การศึกษาการสร้างอากาศยานไร้คนขับสำหรับภาควิชาการวัดคุม

A RESEARCH AND DEVELOPMENT OF UNMANNED AERIAL VEHICLE (UAV) FOR INSTRUMENTATION ENGINEERING DEPARTMENT

จารุสิทธิ์ พูดเพราะ

เจษฎากร ชวนชม

ภูมิภัทร สิงห์พรหมสาร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2563

A RESEARCH AND DEVELOPMENT OF UNMANNED AERIAL VEHICLE (UAV) FOR INSTRUMENTATION ENGINEERING DEPARTMENT

Jarusit Poodprox
Jedsadakorn Chuanchom
Phoomphat Singhaphromsarn

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING
FACULTY OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
ACADEMIC YEAR 2020

ปริญญานิพนธ์ปีการศึกษา 2563

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

.....

หัวข้อปริญญานิพนธ์ การศึกษาการสร้างอากาศยานไร้คนขับสำหรับภาควิชาการวัดคุม
A RESEARCH AND DEVELOPMENT OF UNMANNED AERIAL VEHICLE (UAV) FOR INSTRUMENTATION ENGINEERING DEPARTMENT

นักศึกษาผู้จัดทำ นายจารุสิทธิ์ พูดเพราะ **รหัสนักศึกษา** 60010130

นายเจษฎากร ชวนชม **รหัสนักศึกษา** 60010168

นายภูมิภัทร สิงห์พรหมสาร **รหัสนักศึกษา** 60010803

ปริญญา วิศวกรรมศาสตร์บัณฑิต

สาขาวิชา วิศวกรรมการวัดคุม

ปีการศึกษา 2563

อาจารย์ผู้ควบคุมปริญญานิพนธ์	**ลายมือชื่อ**
ผศ.ดร.นภศูล วงษ์วานิช	
รศ.วิริยะ กองรัตน์	

หัวข้อปริญญานิพนธ์	การศึกษาการสร้างอากาศยานไร้คนขับสำหรับภาควิชาการวัดคุม A RESEARCH AND DEVELOPMENT OF UNMANNED AERIAL VEHICLE (UAV) FOR INSTRUMENTATION ENGINEERING DEPARTMENT		
นักศึกษาผู้จัดทำ	นายจารุสิทธิ์	พูดเพราะ	รหัสนักศึกษา 60010130
	นายเจษฎากร	ชวนชม	รหัสนักศึกษา 60010168
	นายภูมิภัทร	สิงห์พรหมสาร	รหัสนักศึกษา 60010803
อาจารย์ที่ปรึกษา	ผศ.ดร.นภศูล	วงษ์วานิช	
	รศ.วิริยะ	กองรัตน์	
ปีการศึกษา	2563		

บทคัดย่อ

ในยุคสมัยปัจจุบันความสนใจในด้านอากาศยานไร้คนขับต่างเป็นที่นิยมมากขึ้นอย่างต่อเนื่อง โครงงานนี้จะอธิบายถึงขั้นตอนการสร้างและวิธีการควบคุมอากาศยานไร้คนขับ โดยใช้ Fast Complementary Filter คือ Sensor Fusion ที่จะนำค่าของเซนเซอร์ 3 ตัว บน GY-86 นำมาคำนวณเพื่อหาค่า Roll Pitch Yaw System identification คือ ระบบในการนำค่าที่วัดได้มาใส่ในสมการ Characteristic เพื่อหาค่าตัวค่าคงที่ที่มีผลต่อการเคลื่อนที่ของอากาศยานไร้คนขับ PID controller (proportional–integral–derivative controller) คือ ระบบควบคุมที่ทำให้ตัวอากาศยานไร้คนขับรักษาระดับได้อย่างคงที่ และมีจุดประสงค์เพื่อพัฒนาหรือต่อยอดให้แก่บุคคลที่จะนำไปใช้ต่อ โดยวิธีที่ใช้จะสร้างจากองค์ความรู้ต่าง ๆที่ได้ทำการศึกษามาโดยแตกต่างจากวิธีทั่วไปโดยอากาศยานไร้คนขับสามารถควบคุมได้ทั้งระบบ Automatic และ Manual

Thesis Title	A RESEARCH AND DEVELOPMENT OF UNMANNED AERIAL VEHICLE (UAV) FOR INSTRUMENTATION ENGINEERING DEPARTMENT	
Authors	Mr. Jarusit	Poodprox
	Mr. Jedsadakorn	Chuanchom
	Mr. Phoomphat	Singhaphromsarn
Thesis Advisor	Asst.Prof.Dr.Napasool	Wongvanich
	Assoc.Prof.Viriya	Kongratana
Year	2020	

ABSTRACT

Nowadays, the interest in Unmanned Aerial Vehicles(UAV) is steadily growing in popularity. This project explains the process of building and how to operate an UAV using the Fast Complementary Filter, the Sensor Fusion that will bring The values of the three sensors on the GY-86 are calculated to determine the Roll Pitch Yaw, System identification is a system for applying the measured values into the Characteristic equation to determine the constants that affect the movement of UAV. A casacaded PID controller (proportional – integral – derivative controller) is also designed on unmanned aircraft to maintain a constant level in both auto and manual modes. The main goal of this control is also provide a solid platform for a closed loop system identification process whichis to be conducted in the future. Result show that the design cascaded PID controller yields an average leteral error of x y z and an average attitude error of roll pitch yaw.

กิตติกรรมประกาศ

ปริญญานิพนธ์เรื่อง การศึกษาการสร้างอากาศยานไร้คนขับสำหรับภาควิชาการวัดคุม (A RESEARCH AND DEVELOPMENT OF UNMANNED AERIAL VEHICLE (UAV) FOR INSTRUMENTATION ENGINEERING DEPARTMENT) สำเร็จลุล่วงไปได้ด้วยดี เนื่องจากได้รับความอุปถัมภ์จากบุคคลหลายฝ่ายที่ให้คำปรึกษา และชี้แนวทาง ทำให้ปริญญานิพนธ์ฉบับนี้บรรลุเป้าหมายตามวัตถุประสงค์ได้เป็นอย่างดี

คณะผู้จัดทำขอขอบพระคุณ ผศ.ดร.นภศูล วงศ์วาณิช และรศ.วิริยะ กองรัตน์ อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้คำปรึกษา ชี้แนะ ให้ข้อคิดในการแก้ไขปัญหา รวมถึงช่วยตรวจทานแก้ไขข้อบกพร่องต่าง ๆ ทำให้ปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ นอกจากนี้คณะผู้จัดทำขอขอบพระคุณคณะอาจารย์ประจำสาขาวิชาวิศวกรรมการวัดคุม ที่ให้คำปรึกษา

ขอขอบพระคุณชุมนุมโรบอท ที่ให้คำปรึกษาในด้านเทคนิคการทำหุ่นยนต์ อีกทั้งยังเอื้อเฟื้ออุปกรณ์ และเครื่องมือในการทำงานต่าง ๆ

และสุดท้ายนี้คณะผู้จัดทำขอกราบขอบพระคุณ บิดา มารดา และครอบครัว เป็นอย่างสูง สำหรับความรัก กำลังใจ คำปรึกษา และการสนับสนุนในด้านต่าง ๆ ที่มอบให้อยู่อย่างสม่ำเสมอจนเกิดเป็นแรงผลักดัน ทำให้โครงงานนี้ประสบความสำเร็จ และผ่านลุล่วงมาได้อย่างสมบูรณ์

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูปภาพ

รูปที่ **หน้า**

สารบัญรูปภาพ (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 สาระสำคัญของโครงการ

โครงการนี้ต่อเนื่องมาจากการฝึกงานช่วงฤดูร้อนที่ผ่านมา ศึกษา ออกแบบและพัฒนาเกี่ยวกับการสร้างอากาศยานไร้คนขับสำหรับภาควิชาการวัดคุม โดยที่เราจะศึกษา Sensor Fusion และ Fast Complementary Filter ของระบบ โดยใช้วิธีการเขียนโปรแกรมผ่าน Arduino และคำนวณผ่าน MATLAB เพื่อคำนวณหาค่าการเคลื่อนที่ของโดรนซึ่งประกอบไปด้วย Pitch Roll และ Yaw รวมทั้งศึกษา PID Controller เพื่อที่จะควบคุมอากาศยานให้ถึงจุด Set Point และไม่เกิดค่าความผิดพลาดขึ้นมา ซึ่งในปัจจุบันนี้ อากาศยานไร้คนขับได้เป็นที่แพร่หลายเป็นอย่างมาก ในการประยุกต์ใช้งานต่าง ๆ เช่น การใช้งานในเกษตรกรรมในการฉีดพ่น การใช้งานในการดับเพลิง เป็นต้น จะเห็นได้ว่าอากาศยานไร้คนขับมีประโยชน์อย่างมากเนื่องจากสามารถทำอะไรได้หลายอย่างที่ตัวเราเองไม่สามารถทำมันได้ เราจึงจำเป็นต้องศึกษา พัฒนาและควบคุมอากาศยานของเราเองเพื่อที่จะสามารถนำไปใช้ประยุกต์ได้ โดยการศึกษาของเราจึงจำเป็นต้องศึกษา Sensor Fusion ซึ่งเป็นหัวใจสำคัญของอากาศยานไร้คนขับ ซึ่งจะสามารถวัดค่าซึ่งได้แก่ Gyro Accelerate และ Magnetic ถ้าเราสามารถวัด 3 ค่านี้ได้ เราจะสามารถนำ 3 ค่า นี้ไปควบคุมใน PID Controller อีกทั้งเรายังศึกษา Complementary Filter เพื่อลดความผิดพลาดของค่าที่เราอ่านได้ให้มีความแม่นยำมากขึ้นด้วย

1.2 วัตถุประสงค์

1. เพื่อต่อยอดการศึกษาสร้างอากาศไร้คนขับจากช่วงฝึกงานฤดูร้อนให้สมบูรณ์
2. ศึกษา Sensor Fusion เพื่อที่จะวัดค่า Pitch Roll และ Yaw
3. ศึกษา PID Controller และ Superfast Attitude Filter เพื่อควบคุมอากาศไร้คนขับ
4. ฝึกการเขียนโปรแกรม Arduino และ MATLAB ด้วยภาษา C
5. เพื่อนำประยุกต์ใช้ให้เกิดประโยนชน์ได้

1.3 ขอบเขตของปริญญานิพนธ์

1. ศึกษา Superfast Attitude Filter ในการทำ Sensor Fusion เพื่อที่จะทำการอ่านค่า roll, pitch, yaw ที่ได้มาจากการวัดค่า accelerometer, magnetic
2. ศึกษาการอ่านค่าความสูงจาก barometer
3. ศึกษา Math model ของระบบเพื่อที่นำค่าที่วัดได้มาคำนวณ
4. ออกแบบการเขียนโปรแกรมเพื่อที่จะใช้งานการควบคุมอากาศยานไร้คนขับ

5. ศึกษาและออกแบบ PID Controller เพื่อควบคุมตำแหน่งของอากาศยาน

1.4 ขั้นตอนการศึกษา

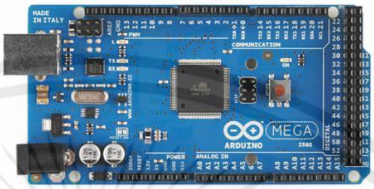
1. ศึกษารูปแบบการเคลื่อนที่ของอากาศยานไร้คนขับ
2. ศึกษาทำงานของมอเตอร์ และBrushless Driver ชนิด Speed Controller 30A
3. ศึกษาการทำงานของไมโครคอนโทรลเลอร์ Arduino MEGA 2560
4. ศึกษาการทำงานของอุปกรณ์ตรวจรู้ GY-86
5. ศึกษาการแก้ปัญหาคณิตศาสตร์ผ่านโปรแกรม MATLAB
6. ศึกษาแบบจำลองทางคณิตศาสตร์(Math model) เกี่ยวข้องกับการสร้างอากาศยานไร้คนขับ
7. ศึกษาการเขียนโปรแกรม Arduino เพื่อควบคุมการเคลื่อนที่ของอากาศไร้คนขับ
8. ศึกษาการออกแบบตัวควบคุม PID ของอากาศยานไร้คนขับ

1.5 ปัญหาหรือประโยชน์ที่ได้รับ

1. เข้าใจการทำงานของอุปกรณ์ตรวจรู้ GY-86
2. ศึกษาและเข้าใจ Math model ในการศึกษาการสร้างอากาศไร้คนขับ
3. สามารถออกแบบตัวควบคุม PID ของอากาศไร้คนขับได้
4. ได้ออกแบบและวางแผนการทำงานอย่างเป็นระบบ
5. สามารถนำไปต่อยอดเพื่อสร้างรายได้

บทที่ 2
ทฤษฎีพื้นฐานที่เกี่ยวข้อง

2.1 Arduino MEGA 2560



รูปที่ 2.1 Arduino MEGA 2560

Arduino Mega 2560 เป็นบอร์ดไมโครคอนโทรลเลอร์ที่ใช้ชิพ ATmaga2560 ซึ่งมี 54 ดิจิตอล อินพุต/เอาท์พุต โดยในขาเหล่านั้นสามารถใช้งานเป็น PWM ได้ 15 ขา, อนาล็อคอินพุต 16 ขา, UART 4 ชุด โดยความถี่คริสตรัลบนบอร์ดคือ 16 MHz เชื่อมต่อข้อมูลระหว่างคอมพิวเตอร์ผ่านพอร์ต USB บนบอร์ดได้โดยตรง อีกทั้งรูปแบบการออกแบบยังออกแบบให้รองรับการสวมกับ Shield ต่าง ๆได้โดยตรง ทำให้สามารถพัฒนาระบบต่าง ๆ ได้อย่างรวดเร็วและ เรียบร้อยสวยงาม โดยรองรับการพัฒนาโปรแกรมบนแพลตฟอร์ม Arduino อย่างเต็มรูปแบบ

2.2 GY-86 10-DoF IMU (MPU6050 HMC5883L MS5611)

รูปที่ 2.2 GY-86 10-DoF IMU (MPU6050 HMC5883L MS5611)

GY-86 เป็นโมดูล Accelerometers , Gyroscope , Compass , Pressure ในตัวเดียวกัน บนโมดูลประกอบด้วยชิป MPU6050 HMC5883L MS5611 ส่งข้อมูลผ่าน Bus I2C ใช้ในการหาค่า

ของการเคลื่อนไหวทั้ง 3 แกน และทิศทางของการเคลื่อนที่ พร้อมทั้งการหาค่าของความกดอากาศได้อีกด้วย

2.2.1 MPU 6050

MPU 6050 คือโมดูลเซนเซอร์ที่ตรวจจับการเคลื่อนไหวและความเอียงของวัตถุ โดยตรวจวัดจากความเร่งเชิงเส้น (Linear Acceleration) และ ความเร็วเชิงมุม (Angular Velocity หรือใช้ Gyroscope) ถือเป็นอุปกรณ์แบบ 6 DOF (6 Degrees of Freedom) คืออุปกรณ์ที่สามารถตรวจวัดค่าได้ทั้ง 6 แกนคือ Ax, Ay, Az, Gx, Gy และ Gz

2.2.2 HMC 5883l

HMC 5883l คือ เซ็นเซอร์เข็มทิศ สามารถตรวจจับทิศทางที่กำลังหันหน้าไปหา รวมไปถึงความเอียง โดยเซ็นเซอร์ตัวนี้สามารถจับทิศทางได้ทั้งหมด 3 แกน คือ X Y Z

2.2.3 MS 5611

MS 5611 คือเซ็นเซอร์บารอมิเตอร์ที่สามารถวัดค่า ความดัน และ อุณหภูมิได้ ณ ที่ความสูงใด ๆ แล้วนำไปสามารถแปลงจากค่าความดันเป็นความสูงได้

2.3 Brushless Motor DC

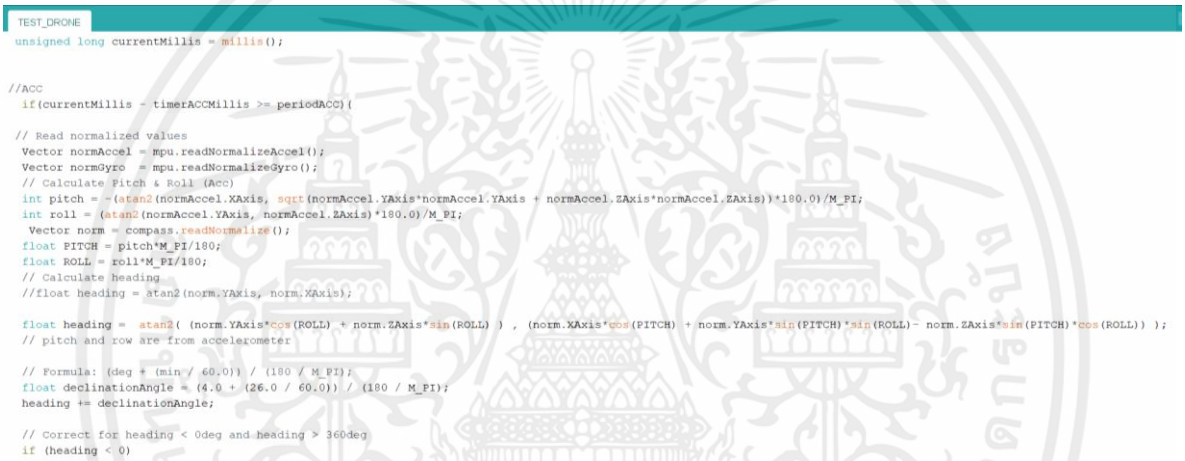
รูปที่ 2.3 Brushless Motor DC

เป็นมอเตอร์ไฟฟ้ากระแสตรง หรือ มอเตอร์ไฟฟ้า DC โดยส่วนหนึ่งจะแปรงเข้าถ่านเข้าคอมมิวเตเตอร์เข้าไปในขดลวดอาร์เมอเจอร์สร้างสนามแม่เหล็กขึ้นและกระแสไฟฟ้าอีกส่วนหนึ่งจะไหลเข้าไปในขดลวดสนามแม่เหล็ก สร้างขั้วเหนือและใต้จะเกิดสนามแม่เหล็ก 2 สนามในขณะเดียวกัน โดยก็จะเกิดตามคุณสมบัติของเส้นแรงแม่เหล็ก ถ้าคนละทิศทางจะหักล้างกัน แต่ถ้าทิศเดียวกันจะเสริมแรงกันในตัวของอารเมเจอร์ซึ่งสวมอยู่กับตลับลูกปืน ทำให้เกิดการหมุนได้เนื่องจากอำนาจของสนามแม่เหล็ก

ส่วน Brushless เป็นตัว IC ที่เอาไว้สามารถควบคุมความเร็วของมอเตอร์ เป็นโปรแกรมพิเศษหลักสำหรับตัวควบคุมหลายใบพัดช่วยเพิ่มการตอบสนองคันเร่ง ควบคุมการบินต่างๆและรองรับความถี่สัญญาณของถึง 612 HZ ไดรฟ์พิเศษมี MOSFET ทำให้มีประสิทธิภาพที่ดี และให้ความมั่นคง

2.4 Arduino IDE

เครื่องมือการเขียนโปรแกรมที่ใช้งานกับ Arduino ได้ทุกรุ่น โดยภายในจะมีเครื่องมือสำหรับติดต่อ Arduino เช่น การค้นหา Arduino ที่ติดต่อกับเครื่องคอมพิวเตอร์ การเลือกรุ่น Arduino ที่ต่ออยู่เพื่อตรวจสอบว่าขนาดของโปรแกรมที่เขียน หรือไรบรารี่ต่างๆซับพอร์ตกับ Arduino รุ่นนั้นๆไหม อีกทั้งยังมีโปรแกรมติดต่อผ่านซีเรียลโดยตรงสำหรับคอมพิวเตอร์



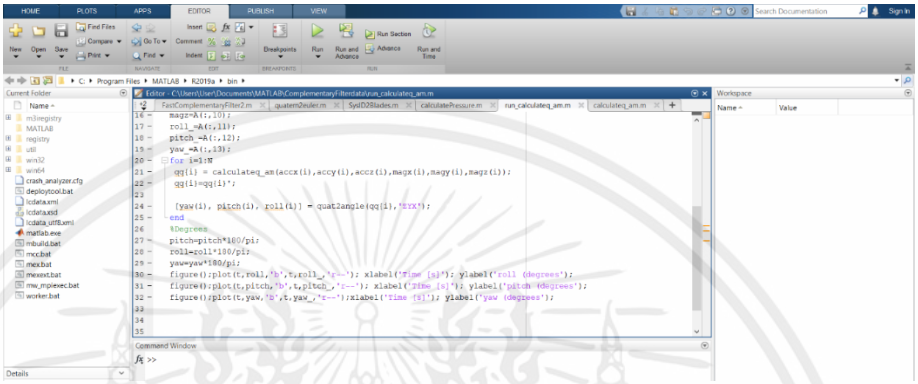
รูปที่ 2.4 หน้าต่างโปรแกรมของ Arduino IDE

2.5 MATLAB

รูปที่ 2.5 โปรแกรม MATLAB

MATLAB เป็นโปรแกรมที่สามารถทำงานได้ทั้งในลักษณะของการติดต่อโดยตรง คือการเขียนคำสั่งเข้าไปทีละคำสั่ง เพื่อให้แมตแล็บประมวลผลไปเรื่อยๆ หรือสามารถที่จะรวบรวม ชุดคำสั่งเรานั้น

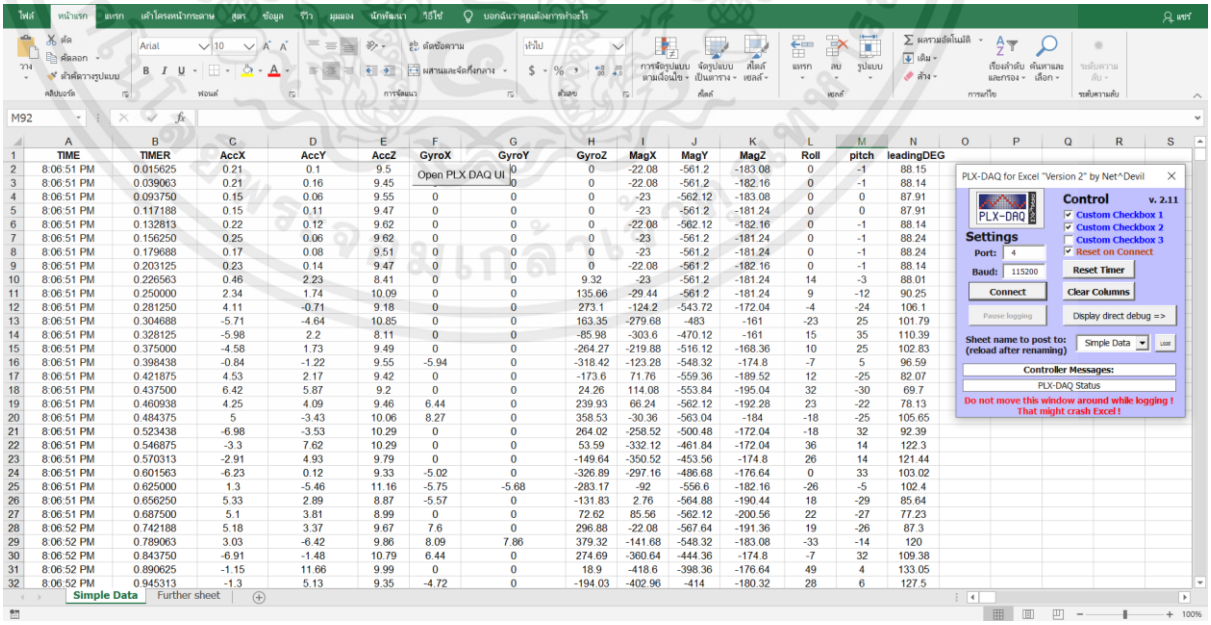
เป็นโปรแกรมก็ได้ ข้อสำคัญอย่างหนึ่งของแมตแล็บก็คือข้อมูลทุกตัวจะถูกเก็บใน ลักษณะของแถวลำดับ คือในแต่ละตัวแปรจะได้รับการแบ่งเป็นส่วนย่อยเล็กๆขึ้น ซึ่งการใช้ตัวแปรเป็นแถวลำดับ ในแมตแล็บเราไม่จำเป็นที่จะต้องจองมิติเหมือนกับ การเขียนโปรแกรมในภาษาขั้นต่ำทั่วไป ซึ่งทำให้เราสามารถที่จะแก้ปัญหาของตัวแปรที่อยู่ในลักษณะ ของเมทริกซ์และเวกเตอร์ได้โดยง่าย ซึ่งทำให้เราลดเวลาการทำงานลงได้อย่างมากเมื่อเทียบกับการเขียน โปรแกรมโดยภาษาซีหรือภาษาฟอร์แทรน



รูปที่ 2.6 เป็นการโปรแกรมและคำนวณใน MATLAB

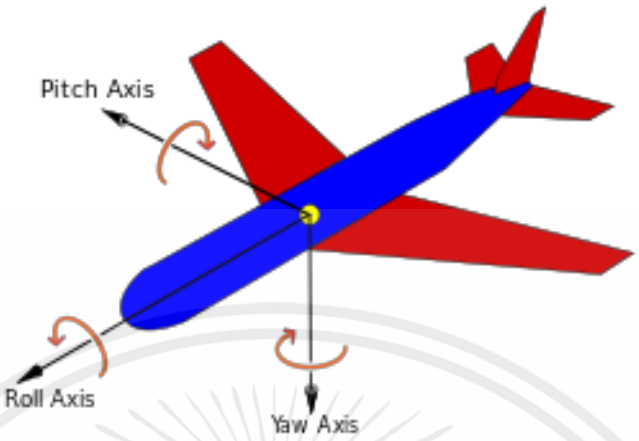
2.6 PLX-DAQ

PLX-DAQ เป็นโปรแกรมที่ทำหน้าที่เชื่อมต่อระหว่างบอร์ด Arduino กับคอมพิวเตอร์ เพื่อส่งข้อมูลไปยังโปรแกรม Excel ข้อมูลตรงกับ Serial monitor ใน Arduino IDE



รูปที่ 2.7 การอัปโหลดข้อมูลใน Arduino เข้า PLX-DAQ

2.7 หลักการบินของQuadrotor (UAV Mechanics)



รูปที่ 2.8 ลักษณะของแกนที่ใช้ในการเคลื่อนที่ของการบิน

การบินของ Quadrotor จะการเคลื่อนที่ 4 ทิศทาง จะเรียกว่า 4CH คือประกอบด้วย ขึ้น-ลง ,เดินหน้า-ถอยหลัง ,เอียงซ้าย-เอียงขวา และ หมุนซ้าย-หมุนขวา มันทำได้ไง ใบพัดหมุนอย่างไง มันถึงบินได้ ความจริงจะมี quadrotor แบบ 6 ใบพัด 8 ใบพัด เทคนิคการควบคุมจะคล้ายๆ แบบนี้

1. Hovering หรือ การลอยตัวเฉยๆ ทำได้โดยควบคุมให้ความเร็ว ใบพัดทั้งสี่ตัว มีความเร็วที่เท่ากัน เพื่อสร้างแรงบิดและหักล้างแรงบิด ดูจากรูปจะเห็นว่า ใบพัดจะหมุนกันคนล่ะทิศทาง ใบพัดหน้าและหลังจะหมุนตามเข็มใบพัดซ้ายและขวาจะหมุนทวนเข็มครับทำให้เครื่องบินไม่หมุนตัว

2. Throttle คันเร่ง ความเร็ว ให้เครื่องบิน บิน ขึ้นลง ใบพัดทั้งสี่ใบจะต้องเพิ่มความเร็ว ทุกใบพัดที่เท่ากันทำให้เครื่องบินลอยตัวขึ้นได้

3. Roll เอียงตัวซ้าย-ขวา ใบพัด หน้า(FRONT) หลัง(REAR) จะความเร็วเท่าเดิม แต่ความเร็วใบพัดซ้าย (LEFT) จะหมุนเร็วขึ้น ทิศทางนี้จะยกตัว ใบพัดขวา (Right) จะช้าลงทิศทางนี้จะตกลง จึงทำให้เกิดการเอียงตัวไปทางขวาได้ส่วนเอียงตัวซ้ายก็ใช้วิธีคล้ายกัน

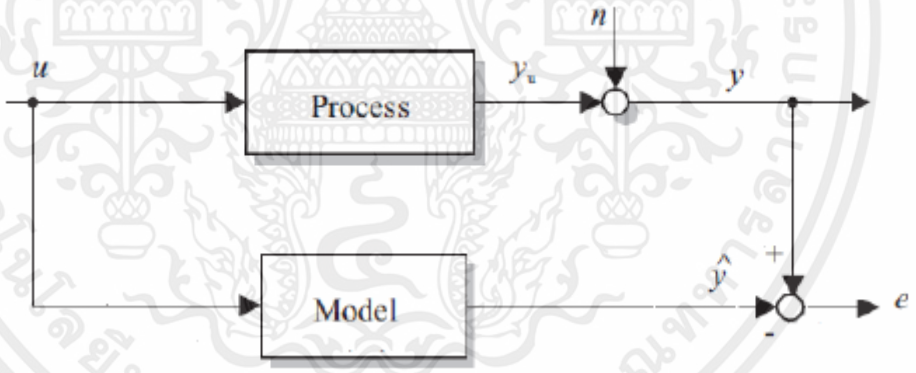
4. Pitch เอียงหน้าและหลัง อันนี้คล้ายๆกับการ Roll แต่เปลี่ยนเป็น ใบพัดซ้าย(LEFT) ขวา (RIGHT) จะความเร็วคงที่ แต่ความเร็วใบพัดหลัง(REAR) จะหมุนเร็วขึ้น ทางหลังจะยก ใบพัดหน้า (FRONT)จะหมุนช้ากว่าทางหน้าจะตกจึงทำให้เครื่องบินเอียงไปข้างหน้า

5. Yaw หรือการหมุนตัว ให้ความเร็วใบพัด หน้า(FRONT)-หลัง(REAR) มากกว่า ความเร็วใบพัด ซ้าย(LEFT) -ขวา(RIGHT) เพื่อให้แรงบิด ด้านซ้าย หรือ ขวา มากกว่า จึงทำให้เครื่องบินหมุนตัวได้

2.8 System Identification

System identification เป็นวิธีการหนึ่งที่ใช้สำหรับหาแบบจำลองทางคณิตศาสตร์ของระบบ Plant ซึ่งเป็นวิธีง่ายกว่าใช้กฎทางฟิสิกส์ทั่วไปเช่น Newton law หรือ Lagrange method เนื่องจากการใช้วิธี newton law เราจะต้องมีความรู้ทางด้าน Mechanic พอสมควรเราจะต้องสามารถเขียน FBD เป็นแล้วจึงสามารถหา plantได้ ส่วนวิธี Lagrange นั้นเราต้องมีความรู้เกี่ยวกับพลังงานในระดับหนึ่งเพราะวิธีการนี้จะมองระบบอยู่ในรูปของพลังงาน ซึ่งมี ทั้งพลังงานงานจลน์ และพลังงานศักย์ ยิ่งถ้าระบบที่มีความซับซ้อนมาก ๆ แล้วทั้งสองวิธีที่กล่าวมาข้างต้นนั้นอาจจะไม่สามารถใช้งานได้เลย บทความนี้จะพูดถึงวิธีการหา และวิธีการก็คือ

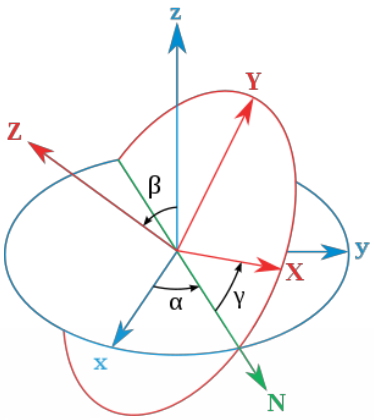
ต้องการข้อมูลของ Input และ output ตัวอย่างเช่นเราต้องการหาแบบจำลองทางคณิตศาสตร์ของมอเตอร์ inputของระบบก็คือแรงดันไฟฟ้าที่ป้อนให้มอเตอร์ ส่วนสัญญาณ output ก็อาจจะเป็นมุม หรือความเร็วเชิงมุมก็ได้ ถ้ามองที่ความเร็วเชิงมุมเราจะได้แบบจำลองเป็นสมการเชิงอนุพันธ์อันดับหนึ่ง ถ้ามอง output เป็นมุม เราก็จะได้แบบจำลองที่มีสมการเชิงอนุพันธ์อันดับสอง จากนั้นก็นำข้อมูล import เข้ามาที่โปรแกรม MATLAB แล้วใช้คำสั่ง ident แล้วก็สามารถเลือกได้ว่าต้องการให้แบบจำลองเราเป็นสมการเชิงอนุพันธ์อันดับเท่าใดก็ได้ เมื่อเราได้สมการเชิงอนุพันธ์แล้วเราก็ Laplace Transform เราก็จะได้ Transfer function เราก็สามารถนำไปออกแบบหาเกนต่อไปได้



รูปที่ 2.9 System Identification

2.9 การวัดการเคลื่อนไหวของอากาศยาน (AHRS)

การแสดงมุมของออยเลอร์ใช้การหมุนแกนเดี่ยวสามครั้งเพื่อแสดงภาพ 3 มิติที่สมบูรณ์การหมุน การหมุนทั้งสามนี้สามารถแบ่งออกเป็น 12 รูปแบบที่แตกต่างกัน 6 เรียกว่ามุม TaitBryan 6 เรียกว่ามุมออยเลอร์ที่เหมาะสม กระดาษนี้จะพิจารณาเฉพาะ Z-X'-Z "Tait-Bryan การประชุม เนื่องจากสิ่งนี้แสดงถึงการหมุนรอบแกนหันเหการหมุนรอบแกน pitch ใหม่



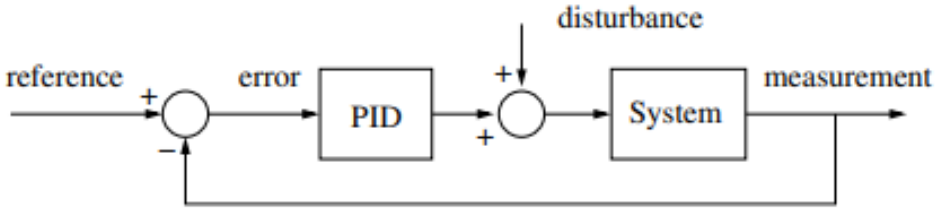
รูปที่ 2.10 การวัดการเคลื่อนไหวของอากาศยาน(AHRS)

วิธีการแสดงการวางแนวนี้มีข้อเสียจำนวนมาก

ประการแรกเป็นเพียงโอกาสที่จะเกิดความสับสนจำนวนมากเนื่องจาก 12 ข้อแตกต่างกันลำดับการหมุนที่เป็นไปได้ ตัวอย่างเช่นเครื่องบินที่พุ่งไป 90 °แล้วกลิ้งไป 90 °จะเข้ามาตำแหน่งที่แตกต่างกันมากกับเครื่องบินที่หมุน 90 °แล้วขว้าง 90 °

ประการที่สองมีความเป็นเอกฐานหลายประการในการแทนมุมของออยเลอร์ ในกรณีของการสั่งซื้อระบบที่ใช้ข้างต้นสิ่งเหล่านี้เกิดขึ้นที่การหันเห±2**π** ระยะห่าง ±0.5**π** และม้วน ±2**π** อย่างกะทันหันเหล่านี้การเปลี่ยนแปลงสามารถสร้างความเสียหายให้กับอัลกอริทึม Sensor Fushion ได้ซึ่งต้องใช้อย่างกว้างขวางและยุ่งเหยิงข้อความที่มีเงื่อนไขเพื่อพยายามแก้ไขซึ่งยากต่อการนำไปใช้เพิ่มความซับซ้อนของตัวกรอง ในที่สุดมุมออยเลอร์ของคำสั่งใด ๆ ต้องทนทุกข์ทรมานจากแนวคิดที่เรียกว่า gimbal lock สิ่งนี้เกิดขึ้นเมื่อไฟล์แกนของสองในสามของ gimbals ในระบบจัดแนวขนานกันนี้ได้อย่างมีประสิทธิภาพ ป้องกันไม่ให้ระบบเคลื่อนที่ในแกนภายนอกหนึ่งแกนทำให้ระบบทำหน้าที่เหมือน gimbals สองมิติ โดยจินตนาการว่าเครื่องบินหมุนผ่านแกนYawโดย 0 rad, แกนPitch 0.5**π** rad และแกนRoll ด้วย 0 rad แกนYaw และแกนRoll ตอนนี้ทั้งสองอยู่ใน ระนาบหมายถึงเครื่องบินไม่สามารถหมุนในเครื่องบินได้อีกต่อไป สิ่งนี้อาจทำให้เกิดการเคลื่อนที่แบบไม่มีทิศทางได้

2.10 PID Controller



เป็นกระบวนการควบคุมอย่างหนึ่งที่นิยมนำมาใช้ในอุตสาหกรรม ในระบบควบคุมมีตัวควบคุมหลายชนิด ตัวควบคุมส่วนใหญ่ที่ใช้ในการควบคุมกระบวนการเป็นแบบ PID โดยต่ออนุกรมกับระบบที่ต้องการควบคุม ดังแสดงในรูปที่ 2.14 สัญญาณออกจากตัวควบคุม PID สามารถบรรยายได้ดังนี้

$$u(t) = K_p(e(t) + \frac{1}{T_i}\int_0^t e(t)dT + T_d\frac{de(t)}{dt}) \qquad (2.1)$$

โดย *u(t)* คือสัญญาณควบคุม *e(t)* คือค่าความคลาดเคลื่อนของสัญญาณออกจากค่ากำหนด ตัวควบคุม PID ประกอบไปด้วยเทคนิคการควบคุมพื้นฐาน 3 แบบ 1. แบบสัดส่วน (Proportional หรือ P) 2. แบบอินทิกรัล (Integral หรือ I) และ3.แบบอนุพันธ์(Derivative หรือ D) แต่ละแบบสามารถนำมาประกอบกันเพื่อให้ได้ตัวควบคุมที่ต้องการตัวควบคุมมีพารามิเตอร์ 3 ตัว คือ ค่าอัตราขยายแบบสัดส่วน (K_p) ค่า integral time (T_i) และderivative time (T_d) ซึ่งรายละเอียดของแต่ละแบบมีดังนี้

2.10.1 Proportional Action

การควบคุมแบบสัดส่วนเป็นเทคนิคที่ง่ายที่สุด หลักการคือสัญญาณควบคุม u(t) จากตัวควบคุมที่ส่งไปปรับกระบวนการมีค่าเป็นสัดส่วนกับความคลาดเคลื่อน ซึ่งสามารถเขียนได้ในรูป

$$u(t) = K_p e(t) \qquad (2.2)$$

โดยที่ K_p คือค่าอัตราขยาย

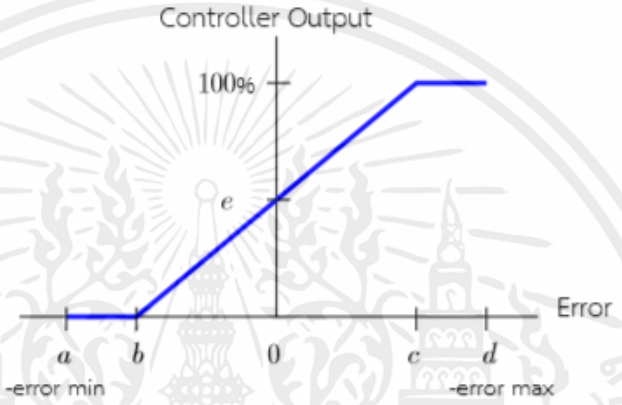
$$e(t) = \text{ความคลาดเคลื่อน} = \text{ค่ากำหนด} - \text{ค่าวัด} \qquad (2.3)$$

ตัวควบคุมบางตัวสัญญาณเข้าและสัญญาณออกอาจมีหน่วยต่างกัน เช่นการเปลี่ยนแปลงของอุณหภูมิที่ทำให้เกิดการเปลี่ยนแปลงความดัน เพื่อหลีกเลี่ยงการแปลงหน่วย ความสัมพันธ์ระหว่างสัญญาณออกและสัญญาณเข้าของตัวควบคุมอาจแสดงเป็นแถบสัดส่วน (Proportional Band หรือ %PB) โดยที่แถบสัดส่วนคือพิสัยของสัญญาณเข้าที่ทำให้ตัวควบคุมปฏิบัติงานเต็มพิสัยการทำงาน

หรือถ้ามองจากตัวควบคุม แถบสัดส่วนคือช่วงความคลาดเคลื่อนที่ทำให้สัญญาณออกของตัวควบคุมเปลี่ยนแปลงจากค่าสูงสุดไปต่ำสุด โดยแสดงเป็นเปอร์เซ็นต์ของพิสัยสัญญาณเข้าตัวควบคุม ความสัมพันธ์ระหว่างอัตราขยายและเปอร์เซ็นต์ แถบสัดส่วนคือ

$$K_P = \frac{100}{\%\, PB} \tag{2.4}$$

ลักษณะสมบัติของการควบคุมแบบสัดส่วนแสดงไว้ในรูปที่ 2.12



รูปที่ 2.11 ลักษณะสมบัติของตัวควบคุมแบบสัดส่วน

ค่าเปอร์เซ็นต์แถบสัดส่วน (%PB) คือระยะ bc แม้ความคลาดเคลื่อนเป็นศูนย์ ยังมีสัญญาณค่าหนึ่ง ออก จากตัวควบคุมที่ป้อนให้กับกระบวนการค่านี้ทำหน้าที่เป็นไบแอส (bias) ทำให้ระบบทำงานที่จุดทำงานต่อไปได้โดยทั่วไป สัญญาณค่านี้มักจะถูกตั้งให้เท่ากับ 50% ของสัญญาณขาออกสูงสุดของตัวควบคุมนั่นคือ

$$\text{สัญญาณออก} = \frac{\%\text{ความคลาดเคลื่อน}}{\%\text{แถบสัดส่วน}} + 50\% \tag{2.5}$$

นอกจากนี้ตัวควบคุมมีย่านทำงานที่เป็นเชิงเส้นช่วงหนึ่ง โดยทำหน้าที่เป็นตัวขยาย (amplifier) แต่ถ้าความคลาดเคลื่อนมีมากเกินระดับหนึ่ง ตัวขยายจะอิ่มตัวทำให้สัญญาณออกมีค่าคงที่ การควบคุมแบบสัดส่วนนี้สามารถควบคุมระบบได้ดีพอสมควร เหมาะสมกับกระบวนการที่ต้องการผลตอบสนองรวดเร็วและยอมให้เกิดความคลาดเคลื่อนขนาดคงที่ขนาดหนึ่ง อย่างไรก็ตาม หากในกระบวนการเกิดมีการเปลี่ยนแปลงพารามิเตอร์ อาจทำให้เกิดปัญหา เช่น มีค่าความ

คลาดเคลื่อนในสภาวะอยู่ตัว (steady-state error) หรือที่เรียกว่า ออฟเซต (offset) ตัวควบคุมแบบสัดส่วนไม่สามารถแก้ไขให้หมดได้

แนวทางการแก้ไขปัญหาที่เกิดขึ้นทำได้ 2 วิธีคือ วิธีแรก คือ เพิ่มอัตราขยาย (gain) ของตัวควบคุมเพื่อเพิ่มผลของความคลาดเคลื่อนที่มีต่อระบบ ถึงแม้ความคลาดเคลื่อนที่เกิดขึ้นจะมีค่าน้อยลง แต่ก็จะทำให้สัญญาณออกจากตัวควบคุมที่เหมาะสมกับกระบวนการขณะนั้นได้ อย่างไรก็ตามการเพิ่มผลของความคลาดเคลื่อนมากเกินไปก็อาจทำให้ระบบแกว่งได้เนื่องจากระบบมีความไว วิธีที่สอง คือ ปรับค่าไบแอสของตัวควบคุมใหม่ด้วยมือ ซึ่งทำให้ตัวควบคุมเลื่อนจุดทำงานไปยังจุดที่ให้สัญญาณออกที่เหมาะสมกับกระบวนการในขณะนั้นได้ปัญหาของวิธีหลังอยู่ตรงที่ต้องปรับค่าไบแอสของตัวควบคุมทุกครั้งที่มีการเปลี่ยนแปลง พารามิเตอร์ของกระบวนการ

2.10.2 Integral Action

ผลตอบของการควบคุมแบบสัดส่วนรวมกับการควบคุมแบบอินทิกรัล สามารถอธิบายได้ในสมการ

$$u(t) = K_p(e(t) + \frac{1}{T_i}\int_0^t e(T)dT) \qquad (2.6)$$

เมื่อ K_p คืออัตราขยาย *e(t)* คือ ความคลาดเคลื่อน และ T_i คือ integral time (วินาที)

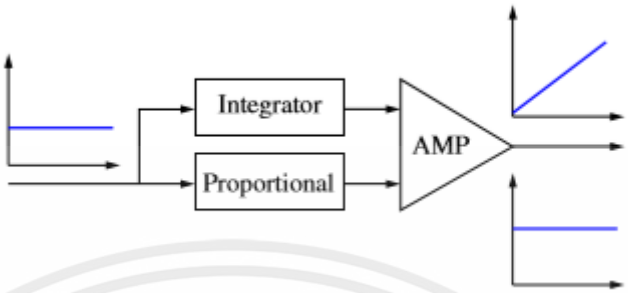
เมื่อเปรียบเทียบกับสมการของตัวควบคุมแบบสัดส่วน ความแตกต่างอยู่ตรงที่เทอมไบแอส นั่นคือตัวควบคุมแบบสัดส่วนถูกจำกัดด้วยส่วนไบแอสเป็นค่าคงที่ ส่วนการควบคุมแบบอินทิกรัลมีการสะสมความคลาดเคลื่อนในการปรับแต่งไบแอส (นั่นคือ ทำหน้าที่เป็นตัวอินทิกรัล) และจะหยุดสะสมเมื่อความคลาดเคลื่อนของระบบเป็นศูนย์ เมื่อผลตอบเข้าที่สมบูรณ์แล้วเทอมไบแอสของระบบจะมีค่ามากน้อยเพียงใดขึ้นอยู่กับลักษณะของการรบกวน(disturbance) การทำงานในลักษณะเช่นนี้ มีลักษณะคล้ายกับฟังก์ชันรีเซตด้วยมือ (manual-reset function) ดังนั้นในบางครั้งจึงเรียกตัวอินทิกรัลว่าฟังก์ชันรีเซต (reset function)

คุณสมบัติของตัวอินทิกรัลในการกำจัดความคลาดเคลื่อน (หรือออฟเซต) เป็นข้อดีอย่างมาก จึงเป็นที่นิยมใช้กับระบบ ควบคุมป้อนกลับ อย่างไรก็ตาม ตัวอินทิกรัลก็มีข้อเสีย นั่นคือทำให้เกิดการล้าหลัง (capacity-like lag) และทำให้ช่วงเวลา ของการแกว่งยาวนานขึ้น โดยทั่วไป ระบบแบบสัดส่วนรวมกับอินทิกรัลจะมีช่วงเวลาของการแกว่งนานกว่าระบบเชิงสัดส่วนอย่างเดียว 50% หรือ T_{pi} = 1.5 T_p สำหรับระบบที่มีค่าคงตัวเวลา (time constant) น้อย (เช่น ระบบควบคุมอัตราการไหล) ปัญหานี้จะไม่มีผลมากนัก แต่สำหรับระบบที่มีค่าคงตัวตัวเวลามาก (เช่น ระบบควบคุมระดับ) ปัญหานี้อาจมีผลมากจนทำให้ระบบเข้าสู่จุดวิกฤติที่ไม่สามารถยอมรับได้

ข้อสรุปของตัวควบคุมอินทิกรัล

- ทำหน้าที่คล้ายรีเซตด้วยมือ (manual reset) เพื่อกำจัดความคลาดเคลื่อน

- มีปัญหาการล้าหลัง ยังทำให้ผลเกิดการหักล้างทางเวลาในตัวควบคุมจึงไม่เหมาะกับระบบที่มีค่าคงตัวเวลายาวนาน
- ทำให้ช่วงเวลาในการแกว่งยาวนานขึ้น



รูปที่ 2.12 แผนภาพกรอบแสดงลักษณะตัวควบคุมแบบ PI

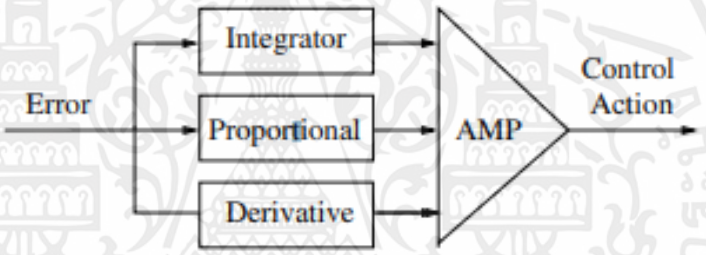
ในระบบควบคุม ค่าที่วัดได้และค่ากำหนดควรเป็นค่าเดียวกันหรือกล่าวอีกนัยหนึ่ง ค่าความคลาดเคลื่อนในสภาวะอยู่ตัวควรเป็นศูนย์ ถ้ามีความคลาดเคลื่อนในสภาวะอยู่ตัว สัญญาณที่ออกจากอินทิเกรเตอร์ (เพิ่มขึ้นด้วยอัตราคงที่ เมื่อสัญญาณเข้ามีค่าคงที่) ส่งต่อให้กับวงจรขยาย ดังแสดงในรูปที่ 2.14 สังเกตว่า ความคลาดเคลื่อนเป็นสัญญาณเข้าของตัวควบคุมทั้งสัดส่วน และอินทิกรัลโดยสัญญาณออกจะมารวมกันที่วงจรขยายและส่งสัญญาณไปควบคุมระบบตัวควบคุมจะทำให้ค่าที่วัดได้เพิ่มขึ้นจนเท่ากับค่ากำหนด นั่นคือทำให้ความคลาดเคลื่อนในสภาวะอยู่ตัวเป็นศูนย์ อย่างไรก็ตามหาก T_i มีค่าน้อย ผลตอบอาจเกิดการแกว่งได้

2.10.3 Derivative Action

ตัวควบคุมแบบสัดส่วนและแบบปริพันธ์ต่างก็มีข้อจำกัดอยู่ที่ความคลาดเคลื่อนขนาดใหญ่ซึ่งเป็นปัญหาต่อการควบคุมกระบวนการแต่ความคลาดเคลื่อนขนาดใหญ่นี้สามารถรู้ได้ล่วงหน้าโดยพิจารณาจากแนวโน้มของความคลาดเคลื่อนหรืออัตราการเปลี่ยนแปลงของสัญญาณนั่นเอง ตัวอนุพันธ์มีหลักการทำงาน คือ ตัวควบคุมตอบสนองต่ออัตราการเปลี่ยนแปลงของความคลาดเคลื่อน ถึงแม้ว่าความคลาดเคลื่อนมียังค่าเล็กอยู่ สัญญาณออกของตัวอนุพันธ์ไม่ได้สัมพันธ์กับขนาดของความคลาดเคลื่อนแต่ขึ้นอยู่กับการเปลี่ยนแปลงของความคลาดเคลื่อนถ้าความคลาดเคลื่อนมีค่าคงที่ตัวอนุพันธ์จะให้สัญญาณออกเป็นศูนย์คุณลักษณะข้อนี้มีผลดีคือ ตัวควบคุมจะมีผลตอบสนองที่เกิดก่อนที่ความคลาดเคลื่อนจะเพิ่มมากขึ้น และทำให้ระบบมีผลตอบสนองที่ เร็วขึ้นตัวควบคุมแบบอนุพันธ์สามารถเขียนได้ดังนี้

$$u(t) = K_p(e(t) + T_d \frac{de(t)}{dt}) \quad (2.7)$$

โดย derivative time (T_d) เป็นเวลาที่แสดงถึงผลตอบสนองเนื่องจากตัวอนุพันธ์ การเพิ่ม T_d จะทำให้ผลตอบสนองของตัวอนุพันธ์มีค่ามากขึ้น เนื่องจากตัวอนุพันธ์มีความไวต่อการเปลี่ยนแปลงมาก ดังนั้นจึงนิยมใช้กับค่าที่วัดได้เท่านั้น แต่ไม่ใช้กับค่ากำหนด เพราะการเปลี่ยนค่ากำหนดมักจะเป็นแบบขั้น (step) ทำให้ผลตอบสนองของตัวอนุพันธ์เป็นพัลส์ และทำให้เกิดการกระแทก (bump) ของอุปกรณ์ในกระบวนการสำหรับค่ากำหนดใช้เฉพาะกับตัวควบคุมสัดส่วนและอินทิกรัล ตัวอนุพันธ์คือตัวควบคุมที่ก่อให้เกิดผลตรงข้ามกับตัวอินทิกรัล ดังนั้นจึงใช้ในการปรับปรุงกระบวนการที่มีการล้าหลังทางเวลา (Timelag) มากๆ ทำให้ผลตอบสนองรวดเร็วขึ้น และช่วงเวลาการแกว่งที่สั้นลง ข้อเสียของตัวอนุพันธ์ คือ มีความไว ต่อสัญญาณรบกวนเป็นอย่างมาก เพราะมีผลตอบสนองโดยตรงต่ออัตราการเปลี่ยนแปลงของสัญญาณที่วัดได้ ดังนั้นแม้สัญญาณรบกวนจะมีขนาดเล็กแต่ก็อาจก่อให้เกิดการเปลี่ยนแปลงต่อสัญญาณออกของตัวควบคุม จึงเป็นไปไม่ได้ที่จะใช้ตัวอนุพันธ์ในการควบคุมผลของสัญญาณรบกวน ยิ่งไปกว่านั้นระบบใดที่มีสัญญาณรบกวนมาก จะไม่สามารถใช้ตัวอนุพันธ์ ในวงการอุตสาหกรรมส่วนใหญ่นิยมใช้เพียงตัวควบคุม PI เท่านั้น



รูปที่ 2.13 แผนภาพกรอบแสดงลักษณะตัวควบคุมแบบ PID

บทสรุปของการตัวอนุพันธ์

• เหมาะสำหรับกระบวนการที่ล้าหลังทางเวลามาก ทำให้การควบคุมถึงจุดที่ต้องการเร็วขึ้น

• ถ้า T_d มากเกินไป ผลของตัวอนุพันธ์จะทำให้ผลตอบสนองไวขึ้น จนกระทั่งระบบอาจขาดเสถียรภาพได้

• ไม่เหมาะกับระบบที่มีตัวแปรกระบวนการเปลี่ยนแปลงได้ง่าย หรือมีการล้าหลังทางเวลาน้อย เพราะจะทำให้ระบบขาดเสถียรภาพ (เช่นระบบควบคุมอัตราการไหล)

• ไม่ควรใช้กับระบบที่มีสัญญาณรบกวนมาก

• ใช้ชดเชยการล้าหลังที่เกิดจากตัวปริพันธ์ด้วยการนำหน้า (lead) ในตัวอนุพันธ์

จะได้ผลรวมเทอมสัดส่วน ปริพันธ์ และอนุพันธ์ จะนำมารวมกันเป็นสัญญาณขาออกของการควบคุมแบบ PID กำหนดให้ เป็นสัญญาณขาออก สมการสุดท้ายของวิธี PID คือ

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(T)dT + K_d \frac{d}{dt} e(t) \qquad (2.8)$$

เมื่อมีการเปลี่ยนแปลงค่าที่กำหนดทันที ความคลาดเคลื่อนจะมีค่าเปลี่ยนแปลงอย่างทันที และส่งผลต่อผลตอบสนองของระบบ ถ้านำอนุพันธ์ของความคลาดเคลื่อน นั่นคือ อัตราการเปลี่ยนแปลงของความคลาดเคลื่อน แล้วไปรวมกับสัญญาณที่ได้จากตัวควบคุมแบบสัดส่วนและปริพันธ์ ดังแสดงในรูป 5 จะทำให้การทำงานของระบบดีขึ้น การควบคุมเชิงอนุพันธ์ไม่มีผลต่อความคลาดเคลื่อนในสภาวะอยู่ตัว แต่จะลดช่วงเวลาเข้าที่ (settling time) โดยลดการแกว่งลง

จากรายละเอียดที่กล่าวมาจะพบว่าตัวควบคุม PID ยังคงมีจุดอ่อนบ้าง ดังนั้นในการใช้งานจริงจึงมีการต่อวงจรเพิ่มเติม สำหรับแก้จุดอ่อน ได้แก่

- วงจรสำหรับป้องกัน integral windup ที่เกิดจากตัวควบคุมแบบอินทิกรัล
- วงจรกรอง (filter) สำหรับลดผลเนื่องจากสัญญาณรบกวนที่มีกับตัวควบคุมแบบอนุพันธ์
- ปรับโครงสร้างให้ตัวควบคุมเชิงอนุพันธ์รับสัญญาณออกของระบบเท่านั้น เพื่อป้องกันการเปลี่ยนแปลงที่มีค่าเกินกว่าที่รับได้ (derivative overrun)

นอกจากปัญหาที่เกิดจากการควบคุมทั้ง 3 แบบแล้ว ยังมีปัญหาที่เกิดจากฟังก์ชันการทำงาน คือตัวควบคุมส่วนมากจะมีโหมดการทำงาน 2 โหมด คือ การควบคุมด้วยมือ (manual) และการควบคุมอัตโนมัติ (automatic) ในโหมดการควบคุมด้วยมือ สัญญาณที่ส่งออกจากตัวควบคุมจะขึ้นกับการปรับโดยตรงของผู้ใช้หากมีการเปลี่ยนโหมดการทำงานกลับมาที่โหมดการควบคุมอัตโนมัติ ตัวควบคุมทำหน้าที่ส่งสัญญาณออกจากตัวควบคุมอาจเกิดปัญหาการกระแทก (bump) ขึ้นได้ เนื่องจากการเปลี่ยนแปลงสัญญาณควบคุมที่ออกจากตัวควบคุมอย่างเฉียบพลัน ดังนั้นในตัวควบคุม PID ส่วนมากจึงต้องมีวงจรลดการกระแทก (Bumpless transfer) สำหรับแก้ปัญหานี้ไว้ด้วย

บทที่ 3
หลักการออกแบบ และการสร้าง

3.1 กล่าวนำ

ในส่วนของการดำเนินงานทางคณะผู้จัดทำได้แบ่งออกเป็น 4 ส่วนหลัก คือ การเลือกวัสดุและอุปกรณ์ของอากาศยานไร้คนขับ, การประกอบชิ้นส่วนต่าง ๆ เข้าด้วยกัน, ,การติดต่อสื่อสาร และการส่งข้อมูลระหว่างอุปกรณ์ และการออกแบบสมการ System ID

3.2 การเลือกวัสดุ และอุปกรณ์

การเลือกใช้วัสดุที่เป็นโครงสร้างของอากาศยานไร้คนขับ จะเลือกใช้แบบ 4 ใบพัด ซึ่งเป็นวัสดุเป็นพลาสติก เพราะมีน้ำหนักเบา และสามารถดัดแปลงได้ง่าย

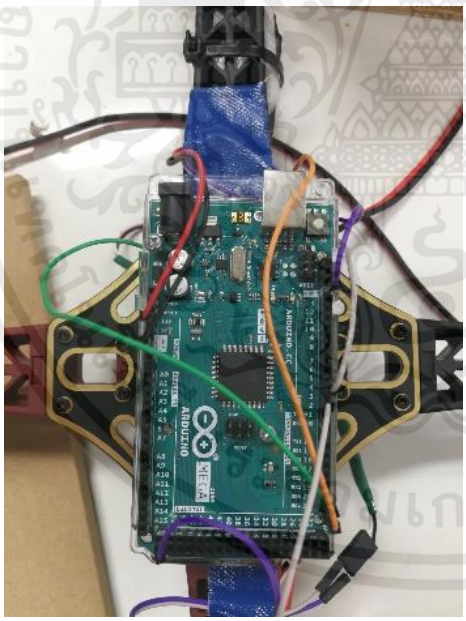
รูปที่ 3.1 โครงสร้างของอากาศยานไร้คนขับแบบ 4 ใบพัด

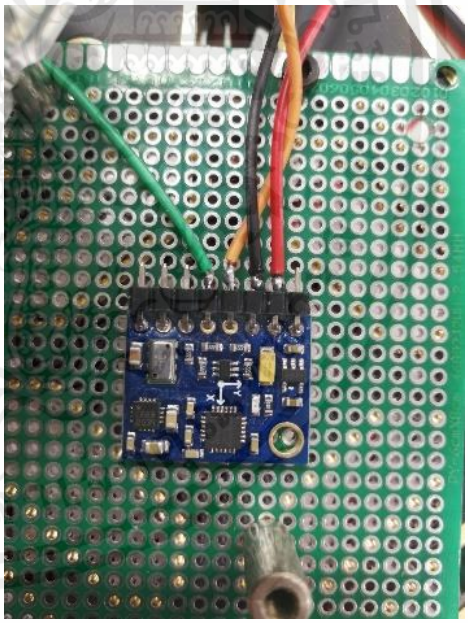
3.3 การประกอบชิ้นส่วน

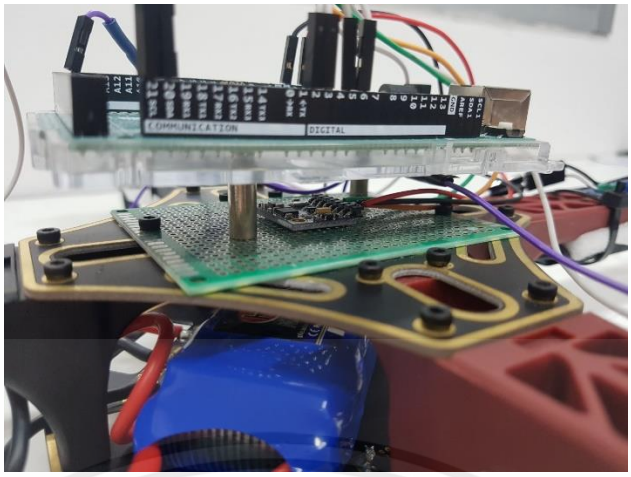
ขั้นตอนแรกเป็นการประกอบชิ้นส่วนโครงของอากาศยานไร้คนขับ แต่ละชิ้นส่วนประกอบเป็นโครงดังรูปที่ 3.1 แล้วทำการติดตั้ง Brushless Motor ทั้ง 4 ด้านของโครง

รูปที่ 3.2 การประกอบ Brushless Motor

ขั้นตอนถัดมา ติดตั้ง Arduino MEGA 2560 Board, Brushless Driver, Propeller, Battery (Lipo) และ GY-86 เข้าด้วยกันโดยทำการเชื่อมต่อแบตเตอรี่ (Lipo) เข้ากับบอร์ดจ่ายไฟ จากนั้นเชื่อมเข้าบอร์ด Arduino เพื่อจ่ายไฟไปยังมอเตอร์ จำนวน 4 ตัว







รูปที่ 3.3 การติดตั้งอุปกรณ์เข้าด้วยกัน

เมื่อทำการติดตั้งอุปกรณ์ต่าง ๆ เข้าด้วยกันแล้วนั้นจะได้อากาศยานไร้คนขับแบบ 4 ใบพัดที่สมบูรณ์ โดยมีมอเตอร์ขับใบพัด 4 ตัว โดยมี Brushless Driver เป็นตัวขับมอเตอร์ ซึ่งได้รับค่าจากไมโครคอนโทรลเลอร์ Arduino MEGA 2560 ซึ่งจะทำหน้าที่ควบคุมความเร็วของมอเตอร์ทั้ง 4 ตัว และมีค่าที่ทำการวัดโดย GY-86 เป็นเซ็นเซอร์ตรวจจับค่าของแกนเคลื่อนที่ โดยมีแหล่งจ่ายไฟจากแบตเตอรี่ Lipo จ่ายไฟให้แก่อากาศยานไร้คนขับ

รูปที่ 3.4 อากาศยานไร้คนขับที่ประกอบเสร็จสมบูรณ์

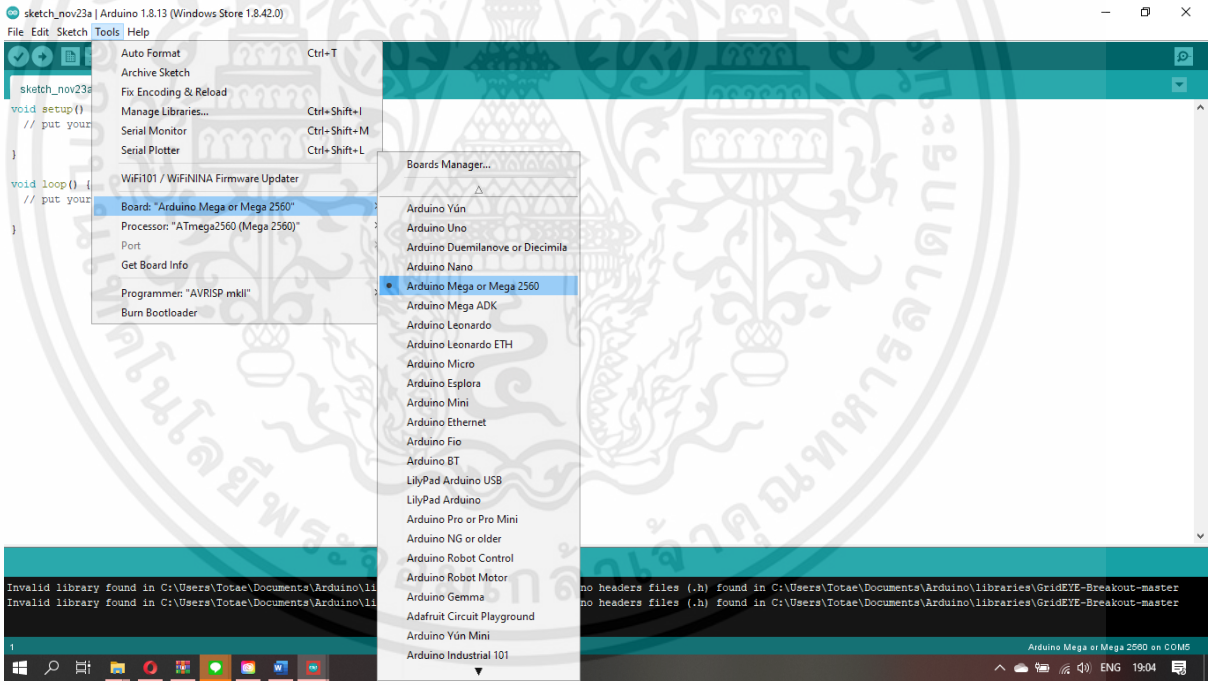
3.4 การส่งข้อมูลระหว่างอุปกรณ์

3.4.1 การโปรแกรมข้อมูลบนบอร์ด Arduino MEGA 2560

1. การควบคุมการเคลื่อนที่ของอากาศยานไร้คนขับด้วยบอร์ด Arduino MEGA 2560 จะต้องทำการเชื่อมต่อกับ Computer ผ่านสาย USB Port

รูปที่ 3.5 สาย USB Port เชื่อมต่อกับ Computer กับบอร์ด Arduino MEGA 2560

2. เปิดโปรแกรม Arduino IDE ซึ่งโปรแกรมนี้จะใช้กับบอร์ด Arduino หลายรุ่น ดังนั้นจะต้องกำหนดบอร์ดที่ใช้ในประกอบนี้ โดยเข้าที่ Tool >>Board >> Arduino MEGA 2560

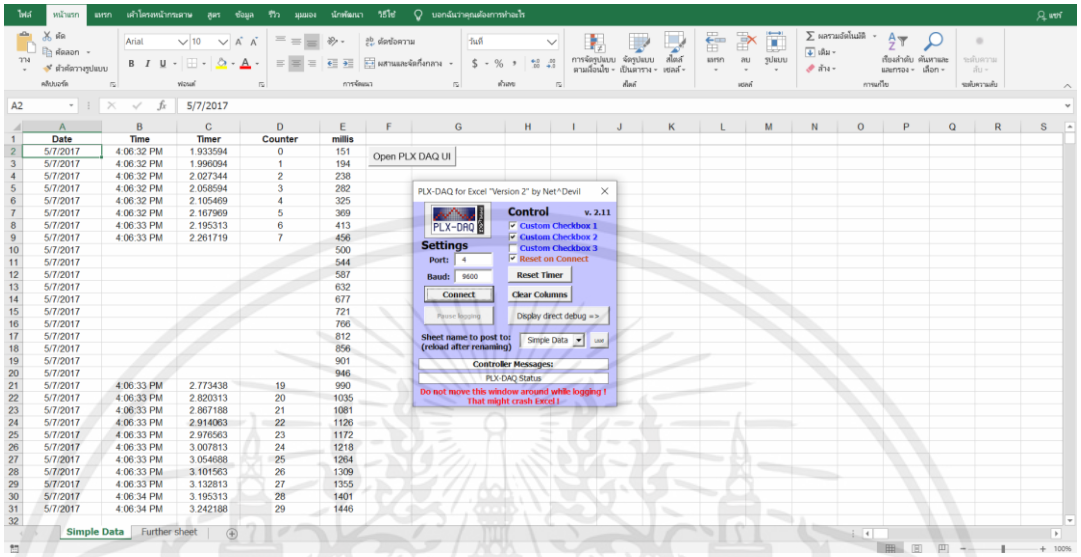


รูปที่ 3.6 การเลือกบอร์ดในโปรแกรม Arduino IDE

3.การตั้งค่าเริ่มต้นของโปรแกรม เป็นการติดต่อสื่อสารระหว่างบอร์ด Arduino MEGA 2560 กับ อุปกรณ์ต่าง ๆ โดยการดาวน์โหลด Library เพื่อทำการเขียนคำสั่งควบคุมการหมุนของมอเตอร์

3.4.2 เขียนโปรแกรมการทำงานของเซ็นเซอร์ GY-86 และ แสดงข้อมูลขึ้น EXCEL

จากนั้นเข้าโปรแกรม PLX-DAQ แล้ว Setup ว่า ต่อสายข้อมูลที่คอมอะไร Baud rate ที่ใช้มีค่าเท่าไร จากนั้นก็กด Connect เพื่อให้ run ข้อมูลออกมา



รูปที่ 3.7 เป็นการแสดงการ Setup ของโปรแกรม PLX-DAQ

3.5 อัลกอริทึมที่ใช้ในการคำนวณ

3.5.1 A Super Fast Attitude Determination Algorithm for Consumer-Level Accelerometer and Magnetometer

อัลกอริทึมนี้เป็นการใช้ค่าที่วัดได้จากเซนเซอร์ ได้แก่ Accelerometer magnetometer ของทั้งแกน X-Y-Z นำมาคำนวณเป็นผลของ Quaternion ซึ่งผลลัพธ์นี้จะนำไปคำนวณหา Roll Pitch & Yaw ได้

โดยสมการมีดังนี้

$$a_x^2 + a_y^2 + a_z^2 = 1 \qquad (3.1)$$

$$m_x^2 + m_y^2 + m_z^2 = 1 \qquad (3.2)$$

$$m_N^2 + m_D^2 = 1 \qquad (3.3)$$

นำสมการ (3.1) (3.2)และ(3.3) มารวมกันจะได้ว่า

$$m_D = a_x m_x + a_y m_y + a_z m_z \quad (3.4)$$

$$m_N = \sqrt{1 - m_D^2} \quad (3.5)$$

นำค่าทั้งหมดที่ได้มาคำนวณเพื่อแปลงเป็น Quaternion จะได้ว่า

$$\widetilde{q_0} = (a_z - 1)(m_N + m_x) + a_x(m_D - m_z) \quad (3.6)$$

$$\widetilde{q_1} = (a_z - 1)(m_y) + a_y(m_D - m_z) \quad (3.7)$$

$$\widetilde{q_2} = a_z m_D - a_x m_N - m_z \quad (3.8)$$

$$\widetilde{q_3} = -a_y(m_N + m_x) + a_x m_y \quad (3.9)$$

แปลงค่าทั้งหมดให้อยู่ในรูปของ unit vector จะได้ว่า

$$q = \frac{(\widetilde{q_3}, \widetilde{q_0}, \widetilde{q_1}, \widetilde{q_2})^T}{\sqrt{\widetilde{q_0}^2 + \widetilde{q_1}^2 + \widetilde{q_2}^2 + \widetilde{q_3}^2}} \quad (3.10)$$

นำค่าทั้งหมดของ Quaternions แปลงให้อยู่ในรูปของ Euler angle จะได้ว่า

$$yaw = atan2(2(q_0 q_3 + q_1 q_2), 1 - 2({q_2}^2 + {q_3}^2)) \quad (3.11)$$

$$pitch = atan2(2(q_0 q_1 + q_3 q_2), 1 - 2({q_2}^2 + {q_1}^2)) \quad (3.12)$$

$$yaw = \text{asin}\,(2(q_0 q_2 - q_3 q_1)) \quad (3.13)$$

3.5.2 LOW PASS-FILTER

LOW PASS-FILTER เป็นวิธีการกรองความถี่ต่ำ โดยกำจัดความถี่สูงออก หรือก็คือ Noise ซึ่งมันมีผลต่อการหาค่าความสูงจาก Barometric Pressure โดยเราจะจำลองในการหาสมการ LOW PASS FILTER ในวงจรที่มีตัวเก็บประจุในการกรองความถี่ต่ำ โดยเราเริ่มต้นโดยการกำหนดความถี่ใน

กรองถี่ต่ำ มีค่าเท่ากับ 100 Hz เราจะนำค่าไปสร้างเป็นสมการ transfer function โดยสมการรูปทั่วไปของตัวเก็บประจุอยู่ในรูปแบบคือ

$$sys = \frac{w_c}{s+w_c} \tag{3.14}$$

จากสมการที่ 3.14 เมื่อนำค่า ความถี่ที่เรากำหนดเขาไปใส่คือ 100 Hz จะได้ว่า

$$sys = \frac{2*\pi*100}{s+2*\pi*100} \tag{3.15}$$

แปลงสมการ Transfer Function ให้อยู่ในรูปของ Discrete transfer function โดยเราจะใช้ฟังก์ชั่นใน MATLAB คือคำสั่ง c2d โดย Sampling ทุกๆ 0.0001 s จะได้ว่า

$$\mathrm{sys} = \frac{0.0609}{Z-0.9391} \tag{3.16}$$

โดย เราจะนำไปค่าของ Discrete transfer function ไปใส่ในอยู่ในรูป transfer function จะได้ว่า

$$Y_{new} = 0.9391 * Y_{old} + 0.0609 * X \tag{3.17}$$

สามารถนำตัวเลขไปเขียนสมการใน Arduino จะได้ว่า

```
Pressure_Filter = 0.9391*Pressure_Filter + 0.0609*(realPressure);
```

เราจะสามารถได้ค่า Pressure ที่ผ่านการ Filter ทำให้ได้ค่าแม่นยำที่มากขึ้น เนื่องกำจัดความถี่สูงออกหมด

3.6 การระบุเอกลักษณ์ของระบบ

สมการเอกลักษณ์ของระบบอากาศยานไร้คนขับ (UAV) มีสมการดังนี้

$$\dot{\emptyset} = p + r[c(\emptyset)t(\theta)] + q[s(\emptyset)t(\theta)] \tag{3.18}$$

$$\dot{\theta} = q[c(\emptyset)t(\theta) - r[s(\emptyset)] \tag{3.19}$$

$$\dot{\varphi} = r\frac{c(\emptyset)}{c(\theta)} + q\frac{s(\emptyset)}{c(\theta)} \tag{3.20}$$

$$\dot{p} = \frac{I_y - I_z}{I_x} rq + \frac{\tau_x + \tau_{wx}}{I_x} \tag{3.21}$$

$$\dot{q} = \frac{I_z - I_x}{I_y} rq + \frac{\tau_y + \tau_{wy}}{I_y} \quad (3.22)$$

$$\dot{r} = \frac{I_x - I_y}{I_z} rq + \frac{\tau_z + \tau_{wz}}{I_z} \quad (3.23)$$

$$\dot{u} = rv - qw - g[s(\theta)] + \frac{f_{wx}}{m} \quad (3.24)$$

$$\dot{v} = pw - ru - g[s(\emptyset)c(\theta)] + \frac{f_{wy}}{m} \quad (3.25)$$

$$\dot{w} = qu - pv + g[c(\theta)c(\emptyset)] + \frac{f_{wz} - f_t}{m} \quad (3.26)$$

$$\dot{x} = w[s(\emptyset)s(\varphi) + c(\emptyset)c(\varphi)s(\theta)] - v[c(\emptyset)s(\varphi) - c(\varphi)s(\emptyset)s(\theta)] + u[c(\varphi)c(\theta)] \quad (3.27)$$

$$\dot{y} = v[c(\emptyset)c(\varphi) + s(\emptyset)s(\varphi)s(\theta)] - w[c(\varphi)s(\emptyset) - c(\emptyset)s(\varphi)s(\theta)] + u[c(\theta)s(\varphi)] \quad (3.28)$$

$$\dot{z} = w[c(\emptyset)c(\theta)] - u[s(\theta)] + v[c(\theta)s(\emptyset)] \quad (3.29)$$

เนื่องจากสมการข้างต้นเป็นสมการ Non-linear จึงต้องทำการทำเป็น Linearization ได้สมการดังนี้

$$\dot{\emptyset} = p \quad (3.30)$$

$$\dot{\theta} = q \quad (3.31)$$

$$\dot{\varphi} = r \quad (3.32)$$

$$\dot{p} = \frac{\tau_x + \tau_{wx}}{I_x} \quad (3.33)$$

$$\dot{q} = \frac{\tau_y + \tau_{wy}}{I_y} \quad (3.34)$$

$$\dot{r} = \frac{\tau_z + \tau_{wz}}{I_z} \quad (3.35)$$

$$\dot{u} = g\theta + \frac{f_{wx}}{m} \tag{3.36}$$

$$\dot{v} = g\emptyset + \frac{f_{wy}}{m} \tag{3.37}$$

$$\dot{w} = \frac{f_{wz} - f_t}{m} \tag{3.38}$$

$$\dot{x} = u \tag{3.39}$$

$$\dot{y} = v \tag{3.40}$$

$$\dot{z} = w \tag{3.41}$$

นำสมการทั้งหมดที่เป็น Linearization มาทำการหา Transfer Function จะได้ว่า

$$\frac{p(s)}{\tau_x(s)} = \frac{1}{I_x s} \tag{3.42}$$

$$\frac{q(s)}{\tau_y(s)} = \frac{1}{I_y s} \tag{3.43}$$

$$\frac{r(s)}{\tau_z(s)} = \frac{1}{I_z s} \tag{3.44}$$

$$\frac{z(s)}{f_t(s)} = \frac{-1}{ms^2} \tag{3.45}$$

$$\frac{\emptyset(s)}{p(s)} = \frac{1}{s} \tag{3.46}$$

$$\frac{\theta(s)}{q(s)} = \frac{1}{s} \tag{3.47}$$

$$\frac{\varphi(s)}{r(s)} = \frac{1}{s} \tag{3.48}$$

$$\frac{y(s)}{\emptyset(s)} = \frac{1}{gs^2} \tag{3.49}$$

$$\frac{x(s)}{\theta(s)} = \frac{-1}{gs^2} \tag{3.50}$$

โดยพารามิเตอร์ของระบบสมการโดรน คือ

$$I_x = 0.05\ kg * m^2\ \ I_y = 0.05\ kg * m^2\ I_z = 0.24\ kg * m^2$$

$$m = 2\ kg\ g = 9.81\ {}^{m}/_{s^2}$$

ในกรณีนี้ เราสนใจในกรณีที่ไม่มีลมเลย เราจึงสมมติโดยให้แรงลมมีค่า เท่ากับ 0

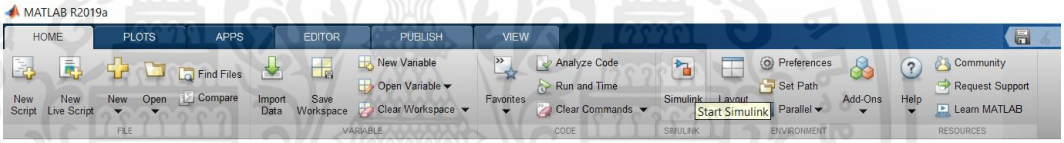
$$f_{wx} = f_{wy} = f_{wz} = 0\ \ N * m$$

3.7 การสร้าง Simulink และการ Tuning PID

ในการ tune pid เพื่อควบคุมโดรน ผู้ค้นคว้าต้องการที่จะจูนค่าพารามิเตอร์ที่ส่งผลต่อการเคลื่อนไหวโดรน(accelerate magnatic pressure) ให้เคลื่อนไหวได้สมบูรณ์หรือเข้าสู่set point (ตำแหน่ง x y z roll pitch yaw) ได้อย่างรวดเร็วและมีเสถียรภาพสูงสุด จากนั้นทำการจำลองการเคลื่อนไหวจากสมการควบคุมทางคณิตศาสตร์ผ่านโปรแกรมเพื่อสังเกตุการเคลื่อนที่ของโดรน

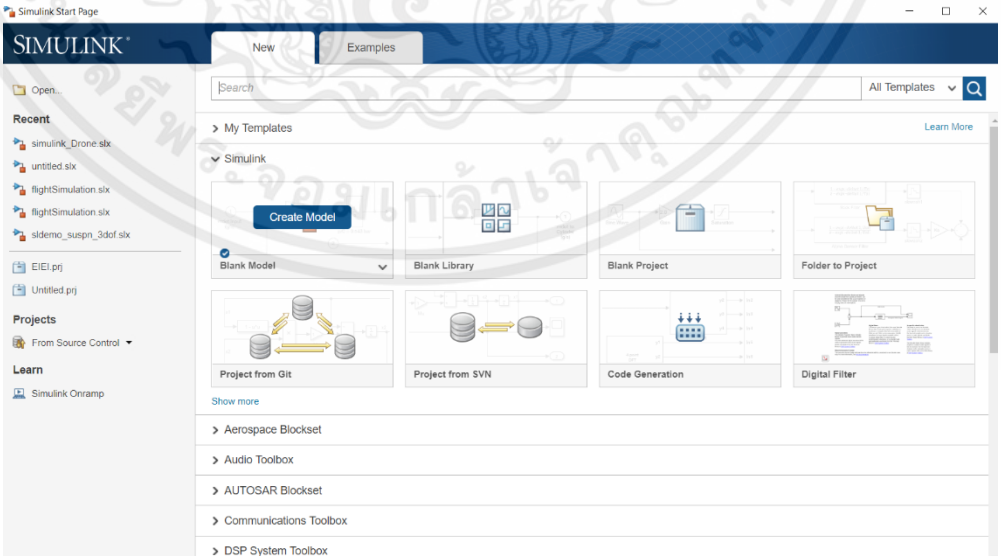
เริ่มต้นการทำ Simulink ใน MATLAB โดยจะนำสมการ Transfer Function จากหัวข้อที่ 3 มาใส่และหา PID Controller โดยขั้นตอนมีดังนี้

1. ไปที่หน้า HOME แล้วคลิกหัวข้อที่ชื่อว่า Simulink



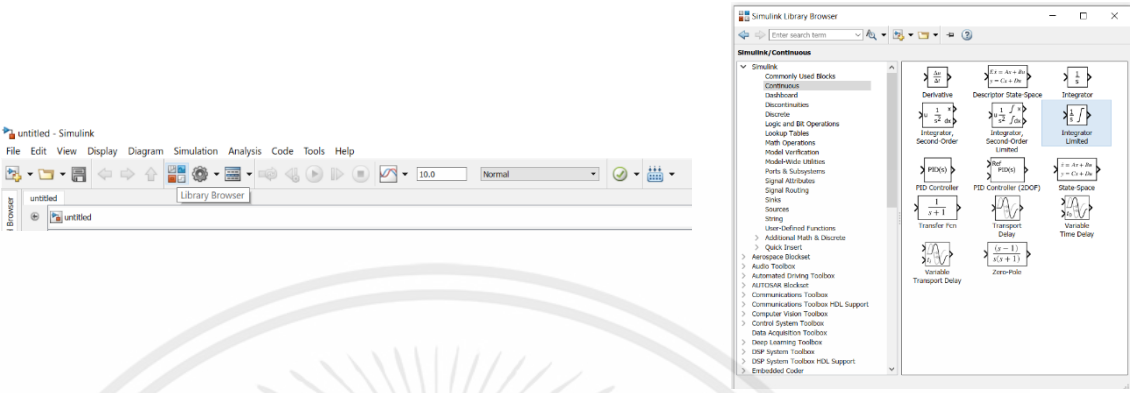
รูปที่ 3.8 รูปในการแสดงในการเริ่มเข้าฟังก์ชั่นการทำงาน Simulink

2. เลือกหัวข้อ Blank Model และคลิก Create Model



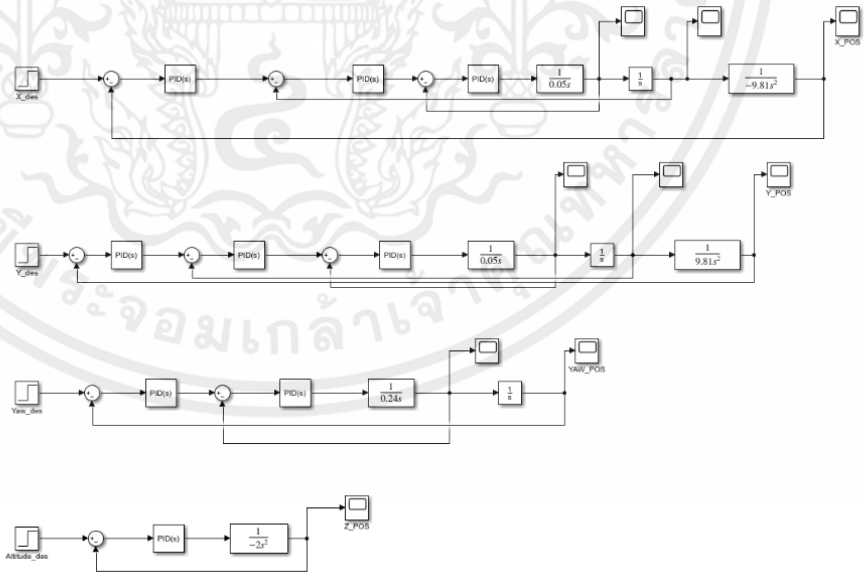
รูปที่ 3.9 แสดงฟังก์ชั่นต่างๆ ใน Simulink

3. เข้าไปที่หัวข้อที่มีชื่อว่า Library Browser เพื่อเรียกใช้งาน ฟังก์ชั่นต่างๆ ใน Blank Model



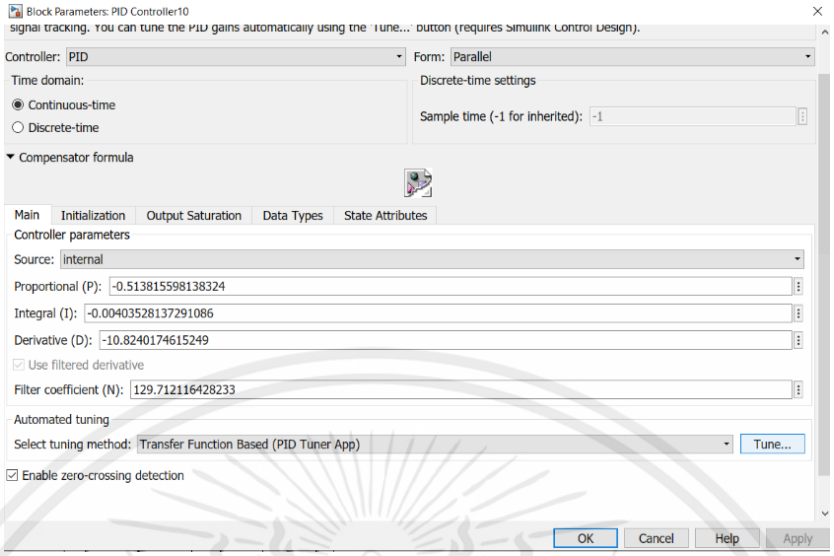
รูปที่ 3.10 ฟังก์ชั่นต่างๆ ใน Library Browser

4. ในขั้นตอนการ Tuning PID เลือก Function PID Controller และ Transfer Function แล้วนำมาเชื่อมต่อกันแบบอนุกรมโดยมี Feedback โดย input เราจะใช้แบบ unit step และ ติด Scope เพื่อดูกราฟ โดยเราจะใช้ Transfer Function จากหัวข้อที่ 3 มาใช้ในการ Tuning PID



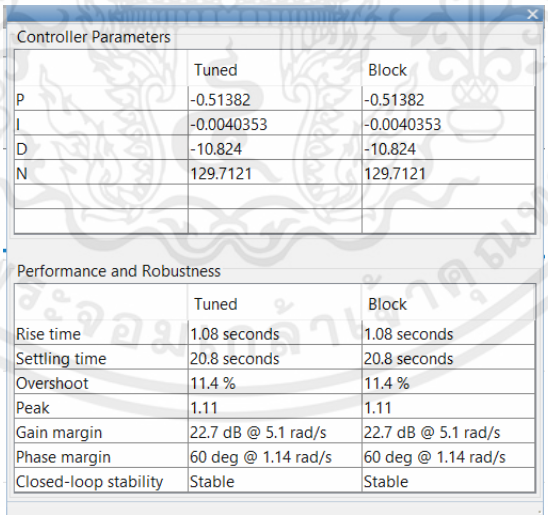
รูปที่ 3.11 เป็นการออกบบ Control loop PID ในการควบคุม UAV

5. ทำการ Tuning PID โดยเลือกที่ บล็อกของ PID Controller แล้วเลือกหัวข้อ Tune



รูปที่ 3.12 แสดงฟังก์ชั่นข้างใน PID Controller

6. เมื่อเข้าในโหมด PID Tuning เราก็ทำการเลือกตัวการควบคุมของเรา โดยเราใช้การควบคุมชนิด PIDF เนื่องจากมีค่า N เข้ามาเกี่ยวข้องด้วย จากนั้นเราก็จะสามารถเห็นค่าพารามิเตอร์ต่าง ๆได้ และนำไปใส่ใน PID Controller



Controller Parameters

	Tuned	Block
P	-0.51382	-0.51382
I	-0.0040353	-0.0040353
D	-10.824	-10.824
N	129.7121	129.7121

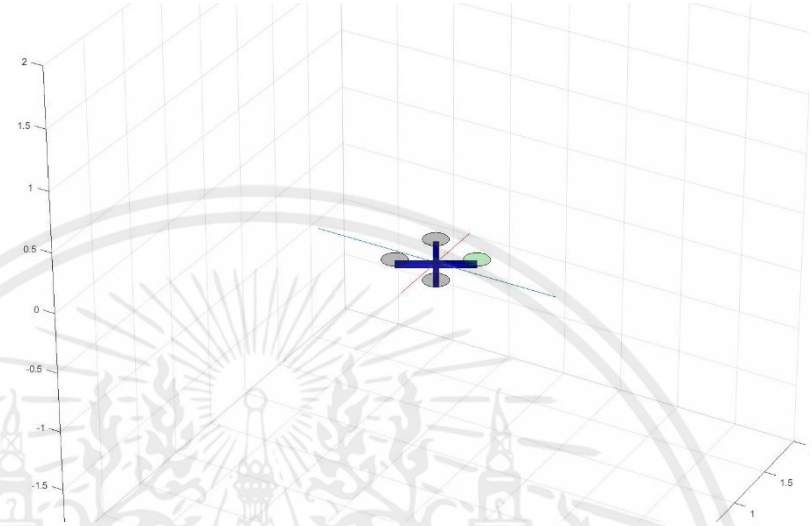
Performance and Robustness

	Tuned	Block
Rise time	1.08 seconds	1.08 seconds
Settling time	20.8 seconds	20.8 seconds
Overshoot	11.4 %	11.4 %
Peak	1.11	1.11
Gain margin	22.7 dB @ 5.1 rad/s	22.7 dB @ 5.1 rad/s
Phase margin	60 deg @ 1.14 rad/s	60 deg @ 1.14 rad/s
Closed-loop stability	Stable	Stable

รูปที่ 3.13 พารามิเตอร์ต่างของ PID Controller

3.8 การสร้างแบบจำลองของโดรน

เราสร้างแบบจำลองเพื่อดู การเคลื่อนที่ของมัน ว่าตัวโดรนนั้นสามารถบินไปถึงจุด Set point ได้หรือไม่ ซึ่งเราจะออกแบบใน MATLAB โดยตัวโดรนมีรูปร่าง ดังนี้



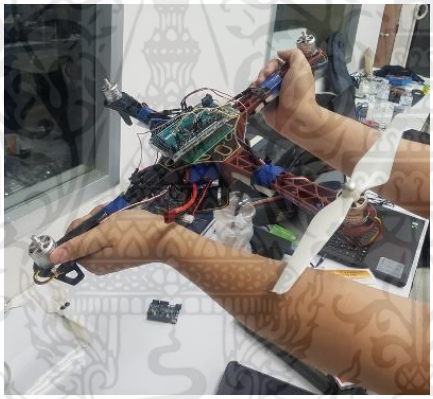
รูปที่ 3.14 เป็นแบบจำลองการเคลื่อนที่ของโดรนในโปรแกรม MATLAB

บทที่ 4
การทดลอง

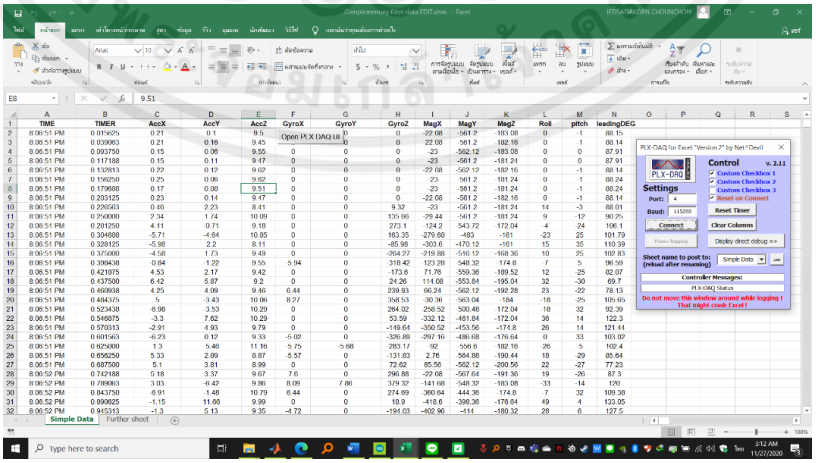
4.1 การหาค่า Roll Pitch & Yaw จาก A Super Fast Attitude Determination Algorithm for Consumer-Level Accelerometer and Magnetometer

4.1.1 ขั้นตอนการทดลอง

เปิดโปรแกรมที่วัดค่าอุปกรณ์ตรวจรู้ GY-86 (จากบทที่ 3) เพื่อที่จะรันค่าออกมาให้ขึ้นใน EXCEL แล้วนำค่าที่ได้ไปคำนวณใน MATLAB โดยการทดลองจะเป็นการจับโดรน แล้วโยกไปมา เพื่อที่วัดค่า ความเร่งที่เกิดขึ้นในแต่ละแกน และค่าแม่เหล็กที่เกิดขึ้นในแต่ละแกน ซึ่งจะต้องทำขั้นตอนนี้หลายๆครั้งเพื่อที่จะสังเกตค่าที่ได้ออกมา



รูปที่ 4.1 เป็นการทดลองเพื่อที่จะวัดค่าความเร่ง และแม่เหล็ก



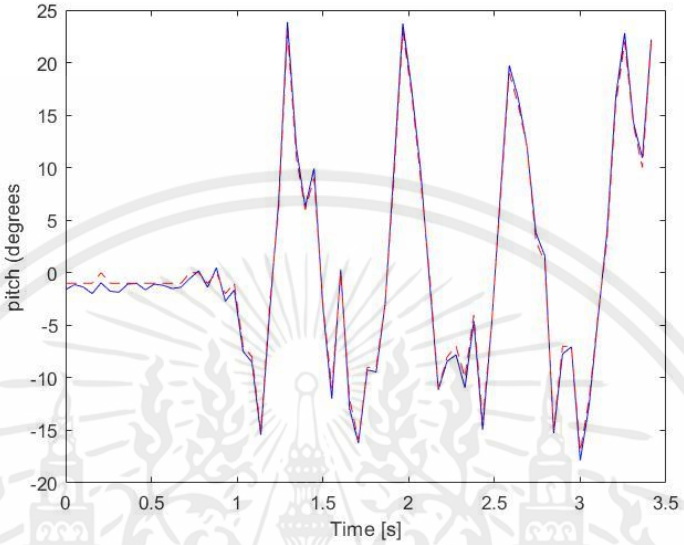
รูปที่ 4.2 เป็นนำค่าที่ทดลองลง PLX-DAQ

จากนั้นนำค่าที่ได้ export เข้าไปใน MATLAB เพื่อที่จะคำนวณออกมา

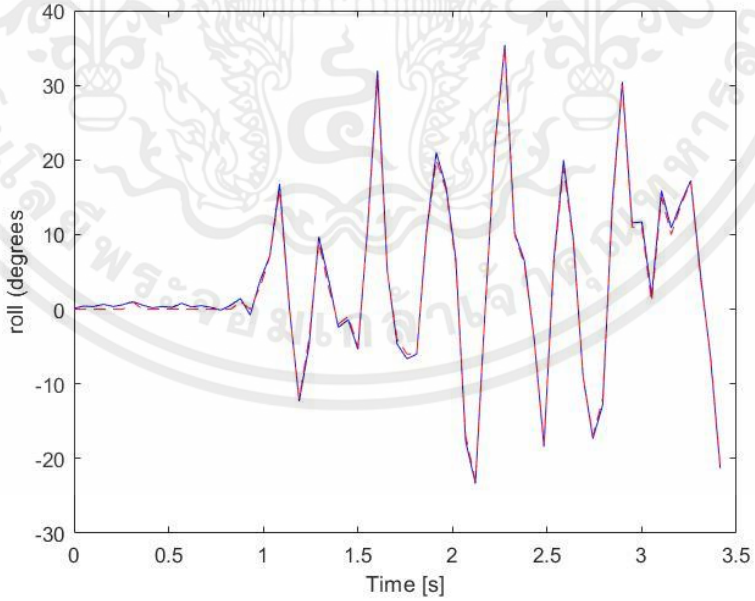
```
clear;
%INPUTS - a = accelerometer readings
% m - magnetometer readings

A = importdata('complementaryfilterdatatest6.xlsx');
[rr,cc]=size(A);
N = rr;
t=linspace(0,3.414,rr);
dt=t(2)-t(1);
accx=A(:,2);
accy=A(:,3);
accz=A(:,4);
magx=A(:,8);
magy=A(:,9);
magz=A(:,10);
roll_=A(:,11);
pitch_=A(:,12);
yaw_=A(:,13);
for i=1:N
 qq{i} = calculateq_am(accx(i),accy(i),accz(i),magx(i),magy(i),magz(i));
 qq{i}=qq{i}';

 [yaw(i), pitch(i), roll(i)] = quat2angle(qq{i},'ZYX');
end
%Degrees
pitch=pitch*180/pi;
roll=roll*180/pi;
yaw=yaw*180/pi;
figure();plot(t,roll,'b',t,roll_,'r--'); xlabel('Time [s]'); ylabel('roll (degrees');
figure();plot(t,pitch,'b',t,pitch_,'r--'); xlabel('Time [s]'); ylabel('pitch (degrees');
figure();plot(t,yaw,'b',t,yaw_,'r--');xlabel('Time [s]'); ylabel('yaw (degrees');
```

4.1.2 ผลการทดลอง

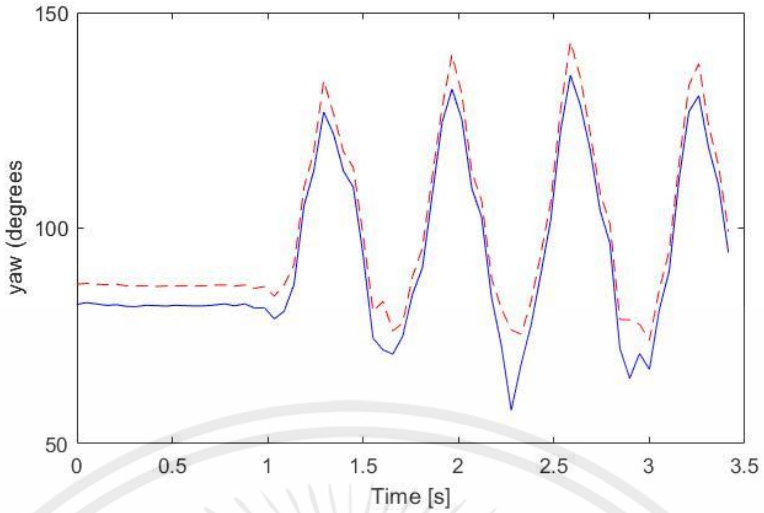
เราจะนำค่าที่ได้จากทฤษฎีมาเปรียบเทียบกับค่าจริงหรือค่าที่วัดได้จากอุปกรณ์ตรวจรู้ GY-86 โดย เส้นสีน้ำเงิน คือ ทฤษฎี เส้นประสีแดง คือ ค่าที่วัดได้



รูปที่ 4.3 เป็นการเปรียบค่า PITCH จากค่าทฤษฎีและค่าที่วัดได้



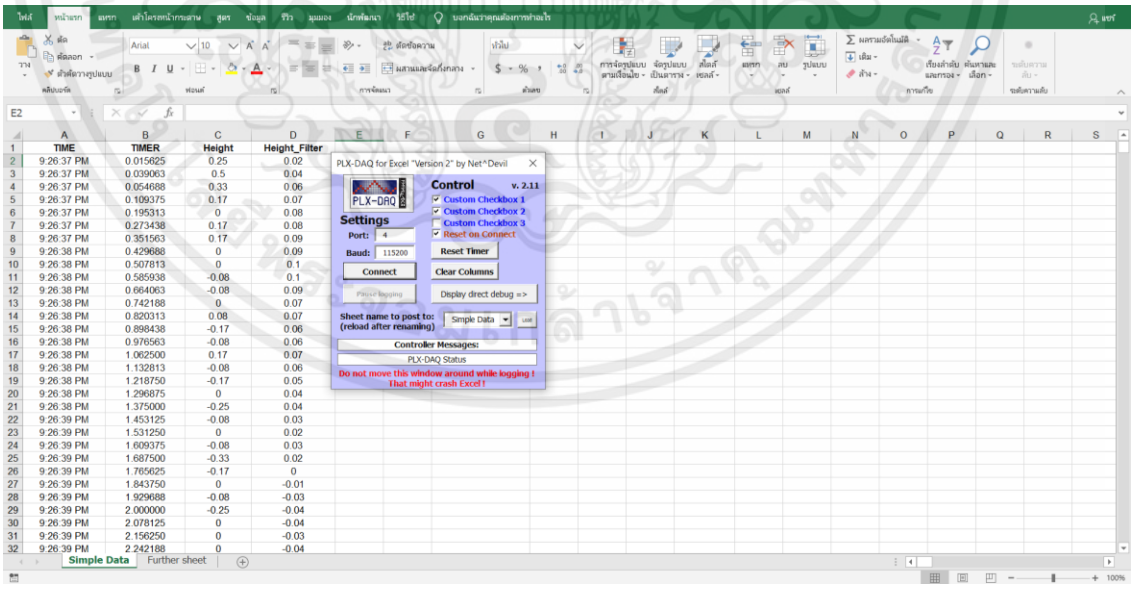
รูปที่ 4.4 เป็นการเปรียบค่า ROLL จากค่าทฤษฎีและค่าที่วัดได้



รูปที่ 4.5 เป็นการเปรียบค่า YAW จากค่าทฤษฎีและค่าที่วัดได้

4.2 การทดลองหาค่า HEIGHT จาก Barometric Pressure

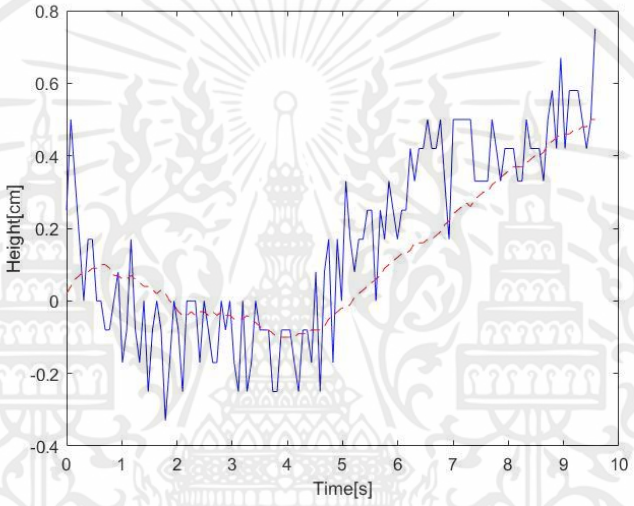
เนื่องจากการหาค่าความสูงจากเซนเซอร์ตรง ๆ เป็นสิ่งที่เป็นไปไม่ได้เลย เนื่องจากมี NOISE เข้ามารบกวน เราจึงต้องใช้สมการ LOW PASS FILTER เข้ามาช่วยเพื่อกรองความถี่ต่ำออกโดยใช้โค้ดและสมการ จาก บทที่ 3 แล้ว Upload ขึ้น EXCEL



รูปที่ 4.6 เป็นการ Upload ค่าความสูงแบบที่มี Noise และ Filter เข้ามา PLX-DAQ

จากนั้นนำค่าที่ได้เข้าไป RUN ใน MATLAB เพื่อ plot graph ออกมา

```
clear all;
A = importdata("HEIGHT_FILTER_DATA.xlsx");
[rr,cc] = size(A);
height = A(:,2);
height_filter = A(:,3);
t = linspace(0,9.570313,rr);
figure();plot(t,height,'b',t,height_filter,'r--');xlabel('Time[s]');ylabel('Height[cm]');
```



รูปที่ 4.7 เป็นการเปรียบเทียบค่าความสูงที่มี Noise (สีน้ำเงิน) และ Filter (เส้นสีแดง)

4.3 ผลการทดลองจากการควบคุม PID Controller

ในการหาค่า PID Controller ทำได้ด้วยการใช้ PID tuner ใช้เพื่อทำการหาค่า PIDF controller โดยทำตามขั้นตอนในบทที่ 3 และทำการทดสอบระบบโดยการเปลี่ยนค่า Kp Ki Kd และ N เพื่อดูผลตอบสนองนั้นเกิดการเปลี่ยนไปอย่างไรจากค่าที่ได้จาก PID tuner

4.3.1 Proportional gain (Kp)

Proportional gain ที่มีค่ามากจึงส่งผลให้ Rise Time เปลี่ยนแปลงเร็ว (เพิ่มเร็ว ลดเร็ว) แต่ถ้า ใส่มากจะส่งผลให้ค่าของผลตอบสนองระบบนั้นมีค่ามากเกินไปก็อาจจะให้เกิด Overshoot ได้ง่าย หรือถ้าหากใส่ค่าน้อยเกินไปก็จะส่งผลให้ค่าของผลตอบสนองระบบนั้นมีค่าน้อยและทำให้ระบบนั้นจะเข้าสู่ค่า set point ช้าลง

4.3.2 Integral gain (Ki)

ค่าของ Integral gain นั้นจะช่วยให้ระบบเข้าสู่ set point ได้เร็วขึ้นแต่ถ้าหากมีค่ามาเกินไปก็จะส่งผลให้เกิด Overshoot มากขึ้น ถ้าหากค่าน้อยเกินไปก็จะทำให้ ระบบนั้นเข้า set point ได้ช้ามาก

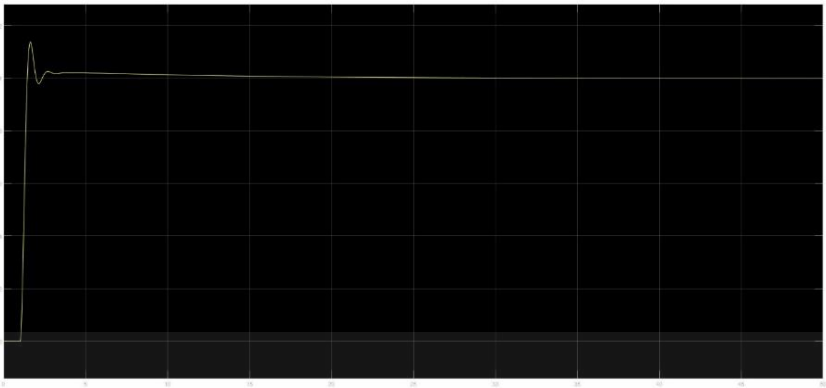
4.3.3 Derivative gain (Kd)

ค่า Derivative gain ส่งผลให้ผลการตอบสนองนั้นมีการแกว่งน้อยลงส่งผลให้ระบบนั้นเข้าสู่ set point นั้นได้เร็วยิ่งขึ้น ซึ่งค่าของ Derivative gain ส่งผลต่อกระบวณการมากเมื่อเทียบกับ Proportional gain และ Integral gain แต่ถ้าหากเราสามารถหาค่าที่เหมาะสมได้ก็จะส่งผลให้ระบบเข้าสู่ set point ได้เร็วยิ่งขึ้นเนื่องจากการแกว่งของกระบวณที่เกิดจาก gain อื่นๆ

4.3.4 Filter Coefficient (N)

ค่า Filter Coefficient เป็นค่าที่ส่งผลต่อ Derivative gain ช่วยในเรื่อง Step Cut Off ว่าจะเยอะหรือน้อย ทำหน้าที่เหมือน Derivative gain ทำให้มีการแกว่งน้อย ทำให้ถึง set point ได้ไวยิ่งขึ้น เป็นการแสดงผลของการควบคุมโดยใช้ PIDF Controller โดยเอาต์พุตจะเป็นการแสดงค่า (x, y, φ, z) ตามลำดับ โดย Step input มีค่าเท่ากับ 1 และ Step time มีค่าเท่ากับ 1

รูปที่ 4.8 แสดง $\boldsymbol{x(t)}$ กำลังเข้าสู่จุด step input



รูปที่ 4.9 แสดง $\boldsymbol{y(t)}$ กำลังเข้าสู่จุด step input

รูปที่ 4.10 แสดง $\boldsymbol{\varphi(t)}$ กำลังเข้าสู่จุด step input

รูปที่ 4.11 แสดง $\boldsymbol{z(t)}$ กำลังเข้าสู่จุด step input

4.3.5 การแสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งในแกน X, Y, Yaw

การแสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งในแกน X, Y, Yaw ผ่านการ Simulation ผ่าน MATLAB ซึ่งสามารถแสดง ดังนี้

ตารางที่ 6. 1 แสดงค่า PID Controller ของแกน X, Y, และ Z

	X			Y			Z	
	X_ Position	**Roll**	**Roll Rate**	**Y_ Position**	**Pitch**	**Pitch Rate**	**Yaw**	**Yaw Rate**
P	-0.51382	1.2191	0.17608	8.3325	0.065933	0.56899	7.7845	1.2363
I	-0.0040353	0.27701	0.1929	0.22354	0.0033098	0.58948	6.4854	2.0011
D	-10.824	1.0892	-0.00013419	30.7418	0.29879	-0.011447	2.2606	-0.00063765

4.3.6 การแสดงค่าพารามิเตอร์ PID ในการควบคุมระดับความสูง (Altitude Controller)

การแสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งความสูง คือ Altitude Controller ผ่านการ Simulation ผ่าน MATLAB ซึ่งสามารถแสดง ดังนี้

ตารางที่ 6. 2 แสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งความสูง

	Altitude Controller
P	-0.37136
I	-0.017385
D	-1.9478

บทที่ 5
สรุปผลการวิจัย และการพัฒนาต่อยอด

5.1 สรุปผลการวิจัย

การทดลองนี้เป็นการทดลองศึกษา ออกแบบ และพัฒนาอากาศยานไร้คนขับให้แก่ภาควิชาการวัดคุม โดยการทดลองครั้งนี้ได้ทำการศึกษาหลักการเคลื่อนที่ของอากาศยานไร้คนขับแบบสี่แกน (Quadrotor) ซึ่งเริ่มจากการออกแบบโครงสร้างของอากาศยานไร้คนขับ ออกแบบสมการทางคณิตศาสตร์ และเก็บค่าต่าง ๆที่จำเป็นในการบิน จากการทดลองในหัวข้อที่ 4.1 เป็นการหาค่าแกนทั้ง 3 แกนของการบิน ซึ่งก็คือแกน Raw, Pitch และ Yaw โดยการใช้ GY-86 เป็นตัวตรวจจับการเป็นไปของค่าการเปลี่ยนแปลงทั้งค่าความเร่ง และค่าสนามแม่เหล็กที่เปลี่ยนแปลงอันเนื่องมาจากการขยับอากาศยานไร้คนขับ จากรูปที่ 4.2 แสดงให้เห็นถึงค่าที่ได้ทำการเก็บจากการขยับอากาสยานไร้คนขับแล้วทำการป้อนข้อมูลเข้า MATLAB คำนวณตามสมการทางคณิตศาสตร์ออกมาเป็นกราฟแสดงความเปรียบเทียบของค่าจริงที่ได้ กับค่าตามทฤษฎี จะรูปที่ 4.3-5 จะเห็นได้ว่ากราฟที่ได้ทั้งสองเส้นทั้งค่าจริง และค่าตามทฤษฎี มีความใกล้เคียงกันมาก จึงสามารถสรุปได้ว่าค่าแกนต่าง ๆที่ต้องการมีความเป็นไปตามทฤษฎีการบินของอากาศยานไร้คนขับ

นอกจากนี้ทางเรายังได้ทำการทดลองหาค่าความสูงที่เกิดจากการบินอากาศบานไร้คนขับโดยใช้ Barometric Pressure เป็นตัวอ้างอิงความสูงที่เกิดอันเนื่องมาจากการบินในระดับความสูงต่าง ๆ โดยใช้ค่าความดันที่แตกต่างกันในระดับความสูงแปลงกลับมาเป็นระยะความสูงที่อากาศยานไร้คนขับบินขึ้นเหนือจากพื้นบิน ซึ่งทางเราได้ทำการใช้ LOW PASS FILTER เพื่อกรองค่าความถี่ต่ำ เพื่อเพิ่มความแม่นยำของค่าที่ได้โดยค่าที่ได้จะแสดงก่อน และหลังการกรองค่าดังรูปที่ 4.6 แล้วทำการอัปโหลดขึ้น MATLAB แสดงเป็นกราฟเปรียบเทียบระหว่างค่าก่อน และหลังการกรอง ดังรูปที่ 4.7

เมื่อได้ค่าที่ต้องการจากผลการทดลองข้างต้น เริ่มทำการควบคุมโดยวิธี PID Control เพื่อทำการ set up ค่าให้แก่ระบบทั้ง 3 แกน โดยกำหนดให้ step setup และ step time มีค่าเท่ากับ 1 โดยแสดงการเปลี่ยนไปดังรูปที่ 4.8-11 ซึ่งแสดงถึงระยะเวลาก่อนเข้าสู่ระบบคงที่ของระบบ โดยป้อนค่า PID ให้กับแกนทั้ง 3 แกน (X, Y, Yaw) ดังตารางที่ 6.1 ส่วนทางด้านความสูงแสดงดังรูปที่ 6.2

5.2 การพัฒนาต่อยอด

เนื่องจากการทดลองมีอุปสรรคทำให้ไม่สามารถทำการทดลองต่อเนื่องจนเสร็จสิ้นได้ จึงได้มีการส่งต่อเพื่อให้ทางรุ่นต่อไปสามารถนำความรู้ดังกล่าวไปพัฒนาต่อให้เสร็จสมบูรณ์ เพื่อใช้เป็นองค์ความรู้ให้กับทางภาควิชาการวัดคุม โดยการพัฒนาการทดลองอาจจะมีแนวทางดังนี้

- ทำการวัดค่ารอบของมอเตอร์ก่อนการทดลอง เนื่องจากมอเตอร์เป็นส่วนสำคัญในการบิน ฉนั้นจะต้องมีค่ารอบต่อวินาทีเท่ากันทั้ง 4 ตัวจะทำเราสามารถควบคุมมอเตอร์ได้อย่างถูกต้อง
- ออกแบบ System ID ให้กับอากาศยานไร้คนขับของจริง โดยที่อ้างอิงจาก Simulate ของการทดลองนี้

บรรณานุกรม

Michael J. Caruso. 2000. “ **Application of Magnetic Sensors for Low Cost Compass Systems**”

Mathew Watson. 2013. “**The Design and Implementation of a Robust AHRS for Integration into a Quadrotor Platform**”

Francesco Sabatino. 2015. “**Quadrotor control: modeling, nonlinear, control design, and simulation**”

Jin Wu, Zebo Zhou, Jingjun Chen ,Hassen Fourati, Rui Li. (2016). “**Fast Complementary Filter for Attitude Estimation Using Low-Cost MARG Sensors**”

Nianrong Zhou, Guolei Xu, Jie Wei, Lijun Tang. 2018. “**Relative height measurement based on collaborative information fusion of acceleration and barometric pressure**”

Jin Wu, Zebo Zhou, Hassen Fourati, Yuhua Cheng. 2018. “**A Super Fast Attitude Determination Algorithm for Consumer-Level Accelerometer and Magnetometer**”

Aws Abdulsalam Najm, Ibraheem Kasim Ibraheem. 2019. “**Nonlinear PID controller design for a 6-DOF UAV quardrotor system**”

Daniel Morris, XiaoQi Chen, and Adriel Kind. 2020. “**Real-time System Identification of Quadrotor Dynamics**”

ThaiSensorModule.com. 2017. **Mega 2560 R3 (Arduino compatible board, MCU : ATmega2560 16MHz).** [Onlline]. Available : http://thaisensormodule.com/index.php/other-module/mega2560r3

Myarduino. 2017. **GY-86 IMU 10DOF MS5611 HMC5883L MPU6050.** [Online]. Available : https://www.myarduino.net/product/3201/gy-86-imu-10dof- ms5611-**hmc5883l-mpu6050**

Commandronestore. (2017). **GY-521 MPU6050.** [Online] Available : https://commandronestore.com/products/bo100.php

Wikipedia. (2021). **MATLAB**. [Online]. Available : https://en.wikipedia.org/wiki/MATLAB

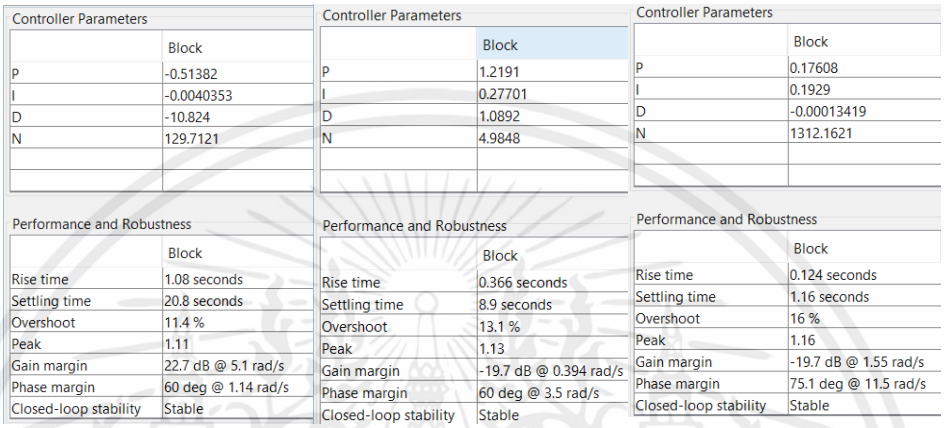
Myarduino. (2015). **MultiRotor Part 1 การทำงานของเครื่องบิน.** [Online] Available : https://www.myarduino.net/article/1/multirotor-part-1-การทำงานของเครื่องบิน

ME KMUTT. (2010). **System identification. [Online]** Available : http://mechanic-top.blogspot.com/2010/04/system-identification.htm

ภาคผนวก

ภาคผนวก ก
การแสดงค่าพารามิเตอร์ PID ในแกนต่าง ๆ

ก.1 การแสดงค่าพารามิเตอร์ที่ PID ที่ได้ในผ่าน Simulink



Controller Parameters

	Block
P	-0.51382
I	-0.0040353
D	-10.824
N	129.7121

Performance and Robustness

	Block
Rise time	1.08 seconds
Settling time	20.8 seconds
Overshoot	11.4 %
Peak	1.11
Gain margin	22.7 dB @ 5.1 rad/s
Phase margin	60 deg @ 1.14 rad/s
Closed-loop stability	Stable

Controller Parameters

	Block
P	1.2191
I	0.27701
D	1.0892
N	4.9848

Performance and Robustness

	Block
Rise time	0.366 seconds
Settling time	8.9 seconds
Overshoot	13.1 %
Peak	1.13
Gain margin	-19.7 dB @ 0.394 rad/s
Phase margin	60 deg @ 3.5 rad/s
Closed-loop stability	Stable

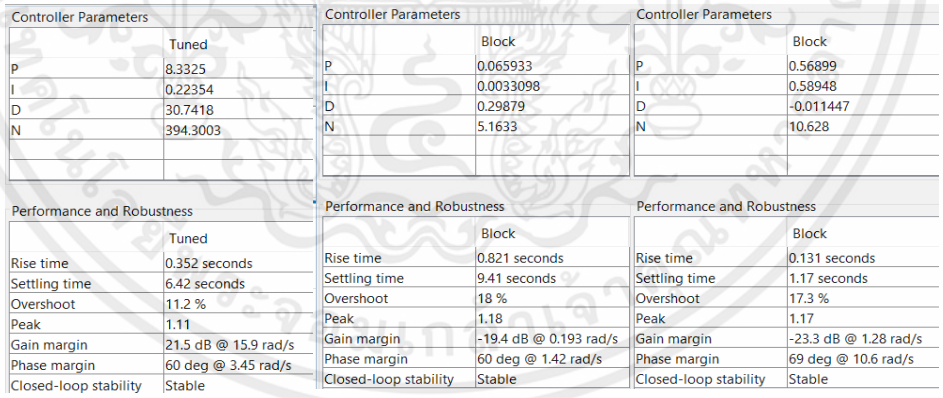
Controller Parameters

	Block
P	0.17608
I	0.1929
D	-0.00013419
N	1312.1621

Performance and Robustness

	Block
Rise time	0.124 seconds
Settling time	1.16 seconds
Overshoot	16 %
Peak	1.16
Gain margin	-19.7 dB @ 1.55 rad/s
Phase margin	75.1 deg @ 11.5 rad/s
Closed-loop stability	Stable

รูปที่ ก.1 แสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งในแกน X คือ X_Position Controller, Roll Controller, Roll Rate Controller ตามลำดับ



Controller Parameters

	Tuned
P	8.3325
I	0.22354
D	30.7418
N	394.3003

Performance and Robustness

	Tuned
Rise time	0.352 seconds
Settling time	6.42 seconds
Overshoot	11.2 %
Peak	1.11
Gain margin	21.5 dB @ 15.9 rad/s
Phase margin	60 deg @ 3.45 rad/s
Closed-loop stability	Stable

Controller Parameters

	Block
P	0.065933
I	0.0033098
D	0.29879
N	5.1633

Performance and Robustness

	Block
Rise time	0.821 seconds
Settling time	9.41 seconds
Overshoot	18 %
Peak	1.18
Gain margin	-19.4 dB @ 0.193 rad/s
Phase margin	60 deg @ 1.42 rad/s
Closed-loop stability	Stable

Controller Parameters

	Block
P	0.56899
I	0.58948
D	-0.011447
N	10.628

Performance and Robustness

	Block
Rise time	0.131 seconds
Settling time	1.17 seconds
Overshoot	17.3 %
Peak	1.17
Gain margin	-23.3 dB @ 1.28 rad/s
Phase margin	69 deg @ 10.6 rad/s
Closed-loop stability	Stable

รูปที่ ก.2 แสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งในแกน Y คือ Y_Position Controller, Pitch Controller, Pitch Rate Controller ตามลำดับ

Controller Parameters

	Block
P	7.7845
I	6.4854
D	2.2606
N	883.7429	

Performance and Robustness

	Block
Rise time	0.199 seconds
Settling time	2.34 seconds
Overshoot	9.22 %
Peak	1.09
Gain margin	-Inf dB @ 0 rad/s
Phase margin	69 deg @ 7.73 rad/s
Closed-loop stability	Stable

Controller Parameters

	Block
P	1.2363
I	2.0011
D	-0.00063765
N	1938.8272	

Performance and Robustness

	Block
Rise time	0.0845 seconds
Settling time	0.73 seconds
Overshoot	16.6 %
Peak	1.17
Gain margin	-19.9 dB @ 2.42 rad/s
Phase margin	75.2 deg @ 17 rad/s
Closed-loop stability	Stable

รูปที่ ก.2 แสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่ง Yaw คือ Yaw Controller, Yaw Rate Controller ตามลำดับ

Controller Parameters

	Block
P	-0.37136
I	-0.017385
D	-1.9478
N	5.454	

Performance and Robustness

	Block
Rise time	1.31 seconds
Settling time	13.9 seconds
Overshoot	14 %
Peak	1.14
Gain margin	-26.4 dB @ 0.0945 rad/s
Phase margin	69 deg @ 1 rad/s
Closed-loop stability	Stable

รูปที่ ก.3 แสดงค่าพารามิเตอร์ PID ในการควบคุมตำแหน่งความสูง คือ Altitude Controller

ภาคผนวก ข
โปรแกรมคำนวณ

ข.1 โปรแกรมคำนวณ Accelerometer magnetometer ของทั้งแกน X-Y-Z

```
function qq = calculateq_am(accx,accy,accz,magx,magy,magz)
%INPUTS - a = accelerometer readings
% m - magnetometer readings
aa = [accx;accy;accz];
mm = [magx;magy;magz];
Ab=aa/norm(aa,2); Mb=mm/norm(mm,2);
ax=Ab(1); ay=Ab(2); az=Ab(3);
mx=Mb(1); my=Mb(2); mz=Mb(3);
mD=dot(Ab,Mb);
mN=sqrt(1-mD^2);
%Quarternions
qq = [- ay * (mN + mx) + ax * my;
(az - 1) * (mN + mx) + ax * (mD - mz);
(az - 1) * my + ay * (mD - mz);
 az * mD - ax * mN - mz];
qq = qq ./ norm(qq,2);
```

ข.2 โปรแกรมการทำงานของเซ็นเซอร์ GY-86 และ อัพโหลดข้อมูลขึ้น EXCEL

```
#include <HMC5883L.h>
#include <Wire.h>
#include <MPU6050.h>
#include <MS5611.h>
#include <Servo.h>
HMC5883L compass;
MS5611 ms5611;
MPU6050 mpu;
//Timers BARO
```

```
double referencePressure;
double Pressure_Filter;
//MOTOR
#define MAX_SIGNAL 2000
#define MIN_SIGNAL 1000
#define MOTORCWBLACK 5
#define MOTORCWRED 6
#define MOTORCCWBLACK 3
#define MOTORCCWRED 2
char data;
int motor1,motor2,motor3,motor4;
int speedmotor;
Servo motorcwblack;
Servo motorcwred;
Servo motorccwblack;
Servo motorccwred;
// Timers
unsigned long timerACCMillis = 0;
const unsigned long periodACC = 50;
void print_time(unsigned long time_millis);
void setup()
{
Serial.begin(115200);
// Initialize HMC5883L
while (!compass.begin())
{
Serial.println("Could not find a valid HMC5883L sensor, check wiring!");
delay(500);
 }
Serial.println("Initialize HMC5883L");
// Initialize MPU6050
while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
{
```

```
Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
delay(500);
}
Serial.println("Initialize MPU6050");
/*BARO*/
while(!ms5611.begin(MS5611_HIGH_RES))
{
Serial.println("Sensor MS5611 NOT FOUND, Check your connection!");
delay(500);
}
Serial.println("Initialize MS5611");
 // Set measurement range
compass.setRange(HMC5883L_RANGE_1_3GA);
// Set measurement mode
compass.setMeasurementMode(HMC5883L_CONTINOUS);
// Set data rate
compass.setDataRate(HMC5883L_DATARATE_30HZ);
// Set number of samples averaged
 compass.setSamples(HMC5883L_SAMPLES_8);
// Set calibration offset. See HMC5883L_calibration.ino
compass.setOffset(56, -107);
// Calibrate gyroscope. The calibration must be at rest.
// If you don't want calibrate, comment this line.
mpu.calibrateGyro();
// Set threshold sensivty. Default 3.
// If you don't want use threshold, comment this line or set 0.
mpu.setThreshold(3);
// setup topic of excel
Serial.println("CLEARDATA");
Serial.println("LABEL,TIME,TIMER,AccX,AccY,AccZ,GyroX,GyroY,Gy
roZ,MagX,MagY,MagZ,Roll,pitch,HeadingDEG,Height");
Serial.println("RESETTIMER");
// Read ReferencePressure p0
```

```
referencePressure = ms5611.readPressure();
Pressure_Filter = referencePressure;
// Checking Settings
checkSettings();
Serial.println("Program begin...");
delay(1000);
 Serial.println("This program will start the ESC.");
//Determine variable MOTOR
 motorcwblack.attach(MOTORCWBLACK);
motorcwred.attach(MOTORCWRED);
motorccwblack.attach(MOTORCCWBLACK);
motorccwred.attach(MOTORCCWRED);
motorcwblack.writeMicroseconds(MIN_SIGNAL);
motorcwred.writeMicroseconds(MIN_SIGNAL);
motorccwblack.writeMicroseconds(MIN_SIGNAL);
motorccwred.writeMicroseconds(MIN_SIGNAL);
Serial.println("Input speed motor Setup : ");
//input Initial Speed motor
for(int i=1;i<i+1;i++){
if(Serial.available()>0)
{
speedmotor = Serial.parseInt();
motorcwblack.writeMicroseconds(speedmotor);
motorcwred.writeMicroseconds(speedmotor);
motorccwblack.writeMicroseconds(speedmotor);
motorccwred.writeMicroseconds(speedmotor);
 break;
}
}
Serial.print("speedmotorsetup :");Serial.println(speedmotor);
Serial.println("The ESC is calibrated");
Serial.println("----");
Serial.println("Now, type a values between 1000 and 2000 and press enter");
```

```
  Serial.println("and the motor will start rotating.");
  Serial.println("Send 1000 to stop the motor and 2000 for full throttle");
  }
  void checkSettings()
  {
  Serial.print("Oversampling: ");
  Serial.println(ms5611.getOversampling());
  }
  void loop() {
  //Select motor then input speed
  if (Serial.available() > 0)
   {
  //motor a
  data = Serial.read();
  switch(data)
  {
  case 'a':Serial.print("input motor1 :");
  for(int i=1;i<i+1;i++)
   {
   if(Serial.available() > 0)
   {
  motor1 = Serial.parseInt();Serial.println(motor1);
  motorcwblack.writeMicroseconds(motor1);break;
  }
}
  break;
  //motor b
  case 'b':Serial.print("input motor2 :");
  for(int i=1;i<i+1;i++)
{
  if(Serial.available() > 0)
  {
```

```
motor2 = Serial.parseInt();Serial.println(motor2);
motorccwblack.writeMicroseconds(motor2);break;
 }
 }
break;
//motor c
case 'c':Serial.print("input motor3 :");
for(int i=1;i<i+1;i++)
{
if(Serial.available() > 0)
{
 motor3 = Serial.parseInt();Serial.println(motor3);
motorcwred.writeMicroseconds(motor3);break;
}
}
break;
// motor d
case 'd':Serial.print("input motor4 :");
for(int i=1;i<i+1;i++)
{
f(Serial.available() > 0)
{
motor4 = Serial.parseInt();Serial.println(motor4);
motorccwred.writeMicroseconds(motor4);break;
}
}
break;
//Show speed motor at that time
case 'e':
Serial.print("motor1 : ");
Serial.println(motor1);
Serial.print("motor2 : ");
Serial.println(motor2);
```

```
Serial.print("motor3 : ");
Serial.println(motor3);
Serial.print("motor4 : ");
Serial.println(motor4);
break;
// Stop motor
case 'f':
motor1=1000; motor2=1000; motor3=1000; motor4=1000;
motorcwblack.writeMicroseconds(motor1);
motorccwblack.writeMicroseconds(motor2);
motorcwred.writeMicroseconds(motor3);
motorccwred.writeMicroseconds(motor4);
Serial.println("STOP");
break;
}
}
unsigned long currentMillis = millis();
//DELAY
if(currentMillis - timerACCMillis >= periodACC){
// Read normalized values
Vector normAccel = mpu.readNormalizeAccel();
Vector normGyro  = mpu.readNormalizeGyro();
// Calculate Pitch & Roll (Acc)
int pitch = -(atan2(normAccel.XAxis, sqrt(normAccel.YAxis*normAccel.YAxis +
normAccel.ZAxis*normAccel.ZAxis))*180.0)/M_PI;
int roll = (atan2(normAccel.YAxis, normAccel.ZAxis)*180.0)/M_PI;
Vector norm = compass.readNormalize();
float PITCH = pitch*M_PI/180;
float ROLL = roll*M_PI/180;
// Calculate heading
float heading =  atan2( (norm.YAxis*cos(ROLL) + norm.ZAxis*sin(ROLL) ) ,
(norm.XAxis*cos(PITCH) + norm.YAxis*sin(PITCH)*sin(ROLL)-
norm.ZAxis*sin(PITCH)*cos(ROLL)) );
```

```
// pitch and row are from accelerometer
float declinationAngle = (4.0 + (26.0 / 60.0)) / (180 / M_PI);
heading += declinationAngle;
// Correct for heading < 0deg and heading > 360deg
 if (heading < 0)
 {
 heading += 2 * PI;
}
if (heading > 2 * PI)
{
heading -= 2 * PI;
}

// Convert to degrees
float headingDegrees = heading * 180/M_PI;
float Degrees = 360-headingDegrees ;
//BAROMETER
double realTemperature = ms5611.readTemperature();
double realPressure = ms5611.readPressure();
double absoluteAltitude = ms5611.getAltitude(realPressure);
// Low pass Filter Pressure
Pressure_Filter = 0.9391*Pressure_Filter +0.0609*(realPressure);
double relativeAltitude = ms5611.getAltitude(Pressure_Filter,
referencePressure);
// Upload to excel PLX-DAQ
Serial.println((String) "Data,TIME,TIMER,"  + normAccel.XAxis+ "," +
normAccel.YAxis+ "," + normAccel.ZAxis+ "," + normGyro.XAxis+ "," +
normGyro.YAxis+ "," + normGyro.ZAxis+ "," + norm.XAxis + "," + norm.YAxis +
"," + norm.ZAxis + ","+ roll+ "," + pitch+ "," + Degrees + "," + relativeAltitude);
//Update Timing
timerACCMillis = currentMillis;
}
}
```