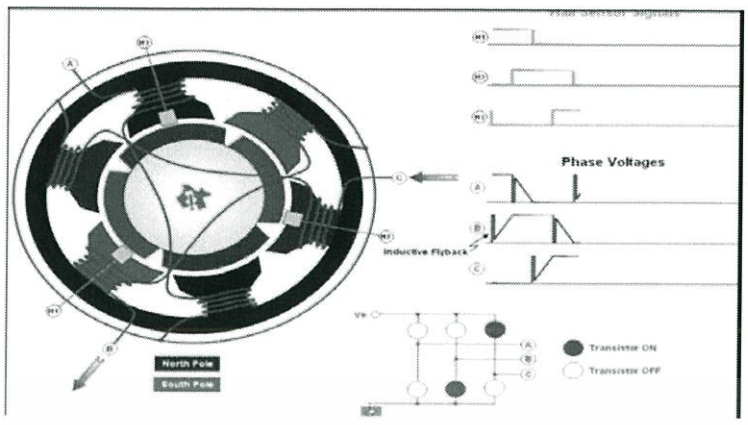
รูปที่ 2.19 สัญญาณ PWM ที่เป็น Input ของ Electronic Speed Controller

สัญญาณOutput ของ Electronic Speed Controller

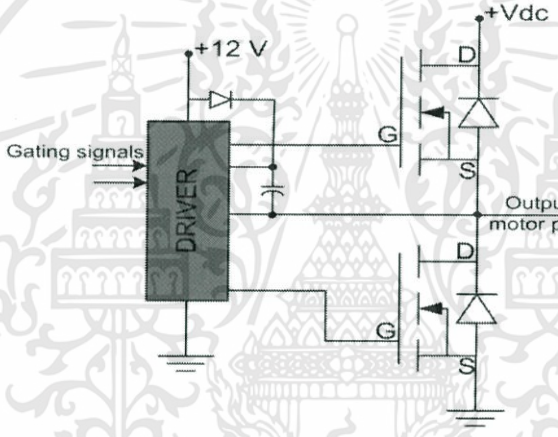
สัญญาณOutput จะเป็นสัญญาณ DC 3 เฟส ที่ได้จากการควบคุมการทำงานของทรานซิสเตอร์ ที่ถูกสั่งมาจาก controller โดยทรานซิสเตอร์จะสลับการทำงานทีละ 2 ตัว



รูปที่ 2.20 บล็อกไดอะแกรม การทำงาน Electronic Speed Controller



รูปที่ 2.21 แสดงการทำงานและสัญญาณที่ได้จากการควบคุมของ Electronic Speed Controller



รูปที่ 2.22วงจร Driver ภายใน Electronic Speed Controller ในแต่ละเฟสของมอเตอร์

Driver จะถูกควบคุมโดย Microprocessor ที่มีอยู่ใน Electronic Speed Controller และทำการประมวลผลจาก สัญญาณ PWM ที่เป็น Input ซึ่งขึ้นอยู่กับ Duty Cycle ของ PWM

2.7 ระบบควบคุม PID (Proportional-Integral-Derivative)

คอนโทรลเลอร์แบบ PID (หรือ PID controller) จากชื่อจะเห็นได้ว่ามีสองส่วน คือ คอนโทรลเลอร์ และ PID ส่วนคอนโทรลเลอร์ก็คืออุปกรณ์ในการควบคุมระบบตามที่ต้องการโดยจะมีการตั้งค่า SP (set point : เซ็ตพอยน์ต หรือค่าที่ต้องการ)ไว้ และนำ มาเปรียบเทียบกับค่า PV(Process Variable : ตัวแปรกระบวนการหรือค่าจริงที่เกิดจากผลการทำงานของระบบ) เพื่อให้ได้ค่าError แล้วคอนโทรลเลอร์จะนำค่า Error นั้นมาทำการปรับแต่งค่าเอาท์พุตหรือ MV (ManipulatedVariable : ตัวแปรที่ถูกควบคุม บางที่จะเรียกว่า CV(Control Variable : ตัวแปรใน

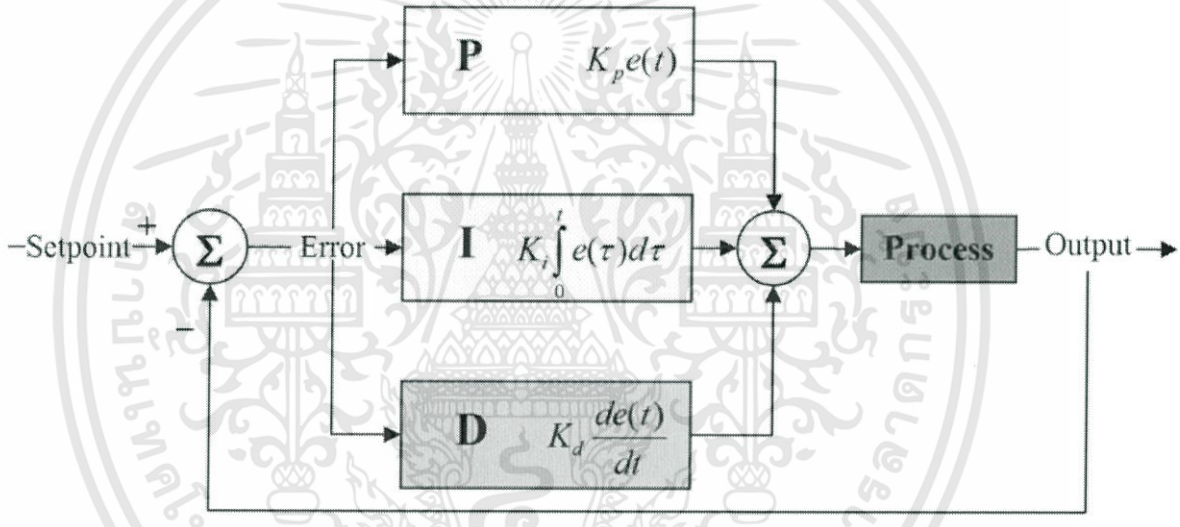
การคอนโทรล) เพื่อลดค่า Error ให้ได้ต่ำที่สุด ส่วน PID ย่อมาจากProportional-Integral-Derivative แปลเป็นไทยก็จะได้ว่า อัตราส่วน-อินทีกรัล-อนุพันธ์ โดย

1) Proportional เป็นส่วนปฏิกิริยาต่อ Errorณ ขณะนั้น (Current error)

2) Integral เป็นส่วนปฏิกิริยาที่เกิดจากผลรวมของค่า Error ที่ผ่านมาล่าสุด (Sum of recent errors)

3) Derivative เป็นส่วนปฏิกิริยาที่เกิดจากอัตราการเปลี่ยนแปลงของค่า Error (Rate at which the error has been changing)

ผลรวมตามน้ำหนัก(Weighted sum) ของทั้งสาม ซึ่งปรับแต่งโดยค่าคงที่ Kp, Ki,และ Kd เพื่อให้ได้การตอบสนองที่ต้องการ จะถูกนำไปใช้ในการควบคุมกระบวนการต่างๆ



รูปที่ 2.23 Block diagram ของคอนโทรลเลอร์แบบ PID

ทฤษฎีคอนโทรเลอร์แบบ PID

การควบคุมแบบ PID นั้นประกอบด้วย

$$MV(t) = P_{out} + I_{out} + D_{out} \quad (21)$$

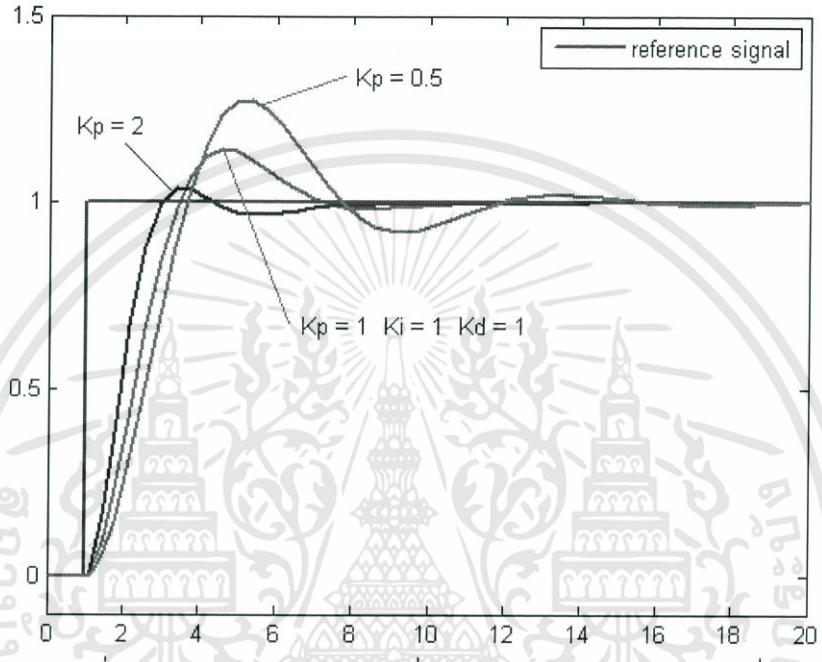
เทอมสำหรับการปรับแต่ง 3 เทอมรวมกันเป็น MVโดยที่ Pout Iout และ Dout เป็นเอาท์พุตจากแต่ละเทอมตามลำดับ

เทอม Proportional

เทอม Proportional เปลี่ยนแปลงตามอัตราส่วนของค่า Error ปัจจุบัน ซึ่งค่า Pout สามารถคำนวณได้โดยการนำค่า Error มาคูณกับค่าคงที่ Kp

$$P_{out} = K_p e(t) \quad (22)$$

ค่า K_p ที่สูง จะเป็นผลให้ค่าเอาท์พุตมากขึ้นตาม หากค่า K_p มากเกินไป ระบบอาจจะไม่เสถียรได้ ในทางตรงข้ามหากค่า K_p น้อยเกินไปอาจทำให้ระบบตอบสนองช้าเกินไป ซึ่งในกรณีนี้เป็นไปได้ว่า การตอบสนองดังกล่าว อาจจะสู้สิ่งรบกวน (Disturbance) ในระบบได้ทันการ เช่นในการเร่งรอบเครื่องยนต์ เมื่อมีโหลดมากระทำกับเครื่อง หากคอนโทรลเลอร์เร่งรอบเครื่องช้าไม่ทันกับโหลดที่เพิ่มขึ้นมาทันที รอบเครื่องก็จะค่อยๆตกลง และเครื่องก็จะดับในที่สุด



รูปที่ 2.24 PV vs time โดยเปลี่ยนค่า Kp (ค่า Ki และ Kd คงที่)

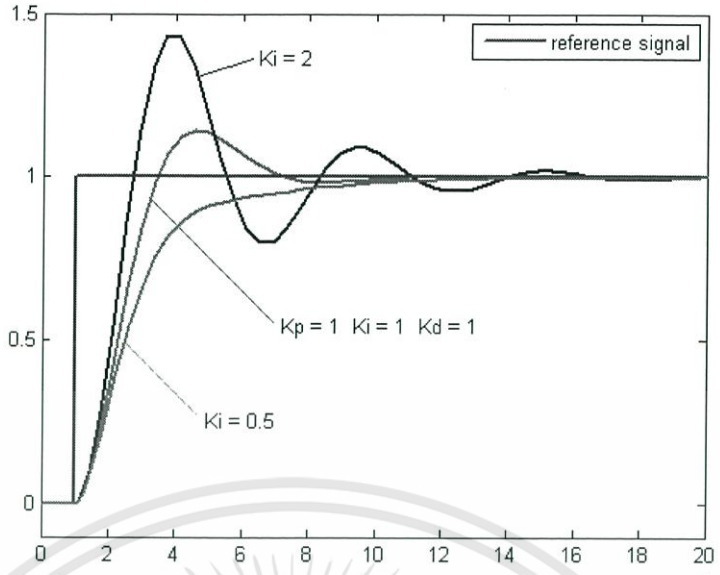
ในกรณีระบบที่ไม่มี Disturbance การใช้เทอม Proportional อย่างเดียวจะทำให้ระบบเกิดการ Oscillate รอบๆ SP จะไม่อยู่ที่ SP พอดี

เทอม Integral

เทอม Integral เป็นอัตราส่วนของค่า Error สะสมในหนึ่งช่วงเวลา (ปัจจุบัน ย้อนไปในอดีต) ค่า Iout เกิดจากผลคูณของค่าคงที่ Ki กับผลรวมของ e(t) ซึ่งเป็นค่าสะสมของ Error ที่ควรจะต้องถูกแก้ไขมาก่อนหน้านี้

$$I_{out} = K_i \int_0^t e(t)dt \qquad (23)$$

เทอม Integral (เมื่อใช้ร่วมกับเทอมProportional) จะช่วยเร่งให้ระบบวิ่งเข้าหา SP เร็วขึ้น และช่วยลด Error ที่เกิดจากการใช้เทอม Proportional อย่างเดียว อย่างไรก็ตาม เนื่องจากว่าเทอม Integral นั้นเกิดจากการคำนวณโดยรวมError ที่เกิดขึ้นในอดีตด้วย อาจจะทำให้เกิด Overshoot เกิน SP ในค่าปัจจุบันด้วย



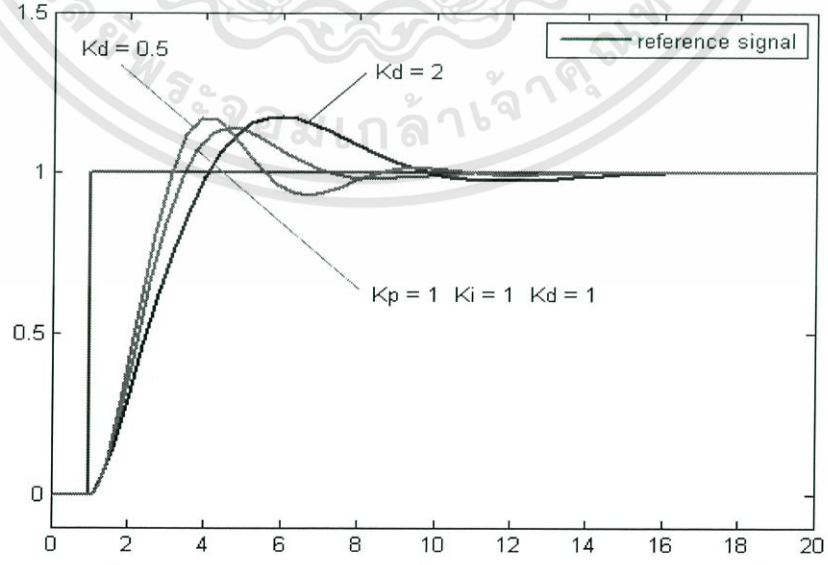
รูปที่ 2.25 PV vs time โดยเปลี่ยนค่า Ki (ค่า Kp และ Kd คงที่)

เทอม Derivative

อัตราการเปลี่ยนแปลงของค่า Error ในโปรเซสนั้นคำนวณได้โดยสโลป (Slope) ของกราฟ Error vs time ซึ่งก็คืออนุพันธ์แรกเทียบกับเวลา และคูณค่าสโลปนี้กับค่าคงที่ Kd ก็จะได้เทอม Derivative

$$D_{out} = K_d \frac{de}{dt}(t) \qquad (24)$$

เทอม Derivative จะช่วยดึงเอาท์พุตจากคอนโทรลเลอร์ให้ช้าลง ซึ่งจะเห็นผลได้ชัดเมื่อโปรเซสเข้าใกล้ Set point ฉะนั้นเทอม Derivativeจึงช่วยลดอาการ Overshoot ซึ่งเกิดจากเทอม Integral และช่วยปรับปรุงเสถียรภาพของระบบโดยรวม อย่างไรก็ตาม ค่า Derivative นั้นค่อนข้างไวต่อ Noise ซึ่งอาจทำให้ระบบไม่เสถียรได้ หาก Noise และค่า Kp สูงเกินไป



รูปที่ 2.26 PV vs time โดยเปลี่ยนค่า Kd (ค่า Kp และ Ki คงที่)

รวมสามเทอมเข้าด้วยกัน

เมื่อรวมเทอม Proportional, Integral, และDerivative เข้าด้วยกันก็จะได้ เอาท์พุตจากคอนโทรลเลอร์ PID ดังนี้

$$MV(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de}{dt}(t) \quad (25)$$

โดยค่าพารามิเตอร์ต่างๆ คือ

ค่า Proportional gain, Kp: ค่าที่มากขึ้นหมายถึงการตอบสนองที่เร็วขึ้น เพราะค่า Error ยิ่งมาก ค่าชดเชยจากเทอมนี้ก็จะมากขึ้นตาม ค่า Gainที่มากเกินไปจะนำไปสู่โปรเซสที่ไม่เสถียรและการแกว่ง(Oscillation)

ค่า Integral gain, Ki: ค่าที่มากขึ้นหมายถึง ค่า error แบบ Steady-state จะถูกกำจัดได้เร็วขึ้น ข้อเสียก็คือ Overshoot ค่า Error ที่เป็นลบจะต้องถูกแก้ด้วย Error ที่เป็นบวก ก่อนที่ระบบจะเข้าสู่ Steady-state

ค่า Derivative gain, Kd: ค่าที่มากขึ้นหมายถึงขนาด Overshoot ที่ลดลง แต่ก็อาจทำให้การตอบสนองช้าลงบ้าง และอาจนำไปสู่ความไม่เสถียรของระบบเนื่องจาก Noise ได้

PID Algorithm

```
previous_error = 0
integral = 0
start:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
```

การปรับแต่งค่า (Loop Tuning)

ในการปรับแต่งค่ามี 3 วิธีหลักดังนี้

การปรับแต่งด้วยมือ (Manual tuning) จะเริ่มโดยการเซตค่า Ki และ Kd เป็นศูนย์ และค่อยๆเพื่อค่า Kp เรื่อยๆ จนกระทั้งระบบเริ่มเกิดการแกว่ง จากนั้นให้เซตค่า Kp เป็นครึ่งหนึ่งของค่านั้น จากนั้นให้เริ่มเพิ่มค่า Ki จนได้เวลาการตอบสนองของระบบที่ต้องการ จากนั้นหากจำเป็น เพิ่มค่า Kd จนกระทั่งการควบคุมเร็วพอที่ยอมรับได้โดยเทียบกับเมื่อระบบมี Disturbanceอ้างอิง รายละเอียดการปรับแต่งค่า Gain ต่างๆด้วยมือมีรายละเอียดเพิ่มเติม ซึ่งผู้ที่สนใจจะต้องศึกษาเพิ่มเติม

การปรับแต่งด้วยวิธี Ziegler–Nichols

จะเริ่มด้วยการเซตค่า Ki และ Kd เป็นศูนย์ จากนั้นจะเพิ่มค่า Kp ไปจนถึงค่า Kc (Critical Gain) ระบบจะเริ่มแกว่ง ให้วัดคาบของการแกว่ง Pc และให้ใช้ตารางข้างล่างเพื่อหาค่า Gain อื่นๆ

วิธีZiegler–Nichols			
Control Type	*Kp*	*Ki*	*Kd*
P	0.50 *Kc*	–	–
PI	0.45 *Kc*	1.2 *Kp / Pc*	–
PID	0.60 *Kc*	2 *Kp / Pc*	*Kp Pc* / 8

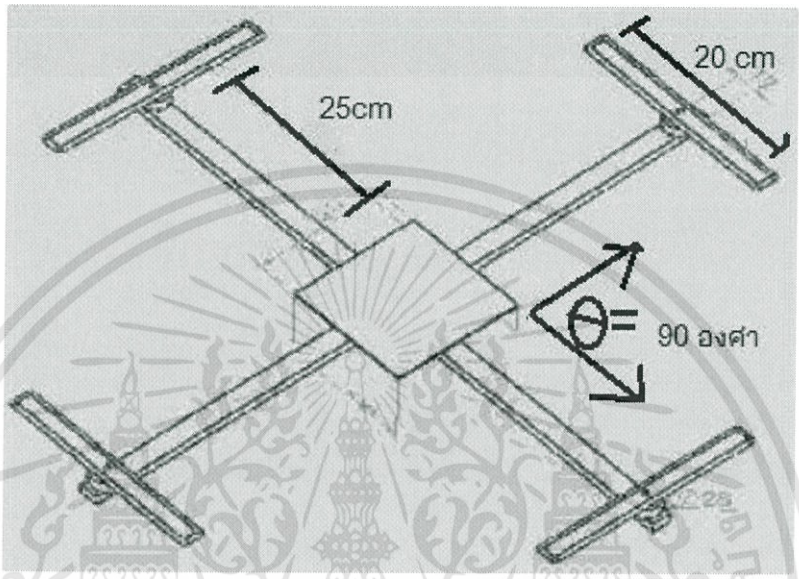
ตาราง ที่ 2.1 **วิธีZiegler–Nichols**

ข้อจำกัดของคอนโทรลเลอร์แบบ PID

ในกรณีที่คอนโทรลเลอร์ทำงานได้ไม่ดีเกิดอาการฮันต์ อาจจะต้องใช้รวมคอนโทรลเลอร์แบบ Feed forward ด้วย โดยการบวกค่า Bias เพิ่มเข้าไปในสมการ MV(t)เนื่องจากคอนโทรลเลอร์ PID เป็นลิเนียร์(Linear: เชิงเส้น)ในกรณีที่ใช้กับระบบที่เป็นนอนลิเนียร์ (Non-linear: ไม่เป็นเชิงเส้น) เช่น ระบบ HVAC (Heating – Ventilating – Air –Conditioning หรือบางครั้งเรียกว่าClimate control) อาจจะต้องใช้การปรับค่า Gain ตามตารางเวลาหรือใช้คอนโทรลเลอร์แบบ Fuzzy logic แทน

บทที่ 3
การออกแบบ Quad copter

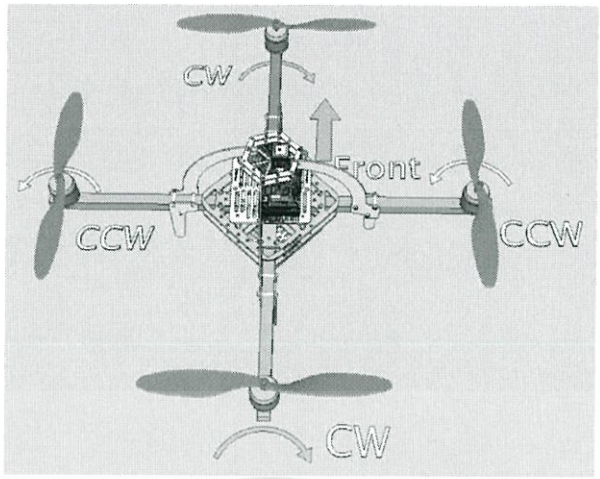
3.1 โครงสร้าง Quad copter และการออกแบบ



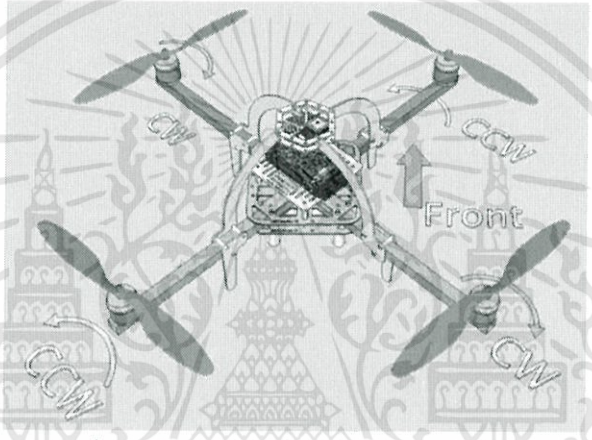
รูปที่ 3.1 โครงสร้างพื้นฐานของ Quad copter

จะมีลักษณะเป็นรูปสี่เหลี่ยมสมมาตรเส้นผ่านศูนย์กลาง 50 cm มุมตั้งฉาก 90 องศา ซึ่งจะทำให้ Quad copter มีความสมดุลในการทรงตัวและการบินในอากาศ และจะใช้ใบพัดจำนวน 4 ใบพัด ขนาดเส้นผ่านศูนย์กลางประมาณ 20 cm (8 นิ้ว)

โครงสร้างและการทำงานของ Quad copter ซึ่งจะใช้ใบพัด ทั้งหมด 4 ใบพัด ซึ่งตัวอย่างในการทำงานหรือการบินเบื้องต้น และทิศทางการบิน จะมีลักษณะดังภาพ ลักษณะการบินของ Quad copter และภาพทิศทางการบิน Quad copter



รูปที่3.2 ลักษณะการบินของ Quad copter



รูปที่3.3 ลักษณะการบินของ Quad copter

จากการศึกษา พบว่าในการทำงานของเครื่องบิน ที่ใช้ 4 ใบพัด มีลักษณะการบิน ดังนี้

บินขึ้น-บินลง มอเตอร์ทั้ง 4 ตัว หมุนด้วยความเร็วเท่ากัน เพื่อลอยตัว ในแนวดิ่ง

บินเอียงขวา ลดความเร็วมอเตอร์ด้านขวา เพิ่มความเร็วมอเตอร์ด้านซ้าย มอเตอร์หน้าหลังระดับเดียวกัน

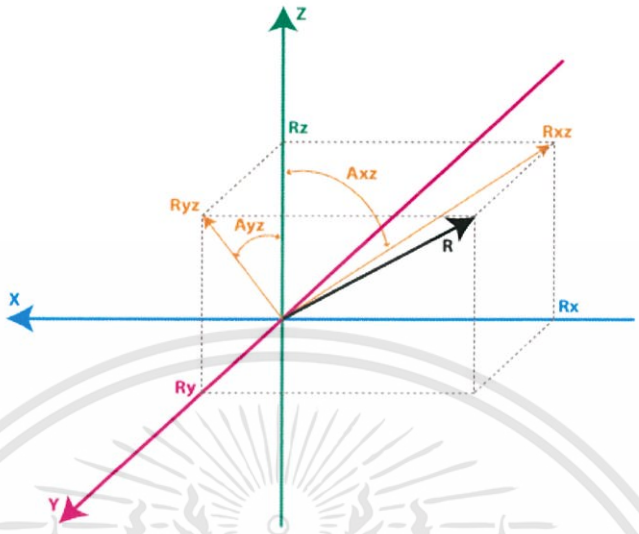
บินเอียงซ้าย ลดความเร็วมอเตอร์ด้านซ้าย เพิ่มความเร็วมอเตอร์ด้านขวา มอเตอร์หน้าหลังระดับเดียวกัน

บินไปข้างหน้า ลดความเร็วมอเตอร์ด้านหน้า เพิ่มความเร็วมอเตอร์ด้านหลัง มอเตอร์ซ้ายขวาระดับเดียวกัน

บินไปข้างหลัง ลดความเร็วมอเตอร์ด้านหลัง เพิ่มความเร็วมอเตอร์ด้านหน้า มอเตอร์ซ้ายขวาระดับเดียวกัน

บินหมุนตัวซ้ายขวา โดยการลดความเร็วมอเตอร์สองตัวที่อยู่ตรงข้ามกันและเพิ่มความเร็วมอเตอร์อีกคู่หนึ่งที่อยู่ตรงข้ามกันพร้อมกัน ตามด้านที่ต้องการหมุน

3.2 การใช้ Kalman filter ในการคำนวณหาองศา



รูป 3.4 เวกเตอร์ต่างๆ

Racc = [RxAcc,RyAcc,RzAcc] เป็นเวกเตอร์ของแรงที่กระทำต่อ Accelerometer ซึ่งประกอบด้วย RxAcc RyAcc และ RzAcc

$$|R_{acc}| = \sqrt{R_{xAcc}^2 + R_{yAcc}^2 + R_{zAcc}^2} \quad (26)$$

$$R_{acc}(normalized) = \left[\frac{R_{xAcc}}{|R_{acc}|}, \frac{R_{yAcc}}{|R_{acc}|}, \frac{R_{zAcc}}{|R_{acc}|}\right] \quad (27)$$

กำหนด Rest = [RxEst, RyEst, RzEst] เป็นค่า ทำนาย และค่าที่เราจะนำไปใช้

จากนั้นกำหนดค่าเริ่มต้นโดยใช้ค่าจาก Accelerometer

Rest(0)=Racc(0):

RxEst(0) = RxAcc (0)

RyEst(0) = RyAcc (0)

RzEst(0) = RzAcc (0)

Rest(n-1) คือการประมาณค่าก่อนหน้า

กำหนด เวกเตอร์ขึ้นมาใหม่ ซึ่งเป็นค่าที่มาจาก Gyroscope และ ค่าประมาณก่อนหน้า

Rgyro = [RxGyro, RyGyro, RzGyro]

จากรูป 3.5 สังเกตความสัมพันธ์ของ Gyroscope กับ Rz และ Rxz จาก พีธากอรัส

$$\tan(A_{xz}) = \frac{R_x}{R_z} => A_{xz} = \tan^{-1}\frac{R_x}{R_z} \quad (28)$$

ดังนั้น

$$A_{xz}(n-1) = \tan^{-1}\frac{R_{xEst}(n-1)}{R_{zEst}(n-1)} \quad (29)$$

Gyroscope วัดอัตราการเปลี่ยนแปลงของมุม ดังนั้นเราจึงทำนายมุมใหม่ได้

$$A_{xz}(n) = A_{xz}(n-1) + Rate\ A_{xz} \times T \quad (30)$$

$$A_{yz}(n) = A_{yz}(n-1) + Rate\ A_{yz} \times T \quad (31)$$

เราจะนำค่า $A_{xz}(n)$ และ $A_{yz}(n)$ นี้ไปใช้งาน จากนั้นจะทำการปรับปรุงค่าประมาณขึ้นใหม่

$$|R_{Gyro}| = \sqrt{R_{xGyro}^2 + R_{yGyro}^2 + R_{zGyro}^2} \quad (32)$$

$$|R_{Gyro}| = 1 \quad (33)$$

$$R_{xGyro} = \frac{R_{xGyro}}{1} = \frac{R_{xGyro}}{\sqrt{{R_{xGyro}}^2 + {R_{yGyro}}^2 + {R_{zGyro}}^2}}$$

$$= \frac{\frac{R_{xGyro}}{\sqrt{{R_{xGyro}}^2 + {R_{zGyro}}^2}}}{\sqrt{\frac{{R_{xGyro}}^2 + {R_{yGyro}}^2 + {R_{zGyro}}^2}{{R_{xGyro}}^2 + {R_{zGyro}}^2}}}$$

$$= \frac{\frac{R_{xGyro}}{\sqrt{{R_{xGyro}}^2 + {R_{zGyro}}^2}}}{\sqrt{1 + \frac{{R_{yGyro}}^2 {R_{zGyro}}^2}{{R_{zGyro}}^2({R_{xGyro}}^2 + {R_{zGyro}}^2)}}}$$

จาก $\frac{R_{xGyro}}{\sqrt{{R_{xGyro}}^2+{R_{zGyro}}^2}} = \sin(Axz)$, $\frac{R_{zGyro}}{\sqrt{{R_{xGyro}}^2+{R_{zGyro}}^2}} = \cos(Axz)$

และ $\frac{R_{yGyro}}{R_{zGyro}} = \tan(Ayz)$ จะได้

$$R_{xGyro}(n) = \frac{\sin(Axz(n))}{\sqrt{1+[\cos(Axz(n))]^2[\tan(Axz(n))]^2}} \quad (34)$$

ทำนองเดียวกัน

$$R_{yGyro}(n) = \frac{\sin(Ayz(n))}{\sqrt{1+[\cos(Ayz(n))]^2[\tan(Axz(n))]^2}} \quad (35)$$

และจาก สมการที่ 32และ 33

จะได้

$$R_{zGyro} = \pm\sqrt{1 - R_{xGyro}^2 - R_{yGyro}^2} \quad (36)$$

R_{Gyro} เป็นค่าที่ได้จาก Rest(n-1) และ จาก Gyroscope ในปัจจุบัน

กำหนดให้ w_1, w_2 เป็นweight

ดังนั้นจะได้ค่าประมาณป็น :

$$R_{est}(n) = \frac{w_1 R_{acc} + w_2 R_{gyro}}{w_1 + w_2} = \frac{\frac{w_1}{w_1} R_{acc} + \frac{w_2}{w_1} R_{gyro}}{\frac{w_1}{w_1} + \frac{w_2}{w_1}}$$

$$- \frac{R_{acc} + W R_{gyro}}{1 + W}$$

$$R_{xEst}(n) = \frac{R_{xAcc} + W R_{xGyro}}{1+W} \quad (37)$$

$$R_{yEst}(n) = \frac{R_{yAcc} + W R_{yGyro}}{1+W} \quad (38)$$

$$R_{zEst}(n) = \frac{R_{zAcc} + W R_{zGyro}}{1+W} \quad (39)$$

จากนั้น Normalize

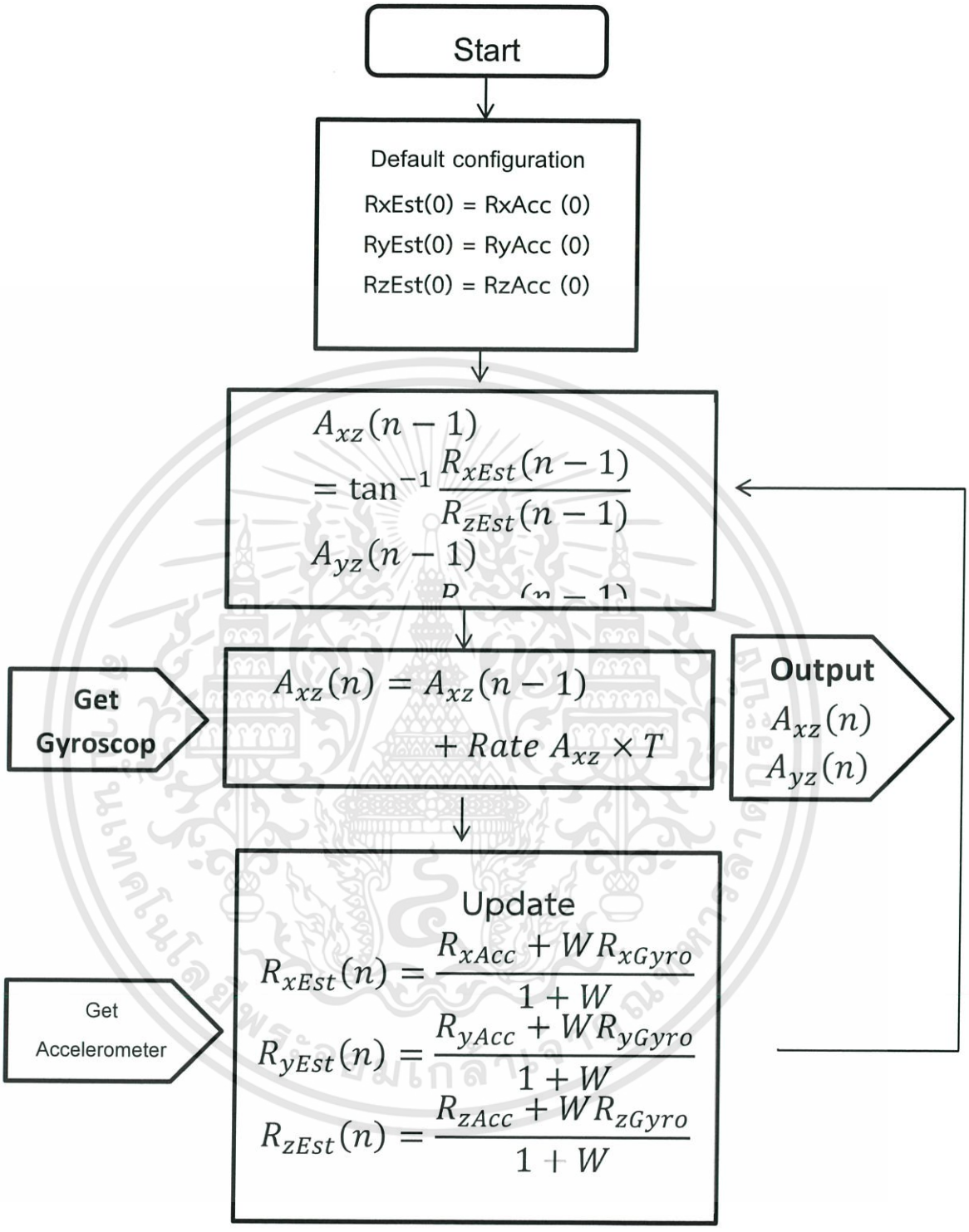
$$\mathrm{R} = \sqrt{R_{xEst}^2 + R_{yEst}^2 + R_{zEst}^2} \tag{40}$$

$$R_{xEst}(n) = \frac{R_{xEst}(n)}{R} \tag{41}$$

$$R_{yEst}(n) = \frac{R_{yEst}(n)}{R} \tag{42}$$

$$R_{zEst}(n) = \frac{R_{zEst}(n)}{R} \tag{43}$$

จากนั้นก็จะวนเพื่อทำอีกรอบ



รูปที่ 3.5 โฟลชาร์ตการทำงานของ Algorithm

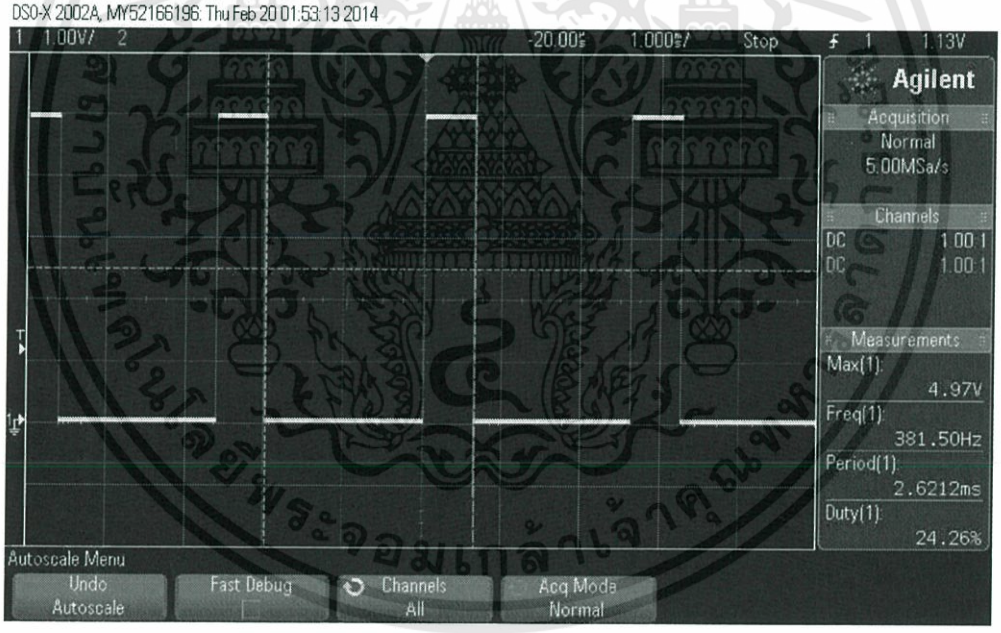
3.3 การสร้าง PWM

กำหนดฐานเวลาอ้างอิง ในที่นี้จะใช้ timer1ซึ่ง timer1 จะนับค่าได้สูงสุด 65536 ค่า และใช้ปรีสเกลเลอร์ขนาด 8 บิต โดยกำหนดให้ 1 รอบการทำงานของ timer ใช้เวลาเท่ากับ 0.0025 วินาที หรือ ความถี่ในการเกิด Over flow เท่ากับ 400 Hz ซึ่งเป็นความถี่ที่ใช้ในการควบคุม ESC (*Electronic speed control*)

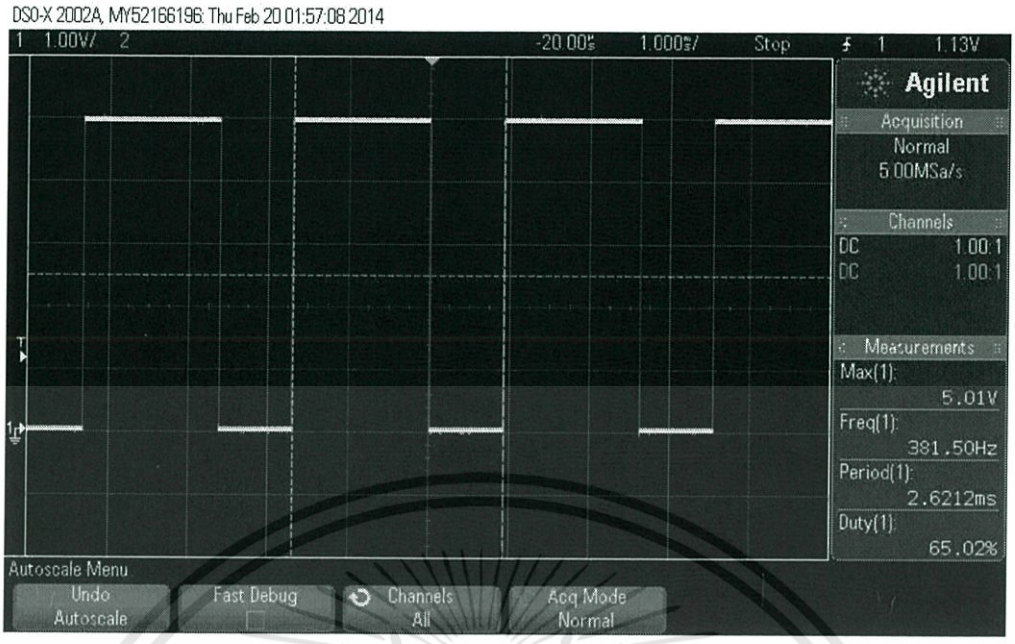
ทำการคำนวนหาจำนวนในการนับเพื่อสร้าง Duty cycle เช่น ถ้าต้องการ Duty cycle 30%

$$\text{จำนวนการนับ} = \frac{30 \times 65536}{100} = 19660$$

เมื่อเกิดการ Interrupt เนื่องจาก Over flow จะสั่งให้สัญญาณขาออกเป็น High แล้วจากนั้นจะใช้ Interrupt ccp ในโหมดเปรียบเทียบข้อมูล ว่าเท่ากับค่าจำนวนการนับที่คำนวนไว้หรือไม่ ถ้าเท่าก็ให้สัญญาณขาออกเป็น Low จากนั้นก็รอให้เกิดการ Interrupt เนื่องจาก Over flow เพื่อกลับไปทำซ้ำไปซ้ำมาเรื่อยๆ



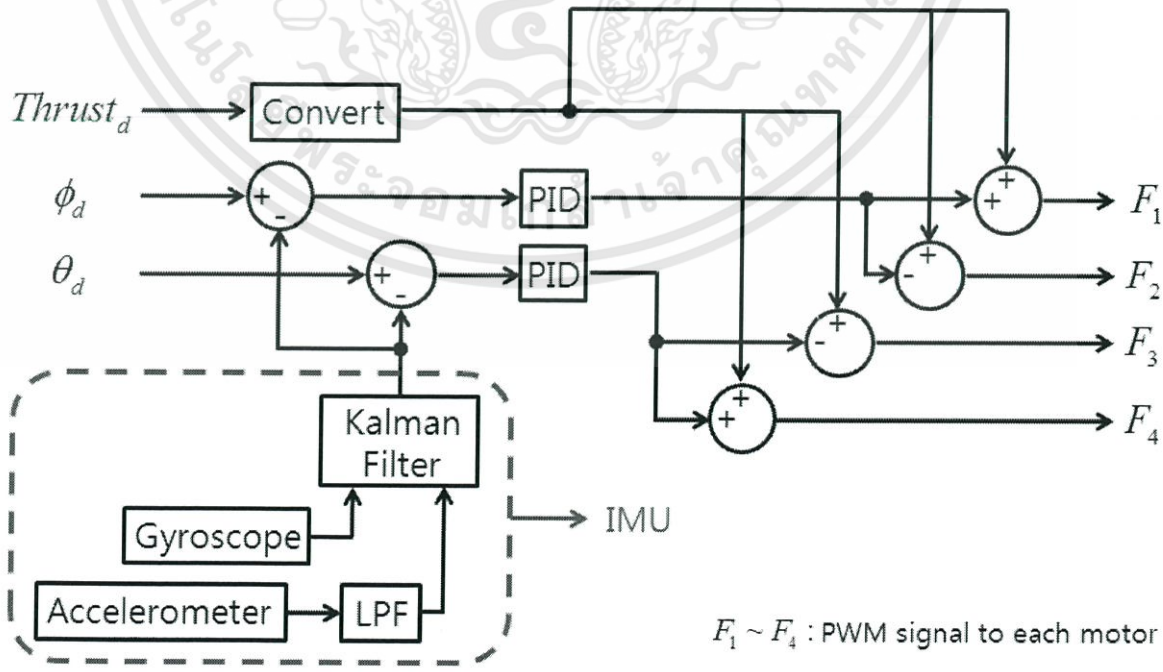
รูปที่ 3.6 สัญญาณ PWM ที่มอเตอร์เริ่มหมุน



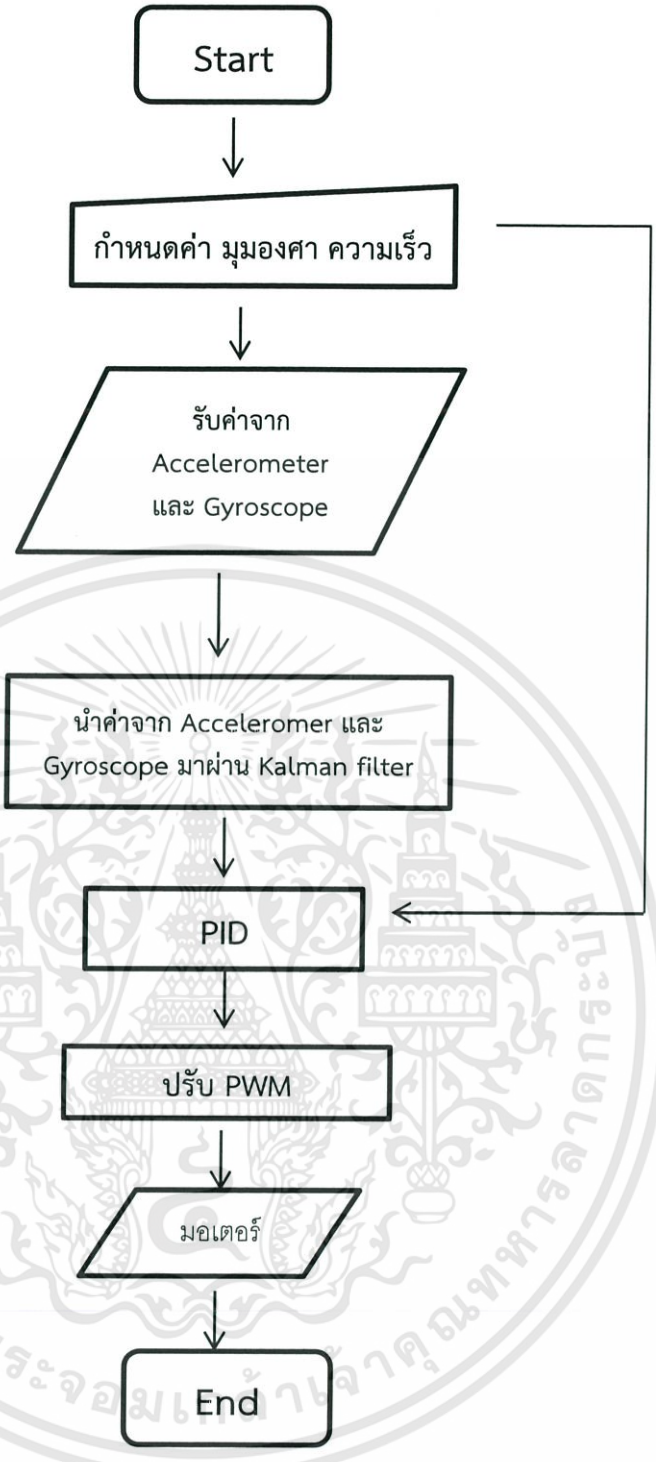
รูปที่ 3.7 สัญญาณ PWM ที่ตัวเครื่อง Quad copter เริ่มยกตัวขึ้นจากพื้นได้

3.4 ระบบควบคุม PID (Proportional-Integral-Derivative)

เราจะใช้ระบบควบคุม PID เพื่อมาในการควบคุมองศาของคอปเตอร์โดยจะทำการเปรียบเทียบองศาที่เรากำหนดกับองศาปัจจุบันที่ได้จาก Kalman filter มาเป็นค่า Error แล้วนำค่า Error นั้นมาเข้ากระบวนการ PID



รูปที่ 3.8 ระบบควบคุม



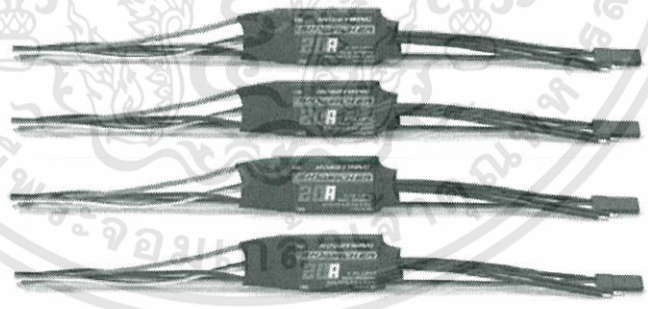
รูปที่ 3.9 โฟลชาร์ตการทำงานของระบบ

3.5 อุปกรณ์สำคัญ

-มอเตอร์กระแสตรงแบบไร้แปรงถ่าน (Brushless DC motor) จำนวน 4 ชิ้น

รูปที่ 3.10 มอเตอร์กระแสตรงแบบไร้แปลงถ่าน

- เครื่องมือควบคุมความเร็วมอเตอร์แบบอิเล็กทรอนิกส์ (Electronic Speed Controller: ESC) จำนวน 4 ชิ้น



รูปที่ 3.11 เครื่องมือควบคุมความเร็วมอเตอร์

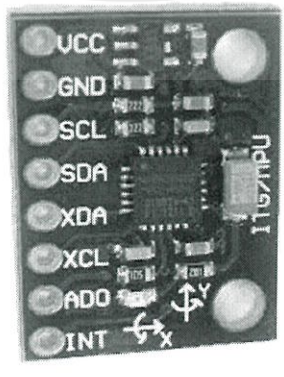
-ใบพัด ขนาด8 นิ้ว จำนวน 4 ใบ

รูปที่ 3.12 ใบพัด

-แบตเตอรี่ แบบลิโพ Lipo HeliCox 11.1V 3000 mAh 30C 3Cell Nano-Platium

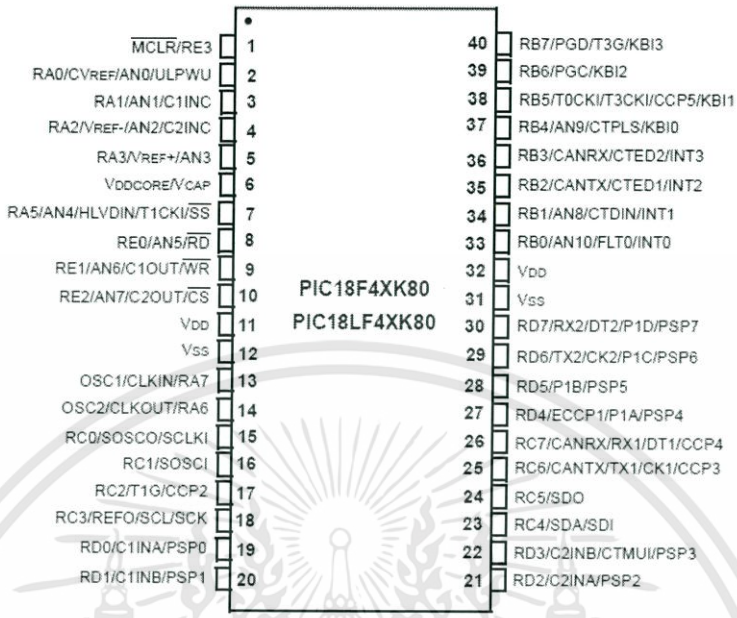
รูปที่ 3.13 **แบตเตอรี่**

-เซนเซอร์ MPU-6050 เป็นเซนเซอร์ Accelerometer และ Gyroscope ด้วยกัน



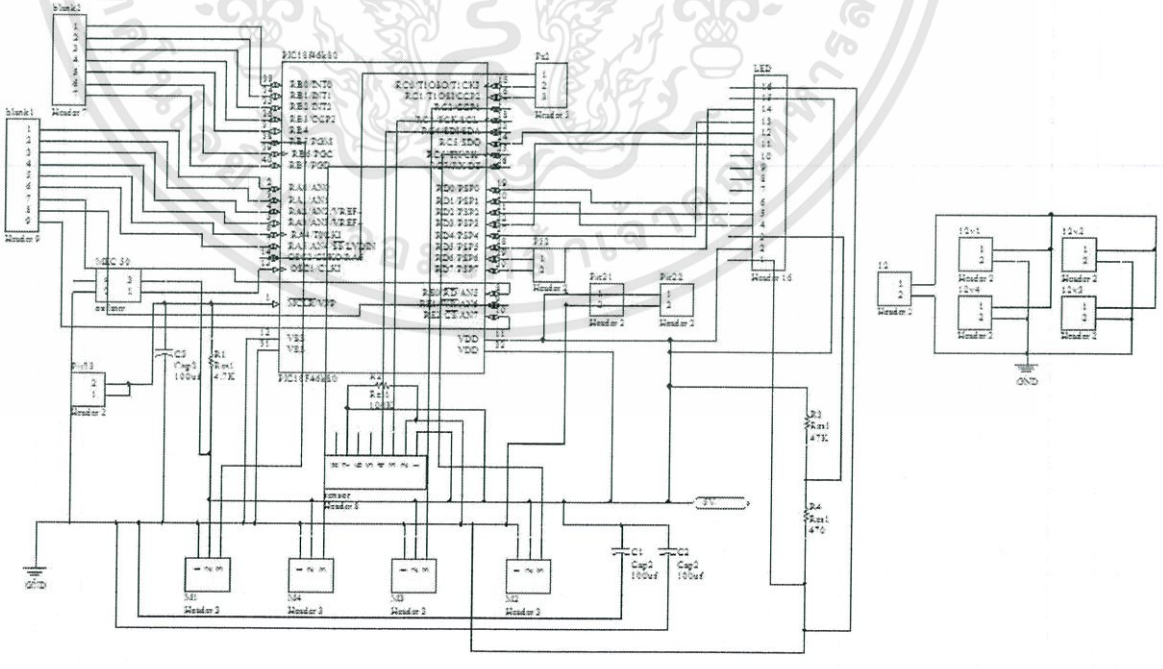
รูปที่ 3.14 เซนเซอร์ MPU-6050

-ไมโครคอนโทรเลอร์ PIC 18F46K80



รูปที่ 3.15 PIC 18F46K80

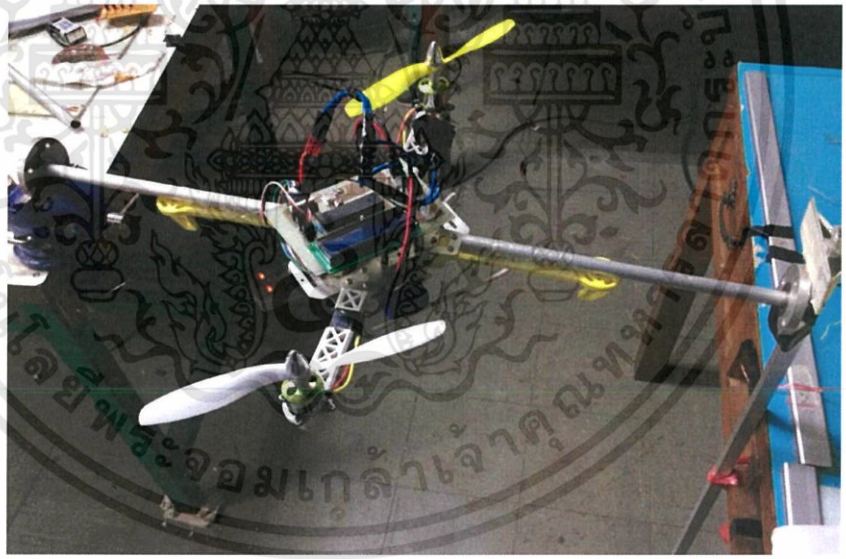
3.6 วงจร ที่ออกแบบ



รูปที่ 3.16 วงจรที่ออกแบบ

3.7 ตัวเครื่องQuad Copter

รูปที่ 3.17 ตัวเครื่องที่ประกอบเสร็จ



รูปที่ 3.18 การต่ออุปกรณ์ทำการทดลองการควบคุมด้วยระบบ PID

บทที่4
ผลการทดลอง

4.1 การทดลองการปรับค่า ระบบ PID (Proportional-Integral-Derivative)

จากทฤษฎีบทที่ 2.7 ด้วยวิธีการการปรับแต่งด้วยมือ จากการทดลองทำให้ได้ค่าผลการทดลองที่เหมาะสมดังนี้

ค่า Gain	ค่าที่ได้จากการทดลอง
Kp	0.007
Ki	0.005
Kd	0.04

ตารางที่ 4.1 ผลการทดลองหาค่า Gain ที่เหมาะสม

-เนื่องจากใช้ค่า dt ในโปรแกรมมีค่ามาก ดังนั้นค่า Gain ต่างๆที่ใช้เลยมีค่าน้อย

-ค่าKp มีค่าน้อยเพราะป้องกันไม่ให้ระบบไม่เสถียรเนื่องจากการปรับความเร็วของมอเตอร์ปรับได้ไม่ทันโปรแกรม ผู้ทดลองจึงปรับให้มีผลของค่า Kp น้อยๆ

4.2 ผลการทดลองการควบคุมการทรงตัวของเฮลิคอปเตอร์ด้วยระบบ PID

การควบคุมการทรงตัวของเฮลิคอปเตอร์ด้วยระบบ PID (Proportional-Integral-Derivative) โดยการรบกวนระบบทำให้เฮลิคอปเตอร์ไปอยู่ในระดับองศาต่างๆ แล้วให้เฮลิคอปเตอร์ทำการปรับสมดุลกลับไปยังองศาที่กำหนดโดยอัตโนมัติจากผลการทดลองได้ผลการทดลองตามตารางดังต่อไปนี้

ผลการทดลอง		
องศาที่กำหนด (องศา)	การรบกวนระบบ (องศา)	เวลา (วินาที)
140	100	5.40
	120	5.20
	140	0.00
	160	4.80
	180	4.50
	200	3.10
	220	3.50
	240	4.00
	260	4.40
เวลาเฉลี่ย		4.36
160	100	5.30
	120	4.40
	140	3.90
	160	0.00
	180	4.40
	200	3.90
	220	4.10
	240	4.70
	260	4.40
เวลาเฉลี่ย		4.39
180	100	4.40
	120	4.00
	140	4.20
	160	3.40
	180	0.00
	200	3.40
	220	3.50
	240	5.40
	260	5.40
เวลาเฉลี่ย		4.21

ตารางที่ 4.2 ผลการทดลองการควบคุมการทรงตัวของเฮลิคอปเตอร์ด้วยระบบ PID

200	100	3.30
	120	3.20
	140	3.90
	160	3.50
	180	3.20
	200	0.00
	220	4.00
	240	3.10
	260	3.10
เวลาเฉลี่ย		3.41
220	100	5.60
	120	4.70
	140	4.30
	160	4.20
	180	3.60
	200	2.90
	220	0.00
	240	3.00
	260	3.90
เวลาเฉลี่ย		4.03

ตารางที่ 4.2 ผลการทดลองการควบคุมการทรงตัวของเฮลิคอปเตอร์ด้วยระบบ PID (ต่อ)

บทที่5
วิเคราะห์และสรุปผลการทดลอง

วิเคราะห์และสรุปผลการทดลอง

5.1 Kalman filter

Kalman filter ที่ใช้สามารถแก้ปัญหาการสั่นของเซนเซอร์ได้ส่วนหนึ่ง แต่ความเร็วสูงๆซึ่งการสั่นมากขึ้นยังมีความคลาดเคลื่อนขององศาอยู่ประมาณ 5-10 องศา

5.2 การควบคุมด้วยระบบ PID (Proportional-Integral-Derivative)

PID (Proportional-Integral-Derivative) สามารถใช้ควบคุมการทรงตัวของเฮลิคอปเตอร์ 4 ใบพัดได้ ซึ่งเมื่อทำการรบกวนระบบเฮลิคอปเตอร์จะกลับคืนสู่องศาที่กำหนดได้อย่างรวดเร็ว ตามตารางที่ 4.1 ทำให้เฮลิคอปเตอร์สามารถทรงตัวได้ โดยขึ้นอยู่กับค่า PIDที่เหมาะสม

ในการสร้างเฮลิคอปเตอร์ 4 ใบพัด ยังมีความไม่สมบูรณ์ตรงที่ Kalman filter ที่ใช้ไม่ใช้ค่ามาลที่สมบูรณ์คือค่า Gain ที่ใช้เป็นค่าคงที่ซึ่งจะให้ค่าที่ “ดีพอ” แต่ถ้าเราเปลี่ยนไปใช้ค่าKalman gain เลยซึ่งเป็นค่าที่มีของทางหลักการสถิติ และค่าสามารถปรับตัวเองได้เพื่อให้ได้ค่าที่ “ดีที่สุด” และในส่วนของการปรับ PID อาจจะยังไม่สมบูรณ์เนื่องจากความไม่เป็นเชิงเส้นของระบบ ซึ่งอาจสามารถแก้ได้โดยการใช้ Fuzzy มาแก้ปัญหาตรงนี้ได้

เอกสารอ้างอิง

[1] Hutchinson, C. E.(1984): *The Kalman Filter Applied to Aerospace and Electronic Systems*. IEEE Transactions on Aerospace and Electronic Systems, Vol.20, No.4, July 1984, pp.500-504.

[2] Gelb, A. (Editor).(1996): *Applied Optimal Estimation*. The M.I.T. Press - Fourteenth Printing, Cambridge, Massachusetts, and London, England 1996.

[3] น.ต.ดร. กฤษฎา แสงเพ็ชร์ส่อง.(2547). แนวคิดพื้นฐานและ หลักการทางานของ Kalman Filter Algorithm,วารสาร โรงเรียนนายเรือ, ตุลาคม – ธันวาคม พ.ศ. 2547,หน้า 37 – 48.

[4] Chul woo Kang and Chan Gook Park.(2009). Attitude estimation with Accelerometer and Gyros using fuzzy tuned Kalman filter. Proceedings of European control conference 2009. Budapest, Hungary, August 23 – 26

[5] Jorge Miguel Brito Domingues.(2009), Quadrotor prototype, Instituto superior technico, University Tecnica de Lisboa, 2009.

[6] Jean-Jacques E. Slotine, Weiping Li. (1991). Applied Nonlinear Control, ISBN: 0-13-040049-1, Prentice-Hall, Inc.

[7] น.ต.กัปตัน เตียวตระกูล. ทบทวนความรู้แบบ PID. วารสารกรมอิเล็กทรอนิกส์ทหารเรือ ปีที่ 15 ฉบับที่ 16

[8] A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. (http://www.starlino.com/imu_guide.html)

[9] PID controller From Wikipedia, the free encyclopedia (http://en.wikipedia.org/wiki/PID_controller)

ภาคผนวก

โปรแกรมที่ใช้งาน

```
#include <copter.h>
#include <math.h>
#include <mpu.h>
#use I2C(master,sda=PIN_C4,scl=PIN_C3)

#include <lcd16x2.c>
//////////////////PWM//////////////////////
unsigned int16 fall1=0;    //time falling ccp1
unsigned int16 fall2=0;    //time falling ccp2
unsigned int16 fall3=0;    //time falling ccp3
unsigned int16 fall4=0;    //time falling ccp4

void set_duty1(float duty)
{
if(duty<=92)
  {
    fall1=((duty*65536)/100);
  }
  else
  {
    fall1=60000;
  }

}
void set_duty2(float duty)
{

  if(duty<=92)
  {
    fall2=((duty*65536)/100);
  }
  else
  {
    fall2=60000;
  }
}
void set_duty3(float duty)
{

  if(duty<=92)
  {
    fall3=((duty*65536)/100);
  }
  else
  {
    fall3=60000;
  }
}
void set_duty4(float duty)
{

  if(duty<=92)
  {
    fall4=((duty*65536)/100);
  }
  else
  {
    fall4=60000;
  }
}

#INT_TIMER1
void rising(void)
{
  if(fall1>0){
  output_high(PIN_C2);}
  if(fall2>0){
  output_high(PIN_C6);}
  if(fall3>0){
  output_high(PIN_C7);}
```

```
    if(fall4>0){
    output_high(PIN_B5);}
 }
 #INT_CCP2
 void CCP2_isr()
 {
    output_low(PIN_C2);
 }
 #INT_CCP3
 void CCP3_isr()
 {
    output_low(PIN_C6);
 }
 #INT_CCP4
 void CCP4_isr()
 {
    output_low(PIN_C7);
 }
 #INT_CCP5
 void CCP5_isr()
 {
    output_low(PIN_B5);
 }
 ////////END PWM//////////////

 ///////////FUNCTION I2C/////////////////
 int write(unsigned char Control, unsigned
 char data)
 {
    i2c_start();
    i2c_write(MPU6050_ADDRESS);
    i2c_write(control);
    i2c_write(data);
    i2c_stop();
    return 0;
 }
 int read(unsigned char control, unsigned
 char *data)
 {
    i2c_start();
    i2c_write(MPU6050_ADDRESS);
    i2c_write(control);
    i2c_start();
    i2c_write(MPU6050_ADDRESS | 0x01);
    *data=i2c_read(0);
    i2c_stop();
    return 0;
 }
 //////END FUNCTION
 I2C//////////////////////

 ///////CONFIG MPU//////////////////

 void setupMPU()
 {
    write(MPU6050_RA_SMPLRT_DIV, 0x01);

    write(MPU6050_RA_CONFIG, 0x03);
    write(MPU6050_RA_GYRO_CONFIG,
 0b00001000);
      write(MPU6050_RA_ACCEL_CONFIG,
 0b00001000);

    write(MPU6050_RA_FF_THR, 0x00);
    write(MPU6050_RA_FF_DUR, 0x00);
    write(MPU6050_RA_MOT_THR, 0x00);
    write(MPU6050_RA_MOT_DUR, 0x00);
    write(MPU6050_RA_ZRMOT_THR, 0x00);
    write(MPU6050_RA_ZRMOT_DUR, 0x00);
    write(MPU6050_RA_FIFO_EN, 0x00);
    write(MPU6050_RA_I2C_MST_CTRL, 0x00);
    write(MPU6050_RA_I2C_SLV0_ADDR,
 0x00);
    write(MPU6050_RA_I2C_SLV0_REG, 0x00);
    write(MPU6050_RA_I2C_SLV0_CTRL,
 0x00);
```

```
    write(MPU6050_RA_I2C_SLV1_ADDR,
0x00);
    write(MPU6050_RA_I2C_SLV1_REG, 0x00);
    write(MPU6050_RA_I2C_SLV1_CTRL,
0x00);
    write(MPU6050_RA_I2C_SLV2_ADDR,
0x00);
    write(MPU6050_RA_I2C_SLV2_REG, 0x00);
    write(MPU6050_RA_I2C_SLV2_CTRL,
0x00);
    write(MPU6050_RA_I2C_SLV3_ADDR,
0x00);
    write(MPU6050_RA_I2C_SLV3_REG, 0x00);
    write(MPU6050_RA_I2C_SLV3_CTRL,
0x00);
    write(MPU6050_RA_I2C_SLV4_ADDR,
0x00);
    write(MPU6050_RA_I2C_SLV4_REG, 0x00);
    write(MPU6050_RA_I2C_SLV4_DO, 0x00);
    write(MPU6050_RA_I2C_SLV4_CTRL,
0x00);
    write(MPU6050_RA_I2C_SLV4_DI, 0x00);

   write(MPU6050_RA_I2C_SLV0_DO, 0x00);
   write(MPU6050_RA_I2C_SLV1_DO, 0x00);
   write(MPU6050_RA_I2C_SLV2_DO, 0x00);
   write(MPU6050_RA_I2C_SLV3_DO, 0x00);
   write(MPU6050_RA_I2C_MST_DELAY_CTRL,
0x00);
   write(MPU6050_RA_SIGNAL_PATH_RESET,
0x00);
   write(MPU6050_RA_MOT_DETECT_CTRL,
0x00);
   write(MPU6050_RA_USER_CTRL, 0x00);
   write(MPU6050_RA_PWR_MGMT_1,
0b00000010);
   write(MPU6050_RA_PWR_MGMT_2, 0x00);
   write(MPU6050_RA_FIFO_R_W, 0x00);
}
////////////////////////////////////////////////

//////////////////KALMAN/////////////////////
signed int16 ACCEL_XOUT;
signed int16 ACCEL_YOUT;
signed int16 ACCEL_ZOUT;
float ACCEL_FXOUT;
float ACCEL_FYOUT;
float ACCEL_FZOUT;

signed int16 GYRO_XOUT;
signed int16 GYRO_YOUT;
signed int16 GYRO_ZOUT;
float GYRO_FXOUT;
float GYRO_FYOUT;
float GYRO_FZOUT;
float GYRO_OUT;
float Axz=0;
float Ayz=0;
float RxAcc=0;
float RyAcc=0;
float RzAcc=0;
float RxEst=0;
float RyEst=0;
float RzEst=0;
float RzGyro;
#define w 15   //weight KALMAN

void Get_Accel_Values()
{
int16 ACCEL_XOUT_H=0;
int16 ACCEL_XOUT_L=0;
int16 ACCEL_YOUT_H=0;
int16 ACCEL_YOUT_L=0;
int16 ACCEL_ZOUT_H=0;
int16 ACCEL_ZOUT_L=0;
float r;
```

```
  read(MPU6050_RA_ACCEL_XOUT_H,
&ACCEL_XOUT_H);
  read(MPU6050_RA_ACCEL_XOUT_L,
&ACCEL_XOUT_L);
  read(MPU6050_RA_ACCEL_YOUT_H,
&ACCEL_YOUT_H);
  read(MPU6050_RA_ACCEL_YOUT_L,
&ACCEL_YOUT_L);
  read(MPU6050_RA_ACCEL_ZOUT_H,
&ACCEL_ZOUT_H);
  read(MPU6050_RA_ACCEL_ZOUT_L,
&ACCEL_ZOUT_L);

  ACCEL_XOUT =
~(((ACCEL_XOUT_H<<8)|ACCEL_XOUT_L)-1);
  ACCEL_YOUT =
~(((ACCEL_YOUT_H<<8)|ACCEL_YOUT_L)-1);
  ACCEL_ZOUT =
~(((ACCEL_ZOUT_H<<8)|ACCEL_ZOUT_L)-1);

  ACCEL_FXOUT=(float)ACCEL_XOUT/8192;
  ACCEL_FYOUT=(float)ACCEL_YOUT/8192;
  ACCEL_FZOUT=(float)ACCEL_ZOUT/8192;

r=sqrt((ACCEL_FXOUT*ACCEL_FXOUT)+(ACCE
L_FYOUT*ACCEL_FYOUT)+(ACCEL_FZOUT*A
CCEL_FZOUT));
  RxAcc=ACCEL_FXOUT/r;
  RyAcc=ACCEL_FYOUT/r;
  RzAcc=ACCEL_FZOUT/r;
}

void Get_Gyro_Values()
{
int16 GYRO_XOUT_H=0;
int16 GYRO_XOUT_L=0;
int16 GYRO_YOUT_H=0;
int16 GYRO_YOUT_L=0;
int16 GYRO_ZOUT_H=0;
int16 GYRO_ZOUT_L=0;

  read(MPU6050_RA_GYRO_XOUT_H,
&GYRO_XOUT_H);
  read(MPU6050_RA_GYRO_XOUT_L,
&GYRO_XOUT_L);
  read(MPU6050_RA_GYRO_YOUT_H,
&GYRO_YOUT_H);
  read(MPU6050_RA_GYRO_YOUT_L,
&GYRO_YOUT_L);
  read(MPU6050_RA_GYRO_ZOUT_H,
&GYRO_ZOUT_H);
  read(MPU6050_RA_GYRO_ZOUT_L,
&GYRO_ZOUT_L);

  GYRO_XOUT =
~(((GYRO_XOUT_H<<8)|GYRO_XOUT_L)-1);
  GYRO_YOUT =
~(((GYRO_YOUT_H<<8)|GYRO_YOUT_L)-1);
  GYRO_ZOUT =
~(((GYRO_ZOUT_H<<8)|GYRO_ZOUT_L)-1);

  GYRO_FXOUT=(float)GYRO_XOUT/65.5;
  GYRO_FYOUT=(float)GYRO_YOUT/65.5;
  GYRO_FZOUT=(float)GYRO_ZOUT/65.5;

GYRO_OUT=sqrt((GYRO_FXOUT*GYRO_FXOU
T)+(GYRO_FYOUT*GYRO_FYOUT)+(GYRO_FZ
OUT*GYRO_FZOUT));
  GYRO_FXOUT=GYRO_FXOUT/GYRO_OUT;
  GYRO_FYOUT=GYRO_FYOUT/GYRO_OUT;
  GYRO_FZOUT=GYRO_FZOUT/GYRO_OUT;
}

void kalman()
{
```

```
float RxGyro,RyGyro;
float r=0;

   Axz=atan2(RxEst,RzEst);
   Ayz=atan2(RyEst,RzEst);
   Axz=Axz+GYRO_FYOUT*0.02;
   Ayz=Ayz+GYRO_FXOUT*0.02;

RxGyro=(sin(Axz))/(sqrt(1+(cos(Axz)*cos(Axz)*t
an(Ayz)*tan(Ayz))));
RyGyro=(sin(Ayz))/(sqrt(1+(cos(Ayz)*cos(Ayz)*
tan(Axz)*tan(AXZ))));
   if(RzEst<0){
      RzGyro=-1*(sqrt(1-(RxGyro*RxGyro)-
(RyGyro*RyGyro)));
      }
   else if(RzEst>0){
      RzGyro=sqrt(1-(RxGyro*RxGyro)-
(RyGyro*RyGyro));
      }
   RxEst=(RxAcc+(RxGyro*w))/(w+1);
   RyEst=(RyAcc+(RyGyro*w))/(w+1);
   RzEst=(RzAcc+(RzGyro*w))/(w+1);

r=sqrt((RxEst*RxEst)+(RyEst*RyEst)+(RzEst*RzE
st));
   RxEst=RxEst/r;
   RyEst=RyEst/r;
   RzEst=RzEst/r;
}
////////////////////////////////////////////////
///
void main()
{
int de=0;          //delay for kalman
int dela=0;
//int16 xx=170;
//int16 yy=170;
int force=55;      //duty max of motor
int16 degreeX=140;
int16 degreeY=180;
int start=24;      //duty start motor
int b=0;           //step
float DAyz=0;
float DAxz=0;

//////////////////PID//////////////////////////
float NAyz=0;
float cy=0;
float erY=0;
float pre_erY=0;      //precious-error
float integralY=0;    //integral
float dvY=0;          //derivative
float NAxz=0;
float cx=0;
float erX=0;
float pre_erX=0;      //precious-error
float integralX=0;    //integral
float dvX=0;          //derivative
////////////////////////////////////

////////////////SET PWM///////////////

 set_tris_c(0);
 set_tris_b(0);
 enable_interrupts(GLOBAL);
 enable_interrupts(INT_TIMER1);
 enable_interrupts(INT_CCP2);
 enable_interrupts(INT_CCP3);
 enable_interrupts(INT_CCP4);
 enable_interrupts(INT_CCP5);
 setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
 set_timer1(0);
 setup_ccp2(CCP_COMPARE_INT);
 setup_ccp3(CCP_COMPARE_INT);
```

```
 setup_ccp4(CCP_COMPARE_INT);
 setup_ccp5(CCP_COMPARE_INT);

 set_duty1(start+0.2);
 CCP_2=fall1;
 set_duty2(start+0.1);
 CCP_3=fall2;
 set_duty3(start-0.1);
 CCP_4=fall3;
 set_duty4(start-0.3);
 CCP_5=fall4;
 delay_ms(9000);
//////////////////////////////////////////

 /////////check MPU////////////////////
 unsigned char data;
 lcd_init();
 read(MPU6050_RA_WHO_AM_I, &data);
 lcd_gotoxy(1,1);
 printf(lcd_putc,"%x ",data);
 delay_ms(200);
//////////////////////////////////////

 setupMPU();        //config MPU

 ///////////Begin KALMAN//////////////
 Get_Accel_Values();
 RxAcc=RxEst;
 RyAcc=RyEst;
 RzAcc=RzEst;
//////////////////////////////////

  while(true)
  {
     Get_Gyro_Values();
     Get_Accel_Values();
     kalman();
     DAyz=Ayz*57.295;
     DAxz=Axz*57.295;
     if(dela>15)
     {
        lcd_gotoxy(1,1);
        printf(lcd_putc,"%f %f \n%f
%f",cx,cy,NAxz,NAyz);
        dela=0;
     }
     dela++;

     if((start<force)&&(b==0))
     {
      set_duty1(start+0.2);
      CCP_2=fall1;
      set_duty2(start+0.1);
      CCP_3=fall2;
      set_duty3(start-0.1);
      CCP_4=fall3;
      set_duty4(start-0.3);
      CCP_5=fall4;
      start++;
      delay_ms(700);
     }
     if((start>=force)&&(b==0))
     {
       if(de>=80)
       {
              b=1;
              delay_ms(1000);
       }
       de++;
     }

     if(b==1)
       {
       set_duty1(force+0.1);
       CCP_2=fall1;
```

```
    set_duty2(force+0.1);
    CCP_3=fall2;
     set_duty3(force-0.1);
    CCP_4=fall3;
    set_duty4(force-0.3);
    CCP_5=fall4;
    b=2;
}
if(b==2)
{
 ///////////////set new angle//////////
 if(DAyz<0)
 {
   NAyz=DAyz*(-1);
 }
 else if(DAyz>=0)
 {
   NAyz=360-DAyz;
 }
 if(DAxz<0)
 {
   NAxz=DAxz*(-1);
 }
 else if(DAxz>=0)
 {
   NAxz=360-DAxz;
 }
 ////////////////////////////////////

 /////////////PID/////////////////////
 erX=degreeX-NAxz;
 integralX=integralX+(erX*0.1);
 dvX=(erX-pre_erX)/0.1;

cx=(0.007*erX)+(0.005*integralX)+(0.04*dvX);
//output=(Kp*error)+(Ki*integral)+(Kd*derivati
ve)
      pre_erX=erX;    //update

      erY=degreeY-NAyz;
      integralY=integralY+(erY*0.1);
      dvY=(erY-pre_erY)/0.1;

cy=(0.007*erY)+(0.005*integralY)+(0.04*dvY);
//output=(Kp*error)+(Ki*integral)+(Kd*derivati
ve)
      pre_erY=erY;    //update

      ////////////////////////////////////
      if(cy>=0)//m3
      {
        set_duty1(force+cy+1);
        CCP_2=fall1;
      }
       if(cx>=0)//m2
      {
        set_duty2(force+cx);
        CCP_3=fall2;
      }else if(cy<0)//m4
      {
        set_duty3(force-cy-0.5);
        CCP_4=fall3;
      }
      else if(cx<0)//m1
      {
        set_duty4(force-cx-0.5);
        CCP_5=fall4;
      }
    }   //end step b=2
  }     //end while(1)
}       //end main
```