รถเครื่องบินปีกหมุน 4 ใบพัด
QUADCOPTER CAR

เกริกชัย จันต๊ะคาด
คงศักดิ์ เพียรพิทักษ์
ภูริทัศน์ ปานสังข์
มานะชัย อดิศัยสัมพันธ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมระบบควบคุม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2557

รถเครื่องบินปีกหมุน 4 ใบพัด

QUADCOPTER CAR

เกริกชัย จันต๊ะคาด

คงศักดิ์ เพียรพิทักษ์

ภูริทัศน์ ปานสังข์

มานะชัย อดิศัยสัมพันธ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

QUADCOPTER CAR

KRERKCHAI JANTAKAD

KONGSAK PHIANPHITHAK

PURITHAD PANSANG

MANACHAI ADISAISAMPHAN

THIS THESIS IS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN CONTROL ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2014

ปริญญานิพนธ์ปีการศึกษา 2557

ภาควิชาวิศวกรรมการวัดและควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง รถเครื่องบินปีกหมุน 4 ใบพัด

QUADCOPTER CAR

ผู้จัดทำ

นายเกริกชัย	จันต๊ะคาด	54010126
นายคงศักดิ์	เพียรพิทักษ์	54010142
นายภูริทัศน์	ปานสังข์	54011012
นายมานะชัย	อดิศัยสัมพันธ์	54011029

.....อาจารย์ที่ปรึกษา

(ศาสตราจารย์ ดร.วันชัย ริ้วรุจา)

.....อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์เทพจิตร์ เชยโภคา)

.....อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ดร.วรรณดี เพชรมณีล้ำค่า)

รถเครื่องบินปีกหมุน 4 ใบพัด

โดย

นายเกริกชัย	จันต๊ะคาด	54010126
นายคงศักดิ์	เพียรพิทักษ์	54010142
นายภูริทัศน์	ปานสังข์	54011012
นายมานะชัย	อดิศัยสัมพันธ์	54011029

อาจารย์ที่ปรึกษา

ศาสตราจารย์ ดร.วันชัย ริ้วรุจา

ผู้ช่วยศาสตราจารย์เทพจิตร์ เชยโภคา

ผู้ช่วยศาสตราจารย์ ดร.วรรณดี เพชรมณีล้ำค่า

ปีการศึกษา 2557

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอการออกแบบและสร้างรถเครื่องบินปีกหมุนสี่ใบพัด ที่สามารถวิ่งได้ทั้งบนพื้นและบินได้ในอากาศ โดยที่ใช้หลักการทางฟิสิกส์ที่สำคัญในเรื่องของกลศาสตร์และไฟฟ้า-แม่เหล็ก โดยควบคุมผ่านคลื่นวิทยุ โดยรถเครื่องบินปีกหมุนสี่ใบพัดประกอบไปด้วยโครงรถ ใบพัด แบตเตอรี่ ไมโครคอนโทรลเลอร์ มอเตอร์แบบไร้แปรงถ่าน ไจโรสโคป ตัวควบคุมความเร็วแบบอิเล็กทรอนิกส์ เซอร์โวมอเตอร์ และชุดรับส่งสัญญาณวิทยุ จุดมุ่งหมายของการทำโครงงานนี้คือ สร้างรถเครื่องบินปีกหมุนที่สามารถบินและวิ่งกับพื้นได้จริงโดยใช้ไมโครคอนโทรลเลอร์ในการสั่งให้ระบบทำงาน และควบคุมโดยใช้สัญญาณวิทยุ สำหรับขั้นตอนการดำเนินงานนั้นเริ่มจากการออกแบบโครงรถโดยใช้โปรแกรม Solidworks ในการออกแบบ และนำแบบที่ได้ไปพิมพ์เป็นโครงรถจากเครื่องพิมพ์สามมิติ นำโครงรถที่ได้มาประกอบ ทำการติดตั้งวงจร มอเตอร์ ใบพัดลงบนโครงรถ จากนั้นทำการเขียนโปรแกรมลงบนไมโครคอนโทรลเลอร์ เพื่อทำการควบคุมระบบ ทำการทดลองวิ่งบนพื้นราบและบิน โดยใช้รีโมทบังคับวิทยุในการควบคุม

QUADCOPTER CAR

By

Mr. Krerkchai Jantakad 54010126

Mr. Kongsak Phianphithak 54010142

Mr. Purithad Pansang 54011012

Mr. Manachai Adisaisamphan 54011029

Advisors

Prof.Dr. Vanchai Riewruja

Asst.Prof. Thepjit Cheypoca

Asst.Prof.Dr. Wandee Petchmaneelumka

Academic Year 2014

ABSTRACT

This thesis presents design and build quadcopter car that provide running and flying modes controlled via radio signal. The principle of physics including mechanics and electromagnetic are used in this project. The component of quadcopter car consists of structure, propeller, battery, microcontroller, brushless motor, gyroscope, ESC, servo motor and transmitter/receiver for radio signal. The purpose of this project is to build quadcopter car that can actually run on the ground and fly in the air. Microcontroller is used to control the operating system by command via radio signal. For operating procedure, design of the frame in solidworks program and printing it by 3D printer are started. Then all electronic system, circuit board, motor and propeller are installed on the structure. After that microcontroller is programmed for control the system. Finally, the system is tested, both driving on the ground and flying in the air by control via remote control.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้ด้วยดี คณะผู้จัดทำขอกราบขอบพระคุณอาจารย์ที่ปรึกษา ศาสตราจารย์ ดร.วันชัย ริ้วรุจา ผู้ช่วยศาสตราจารย์ ดร.วรรณดี เพชรมณีล้ำค่า และผู้ช่วยศาสตราจารย์เทพจิตร์ เชยโภคา เป็นอย่างสูงที่คอยให้คำปรึกษาแนะนำวิธีการแก้ปัญหาต่างๆ ในโครงงานทั้งทางทฤษฎี และทางปฏิบัติแก่คณะผู้จัดทำมาโดยตลอด ทำให้ผู้จัดทำเข้าใจถึงที่มาของปัญหาและสามารถแก้ไขปัญหาต่างๆ ได้อย่างถูกวิธี รวมทั้งยังเอื้อเฟื้ออุปกรณ์ที่จำเป็น และความช่วยเหลืออื่นๆ ที่เป็นประโยชน์ต่อโครงงาน

ขอขอบพระคุณ รุ่นพี่ภาควิชาวิศวกรรมการวัดและควบคุม ที่ช่วยให้คำปรึกษาและช่วยแก้ไขในส่วนที่คณะผู้จัดทำยังไม่มีความเข้าใจให้เป็นอย่างดี

ขอขอบคุณเพื่อนๆ ทุกคน ที่คอยให้กำลังใจสนับสนุนอุปกรณ์ที่ขาดเหลือ รวมทั้งคอยถามไถ่ความคืบหน้าของโครงการอยู่เสมอ

สุดท้ายนี้คณะผู้จัดทำขอขอบพระคุณบิดา มารดา และครอบครัว ที่เป็นกำลังใจที่ดีตลอดมา รวมถึงการสนับสนุนในเรื่องของงบประมาณที่ขาดเหลือ ตลอดจนเป็นแรงบันดาลใจที่ทำให้โครงงานนี้สำเร็จอย่างสมบูรณ์ได้

คณะผู้จัดทำ

เกริกชัย จันต๊ะคาด
คงศักดิ์ เพียรพิทักษ์
ภูริทัศน์ ปานสังข์
มานะชัย อดิศัยสัมพันธ์

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1

บทนำ

1.1 กล่าวนำ

ปัจจุบันการพัฒนาเครื่องบินที่สามารถบินนิ่งในอากาศถูกพัฒนาขึ้นอย่างแพร่หลาย ซึ่งมีรูปแบบในการพัฒนาที่แตกต่างกัน ตามวัตถุประสงค์ของการใช้งาน หลักการทางฟิสิกส์ที่สำคัญก็จะเป็นเรื่องของกลศาสตร์และไฟฟ้า-แม่เหล็ก ในระบบการบังคับวิทยุนี้ จะมีส่วนสำคัญคือเครื่องส่งและเครื่องรับสัญญาณวิทยุ เครื่องส่งวิทยุจะทำหน้าที่สร้างคลื่นวิทยุ แล้วนำสัญญาณการควบคุมที่ได้รับจากผู้บังคับผสมเข้ากับคลื่นวิทยุที่สร้าง เมื่อเครื่องส่งคลื่นที่มีสัญญาณการควบคุมจากผู้เล่นออกไปยังผู้รับ เครื่องรับเองก็จะนำคลื่นที่ได้รับมาแยกสัญญาณการควบคุมออกมาแล้วไปบังคับอุปกรณ์ที่ทำหน้าที่ตามที่ออกแบบไว้ให้ทำงานตามที่สั่ง

เครื่องบิน 4 ใบพัด และรถบังคับ 4 ล้อ มีรูปแบบการใช้งานที่แตกต่างกัน แต่คุณสมบัติของการทำงานที่เป็นไปในทิศทางเดียวกัน ทางผู้จัดทำเล็งเห็นว่าอุปกรณ์บังคับทั้งสองประเภทสามารถนำมาประยุกต์ใช้ร่วมกันได้ จึงออกแบบเป็นรถเครื่องบิน 4 ใบพัด ที่สามารถขับเคลื่อนได้ทั้งบนพื้นดินและบนอากาศยาน ซึ่งจะเป็นอุปกรณ์การใช้งานที่ตอบสนองต่อความต้องการของผู้ใช้ที่ต้องการใช้งานทั้งสองอย่างในอุปกรณ์เพียงชิ้นเดียว มีประสิทธิภาพที่ตอบสนองต่อความต้องการและการใช้งาน ผู้จัดทำหวังเป็นอย่างยิ่งว่ารถเครื่องบิน 4 ใบพัดจะเป็นต้นแบบในการนำไปประยุกต์ใช้งานในด้านอื่นๆ ได้อีกด้วย

1.2 วัตถุประสงค์ในการทำปริญญานิพนธ์

ในการศึกษาโครงงานนี้ เป็นการพัฒนา Quadcopter ที่สามารถเคลื่อนที่บนพื้นได้ ซึ่งจะใช้ไมโครคอนโทรลเลอร์ในการควบคุมการทำงานร่วมกับตัวสัญญาณวิทยุมีวัตถุประสงค์ดังต่อไปนี้

1. เพื่อนำความรู้และทฤษฎีที่ได้ศึกษา มาปรับปรุงและประยุกต์ให้ใช้งานได้จริง
2. เพื่อเรียนรู้หลักการทำงาน การควบคุมสั่งการทำงานโดยการเขียนโปรแกรมลงในตัวไมโครคอนโทรลเลอร์
3. เพื่อฝึกการพัฒนาความคิด การวางแผน และความรอบคอบในการปฏิบัติงาน
4. สร้าง Quadcopter ที่สามารถเคลื่อนที่กับพื้นด้วยได้จริง

1.3 ขอบเขตของโครงงาน

โครงงานนี้ผู้จัดทำจำเป็นที่จะต้องกำหนดขอบเขตในการศึกษา เพื่อให้มีแนวทางในการดำเนินงานให้สำเร็จตามขอบเขตที่ได้วางไว้ดังนี้

1. สร้างชุดควบคุมซึ่งประกอบไปด้วย ตัวรับ-ส่ง สัญญาณวิทยุ และไมโครคอนโทรลเลอร์ รวมถึงการเขียนโปรแกรมคำสั่ง

2. สร้างโครงสร้าง Quadcopter และล้อในการเคลื่อนที่กับพื้นให้สามารถบินได้และเคลื่อนที่กับพื้นได้จริง

3. สามารถบังคับผ่านรีโมทวิทยุให้ Quadcopter สามารถลอยนิ่งในอากาศ หรือเคลื่อนที่ตามต้องการได้ รวมถึงศึกษาปัจจัยที่มีผลต่อการบิน

1.4 ผลที่คาดว่าจะได้รับ

การจัดทำโครงงานในครั้งนี้ ถือว่ามีส่วนช่วยเสริมสร้างศักยภาพในด้านต่างๆ และอาจจะมีส่วนที่จะสามารถนำไปประยุกต์ใช้กับการทำงานในอนาคตได้ ซึ่งผลที่คาดว่าจะได้รับมีดังต่อไปนี้

1. ได้เรียนรู้การออกแบบและควบคุมโดยการเขียนโปรแกรมของไมโครคอนโทรลเลอร์
2. ได้รู้ถึงการออกแบบส่วนประกอบต่างๆ ของ Quadcopter
3. เข้าใจหลักการทำงานของ Quadcopter รวมถึงการบังคับการบินในลักษณะต่างๆ
4. ได้ฝึกทักษะการทำงานเป็นกลุ่ม การแก้ปัญหาและการบริหารเวลา

1.5 รายละเอียดของปริญญานิพนธ์

ในปริญญานิพนธ์ฉบับนี้ประกอบด้วย 5 บท ดังนี้

บทที่ 1 บทนำ กล่าวถึงที่มา วัตถุประสงค์ในการทำปริญญานิพนธ์ ขอบเขตของโครงงาน ประโยชน์ที่คาดว่าจะได้รับ และรายละเอียดของปริญญานิพนธ์

บทที่ 2 ทฤษฎีและความรู้ที่เกี่ยวข้อง หลักการควบคุมการบิน เครื่องส่งสัญญาณวิทยุ เครื่องรับสัญญาณวิทยุ พลศาสตร์การเคลื่อนที่สำหรับการบิน เซอร์โวมอเตอร์ และแบตเตอรี่

บทที่ 3 การออกแบบโครงสร้างและการควบคุม กล่าวถึงหลักการออกแบบ ขั้นตอนการออกแบบโครงสร้าง การออกแบบการเคลื่อนที่บนพื้นราบ การใช้งานบอร์ด MultiWii

บทที่ 4 การทดลองและผลการทดลอง การพิมพ์ชิ้นงานจากเครื่องพิมพ์สามมิติ การประกอบชิ้นงาน เครื่องพิมพ์สามมิติ การทดลองบังคับทิศทางการเคลื่อนที่ของรถ RC การทดลองบังคับทิศทางการเคลื่อนที่ของ Quadcopter

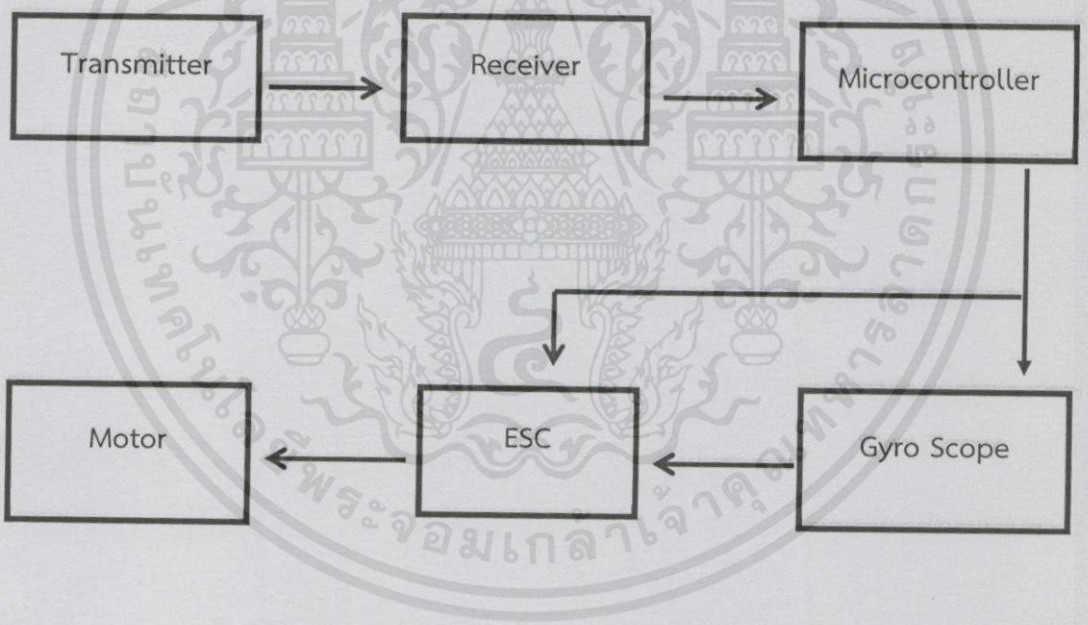
บทที่ 5 บทวิจารณ์และสรุป สรุปผลการทดลอง ปัญหาที่พบและแนวทางการแก้ไข ผลที่คาดว่าจะได้รับจากโครงงาน และข้อเสนอแนะและแนวทางในการค้นคว้าพัฒนา

บทที่ 2

ทฤษฎีและความรู้ที่เกี่ยวข้อง

หลักการควบคุมนั้นเริ่มโดยการบังคับคันโยกจากรีโมทบังคับ (Transmitter) เพื่อส่งสัญญาณไปเข้าสู่ตัวรับสัญญาณ (Receiver) จากนั้นจะส่งสัญญาณคำสั่งเข้าสู่ Microcontroller เพื่อทำการประมวลผล เมื่อประมวลผลเสร็จ ESC (Electronic Speed Controller) จะรับอินพุตจาก Microcontroller ซึ่ง ESC จะควบคุมความเร็วในการหมุนของมอเตอร์ให้หมุนช้าหรือหมุนเร็วขึ้นอยู่กับอินพุตที่ ESC รับเข้ามา

ในส่วนของการบินจะมี Gyro Scope คอยทำหน้าที่วัดความเอียงของรถเครื่องบินปีกหมุน 4 ใบพัด หากตัวรถเอียง Gyro Scope จะทำการตรวจเช็คแล้วบังคับอินพุตที่เข้า ESC ให้ไปปรับความเร็วรอบมอเตอร์



รูปที่ 2.1 หลักการทำงานของรถเครื่องบินปีกหมุน 4 ใบพัด

2.1 หลักการควบคุมการบิน

การควบคุมรถเครื่องบินปีกหมุน 4 ใบพัด ทำได้โดยการเปลี่ยนความเร็วของใบพัดซึ่งทำให้แรงบิดและแรงยกของแต่ละใบพัดเปลี่ยนไป โดยได้กำหนดสัญลักษณ์และความหมายดังต่อไปนี้

1. Ω หมายถึง ความเร็วขณะนั้น

2. Δ หมายถึง ความเร็วที่เปลี่ยนไป

3. Ö หมายถึง การเร่งความเร็วในแนวดิ่ง (Throttle)

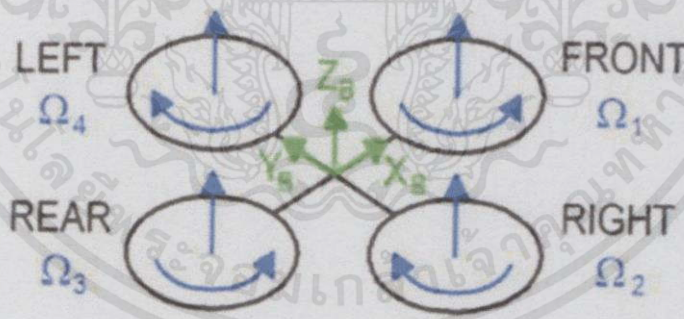
4. R หมายถึง การเอียงตัวไปทางขวา (Roll)

5. P หมายถึง การเอียงตัวไปข้างหน้า (Pitch)

หลักการควบคุมรถเครื่องบินปีกหมุน 4 ใบพัดในแบบต่างๆ ทำได้ดังนี้

2.1.1 Hovering

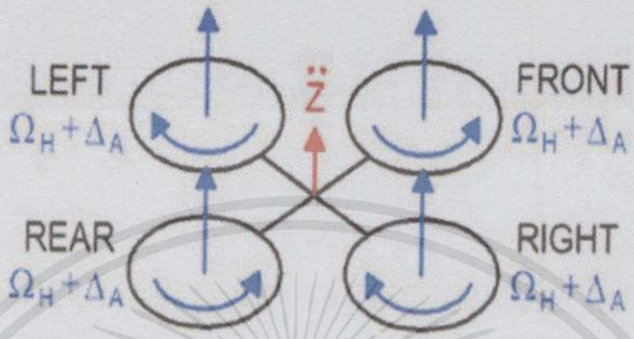
การลอยตัวเฉยๆ ทำได้โดยควบคุมให้ความเร็วใบพัดทั้ง 4 ตัวให้มีความเร็วที่เท่ากันเพื่อสร้างแรงบิด (Torque) และหักล้างแรงบิด จากรูปที่ 2.2 จะเห็นว่าใบพัดจะหมุนกันคนละทิศทาง ใบพัดหน้าและหลัง จะหมุนตามเข็ม ใบพัดซ้ายและขวาจะหมุนทวนเข็ม ทำให้เครื่องบินไม่หมุนตัว



รูปที่ 2.2 การบังคับแบบ Hovering

2.1.2 Throttle

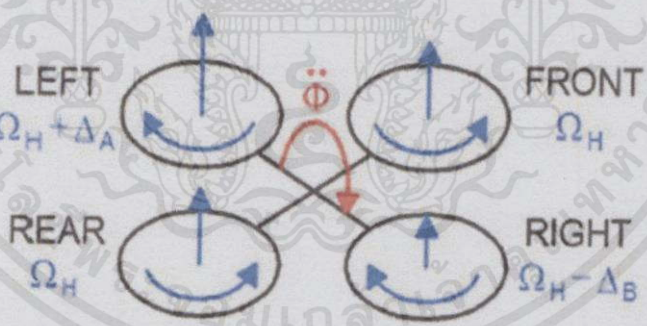
การเร่งความเร็วให้ตัวรถบิน ขึ้น-ลง จากรูปที่ 2.3 ใบพัดทั้งสี่ใบจะต้องเพิ่มความเร็วทุกใบพัดที่เท่ากัน ทำให้เครื่องบินลอยตัวขึ้นได้



รูปที่ 2.3 การบังคับแบบ Throttle

2.1.3 Roll

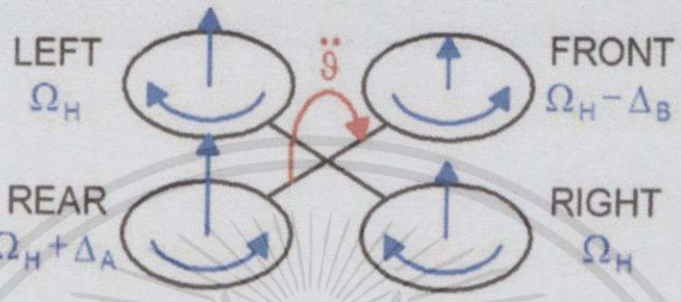
การเอียงตัวซ้าย-ขวา จากรูปที่ 2.4 ใบพัดหน้า (Front) หลัง (Rear) จะความเร็วเท่าเดิม แต่ความเร็วใบพัดซ้าย (Left) จะหมุนเร็วขึ้นทิศทางนี้จะยกตัวใบพัดขวา (Right) จะช้าลงทิศทางนี้จะตกลงจึงทำให้เกิดการเอียงตัวไปทางขวาได้ส่วนเอียงตัวซ้ายก็ใช้วิธีคล้ายกัน



รูปที่ 2.4 การควบคุมแบบ Roll

2.1.4 Pitch

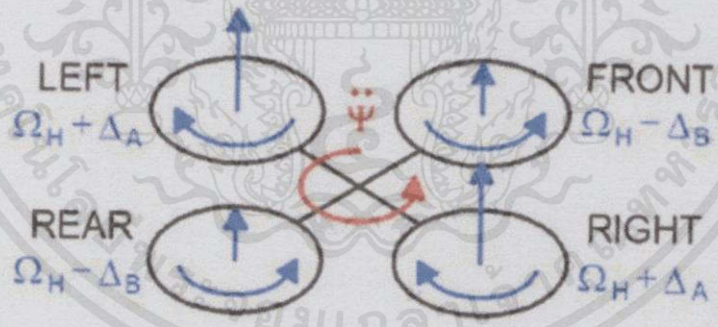
เอียงหน้าและหลังมีลักษณะคล้ายๆ กับการ Roll แต่เปลี่ยนเป็นใบพัดซ้าย-ขวา จะมีความเร็วคงที่ แต่ความเร็วใบพัดหลังจะหมุนเร็วขึ้นทำให้ด้านหลังยก ใบพัดหน้าจะหมุนช้ากว่า ทำให้ด้านหน้าเอียงลง จึงทำให้เครื่องบินเอียงไปข้างหน้า



รูปที่ 2.5 การควบคุมแบบ Pitch

2.1.5 Yaw

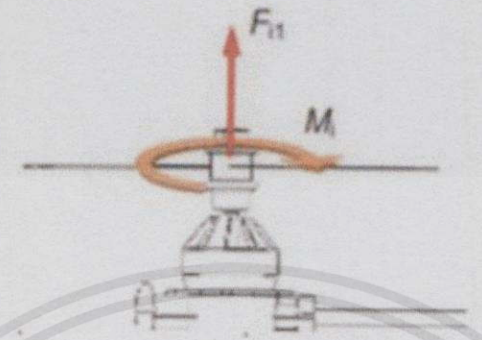
การหมุนตัวจะต้องให้ความเร็วใบพัดหน้า-หลังมากกว่า ความเร็วใบพัดซ้าย-ขวา เพื่อให้แรงบิดด้านซ้ายหรือขวามากกว่าจึงทำให้เครื่องบินหมุนตัวได้



รูปที่ 2.6 การควบคุมแบบ Yaw

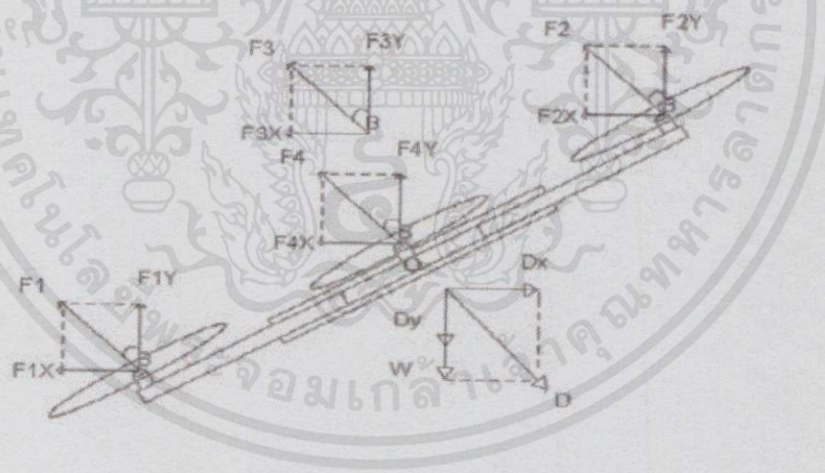
2.2 พลศาสตร์การเคลื่อนที่สำหรับการบิน

เมื่อมอเตอร์หมุนจะทำให้เกิดแรงยกที่มีทิศทางตามแรง F ซึ่งจะอธิบายได้ดังต่อไปนี้



รูปที่ 2.7 แรงยกเมื่อเกิดการหมุนของใบพัด

ในการเคลื่อนที่ของเฮลิคอปเตอร์จะมีทั้งหมด 3 แกน คือแกน X, Y และ Z แต่ถ้าพิจารณาในเรื่องสมดุลแล้วแรงยกจะถูกแบ่งออกเป็นในแนวแกน X และแกน Y



รูปที่ 2.8 แรงยกเมื่อเกิดการเคลื่อนที่

จากกฎของนิวตัน $\Sigma F = ma$ (2.1)

เมื่อ m = มวล

จะได้ $F1 + F2 + F3 + F4 - D - W = ma$ (2.2)

รวมแรงในแนวแกน X จะได้ $F1X + F2X + F3X + F4X - DX = ma_x$ (2.3)

แตกแรงเข้าแกน X จะได้ $F1\sin B + F2\sin B + F3\sin B + F4\sin B - Dx = ma_x$ (2.4)

หาความเร่งในแนวแกน X ได้เป็น $a_x = \frac{(F1+F2+F3B+F4)\sin B - Dx}{m}$ (2.5)

รวมแรงในแนวแกน Y จะได้ $F1Y + F2Y + F3Y + F4Y - DY = ma_y$ (2.6)

แตกแรงเข้าแกน Y จะได้ $F1\cos B + F2\cos B + F3\cos B + F4\cos B - W - Dy = ma_y$ (2.7)

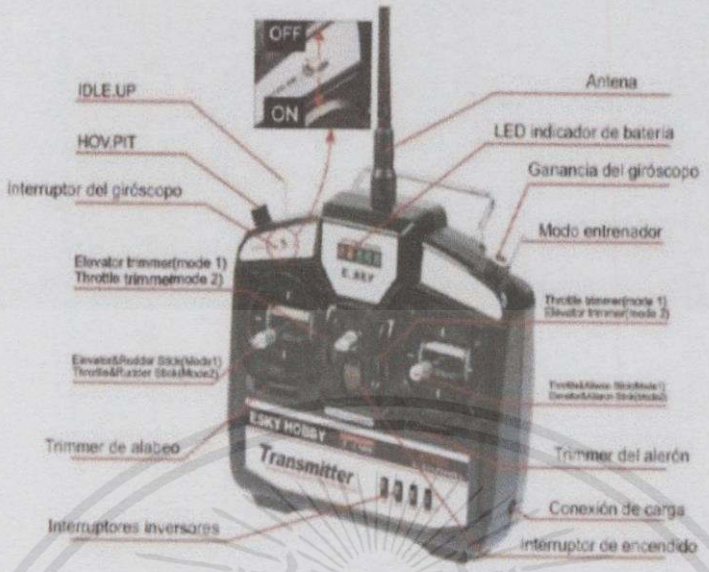
หาความเร่งในแนวแกน Y ได้เป็น $a_y = \frac{(F1+F2+F3+F4)\cos B - W - Dy}{m}$ (2.8)

2.3 เครื่องส่งสัญญาณวิทยุ

FLY SKY FS-T6 เป็นรีโมทบังคับวิทยุที่เหมาะสำหรับเครื่องบิน มีระบบโมดูลแยกแบบ 2.4 GHz มีเครื่องรับสัญญาณ 6 CH ส่งสัญญาณได้ 6 CH สามารถใช้ได้กับรถเครื่องบินปีกหมุน 4 ใบพัด ที่ต้องใช้สัญญาณควบคุมหลาย CH เพื่อเปิดโหมดต่างๆ เครื่องส่งสัญญาณวิทยุ (Transmitter) จะมีคันโยกบังคับ 2 ช่อง ซึ่งแต่ละช่องจะสามารถควบคุมได้ 2 CH ซึ่งจะผลิตความถี่ตามคำสั่งจากการโยกคันโยก อาศัยหลักการ Modulation แปลงสัญญาณส่งผ่านทางอากาศไปยังส่วนของภาครับ

2.3.1 คุณสมบัติเครื่องส่งสัญญาณวิทยุ

Channels: 6 Channels	**Sensitivity:** 1024
Model Type: Glider/Heli/Airplane	**Low Voltage Warning:** 9V
RF Range: 2.40-2.48GHz	**DSC Port:** PS2;Output: pPM
Bandwidth: 500Hz	**Power:** 12V DC (1.5AA*8)
Band: 160	**Weight:** 590g
RF Power: Less Than 20dBm	**ANT Length:** 26mm
2.4ghz System: AFHDS	**Size:** 302x190x93mm
Code Type: GFSK	**Charger Port:** Yes

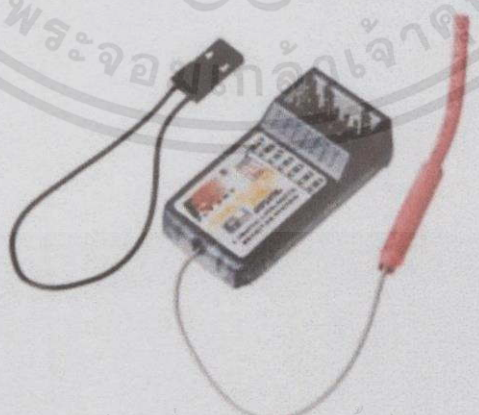


รูปที่ 2.9 ตัวส่งสัญญาณ

2.4 เครื่องรับสัญญาณวิทยุ

เครื่องรับสัญญาณวิทยุ (Receiver) คือ อุปกรณ์ที่ใช้สำหรับรับสัญญาณจากเครื่องส่ง แล้วทำการแปลงสัญญาณเพื่อไปควบคุมการทำงานของ ESC ของแต่ละสัญญาณ ประกอบด้วยเสาอากาศและแผนวงจรไฟฟ้าเพื่อคอยรับสัญญาณจากเครื่องส่ง และควบคุมให้มอเตอร์ของเฮลิคอปเตอร์ทำงานโดยผ่าน ESC

ใช้เครื่องรับสัญญาณวิทยุรุ่น FS-R6B 2.4 GHz 6CH ซึ่งมาพร้อมกับรีโมทบังคับวิทยุรุ่น FLY SKY FS-T6

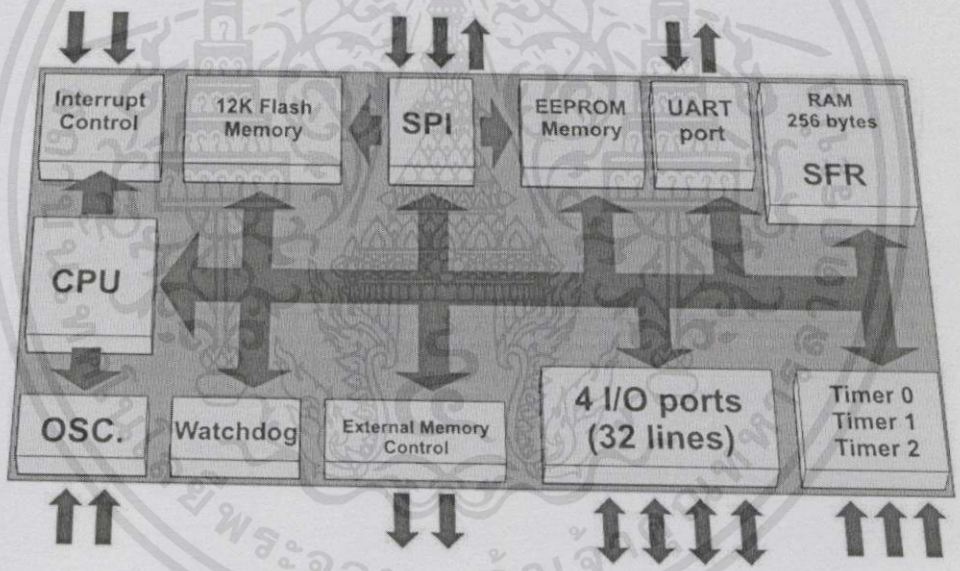


รูปที่ 2.10 เครื่องรับสัญญาณวิทยุ

2.5 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นอุปกรณ์อิเล็กทรอนิกส์ที่ใช้ควบคุมอุปกรณ์ไฟฟ้าหรือระบบอิเล็กทรอนิกส์ต่างๆ ไมโครคอนโทรลเลอร์นั้นเปรียบเสมือนคอมพิวเตอร์ขนาดเล็ก ทำงานโดยการที่ผู้ควบคุมเขียนโปรแกรมคำสั่งตามที่ต้องการ ส่งไปให้ตัวไมโครคอนโทรลเลอร์ทำงาน โดยมีส่วนประกอบดังนี้

1. หน่วยประมวลผล (CPU)
2. หน่วยความจำชั่วคราว (RAM)
3. หน่วยความจำถาวร (ROM)
4. ขาวงจรขนานทั้งแอนะล็อกและดิจิตอล ในการรับส่งข้อมูล (Digital and Analog I/O)



รูปที่ 2.11 การส่งข้อมูลของไมโครคอนโทรลเลอร์

2.5.1 คุณสมบัติของ MultiWii Standard Edition Flight Controller v2.5

MultiWii Broad v2.5 เป็นระบบคอมพิวเตอร์ควบคุมการบิน (Flight Controller) ซึ่งหมายถึงบอร์ดไมโครคอนโทรลเลอร์ขนาดเล็กที่มีเซนเซอร์ประเภทต่างๆ รวมไว้ด้วย และในปัจจุบันมีผู้พัฒนาเฟิร์มแวร์ (Firmware) สำหรับระบบดังกล่าวให้เป็น Open Source และมีหลายระบบควบคุมที่ใช้ Arduino เป็นพื้นฐานในการพัฒนา

คุณสมบัติของบอร์ด

- Small size, 35x35mm Mounting Holes
- 6 PWM Input Channels for Standard Receiver or PPM SUM Receiver
- Up to 8-Axis Motor Output
- Supported 2-Axis Gimbal and Auto Trigger Control
- FTDI/UART Port for Upload Firmware, Debug, Bluetooth Module or LCD Display
- I2C Port for Extend Sensor, I2C LCD/OLED Display or I2C-GPS NAV Board for GPS and Sonar
- Ultra Low Noise 3.3V LDO Voltage Regulator
- ATMega 328P Microcontroller
- MPU6050C 6 Axis Gyro/Accelerate with Motion Processing Unit
- HMC5883L 3-Axis Digital Magnetometer
- BMP085 Digital Pressure Sensor
- PCA9306DP1 Logic Level Converter

รูปที่ 2.12 บอร์ดไมโครคอนโทรลเลอร์ MultiWii Standard Edition Flight Controller v2.5

2.6 มอเตอร์แบบไร้แปรงถ่าน

มอเตอร์แบบไร้แปรงถ่าน (Brushless Motor) คือ มอเตอร์ชนิดที่ไม่มีแปรงถ่าน หรือมอเตอร์ซิงโครนัส 3 เฟส ที่ทำงานโดยอาศัยอุปกรณ์อิเล็กทรอนิกส์กำลังเป็นสวิตช์ในการตัดต่อกระแสที่จ่ายให้กับขดลวดมอเตอร์ โดยที่ชนิดของมอเตอร์จะพิจารณาตามลักษณะรูปคลื่นและคุณสมบัติของแรงบิดหรือทอร์ค โดยจะนิยมเรียกว่า Brushless DC Motor ในกรณีที่รูปแบบของกระแส และทอร์คของมอเตอร์ที่ใช้มีลักษณะเป็นสี่เหลี่ยม (Trapezoidal Current/Torque) มอเตอร์แบบไร้แปรงถ่าน เป็นมอเตอร์ไฟฟ้าที่ใช้พลังงานไฟฟ้ากระแสตรงและอาศัยระบบอิเล็กทรอนิกส์คอมพิวเตอร์ในการหมุนโดยภายในจะมีขดลวดทำมุมกันห่าง 120 องศา

2.6.1 การคำนวณรอบของมอเตอร์

$$Speed_M = KV \times Input\ Volt \times Efficiency$$

เมื่อ

$Speed_M$	คือ ค่าความเร็วของมอเตอร์ (มีหน่วยเป็นรอบ/นาที)
KV	คือ ค่าความเร็วรอบของการหมุนต่อ 1 Volt
Input Volt	คือ แรงดันที่ป้อนเข้ามอเตอร์
Efficiency	คือ เป็นค่าความสามารถที่มอเตอร์จะใช้งานได้จริงๆ (โดยปกติจะมีการสูญเสียไปในรูปแบบพลังงานความร้อน โดยมอเตอร์แบบไร้แปรงถ่านทั่วไป จะมีค่าประสิทธิภาพอยู่ที่ 80% หรือ 0.8)

จากโครงงานเลือกใช้มอเตอร์ BL2210/25 ขนาด 1560 KV จำนวน 4 ตัว และแบตเตอรี่ขนาด 11.1 V สำหรับการบิน

จะได้ $Speed_M = 1560 \times 11.1 \times 0.8$

$= 13,852.8$ RPM

และมอเตอร์ BL2215/25 ขนาด 950 KV จำนวน 1 ตัว สำหรับการเคลื่อนที่บนพื้นราบ (เนื่องจากต้องขับเคลื่อนทั้ง 4 ล้อ ทำให้ต้องการแรงบิดที่มาก จึงต้องให้มอเตอร์มีความเร็วรอบที่ต่ำ)

จะได้ $Speed_M = 950 \times 11.1 \times 0.8$

$= 8,436$ RPM

สำหรับมอเตอร์ที่ใช้บินจะใช้คู่กับใบพัดขนาด 7 x 4.5 นิ้ว เนื่องจาก Quadcopter มีน้ำหนักไม่มากและต้องการให้สามารถบินผาดโผนได้ จึงต้องเลือกใช้ใบพัดขนาดเล็กกับรอบของมอเตอร์สูง เพราะถ้าใช้มอเตอร์ที่มีค่า KV ต่ำจะมีแรงบิดสูง (Torque) ความเร็วรอบจึงน้อย เหมาะที่จะใช้คู่กับใบพัดขนาดใหญ่และ Quadcopter ที่มีน้ำหนักมาก หรือต้องการให้ Quadcopter บินได้นิ่งและเสถียร

2.6.2 ตารางการเลือกใช้อุปกรณ์ต่างๆ ให้สอดคล้องกัน

EMAX BL2210

Light weight (42g) brushless motors with rotating case suitable for all models of 300-400 size .Using the latest ferromagnetic materials the EMAX 2210/xx motors offer extremely high efficiency and high load capability for their weight. Includes all mounting hardware, 3 gold connectors and Prop adapter.

BL2210/25 KV=1500 Performance Chart

Model	Voltage	Propeller	RPM	Max Current	Max Trust
BL2210/25	8.1V	7X5(Thin)	11500	14.5A	590g
BL2210/25	10.6V	7X5(Thin)	13700	23A	870g
BL2210/25	10.6V	6X5.5(Thin)	15750	16A	595g
BL2210/25	8.1V	6X5.5(Thin)	12500	10A	350g

	Weight Of Model	LiPo	EMAX ESC	Propeller
TRAINER	480g./17oz.	3S 1300mAH	EMAX 18A	7X5(Thin)
3D	350g./12.5oz.	3S 1300mAH	EMAX 18A	8X3.8(Slow)
AEROBATIC	400g./14oz.	3S 1600mAH	EMAX 18A	7X6 (Thin)

รูปที่ 2.13 ข้อมูลสำหรับการเลือกอุปกรณ์ชนิดต่างๆ

2.6.3 คุณสมบัติมอเตอร์แบบไร้แปรงถ่าน BL2210/25 ขนาด 1560 KV

Specification	
No. Of cells	2-3x Li-Poly
RPM/V	1450/1560
Max. efficiency current	>75%
No load current / 10 V	0,5 A
Current capacity	2210/25 15A/60s
Dimensions	22x10 mm
Shaft diameter	3 mm
Weight	45 g
Recommended model weight	200 - 500 g

2.6.4 คุณสมบัติมอเตอร์แบบไร้แปรงถ่าน BL2215/25 ขนาด 950 KV

Specification	
No. Of cells	2-3x Li-Poly
Max. efficiency	82%
Max. efficiency current	14 - 24.5A (>75%)
No load current / 10 V	0,5 A
Current capacity	16.5 A/60s
Internal Resistance	275 mohm
Dimensions	22x15 mm
Shaft diameter	3 mm
Weight	59 g/2.08oz.
Recommended model weight	2215/25 300-900g

รูปที่ 2.14 มอเตอร์แบบไร้แปรงถ่าน

2.7 Gyro Scope

Gyro Scope เป็นตัวที่ใช้ในการตรวจสอบการหมุน ส่วน Gyro Scope สำหรับ RC Helicopter เรียกว่า RC Gyro คือ อุปกรณ์การวัดความเร็วเชิงมุม (Angular Velocity Measurement Device) และให้ค่าเอาต์พุตออกมาเป็นสัญญาณไปควบคุม Speed Control ซึ่งใช้ขับมอเตอร์ ส่วน Gyro Scope ที่เซนเซอร์ที่เป็นอุปกรณ์อิเล็กทรอนิกส์เชิงกล ซึ่งเป็นหัวใจหลักสำหรับใช้ในการวัดความเร็วเชิงมุม ซึ่งเมื่อเกิดการเอียงขึ้นจะทำให้มีความเร็วเชิงมุม ทำให้เอาต์พุตที่ออกมามีความกว้างของพัลส์ที่เปลี่ยนไป ซึ่งจะต้องป้อนสัญญาณที่เป็นอินพุตเข้าไปก่อนและตัว Gyro Scope จะทำการเก็บค่าอินพุตนั้นไว้ แล้วส่งไปที่เอาต์พุต เช่น ถ้า Gyro Scope ยังไม่มีการหมุน เมื่อทำการปรับอินพุตที่มีความกว้างเปลี่ยนไป จะทำให้เอาต์พุตที่ออกมาจาก Gyro Scope เปลี่ยนแปลงไปตามอินพุต แต่ถ้าตัว Gyro Scope มีการเอียงเกิดขึ้นก็จะทำให้สัญญาณที่เอาต์พุต มีการเปลี่ยนแปลงความกว้างของพัลส์เกิดขึ้น ซึ่งค่าความกว้างของพัลส์ที่เปลี่ยนแปลงไปนั้นจะ Oscillate อยู่ค่าอินพุต ที่ป้อนเข้ามายัง Gyro Scope เช่น เมื่อป้อนสัญญาณเป็นอินพุต ให้กับ Gyro Scope ที่มีความกว้าง 1.5 ms และตัว Gyro Scope เกิดการเอียงซ้ายขึ้นก็จะทำให้ตัวเฮลิคอปเตอร์เกิดการเอียงซ้าย ตัว Gyro Scope จะตรวจจับความเร็วเชิงมุม ซึ่งเมื่อมีการเอียงซ้ายเกิดขึ้นจะส่งสัญญาณที่มีความกว้างพัลส์มากกว่า 1.5 ms ไปที่มอเตอร์ซ้ายเพื่อทำให้มอเตอร์ซ้ายหมุนเร็วขึ้นและสร้างแรงยกให้ตัวเฮลิคอปเตอร์กลับมาอยู่ที่สภาวะสมดุล

ในโครงงานนี้ตัว Gyro Scope ได้ฝังมากับบอร์ดไมโครคอนโทรลเลอร์ MultiWii Standard Edition Flight Controller v2.5 เป็นที่เรียบร้อยแล้ว จึงสะดวกในการใช้งานเป็นอย่างมาก

2.8 ESC (Electronic Speed Controller)

ESC เป็นอุปกรณ์อิเล็กทรอนิกส์ประเภทหนึ่ง ที่ประกอบด้วยวงจรในการควบคุมความเร็วของมอเตอร์ ซึ่งสำหรับเฮลิคอปเตอร์ 4 ใบพัดนั้นจำเป็นต้องมีอุปกรณ์นี้ เพื่อใช้ในการควบคุมมอเตอร์ โดยรับสัญญาณการควบคุมมาจากตัวรับสัญญาณ ซึ่งจะสัมพันธ์กับตำแหน่งของคันโยกที่ขยับ เช่น ถ้าตำแหน่งคันโยกอยู่ที่ตำแหน่งต่ำสุด ESC จะไม่จ่ายกำลังไฟให้กับมอเตอร์ ถ้าตำแหน่งคันโยกอยู่ที่กึ่งกลาง ESC จะจ่ายกำลังไฟให้กับมอเตอร์ 50% เป็นต้น ทั้งนี้ขึ้นอยู่กับการติดตั้งโปรแกรมที่ตัววิทยุและ ESC

ESC สำหรับมอเตอร์แบบไร้แปรงถ่านนั้น ทำหน้าที่สร้างพัลส์ให้กับมอเตอร์สามเฟส เพื่อสลับเส้นแรงแม่เหล็ก ในขดลวดของมอเตอร์ให้สัมพันธ์กับการหมุน โดยเส้นแรงแม่เหล็กเหล่านี้จะผลักดันกับแม่เหล็กถาวรในมอเตอร์ ทำให้มอเตอร์หมุนอย่างต่อเนื่องตลอดเวลา

มอเตอร์แบบไร้แปรงถ่านนั้นมีหลายขนาดการกินกระแสของมอเตอร์ แต่ละตัวจึงไม่เท่ากัน และแรงดันที่ใช้กับมอเตอร์แต่ละขนาดแต่ละชนิดก็ไม่เท่ากัน ดังนั้นจึงต้องมีการระบุปริมาณกระแสและแรงดันสูงสุดของ ESC ที่จะใช้โดยดูคุณสมบัติมอเตอร์เป็นหลัก เช่น มอเตอร์กินกระแสสูงสุดที่ 15 A ใช้กับแบตเตอรี่ LiPo 2-3 เซลล์ ควรเลือกใช้ ESC ขนาด 18-20 A อย่าเลือกขนาดกระแสสูงเกินไป เพราะจะเพิ่มทั้งน้ำหนักและราคา นอกจากนี้ต้องดูคุณสมบัติของ ESC ที่ใช้ด้วยว่าใช้กับแบตเตอรี่ LiPo สูงสุดได้กี่เซลล์

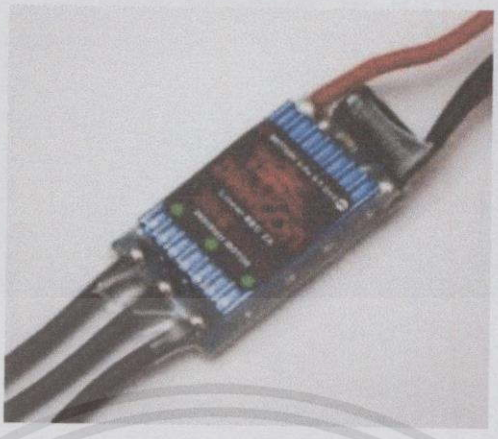
ในโครงงานนี้เลือกใช้ Brushless Multicopter ESC 30 A คู่กับมอเตอร์แบบไร้แปรงถ่าน BL ขนาด 1560 2210/25KV ในการขับเคลื่อนในอากาศ และเลือกใช้ Brushless Car ESC แบบ Reverse คู่กับมอเตอร์แบบไร้แปรงถ่าน BL ขนาด 950 2215/25KV ในการเคลื่อนที่บนภาคพื้นดิน

2.8.1 คุณสมบัติของ Dragon 30A Brushless ESC for Airplane

Specification	
Weight	28g /0.99 oz
Size	45X24X9mm
Constant current	30A Max 40A <10s
Battery	Li-Poly 2-3 cells
Break On/Off	Programmable
BEC	2A
MAX RPM	20,000 with 2212 Motors
Auto Low battery	Slow down at 3.0V/cell Lipo, cut-off at 2.9V/cell Lipo

2.8.2 คุณสมบัติของ Brushless car ESC 45 A w/reverse for Car

Specification	
Weight	60 g
Size	45X32X20 mm
Input Voltage	2-3S Lithium batteries / 4~12 Ni-xx
Constant current	45 A
Battery	Li-Poly 2-3 cells
BEC	2A/5V (Switch)

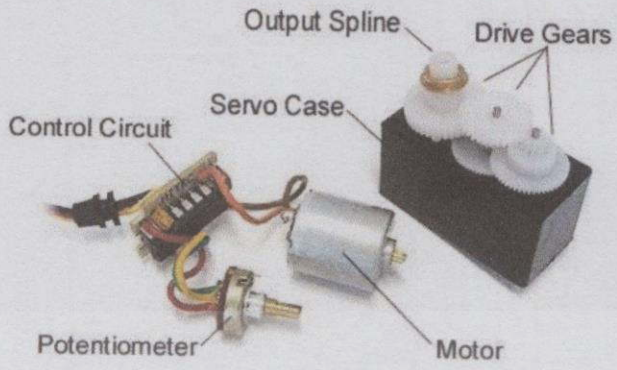


รูปที่ 2.15 Dragon 30A Brushless ESC for Airplane

รูปที่ 2.16 Brushless car ESC 45 A w/reverse for Car

2.9 เซอร์โวมอเตอร์

เซอร์โวมอเตอร์ เป็นมอเตอร์ที่ทำงานโดยใช้สัญญาณพัลส์ โดยภายในเซอร์โวมอเตอร์จะประกอบไปด้วย มอเตอร์ไฟฟ้ากระแสตรง ชุดเกียร์และส่วนควบคุม โดยจะประกอบกันอยู่ภายในชุดเดียวกัน ตัวเซอร์โวมอเตอร์จะมีสายสัญญาณ 3 เส้น คือ สายใช้งาน 1 เส้น อีก 2 เส้นจะเป็นสายสำหรับจ่ายไฟให้เซอร์โวมอเตอร์และสายสำหรับต่อลงกราวด์ ในการควบคุมเซอร์โวมอเตอร์นั้นจะทำให้หมุนไปทางซ้ายได้ 90 องศา ไปทางขวาได้ 90 องศา (180 องศา) และสามารถสั่งให้หมุนไปตามองศาที่กำหนดได้ (ในการที่จะทำให้หมุน 360 องศาจะต้องดัดแปลงแก้ไขส่วนประกอบภายใน) การใช้งานเซอร์โวมอเตอร์นั้นจะนำไปใช้ในที่ที่ต้องการความแม่นยำในเรื่ององศา หรือการหมุนไปตามองศาที่ต้องการ เพราะตัวเซอร์โวมอเตอร์เองจะมีแรงบิดค่อนข้างสูง ซึ่งในโครงงานนี้จะใช้เซอร์โวมอเตอร์ควบคุมการเลี้ยวสำหรับการเคลื่อนที่บนพื้นราบ

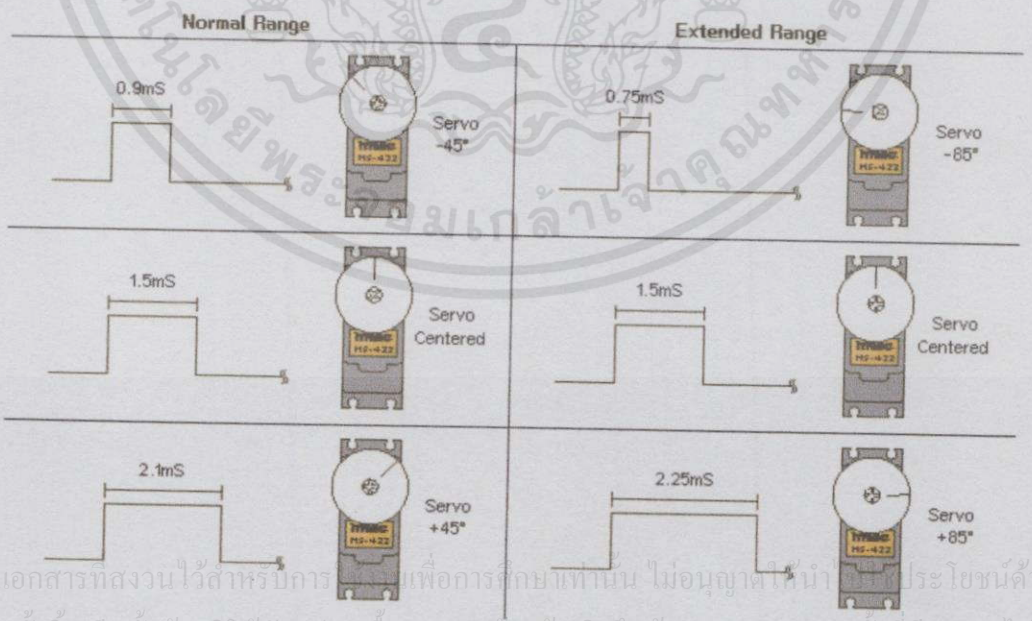


รูปที่ 2.17 ส่วนประกอบของเซอร์โวมอเตอร์

2.9.1 การควบคุมการทำงานของเซอร์โวมอเตอร์

ในการควบคุมเซอร์โวมอเตอร์นั้น ทำได้โดยอาศัยความกว้างของพัลส์ที่ทำการป้อนให้เซอร์โวมอเตอร์ โดยสัญญาณพัลส์นี้จะเป็นสัญญาณ TTL จะมีแรงดัน 5VDC และแรงดัน 0 VDC

ตัวอย่างเช่น เซอร์โวมอเตอร์หนุนไปทางขวา หรือตามเข็มนาฬิกา จะต้องสร้างสัญญาณพัลส์ความกว้างขนาด 1 ms และมีช่วงระยะห่างระหว่างพัลส์ 20 ms เซอร์โวมอเตอร์หนุนไปทางซ้าย หรือ ทวนเข็มนาฬิกา จะต้องสร้างสัญญาณพัลส์ ขนาด 2ms และมีช่วงระยะห่างระหว่างพัลส์ 20 เซอร์โวมอเตอร์อยู่ในตำแหน่งกึ่งกลาง จะต้องสร้างสัญญาณพัลส์ ขนาด 1.5 ms และมีช่วงระยะห่างระหว่างพัลส์ 20



รูปที่ 2.18 การทำงานของเซอร์โวมอเตอร์

2.9.2 คุณสมบัติของเซอร์โวมอเตอร์รุ่น Tower Pro SG91R

Modulation	Analog
Torque	4.8V: 25.0 oz-in (1.80 kg-cm)
Speed	4.8V: 0.10 sec/60°
Weight	0.32 oz (9.0 g)
Dimensions	Length: 0.91 in (23.1 mm) Width: 0.48 in (12.2 mm) Height: 1.14 in (29.0 mm)

รูปที่ 2.19 เซอร์โวมอเตอร์

2.10 แบตเตอรี่

จากโครงงานเลือกใช้แบตเตอรี่ชนิด LiPo (Lithium Polymer) ขนาด 11.1 V ซึ่งเป็นตระกูลเดียวกับแบตเตอรี่แบบ Li-ion ข้อดีของแบตเตอรี่ LiPo คือ มีน้ำหนักเบา ให้พลังงาน ความจุ และปริมาตรสูง (ซึ่งเป็นลักษณะที่ต้องการสำหรับการบิน) นอกจากนี้แบตเตอรี่แบบ LiPo ยังมีอัตราการคายประจุสูงมาก ทำให้มอเตอร์ได้รับพลังงานเพียงพอกับกำลังที่ต้องการ

2.10.1 ความแตกต่างระหว่างแบตเตอรี่แบบ LiPo และแบบ Li-ion

แบตเตอรี่แบบ Li-ion ใช้ตัวทำละลายอินทรีย์เป็นสารอิเล็กโทรไลต์ซึ่งติดไฟได้ง่ายทำให้เกิดการระเบิดได้ โดยปกติแบตเตอรี่แบบนี้จะต้องเก็บไว้ในถังโลหะ ซึ่งเป็นการเพิ่มน้ำหนักให้กับแบตเตอรี่จึงไม่ค่อยนิยมใช้กับเฮลิคอปเตอร์บังคับ ส่วนแบตเตอรี่แบบ LiPo จะใช้สารพอลิเมอร์เป็นสารอิเล็กโทรไลต์แบบแห้งคล้ายๆ กับฟิล์มพลาสติกทำให้เกิดเป็นรูปร่างและขนาดได้หลากหลาย แต่มีข้อเสียคือ แบตเตอรี่แบบนี้มีอัตราการถ่ายเทประจุช้ามากส่งผลให้อัตราการชาร์จและคายประจุต่ำ แบตเตอรี่แบบ LiPo ที่ใช้กันในปัจจุบันนี้เรียกว่าแบบ LiPo Hybrid สิ่งที่เพิ่มเข้ามาจากแบตเตอรี่ LiPo คือ การใส่เจลอิเล็กโทรไลต์เข้าไป เพื่อให้การถ่ายโอนประจุเร็วขึ้นอีกประการนั่นเอง

โดยทั่วไปเซลล์ของแบตเตอรี่แบบนี้จะห่อหุ้มด้วยแผ่นฟอยล์ แต่ละเซลล์จะให้ความต่างศักย์ 3.7 V จำนวนเซลล์ที่มาต่ออนุกรมกันจะมีประมาณ 1-4 เซลล์ โดยจะระบุไว้ที่ข้างแบตเตอรี่ว่า 1S 2S 3S หรือ 4S ซึ่งบอกว่ามีกี่เซลล์ต่อกันแบบอนุกรม (S-series) บางแบบก็จะมีการต่อเซลล์แบบขนานด้วยเพื่อเพิ่มปริมาณกระแสที่จ่ายออกมา ซึ่งในกรณีนี้ก็จะระบุตัว P (Parallel) เพื่อเพิ่มความจุขึ้นมา เช่น 3S2P ก็หมายความว่ามีเซลล์สองชุดต่อขนานกัน โดยที่แต่ละชุดมีสามเซลล์ต่ออนุกรมกันอยู่ซึ่งโดยรวมแล้วจะให้ความต่างศักย์ 3.7 x 3 = 11.1 V

2.10.2 ความจุและอัตราการคายประจุของแบตเตอรี่ LiPo

ความจุของแบตเตอรี่แบบนี้ก็ระบุด้วยค่า mAh และมีความหมายเช่นเดียวกัน ปริมาณที่สำคัญอีกอย่างคือ อัตราการคายประจุหรือบางทีเรียกว่าค่า C-rating (C ย่อมาจาก Capacity) ค่านี้จะปรากฏอยู่บนแบตเตอรี่แต่ละก้อน ตัวอย่างเช่น แบตเตอรี่ก้อนหนึ่งอาจจะระบุว่า 3S 11.1 V 25 C 2000 mAh หมายความว่าแบตเตอรี่ก้อนนี้ประกอบด้วย 3 เซลล์ต่ออนุกรมกันมีความต่างศักย์รวม 11.1 V และคายประจุได้สูงสุดถึง 25 เท่าของค่าความจุที่กำหนด นั่นคือ 25x2000 mA = 50 A ในเวลาหนึ่งชั่วโมง

สามารถคำนวณเวลาที่แบตเตอรี่สามารถจ่ายกระแสได้โดยนำค่าความจุมาหาร 60 นาทีในหนึ่งชั่วโมง จะได้ 2000/60 = 33.3 mAh/min จากนั้นคูณด้วย C-rating ได้ 833 mAh/min ซึ่งคิดเป็นเวลา 2000/833 = 2.4 นาที นั่นคือถ้าดึงกระแสออกมาด้วยอัตราสูงสุด 25 C จะใช้งานแบตเตอรี่นี้ได้เพียง 2.4 นาทีเท่านั้น

ในการเลือกแบตเตอรี่มาใช้งานนั้นควรเลือกซื้อแบตเตอรี่ที่มีค่า C สูงๆ ไว้ก่อนเท่าที่จะสามารถซื้อได้ (เนื่องจากแบตเตอรี่ที่มีค่า C สูงๆ จะมีราคาสูงตามไปด้วย) การใช้แบตเตอรี่มีค่า C ต่ำเกินไปจะทำให้มอเตอร์และ ESC เสียหายได้ บางครั้งค่า C-rating ที่กำหนดมาให้จะระบุทั้งค่าต่อเนื่องและค่าการระเบิดพลังงานออกมาในช่วงเวลาสั้นๆ (Burst) เช่น 20C Continuous/40C Bursts หมายความว่าในการใช้งานปกติสามารถจ่ายกระแสได้ต่อเนื่องที่อัตราสูงสุด 20 C แต่ถ้าในการบินที่รุนแรงซึ่งมอเตอร์ต้องการกำลังมหาศาลในเวลาอันสั้น แบตเตอรี่นี้จะจ่ายกระแสได้เต็มที่ 40 เท่าของค่าความจุในเวลาอันรวดเร็ว (โดยปกติจะอยู่ที่ 10 วินาที) นอกจากเรื่องของปริมาณและอัตราการจ่ายกระแสแล้ว การใช้แบตเตอรี่ที่มีค่า C ต่ำเกินไปจะทำให้แบตเตอรี่ร้อนมาก สำหรับผู้อ่านที่เคยเล่นจะพบว่าทุกครั้งที่นำเฮลิคอปเตอร์ลงจอด แบตเตอรี่จะร้อนและจะต้องรอให้แบตเตอรี่เย็นเสียก่อน จึงจะนำไปชาร์จได้รวมทั้งจะต้องพักมอเตอร์ และ ESC สักครู่เพื่อให้เย็นตัวลงก่อนที่จะนำเฮลิคอปเตอร์ขึ้นบินในรอบต่อไป การเปลี่ยนไปใช้แบตเตอรี่ที่มีค่า C สูงขึ้นจะทำให้แบตเตอรี่เย็นกว่าในขณะใช้งานทำให้อายุการใช้งานของแบตเตอรียาวนานขึ้น

รูปที่ 2.20 Battery 11.1 V ขนาด 3000 mA

บทที่ 3

การออกแบบโครงสร้างและการควบคุม

3.1 หลักการออกแบบ

หลักการออกแบบรถเครื่องบินปีกหมุน 4 ใบพัดนั้น พิจารณาการออกแบบจากค่าทฤษฎีของเครื่องบินเป็นหลัก เนื่องจากขนาดของมอเตอร์และใบพัดจะส่งผลต่อการบินโดยตรง ซึ่งหลักการของแรงที่ต้องใช้ยกตัวเครื่องบินนั้นจะต้องมากกว่าน้ำหนักของวัตถุอย่างน้อย 1 เท่า และการกำหนดขนาดของใบพัดนั้นก็จะส่งผลการการออกแบบโครงสร้างของล้อและโครงรถ ความกว้าง ความยาว ความสูง จากการศึกษาทฤษฎีการบินนั้นจะต้องออกแบบโครงรถเพื่อให้สามารถวางมอเตอร์ที่ขับเคลื่อนในลักษณะสี่เหลี่ยมจัตุรัส หลังจากที่ทราบขนาดในส่วนต่างๆ ของโครงรถนั้น จะสามารถออกแบบชิ้นส่วนต่างๆ ให้เหมาะสมกับอุปกรณ์ที่เลือกใช้ ซึ่งนอกจากนี้การออกแบบชิ้นส่วนให้มีความคงทนแข็งแรงนั้นจะส่งผลดีต่อการทดลองบิน ซึ่งจะเป็นการป้องกันไม่ให้ชิ้นส่วนโครงสร้างแตกเสียหายได้

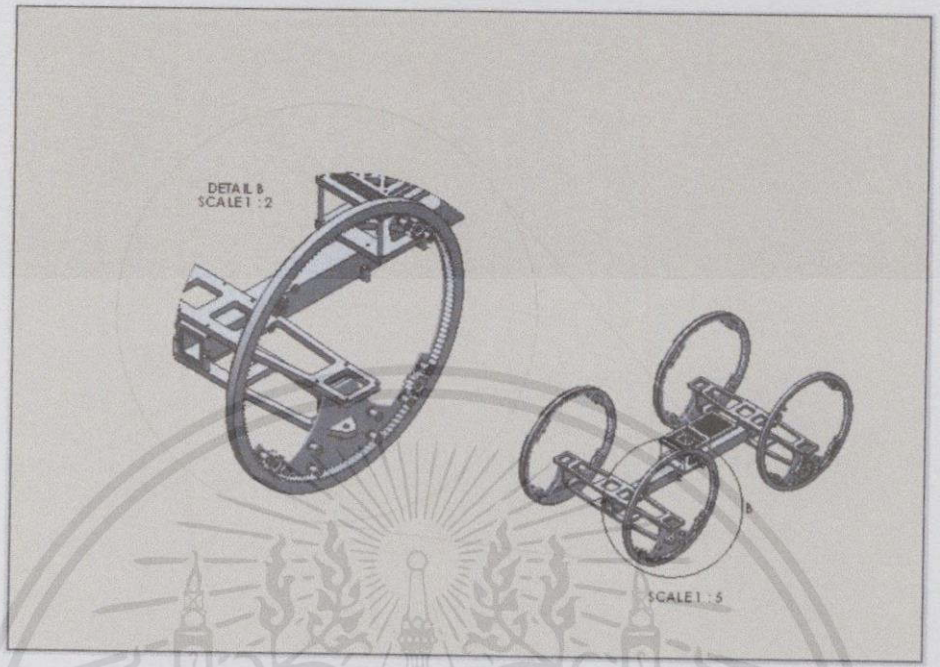
3.2 ขั้นตอนการออกแบบโครงสร้าง

3.2.1 ออกแบบโครงสร้างด้วยโปรแกรมโซลิดเวิร์ค (SolidWorks)

หลังจากที่ได้ศึกษาค้นคว้าหาข้อมูลต่างๆ ที่เกี่ยวข้องกับเฮลิคอปเตอร์, รถบังคับ และเครื่องบินปีกหมุน 4 ใบพัด อาทิเช่น ขนาดความกว้าง ความยาว ความสูงของตัวรถ รวมถึงขนาดของล้อ เพื่อที่จะเลือกขนาดมอเตอร์ และใบพัดที่เหมาะสมกับขนาดโดยรวมของตัวรถ เนื่องจากขนาดมอเตอร์สัมพันธ์กับความกว้างของใบพัด จากนั้นจึงนำข้อมูลที่ได้ทั้งหมดมาออกแบบในโปรแกรมโซลิดเวิร์ค (SolidWorks) โดยมีโครงสร้างต่างๆ ดังนี้

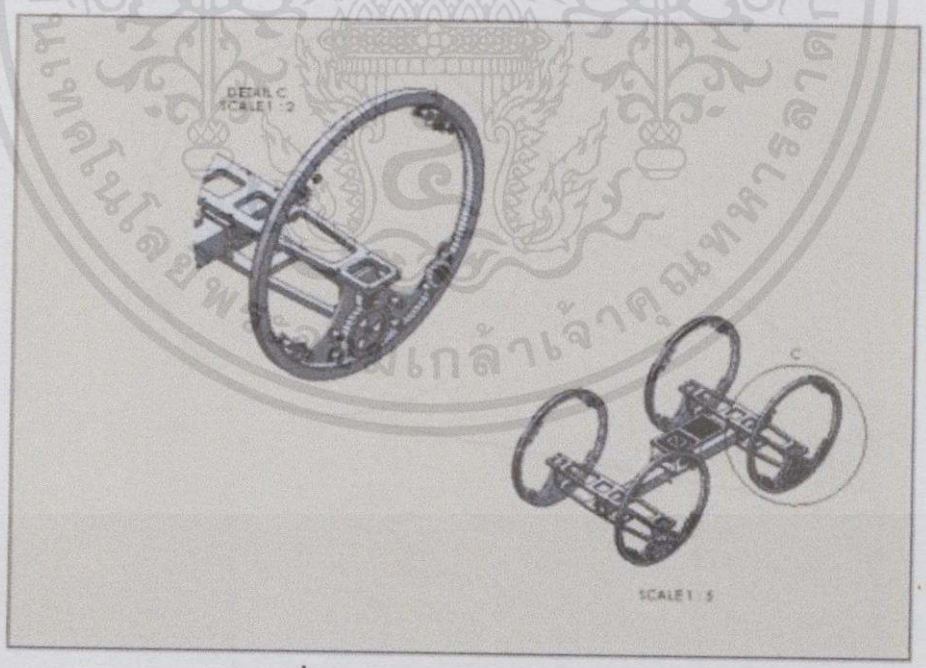
รูปที่ 3.1 โปรแกรมโวลิดเวิร์ค

ก. ล้อหน้า



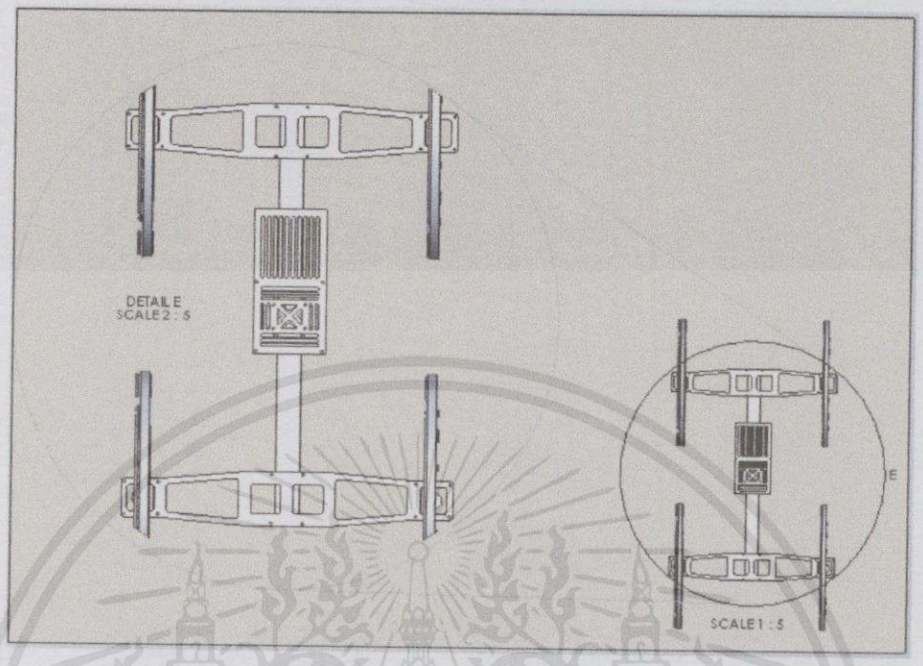
รูปที่ 3.2 แบบโครงสร้างล้อหน้า

ข. ล้อหลัง



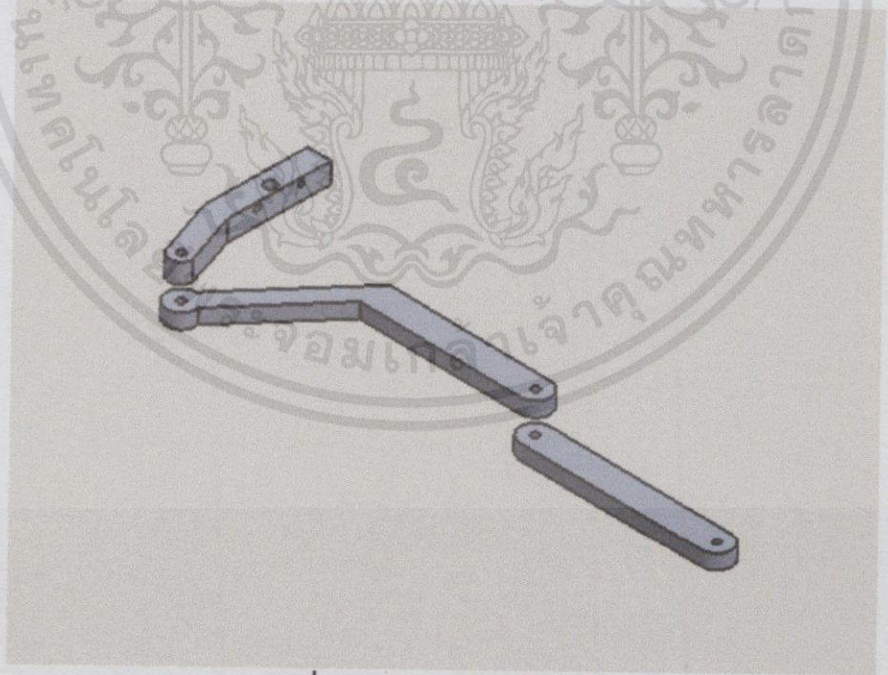
รูปที่ 3.3 แบบโครงสร้างล้อหลัง

ค. โครงสร้างฐาน

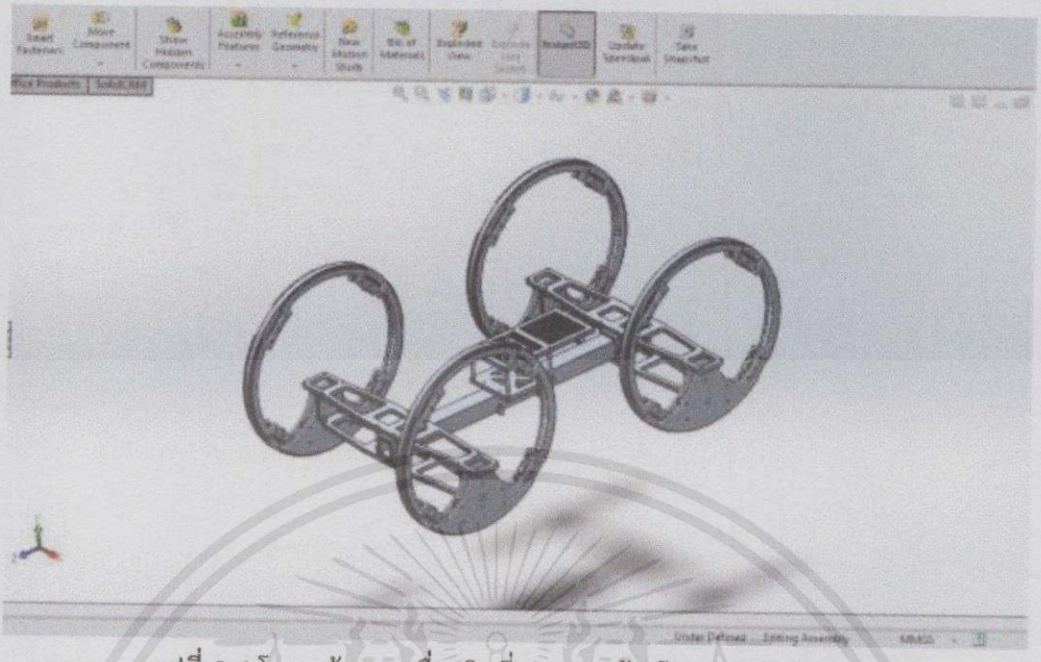


รูปที่ 3.4 แบบโครงสร้างฐาน

ง. แกนเซอร์โว



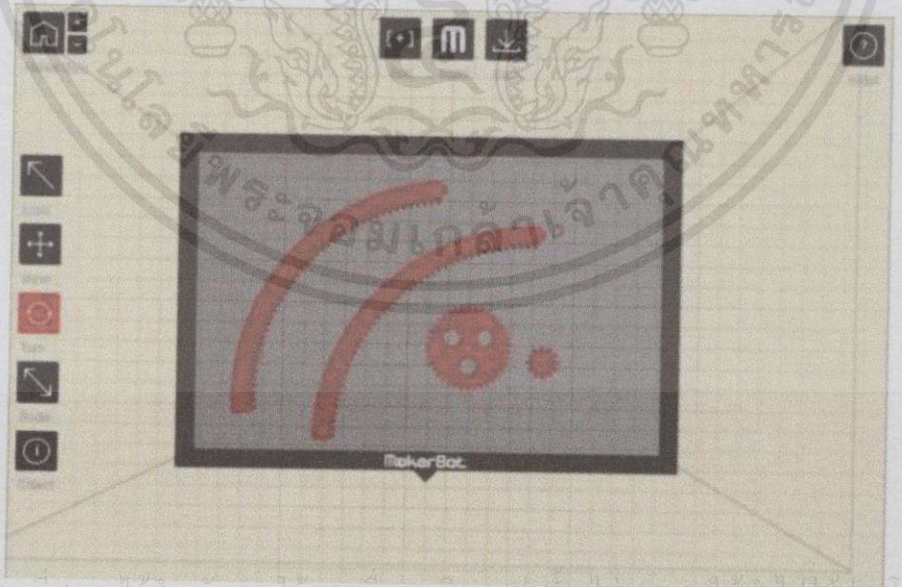
รูปที่ 3.5 แบบแกนเซอร์โว



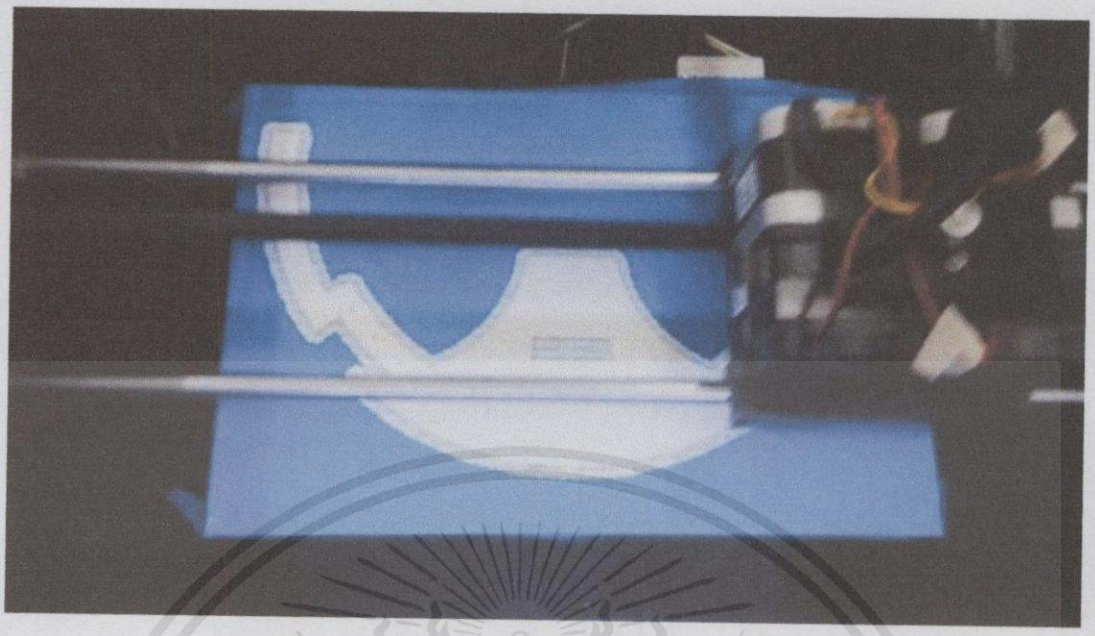
รูปที่ 3.6 โครงสร้างรถเครื่องบินที่ออกแบบด้วยโปรแกรมโซลิดเวิร์ค

3.2.2 พิมพ์ชิ้นงานด้วยเครื่องพิมพ์แบบ 3 มิติ

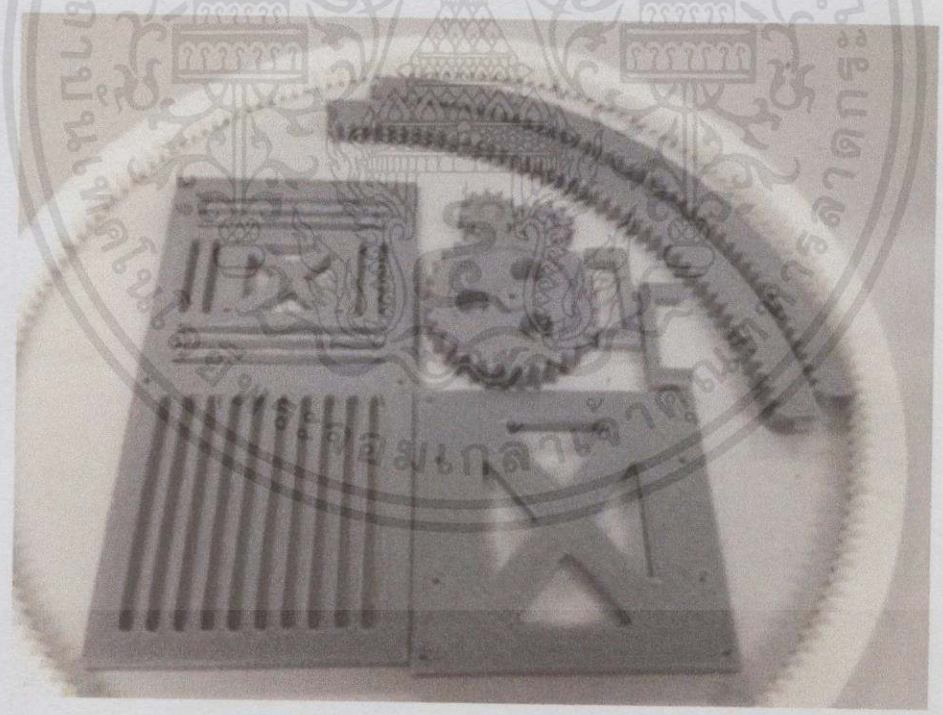
นำชิ้นส่วนที่ออกแบบโดยโปรแกรมโซลิดเวิร์ด (SolidWorks) แปลงไฟล์ข้อมูลเป็นนามสกุล *.STL จากนั้นนำไฟล์ข้อมูลที่ได้ มาแปลงเป็นไฟล์ข้อมูลนามสกุล *.x3g ด้วยโปรแกรม MakerWare แล้วส่งไฟล์ข้อมูลลง SD Card เพื่อนำไปใช้พิมพ์ชิ้นงานที่เครื่องพิมพ์แบบ 3 มิติ



รูปที่ 3.7 แปลงไฟล์ข้อมูล *.STL ให้เป็น *.x3g ด้วยโปรแกรม MakerWare



รูปที่ 3.8 ทำการพิมพ์ชิ้นส่วนด้วยเครื่องพิมพ์แบบ 3 มิติ



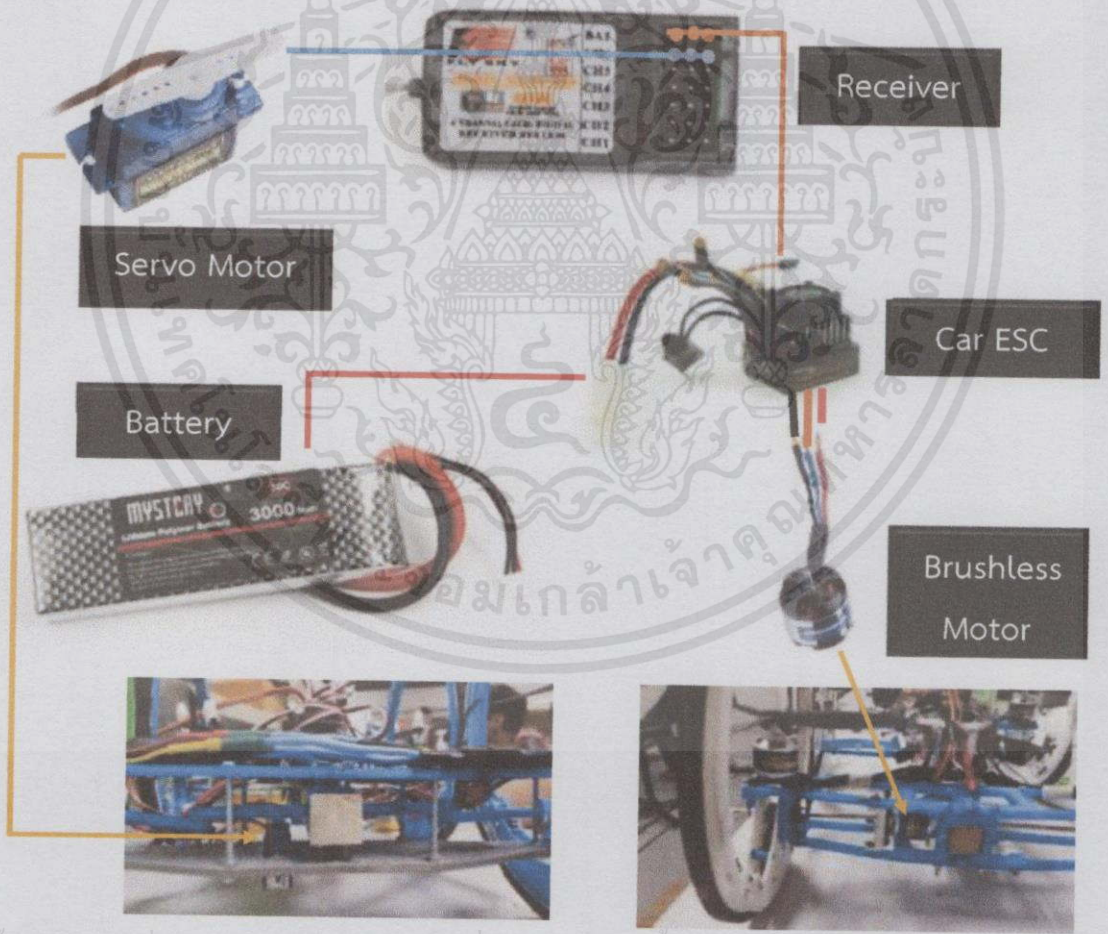
รูปที่ 3.9 ตัวอย่างชิ้นงานที่พิมพ์ออกมาจากเครื่องพิมพ์แบบ 3 มิติ

3.3 การออกแบบการเคลื่อนที่บนพื้นราบ

3.3.1 อุปกรณ์

1. มอเตอร์แบบไร้แปรงถ่าน BL2215/25 ขนาด 950 KV
2. Brushless Speed Control (45A Brushless car ESC with Reverse)
3. เซอร์โวมอเตอร์ รุ่น Tower Pro SG91R
4. แกนเซอร์โวมอเตอร์
5. แบตเตอรี่ LiPo ขนาด 11.1 V
6. ตัวส่งสัญญาณวิทยุ
7. เครื่องรับสัญญาณวิทยุ

3.3.2 วงจรควบคุมการขับเคลื่อนแนวราบ



รูปที่ 3.10 วงจรควบคุมการขับเคลื่อนแนวราบ

3.3.3 หลักการทำงานการเคลื่อนที่บนแนวราบ

มอเตอร์แบบไร้แปรงถ่านที่ทำหน้าที่ขับเคลื่อนหน้า-หลัง จะถูกต่อเข้ากับ Brushless Speed Control จากนั้น Brushless Speed Control จะถูกต่อเข้ากับเครื่องรับสัญญาณวิทยุใน Channel ที่ 6 ส่วนเซอร์โวมอเตอร์ ที่ทำหน้าที่ในการเลี้ยวซ้าย-ขวา จะถูกต่อเข้ากับเครื่องรับสัญญาณวิทยุใน Channel ที่ 5 ซึ่งทั้งหมดจะถูกควบคุมโดยรีโมทคอนโทรล ผ่านการโยกคันโยกและหมุนปุ่มควบคุมบนรีโมทคอนโทรล

3.4 การออกแบบการเคลื่อนที่ในอากาศ

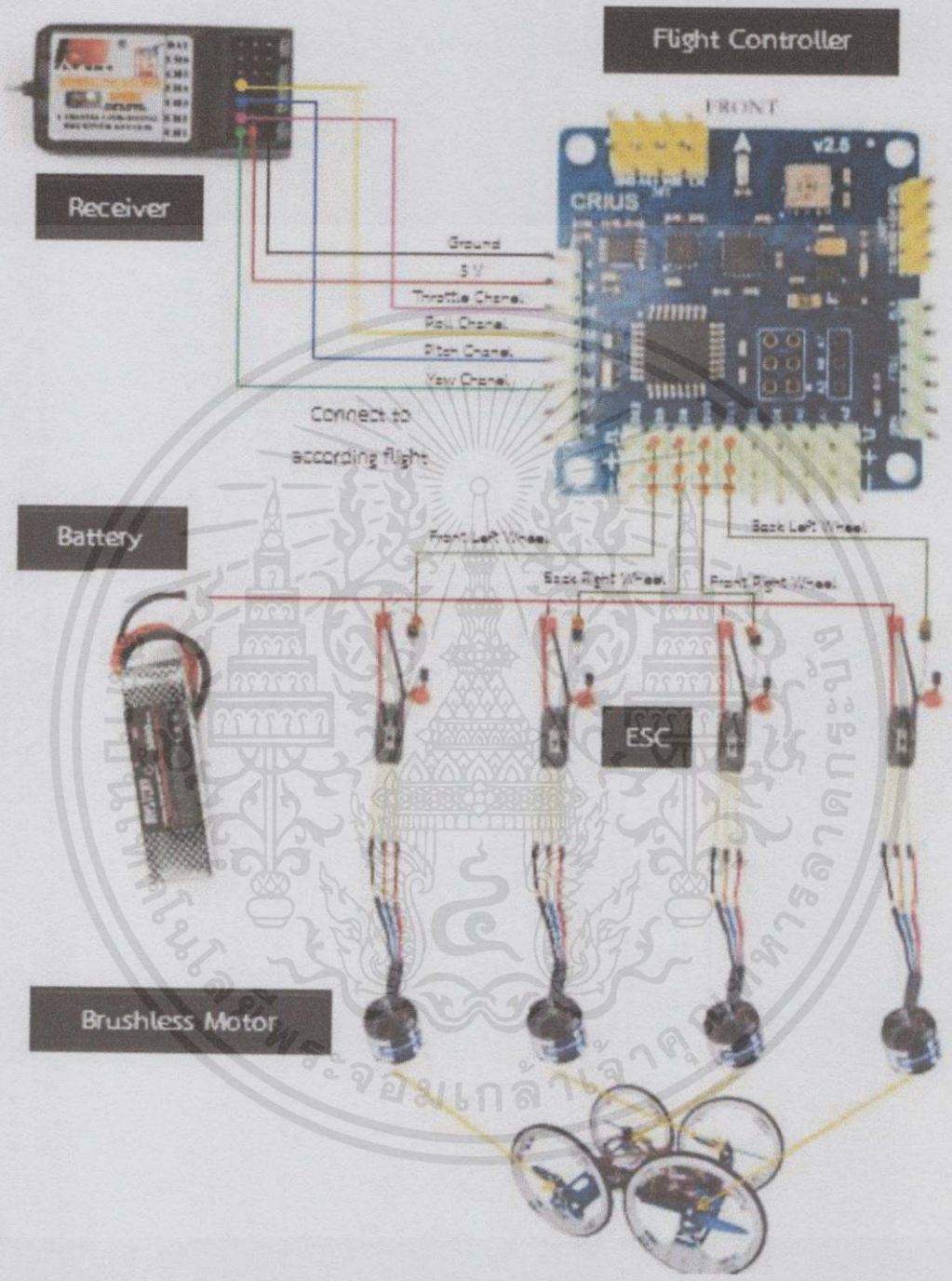
3.4.1 อุปกรณ์

1. บอร์ดไมโครคอนโทรลเลอร์ MultiWii SE v2.5
2. มอเตอร์แบบไร้แปรงถ่าน BL2210/25 ขนาด 1560 KV
3. Brushless Speed Control (30A Brushless ESC)
4. ใบพัด 3 แฉก ขนาด 7 x 4.5 นิ้ว
5. แบตเตอรี่ LiPo ขนาด 11.1 V
6. ตัวส่งสัญญาณวิทยุ
7. เครื่องรับสัญญาณวิทยุ

3.4.2 หลักการทำงานการเคลื่อนที่ในอากาศ

ควบคุมโดยโปรแกรม Arduino โดยเริ่มจากการบังคับคันโยกจากรีโมทบังคับ เพื่อส่งสัญญาณไปเข้าสู่ตัวรับสัญญาณ จากนั้นจะส่งสัญญาณคำสั่งเข้าสู่ไมโครคอนโทรลเลอร์ เพื่อทำการประมวลผลเมื่อประมวลผลเสร็จจะรับอินพุตจากไมโครคอนโทรลเลอร์ ซึ่ง ESC จะควบคุมความเร็วในการหมุนของมอเตอร์ให้หมุนช้าหรือหมุนเร็วขึ้นอยู่กับอินพุตที่ ESC รับเข้ามา ในส่วนของการบินจะมี Gyro Scope คอยทำหน้าที่วัดความเอียงของรถเครื่องบินปีกหมุน หากตัวรถเอียง Gyro Scope จะทำการตรวจเช็คแล้วบังคับอินพุตที่เข้า ESC ให้ไปปรับความเร็วรอบมอเตอร์

3.4.3 วงจรควบคุมการเคลื่อนที่ในอากาศ



รูปที่ 3.11 วงจรควบคุมการเคลื่อนที่ในอากาศ

3.4.4 การใช้งานบอร์ด MultiWii SE v2.5

3.4.4.1 ส่วนประกอบของบอร์ด

1. ATmega328P Microcontroller (5V, 16MHz)

2. MPU6050C 6-Axis Gyro/Accelerometer with Motion Processing Unit เป็นไอซีวัดความเร่ง (Accelerometer) และไจโร (Gyro) อย่างละสามแกน

3. HMC5883L 3-Axis Digital Magnetometer เป็นไอซีเข็มทิศแบบสามแกน (ใช้ระบุทิศทาง)

4. BMP085 (Barometric) Pressure Sensor เป็นไอซีวัดความดันบรรยากาศ (นำค่าที่ได้ไปคำนวณความสูงจากระดับน้ำทะเล)

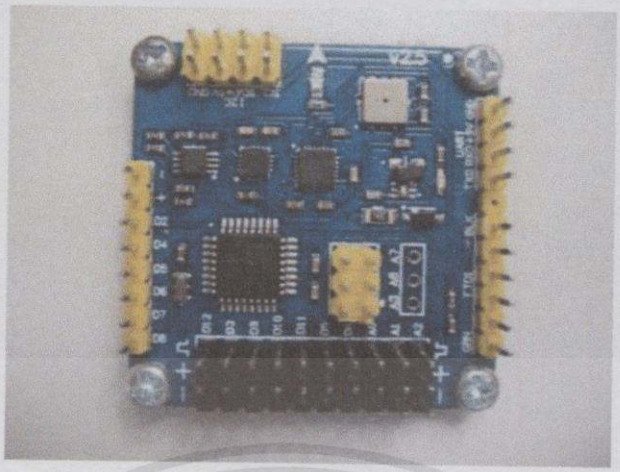
3.4.4.2 ส่วนเชื่อมต่อภายในบอร์ด

1. ESC เป็นส่วนเชื่อมต่อแบบ Pin Headers ช่องละ 3 ขา (Signal, +, -) และแต่ละช่องตรงกับขาของ Arduino ดังนี้ D12, D3, D9, D10, D11, D5, D6, A0, A1, A2 ตามลำดับ

2. ตัวรับสัญญาณคลื่นวิทยุ (Radio Receiver) เป็นแบบ Pin Headers ที่มีแถวเดียว 8 ขา ได้แก่ ขา+, -, D2, D4, D5, D6, D7, D8 ตามลำดับ (+ และ – หมายถึง +5V และ GND และโดยปกติแล้วจะใช้สำหรับป้อนแรงดันไฟเลี้ยงให้ตัวรับสัญญาณคลื่นวิทยุของรีโมท)

3. โปรแกรมชิปไมโครคอนโทรลเลอร์ (ใช้ Arduino Bootloader) โดยผ่าน Serial ไปยังคอมพิวเตอร์ ซึ่งใช้รูปแบบการเชื่อมต่อเหมือนในกรณีของโมดูล FTDI USB-to-Serial โดยเรียงขาจาก GRN ไปยัง BLK ดังนี้ ขา DTR, RXD, TXD, +5V, CTS, GND และมีส่วนเชื่อมต่ออีกหนึ่งชุด (ขา TXD, RXD, +5V, GND) สำหรับต่อกับโมดูล Bluetooth-to-Serial เพื่อใช้กำหนดค่าพารามิเตอร์หรือไว้ดูสถานะการทำงานของระบบ และต้องใช้ Baud Rate เท่ากับ 115200 (ค่านี้ถูกกำหนดไว้ใน Firmware ของ MultiWii)

4. บัส I2C ได้แก่ ขา SCL (A5), SDA (A4), +5V, GND ซึ่งมี 2 ชุด (2x4 pins), (ขา SCL และ SDA ใช้แรงดันลอจิกอยู่ที่ระดับ +5V)



รูปที่ 3.12 บอร์ด MultiWii SE v2.5

3.4.4.3 การเชื่อมต่อบอร์ด

ในกรณีที่เลือกใช้ Quadrotor แบบ "Quad-X" (4 ใบพัด)

1. ต่อกับตัวรับสัญญาณ (โดยใช้สัญญาณ 5 หรือ 6 ช่อง) : D2, D4, D5, D6, D7, D8 จะตรงกับ Throttle, Roll, Pitch, Yaw, Aux1, Aux2 ตามลำดับ

- CH1=Roll ⟶ D4
- CH2=Pitch ⟶ D5
- CH3=Throttle⟶ D2
- CH4=Yaw ⟶ D6
- CH5=Aux1 ⟶ D7
- CH6=Aux2 ⟶ D8 (optional)

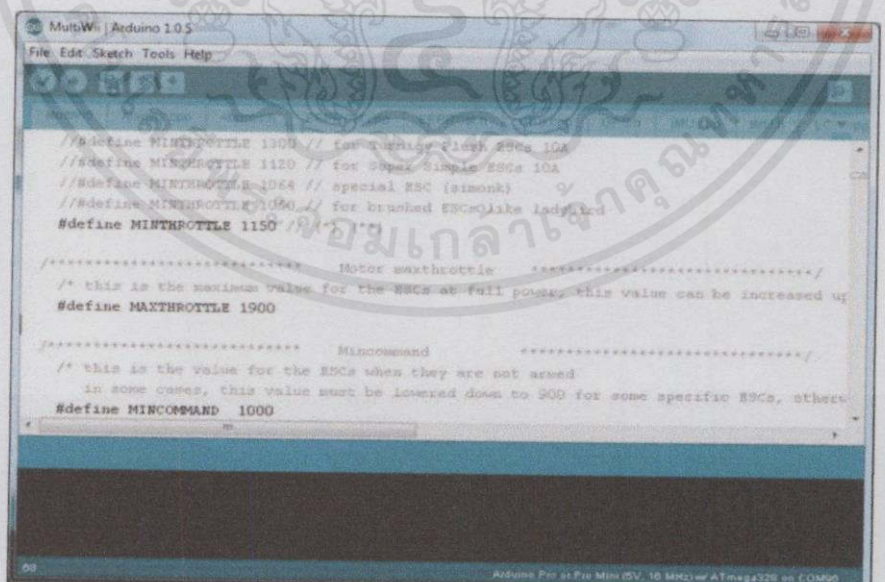
2. ต่อกับ ESC สำหรับมอเตอร์แต่ละตัว ทั้งหมด 4 ชุด

- D3 = Front Left Motor (CW) สำหรับมอเตอร์ด้านหน้าซ้าย ใบพัดหมุนตามเข็มนาฬิกา
- D9 = Rear Right Motor (CW) สำหรับมอเตอร์ด้านหลังขวา ใบพัดหมุนตามเข็มนาฬิกา
- D10=Front Right Motor(CCW) สำหรับมอเตอร์ด้านหน้าขวา ใบพัดหมุนทวนเข็มนาฬิกา
- D11 = Rear Left Motor (CCW) สำหรับมอเตอร์ด้านหลังซ้าย ใบพัดหมุนทวนเข็มนาฬิกา

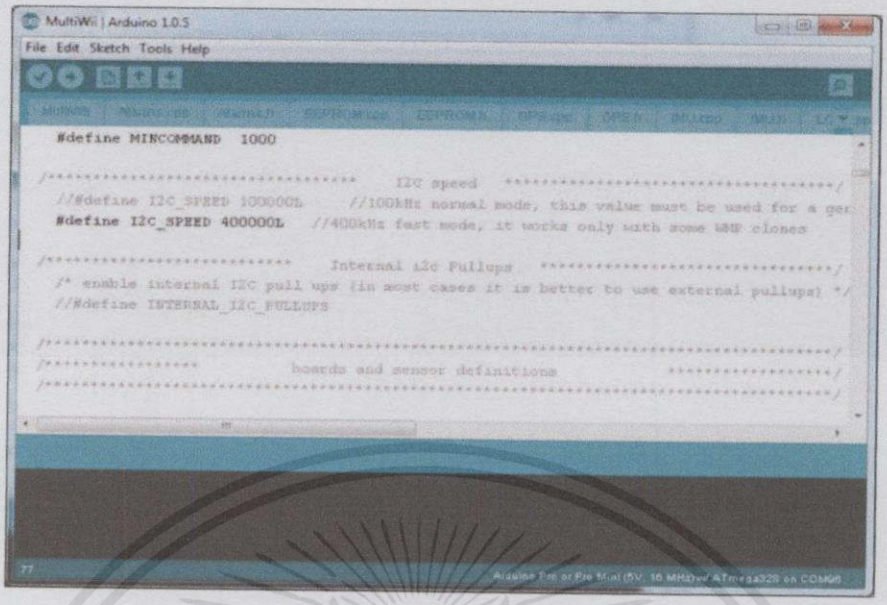
3.4.5 การคอมไพล์และอัพโหลดเฟิร์มแวร์สำหรับ MultiWii v2.3 ด้วย Arduino IDE

เริ่มต้นด้วยการเปิดโปรแกรม Arduino IDE เพื่อใช้งาน (ในการทดลองได้ใช้ Arduino IDE v1.0.5) และเปิดไฟล์ Arduino Sketch ที่มีชื่อว่า "MultiWii.ino" ของ MultiWii จากนั้นจะเห็นว่ามีไฟล์อีกหลายๆ ไฟล์ ถัดไปเป็นการปรับเลือกค่าให้ตรงกับบอร์ด MultiWii ที่เลือกใช้ โดยให้ดูและแก้ไขในไฟล์ config.h ของ MultiWii

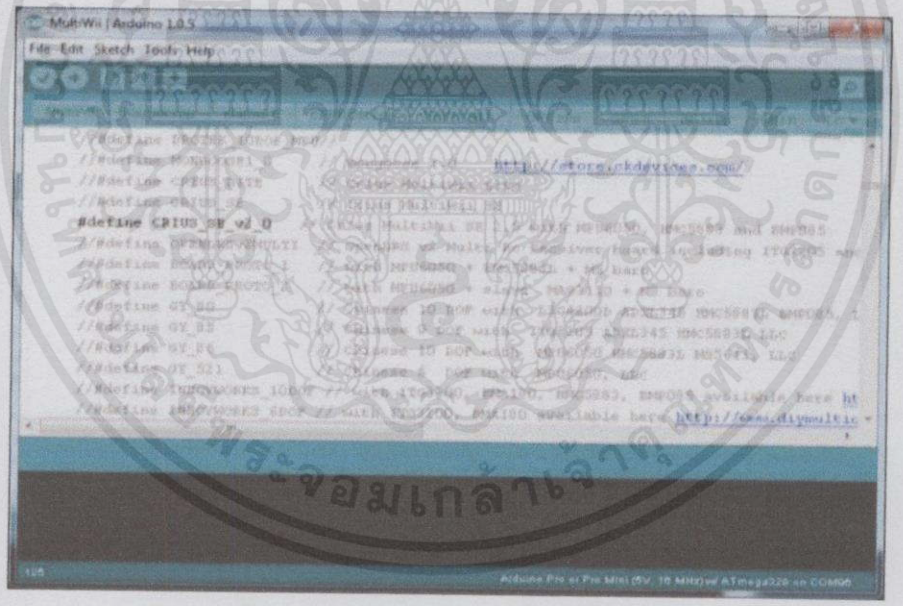
รูปที่ 3.13 การเลือกใช้ QUADX โดยแก้ไขโค้ดในไฟล์ config.h ของ MultiWii



รูปที่ 3.14 การกำหนดค่า MINTHROTTLE และ MAXTHROTTLE ในไฟล์ config.h

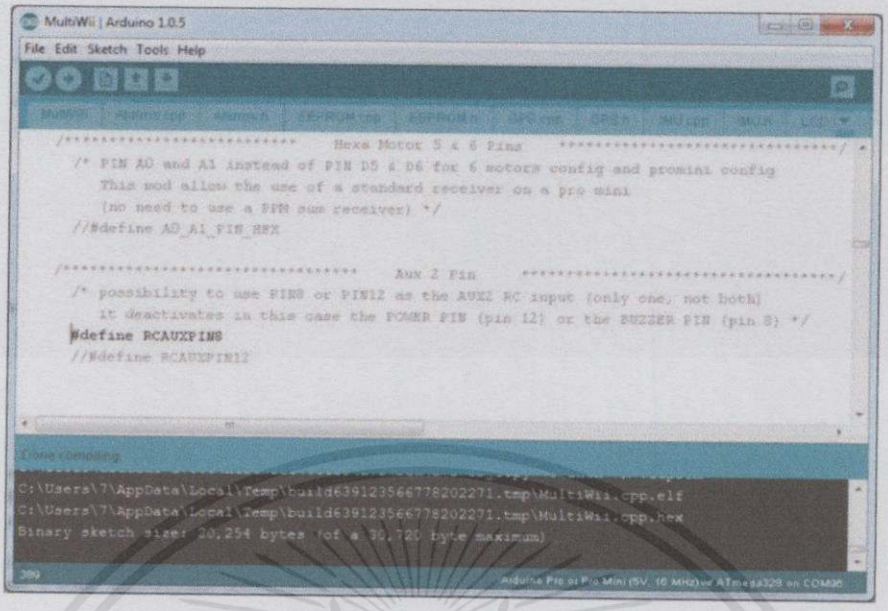


รูปที่ 3.15 การกำหนดค่า I2C_SPEED เท่ากับ 400000L ซึ่งหมายถึง 400kHz (ไอซีเซนเซอร์บนบอร์ด MultiWii เชื่อมต่อแบบบัส I2C)



รูปที่ 3.16 การเลือกใช้บอร์ดควบคุม โดยเลือก CRIUS_SE_V2_0

ซึ่งตรงกับบอร์ดที่ใช้ไอซีเซนเซอร์ MPU6050, HMC5883L และ BMP085

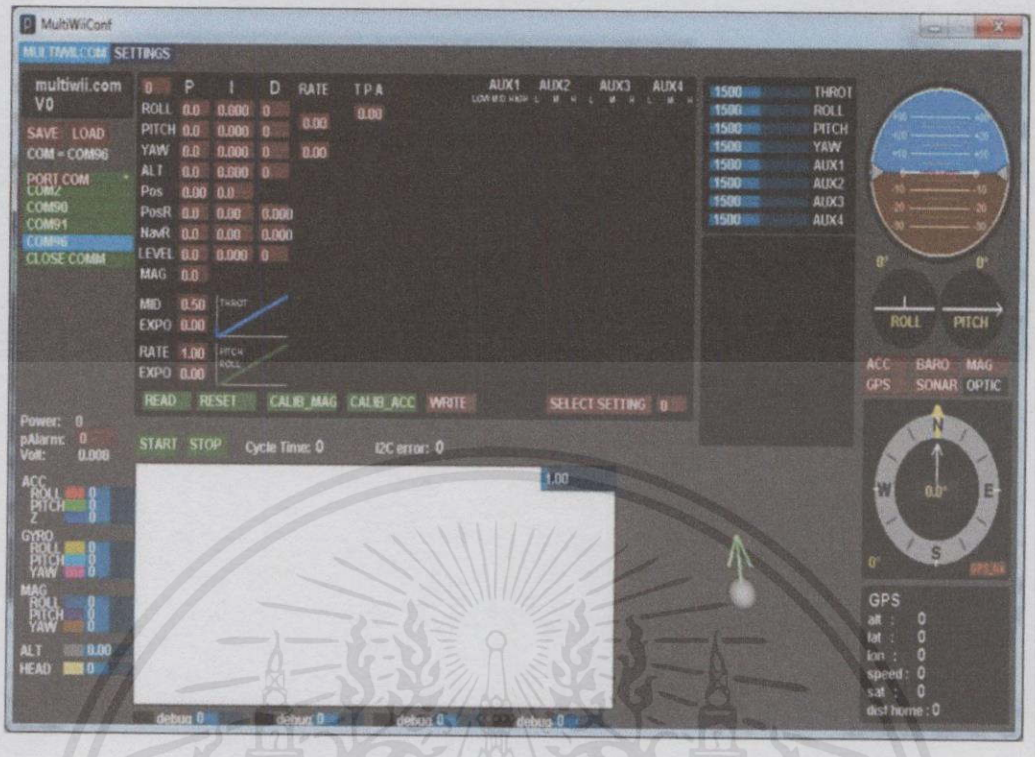


รูปที่ 3.17 การเลือกใช้ขา D8 สำหรับช่องสัญญาณอินพุต AUX2 จาก Receiver

เมื่อได้กำหนดค่าต่างๆ ใหม่แล้ว ให้คอมไพล์ Arduino Sketch (เลือกบอร์ดเป็น Arduino Pro Mini 328P/5V) และอัพโหลดไปยังบอร์ด MultiWii โดยใช้โมดูล FTDI USB-to-Serial หรือโมดูลอื่นที่ใช้งานแทนกันได้

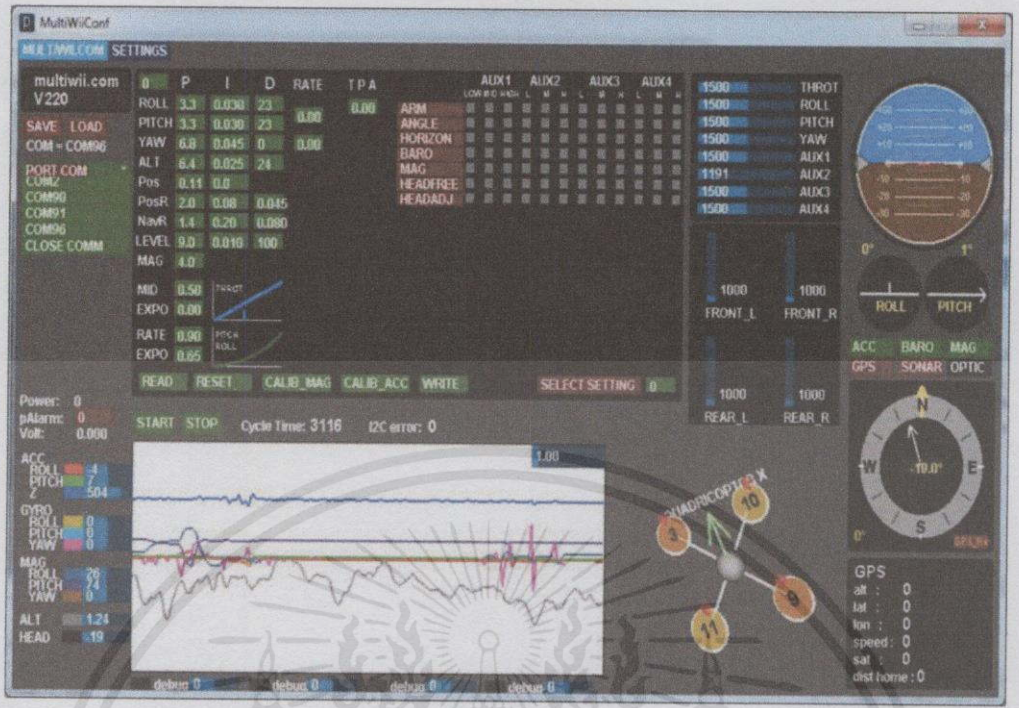
3.4.6 การใช้โปรแกรม MultiWiiConf สำหรับปรับค่าและดูสถานการณ์การทำงานของบอร์ด MultiWii

โปรแกรม MultiWiiConf (ใช้ภาษา Processing ในการเขียนโค้ด) ทำหน้าที่เป็นส่วนเชื่อมต่อผู้ใช้แบบกราฟิก ทำให้สามารถดูสถานการณ์ทำงานของบอร์ด MultiWii รวมถึงการปรับค่าต่างๆ และบันทึกค่าลง EEPROM ในไมโครคอนโทรลเลอร์บนบอร์ด MultiWii ได้



รูปที่ 3.18 หน้าต่างของโปรแกรม MultiWiiConf

เมื่อเปิดโปรแกรมดังกล่าวใช้งานแล้ว ให้เลือกพอร์ตอนุกรมที่ตรงกับ USB-to-Serial ที่เชื่อมต่ออยู่กับบอร์ด MultiWii แล้วกดปุ่ม START จากนั้นโปรแกรมจะทำการเชื่อมต่อกับบอร์ด MultiWii และให้สังเกตการเปลี่ยนแปลงของค่าจากเซนเซอร์ที่ได้จากบอร์ด MultiWii เช่น ACC, GYRO, MAG, และ ALT เป็นต้น ถ้าบอร์ดวางอยู่ได้ระนาบ ค่า ROLL และ PITCH ของ ACC ควรจะได้ใกล้เคียง 0 ในกรณีที่มีการปรับเปลี่ยนค่าของ P, I, D สำหรับการควบคุมในแกน Roll, Pitch, Yaw เป็นต้น ให้ทำการบันทึกค่าโดยกดปุ่ม WRITE และถ้าต้องการจบการเชื่อมต่อกับบอร์ด MultiWii ให้กดปุ่ม STOP



รูปที่ 3.19 หน้าต่างของโปรแกรม MultiWiiConf เมื่อเชื่อมต่อกับบอร์ด MultiWii

การทำงานของ MultiWii มีหลายโหมดสำหรับการบิน (สามารถเปิดใช้งานได้พร้อมกัน) เช่น HORIZON สำหรับรักษาความเสถียรในการบิน โดยใช้ Accelerometer/Gyro โหมด MAG สำหรับเปิดใช้ Magnetometer (เพื่อรักษาทิศทางของตัวเครื่อง ถ้าอินพุตสำหรับ YAW อยู่ตรงกลาง) โหมด BARO สำหรับเปิดใช้งาน Barometric Pressure Sensor (สำหรับรักษาระดับความสูง หรือ Altitude Hold) เป็นต้น ช่องสัญญาณ AUX1 และ/หรือ AUX2 จะใช้สำหรับกำหนดเลือกเปิด/ปิดโหมดการบินผ่านรีโมทบังคับได้

ในการทดลองการทำงานของบอร์ด MultiWii ร่วมกับตัวรับสัญญาณและ Multirotor ในระยะแรกนั้น ให้ทดลองเปิดระบบใช้งาน และทดสอบการทำงานของอุปกรณ์รีโมทหรือตัวส่งสัญญาณ และตรวจสอบว่าสามารถสั่งให้ระบบเริ่มทำงาน (ARM) มอเตอร์แต่ละตัวหมุนอย่างไรบ้าง เมื่อมีการเปลี่ยนตำแหน่งของคันโยกทั้ง 4 แกน (Pitch, Roll, Throttle, Yaw) แต่ยังไม่จำเป็นต้องใส่ใบพัดให้มอเตอร์ ทั้งนี้เพื่อความปลอดภัย

บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้จะกล่าวถึงการทดลองและผลการทดลองการสั่งพิมพ์ชิ้นงาน ที่ออกแบบมาประกอบชิ้นส่วนเข้าเป็นโครงรถ RC การประกอบอุปกรณ์การสั่งการทั้งการวิ่งและการบิน และผลการทดลองการบังคับทิศทางและการเคลื่อนที่ของรถ RC กับการบินของ Quadcopter โดยมีผลการทดลองดังต่อไปนี้

4.1 การพิมพ์ชิ้นงานจากเครื่องพิมพ์สามมิติ

สามารถพิมพ์ชิ้นส่วนต่างๆ ได้จากเครื่องพิมพ์สามมิติ แต่ในการสั่งพิมพ์ชิ้นงานแต่ละชิ้นจะใช้เวลาค่อนข้างนาน เนื่องจากต้องตั้งค่าคุณภาพของชิ้นงานที่ดี เพื่อให้ชิ้นงานแข็งแรงกันกระแทกได้ดี แต่ในการทดลองการบินชิ้นส่วนได้เกิดการแตกหัก เนื่องจาก Quadcopter เกิดร่วงตกลงพื้น เนื่องจากการบังคับที่ยังไม่ค่อยดี และชิ้นส่วนที่พิมพ์ออกมานั้นเป็นพลาสติกทำให้เกิดการแตกหักได้ง่าย

4.2 การประกอบชิ้นงาน

ในการประกอบชิ้นส่วนต่างๆ ของ Quadcopter Car สามารถประกอบได้ตามที่ออกแบบไว้ แต่มีบางส่วนจะเกิดปัญหาการไม่พอดีกันของชิ้นส่วนหรือรูสวมน็อตต่างๆ จึงต้องมีการแก้ปัญหาด้วยการขยายรูหรือการตัดแต่งชิ้นส่วน และประกอบอุปกรณ์ต่างๆ เข้ากับตัวโครงรถ

รูปที่ 4.1 รถเครื่องบินปีกหมุนสี่ใบพัดที่ประกอบชิ้นส่วนต่างๆ วงจรและอุปกรณ์ควบคุมสมบูรณ์

4.3 การทดลองบังคับทิศทางการเคลื่อนที่ของรถ RC

การทดลองการเคลื่อนที่ของล้อรถสามารถขับเคลื่อนได้อย่างดี สามารถเคลื่อนที่ไปข้างหน้า ถอยหลัง เลี้ยวซ้าย เลี้ยวขวา ได้อย่างรวดเร็ว โดยได้ทำการวัดอัตราการเคลื่อนที่ของล้อกับอัตราการเร่งความเร็วจากรีโมทควบคุม และอัตราการเคลื่อนที่ของรถ RC ดังนี้

ตารางที่ 4.1 ผลการวัดอัตราการหมุนของมอเตอร์รถเทียบกับอัตราการใช้งานคันโยกของรีโมทควบคุม

อัตราการหมุน (RPM) / อัตราการใช้งานคันโยก (%)	ทฤษฎี	ผลการทดลองครั้งที่ 1	ผลการทดลองครั้งที่ 2	ผลการทดลองครั้งที่ 3	ค่าเฉลี่ยผลการทดลอง
25	2109	1965	1895	2045	1968.3
50	4218	4110	4002	3725	3945.6
75	6327	5978	6235	5776	5996.3
100	8436	8255	7968	7887	8036.6

ตารางที่ 4.2 ผลการวัดอัตราการเคลื่อนที่ของรถเทียบกับอัตราการใช้งานคันโยกของรีโมทควบคุม

อัตราความเร็ว (เมตร/วินาที) / อัตราการใช้งานคันโยก (%)	ผลการทดลองครั้งที่ 1	ผลการทดลองครั้งที่ 2	ผลการทดลองครั้งที่ 3	ค่าเฉลี่ยผลการทดลอง
25	1.12	1.11	1.09	1.11
50	1.19	1.20	1.14	1.18
75	1.43	1.40	1.42	1.42
100	1.50	1.60	1.54	1.55

4.4 การทดลองบังคับทิศทางการเคลื่อนที่ของ Quadcopter

การทดลองการเคลื่อนที่ของ Quadcopter ในช่วงแรกที่ประกอบรถเสร็จแล้วนั้น ในส่วนของเครื่องบินไม่สามารถลอยขึ้นได้สูง เนื่องจากน้ำหนักเครื่องที่มากเกินไป จึงพยายามตัดส่วนที่ไม่สำคัญออก และเปลี่ยนใบพัดเป็น 3 แฉก จากของเดิมที่ใช้ใบพัดขนาด 7 x 5 นิ้ว 2 แฉก จึงทำให้สามารถเคลื่อนที่ได้ตามระยะที่ต้องการ นอกจากนี้การบังคับเครื่องบินนั้นจะต้องมีการทรงตัวของตัวเครื่องบิน เพื่อให้ลอยได้นิ่งแล้วสามารถบังคับทิศทางการเลี้ยวได้ตามต้องการ ซึ่งผู้บังคับการบินจำเป็นต้องฝึกหัดการใช้รีโมทควบคุมให้ดี โดยได้ทำการวัดความเร็วของใบพัด จากการเร่งความเร็วจากรีโมทควบคุม ดังนี้

ตารางที่ 4.3 ผลการวัดอัตราการหมุนของมอเตอร์เครื่องบินตัวที่ 1 เทียบกับอัตราการใช้งานคันโยกของรีโมทควบคุม

อัตราการหมุน (RPM) / อัตราการใช้งานคันโยก (%)	ทฤษฎี	ผลการทดลองครั้งที่ 1	ผลการทดลองครั้งที่ 2	ผลการทดลองครั้งที่ 3	ค่าเฉลี่ยผลการทดลอง
25	3463.2	3250	2980	3114	3114.6
50	6926.4	6495	6602	6415	6504.0
75	10389.6	10850	9880	10057	10262.3
100	13852.8	12950	13705	13200	13285.0

ตารางที่ 4.4 ผลการวัดอัตราการหมุนของมอเตอร์เครื่องบินตัวที่ 2 เทียบกับอัตราการใช้งานคันโยกของรีโมทควบคุม

อัตราการหมุน (RPM) / อัตราการใช้งานคันโยก (%)	ทฤษฎี	ผลการทดลองครั้งที่ 1	ผลการทดลองครั้งที่ 2	ผลการทดลองครั้งที่ 3	ค่าเฉลี่ยผลการทดลอง
25	3463.2	2996	2769	3002	2922.3
50	6926.4	6578	6335	6650	6521.0
75	10389.6	10004	9785	9856	9881.6
100	13852.8	11985	12450	12756	12397.0

ตารางที่ 4.5 ผลการวัดอัตราการหมุนของมอเตอร์เครื่องบินตัวที่ 3 เทียบกับอัตราการใช้งานคันโยกของรีโมทควบคุม

อัตราการหมุน (RPM) / อัตราการใช้งานคันโยก (%)	ทฤษฎี	ผลการทดลองครั้งที่ 1	ผลการทดลองครั้งที่ 2	ผลการทดลองครั้งที่ 3	ค่าเฉลี่ยผลการทดลอง
25	3463.2	2873	3135	3212	3073.3
50	6926.4	6690	6824	6598	6704.0
75	10389.6	9870	9457	10024	9783.6
100	13852.8	12590	11856	12872	12439.3

ตารางที่ 4.6 ผลการวัดอัตราการหมุนของมอเตอร์เครื่องบินตัวที่ 4 เทียบกับอัตราการใช้งานคันโยกของรีโมทควบคุม

อัตราการหมุน (RPM) / อัตราการใช้งานคันโยก (%)	ทฤษฎี	ผลการทดลองครั้งที่ 1	ผลการทดลองครั้งที่ 2	ผลการทดลองครั้งที่ 3	ค่าเฉลี่ยผลการทดลอง
25	3463.2	3110	2750	3203	3021.0
50	6926.4	6473	6580	6802	6618.3
75	10389.6	9680	9390	9822	9630.6
100	13852.8	11890	12279	12754	12307.6

จากผลการทดลองที่ได้จะเห็นว่ามอเตอร์แต่ละตัว จะมีค่าคลาดเคลื่อนที่แตกต่างกันค่อนข้างมาก แม้ค่าเฉลี่ยจะใกล้เคียงกับค่าทฤษฎี แต่ในการทดลองจริงมักพบปัญหามีมอเตอร์ตัวหนึ่งมีการหมุนช้ากว่าตัวอื่นๆ ทำให้การบังคับการทรงตัวของเครื่องบินค่อนข้างยาก ซึ่งต้องมีการถ่วงน้ำหนักหรือแก้ไขด้วยการวางอุปกรณ์ต่างๆ บนตัวเครื่องบินให้เครื่องบินทรงตัวได้ดีที่สุด

บทที่ 5

บทวิจารณ์และสรุป

5.1 สรุปผลการทดลอง

จากผลการทดลองที่ได้จะพบว่าในส่วนของการเคลื่อนที่แบบรถ RC นั้น จะสามารถทำได้ง่ายกว่าในส่วนของการเคลื่อนที่แบบบิน เนื่องจากรถ RC จะสามารถเคลื่อนที่ได้ดีด้วยปัจจัยเพียงแค่การออกแบบโครงสร้างรถที่ดีทั้งในส่วนของล้อหรือโครงรถ ซึ่งผลการทดลองก็เป็นที่น่าพอใจเนื่องจากสามารถแก้ไขแบบที่ออกมาได้เอง และสั่งพิมพ์ใหม่เพื่อให้พอดีและสามารถเคลื่อนที่ได้รวดเร็ว แต่ในส่วนของเครื่องบินนั้นจะมีหลายปัจจัยที่ส่งผลต่อการบินทั้งน้ำหนักของโครง การปรับตั้งค่า PID ในบอร์ด MultiWii กับเสถียรภาพของ Gyroscope และการควบคุมของผู้บังคับเครื่องบินที่ไม่ชำนาญก็อาจมีผลกระทบได้ ซึ่งผลการทดลองก็ยังถือว่าไม่เป็นไปตามที่คาดหวัง เนื่องจากใบพัดมีแรงยกไม่เพียงพอต่อการบินในที่สูง ถ้าจะปรับแก้ต้องแก้ไขขนาดของใบพัดและขนาดความแรงของมอเตอร์ ซึ่งต้องออกแบบล้อกับโครงรถใหม่หรือซื้อมอเตอร์ใหม่

5.2 ปัญหาที่พบและแนวทางการแก้ไข

สำหรับปัญหาในการทำโครงงานนั้น ปัญหาแรกที่พบคือโครงรถที่ออกแบบมานั้นหลายๆ ชิ้นไม่สามารถพิมพ์ได้ เนื่องจากมีขนาดที่ใหญ่เกินกว่าฐานรองพิมพ์ของเครื่องพิมพ์สามมิติ เช่น ล้อรถ โครงรถ จึงต้องแก้ปัญหาด้วยการแบ่งชิ้นงานเป็นส่วนๆ ในการสั่งพิมพ์แล้วนำมาติดกาวเข้าด้วยกัน

ในการประกอบโครงรถและวงจรควบคุมนั้น ปัญหาที่พบคือการหมุนของล้อจากเฟืองขับเคลื่อนยังเคลื่อนที่ได้ไม่ค่อยดี จึงต้องมีการปรับแก้ตำแหน่งของเฟืองขับเคลื่อนและใส่จารบีช่วยให้ขับเคลื่อนได้ดีขึ้น นอกจากนี้การติดเซอร์โวบริเวณล้อหน้าจะติดค่อนข้างยาก เนื่องจากได้เปลี่ยนมาใช้เซอร์โวขนาดเล็ก เพื่อจะได้ไม่ต้องใช้ตัวแปลงไฟ ซึ่งต้องแก้ไขดัดแปลงโครงรถให้สามารถวางเซอร์โวได้

ปัญหาของการควบคุมการเคลื่อนที่ของเครื่องบินคือ น้ำหนักของตัวเครื่องบินที่ค่อนข้างหนัก ซึ่งทำให้เครื่องบินบินขึ้นได้ไม่สูง ซึ่งต้องทำการแก้ไขด้วยการตัดแต่งตัวโครงรถให้มีน้ำหนักเบา และเปลี่ยนใบพัดเพื่อให้มีแรงยกมากขึ้น นอกจากนี้ในการทดลองบินตัวเครื่องบินอาจไม่สมดุลเพราะตำแหน่งในการวางอุปกรณ์ต่างๆ เช่น มอเตอร์รถ เซอร์โวไม่สามารถติดอยู่กึ่งกลางของตัวเครื่องบินได้ จึงเป็นต้องมีการถ่วงน้ำหนักต่างๆ เช่น ติดน็อตบนแกนของรถ

ในส่วนของการบังคับการบินจะค่อนข้างบังคับได้ยากกว่าเครื่องบินทั่วไป เนื่องจากรถปีกหมุนสี่ใบพัดนั้นส่วนบริเวณใบพัดจะมีล้อที่บดบังทิศทางลมของใบพัด ซึ่งส่งผลให้การบังคับการบินอาจมีการคลาดเคลื่อนได้ง่าย และประกอบกับการใช้ใบพัดสามแฉกที่จะบังคับการบินได้ยากกว่าใบพัดสองแฉก ทำให้การบังคับทิศทางในการบินยังไม่เป็นไปตามที่คาดหวัง

5.3 ประโยชน์ที่ได้รับจากโครงงาน

คาดหวังว่าจะได้รับความรู้ในเรื่องของการควบคุมสั่งการทำงาน โดยการเขียนโปรแกรมลงในตัวไมโครคอนโทรลเลอร์ นอกจากนี้หวังว่าจะสามารถแก้ปัญหาต่างๆ ที่เกิดขึ้นในระหว่างการทดลองได้อย่างสมบูรณ์ และประสบการณ์จากการทำงานที่ได้รับมอบหมายให้สำเร็จลุล่วง ซึ่งความรู้และประสบการณ์จากการทำโครงงานครั้งนี้จะเกิดผลต่อผู้ศึกษาจากโครงงานเล่มนี้ไม่มากก็น้อย

5.4 ข้อเสนอแนะและแนวทางในการค้นคว้าพัฒนา

จากปัญหาที่พบในระหว่างการทดลองนั้น สำหรับการเคลื่อนที่ของรถ ควรคำนึงถึงเฟืองขับเคลื่อนว่าสามารถขบกันได้อย่างดีหรือไม่ และขนาดของโครงรถเป็นไปตามทฤษฎีการบินหรือไม่ ก่อนที่จะออกแบบควรคำนึงถึงปัจจัยต่างๆ ควรศึกษาทฤษฎีของการวิ่งและการบินให้เข้าใจ เพื่อจะได้ไม่เสียเวลาต้องกลับมาออกแบบหรือพิมพ์โครงรถใหม่ และที่สำคัญควรคำนึงถึงขนาดของรถ ล้อ ใบพัด และมอเตอร์ว่ามีขนาดที่เหมาะสมหรือไม่ ซึ่งปัจจัยเหล่านี้จะส่งผลต่อการทดลองโดยตรง ซึ่งถ้าไม่สามารถใช้อุปกรณ์ที่ซื้อมาได้ ก็จะต้องซื้อใหม่ทำให้เปลืองงบประมาณเพราะอุปกรณ์แต่ละชิ้นมีราคาค่อนข้างสูง

เอกสารอ้างอิง

[1] **แบ่งปันความรู้ด้านเทคโนโลยี** Microcontroller, http://pic16f627a.blogspot.com
สืบค้นเมื่อ 1 ตุลาคม 2557

[2] **ฟิสิกส์ของเฮลิคอปเตอร์บังคับ**, http://www.vcharkarn.com/varticle/44147
สืบค้นเมื่อ 1 ตุลาคม 2557

[3] **ความรู้พื้นฐานเกี่ยวกับเครื่องบินบังคับ**, http://tamoochi.blogspot.com
สืบค้นเมื่อ 1 ตุลาคม 2557

[4] **ความรู้พื้นฐานการใช้งานเซอร์โว**, http://www.tdhobby.com
สืบค้นเมื่อ 12 พฤศจิกายน 2557

[5] **B-EX**, http://www.bgobeyond.co.uk
สืบค้นเมื่อ 12 พฤศจิกายน 2557

[6] **Radio control car**, http://radiocontrolcarr.blogspot.com/2007/10/1.html
สืบค้นเมื่อ 15 พฤศจิกายน 2557

[7] **RC-Thai**, http://www.rcthai.net/board/
สืบค้นเมื่อ 8 ธันวาคม 2557

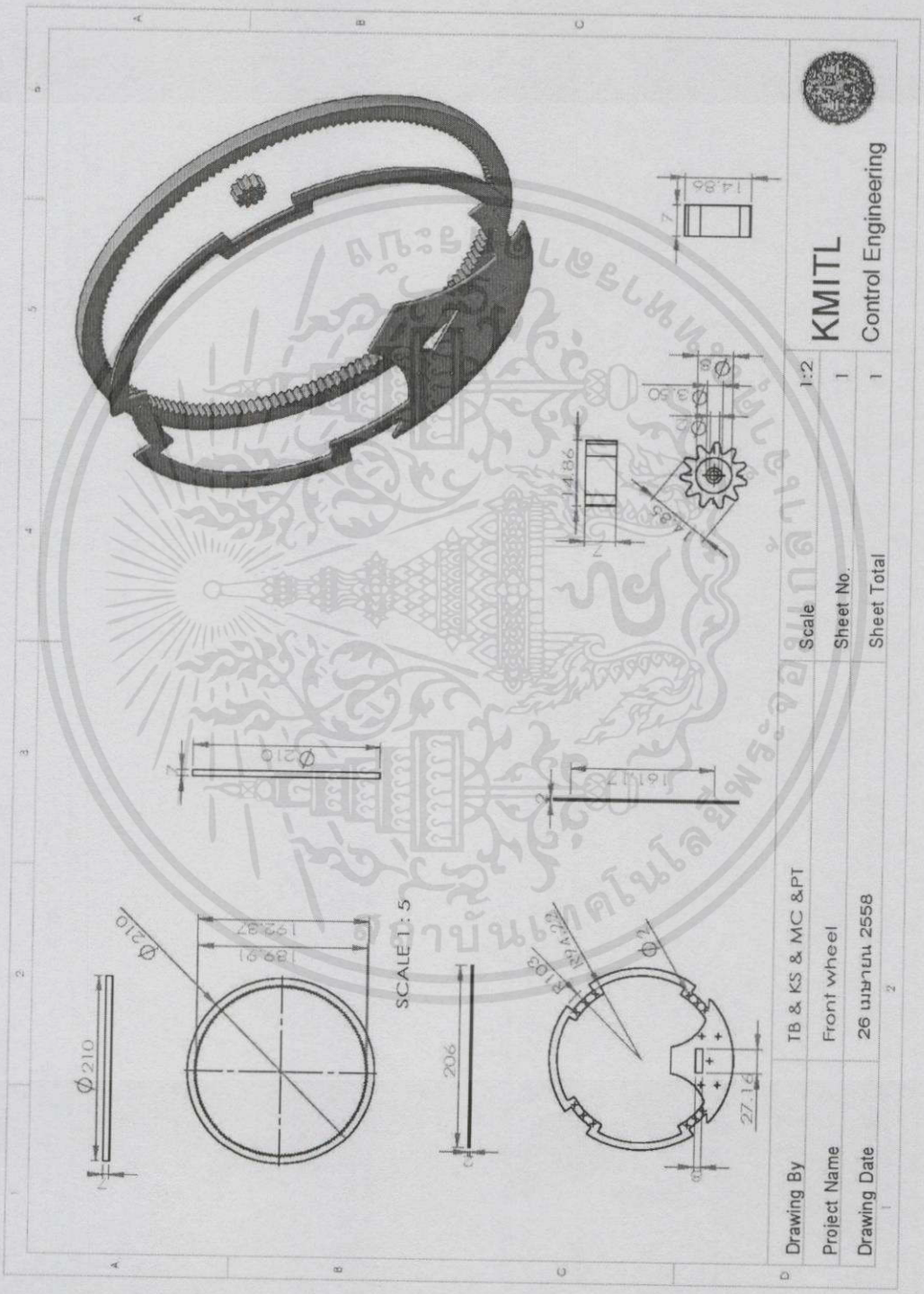
[8] **How to choose motor and propeller** ,http://blog.oscarliang.net
สืบค้นเมื่อ 10 ธันวาคม 2557

[9] **การทดลองใช้บอร์ด MultiWii SE**, http://cpre.kmutnb.ac.th
สืบค้นเมื่อ 5 มกราคม 2558

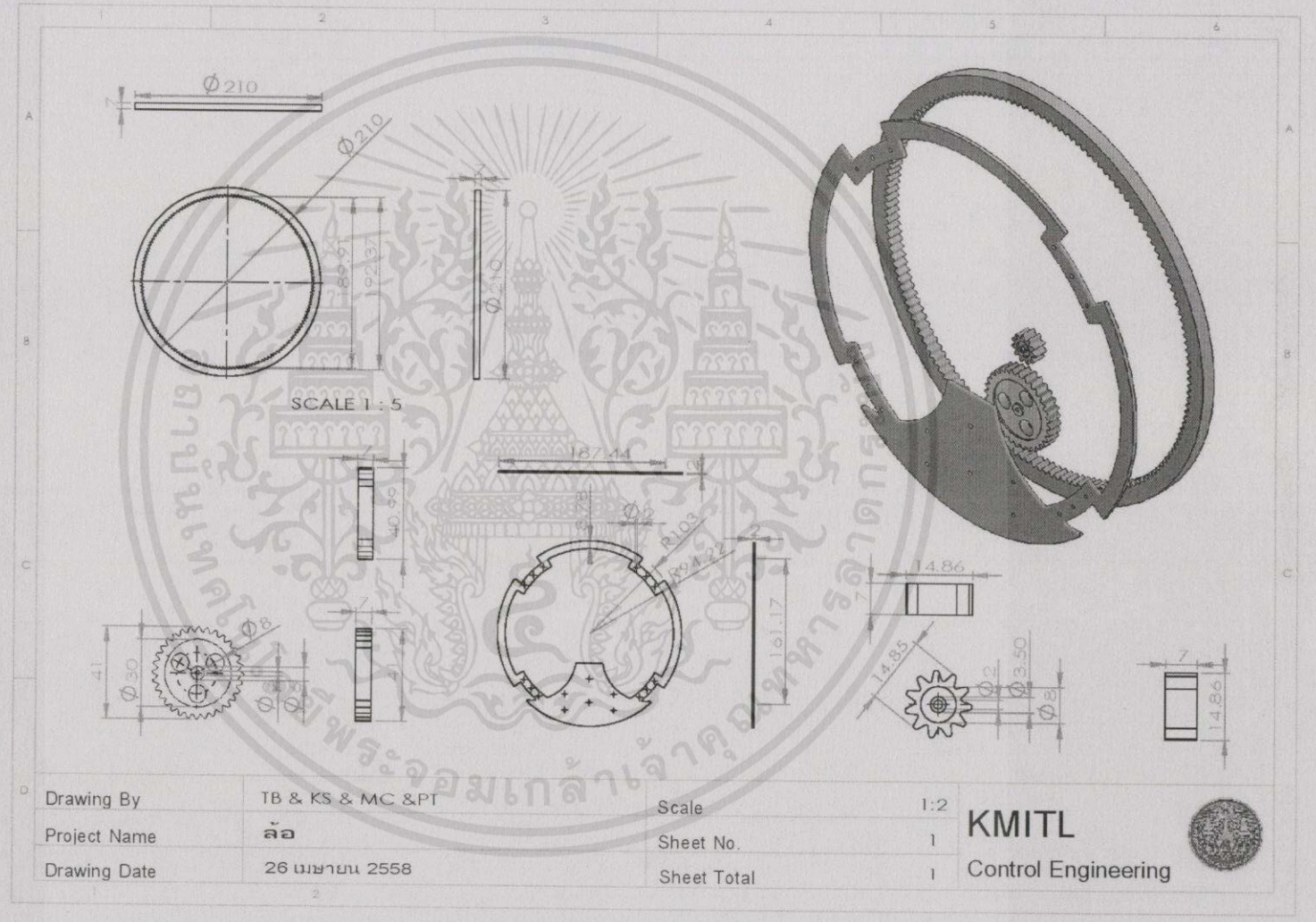
[10] **DIY Quadcopter**, http://www.instructables.com/id/DIY-Quadcopter/
สืบค้นเมื่อ 12 กุมภาพันธ์ 2558

ภาคผนวก

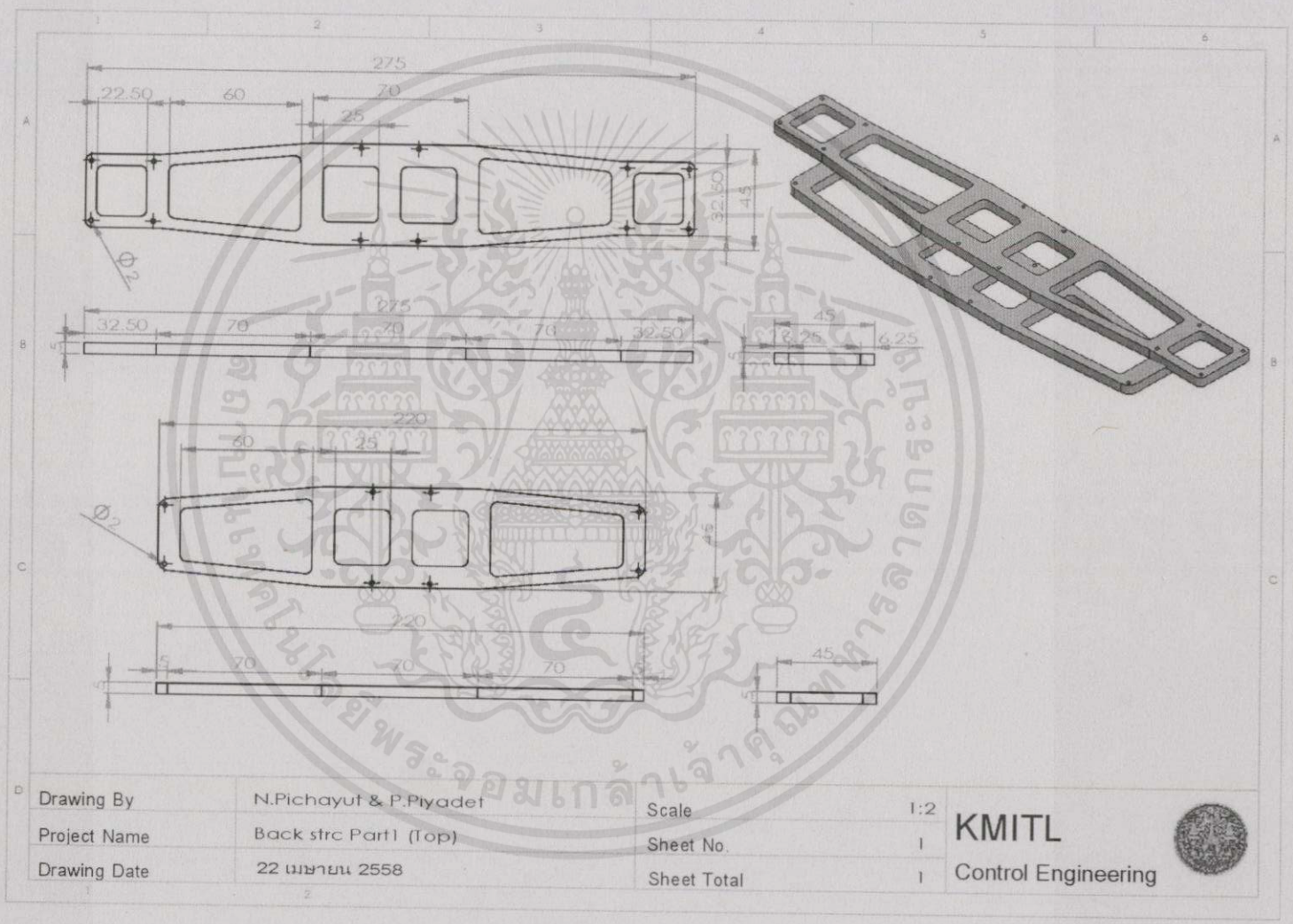
ภาคผนวก ก
แบบร่างส่วนประกอบรถเครื่องบินปีกหมุนสี่ใบพัด



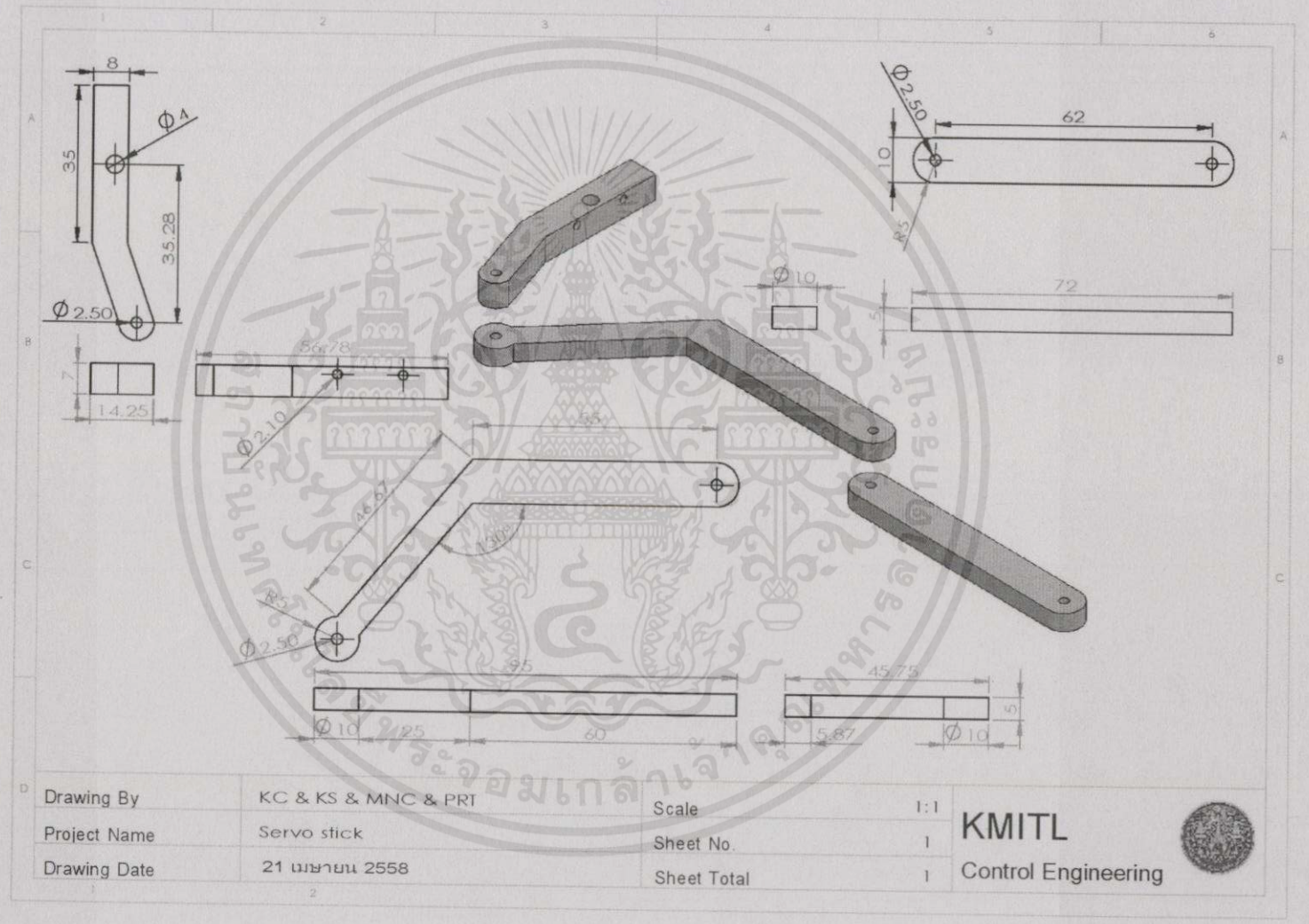
รูปที่ ก.1 แบบร่างล้อรถด้านหน้า



รูปที่ ก.2 แบบร่างล้อรถด้านหลัง



รูปที่ ก.3 แบบร่างฐานรถ



รูปที่ ก.4 แบบร่างแกนเซอร์โว

ภาคผนวก ข

Code คำสั่งควบคุม

ข.1 Code คำสั่ง EEPROM.ccp

```
#include <avr/eeprom.h>
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "EEPROM.h"
#include "MultiWii.h"
#include "Alarms.h"
#include "GPS.h"
void LoadDefaults(void);
uint8_t calculate_sum(uint8_t *cb , uint8_t siz) {
  uint8_t sum=0x55;  // checksum init
  while(--siz) sum += *cb++;  // calculate checksum (without checksum byte)
  return sum;
}
void readGlobalSet() {
  eeprom_read_block((void*)&global_conf, (void*)0, sizeof(global_conf));
  if(calculate_sum((uint8_t*)&global_conf,                sizeof(global_conf))                !=
global_conf.checksum) {
    global_conf.currentSet = 0;
    global_conf.accZero[ROLL] = 5000;    // for config error signalization
  }
}
bool readEEPROM() {
  uint8_t i;
  #ifdef MULTIPLE_CONFIGURATION_PROFILES
  if(global_conf.currentSet>2) global_conf.currentSet=0;
  #else
```

```
    global_conf.currentSet=0;
  #endif
  eeprom_read_block((void*)&conf, (void*)(global_conf.currentSet * sizeof(conf) +
sizeof(global_conf)), sizeof(conf));
  if(calculate_sum((uint8_t*)&conf, sizeof(conf)) != conf.checksum) {
    blinkLED(6,100,3);
    #if defined(BUZZER)
      alarmArray[7] = 3;
    #endif
    LoadDefaults();                    // force load defaults
    return false;                      // defaults loaded, don't reload constants (EEPROM life
saving)
  }
  // 500/128 = 3.90625    3.9062 * 3.9062 = 15.259   1526*100/128 = 1192
  for(i=0;i<5;i++) {
    lookupPitchRollRC[i] = (1526+conf.rcExpo8*(i*i-15))*i*(int32_t)conf.rcRate8/1192;
  }
  for(i=0;i<11;i++) {
    int16_t tmp = 10*i-conf.thrMid8;
    uint8_t y = 1;
    if (tmp>0) y = 100-conf.thrMid8;
    if (tmp<0) y = conf.thrMid8;
    lookupThrottleRC[i] = 10*conf.thrMid8 + tmp*( 100-
conf.thrExpo8+(int32_t)conf.thrExpo8*(tmp*tmp)/(y*y) )/10; // [0;1000]
    lookupThrottleRC[i] = conf.minthrottle + (int32_t)(MAXTHROTTLE-
conf.minthrottle)* lookupThrottleRC[i]/1000; // [0;1000] ->
[conf.minthrottle;MAXTHROTTLE]
  }
#if defined(POWERMETER)
    pAlarm = (uint32_t) conf.powerTrigger1 * (uint32_t) PLEVELSCALE * (uint32_t)
PLEVELDIV; // need to cast before multiplying
  #endif
  #if GPS
```

```
    GPS_set_pids();    // at this time we don't have info about GPS init done
   #endif
   #if defined(ARMEDTIMEWARNING)
    ArmedTimeWarningMicroSeconds = (conf.armedtimewarning *1000000);
   #endif
   return true;    // setting is OK
 }
 void writeGlobalSet(uint8_t b) {
  global_conf.checksum = calculate_sum((uint8_t*)&global_conf, sizeof(global_conf));
  eeprom_write_block((const void*)&global_conf, (void*)0, sizeof(global_conf));
  if (b == 1) blinkLED(15,20,1);
  #if defined(BUZZER)
   alarmArray[7] = 1;
  #endif
}
 void writeParams(uint8_t b) {
  #ifdef MULTIPLE_CONFIGURATION_PROFILES
   if(global_conf.currentSet>2) global_conf.currentSet=0;
  #else
   global_conf.currentSet=0;
  #endif
  conf.checksum = calculate_sum((uint8_t*)&conf, sizeof(conf));
  eeprom_write_block((const void*)&conf, (void*)(global_conf.currentSet * sizeof(conf)
+ sizeof(global_conf)), sizeof(conf));
  readEEPROM();
  if (b == 1) blinkLED(15,20,1);
  #if defined(BUZZER)
   alarmArray[7] = 1; //beep if loaded from gui or android
  #endif
}
 void update_constants() {
  #if defined(GYRO_SMOOTHING)
   {
```

```
    uint8_t s[3] = GYRO_SMOOTHING;
    for(uint8_t i=0;i<3;i++) conf.Smoothing[i] = s[i];
  }
#endif
#if defined (FAILSAFE)
  conf.failsafe_throttle = FAILSAFE_THROTTLE;
#endif
#ifdef VBAT
  conf.vbatscale = VBATSCALE;
  conf.vbatlevel_warn1 = VBATLEVEL_WARN1;
  conf.vbatlevel_warn2 = VBATLEVEL_WARN2;
  conf.vbatlevel_crit = VBATLEVEL_CRIT;
#endif
#ifdef POWERMETER
  conf.pint2ma = PINT2mA;
#endif
#ifdef POWERMETER_HARD
  conf.psensornull = PSENSORNULL;
#endif
#ifdef MMGYRO
  conf.mmgyro = MMGYRO;
#endif
#if defined(ARMEDTIMEWARNING)
  conf.armedtimewarning = ARMEDTIMEWARNING;
#endif
conf.minthrottle = MINTHROTTLE;
#if MAG
  conf.mag_declination = (int16_t)(MAG_DECLINATION * 10);
#endif
#ifdef GOVERNOR_P
  conf.governorP = GOVERNOR_P;
  conf.governorD = GOVERNOR_D;
#endif
```

```
  #if defined(MY_PRIVATE_DEFAULTS)
   #include MY_PRIVATE_DEFAULTS
  #endif
  writeParams(0); // this will also (p)reset checkNewConf with the current version
number again.
}
void LoadDefaults() {
  uint8_t i;
  #ifdef SUPPRESS_DEFAULTS_FROM_GUI
   // do nothing
  #elif defined(MY_PRIVATE_DEFAULTS)
   // #include MY_PRIVATE_DEFAULTS
   // do that at the last possible moment, so we can override virtually all defaults
and constants
  #else
        #if PID_CONTROLLER == 1
    conf.pid[ROLL].P8   = 33;  conf.pid[ROLL].I8   = 30; conf.pid[ROLL].D8   = 23;
    conf.pid[PITCH].P8  = 33; conf.pid[PITCH].I8  = 30; conf.pid[PITCH].D8  = 23;
    conf.pid[PIDLEVEL].P8 = 90;  conf.pid[PIDLEVEL].I8 = 10;  conf.pid[PIDLEVEL].D8 =
100;
   #elif PID_CONTROLLER == 2
    conf.pid[ROLL].P8   = 28;  conf.pid[ROLL].I8   = 10; conf.pid[ROLL].D8   = 7;
    conf.pid[PITCH].P8  = 28; conf.pid[PITCH].I8  = 10; conf.pid[PITCH].D8  = 7;
    conf.pid[PIDLEVEL].P8 = 30; conf.pid[PIDLEVEL].I8 = 32; conf.pid[PIDLEVEL].D8 = 0;
   #endif
conf.pid[YAW].P8    = 68;  conf.pid[YAW].I8    = 45;  conf.pid[YAW].D8    = 0;
conf.pid[PIDALT].P8  = 64; conf.pid[PIDALT].I8  = 25; conf.pid[PIDALT].D8  = 24;
conf.pid[PIDPOS].P8  = POSHOLD_P * 100;     conf.pid[PIDPOS].I8    = POSHOLD_I *
100;     conf.pid[PIDPOS].D8    = 0;
conf.pid[PIDPOSR].P8  =  POSHOLD_RATE_P  *  10;  conf.pid[PIDPOSR].I8        =
POSHOLD_RATE_I * 100;  conf.pid[PIDPOSR].D8  = POSHOLD_RATE_D * 1000;
conf.pid[PIDNAVR].P8  = NAV_P * 10;          conf.pid[PIDNAVR].I8   = NAV_I * 100;
conf.pid[PIDNAVR].D8   = NAV_D * 1000;
```

```
conf.pid[PIDMAG].P8   = 40;
conf.pid[PIDVEL].P8 = 0;       conf.pid[PIDVEL].I8 = 0;    conf.pid[PIDVEL].D8 = 0;
conf.rcRate8 = 90; conf.rcExpo8 = 65;
    conf.rollPitchRate = 0;
    conf.yawRate = 0;
    conf.dynThrPID = 0;
    conf.thrMid8 = 50; conf.thrExpo8 = 0;
    for(i=0;i<CHECKBOXITEMS;i++) {conf.activate[i] = 0;}
    conf.angleTrim[0] = 0; conf.angleTrim[1] = 0;
    conf.powerTrigger1 = 0;
  #endif // SUPPRESS_DEFAULTS_FROM_GUI
  #if defined(SERVO)
    static int8_t sr[8] = SERVO_RATES;
    #ifdef SERVO_MIN
      static int16_t smin[8] = SERVO_MIN;
      static int16_t smax[8] = SERVO_MAX;
      static int16_t smid[8] = SERVO_MID;
    #endif
    for(i=0;i<8;i++) {
      #ifdef SERVO_MIN
        conf.servoConf[i].min = smin[i];
        conf.servoConf[i].max = smax[i];
        conf.servoConf[i].middle = smid[i];
      #else
        conf.servoConf[i].min = 1020;
        conf.servoConf[i].max = 2000;
        conf.servoConf[i].middle = 1500;
      #endif
      conf.servoConf[i].rate = sr[i];
    }
  #endif
  #ifdef FIXEDWING
    conf.dynThrPID = 50;
```

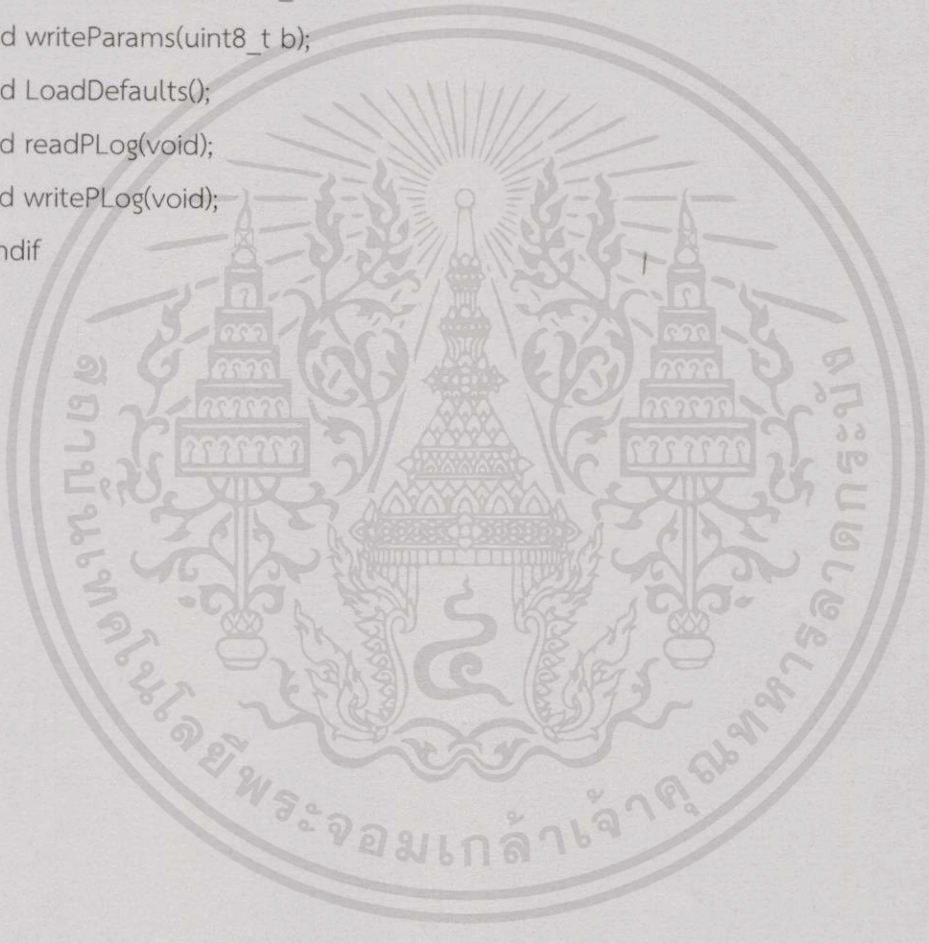
```
    conf.rcExpo8   =  0;
  #endif
  update_constants(); // this will also write to eeprom
 }
 #ifdef LOG_PERMANENT
 void readPLog(void) {
  eeprom_read_block((void*)&plog, (void*)(E2END - 4 - sizeof(plog)), sizeof(plog));
  if(calculate_sum((uint8_t*)&plog, sizeof(plog)) != plog.checksum) {
   blinkLED(9,100,3);
   #if defined(BUZZER)
    alarmArray[7] = 3;
   #endif
      plog.arm = plog.disarm = plog.start = plog.failsafe = plog.i2c = 0;
   plog.running = 1;
   plog.lifetime = plog.armed_time = 0;
   writePLog();
  }
 }
 void writePLog(void) {
  plog.checksum = calculate_sum((uint8_t*)&plog, sizeof(plog));
  eeprom_write_block((const   void*)&plog,   (void*)(E2END   -   4   -   sizeof(plog)),
sizeof(plog));
 }
 #endif
 #if defined(GPS_NAV)
 //Stores the WP data in the wp struct in the EEPROM
 void storeWP() {
 #ifdef MULTIPLE_CONFIGURATION_PROFILES
   #define PROFILES 3
 #else
   #define PROFILES 1
 #endif
       if (mission_step.number >254) return;
```

```
        mission_step.checksum = calculate_sum((uint8_t*)&mission_step,
sizeof(mission_step));
        eeprom_write_block((void*)&mission_step, (void*)(PROFILES * sizeof(conf) +
sizeof(global_conf)+(sizeof(mission_step)*mission_step.number)),sizeof(mission_step));
}
bool recallWP(uint8_t wp_number) {
#ifdef MULTIPLE_CONFIGURATION_PROFILES
  #define PROFILES 3
#else
  #define PROFILES 1
#endif
        if (wp_number > 254) return false;
eeprom_read_block((void*)&mission_step, (void*)(PROFILES * sizeof(conf) +
sizeof(global_conf)+(sizeof(mission_step)*wp_number)), sizeof(mission_step));
        if(calculate_sum((uint8_t*)&mission_step, sizeof(mission_step)) !=
mission_step.checksum) return false;

        return true;
}

uint8_t getMaxWPNumber() {
#ifdef LOG_PERMANENT
  #define PLOG_SIZE sizeof(plog)
#else
  #define PLOG_SIZE 0
#endif
#ifdef MULTIPLE_CONFIGURATION_PROFILES
  #define PROFILES 3
#else
        #define PROFILES 1
#endif
uint16_t first_avail = PROFILES*sizeof(conf) + sizeof(global_conf)+ 1; //Add one byte
for addtnl separation
```

```
        uint16_t last_avail  = E2END - PLOG_SIZE - 4;
        uint16_t wp_num = (last_avail-first_avail)/sizeof(mission_step);
        if (wp_num>254) wp_num = 254;
        return wp_num;
}
#endif
```

ข.2 Code คำสั่ง EEPROM.h

```
#ifndef EEPROM_H_
#define EEPROM_H_
void readGlobalSet();
bool readEEPROM();
void update_constants();
void writeGlobalSet(uint8_t b);
void writeParams(uint8_t b);
void LoadDefaults();
void readPLog(void);
void writePLog(void);
#endif
```



ข.3 Code คำสั่ง IMU.ccp

```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
#include "IMU.h"
#include "Sensors.h"
void getEstimatedAttitude();
void computeIMU () {
  uint8_t axis;
  static int16_t gyroADCprevious[3] = {0,0,0};
  int16_t gyroADCp[3];
  int16_t gyroADCinter[3];
  static uint32_t timeInterleave = 0;
//we separate the 2 situations because reading gyro values with a gyro only setup can be acchieved at a higher rate
//gyro+nunchuk: we must wait for a quite high delay betwwen 2 reads to get both WM+ and Nunchuk data. It works with 3ms
//gyro only: the delay to read 2 consecutive values can be reduced to only 0.65ms
  #if defined(NUNCHUCK)
    annexCode();
    while((uint16_t)(micros()-timeInterleave)<INTERLEAVING_DELAY)   ;   //interleaving delay between 2 consecutive reads
    timeInterleave=micros();
    ACC_getADC();
    getEstimatedAttitude(); // computation time must last less than one interleaving delay
    while((uint16_t)(micros()-timeInterleave)<INTERLEAVING_DELAY)   ;   //interleaving delay between 2 consecutive reads
    timeInterleave=micros();
    f.NUNCHUKDATA = 1;
```

```
    while(f.NUNCHUKDATA) ACC_getADC(); // For this interleaving reading, we must
have a gyro update at this point (less delay)
        for (axis = 0; axis < 3; axis++) {
      // empirical, we take a weighted value of the current and the previous values
      // /4 is to average 4 values, note: overflow is not possible for WMP gyro here
      imu.gyroData[axis] = (imu.gyroADC[axis]*3+gyroADCprevious[axis])>>2;
      gyroADCprevious[axis] = imu.gyroADC[axis];
    }
  #else
    #if ACC
      ACC_getADC();
      getEstimatedAttitude();
    #endif
    #if GYRO
      Gyro_getADC();
    #endif
    for (axis = 0; axis < 3; axis++)
      gyroADCp[axis] =  imu.gyroADC[axis];
    timeInterleave=micros();
    annexCode();
    uint8_t t=0;
    while((uint16_t)(micros()-timeInterleave)<650) t=1; //empirical, interleaving delay
between 2 consecutive reads
    if (!t) annex650_overrun_count++;
    #if GYRO
      Gyro_getADC();
    #endif
    for (axis = 0; axis < 3; axis++) {
      gyroADCinter[axis] =  imu.gyroADC[axis]+gyroADCp[axis];
      // empirical, we take a weighted value of the current and the previous values
      imu.gyroData[axis] = (gyroADCinter[axis]+gyroADCprevious[axis])/3;
      gyroADCprevious[axis] = gyroADCinter[axis]>>1;
      if (!ACC) imu.accADC[axis]=0;
```

```
    }
  #endif
  #if defined(GYRO_SMOOTHING)
    static int16_t gyroSmooth[3] = {0,0,0};
    for (axis = 0; axis < 3; axis++) {
      imu.gyroData[axis] = (int16_t) ( ( (int32_t)((int32_t)gyroSmooth[axis] *
(conf.Smoothing[axis]-1) )+imu.gyroData[axis]+1 ) / conf.Smoothing[axis]);
      gyroSmooth[axis] = imu.gyroData[axis];
    }
  #elif defined(TRI)
    static int16_t gyroYawSmooth = 0;
    imu.gyroData[YAW] = (gyroYawSmooth*2+imu.gyroData[YAW])/3;
    gyroYawSmooth = imu.gyroData[YAW];
  #endif
}
#ifndef ACC_LPF_FACTOR
  #define ACC_LPF_FACTOR 4 // that means a LPF of 16
#endif
/* Set the Gyro Weight for Gyro/Acc complementary filter
   Increasing this value would reduce and delay Acc influence on the output of the
filter*/
#ifndef GYR_CMPF_FACTOR
  #define GYR_CMPF_FACTOR 600
#endif

/* Set the Gyro Weight for Gyro/Magnetometer complementary filter
   Increasing this value would reduce and delay Magnetometer influence on the
output of the filter*/
#define GYR_CMPFM_FACTOR 250
#define INV_GYR_CMPF_FACTOR   (1.0f / (GYR_CMPF_FACTOR  + 1.0f))
#define INV_GYR_CMPFM_FACTOR  (1.0f / (GYR_CMPFM_FACTOR + 1.0f))
typedef struct fp_vector {
```

```
  float X,Y,Z;
} t_fp_vector_def;

typedef union {
  float A[3];
  t_fp_vector_def V;
} t_fp_vector;

typedef struct int32_t_vector {
  int32_t X,Y,Z;
} t_int32_t_vector_def;

typedef union {
  int32_t A[3];
  t_int32_t_vector_def V;
} t_int32_t_vector;

int16_t _atan2(int32_t y, int32_t x){
  float z = (float)y / x;
  int16_t a;
  if ( abs(y) < abs(x) ){
     a = 573 * z / (1.0f + 0.28f * z * z);
   if (x<0) {
     if (y<0) a -= 1800;
     else a += 1800;
   }
  } else {
   a = 900 - 573 * z / (z * z + 0.28f);
   if (y<0) a -= 1800;
  }
  return a;
}
float InvSqrt (float x){
```

```
  union{
    int32_t i;
    float   f;
  } conv;
  conv.f = x;
  conv.i = 0x5f3759df - (conv.i >> 1);
  return 0.5f * conv.f * (3.0f - x * conv.f * conv.f);
}
// Rotate Estimated vector(s) with small angle approximation, according to the gyro
data
void rotateV(struct fp_vector *v,float* delta) {
  fp_vector v_tmp = *v;
  v->Z -= delta[ROLL]  * v_tmp.X + delta[PITCH] * v_tmp.Y;
  v->X += delta[ROLL]  * v_tmp.Z - delta[YAW]   * v_tmp.Y;
  v->Y += delta[PITCH] * v_tmp.Z + delta[YAW]   * v_tmp.X;
}
static int32_t accLPF32[3]    = {0, 0, 1};
static float invG; // 1/|G|

static t_fp_vector EstG;
static t_int32_t_vector EstG32;
#if MAG
 static t_int32_t_vector EstM32;
 static t_fp_vector EstM;
#endif

void getEstimatedAttitude(){
 uint8_t axis;
 int32_t accMag = 0;
 float scale, deltaGyroAngle[3];
 uint8_t validAcc;
 static uint16_t previousT;
 uint16_t currentT = micros();
```

```
  scale = (currentT - previousT) * GYRO_SCALE; // GYRO_SCALE unit:
radian/microsecond
  previousT = currentT;
  for (axis = 0; axis < 3; axis++) {
    deltaGyroAngle[axis] = imu.gyroADC[axis]  * scale; // radian

    accLPF32[axis]    -= accLPF32[axis]>>ACC_LPF_FACTOR;
    accLPF32[axis]    += imu.accADC[axis];
    imu.accSmooth[axis]    = accLPF32[axis]>>ACC_LPF_FACTOR;

    accMag += (int32_t)imu.accSmooth[axis]*imu.accSmooth[axis] ;
  }
  rotateV(&EstG.V,deltaGyroAngle);
  #if MAG
    rotateV(&EstM.V,deltaGyroAngle);
  #endif
 accMag = accMag*100/((int32_t)ACC_1G*ACC_1G);
  validAcc = 72 < (uint16_t)accMag && (uint16_t)accMag < 133;
  // Apply complimentary filter (Gyro drift correction)
  // If accel magnitude >1.15G or <0.85G and ACC vector outside of the limit range
=> we neutralize the effect of accelerometers in the angle estimation.
  // To do that, we just skip filter, as EstV already rotated by Gyro
  for (axis = 0; axis < 3; axis++) {
    if ( validAcc )
      EstG.A[axis] = (EstG.A[axis] * GYR_CMPF_FACTOR + imu.accSmooth[axis]) *
INV_GYR_CMPF_FACTOR;
    EstG32.A[axis] = EstG.A[axis]; //int32_t cross calculation is a little bit faster than
float
    #if MAG
      EstM.A[axis] = (EstM.A[axis] * GYR_CMPFM_FACTOR + imu.magADC[axis]) *
INV_GYR_CMPFM_FACTOR;
      EstM32.A[axis] = EstM.A[axis];
```

```
    #endif
  }

  if ((int16_t)EstG32.A[2] > ACCZ_25deg)
    f.SMALL_ANGLES_25 = 1;
  else
    f.SMALL_ANGLES_25 = 0;

  // Attitude of the estimated vector
  int32_t sqGX_sqGZ = sq(EstG32.V.X) + sq(EstG32.V.Z);
  invG = InvSqrt(sqGX_sqGZ + sq(EstG32.V.Y));
  att.angle[ROLL]  = _atan2(EstG32.V.X , EstG32.V.Z);
  att.angle[PITCH] = _atan2(EstG32.V.Y , InvSqrt(sqGX_sqGZ)*sqGX_sqGZ);

  #if MAG
    att.heading = _atan2(
      EstM32.V.Z * EstG32.V.X - EstM32.V.X * EstG32.V.Z,
      (EstM.V.Y * sqGX_sqGZ  - (EstM32.V.X * EstG32.V.X + EstM32.V.Z * EstG32.V.Z) *
EstG.V.Y)*invG );
    att.heading += conf.mag_declination; // Set from GUI
    att.heading /= 10;
  #endif

  #if defined(THROTTLE_ANGLE_CORRECTION)
    cosZ = EstG.V.Z / ACC_1G * 100.0f;
// cos(angleZ) * 100
    throttleAngleCorrection = THROTTLE_ANGLE_CORRECTION * constrain(100 - cosZ,
0, 100) >>3;  // 16 bit ok: 200*150 = 30000
  #endif
}
```



```
#define UPDATE_INTERVAL 25000    // 40hz update rate (20hz LPF on acc)
#define BARO_TAB_SIZE   21
```

```
#define ACC_Z_DEADBAND (ACC_1G>>5) // was 40 instead of 32 now

#define applyDeadband(value, deadband)
  if(abs(value) < deadband) {
    value = 0;
  } else if(value > 0){
    value -= deadband;
  } else if(value < 0){
    value += deadband;
  }

#if BARO
uint8_t getEstimatedAltitude(){
 int32_t  BaroAlt;
  static float baroGroundTemperatureScale,logBaroGroundPressureSum;
  static float vel = 0.0f;
  static uint16_t previousT;
  uint16_t currentT = micros();
  uint16_t dTime;

  dTime = currentT - previousT;
  if (dTime < UPDATE_INTERVAL) return 0;
  previousT = currentT;

  if(calibratingB > 0) {
    logBaroGroundPressureSum = log(baroPressureSum);
    baroGroundTemperatureScale = (baroTemperature + 27315) *  29.271267f;
    calibratingB--;
  }
  BaroAlt     =    (    logBaroGroundPressureSum    -    log(baroPressureSum)    )    *
  baroGroundTemperatureScale;
```

```
alt.EstAlt = (alt.EstAlt * 6 + BaroAlt * 2) >> 3; // additional LPF to reduce baro noise (faster by 30 µs)

#if (defined(VARIOMETER) && (VARIOMETER != 2)) || !defined(SUPPRESS_BARO_ALTHOLD)
  //P
  int16_t error16 = constrain(AltHold - alt.EstAlt, -300, 300);
  applyDeadband(error16, 10); //remove small P parametr to reduce noise near zero position
  BaroPID = constrain((conf.pid[PIDALT].P8 * error16 >>7), -150, +150);


  errorAltitudeI += conf.pid[PIDALT].I8 * error16 >>6;
  errorAltitudeI = constrain(errorAltitudeI,-30000,30000);
  BaroPID += errorAltitudeI>>9; //I in range +/-60

  // projection of ACC vector to global Z, with 1G subtructed
  // Math: accZ = A * G / |G| - 1G
  int16_t accZ = (imu.accSmooth[ROLL] * EstG32.V.X + imu.accSmooth[PITCH] * EstG32.V.Y + imu.accSmooth[YAW] * EstG32.V.Z) * invG;

  static int16_t accZoffset = 0;
  if (!f.ARMED) {
    accZoffset -= accZoffset>>3;
    accZoffset += accZ;
  }
  accZ -= accZoffset>>3;
  applyDeadband(accZ, ACC_Z_DEADBAND);

  static int32_t lastBaroAlt;
  //int16_t baroVel = (alt.EstAlt - lastBaroAlt) * 1000000.0f / dTime;
  int16_t baroVel = (alt.EstAlt - lastBaroAlt) * (1000000 / UPDATE_INTERVAL);
  lastBaroAlt = alt.EstAlt;
```

```
    baroVel = constrain(baroVel, -300, 300); // constrain baro velocity +/- 300cm/s
    applyDeadband(baroVel, 10); // to reduce noise near zero

    // Integrator - velocity, cm/sec
    vel += accZ * ACC_VelScale * dTime;

    // apply Complimentary Filter to keep the calculated velocity based on baro velocity (i.e. near real velocity).
    // By using CF it's possible to correct the drift of integrated accZ (velocity) without loosing the phase, i.e without delay
    vel = vel * 0.985f + baroVel * 0.015f;

    //D
    alt.vario = vel;
    applyDeadband(alt.vario, 5);
    BaroPID -= constrain(conf.pid[PIDALT].D8 * alt.vario >>4, -150, 150);
  #endif
  return 1;
}
#endif //BARO
```

ข.4 Code คำสั่ง IMU.h

```
#ifndef IMU_H_
#define IMU_H_

#define BARO_TAB_SIZE   21

#if BARO
uint8_t getEstimatedAltitude();
#endif

void computeIMU();

#endif
```

ข.5 Code คำสั่ง Output.ccp

```
#include "Arduino.h"
#include "config.h"
#include "def.h"
#include "types.h"
#include "MultiWii.h"
void initializeSoftPWM(void);
#if defined(SERVO)
void initializeServo();
#endif
#if  defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) || (defined(MEGA)
&& defined(MEGA_HW_PWM_SERVOS))
#if (NUMBER_MOTOR > 4)
   //for HEX Y6 and HEX6/HEX6X/HEX6H flat for promini
   volatile uint8_t atomicPWM_PIN5_lowState;
   volatile uint8_t atomicPWM_PIN5_highState;
   volatile uint8_t atomicPWM_PIN6_lowState;
   volatile uint8_t atomicPWM_PIN6_highState;
 #endif
#if defined(SERVO)
 #if defined(HW_PWM_SERVOS)
  // hw servo pwm does not need atomicServo[]
 #elif defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6))
  #if defined(AIRPLANE) || defined(HELICOPTER)
   // To prevent motor to start at reset. atomicServo[7]=5 or 249 if reversed servo
   volatile uint8_t atomicServo[8] = {125,125,125,125,125,125,125,5};
  #else
   volatile uint8_t atomicServo[8] = {125,125,125,125,125,125,125,125};
  #endif
 #else
  #if defined(AIRPLANE)|| defined(HELICOPTER)
   // To prevent motor to start at reset. atomicServo[7]=5 or 249 if reversed servo
```

```
      volatile uint16_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,320};
    #else
      volatile uint16_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,8000};
    #endif
  #endif
#endif
#if defined(SERVO)
  #if defined(PRI_SERVO_FROM) && defined(SEC_SERVO_FROM)
    #if PRI_SERVO_FROM < SEC_SERVO_FROM
      #define SERVO_START PRI_SERVO_FROM
    #else
      #define SERVO_START SEC_SERVO_FROM
    #endif
  #else
    #if defined(PRI_SERVO_FROM)
      #define SERVO_START PRI_SERVO_FROM
    #endif
    #if defined(SEC_SERVO_FROM)
      #define SERVO_START SEC_SERVO_FROM
    #endif
  #endif
  #if defined(PRI_SERVO_TO) && defined(SEC_SERVO_TO)
    #if PRI_SERVO_TO > SEC_SERVO_TO
      #define SERVO_END PRI_SERVO_TO
    #else
      #define SERVO_END SEC_SERVO_TO
    #endif
  #else
    #if defined(PRI_SERVO_TO)
      #define SERVO_END PRI_SERVO_TO
    #endif
    #if defined(SEC_SERVO_TO)
      #define SERVO_END SEC_SERVO_TO
```

```
    #endif
  #endif


#endif
void writeServos() {
 #if defined(SERVO)
   #if defined(PRI_SERVO_FROM) && !defined(HW_PWM_SERVOS)   // write primary
servos
     for(uint8_t i = (PRI_SERVO_FROM-1); i < PRI_SERVO_TO; i++){
      #if   defined(PROMINI)   ||   (defined(PROMICRO)   &&   defined(HWPWM6))   ||
(defined(MEGA) && defined(MEGA_HW_PWM_SERVOS))
       atomicServo[i] = (servo[i]-1000)>>2;
      #else
       atomicServo[i] = (servo[i]-1000)<<4;
      #endif
     }
   #endif
   #if defined(SEC_SERVO_FROM) && !defined(HW_PWM_SERVOS)  // write secundary
servos
    #if          (defined(SERVO_TILT)||          defined(SERVO_MIX_TILT))          &&
defined(MMSERVOGIMBAL)
     // Moving Average Servo Gimbal by Magnetron1
     static                                                              int16_t
mediaMobileServoGimbalADC[3][MMSERVOGIMBALVECTORLENGHT];
     static int32_t mediaMobileServoGimbalADCSum[3];
     static uint8_t mediaMobileServoGimbalIDX;
     uint8_t axis;

     mediaMobileServoGimbalIDX      =      ++mediaMobileServoGimbalIDX      %
MMSERVOGIMBALVECTORLENGHT;
     for (axis=(SEC_SERVO_FROM-1); axis < SEC_SERVO_TO; axis++) {
```

```
            mediaMobileServoGimbalADCSum[axis]                    -=
mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];
            mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX] = servo[axis];
            mediaMobileServoGimbalADCSum[axis]                    +=
mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];
            #if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6))
              atomicServo[axis]    =    (mediaMobileServoGimbalADCSum[axis]    /
MMSERVOGIMBALVECTORLENGHT - 1000)>>2;
            #else
              atomicServo[axis]    =    (mediaMobileServoGimbalADCSum[axis]    /
MMSERVOGIMBALVECTORLENGHT - 1000)<<4;
            #endif
          }
        #else
          for(uint8_t i = (SEC_SERVO_FROM-1); i < SEC_SERVO_TO; i++){
            #if  defined(PROMINI)  ||  (defined(PROMICRO)  &&  defined(HWPWM6))  ||
(defined(MEGA) && defined(MEGA_HW_PWM_SERVOS))
              atomicServo[i] = (servo[i]-1000)>>2;
            #else
              atomicServo[i] = (servo[i]-1000)<<4;
            #endif
          }
        #endif
      #endif
      // write HW PWM servos for the mega
      #if defined(MEGA) && defined(MEGA_HW_PWM_SERVOS)
        #if (PRI_SERVO_FROM == 1 || SEC_SERVO_FROM == 1)
          OCR5C = servo[0];
        #endif
        #if (PRI_SERVO_FROM <= 2 && PRI_SERVO_TO >= 2) || (SEC_SERVO_FROM <= 2
&& SEC_SERVO_TO >= 2)
          OCR5B = servo[1];
        #endif
```

```
    #if (PRI_SERVO_FROM <= 3 && PRI_SERVO_TO >= 3) || (SEC_SERVO_FROM <= 3
&& SEC_SERVO_TO >= 3)
      OCR5A = servo[2];
    #endif
    #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4) || (SEC_SERVO_FROM <= 4
&& SEC_SERVO_TO >= 4)
      OCR1A = servo[3];
    #endif
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5) || (SEC_SERVO_FROM <= 5
&& SEC_SERVO_TO >= 5)
      OCR1B = servo[4];
    #endif
    #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6) || (SEC_SERVO_FROM <= 6
&& SEC_SERVO_TO >= 6)
      OCR4A = servo[5];
    #endif
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7) || (SEC_SERVO_FROM <= 7
&& SEC_SERVO_TO >= 7)
      OCR4B = servo[6];
    #endif
    #if (PRI_SERVO_FROM <= 8 && PRI_SERVO_TO >= 8) || (SEC_SERVO_FROM <= 8
&& SEC_SERVO_TO >= 8)
      OCR4C = servo[7];
    #endif
  #endif
  // write HW PWM servos for the promicro
  #if defined(PROMICRO) && defined(A32U4_4_HW_PWM_SERVOS)
    #if (PRI_SERVO_FROM <= 7 && PRI_SERVO_TO >= 7)
      OCR1A = servo[6];// Pin 9
    #endif
    #if (PRI_SERVO_FROM <= 5 && PRI_SERVO_TO >= 5)
      OCR1B = servo[4];// Pin 10
    #endif
```

```
      #if (PRI_SERVO_FROM <= 6 && PRI_SERVO_TO >= 6)
        OCR3A = servo[5];// Pin 5
      #endif
      #if (PRI_SERVO_FROM <= 4 && PRI_SERVO_TO >= 4)
        OCR1C = servo[3];// Pin 11
      #endif
    #endif
  #endif
}
void writeMotors() { // [1000;2000] => [125;250]
  /****************        Specific PWM Timers & Registers for the MEGA's
*******************/
  #if defined(MEGA)// [1000:2000] => [8000:16000] for timer 3 & 4 for mega
    #if (NUMBER_MOTOR > 0)
      #ifndef EXT_MOTOR_RANGE
        OCR3C = motor[0]<<3; //  pin 3
      #else
        OCR3C = ((motor[0]<<4) - 16000);
      #endif
    #endif
    #if (NUMBER_MOTOR > 1)
      #ifndef EXT_MOTOR_RANGE
        OCR3A = motor[1]<<3; //  pin 5
      #else
        OCR3A = ((motor[1]<<4) - 16000);
      #endif
    #endif
    #if (NUMBER_MOTOR > 2)
      #ifndef EXT_MOTOR_RANGE
        OCR4A = motor[2]<<3; //  pin 6
      #else
        OCR4A = ((motor[2]<<4) - 16000);
      #endif
```

```
    #endif
    #if (NUMBER_MOTOR > 3)
     #ifndef EXT_MOTOR_RANGE
       OCR3B = motor[3]<<3; //  pin 2
     #else
       OCR3B = ((motor[3]<<4) - 16000);
     #endif
    #endif
    #if (NUMBER_MOTOR > 4)
     #ifndef EXT_MOTOR_RANGE
       OCR4B = motor[4]<<3; //  pin 7
       OCR4C = motor[5]<<3; //  pin 8
     #else
       OCR4B = ((motor[4]<<4) - 16000);
       OCR4C = ((motor[5]<<4) - 16000);
     #endif
    #endif
    #if (NUMBER_MOTOR > 6)
     #ifndef EXT_MOTOR_RANGE
       OCR2B = motor[6]>>3; //  pin 9
       OCR2A = motor[7]>>3; //  pin 10
     #else
       OCR2B = (motor[6]>>2) - 250;
       OCR2A = (motor[7]>>2) - 250;
     #endif
    #endif
   #endif
  #endif

  /******** Specific PWM Timers & Registers for the atmega32u4 (Promicro)
************/
  #if defined(PROMICRO)
   #if (NUMBER_MOTOR > 0)
    #if defined(A32U4_4_HW_PWM_SERVOS)
```

```
        // write motor0 to pin 6
        // Timer 4 A & D [1000:2000] => [1000:2000]
        #ifndef EXT_MOTOR_RANGE
          TC4H = motor[0]>>8; OCR4D = (motor[0]&0xFF); //  pin 6
        #else
          TC4H    =    (((motor[0]-1000)<<1)+16)>>8;    OCR4D    =    ((((motor[0]-
1000)<<1)+16)&0xFF); //  pin 6
        #endif
      #else
        // write motor0 to pin 9
        // Timer 1 A & B [1000:2000] => [8000:16000]
        #ifndef EXT_MOTOR_RANGE
          OCR1A = motor[0]<<3; //  pin 9
        #else
          OCR1A = ((motor[0]<<4) - 16000) + 128;
        #endif
      #endif
    #endif
    #if (NUMBER_MOTOR > 1)
      #ifndef EXT_MOTOR_RANGE
        OCR1B = motor[1]<<3; //  pin 10
      #else
        OCR1B = ((motor[1]<<4) - 16000) + 128;
      #endif
    #endif
    #if (NUMBER_MOTOR > 2) // Timer 4 A & D [1000:2000] => [1000:2000]
      #if !defined(HWPWM6)
        // to write values > 255 to timer 4 A/B we need to split the bytes
        #ifndef EXT_MOTOR_RANGE
          TC4H = (2047-motor[2])>>8; OCR4A = ((2047-motor[2])&0xFF); //  pin 5
        #else
          TC4H  =  2047-(((motor[2]-1000)<<1)+16)>>8;  OCR4A  =  (2047-((((motor[2]-
1000)<<1)+16)&0xFF); //  pin 5
```

```
      #endif
     #else
      #ifndef EXT_MOTOR_RANGE
       OCR3A = motor[2]<<3; //  pin 5
      #else
       OCR3A = ((motor[2]<<4) - 16000) + 128;
      #endif
     #endif
    #endif
    #if (NUMBER_MOTOR > 3)
     #ifndef EXT_MOTOR_RANGE
      TC4H = motor[3]>>8; OCR4D = (motor[3]&0xFF); //  pin 6
     #else
      TC4H = (((motor[3]-1000)<<1)+16)>>8; OCR4D = ((((motor[3]-
1000)<<1)+16)&0xFF); //  pin 6
     #endif
    #endif
    #if (NUMBER_MOTOR > 4)
     #if !defined(HWPWM6)
      #if (NUMBER_MOTOR == 6) && !defined(SERVO)
       atomicPWM_PIN5_highState = motor[4]<<3;
       atomicPWM_PIN5_lowState = 16383-atomicPWM_PIN5_highState;
       atomicPWM_PIN6_highState = motor[5]<<3;
       atomicPWM_PIN6_lowState = 16383-atomicPWM_PIN6_highState;
      #else
       atomicPWM_PIN5_highState = ((motor[4]-1000)<<4)+320;
       atomicPWM_PIN5_lowState = 15743-atomicPWM_PIN5_highState;
       atomicPWM_PIN6_highState = ((motor[5]-1000)<<4)+320;
       atomicPWM_PIN6_lowState = 15743-atomicPWM_PIN6_highState;
      #endif
     #else
      #ifndef EXT_MOTOR_RANGE
       OCR1C = motor[4]<<3; //  pin 11
```

```
        TC4H = motor[5]>>8; OCR4A = (motor[5]&0xFF); //  pin 13
      #else
        OCR1C = ((motor[4]<<4) - 16000) + 128;
        TC4H    =    (((motor[5]-1000)<<1)+16)>>8;    OCR4A    =    ((((motor[5]-
1000)<<1)+16)&0xFF); //  pin 13
      #endif
    #endif
  #endif
  #if (NUMBER_MOTOR > 6)
    #if !defined(HWPWM6)
      atomicPWM_PINA2_highState = ((motor[6]-1000)<<4)+320;
      atomicPWM_PINA2_lowState = 15743-atomicPWM_PINA2_highState;
      atomicPWM_PIN12_highState = ((motor[7]-1000)<<4)+320;
      atomicPWM_PIN12_lowState = 15743-atomicPWM_PIN12_highState;
    #else
      atomicPWM_PINA2_highState = ((motor[6]-1000)>>2)+5;
      atomicPWM_PINA2_lowState = 245-atomicPWM_PINA2_highState;
      atomicPWM_PIN12_highState = ((motor[7]-1000)>>2)+5;
      atomicPWM_PIN12_lowState = 245-atomicPWM_PIN12_highState;
    #endif
  #endif
#endif

/********  Specific PWM Timers & Registers for the atmega328P (Promini)
**********/
#if defined(PROMINI)
  #if (NUMBER_MOTOR > 0)
    #ifndef EXT_MOTOR_RANGE
      OCR1A = motor[0]>>3; //  pin 9
    #else
      OCR1A = ((motor[0]>>2) - 250);
    #endif
  #endif
```

```
#if (NUMBER_MOTOR > 1)
  #ifndef EXT_MOTOR_RANGE
    OCR1B = motor[1]>>3; //  pin 10
  #else
    OCR1B = ((motor[1]>>2) - 250);
  #endif
#endif
#if (NUMBER_MOTOR > 2)
  #ifndef EXT_MOTOR_RANGE
    OCR2A = motor[2]>>3; //  pin 11
  #else
    OCR2A = ((motor[2]>>2) - 250);
  #endif
#endif
#if (NUMBER_MOTOR > 3)
  #ifndef EXT_MOTOR_RANGE
    OCR2B = motor[3]>>3; //  pin 3
  #else
    OCR2B = ((motor[3]>>2) - 250);
  #endif
#endif
#if (NUMBER_MOTOR > 4)
  #if (NUMBER_MOTOR == 6) && !defined(SERVO)
    #ifndef EXT_MOTOR_RANGE
      atomicPWM_PIN6_highState = motor[4]>>3;
      atomicPWM_PIN5_highState = motor[5]>>3;
    #else
      atomicPWM_PIN6_highState = (motor[4]>>2) - 250;
      atomicPWM_PIN5_highState = (motor[5]>>2) - 250;
    #endif
    atomicPWM_PIN6_lowState  = 255-atomicPWM_PIN6_highState;
    atomicPWM_PIN5_lowState  = 255-atomicPWM_PIN5_highState;
  #else //note: EXT_MOTOR_RANGE not possible here
```

```
        atomicPWM_PIN6_highState = ((motor[4]-1000)>>2)+5;
        atomicPWM_PIN6_lowState  = 245-atomicPWM_PIN6_highState;
        atomicPWM_PIN5_highState = ((motor[5]-1000)>>2)+5;
        atomicPWM_PIN5_lowState  = 245-atomicPWM_PIN5_highState;
      #endif
    #endif
    #if (NUMBER_MOTOR > 6) //note: EXT_MOTOR_RANGE not possible here
      atomicPWM_PINA2_highState = ((motor[6]-1000)>>2)+5;
      atomicPWM_PINA2_lowState  = 245-atomicPWM_PINA2_highState;
      atomicPWM_PIN12_highState = ((motor[7]-1000)>>2)+5;
      atomicPWM_PIN12_lowState  = 245-atomicPWM_PIN12_highState;
    #endif
  #endif
}
void writeAllMotors(int16_t mc) {   // Sends commands to all motors
  for (uint8_t i =0;i<NUMBER_MOTOR;i++) {
    motor[i]=mc;
  }
  writeMotors();
}
```

ข.6 Code คำสั่ง Config.h

```
#ifndef CONFIG_H_
#define CONFIG_H_
#define QUADX
#define MINTHROTTLE 1300 // for Turnigy Plush ESCs 10A
#define MAXTHROTTLE 2000
#define MINCOMMAND  1000

#define  CRIUS_SE_v2_0     // Crius MultiWii SE 2.0 with MPU6050, HMC5883 and BMP085
#define PID_CONTROLLER 1
#define YAW_DIRECTION -1
#define ONLYARMWHENFLAT //prevent the copter from arming when the copter is tilted
#define ALLOW_ARM_DISARM_VIA_TX_YAW
#define CAM_TIME_HIGH 1000   // the duration of HIGH state servo expressed in ms
#define FLAPPERON_EP   { 1500, 1700 } // Endpooints for flaps on a 2 way switch else set {1020,2000} and program in radio.
#define FLAPPERON_INVERT { -1, 1 }    // Change direction om flapperons { Wing1, Wing2 }
#define COLLECTIVE_PITCH      THROTTLE
#define COLLECTIVE_RANGE { 80, 0, 80 }// {Min%, ZeroPitch offset from 1500, Max%}.
#define YAWMOTOR                0    // If a motor is used as YAW Set to 1 else set to 0.
#define SERVO_NICK   { +10, -10,  0 }
#define SERVO_LEFT   { +10, +5, +10 }
#define SERVO_RIGHT  { +10, +5, -10 }
#define CONTROL_RANGE   { 100, 100 }      // { ROLL,PITCH }
#define SBUS_MID_OFFSET 988 //SBUS Mid-Point at 1500
#define RCAUXPIN8
#define SERIAL0_COM_SPEED 115200
#define SERIAL1_COM_SPEED 115200
```

```
#define SERIAL2_COM_SPEED 115200
#define SERIAL3_COM_SPEED 115200
#define INTERLEAVING_DELAY 3000
#define NEUTRALIZE_DELAY 100000
#define AP_MODE 40
#define FAILSAFE_DELAY     10
#define FAILSAFE_OFF_DELAY 200
#define FAILSAFE_THROTTLE  (MINTHROTTLE + 200)
#define FAILSAFE_DETECT_TRESHOLD  985
#define GPS_BAUD   57600
#define GPS_LED_INDICATOR
#define NAV_CONTROLS_HEADING       true     // copter faces toward the navigation
point, maghold must be enabled for it
#define NAV_TAIL_FIRST             false    // true - copter comes in with tail first
#define NAV_SET_TAKEOFF_HEADING    true     // true - when copter arrives to
home position it rotates it's head to takeoff direction
#define MAG_DECLINATION  0.0f
#define GPS_LEAD_FILTER
#define GPS_WP_RADIUS              200
#define NAV_SLEW_RATE              30
#define LCD_SERIAL_PORT 0
#define LCD_MENU_PREV 'p'
#define LCD_MENU_NEXT 'n'
#define LCD_VALUE_UP 'u'
#define LCD_VALUE_DOWN 'd'
#define LCD_MENU_SAVE_EXIT 's'
#define LCD_MENU_ABORT 'x'
#define VBATSCALE        131 // (*) (**) change this value if readed Battery voltage is
different than real voltage
#define VBATNOMINAL      126 // 12,6V full battery nominal voltage - only used for
lcd.telemetry
#define VBATLEVEL_WARN1 107 // (*) (**) 10,7V
#define VBATLEVEL_WARN2  99 // (*) (**) 9.9V
```

```
#define VBATLEVEL_CRIT    93 // (*) (**) 9.3V - critical condition: if vbat ever goes below this value, permanent alarm is triggered
#define NO_VBAT           16 // Avoid beeping without any battery
#define PSENSORNULL 510 /* (*) hard only: set to analogRead() value for zero current; for I=0A my sensor
#define PINT2mA 132
```

ภาคผนวก ค

โปสเตอร์

รูปที่ ค.1 โปสเตอร์รถเครื่องบินปีกหมุน 4 ใบพัด