ระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม

INVERTED PENDULUM ROBOT CONTROL SYSTEM

นายกฤตณัฐ เกิดสว่าง
นางสาวพรจุติ ทรวงแก้ว
นายภูริเดช จันทะโก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมระบบควบคุม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2551

ปริญญานิพนธ์ปีการศึกษา 2551

ภาควิชาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม

INVERTED PENDULUM ROBOT CONTROL SYSTEM

ผู้จัดทำ นายกฤตณัฐ เกิดสว่าง 48010019

นางสาวพรจุติ ทรวงแก้ว 48010575

นายภูริเดช จันทะโก 48010689

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. ถาวร เบญจนราสุทธิ์)

ระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม

โดย

นายกฤตณัฐ เกิดสว่าง 48010019

นางสาวพรจุติ ทรวงแก้ว 48010575

นายภูริเดช จันทะโก 48010689

อาจารย์ที่ปรึกษา

ผศ.ดร. ถาวร เบญจนราสุทธิ์

ปีการศึกษา 2551

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ นำเสนอการศึกษา และการออกแบบระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม โดยระบบอินเวิร์ทเพนดูลัมได้รับความสนใจมานานในแวดวงวิศวกรรมระบบควบคุม เนื่องจากมีคุณสมบัติที่เป็นระบบไม่เป็นเชิงเส้นและไม่มีเสถียรภาพ

การออกแบบและสร้างหุ่นยนต์อินเวิร์ทเพนดูลัม มีพื้นฐานมาจากพาหนะขนส่งส่วนบุคคลสองล้อ เซกเวย์เอชที (Segway HT) ที่อาศัยหลักการของระบบควบคุมอินเวิร์ทเพนดูลัม มาออกแบบตัวควบคุมเพื่อรักษาสมดุล และเสถียรภาพของหุ่นยนต์ โดยในโครงงานนี้ใช้วิธีการออกแบบตัวควบคุมแบบตัวคุมค่ากำลังสองเชิงเส้น และใช้เซนเซอร์สองชนิดวัดค่ามุมเบี่ยงเบน คือ เซนเซอร์ไจโรสโคป และเซนเซอร์ความเร่ง โดยได้นำแคลแมนฟิลเตอร์มาใช้ในการประมวลผลเซนเซอร์ร่วมกัน เพื่อปรับปรุงค่ามุมให้มีความแม่นยำ โดยอาศัยหลักการข้างต้น ตัวควบคุมที่ได้ออกแบบจะถูกนำมาโปรแกรมให้กับไมโครคอนโทรลเลอร์ จากการทดลองพบว่าระบบควบคุมที่ออกแบบสามารถรักษาสมดุลของหุ่นยนต์อินเวิร์ทเพนดูลัมได้

INVERTED PENDULUM ROBOT CONTROL SYSTEM

By

Mr.Krittanat Kurdsawang

Miss Pornjuti Soungkaew

Mr.Puridech Juntago

Advisor

Asst. Prof. Dr. Taworn Benjanarasuth

Academic Year 2008

Abstract

This thesis presents study and design of inverted pendulum robot control system. The inverted pendulum system has long been interesting of control engineers. It is due to the fact that the system is inherited the non-linearity and unstablility attributes

Designing and building of inverted pendulum robot is based on individual vehicle transportation called Segway HT. The controller is designed to control the system so that it is balanced and stable. In this thesis, Linear Quadratic Regulator (LQR) technique is applied to design the controller, Two types of sensors are used to provide tilt angle are gyroscopes and accelerometer. Kalman filter is then utilized for data fusion to improve the measurement of angle. The Theoretical concepts are finally realized in the microcontroller. From the experiments, the designed controller system can balance and stabilize the Inverted pendulum robot.

กิตติกรรมประกาศ

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

ขอบคุณเพื่อนๆ ทุกคนที่ให้กำลังใจ สนับสนุนอุปกรณ์ที่ขาดเหลือ กระตุ้นเตือน รวมทั้งคอยถามไถ่ความคืบหน้าของโครงงานอยู่เสมอ

สุดท้ายนี้ผู้จัดทำขอกราบขอบพระคุณบิดา มารดา และครอบครัว ที่คอยเป็นกำลังใจที่ดีตลอดมา รวมถึงการสนับสนุนในเรื่องของงบประมาณที่ขาดเหลือ ตลอดจนเป็นแรงบันดาลใจที่ดีที่สุดที่ทำให้โครงงานนี้สำเร็จสมบูรณ์ลงได้

ผู้จัดทำ
นายกฤตณัฐ เกิดสว่าง
นางสาวพรจุติ ทรวงแก้ว
นายภูริเดช จันทะโก

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญรูป(ต่อ)

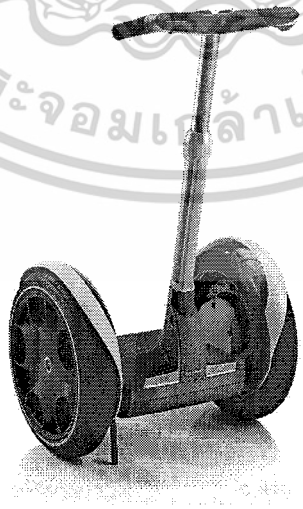
สารบัญตาราง

บทที่ 1
บทนำ

1.1 กล่าวนำ

วิศวกรรมระบบควบคุม เป็นสาขาวิชาที่ศึกษาการประยุกต์ใช้ทฤษฎีต่างๆ เพื่อออกแบบควบคุมระบบให้มีประสิทธิภาพที่ดี และเป็นไปตามวัตถุประสงค์ ซึ่งการศึกษาทางวิศวกรรมระบบควบคุมจะอาศัยการหาแบบจำลองของระบบด้วยสมการทางคณิตศาสตร์ การออกแบบตัวควบคุมหรือตัวชดเชยแบบต่างๆ วงจรอิเล็กทรอนิกส์ และวงจรอิเล็กทรอนิกส์กำลัง รวมถึงศึกษาเกี่ยวกับการพัฒนาโปรแกรมคอมพิวเตอร์ อุปกรณ์วัดและแปลงสัญญาณ โดยการนำความรู้ข้างต้นมาประยุกต์ใช้ควบคุมระบบทางกายภาพจริงเป็นเป้าหมายของการศึกษาในสาขาวิชานี้

เนื่องจากโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัมที่นำเสนอในปริญญานิพนธ์ฉบับนี้ เป็นเรื่องที่นิยมศึกษากันอย่างแพร่หลาย โดยผลงานที่ถูกกล่าวถึงอย่างมาก คือเซกเวย์เอชที (Segway Human Transporter: Segway HT) ของดีน คาเมน (Dean Kamen) นักประดิษฐ์ และผู้ประกอบการ ที่ได้ริเริ่มความคิดที่จะสร้างระบบขนส่งส่วนบุคคลแบบใหม่ในช่วงกลางของทศวรรษ 1990 เซกเวย์เอชที มีลักษณะเป็นพาหนะที่มีสองล้อวางขนานในแนวเดียวกันแต่ละล้อขับเคลื่อนแยกกันอย่างอิสระ ดังรูปที่ 1.1 โดยมีตัวควบคุมสมดุล ที่แม้ว่ามีหรือไม่มีผู้ขับขี่ ก็สามารถทรงตัวอยู่ได้ ซึ่งการควบคุมสมดุลนั้น จำเป็นต้องออกแบบตัวควบคุมมาช่วยให้เกิดเสถียรภาพแก่ระบบ เซกเวย์เอชทีใช้ตัวควบคุมแบบปริภูมิสเตท (State space) ดังนั้นจึงสนใจศึกษาแบบจำลองหุ่นยนต์อินเวิร์ทเพนดูลัมที่ใช้เซกเวย์เอชทีเป็นต้นแบบ และออกแบบตัวควบคุมด้วยปริภูมิสเตท



รูปที่ 1.1 เซกเวย์เอชที

1.2 วัตถุประสงค์ของปริญญานิพนธ์

ศึกษาหลักการทำงานพื้นฐานของหุ่นยนต์อินเวิร์ทเพนดูลัมและระบบควบคุม โดยออกแบบและสร้างระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัมซึ่งสามารถรักษาสมดุลได้ด้วยตัวเองบนล้อที่ขนานอยู่บนแกนเดียวกัน โดยมีหลักการทำงานที่คล้ายกับเซกเวย์เอชที และสามารถสร้างได้ง่ายจากชิ้นส่วนที่หาได้โดยทั่วไปอีกด้วย

1.3 ขั้นตอนการศึกษาและการจัดทำโครงงาน

1. ศึกษาระบบควบคุมอินเวิร์ทเพนดูลัม และอุปกรณ์ต่างๆ ที่ใช้ในโครงงาน
2. ออกแบบและสร้างชิ้นงานของโครงงาน ด้วยโปรแกรม SolidWorks
3. ออกแบบและสร้างวงจรใช้งานให้กับอุปกรณ์ที่ใช้ในโครงงาน
4. พัฒนาโปรแกรมด้วยภาษาซี ในการทดลอง และควบคุมหุ่นยนต์
5. ออกแบบตัวควบคุมที่เหมาะสมต่อการควบคุมเสถียรภาพของระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม และทดลองด้วยแบบจำลองของ MATLAB ก่อนใช้งานกับระบบจริง
6. ศึกษาเทคนิคใหม่ๆ ที่เข้ามาช่วยปรับปรุงประสิทธิภาพของระบบอินเวิร์ทเพนดูลัม เช่น แคลแมนฟิลเตอร์ เป็นต้น

1.4 รายละเอียดของปริญญานิพนธ์

เนื้อหาในปริญญานิพนธ์ฉบับนี้ แบ่งออกเป็น 5 บท ซึ่งครอบคลุมเนื้อหาดังนี้

บทที่ 1 บทนำ กล่าวนำถึงวัตถุประสงค์ ขั้นตอนการศึกษา และการจัดทำโครงงาน พร้อมด้วยรายละเอียดโดยย่อของปริญญานิพนธ์แต่ละบท

บทที่ 2 ทฤษฎีและความรู้ที่เกี่ยวข้อง ในบทนี้จะกล่าวถึงแบบจำลองทางคณิตศาสตร์ของหุ่นยนต์อินเวิร์ทเพนดูลัม ทฤษฎีการออกแบบตัวควบคุม แคลแมนฟิลเตอร์ หลักการทำงานของเซนเซอร์ ไมโครคอนโทรลเลอร์ และการควบคุมมอเตอร์ไฟฟ้ากระแสตรง

บทที่ 3 หลักการออกแบบ กล่าวถึงการออกแบบโครงสร้างของหุ่นยนต์อินเวิร์ทเพนดูลัม การออกแบบวงจร และการออกแบบตัวควบคุม

บทที่ 4 การทดลอง นำเสนอวิธีการทดลอง ผลการทดลอง และกราฟผลการทดลอง

บทที่ 5 บทวิจารณ์ และสรุป นำเสนอปัญหาที่เกิดขึ้นในการทดลอง และแนวทางพัฒนาโครงงานในอนาคต

บทที่ 2
ทฤษฎีและความรู้พื้นฐาน

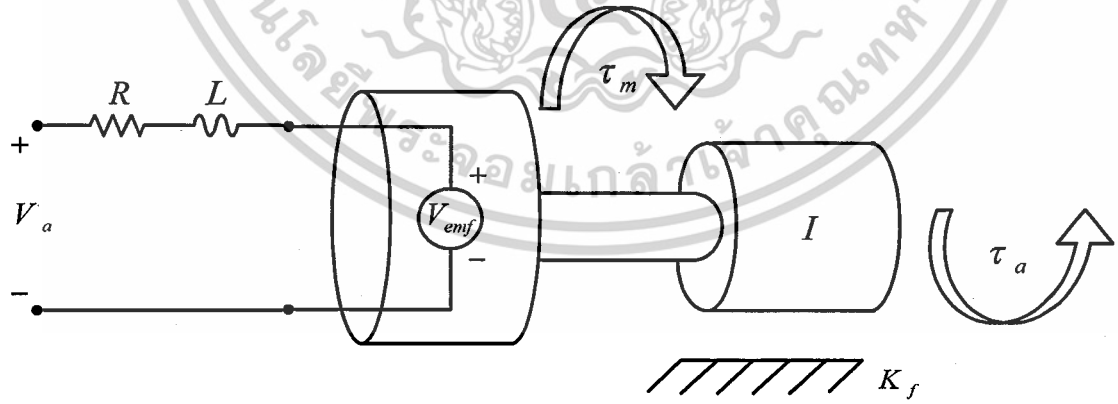
จากที่กล่าวมาแล้วในบทที่ 1 ก่อนเริ่มจัดทำโครงงานต้องศึกษาทฤษฎีและความรู้พื้นฐานในปริญญานิพนธ์เรื่องระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัมฉบับนี้ ได้รวบรวมเอาหลักการและทฤษฎีที่มีความสำคัญต่อการจัดทำโครงงาน เช่น แบบจำลองทางคณิตศาสตร์ของหุ่นยนต์ ทฤษฎีของการออกแบบตัวควบคุม และหลักการทำงานของอุปกรณ์ต่างๆ ดังต่อไปนี้

2.1 แบบจำลองทางคณิตศาสตร์ของหุ่นยนต์อินเวิร์ทเพนดูลัม

ในโครงงานเรื่องระบบควบคุมอินเวิร์ทเพนดูลัม ระบบพลวัตของหุ่นยนต์อินเวิร์ทเพนดูลัมสามารถอธิบายด้วยแบบจำลองทางคณิตศาสตร์ ซึ่งสามารถนำมาพัฒนาในการออกแบบตัวควบคุมสมดุลให้กับหุ่นยนต์ ดังนั้นในหัวข้อนี้กล่าวถึง สมการการเคลื่อนที่ และแบบจำลองทางคณิตศาสตร์ของหุ่นยนต์อินเวิร์ทเพนดูลัม

2.1.1 แบบจำลองของมอเตอร์ไฟฟ้ากระแสตรง

ในหัวข้อนี้อธิบายถึงระบบพลวัตของมอเตอร์ไฟฟ้ากระแสตรงด้วยแบบจำลองปริภูมิสเตท โดยอธิบายจากความสัมพันธ์ของแรงดันอินพุทของมอเตอร์ไฟฟ้ากระแสตรง และแรงบิด (Torque) ที่ต้องการในการควบคุมสมดุลของหุ่นยนต์



รูปที่ 2.1 แบบจำลองโครงสร้างมอเตอร์ไฟฟ้ากระแสตรง

จากรูปที่ 2.1 เมื่อมอเตอร์ไฟฟ้ากระแสตรงได้รับแรงดันไฟฟ้าอินพุท $V_a(t)$ เข้ามา ผ่านตัวต้านทานและตัวเหนี่ยวนำที่อนุกรมอยู่ จะเหนี่ยวนำให้เกิดกระแสไฟฟ้า $i(t)$ ในขดลวดอาร์เมเจอร์

(Armature) และมอเตอร์จะส่งแรงบิดออกไปเป็นเอาท์พุท ซึ่งค่าแรงบิด $\tau_m(t)$ ของมอเตอร์เป็นสัดส่วนกับกระแสไฟฟ้า $i(t)$ โดยมีค่า k_m เป็นค่าคงที่ของแรงบิด

$$\tau_m(t) = k_m i(t) \tag{2.1.1}$$

เมื่อมีแรงบิด คอยล์ของมอเตอร์จะเคลื่อนที่ผ่านสนามแม่เหล็กด้วยความเร็วเชิงมุม $\omega(t)$ ทำให้เกิดแรงเคลื่อนกระแสไฟฟ้าต้านกลับขึ้น (Back electromotive force voltage, $V_{emf}(t)$) โดยมีค่า k_e เป็นค่าคงที่ของแรงดันไฟฟ้าต้านกลับ ซึ่งสามารถอธิบายด้วยสมการที่ (2.1.2)

$$V_{emf}(t) = k_e \omega(t) \tag{2.1.2}$$

จากกฎแรงดันไฟฟ้าของเคอร์ชอฟฟ์ (Kirchoff's voltage law) สามารถสร้างสมการอนุพันธ์เชิงเส้น (Linear differential) ของมอเตอร์ ได้ดังนี้

$$V_a(t) - Ri(t) - L\frac{di(t)}{dt} - V_{emf}(t) = 0 \tag{2.1.3}$$

เมื่อ R และ L คือค่าตัวต้านทาน และค่าตัวเหนี่ยวนำที่อนุกรมอยู่ในขดลวดอาร์เมเจอร์

จากสมการการเคลื่อนที่ของมอเตอร์ และจากกฎการเคลื่อนที่ของนิวตัน แรงเสียดทานของมอเตอร์สามารถจัดให้อยู่ในรูปสมการเชิงเส้นของความเร็ว เมื่อสัมประสิทธิ์แรงเสียดทานของแกนมอเตอร์คือ k_f และผลรวมของแรงบิด เท่ากับ ความเร่งที่แกนมอเตอร์ คูณกับค่าความเฉื่อยของโรเตอร์ $I_R(t)$ ซึ่งสามารถแสดงเป็นสมการได้ดังนี้

$$\sum M(t) = \tau_m(t) - k_f \omega(t) - \tau_a(t) = I_R \dot{\omega}(t) \tag{2.1.4}$$

จากสมการที่ (2.1.1) ถึง (2.1.4) สามารถจัดรูปสมการใหม่ และเขียนแทนด้วยสมการของมอเตอร์พื้นฐานสองสมการ คือ

$$\frac{di(t)}{dt} = -\frac{R}{L}i(t) - \frac{k_e}{L}\omega(t) + \frac{V_a(t)}{L} \tag{2.1.5}$$

$$\frac{d\omega(t)}{dt} = \frac{k_m}{I_R} i(t) - \frac{k_f}{I_R} \omega(t) - \frac{\tau_a(t)}{I_R} \tag{2.1.6}$$

เนื่องจากค่าความเหนี่ยวนำและแรงเสียดทานของมอเตอร์ โดยทั่วไปมีค่าน้อยมาก ดังนั้น หากประมาณให้ค่าทั้งสองเป็นศูนย์ จะได้สมการอย่างง่ายของมอเตอร์ไฟฟ้ากระแสตรง ดังสมการที่ (2.1.7) และ (2.1.8)

$$i(t) = -\frac{k_e}{R} \omega(t) + \frac{1}{R} V_a(t) \tag{2.1.7}$$

$$\frac{d\omega(t)}{dt} = \frac{k_m}{I_R} i(t) - \frac{\tau_a(t)}{I_R} \tag{2.1.8}$$

และจากความสัมพันธ์ของสมการที่ (2.1.7) และ (2.1.8) สามารถเขียนสมการใหม่ โดยแทนตัวแปรกระแสไฟฟ้าในสมการที่ (2.1.7) ลงในสมการที่ (2.1.8) จะได้สมการใหม่ดังนี้

$$\frac{d\omega(t)}{dt} = -\frac{k_m k_e}{I_R R} \omega(t) + \frac{k_m}{I_R R} V_a(t) - \frac{\tau_a(t)}{I_R} \tag{2.1.9}$$

ระบบพลวัตของมอเตอร์สามารถแสดงเป็นแบบจำลองปริภูมิสเตทได้ด้วยตัวแปรตำแหน่งเชิงมุม $\theta(t)$ และตัวแปรความเร็วเชิงมุม $\omega(t)$ เมื่ออินพุทของมอเตอร์คือแรงดันไฟฟ้าและแรงบิด จะได้แบบจำลองปริภูมิสเตทของระบบพลวัตของมอเตอร์ดังนี้

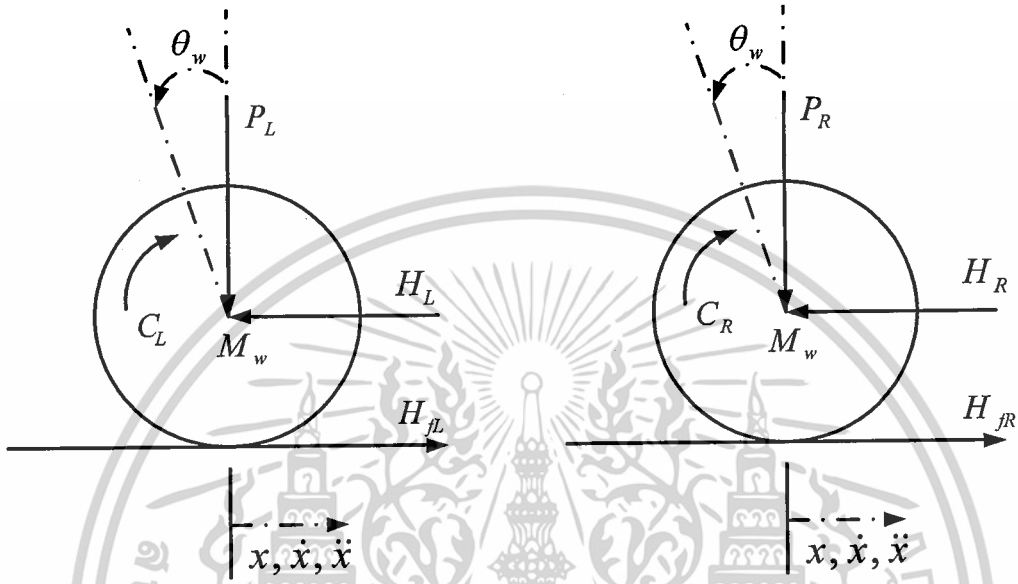
$$\begin{bmatrix} \dot{\theta}(t) \\ \dot{\omega}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \frac{-k_m k_e}{I_R R} \end{bmatrix} \begin{bmatrix} \theta(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{k_m}{I_R R} & \frac{-1}{I_R} \end{bmatrix} \begin{bmatrix} V_a(t) \\ \tau_a(t) \end{bmatrix} \tag{2.1.10}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} V_a(t) \\ \tau_a(t) \end{bmatrix} \tag{2.1.11}$$

2.1.2 แบบจำลองล้อทั้งสองของหุ่นยนต์อินเวิร์ทเพนดูลัม

หุ่นยนต์อินเวิร์ทเพนดูลัมแบบสองล้อ มีระบบเชิงกลที่ยุ่งยาก เนื่องจากต้องวิเคราะห์ระบบของล้อและก้านเพนดูลัมแยกกันในตอนแรก แต่สุดท้ายระบบทั้งสองต้องถูกนำมาใช้อธิบายพฤติกรรมการรักษาสมดุลของหุ่นยนต์อินเวิร์ทเพนดูลัมร่วมกัน

เมื่อหุ่นยนต์ถูกทำให้เบี่ยงเบนออกจากสมดุล แรงบิดจะถูกส่งออกจากมอเตอร์ แบบจำลองทางคณิตศาสตร์จึงปรับเปลี่ยนไปตามแรงที่เกิดขึ้น เริ่มต้นจากสมการการเคลื่อนที่ของล้อซ้าย และล้อขวา ซึ่งได้มาจากการเขียนแผนภาพแรงของล้อทั้งสอง จากนั้นจะได้สมการสำเร็จของล้อทั้งสองที่เหมือนกัน โดยในที่นี้จะแสดงเฉพาะสมการจากล้อขวา



รูปที่ 2.2 แผนภาพแรงของล้อทั้งสอง

จากกฎการเคลื่อนที่ของนิวตัน ผลรวมของแรงในแนวนอน อธิบายได้ดังสมการที่ (2.1.12) เมื่อ M_w คือ มวลของล้อและหุ่นยนต์ $\ddot{x}(t)$ คือ ความเร่งในแนวนอน H_{fR} คือ แรงเสียดทานระหว่างพื้นกับล้อ และ H_R คือ แรงปฏิกิริยาระหว่างล้อกับหุ่นยนต์ในแนวนอน

$$\sum F(t) = Ma(t)$$

$$M_w\ddot{x}(t) = H_{fR}(t) - H_R(t) \tag{2.1.12}$$

และผลรวมของแรงที่กระทำที่จุดศูนย์กลางของล้อ อธิบายได้โดยสมการที่ (2.1.13) เมื่อ I_w คือ โมเมนต์ความเฉื่อยของล้อ θ_w คือ มุมที่หมุนของล้อ C_R คือ แรงบิดของมอเตอร์ที่กระทำต่อล้อ และ r คือ รัศมีของล้อ

$$\sum M_0(t) = I\alpha(t)$$

$$I_w\ddot{\theta}_w(t) = C_R(t) - rH_{fR}(t) \tag{2.1.13}$$

จากสมการพลวัตของมอเตอร์ในสมการที่ (2.1.4) เมื่อพิจารณาที่ $k_f = 0$ แรงบิดของมอเตอร์สามารถแสดงได้ดังสมการที่ (2.1.14)

$$\tau_m(t) = I_R \frac{d\omega(t)}{dt} + \tau_a(t) \tag{2.1.14}$$

เมื่อจัดรูปสมการ และแทนค่าพารามิเตอร์จากมอเตอร์ไฟฟ้ากระแสตรง จะได้สมการของแรงบิดเอาท์พุทสุทธิที่ให้กับล้อ ดังนี้

$$C_R(t) = I_R \frac{d\omega(t)}{dt} = -\frac{k_m k_e}{R}\dot{\theta}_w(t) + \frac{k_m}{R}V_a(t) \tag{2.1.15}$$

เพราะฉะนั้นสามารถเขียนสมการ (2.1.13) ใหม่ได้ดังนี้

$$I_w\ddot{\theta}_w(t) = -\frac{k_m k_e}{R}\dot{\theta}_w(t) + \frac{k_m}{R}V_a(t) - rH_{fR}(t) \tag{2.1.16}$$

ดังนั้นจะได้

$$H_{fR}(t) = -\frac{k_m k_e}{Rr}\dot{\theta}_w(t) + \frac{k_m}{Rr}V_a(t) - \frac{I_w}{r}\ddot{\theta}_w(t) \tag{2.1.17}$$

เมื่อนำสมการที่ (2.1.15) แทนในสมการที่ (2.1.12) จะได้สมการสำหรับล้อทางซ้าย และขวาดังนี้

สมการล้อทางซ้าย

$$M_w\ddot{x}(t) = -\frac{k_m k_e}{Rr}\dot{\theta}_w(t) + \frac{k_m}{Rr}V_a(t) - \frac{I_w}{r}\ddot{\theta}_w(t) - H_L(t) \tag{2.1.18}$$

สมการล้อทางขวา

$$M_w\ddot{x}(t) = -\frac{k_m k_e}{Rr}\dot{\theta}_w(t) + \frac{k_m}{Rr}V_a(t) - \frac{I_w}{r}\ddot{\theta}_w(t) - H_R(t) \tag{2.1.19}$$

เนื่องจากการเคลื่อนที่เกิดขึ้นที่จุดศูนย์กลางของล้อ การเคลื่อนที่เชิงมุม สามารถแปลงเป็นการเคลื่อนที่เชิงเส้นได้ ดังนี้

$$\ddot{\theta}_w(t)r = \ddot{x}(t) \quad \text{หรือ} \quad \ddot{\theta}_w(t) = \frac{\ddot{x}(t)}{r}$$

$$\dot{\theta}_w(t)r = \dot{x}(t) \quad \text{หรือ} \quad \dot{\theta}_w(t) = \frac{\dot{x}(t)}{r}$$

ดังนั้นสมการที่ (2.1.18) และ (2.1.19) สามารถแปลงเป็นสมการการเคลื่อนที่เชิงเส้นได้ ดังนี้

สมการล้อทางซ้าย

$$M_w\ddot{x}(t) = -\frac{k_m k_e}{Rr^2}\dot{x}(t) + \frac{k_m}{Rr}V_a(t) - \frac{I_w}{r^2}\ddot{x}(t) - H_L(t) \tag{2.1.20}$$

สมการล้อทางขวา

$$M_w\ddot{x}(t) = -\frac{k_m k_e}{Rr^2}\dot{x}(t) + \frac{k_m}{Rr}V_a(t) - \frac{I_w}{r^2}\ddot{x}(t) - H_R(t) \tag{2.1.21}$$

เมื่อนำสมการที่ (2.1.20) และ (2.1.21) มารวมกัน จะได้เป็นสมการรวมของล้อทั้งสอง ดังนี้

$$2\left(M_w + \frac{I_w}{r^2}\right)\ddot{x}(t) = \frac{-2k_m k_e}{Rr^2}\dot{x}(t) + \frac{2k_m}{Rr}V_a(t) - \left(H_L(t) + H_R(t)\right) \tag{2.1.22}$$

2.1.3 แบบจำลองของก้านอินเวิร์ทเพนดูลัม

ในการวิเคราะห์ระบบเชิงกลของก้านอินเวิร์ทเพนดูลัม มีความต่อเนื่องมาจากการวิเคราะห์ระบบเชิงกลของล้อ โดยสามารถแสดงแผนภาพแรงของก้านอินเวิร์ทเพนดูลัมได้ดังรูป 2.3

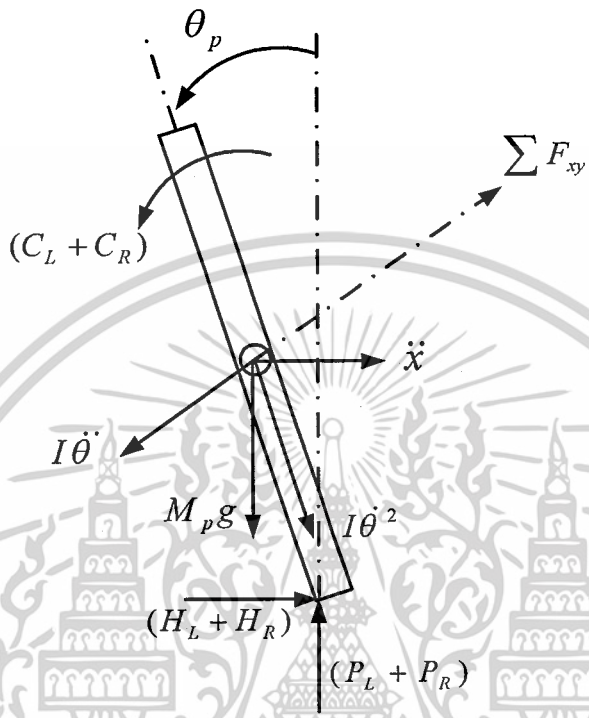
จากกฎการเคลื่อนที่ของนิวตัน ผลรวมของแรงในแนวนอน อธิบายได้ดังสมการที่ (2.1.23) โดย M_p คือมวลของก้านเพนดูลัม l คือระยะระหว่างจุดศูนย์กลางล้อถึงจุดศูนย์ถ่วงของก้านเพนดูลัม θ_p คือมุมที่ก้านเพนดูลัมเบี่ยงเบนจากแกนสมดุล และ

$$\sum F_x(t) = M_p\ddot{x}(t)$$

$$\left(H_L(t) - H_R(t)\right) - M_p l\ddot{\theta}_p(t)\cos\theta_p(t) + M_p l\dot{\theta}_p^2(t)\sin\theta_p(t) = M_p\ddot{x}(t) \tag{2.1.23}$$

ดังนั้นจะได้

$$\left(H_L(t)-H_R(t)\right)=M_p\ddot{x}(t)+M_pl\ddot{\theta}_p(t)\cos\theta_p(t)-M_pl\dot{\theta}_p^2(t)\sin\theta_p(t) \quad (2.1.24)$$



รูปที่ 2.3 แผนภาพแรงของก้านอินเวิร์ทเพนดูลัม

และผลรวมแรงในแนวตั้งฉากของก้านเพนดูลัม อธิบายได้ดังสมการที่ (2.1.25) เมื่อ P_L, P_R คือ แรงปฏิกิริยาระหว่างล้อและหุ่นยนต์ในแนวตั้ง

$$\sum F_{xp}(t)=M_p\ddot{x}(t)\cos\theta_p(t)$$

$$\left(H_L(t)+H_R(t)\right)\cos\theta_p(t)+\left(P_L(t)+P_R(t)\right)\sin\theta_p(t) - M_pg\sin\theta_p(t)-M_pl\ddot{\theta}_p(t) = M_p\ddot{x}(t)\cos\theta_p(t) \quad (2.1.25)$$

ผลรวมของโมเมนต์ที่จุดศูนย์กลางมวลของก้านเพนดูลัม คือ

$$\sum M_o(t)=I\alpha(t)$$

$$-\left(H_L(t)+H_R(t)\right)l\cos\theta_p(t)-\left(P_L(t)+P_R(t)\right)l\sin\theta_p(t)$$
$$-\left(C_L(t)+C_R(t)\right)=I_p\ddot{\theta}_p(t) \qquad (2.1.26)$$

จากสมการที่ (2.1.15) แรงบิดจากมอเตอร์ที่ใช้ควบคุมสมดุลของก้านเพนดูลัม เมื่อผ่านการแปลงเชิงเส้น (Linear transformation) หรือแผนภาพเชิงเส้น (Linear map) จะได้สมการดังนี้

$$\left(C_L(t)+C_R(t)\right)=\frac{-2k_mk_e}{R}\frac{\dot{x}(t)}{r}+\frac{2k_m}{R}V_a(t)$$

ซึ่งสามารถแทนค่าในสมการที่ (2.1.26) ได้ดังนี้

$$-\left(H_L(t)+H_R(t)\right)l\cos\theta_p(t)-\left(P_L(t)+P_R(t)\right)l\sin\theta_p(t)$$
$$-\left(\frac{-2k_mk_e}{R}\frac{\dot{x}(t)}{r}+\frac{2k_m}{R}V_a(t)\right)=I_p\ddot{\theta}_p(t)$$

ดังนั้นจะได้

$$-\left(H_L(t)+H_R(t)\right)l\cos\theta_p(t)-\left(P_L(t)+P_R(t)\right)l\sin\theta_p(t)$$
$$=I_p\ddot{\theta}_p(t)-\frac{2k_mk_e}{R}\frac{\dot{x}(t)}{r}+\frac{2k_m}{R}V_a(t) \qquad (2.1.27)$$

คูณสมการที่ (2.1.25) ด้วย $-l$

$$-\left(H_L(t)+H_R(t)\right)l\cos\theta_p(t)-\left(P_L(t)+P_R(t)\right)l\sin\theta_p(t)$$
$$+M_pgl\sin\theta_p(t)+M_pl^2\ddot{\theta}_p(t)=-M_pl\ddot{x}(t)\cos\theta_p(t) \qquad (2.1.28)$$

แทนค่าในสมการที่ (2.1.27) ในสมการที่ (2.1.28)

$$I_p\ddot{\theta}_p(t)-\frac{2k_mk_e}{Rr}\dot{x}(t)+\frac{2k_m}{R}V_a(t)+M_pl^2\ddot{\theta}_p(t)$$
$$+M_pgl\sin\theta_p(t)=-M_pl\ddot{x}(t)\cos\theta_p(t) \qquad (2.1.29)$$

เพื่อกำจัด $\left(H_R(t)+H_L(t)\right)$จากสมการของมอเตอร์ แทนสมการที่ (2.1.24) ในสมการที่ (2.1.22) จะได้

$$2\left(M_w+\frac{I_w}{r^2}\right)\ddot{x}(t)=-\frac{2k_mk_e}{Rr^2}\dot{x}(t)+\frac{2k_m}{Rr}V_a(t)-M_p\ddot{x}(t)$$
$$-M_pl\ddot{\theta}_p(t)\cos\theta_p(t)+M_pl\dot{\theta}_p^2(t)\sin\theta_p(t) \qquad (2.1.30)$$

จัดรูปสมการที่ (2.1.29) และ (2.1.30) จะได้สมการการเคลื่อนที่ที่ไม่เป็นเชิงเส้นของระบบดังนี้

$$\left(I_p+M_pl^2\right)\theta_p^2(t)-\frac{2k_mk_e}{Rr}\dot{x}(t)+\frac{2k_m}{R}V_a(t)$$
$$+M_pgl\sin\theta_p(t)=-M_pl\ddot{x}(t)\cos\theta_p(t) \qquad (2.1.31)$$

$$\frac{2k_m}{Rr}V_a(t)=\left(2M_w+\frac{2I_w}{r^2}+M_p\right)\ddot{x}(t)+\frac{2k_mk_e}{Rr^2}\dot{x}(t)$$
$$+M_pl\ddot{\theta}_p(t)\cos\theta_p(t)-M_pl\dot{\theta}_p(t)\sin\theta_p(t) \qquad (2.1.32)$$

จากสมการที่ (2.1.31) และ (2.1.32) ข้างต้น สามารถทำการประมาณเชิงเส้น (Linearise) เพื่อให้ได้แบบจำลองปริภูมิสเตทเชิงเส้น เมื่อพิจารณา $\theta_p=\pi+\phi$ โดย ϕ คือ มุมขนาดเล็กที่เบี่ยงเบนออกจากแนวตั้งฉาก สามารถประมาณ

$$\cos\theta_p(t)\cong-1,\ \sin\theta_p(t)\cong-\phi(t) \quad \text{และ} \quad \left(\frac{d\theta_p(t)}{dt}\right)^2\cong 0$$

ดังนั้น สมการการเคลื่อนที่ซึ่งได้จากการประมาณเชิงเส้น อธิบายได้โดย

$$\left(I_p+M_pl^2\right)\ddot{\phi}(t)-\frac{2k_mk_e}{Rr}\dot{x}(t)+\frac{2k_m}{R}V_a(t)-M_pgl\phi(t)=M_pl\ddot{x}(t) \qquad (2.1.33)$$

$$\frac{2k_m}{Rr}V_a(t)=\left(2M_w+\frac{2I_w}{r^2}+M_p\right)\ddot{x}(t)+\frac{2k_mk_e}{Rr^2}\dot{x}(t)-M_pl\ddot{\phi}(t) \qquad (2.1.34)$$

จัดรูปสมการที่ (2.1.33) และ (2.1.34) ให้อยู่ในรูปของตัวแปรสเตทจะได้สมการดังนี้

$$\ddot{\phi}(t)=\frac{M_pl}{\left(I_p+M_pl^2\right)}\ddot{x}(t)+\frac{2k_mk_r}{Rr\left(I_p+M_pl^2\right)}\dot{x}(t)$$
$$-\frac{2k_m}{R\left(I_p+M_pl^2\right)}V_a(t)+\frac{M_pgl}{\left(I_p+M_pl^2\right)}\phi(t) \qquad (2.1.35)$$

$$\ddot{x}(t)=\frac{2k_m}{Rr\left(2M_w+\frac{2I_w}{r^2}+M_p\right)}V_a(t)-\frac{2k_mk_e}{Rr^2\left(2M_w+\frac{2I_w}{r^2}+M_p\right)}\dot{x}(t)$$
$$+\frac{M_pl}{\left(2M_w+\frac{2I_w}{r^2}+M_p\right)}\ddot{\phi}(t) \qquad (2.1.36)$$

เมื่อแทนสมการที่ (2.1.35) ในสมการที่ (2.1.34) และสมการที่ (2.1.36) ในสมการที่ (2.1.33) และจัดรูปสมการ จะได้สมการปริภูมิสเตทของระบบดังนี้

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\phi}\\ \ddot{\phi}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{2k_mk_e(M_plr-I_p-M_pl^2)}{Rr^2\alpha} & \frac{M_p^2gl^2}{\alpha} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{2k_mk_e(r\beta-M_pl)}{Rr^2\alpha} & \frac{M_pgl\beta}{\alpha} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ \phi\\ \dot{\phi}\end{bmatrix}+\begin{bmatrix}0\\ \frac{2k_m(I_p+M_pl^2-M_plr)}{Rr\alpha}\\ 0\\ \frac{2k_m(M_pl-r\beta)}{Rr\alpha}\end{bmatrix}V_a$$

เมื่อกำหนดให้

$$\beta=\left(2M_w+\frac{2I_w}{r^2}+M_p\right) \qquad \alpha=\left[I_p\beta+2M_pl^2\left(M_w+\frac{I_w}{r^2}\right)\right]$$

จากแบบจำลองปริภูมิสเตทข้างต้น กำหนดให้ล้อของหุ่นยนต์สัมผัสกับพื้นตลอดเวลา ไม่มีการเลื่อนไถลของล้อ และไม่มีแรงเสียดทานเชิงมุมของล้อ

2.2 ทฤษฎีการออกแบบตัวควบคุม

เนื่องจากระบบอินเวิร์ทเพนดูลัมเป็นระบบที่ไม่เป็นเชิงเส้น และไม่มีเสถียรภาพ การศึกษาทฤษฎีทางระบบควบคุมที่นำมาใช้ในการควบคุมเสถียรภาพให้แก่ระบบจึงสำคัญอย่างมาก ดังนั้นในหัวข้อนี้กล่าวถึงทฤษฎีสำคัญที่ช่วยในการออกแบบตัวควบคุมให้กับระบบ ดังนี้

2.2.1 ความควบคุมได้

ในกรณีทั่วไป สำหรับเมตริกซ์ถ่ายโอนใดๆ $H(s)$ พบว่าสามารถมีตัวแปรปริภูมิสเตทได้เป็นจำนวนไม่จำกัด และอาจเป็นไปได้ว่าจำนวนของสเตท $x(t)$ ในปริภูมิสเตทอาจมีจำนวนมากกว่าอันดับของระบบพลวัตในเมตริกซ์ถ่ายโอน ที่เป็นการส่งจากอินพุทไปยังเอาท์พุท ดังนั้นสิ่ง

ที่เกิดขึ้น คือสเตทบางตัวในระบบไม่สามารถที่จะควบคุมได้ด้วยอินพุท คุณสมบัติของระบบพลวัตที่มีความสำคัญต่อการออกแบบระบบควบคุม คือความควบคุมได้ (Controllability)

นิยาม ความควบคุมได้ของสเตท

ระบบพลวัต $\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t)$ หรือนิยมเขียนโดยย่อเฉพาะคู่เมตริกซ์ (A, B) ซึ่งเรียกว่าสามารถควบคุมสเตทได้ (State controllable) เมื่อเลือกสเตทเริ่มต้น $\mathbf{x}(t_0) = \mathbf{x}_0$ และสเตทสุดท้าย $\mathbf{x}(t_1) = \mathbf{x}_1$ ใดๆโดย $t_1 > t_0$ สามารถหาอินพุท $u(t)$ ที่ขับสเตทจาก $\mathbf{x}_0$ ไปสู่ $\mathbf{x}_1$ ได้ แต่ถ้าไม่สามารถหาอินพุทดังกล่าวได้ ระบบจะเรียกว่า ไม่สามารถควบคุมสเตทได้ (State uncontrollable) โดยสามารถตรวจสอบความควบคุมได้ของสเตทโดยทฤษฎีดังนี้

ทฤษฎี (A, B) จะสามารถควบคุมได้ ถ้าเมตริกซ์ Q_c มีค่าลำดับชั้น (Rank) เท่ากับ n โดยที่ n คือค่าลำดับชั้นของเมตริกซ์ A

$$Q_c = [B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B] \tag{2.2.1}$$

สังเกตว่าเงื่อนไขของค่าลำดับชั้น จะเป็นจริงสำหรับค่า s ที่มิใช่ค่าเจาะจงของ A เมื่อศึกษาการป้อนกลับสเตทจะได้ว่า เมื่อ (A, B) สามารถควบคุมสเตทได้ คือสามารถเลือกตัวป้อนกลับสเตท K ที่จะเคลื่อนย้ายค่าเจาะจงของ $A - BK$ ไปยังตำแหน่งใดก็ได้ตามต้องการ และค่าเจาะจงดังกล่าวจะสามารถเลื่อนตำแหน่งได้ โดยการเปลี่ยนค่าตัวป้อนกลับสเตท K ทำให้ (A, B) เรียกว่า สามารถทำให้เสถียรได้ (Stabilizable)

2.2.2 การวางตำแหน่งโพล

หลักการของการออกแบบโดย วิธีวางตำแหน่งโพล (Pole placement) คือการตั้งข้อกำหนดสำหรับตำแหน่งโพลทั้งหมดของระบบวงปิด และออกแบบตัวควบคุมที่จะได้ตำแหน่งโพลตามข้อกำหนดนั้น เงื่อนไขจำเป็นที่ทำให้สามารถเคลื่อนย้ายโพลทั้งหมดไปยังตำแหน่งที่ต้องการได้ คือ ระบบนั้นต้องสามารถควบคุมได้

ในการออกแบบทั่วไปจะมิได้ต้องการให้ระบบมีเสถียรภาพอย่างเดียว แต่จะต้องมีสมรรถนะ หรือผลตอบสนองตามต้องการด้วย ดังนั้นการกำหนดตำแหน่งของโพล ระบบวงปิดจึงมิใช่เพียงแต่ว่าต้องการให้โพลอยู่บนตำแหน่งด้านซ้ายของระนาบเชิงซ้อนเท่านั้น แต่อาจจะต้องอยู่ในพื้นที่ที่จะให้ผลตอบสนองที่ดี ตามต้องการด้วย

ในระบบอันดับ n ทั่วๆไป ความสัมพันธ์ของผลตอบสนองทางเวลาของระบบ และตำแหน่งของโพลมักมีความซับซ้อน จึงเป็นการยากที่จะกำหนดตำแหน่งของโพล เพื่อให้ได้ผลตอบสนองที่ดี ดังนั้นวิธีการออกแบบนี้โดยทั่วไปอาศัยหลักการของระบบที่มีลักษณะเด่นเป็นระบบอันดับสอง (Second – order dominant system) กล่าวคือโพลที่มีผลกระทบต่อผลตอบสนอง

มากที่สุดจะเป็นอันดับสอง เหตุผลคือในระบบอันดับสอง สามารถคำนวณหาความสัมพันธ์ระหว่างคุณสมบัติในโดเมนเวลากับตำแหน่งของโพลได้โดยง่าย

สำหรับวิธีการวางโพล สามารถทำได้ โดยให้ $K = [k_1 \quad k_2 \quad ..... \quad k_n]$ เป็นอัตราขยายสัญญาณป้อนกลับจากสเตท และ $k_i,\ i = 1,2,...,n$ เป็นอัตราขยายที่ต้องทำการออกแบบดังนั้นอินพุทที่ป้อนจะเท่ากับ

$$u(t) = -K\mathbf{x}(t) \tag{2.2.2}$$

เมื่อ $u(t)$ คือ สัญญาณควบคุม $\mathbf{x}(t)$ คือ ตัวแปรสเตทของระบบ และ K คือ อัตราขยายสัญญาณป้อนกลับของระบบ

เมื่อใช้สัญญาณควบคุมนี้จะได้สมการปริภูมิสเตทของระบบป้อนกลับเป็น

$$\dot{\mathbf{x}}(t) = (A - BK)\mathbf{x}(t) \tag{2.2.3}$$

ซึ่งสามารถคำนวณหาโพลได้จาก

$$\det(sI - A + BK) = 0 \tag{2.2.4}$$

ในขณะเดียวกัน หากต้องการออกแบบโดยกำหนดโพลของระบบวงปิดที่ตำแหน่ง $p_1, p_2, ..., p_n$ จะได้สมการคุณลักษณะ (Characteristic equation) เท่ากับ

$$\alpha(s) = (s - p_1)(s - p_2)...(s - p_n) = 0 \tag{2.2.5}$$

ดังนั้นสามารถคำนวณหาอัตราขยาย K ได้จากการเปรียบเทียบสมการ (2.2.4) และ (2.2.5)

2.2.3 ตัวคุมค่ากำลังสองเชิงเส้น

ในหัวข้อที่ 2.2.2 ได้ศึกษาการออกแบบตัวควบคุมป้อนกลับสเตทโดยวิธีการวางตำแหน่งโพล ซึ่งเป็นเครื่องมือออกแบบที่มีประโยชน์สำหรับระบบที่มีอันดับไม่สูงมาก หรือระบบที่มีลักษณะเด่นเป็นระบบพลวัตอันดับสอง เพราะทราบความสัมพันธ์ระหว่างตำแหน่งของโพล หรือค่าเจาะจงกับผลตอบสนองของระบบ เช่น ช่วงเวลาขึ้น (Rise time) หรือ การพุ่งเกิน (Overshoot)

แต่สำหรับระบบที่มีอันดับสูง ความสัมพันธ์ดังกล่าวจะได้ไม่ชัดเจน ข้อจำกัดอีกประการของวิธีการวางโพล คือ ระดับความเป็นอิสระ (Degree of freedom) ของปัญหา

ดังนั้นจึงมีแนวคิดที่เป็นประโยชน์ในการแก้ไขปัญหาดังกล่าวคือ การเลือกตัวควบคุมป้อนกลับด้วยวิธีหาค่าเหมาะที่สุด โดยมีตัวอย่าง ตัววัดคุณภาพของระบบดังกล่าว คือ ผลตอบสนองพลวัต การขจัดสัญญาณรบกวน ความไว หรือความคงทน เป็นต้น ซึ่งได้พยายามพัฒนาวิธีการควบคุมเหมาะสมที่สุดในหลายรูปแบบ เพื่อออกแบบระบบควบคุมป้อนกลับให้สมบูรณ์และให้ระบบเป็นไปตามต้องการ โดยการใช้การหาค่าเหมาะสมที่สุดเป็นเครื่องมือในการช่วยหาผลลัพธ์ให้เป็นที่น่าพอใจได้

ในหัวข้อนี้จะศึกษาเทคนิคในการเลือกอัตราขยายของตัวป้อนกลับสเตทที่เหมาะสมที่สุด (Optimal control) เรียกว่า ตัวคุมค่ากำลังสองเชิงเส้น (Linear Quadratic Regulator: LQR) ซึ่งเป็นวิธีการออกแบบที่ดีสำหรับการควบคุมป้อนกลับสเตท โดยเป็นวิธีการออกแบบพื้นฐานสำหรับระบบหลายอินพุทหลายเอาท์พุท (Multiple Input – Multiple Output: MIMO) อีกด้วย

โดยวิธีตัวคุมค่ากำลังสองเชิงเส้นเป็นวิธีการออกแบบระบบควบคุมที่เหมาะสม ที่ถูกนำมาใช้กับระบบพลวัต วิธีนี้สามารถลดความผิดพลาด หรือความเสียหายของระบบโดยการให้น้ำหนักความสำคัญของตัวแปร (Weighting factor) ที่มีผลต่อระบบ ตามความเหมาะสม ด้วยเมตริกซ์ Q และเมตริกซ์ R

การออกแบบตัวคุมค่ากำลังสองเชิงเส้น สำหรับระบบพลวัตในสมการที่ (2.2.6) ภายใต้เงื่อนไขเริ่มต้นในสมการที่ (2.2.7)

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + Bu(t) \tag{2.2.6}$$

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{2.2.7}$$

เมื่อ A คือเมตริกซ์ระบบ B คือเมตริกซ์ของอินพุท และ $\mathbf{x}_0$ เป็นสเตทเริ่มต้น โดยจะพิจารณาการควบคุมที่ลดค่าตัววัดคุณภาพ J ในสมการที่ (2.2.8) ให้น้อยที่สุด

$$J = \frac{1}{2}\int_0^{\infty}[\mathbf{x}^T(t)Qx(t) + u^T(t)Ru(t)]dt \tag{2.2.8}$$

ซึ่งเห็นว่าตัววัดคุณภาพ J ประกอบด้วยสองพจน์ คือ ตัวแทนขนาดของสเตท $\mathbf{x}(t)$ กับขนาดของตัวแปรควบคุม $u(t)$ ที่ถูกให้น้ำหนักโดยเมตริกซ์ Q และ R

หากอาศัยการป้อนกลับสเตทดังสมการที่ (2.2.9)

$$u(t) = -K\mathbf{x}(t) \tag{2.2.9}$$

ค่า K ที่ลดค่าตัววัดคุณภาพ J ให้น้อยที่สุดตามการออกแบบตัวคุมค่ากำลังสองเชิงเส้น สามารถหาได้โดยการแก้สมการริคัทติ (Ricatti equation) เพื่อหา P ดังนี้

$$PA + A^T P - PBR^{-1}B^T P + Q = 0 \tag{2.2.10}$$

โดย P เป็นโพสิทีฟเดฟฟีนิตเมตริกซ์ (Positive definite matrix) เมื่อได้ค่า P จะสามารถกำหนดอัตราขยายป้อนกลับ $K = R^{-1}B^T P$

จากสมการที่ (2.2.8) จะสังเกตเห็นว่า เมตริกซ์ Q และเมตริกซ์ R ช่วยให้ระดับความสำคัญของตัวแปร $\mathbf{x}(t)$ และ $u(t)$ โดยขนาดของเมตริกซ์ Q เท่ากับขนาดของเมตริกซ์ระบบ และขนาดของเมตริกซ์ R เท่ากับขนาดของเมตริกซ์อินพุท ซึ่งทั้งเมตริกซ์ Q และเมตริกซ์ R เป็นโพสิทีฟเดฟีนิตเมตริกซ์

ในการออกแบบนิยมกำหนดให้เมตริกซ์ Q มีลักษณะดังนี้

$$Q = \begin{bmatrix} w & 0 & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & y & 0 \\ 0 & 0 & 0 & z \end{bmatrix}$$

ซึ่งจะนำมาใช้งานในการออกแบบระบบควบคุมอินเวิร์ทเพนดูลัม ในโครงงานนี้เช่นกัน

2.3 แคลแมนฟิลเตอร์

ในปี 1960 ได้มีการตีพิมพ์ผลงานของรูดอฟ อีมิล แคลแมน (Rudolf Emil Kalman) ที่นำเสนอวิธีการใหม่ในการแก้ปัญหาของตัวกรองเชิงเส้น และการประมาณ (Linear filtering and prediction problem) วิธีของแคลแมนฟิลเตอร์ (Kalman filter) เป็นการหาสเตทของระบบ โดยทั่วไปสิ่งที่ต้องการทราบเมื่อต้องวิเคราะห์ระบบก็คือ ณ เวลานั้นๆ ตัวแปรสเตทของระบบมีค่าเป็นอย่างไร และเปลี่ยนแปลงตามเวลาอย่างไร ซึ่งบ่อยครั้งในทางปฏิบัติ การหาค่าตัวแปรสเตทของระบบไม่ใช่เรื่องง่าย เพราะมีข้อจำกัดหลายปัจจัย เช่น ความไม่สมบูรณ์ของเซนเซอร์ที่ใช้งาน และความคลาดเคลื่อนในการวัด เป็นต้น แคลแมนฟิลเตอร์จึงถูกคิดค้นมาเพื่อแก้ปัญหาเหล่านี้ โดยมีการนำแคลแมนฟิลเตอร์มาใช้เป็นครั้งแรก ในการประมาณสเตทให้กับระบบนำร่องของยานอพอลโล (Apollo) ที่โคจรรอบโลก ซึ่งปัจจุบัน แคลแมนฟิลเตอร์ถูกนำมาใช้อย่างแพร่หลายมากขึ้น โดยเฉพาะอย่างยิ่งในศาสตร์ของการประสานข้อมูล (Data fusion) เพื่อใช้ประมวลข้อมูลจากเซนเซอร์หลายประเภทร่วมกัน ภายใต้สัญญาณรบกวน (Noise) จากหลายแหล่ง เพื่อนำมาใช้ในลักษณะเกื้อกูลซึ่งกันและกันเพื่อหาค่าประมาณของตัวแปรสเตทของระบบที่ดีที่สุด (Optimal)

สำหรับการควบคุมหุ่นยนต์ทรงตัวสองล้อ มีความจำเป็นอย่างมากที่จะต้องใช้เซนเซอร์ที่มีความถูกต้องและแม่นยำสูง ซึ่งมักมีราคาแพง จากปัญหาดังกล่าวทำให้เทคโนโลยีการประมวลผลเซนเซอร์ร่วมกันเพื่อเพิ่มความถูกต้องในการวัดของเซนเซอร์ถูกนำมาใช้ในโครงงานนี้

โครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ได้นำวิธีแคลแมนฟิลเตอร์ มาใช้ในการประมวลผลของเซนเซอร์สองตัวร่วมกัน ได้แก่ เซนเซอร์ไจโรสโคป และเซนเซอร์ความเร่ง ซึ่งจากการศึกษา และทดลองพบว่าเมื่อนำเอาแคลแมนฟิลเตอร์มาใช้ในการประเมินเซนเซอร์ร่วม กันทำให้ค่าที่วัดได้จากเซนเซอร์มีความถูกต้องมากขึ้น และช่วยให้หุ่นยนต์ทรงตัวได้ดีขึ้นด้วย

ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ใช้วิธีการดีสครีตแคลแมนฟิลเตอร์ (Discrete Kalman filter algorithm) ในการประมาณค่าตัวแปรสเตทให้กับระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม

<u>หลักการของการประมวลผลร่วมกันของเซนเซอร์โดยใช้แคลแมนฟิลเตอร์</u>

เมื่อพิจารณาแบบจำลองปริภูมิสเตทแบบดีสครีตของระบบได้ จะได้สมการสเตทดังสมการที่ (2.3.1)

$$\mathbf{x}(t_{i+1}) = F\mathbf{x}(t_i) + B\mathbf{u}(t_i) + G\mathbf{w}(t_i) \tag{2.3.1}$$

เมื่อ F คือ เมตริกซ์ระบบของตัวแปรสเตท $\mathbf{x}(t_i)$, B คือ เมตริกซ์ ของสัญญาณรับเข้า $\mathbf{u}(t_i)$ และ G คือ เมตริกซ์ ของสัญญาณรบกวนขาว (Gaussian white noise) $\mathbf{w}(t_i)$ และสมการการวัดของระบบ ที่เวลาดีสครีต สามารถแสดงดังนี้

$$\mathbf{z}(t_i) = H\mathbf{x}(t_i) + \mathbf{v}(t_i) \tag{2.3.2}$$

เมื่อ H คือ เมตริกซ์ของการวัด $\mathbf{x}(t_i)$ และ $\mathbf{v}(t_i)$ คือ เวกเตอร์สัญญาณรบกวนขาวของการวัด

จากสัญญาณรบกวน $\mathbf{w}(t_i)$ และ $\mathbf{v}(t_i)$ เป็นสัญญาณรบกวนขาว ที่มีสมมติฐานทางสถิติดังต่อไปนี้

$$\begin{aligned} E\{\mathbf{w}(t_i)\} &= 0 \\ E\{\mathbf{v}(t_i)\} &= 0 \end{aligned} \tag{2.3.3}$$

$$E\{\mathbf{w}(t_i)\mathbf{w}^T(t_j)\} = \begin{cases} S_w(t_i), & i = j \\ 0, & i \neq j \end{cases} \tag{2.3.4}$$

$$E\{\mathbf{v}(t_i)\mathbf{v}^T(t_j)\} = \begin{cases} S_v(t_i), & i = j \\ 0, & i \neq j \end{cases} \tag{2.3.5}$$

เมื่อ $E\{\cdot\}$ คือ ค่าความคาดหวัง (Expected value) และ $S_w(t_i), S_v(t_i)$ เป็นโพสิทีฟเดฟีนิตเมตริกซ์ โดยที่สมมติฐานว่าสัญญาณรบกวน $\mathbf{w}(t_i)$ และ $\mathbf{v}(t_i)$ ทั้งสองนี้เป็นอิสระต่อกัน

จากสมการสเตทของระบบในสมการที่ (2.3.1) สามารถหาสเตทเริ่มต้น $\mathbf{x}(t_0)$ ของระบบจริงได้ ก็ต่อเมื่อสเตทเริ่มต้นสามารถสุ่มค่าจากค่าจำเพาะของ $\mathbf{x}(t_0)$ ได้ อย่างไรก็ตามค่าจำเพาะนี้อาจจะไม่ถูกต้อง เพราะถูกจำลองด้วยเวกเตอร์การสุ่มของการกระจายแบบปกติ (Normal distributed) ดังนั้นจึงอธิบายสเตทเริ่มต้นของ $\mathbf{x}(t_0)$ ได้จาก $\hat{\mathbf{x}}_0$ และความแปรปรวน P_0

$$\hat{\mathbf{x}}_0 = E\{\mathbf{x}(t_0)\} \tag{2.3.6}$$

เมื่อ $\hat{\mathbf{x}}_0$ เป็นค่าประมาณโดยไม่มีค่าจากการวัดมาร่วมพิจารณา ดังนั้นย่อมมีความคลาดเคลื่อนเกิดขึ้น กำหนดให้ความคลาดเคลื่อนจากการประมาณ คือ $\mathbf{x}(t_0) - \hat{\mathbf{x}}_0$ ดังนั้นสมการความแปรปรวน P_0 เป็นดังนี้

$$P_0 = E\{[\mathbf{x}(t_0) - \hat{\mathbf{x}}_0][\mathbf{x}(t_0) - \hat{\mathbf{x}}_0]^T\} \tag{2.3.7}$$

เมื่อ P_0 เป็นโพสิทีฟเซมิเดฟีนิตเมตริกซ์ (Positive semi-definite matrix) และเป็นเมตริกซ์ที่สมมาตร จึงทำให้สามารถหาค่าความคาดหวัง ระหว่างสเตทจริง และสเตทที่ถูกประมาณขึ้นได้ด้วยค่าโควาเรียนซ์ (Covariance) จากสเตทจริงของตัวแปรสเตทแต่ละตัว

กำหนดให้ t_i^+ เป็นเวลาของการประมาณสเตทของระบบหลังการวัด และ t_i^- เป็นเวลาก่อนการประมาณสเตทของระบบ ดังนั้นสามารถเขียนสมการ (2.3.6) และ (2.3.7) ใหม่ได้ดังนี้

$$\hat{\mathbf{x}}(t_i^-) = F\hat{\mathbf{x}}(t_{i-1}^+) + B\mathbf{u}(t_{i-1}^+) \tag{2.3.8}$$

เมื่อ $\hat{\mathbf{x}}(t_i^-)$ คือ เมตริกซ์สเตทเริ่มต้นการประมาณของระบบ F คือ เมตริกซ์ระบบ $\hat{\mathbf{x}}(t_{i-1}^+)$ คือ เมตริกซ์สเตทเริ่มต้นหลังการประมาณของระบบ B คือ เมตริกซ์อินพุทของระบบ

$$P(t_i^-) = FP(t_{i-1}^+)F^T + S_w(t_i) \tag{2.3.9}$$

เมื่อ $P(t_i^-)$ คือ เมตริกซ์โควาเรียนซ์ของการประมาณสเตท F คือเมตริกซ์ระบบ $P(t_{i-1}^+)$ คือ เมตริกซ์โควาเรียนซ์หลังการประมาณสเตท และ $S_w(t_i)$ คือเมตริกซ์โควาเรียนซ์ของสัญญาณรบกวนขาวของระบบ $\mathbf{w}(t_i)$

จากสมการที่ (2.3.9) เมตริกซ์โควาเรียนซ์ของการประมาณ $S_w(t_i)$ กำหนดให้ $\mathbf{z}_i(t_i)$ เป็นเวกเตอร์สเตทของการวัด โดยการประมาณสเตทจะถูกปรับปรุงด้วยแคลแมนฟิลเตอร์ ซึ่งสามารถแสดงเป็นสมการได้ 3 สมการดังนี้

$$k(t_i) = P(t_i^-)H^T[HP(t_i^-)H^T + S_v(t_i)]^{-1} \tag{2.3.10}$$

$$\hat{\mathbf{x}}(t_i^+) = \hat{\mathbf{x}}(t_i^-) + k(t_i)[\mathbf{z}_i(t_i) - H\hat{\mathbf{x}}(t_i^-)] \tag{2.3.11}$$

$$P(t_i^+) = P(t_i^-) - k(t_i)HP(t_i^-) \tag{2.3.12}$$

เมื่อ $k(t_i)$ คือ อัตราขยายของแคลแมนฟิลเตอร์ H คือ เมตริกซ์ระบบของการวัด $S_v(t_i)$ คือ เมตริกซ์โควาเรียนซ์ของ $\mathbf{v}(t_i)$ หรือสัญญาณรบกวนขาวของการวัด $\hat{\mathbf{x}}(t_i^+)$ คือ เวกเตอร์สเตทเริ่มต้นหลังประมาณของระบบ และ $P(t_i^+), P(t_{i-1}^+)$ คือ เมตริกซ์โควาเรียนซ์หลังผ่านการประมาณสเตท

ถ้าค่าความผิดพลาดจากการประมาณของตัวประมาณ $\hat{\mathbf{x}}(t_i^-)$ แสดงได้ดังสมการที่ (2.3.13)

$$E(t_i) = HP(t_i^-)H^T + S_v(t_i) \tag{2.3.13}$$

ดังนั้นจากสมการที่ (2.3.13) สามารถเขียนสมการที่ (2.3.10) ใหม่ได้ดังนี้

$$k(t_i) = P(t_i^-)H^T E^{-1}(t_i^-) \tag{2.3.14}$$

ค่าความแตกต่างระหว่างค่าประมาณการวัดที่ดีที่สุด กับค่าจากการวัดจริง $\mathbf{z}(t_i)$ เขียนแทนด้วย $\mathbf{r}(t_i)$ (Measurement residual) ซึ่งสามารถเขียนเป็นสมการได้ดังนี้

$$\mathbf{r}(t_i) = \mathbf{z}(t_i) - H(t_i)\hat{\mathbf{x}}(t_i^-) \tag{2.3.15}$$

โดยสมการแคลแมนฟิลเตอร์แบบต่อเนื่อง (Continuous Kalman filter algorithm) ของการประมวลผลเซนเซอร์ความเร่ง และเซนเซอร์ไจโรสโคป สามารถแสดงได้ดังสมการที่ (2.3.16) เมื่อ θ คือ มุมเบี่ยงเบน γ_m คือ ความเร็วที่ได้จากการวัดของไจโรสโคป และ β คือ ค่าความแตกต่างของความเร็วเชิงมุมที่วัดได้จากไจโรสโคปกับความเร็วเชิงมุมที่แท้จริง

$$\begin{bmatrix}\dot{\theta}(t)\\ \dot{\beta}(t)\end{bmatrix}=\begin{bmatrix}0 & -1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\theta(t)\\ \beta(t)\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\gamma_m(t)+\begin{bmatrix}1\\ 0\end{bmatrix}w(t) \tag{2.3.16}$$

และสมการเอาท์พุทการวัด สามารถแสดงได้ดังสมการที่ (2.3.17)

$$z(t)=[1 \quad 0]\begin{bmatrix}\theta(t)\\ \beta(t)\end{bmatrix}+v(t) \tag{2.3.17}$$

กำหนดให้ $t=t_i \cdot T_s$ เมื่อ T_s คือคาบการชักตัวอย่าง จะได้สมการของการประมวลผลเซนเซอร์ร่วมกัน เพื่อหาค่ามุมดังสมการที่ (2.3.18)

$$\theta(t_i \cdot T_s) \cong \frac{\theta(t_{i+1})-\theta(t_i)}{T_s} = -1 \cdot \beta(t_i)+\gamma_m(t_i) \tag{2.3.18}$$

ดังนั้นจะได้

$$\theta(t_{i+1})=\theta(t_i)-T_s\beta(t_i)+T_s\gamma_m \tag{2.3.19}$$

จากสมการที่ (2.3.16) และ (2.3.17) เมื่อแปลงสมการแคลแมนฟิลเตอร์แบบต่อเนื่อง เป็นสมการแคลแมนฟิลเตอร์แบบดีสครีต สามารถแสดงดังนี้

$$\begin{bmatrix}\theta(t_{i+1})\\ \beta(t_{i+1})\end{bmatrix}=\begin{bmatrix}1 & T_s\\ 0 & 0\end{bmatrix}\begin{bmatrix}\theta(t_i)\\ \beta(t_i)\end{bmatrix}+\begin{bmatrix}T_s\\ 0\end{bmatrix}\gamma_m(t_i) \tag{2.3.20}$$

$$z(t_i)=[1 \quad 0]\begin{bmatrix}\theta(t_i)\\ \beta(t_i)\end{bmatrix} \tag{2.3.21}$$

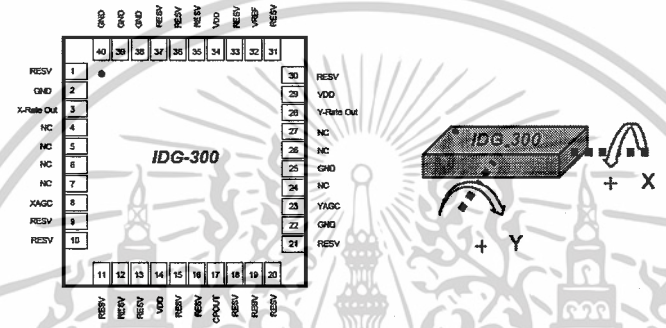
เมื่อสมการที่ (2.3.20) คือสมการระบบการประมวลผลเซนเซอร์ร่วมกันในสมการที่ (2.3.1) และสมการที่ (2.3.21) คือสมการเอาท์พุทการวัดในสมการที่ (2.3.2)

2.4 หลักการทำงานของเซนเซอร์

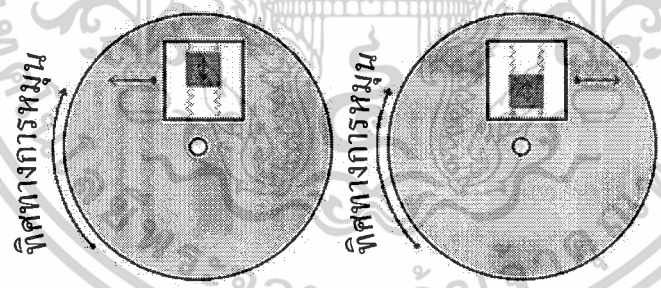
ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ได้ใช้เซนเซอร์ทั้งหมด 3 ชนิดได้แก่ เซนเซอร์ไจโรสโคป เซนเซอร์ความเร่ง และเอนโคดเดอร์ ซึ่งหัวข้อนี้จะได้กล่าวถึงหลักการทำงานของเซนเซอร์เหล่านี้

2.4.1 เซนเซอร์ไจโรสโคป

โครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ใช้เซนเซอร์ไจโรสโคปแบบสองแกน เบอร์ IDG300 ดังรูปที่ 2.4 ของบริษัทอินเวนเซน (Invensense) เป็นเซนเซอร์ประเภทเรทไจโร (Rate gyro) สามารถวัดความเร็วเชิงมุมได้สองแกน ให้สัญญาณเอาท์พุทเป็นสัญญาณอนาลอก โดยเมื่อไม่มีการเคลื่อนที่แบบหมุน หรืออัตราเร็วเชิงมุมเท่ากับศูนย์ สัญญาณเอาท์พุทที่ออกมาเป็นค่าความต่างศักย์มีค่าเท่ากับ 1.5 โวลต์ และจะเพิ่มขึ้นหรือลดลง 2 มิลลิโวลต์/องศา/วินาที ตามทิศทางการหมุน โครงสร้างภายในประกอบด้วยเฟรมหมุนที่ติดมวลขนาดเล็กไว้ ดังรูปที่ 2.5 เพื่อใช้วัดความเร่งคอริโอลิส (Coriolis acceleration)

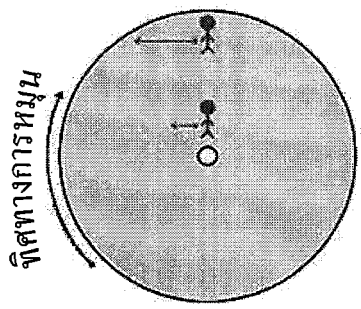


รูปที่ 2.4 เซนเซอร์ไจโรสโคปเบอร์ IDG300



รูปที่ 2.5 โครงสร้างภายในของไจโรสโคป

ความเร่งคอริโอลิสเป็นความเร่งที่เกิดกับวัตถุที่มีการเคลื่อนที่แบบหมุน สามารถอธิบายได้จากการจินตนาการถึงการเคลื่อนที่บนเฟรมหมุนของคนสองคนดังรูปที่ 2.6 เพื่อรักษาตำแหน่งเดิมของเขาไว้ เขาทั้งสองจะต้องเดินสวนทางกับทิศทางการหมุนของเฟรมด้วยความเร็วค่าหนึ่งซึ่งไม่เท่ากัน โดยคนที่อยู่ไกลจากจุดศูนย์กลางเฟรมเขาจะต้องใช้ความเร็วมากกว่าคนที่อยู่ใกล้ อัตราความเร็วที่เขาใช้เพิ่มขึ้นก็ของความเร่งคอริโอลิส



รูปที่ 2.6 ตัวอย่างแสดงความเร่งคอริโอลิส

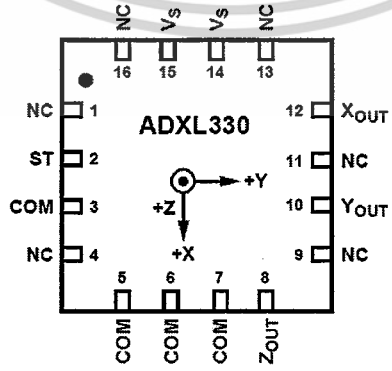
2.4.2 เซนเซอร์ความเร่ง

เซนเซอร์ ความเร่งใช้วัดความเร่งที่เกิดจากแรงภายนอกมากระทำ นอกจากนี้สามารถนำมาประยุกต์เพื่อหาเวกเตอร์ลัพธ์ของความเร่ง วัดความโน้มเอียง และความสั่นสะเทือน จากความสามารถที่หลากหลายดังกล่าวจึงนำมาประยุกต์เพื่อใช้หามุมที่เปลี่ยนแปลงได้

เซนเซอร์ความเร่งที่ใช้ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม คือเซนเซอร์เบอร์ ADXL330 ดังรูปที่ 2.7 ของบริษัทอนาลอกดีไวซ์ (Analog devices) มีความสามารถวัดได้สามแกน ให้สัญญาณเอาท์พุทเป็นสัญญาณอนาลอก โดยที่ขณะความเร่งเท่ากับศูนย์ให้เอาท์พุทมีค่าความต่างศักย์เท่ากับ 1.5โวลต์ และมีค่าเพิ่มหรือลดตามทิศทางความเร่งที่เปลี่ยนแปลงด้วยอัตรา 300 มิลลิโวลต์/จี การนำเซนเซอร์ความเร่งมาหามุมทำได้โดยใช้ความเร่งจากสองแกน ในโครงงานได้เลือกใช้แกน X และแกน Z โดยนำความเร่งจากที่วัดได้จากทั้งสองแกน นำมาแปลงด้วยสมการ

$$Angle = \tan^{-1}\frac{z}{x} \tag{2.4.1}$$

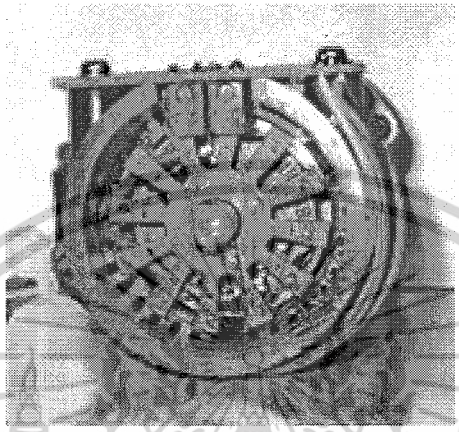
โดยที่ *Angle* คือค่าของมุม z คือความเร่งแกน Z และ x คือค่าความเร่งแกน X



รูปที่ 2.7 เซนเซอร์ความเร่งเบอร์ ADXL330

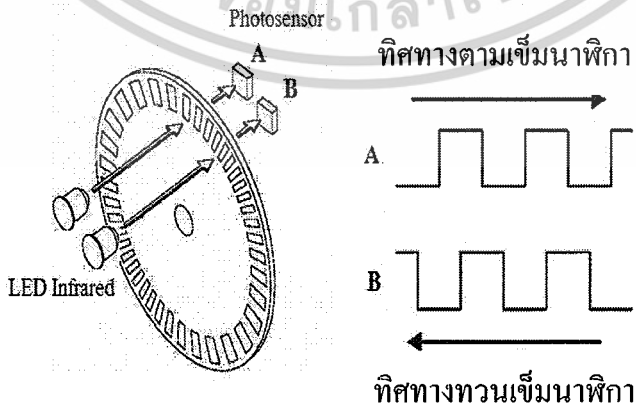
2.4.3 เอนโคดเดอร์

เอนโคดเดอร์เป็นอุปกรณ์ตรวจจับระยะทาง ที่ถูกใช้งานอย่างแพร่หลายในเครื่องจักรและมอเตอร์ทั่วไป เอนโคดเดอร์ที่นำมาใช้โครงงานเป็นเอนโคดเดอร์ประเภทอินครีเมนท์เอนโคดเดอร์ (Increment encoder) ดังรูปที่ 2.8 ซึ่งมีความละเอียด 768 พัลส์/รอบ (Pulse per revolution: ppr) โดยโครงสร้างประกอบด้วยตัวรับส่งสัญญาณอินฟราเรด (Infrared) แผ่นวงกลมที่มีช่องสลิต (Slit)



รูปที่ 2.8 อินครีเมนท์เอนโคดเดอร์

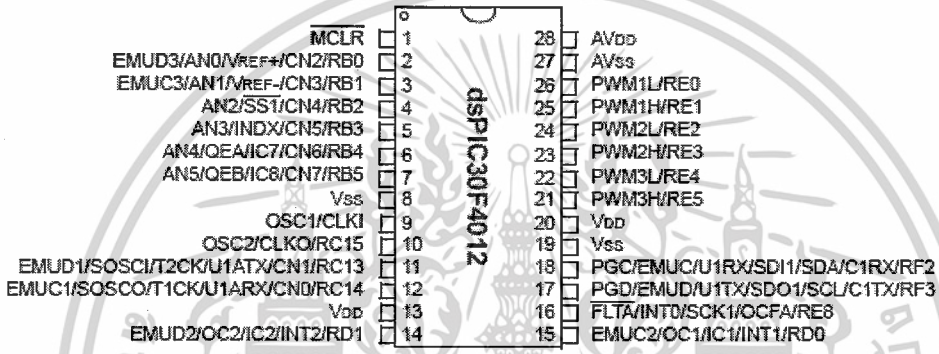
ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัมนี้ ได้นำเอนโคดเดอร์มาใช้หาระยะทางและความเร็วเพื่อเป็นค่าป้อนกลับให้กับตัวควบคุม โดยเอนโคดเดอร์ให้สัญญาณพัลส์ ออกมาตามค่าระยะทางที่มอเตอร์หมุน ซึ่งสามารถนำมาหาค่าระยะทาง และความเร็ว ของมอเตอร์ได้ สำหรับการหาทิศทางของการหมุนของมอเตอร์ทำได้โดยใช้เอนโคดเดอร์ที่มีสลิตชุดเดียวและมีชุดรับและส่งแสงอินฟาเรด 2 ชุด ดังรูปที่ 2.9 ทำให้ได้สัญญาณเอาท์พุทสองชุด ที่มีเฟสต่างกัน 90 องศา กำหนดให้เป็นชุด A และชุด B โดยทิศทางของการหมุนของมอเตอร์สามารถหาได้จากการนำหรือการตามของเฟสของสัญญาณเอาท์พุททั้งสอง



รูปที่ 2.9 โครงสร้างและการทำงานของเอนโคดเดอร์

2.5 ไมโครคอนโทรลเลอร์

ในโครงงานนี้มีความจำเป็นที่จะต้องใช้การประมวลผลแบบตามเวลาจริง (Real time) เพราะว่าเป็นการทำงานที่เกี่ยวกับสถานะในช่วงเวลาหนึ่งๆ เท่านั้น หากประมวลผลที่ล่าช้าเกินไปจะทำให้ผลตอบสนองต่อสถานะนั้นผิดพลาดไปจากที่ต้องการ จึงต้องใช้หน่วยประมวลผลที่มีความเร็วสูง ในโครงงานนี้เลือกใช้ไมโครคอนโทรลเลอร์เบอร์ dsPIC30F4012 ดังรูปที่ 2.10 ซึ่งเป็นหน่วยประมวลผลแบบ 16 บิต ในตระกูล dsPIC ของบริษัทไมโครชิพ (Microchip Technology Inc) และมีหน่วยเชื่อมต่อและหน่วยฟังก์ชันพิเศษเพียงพอกับตามความต้องการในโครงงานนี้



รูปที่ 2.10 ไมโครคอนโทรลเลอร์ dsPIC30F4012

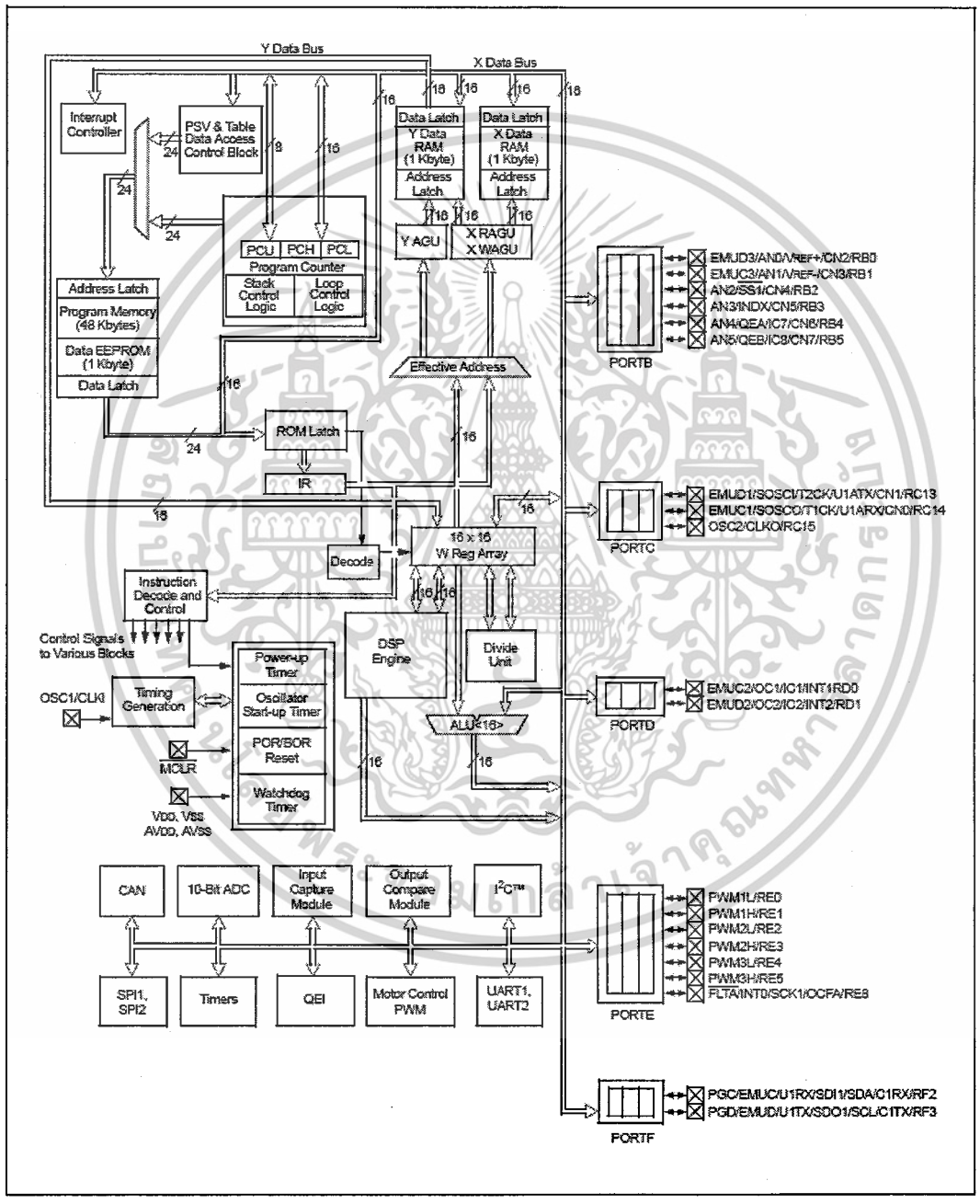
2.5.1 คุณสมบัติของ dsPIC30F4012

dsPIC30F4012 เป็นไมโครคอนโทรลเลอร์ ที่มีการประมวลผลข้อมูลแบบ 16 บิต ซึ่งมีจุดเด่นในด้านความสามารถในการประมวลผลข้อมูลสัญญาณแบบดิจิตอล สำหรับนำไปประยุกต์ใช้ในงานควบคุมต่างๆ โดยโครงสร้างภายในจะเป็นการผสมผสานระหว่างไมโครคอนโทรลเลอร์ และวงจรประมวลผลสัญญาณดิจิตอล (Digital Signal Processing) รวมเข้าไว้ด้วยกัน ดังรูปที่ 2.11 หรืออาจเรียกไมโครคอนโทรลเลอร์ตระกูล dsPIC ว่าเป็นตัวควบคุมแบบสัญญาณดิจิตอล (Digital Signal Controller: DSC)

<u>คุณสมบัติด้านการประมวลผล</u>

- ใช้สถาปัตยกรรมแบบ RISC โดยมี 83 คำสั่งมาตรฐาน รองรับการอ้างตำแหน่งแอดเดรสแบบต่างๆได้โดยอิสระ โดยรูปแบบโครงสร้างการจัดผังหน่วยความจำดัดแปลงมากจากสถาปัตยกรรมแบบฮาวาร์ด (Harvard)
- ชุดคำสั่ง มีขนาด 24 บิต และการประมวลผลข้อมูลได้ 16 บิต

- มีหน่วยความจำโปรแกรมแบบแฟลช (FLASH) ขนาด 48 กิโลไบท์ โดยสามารถลบและโปรแกรมซ้ำใหม่ได้กว่า 100,000 ครั้ง พร้อมระบบป้องกันการอ่านข้อมูล
- มีหน่วยความจำแบบแรม (RAM) ขนาด 2 กิโลไบท์
- มีหน่วยความจำข้อมูลแบบอีอีพรอม (EEPROM) ขนาด 1 กิโลไบท์ สามารถลบและเขียนซ้ำได้กว่า 1,000,000 ครั้ง และสามารถเก็บรักษาข้อมูลได้โดยไม่ต้องจ่ายไฟเลี้ยง



รูปที่ 2.11 แผนภาพของไมโครคอนโทรลเลอร์ dsPIC30F4012

- มีรีจิสเตอร์ขนาด 16 บิต ให้ใช้งานจำนวน 16 ตัว
- สามารถประมวลผลด้วยความเร็วสูงสุดที่ 30 ล้านคำสั่งต่อวินาที (30 MIPS)
- รองรับสัญญาณนาฬิกาภายนอกตั้งแต่ย่านไฟตรงจนถึง 40 เมกะเฮิร์ทซ์
- รองรับการใช้งานกับแหล่งกำเนิดความถี่แบบออสซิเนเลเตอร์ค่า 4-10 เมกะเฮิร์ทซ์
- มีวงจรคูณความถี่ภายในแบบเฟสล็อกลูป (Phase locked loop) โดยสามารถกำหนดค่าอัตราการคูณความถี่ได้ 3 ระดับ คือ 4 เท่า 8 เท่า และ 16 เท่า
- รองรับการอินเตอร์รัปต์ได้ถึง 30 แหล่ง พร้อมสัญญาณอินเตอร์รัปต์ภายนอก 3 แหล่ง และสามารถจัดระดับความสำคัญของการอินเตอร์รัปต์ได้ 8 ระดับ
- มีอินเตอร์รัปต์เวกเตอร์ 48 ตำแหน่ง
- มีวงจรตรวจจับไฟเลี้ยงต่ำกว่ากำหนดแบบโปรแกรมได้
- มีเพาเวอร์-ออนรีเซต เพาเวอร์อัปไทเมอร์ และออสซิเนเลเตอร์สตาร์อัปไทเมอร์
- มีวอตช์ด็อกไทเมอร์แบบโปรแกรมได้
- มีวงจรตรวจสอบการทำงานของวงจรกำเนิดสัญญาณนาฬิกา หากผิดพลาดจะเข้าสู่โหมดสัญญาณนาฬิกา RC พลังงานต่ำทันที
- รองรับการโปรแกรมในวงจรแบบอนุกรม (In-Circuit Serial Programming: ICSP)
- สามารถเลือกโหมดการใช้พลังงานได้
- ย่านไฟเลี้ยง 2.5 ถึง 5.5 โวลต์ กระแสไฟฟ้า 2.6 ถึง 44 มิลลิแอมป์ ที่ไฟเลี้ยง 5 โวลต์ ขึ้นอยู่กับการกำหนดความเร็วในการทำงาน

<u>คุณสมบัติของการประมวลผลสัญญาณดิจิตอล</u>

- มีแอกคิวมูเลเตอร์ขนาด 40 บิต 2 ตัวรองรับการประมวลผลทางคณิตศาสตร์
- มีหน่วยประมวลผลด้านการคูณและหารเลข 17 บิต ในรูปของฮาร์ดแวร์ จึงทำให้สามารถทำการคูณและหารเลขได้อย่างรวดเร็ว
- ทำการคูณเลข 16 บิต ได้ภายในสัญญาณนาฬิกาเพียง 1 ไซเคิล
- มีตัวเลื่อนข้อมูลบาร์เรล 40 สเตท ช่วยให้ประมวลผลข้อมูลที่มีจำนวนบิตมากๆ ทำได้รวดเร็ว
- มีวงจรเฟตช์ข้อมูลคู่ จึงทำให้สามารถประมวลผลข้อมูลได้อย่างรวดเร็ว

<u>คุณสมบัติของหน่วยอุปกรณ์ต่อพ่วง</u>

- ขาสัญญาณอินพุท/เอาท์พุท สามารถจ่ายกระแส (Source) และ รับกระแส (Sink) ได้มากถึง 25 มิลลิแอมป์

- มีไทเมอร์/เคาน์เตอร์ ขนาด 16 บิต จำนวน 3 ชุด และสามารถโปรแกรมให้ทำงานร่วมกันเป็น ไทเมอร์/เคาท์เตอร์ แบบ 32 บิต ได้
- มีหน่วยตรวจจับสัญญาณดิจิตอลขนาด 16 บิต จำนวน 4 ช่อง
- มีหน่วยเปรียบเทียบข้อมูลและกำเนิดสัญญาณพัลส์วิทมอดูเลชันขนาด 16 บิต จำนวน 2 ช่อง ในการเปรียบเทียบข้อมูลสามารถเลือกทำงานได้ 2 โหมด
- มีวงจรสื่อสารอนุกรมแบบ SPI จำนวน 1 ช่อง
- มีวงจรสื่อสารอนุกรมแบบ I2C จำนวน 1 ช่อง
- มีวงจรสื่อสารอนุกรมแบบ UART จำนวน 1 ช่อง
- มีวงจรหน่วยสร้างสัญญาณพัลส์วิทมอดูเลชัน สำหรับควบคุมมอเตอร์ 6 ช่อง
  - เลือกรูปแบบสัญญาณเอาท์พุทได้ทั้งแบบคอมพลีเมนต์และแบบอิสระ
  - มีโหมดปรับตำแหน่งการหมุนทั้งแบบปรับขอบสัญญาณและแบบกึ่งกลาง
  - มีส่วนกำเนิดความกว้างพัลส์ 3 ชุด
  - กำหนดฐานเวลาได้
  - สามารถเลือกขั้วของสัญญาณทางเอาท์พุทได้
  - มีสัญญาณกระตุ้นเพื่อให้ทำงานสัมพันธ์กับวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลภายในไมโครคอนโทรลเลอร์
  - สามารถควบคุมสัญญาณเอาท์พุทได้
- มีหน่วยเชื่อมต่อตัวเข้ารหัสแบบควอดราเจอร์ ขนาด 16 บิต จำนวน 1 ช่อง
  - มีอินพุท เฟสเอ เฟสบี และรับสัญญาณพัลส์เพื่อกำหนดตำแหน่ง
  - มีตัวนับตำแหน่งขนาด 16 บิต นับได้ทั้งขึ้นและลง
  - แสดงสถานะของทิศทางการนับได้
  - กำหนดโหมดการวัดตำแหน่งได้ 2 โหมดคือ คูณ 2 และ คูณ 4
  - มีวงจรกรองสัญญาณรบกวนแบบดิจิตอลจากอินพุทแบบโปรแกรมได้
  - กำหนดให้ทำงานเป็นแบบไทเมอร์/เคาท์เตอร์ ขนาด 16 บิต ได้
  - กำเนิดสัญญาณอินเตอร์รัปต์จากตำแหน่งที่นับเกิน หรือนับขาด
- มีวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลขนาด 10 บิต จำนวน 6 ช่อง
  - อัตราการสุ่มและแปลงสัญญาณ 1 ล้านตัวอย่างต่อวินาที
  - สามารถแปลงสัญญาณเมื่อไมโครคอนโทรลเลอร์ทำงานในโหมดสลีปและไอเดิล

2.5.2 หน่วยอุปกรณ์ต่อพ่วงที่ใช้ในโครงงาน

ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม การประมวลผลสัญญาณอินพุทจากเซนเซอร์มีทั้งที่เป็นสัญญาณอนาลอกและสัญญาณดิจิตอล รวมถึงการควบคุมมอเตอร์ทำด้วยสร้าง

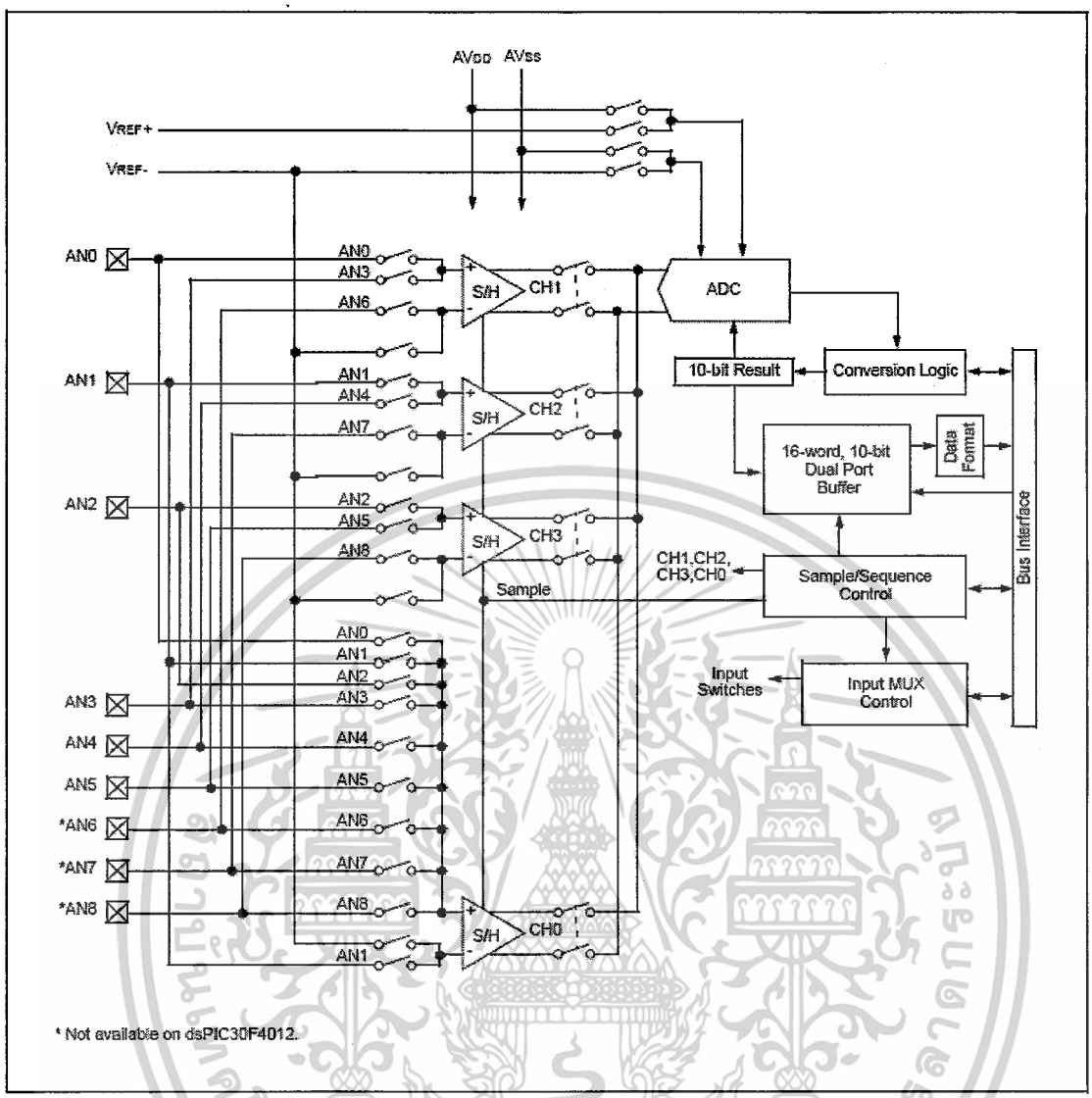
สัญญาณพัลส์วิทมอดูเลชันสำหรับเป็นสัญญาณอินพุทให้กับวงจรขับมอเตอร์ จึงได้ใช้ความสามารถของหน่วยอุปกรณ์ต่อพ่วงต่างๆ ดังนี้

<u>หน่วยการแปลงสัญญาณอนาลอกเป็นดิจิตอล</u>

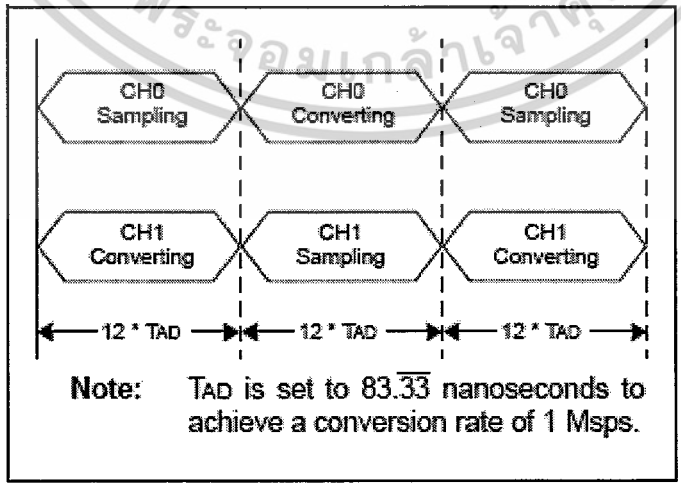
เนื่องจากโครงงานข้อมูลจากเซนเซอร์เป็นสัญญาณอนาลอกเพื่อที่จะนำไปประมวลผลจึงจำเป็นต้องมีการแปลงสัญญาณอนาลอกเป็นดิจิตอลเสียก่อนเป็นอันดับแรก ดังนั้นการแปลงสัญญาณอนาลอกเป็นดิจิตอลให้รวดเร็วจึงมีความสำคัญมาก เพราะว่าหากเกิดการล่าช้าขึ้นก็ย่อมจะส่งผลให้การประมวลผลต่อไปผิดพลาดเนื่องจากได้ข้อมูลที่ไม่เป็นจริงที่เวลานั้นๆ ดังนั้นในโครงงานนี้จะศึกษาการใช้หน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลโดยมีจุดมุ่งหมายให้ทำการแปลงสัญญาณได้เร็วและถูกต้องที่สุดเท่าที่หน่วยประมวลผลจะรองรับได้

ไมโครคอนโทรลเลอร์dsPIC30F4012 มีหน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลขนาด 10 บิต โดยมีแผนภาพโครงสร้างดังรูปที่ 2.12 ซึ่งสามารถทำการแปลงค่าได้สูงถึง 1 ล้านตัวอย่างต่อวินาที หน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลบน dsPIC มีช่องชักตัวอย่างและคงค่า (Sample and hold: S/H) อยู่ 4 ช่อง คือ CH0 CH1 CH2 และ CH3 ขาที่เป็นสัญญาณเข้าอนาลอกจะเชื่อมต่อกับช่องรักษาตัวอย่าง และคงค่าเหล่านี้ผ่านทางระบบของมัลติเพล็กเซอร์ โดยปกติขาที่เป็นสัญญาณเข้าอนาลอกจะเชื่อมต่อกับช่องชักตัวอย่าง และคงค่า CH0 CH1 CH2 หรือ CH3 จำนวน 1 ถึง 2 ช่อง ซึ่งการเลือกว่าจะใช้ขาใดโดยรีจิสเตอร์ฟังก์ชั่นพิเศษ (Special function register: SFR) ชื่อ ADCHS และ บิต CHPS (ADCON2<9:8>) จะใช้กำหนดว่าจะเปิดช่องชักตัวอย่างและคงค่าช่องใดให้ทำงานบ้าง เช่นหากกำหนดค่า ‘01’ ให้แก่ CHPS ช่องชักตัวอย่างและคงค่าช่อง CH0 และ CH1 จะทำงานเป็นต้น

หน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลสามารถกำหนดให้ช่องชักตัวอย่างและคงค่ารับสัญญาณอนาลอกจากขาสัญญาณเข้าได้พร้อมกัน 2 ช่อง 4 ช่อง หรือติดต่อกันแบบสัญญาณหนึ่งต่อจากอีกสัญญาณหนึ่ง โดยการทำงานในรูปแบบของการรับสัญญาณแบบติดต่อกันจะสามารถแปลงค่าได้เร็วที่สุด ซึ่งการกำหนดรูปแบบของการรับสัญญาณแบบติดต่อกัน สามารถทำได้โดยกำหนดค่า บิต SIMSAM (ADCON1<3>) ให้เป็น ‘0’ แสดงแผนผังการทำงานดังรูปที่ 2.13

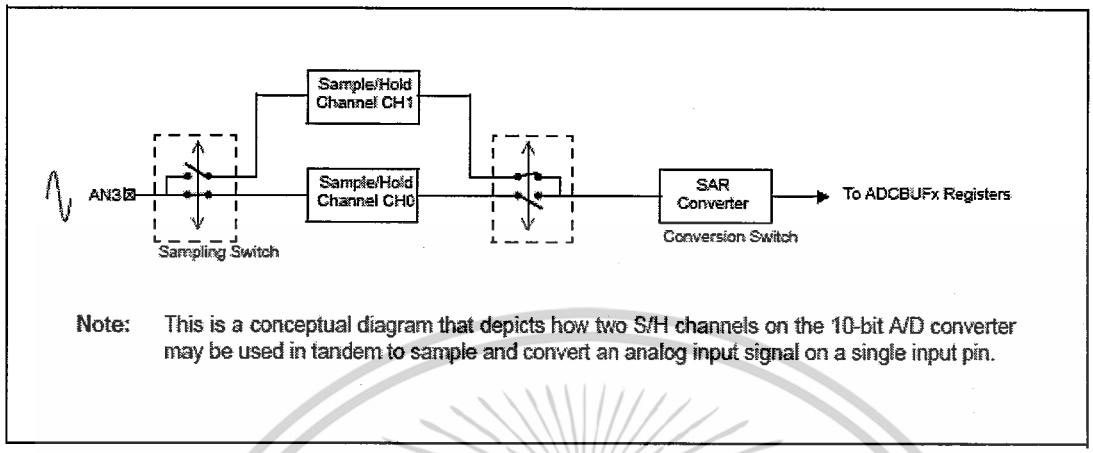


รูปที่ 2.12 แผนภาพหน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอล



รูปที่ 2.13 แผนผังการใช้ช่องชักตัวอย่างและคงค่า 2 ช่องในการรับสัญญาณแบบติดต่อกัน

เพื่อให้ทำการแปลงสัญญาณให้ได้ 1 ล้านครั้งต่อวินาที กำหนดให้ใช้ขาสัญญาณเข้าอนาลอก 1 ขาต่อช่องชักตัวอย่างและคงค่า 2 ช่องเช่น CH0 และ CH1 ดังรูป 2.14



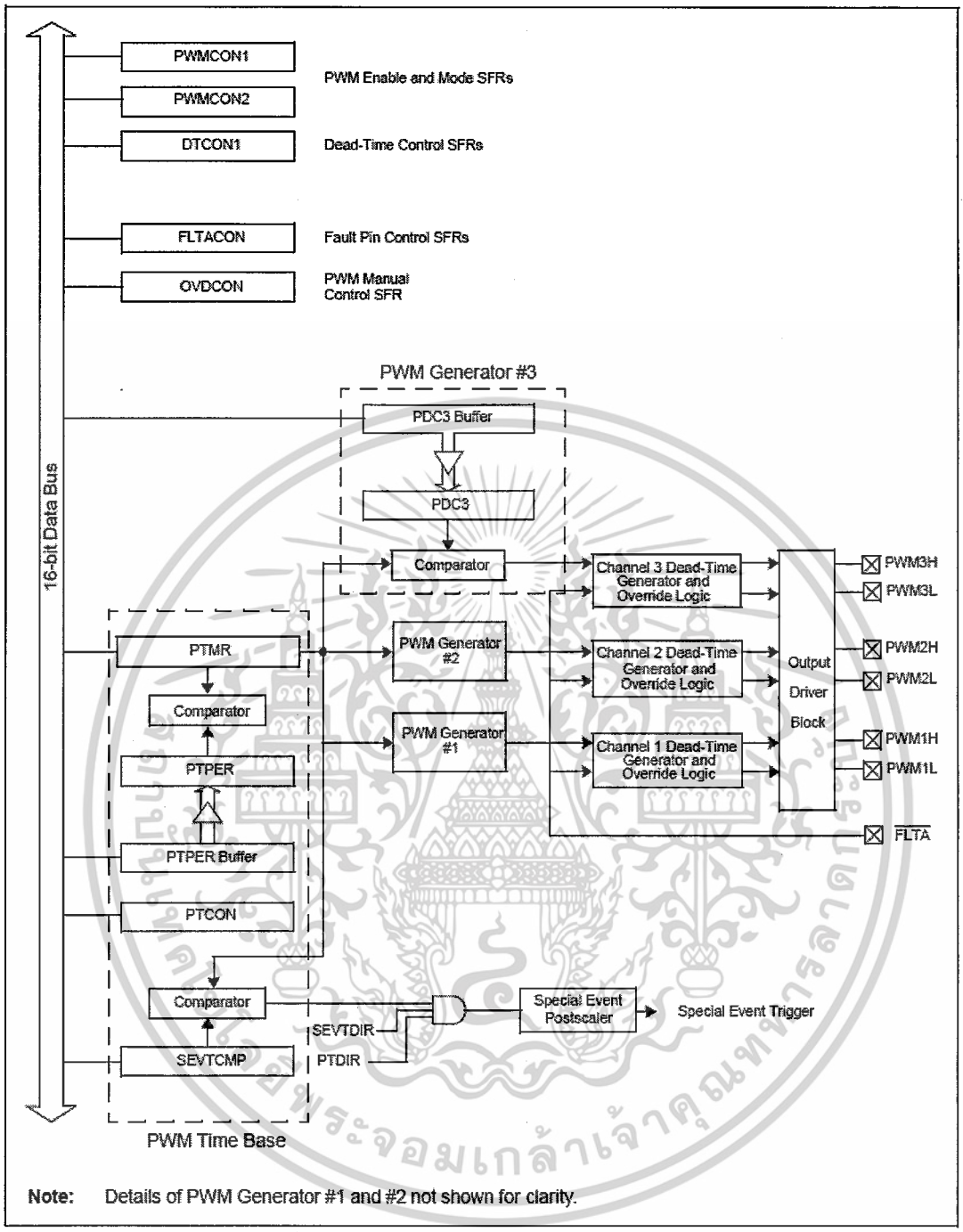
รูปที่ 2.14 แผนผังการเชื่อมต่อเพื่อให้ค่าชักตัวอย่างมากที่สุด

<u>หน่วยกำเนิดสัญญาณพัลส์วิทมอดูเลชัน</u>

อีกหนึ่งหน่วยฟังก์ชั่นที่ใช้ในโครงงาน คือหน่วยกำเนิดสัญญาณพัลส์วิทมอดูเลชัน (Pulse width modulation: PWM) ซึ่งใน dsPIC มีหน่วยกำเนิดสัญญาณพัลส์วิทมอดูเลชัน เพื่อควบคุมมอเตอร์โดยเฉพาะ เรียกว่าหน่วย MCPWM (Motor control PWM) ดังรูปที่ 2.15 สำหรับ dsPIC30F4012 มีช่องใช้งาน 6 ช่อง จึงสามารถขับมอเตอร์ไฟฟ้ากระแสตรงแบบเฟสเดียวได้ 3 ตัว ซึ่งในโครงงานนี้จะใช้เพียง 4 ช่องสำหรับควบคุมมอเตอร์ไฟฟ้ากระแสตรง 2 ตัวเท่านั้น แผนผังการทำงานแสดงในรูปที่ 2.15

ส่วนประกอบหลักของหน่วยนี้คือส่วนกำเนิดสัญญาณพัลส์วิทมอดูเลชัน ซึ่งได้ค่าฐานเวลามาจากรีจิสเตอร์ PTMR และ PTPER ในหน่วย MCPWM นี้สามารถกำหนดค่าความกว้างพัลส์ หรือค่าดิวตี้ไซเคิล (Duty cycle) ในส่วนกำเนิดสัญญาณพัลส์วิทมอดูเลชัน แต่ละส่วนได้เป็นอิสระต่อกัน นอกจากนั้นยังสามารถกำหนดการทำงานของขาพอร์ตเอาท์พุทของหน่วย MCPWMโดยตรงผ่านทางรีจิสเตอร์ OVDCON สัญญาณที่ออกจากหน่วย MCPWM จะมีขาพอร์ต 2 ขาต่อช่องนั่นคือขาเอาท์พุทด้านแรงดันสูง PWMxH และขาเอาท์พุทด้านแรงดันต่ำ PWMxL หรือเรียกว่าคู่เอาท์พุท ฐานเวลาของสัญญาณพัลส์วิทมอดูเลชัน ของหน่วย MCPWM สามารถกำหนดให้ทำงานได้ 4 โหมด คือ

- โหมดเปลี่ยนแปลงค่าอิสระ
- โหมดทำงานครั้งเดียว
- โหมดนับค่าขึ้นหรือลงอย่างต่อเนื่อง
- โหมดนับค่าขึ้นหรือลงอย่างต่อเนื่องพร้อมการอินเตอร์รัปต์เพื่อปรับปรุงค่า



รูปที่ 2.15 แผนภาพของโมดูลMCPWM

การเลือกโหมดทำได้โดยการกำหนดค่าที่ บิต PTMOD1 และ PTMOD0 ซึ่งเป็น บิต 1 และ 0 ในรีจิสเตอร์ PTCON ซึ่งในโครงงานนี้จะเลือกใช้โหมดเปลี่ยนแปลงค่าอิสระ ในโหมดนี้ค่าฐานเวลาจะเพิ่มค่าขึ้นจนกระทั่งตรงกับค่าในรีจิสเตอร์ PTER จากนั้นรีจิสเตอร์ PTMR จะรีเซตและทำการนับค่าเพิ่มขึ้นต่อเนื่องไปอีกตราบเท่าที่ บิต PTEN ยังคงมีค่าเป็น '1' อยู่

การเปลี่ยนค่าความกว้างพัลส์ ของสัญญาณพัลส์วิทมอดูเลชัน ของหน่วย MCPWM กำหนดได้จากรีจิสเตอร์ PDC1 - PDC4 ซึ่งต่างก็มีบัฟเฟอร์เพื่อป้องกันสัญญาณรบกวนเมื่อมีการปรับปรุงค่าของสัญญาณ PWM โดยค่าความกว้างพัลส์ของสัญญาณพัลส์วิทมอดูเลชัน จะถูกปรับปรุงค่าตามข้อมูลที่เขียนลงในรีจิสเตอร์ PDCx จากนั้นค่าจากรีจิสเตอร์ PDCx จะถูกส่งไปยังบัฟเฟอร์เพื่อทำการเปรียบเทียบ เมื่อมีการเปลี่ยนแปลงข้อมูลในรีจิสเตอร์ PDCx เรียบร้อย ข้อมูลนั้นจึงถูกส่งไปบัฟเฟอร์ข้อมูลนั้นจึงถูกส่งไปยังบัฟเฟอร์เพื่อทำงานต่อไป ทำให้ไม่เกิดการติดขัดหรือเกิดความผิดพลาดในขณะเปลี่ยนค่าความกว้างพัลส์ เมื่อฐานเวลาสัญญาณพัลส์วิทมอดูเลชันทำงานในโหมดเปลี่ยนแปลงค่าอิสระหรือโหมดทำงานครั้งเดียว ความกว้างพัลส์ของสัญญาณพัลส์วิทมอดูเลชัน ถูกปรับปรุงเมื่อค่าของรีจิสเตอร์ PTMR เท่ากับ PTPER และเมื่อรีจิสเตอร์ PTMR เกิดการรีเซตเป็น ‘0’

นอกจากนั้นในหน่วย MCPWM ยังมีความสามารถพิเศษอีกประการหนึ่ง คือ การป้องกันการเปลี่ยนค่าความกว้างพัลส์ ทำได้โดยกำหนด บิต UDIS ซึ่งเป็น บิต 2 ของรีจิสเตอร์ PWMCON2 ให้มีค่าเป็น ‘1’

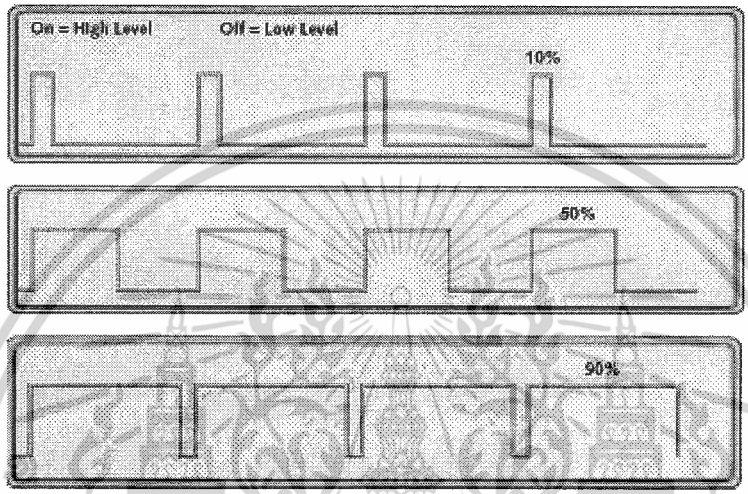
2.6 หลักการควบคุมมอเตอร์ไฟฟ้ากระแสตรง

ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ได้ใช้มอเตอร์ไฟฟ้ากระแสตรงในการขับเคลื่อนตัวหุ่นยนต์ เพื่อรักษาสภาพสมดุลไว้ ซึ่งการควบคุมมอเตอร์ไฟฟ้ากระแสตรงมีการควบคุมความเร็วและทิศทางการหมุนของมอเตอร์ มอเตอร์กระแสตรงเป็นเครื่องกลทางไฟฟ้าที่เปลี่ยนพลังงานไฟฟ้าเป็นพลังงานกลผ่านทางแกนหมุนหรือเพลา มอเตอร์สามารถที่จะหมุนได้เนื่องจากมีสนามแม่เหล็ก 2 แหล่งกระทำต่อกัน โดยที่สนามแม่เหล็กทั้ง 2 แหล่ง อาจจะเป็นแบบที่ได้จากการจ่ายกระแสไฟฟ้าผ่านขดลวดสเตเตอร์ (Stator winding) และขดลวดอาร์เมเจอร์ แต่มอเตอร์กระแสตรงที่นิยมใช้เป็นแบบแม่เหล็กถาวร เป็นตัวสร้างสนามแม่เหล็กแทนขดลวดสเตเตอร์ และใช้การผ่านกระแสไฟฟ้าเข้าไปที่ขดลวดอาร์เมเจอร์ เนื่องจากจะลดความสูญเสียได้เพราะไม่มีขดลวดสนามแม่เหล็ก (Field winding) ทำให้มีประสิทธิภาพที่ดีขึ้น นอกจากนี้มอเตอร์กระแสตรงยังมีขนาดเล็ก และราคาถูกสนามแม่เหล็กที่เกิดจากแม่เหล็กถาวร เกิดจากการจ่ายกระแสไฟฟ้ากระแสตรงเข้าไปในขดลวดอาร์เมเจอร์ จะทำให้เกิดแรงบิดขึ้นที่โรเตอร์ซึ่งจะทำให้เกิดการหมุนได้นั่นเอง

2.6.1 การควบคุมความเร็วมอเตอร์ไฟฟ้ากระแสตรง

เนื่องจากส่วนของขดลวดสเตเตอร์ เป็นแม่เหล็กถาวร การควบคุมความเร็วจึงทำได้โดยการเปลี่ยนค่าความต่างศักย์ที่ขดลวดอาเมเจอร์ ซึ่งค่าความต่างศักย์นี้จะแปรผันตรงกับความเร็วในการหมุนของมอเตอร์ วิธีที่จะเปลี่ยนระดับความต่างศักย์ไฟฟ้า จะใช้คลื่นรูปสี่เหลี่ยมที่สามารถ

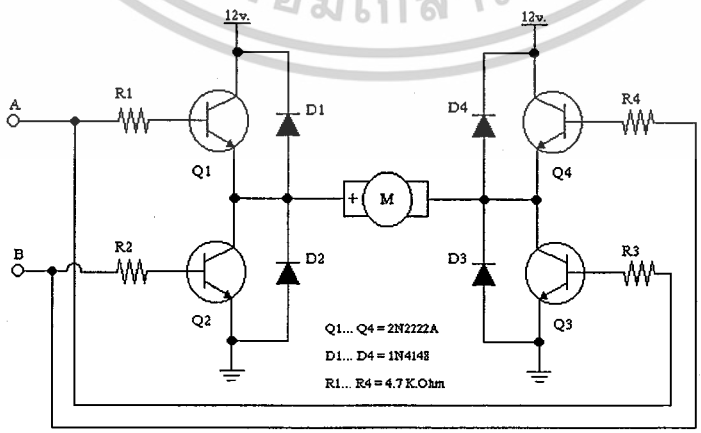
เปลี่ยนแปลงช่วงเวลาในการเปิด และปิดแหล่งจ่ายได้ ซึ่งจะเป็นการเปลี่ยนแปลงค่าเฉลี่ยของแรงดัน เรียกว่า สัญญาณพัลส์วิทมอดูเลชัน ดังรูปที่ 2.16 ซึ่งในโครงงานนี้ได้ใช้ไมโครคอนโทรลเลอร์สร้างสัญญาณคลื่นรูปสี่เหลี่ยมสำหรับใช้ควบคุมมอเตอร์ขึ้นมา โดยการปรับความกว้างของพัลส์ หรือค่าดิวตี้ไซเคิล เมื่อความกว้างของพัลส์มีค่ามากจะทำให้ค่าเฉลี่ยของแรงดันมีค่ามากหรือทำให้มอเตอร์หมุนเร็ว ถ้าความกว้างของพัลส์มีค่าน้อยจะทำให้ค่าเฉลี่ยของแรงดันมีค่าน้อยหรือทำให้มอเตอร์หมุนช้า



รูปที่ 2.16 สัญญาณพัลส์วิทมอดูเลชัน

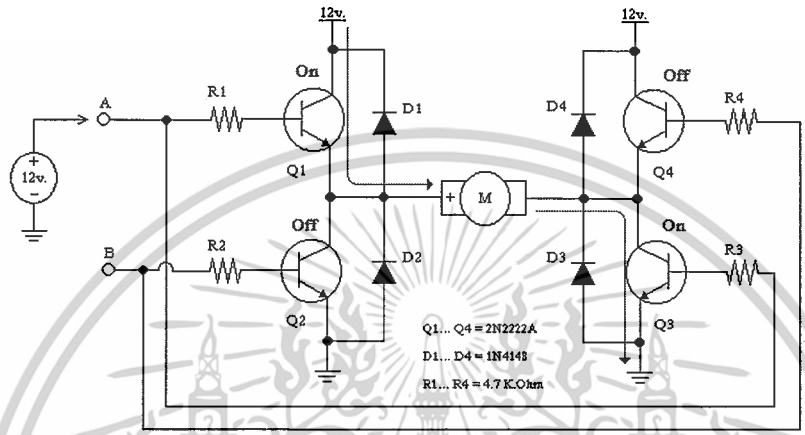
2.6.2 การควบคุมทิศทางของมอเตอร์กระแสตรง

การควบคุมทิศการหมุนของมอเตอร์กระแสตรงสามารถทำได้ โดยการควบคุมทิศทางการไหลของกระแสไฟฟ้าที่ไหลผ่านขดลวดอาร์เมเจอร์ ซึ่งสามารถทำได้โดยการกลับขั้วไฟฟ้าที่ป้อนให้กับขั้วของมอเตอร์โดยการควบคุมการไหลของกระแสไฟฟ้านั้น ส่วนใหญ่แล้วจะใช้วิธีการต่อวงจรอิเล็กทรอนิกส์ที่เรียกว่าเอชบริดจ์ (H-bridge) เข้ากับมอเตอร์ ดังรูปที่ 2.17



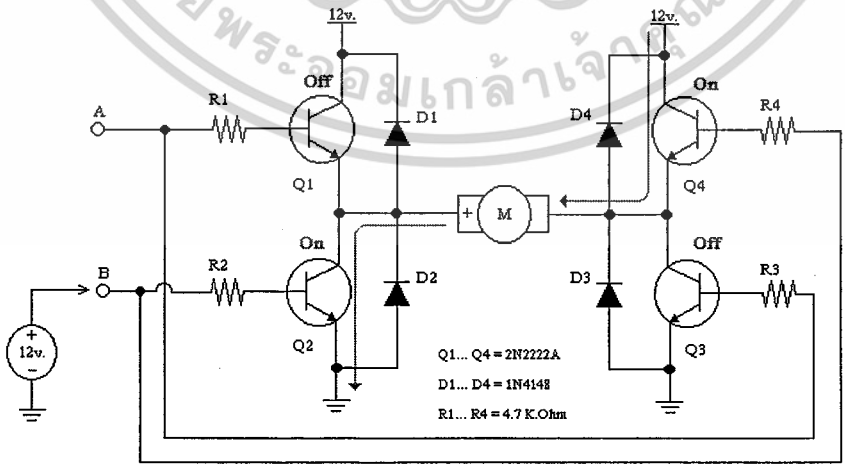
รูปที่ 2.17 วงจรเอชบริดจ์

การทำงานของวงจรเอชบริดจ์ คือเมื่อจ่ายไฟที่จุด A จะทำให้ทรานซิสเตอร์ Q1 และ Q3 ทำงานส่งผลให้มีกระแสไหล จากแหล่งจ่าย ผ่านขาคอลเล็กเตอร์ (Collecter) และอีมิเตอร์ (Emitter) ของ Q1 ผ่านเข้าสู่ขั้วบวก (+) ของมอเตอร์ ผ่านไปยังขาคอลเล็กเตอร์ และอีมิเตอร์ ของ Q3 ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางบวก และครบวงจร จึงทำให้มอเตอร์สามารถหมุน ในทิศทางตามเข็มนาฬิกา ดังรูปที่ 2.18



รูปที่ 2.18 การทำงานของวงจรเอชบริดจ์เมื่อจ่ายไฟที่จุด A

และเมื่อจ่ายไฟที่จุด B จะทำให้ทรานซิสเตอร์ Q2 และ Q4 ทำงานส่งผลให้มีกระแสไหล จากแหล่งจ่ายผ่านขาคอลเล็กเตอร์ และอีมิเตอร์ ของ Q4 ผ่านเข้าสู่ขั้วลบ (-) ของมอเตอร์ ผ่านไปยังขาคอลเล็กเตอร์ และอีมิเตอร์ ของ Q2 ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางลบ และครบวงจร จึงทำให้มอเตอร์สามารถหมุน ในทิศทางทวนเข็มนาฬิกาได้ ดังรูปที่ 2.19



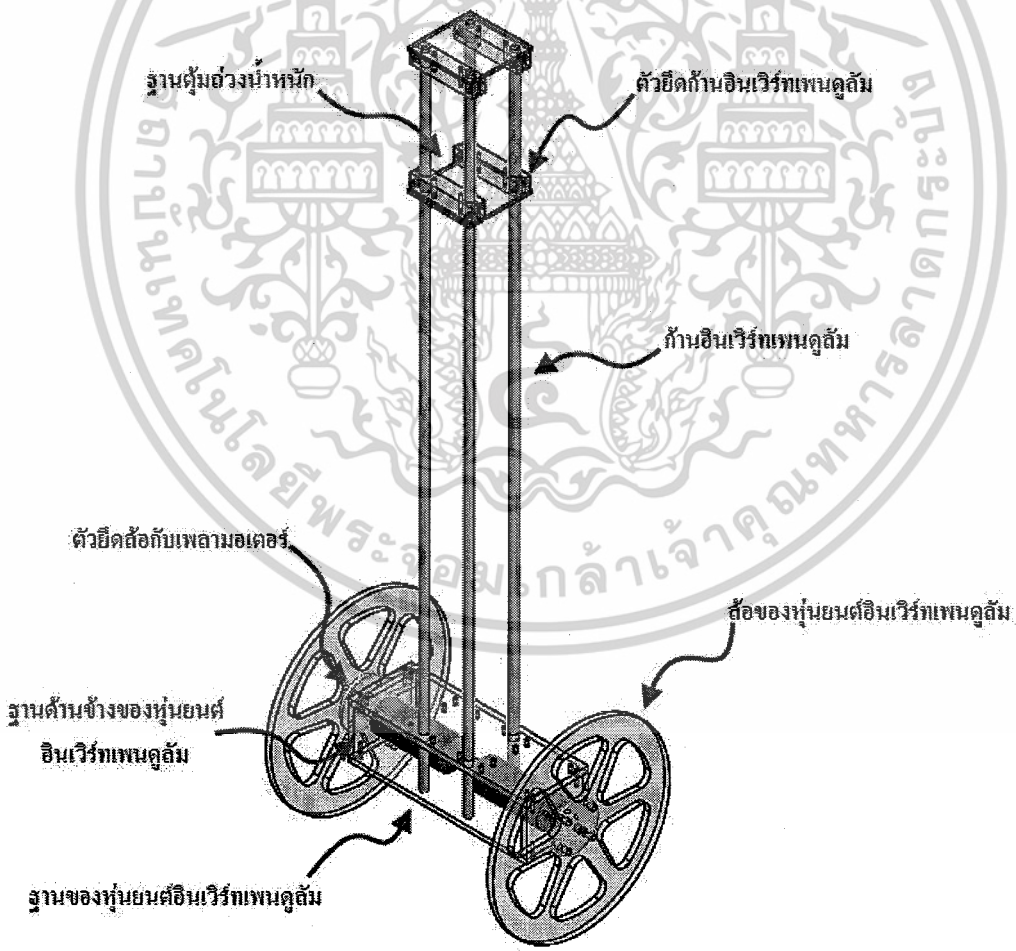
รูปที่ 2.19 การทำงานของวงจรเอชบริดจ์เมื่อจ่ายไฟที่จุด B

บทที่ 3
การออกแบบ

ในบทนี้จะกล่าวถึงการออกแบบส่วนต่างๆ ของโครงงานระบบควบคุมอินเวิร์ทเพนดูลัม ซึ่งแบ่งออกเป็น 2 หัวข้อใหญ่ คือ การออกแบบโครงสร้างของหุ่นยนต์อินเวิร์ทเพนดูลัม และการออกแบบระบบควบคุม ดังนี้

3.1 การออกแบบโครงสร้างของหุ่นยนต์อินเวิร์ทเพนดูลัม

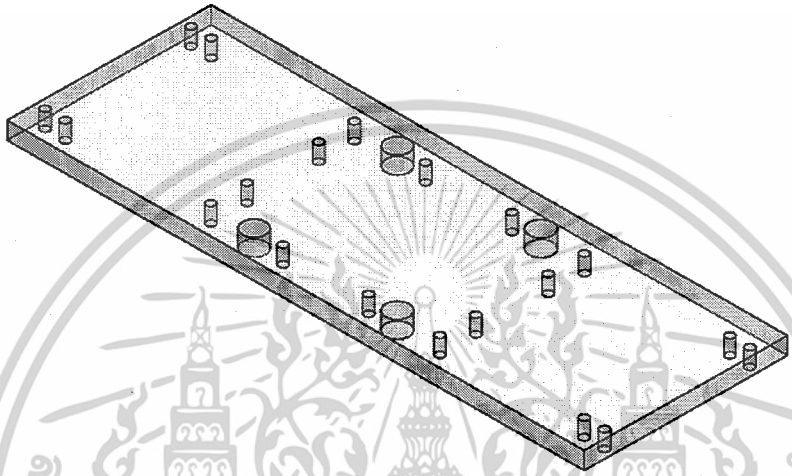
ในหัวข้อนี้ได้กล่าวถึงการออกแบบโครงสร้างของหุ่นยนต์อินเวิร์ทเพนดูลัม โดยจากรูปที่ 3.1 แสดงโครงสร้างของหุ่นยนต์อินเวิร์ทเพนดูลัมที่ออกแบบชิ้นส่วนต่างๆและประกอบเสร็จสมบูรณ์แล้วด้วยโปรแกรม SolidWorks



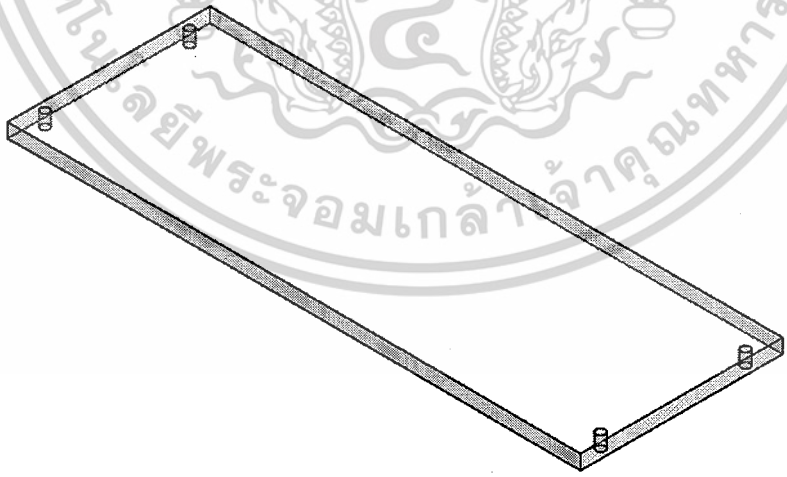
รูปที่ 3.1 โครงสร้างหุ่นยนต์อินเวิร์ทเพนดูลัม

3.1.1 การออกแบบฐานของหุ่นยนต์อินเวิร์ทเพนดูลัม

ส่วนฐานของหุ่นยนต์อินเวิร์ทเพนดูลัมประกอบด้วยแผ่นอะคริลิก ความกว้าง 70 มิลลิเมตร ยาว 200 มิลลิเมตร หนา 6 มิลลิเมตร จำนวน 2 แผ่น เป็นรูปสี่เหลี่ยมผืนผ้า ทำให้ง่ายต่อการติดตั้ง และมีน้ำหนักเบาเพราะทำจากแผ่นอะคริลิก จากการออกแบบด้วยโปรแกรม SolidWorks ฐานของหุ่นยนต์อินเวิร์ทเพนดูลัมทั้งสอง มีลักษณะดังรูปที่ 3.2 และ รูปที่ 3.3



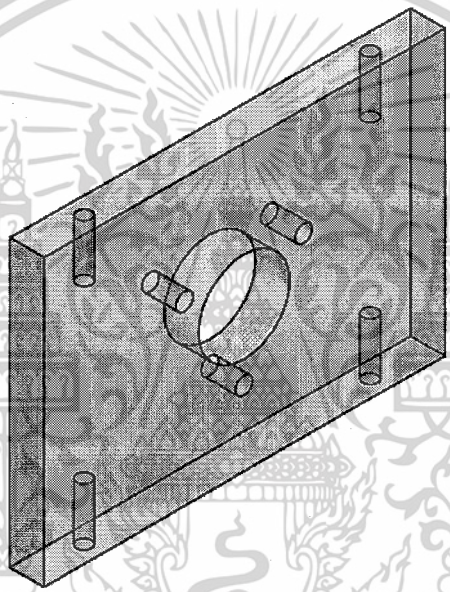
รูปที่ 3.2 ฐานด้านบนของหุ่นยนต์อินเวิร์ทเพนดูลัม



รูปที่ 3.3 ฐานด้านล่างของหุ่นยนต์อินเวิร์ทเพนดูลัม

3.1.2 ฐานด้านข้างของหุ่นยนต์อินเวิร์ทเพนดูลัม

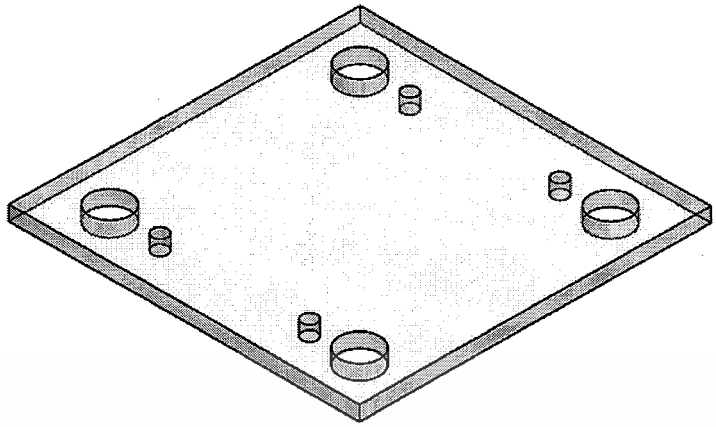
ชิ้นส่วนนี้ได้ออกแบบมาให้อยู่ระหว่าง ส่วนของฐานด้านบนและด้านล่างของหุ่นยนต์อินเวิร์ทเพนดูลัมโดยขนาดของส่วนฐานด้านข้างต้องมีขนาดใหญ่พอสำหรับติดตั้งมอเตอร์ระหว่างฐานด้านบน และฐานด้านล่างได้ โดยออกแบบให้มีขนาดความกว้าง 50 มิลลิเมตร ยาว 70 มิลลิเมตร เจาะรูความความกว้างรัศมีเท่ากับ 6 มิลลิเมตรสำหรับให้เพลามอเตอร์ยื่นออกมา และรัศมี 1.5 มิลลิเมตร ด้านข้างสำหรับยึดตัวมอเตอร์ และด้านบนสำหรับยึดฐานด้านบน และด้านล่างให้ยึดกับฐานด้านข้าง โดยฐานด้านข้างของหุ่นยนต์อินเวิร์ทเพนดูลัมมีทั้งหมด 2 ชิ้น คือ ชิ้นด้านซ้าย และด้านขวา ซึ่งมีลักษณะเหมือนกัน ทำด้วยแผ่นอะคริลิก หนา 5 มิลลิเมตร ดังรูปที่ 3.4



รูปที่ 3.4 ฐานด้านข้างของหุ่นยนต์อินเวิร์ทเพนดูลัม

3.1.3 การออกแบบฐานของตุ้มถ่วงบนก้านเพนดูลัม

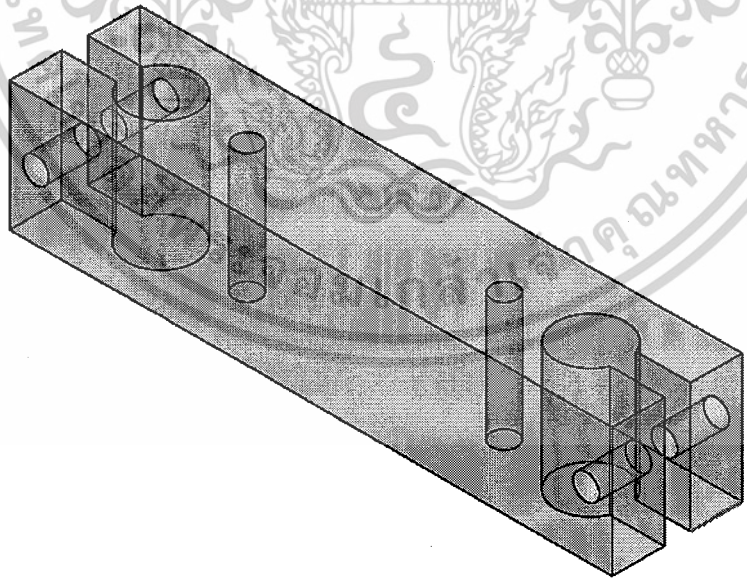
จากการออกแบบให้มีตุ้มถ่วงน้ำหนักบนก้านเพนดูลัม และก้านเพนดูลัมทำด้วยแท่งอะลูมิเนียมสำเร็จรูปจำนวน 4 แท่ง ทำให้ต้องออกแบบฐานของตุ้มน้ำหนักที่รองรับกับแท่งอะลูมิเนียม จึงออกแบบให้ฐานของตุ้มถ่วงน้ำหนักมีขนาดกว้าง 70 มิลลิเมตร ยาว 70 มิลลิเมตร และมีช่องสำหรับแท่งอะลูมิเนียมทั้ง 4 แท่ง ตัวฐานทำด้วยแผ่นอะคริลิก หนา 3 มิลลิเมตร ที่มีความแข็งแรงสำหรับรองรับน้ำหนักของแบตเตอรี่ โดยฐานของตุ้มน้ำหนักบนก้านเพนดูลัม มีลักษณะดังรูปที่ 3.5



รูปที่ 3.5 ฐานของตุ้มถ่วงน้ำหนักบนก้านเพนดูลัม

3.1.4 ตัวยึดก้านเพนดูลัม

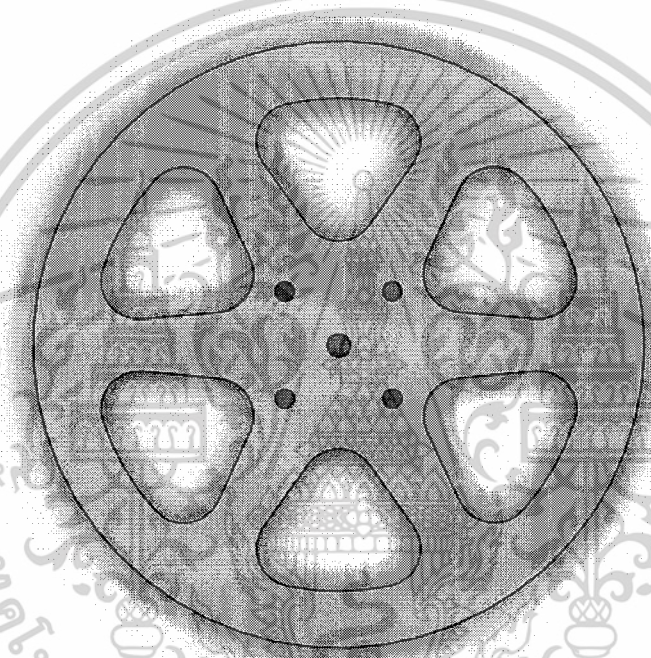
เนื่องจากก้านเพนดูลัมเป็นแท่งอะลูมิเนียม ตัวยึดนี้ออกแบบมาเพื่อยึดแท่งอะลูมิเนียมกับฐานของหุ่นยนต์ และฐานของตุ้มถ่วงน้ำหนัก ตัวยึดจึงต้องมีความแข็งแรงพอสมควร ดังนั้นจึงออกแบบให้ตัวยึดมีความกว้าง 15 มิลลิเมตร ยาว 70 มิลลิเมตร และหนาเท่ากับ 15 มิลลิเมตร เจาะรูที่ปลายทั้งสองด้านสำหรับใส่แท่งอะลูมิเนียม ในโครงงานหุ่นยนต์อินเวิร์ทเพนดูลัมมีตัวยึดบนก้านเพนดูลัมทั้งหมด 6 ตัว มีลักษณะเป็นดังรูปที่ 3.6



รูปที่ 3.6 ตัวยึดบนก้านเพนดูลัม

3.1.5 การออกแบบล้อของหุ่นยนต์อินเวิร์ทเพนดูลัม

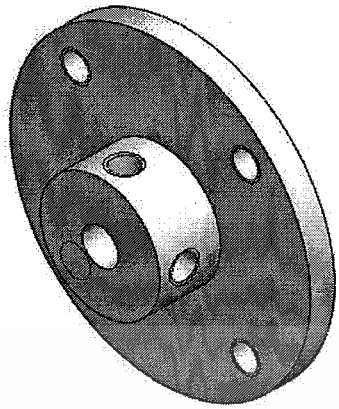
ล้อของหุ่นยนต์อินเวิร์ทเพนดูลัมมีส่วนสำคัญต่อความยากง่ายของการควบคุมสมดุลของหุ่นยนต์เพนดูลัม โดยในโครงงานระบบควบคุมอินเวิร์ทเพนดูลัมนี้ ได้มีการทดลองใช้ล้อที่มีรัศมีความกว้างที่ต่างกัน พบว่าล้อที่มีรัศมีกว้างทำให้การควบคุมสมดุลทำได้ง่ายขึ้น แต่ถ้าล้อมีรัศมีที่กว้างเกินไปก็จะทำให้ไม่สมดุลกับก้านเพนดูลัมเดิม ดังนั้นหุ่นยนต์อินเวิร์ทเพนดูลัมนี้ได้ออกแบบล้อให้มีรัศมีความกว้าง 100 มิลลิเมตร และใช้วัสดุเป็นแผ่นอะคริลิก หนา 6 มิลลิเมตร ซึ่งมีลักษณะเป็นดังรูปที่ 3.7



รูปที่ 3.7 ล้อของหุ่นยนต์อินเวิร์ทเพนดูลัม

3.1.6 ตัวยึดล้อของหุ่นยนต์อินเวิร์ทเพนดูลัม

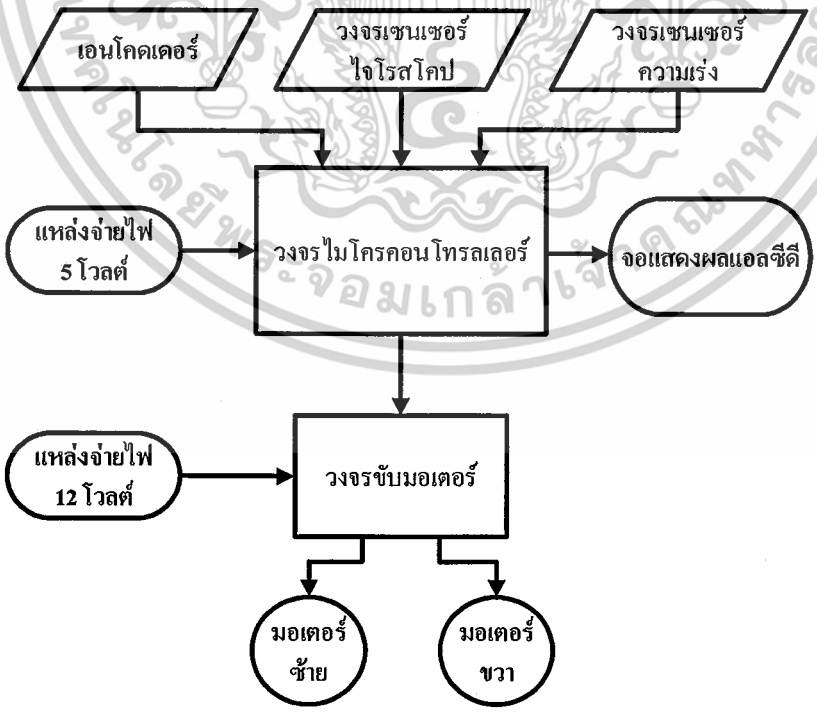
ตัวยึดล้อของหุ่นยนต์อินเวิร์ทเพนดูลัมนี้ มีหน้าที่ยึดล้อ เข้ากับมอเตอร์บนตัวของหุ่นยนต์ เนื่องจากชิ้นส่วนนี้ได้รับแรงกระทำโดยตรงจากมอเตอร์ เนื่องจากต้องการความแข็งแรง จึงนำอะลูมิเนียมมาผลิตเป็นตัวยึดล้อ ทำให้ได้ตัวยึดที่แข็งแรง และมีน้ำหนักเบา โดยตัวยึดล้อนี้มีลักษณะเป็นจานวงกลมมีรัศมีความกว้างเท่ากับ 25 มิลลิเมตร และมีแป้นกลมนูนขนาดรัศมีความกว้าง 10 มิลลิเมตร โดยเจาะรูสำหรับเพลามอเตอร์ขนาดรัศมี 3 มิลลิเมตรพอดี โดยมีลักษณะเป็นดังรูปที่ 3.8



รูปที่ 3.8 ตัวยึดล้อของหุ่นยนต์อินเวิร์ทเพนดูลัม

3.2 การออกแบบวงจร

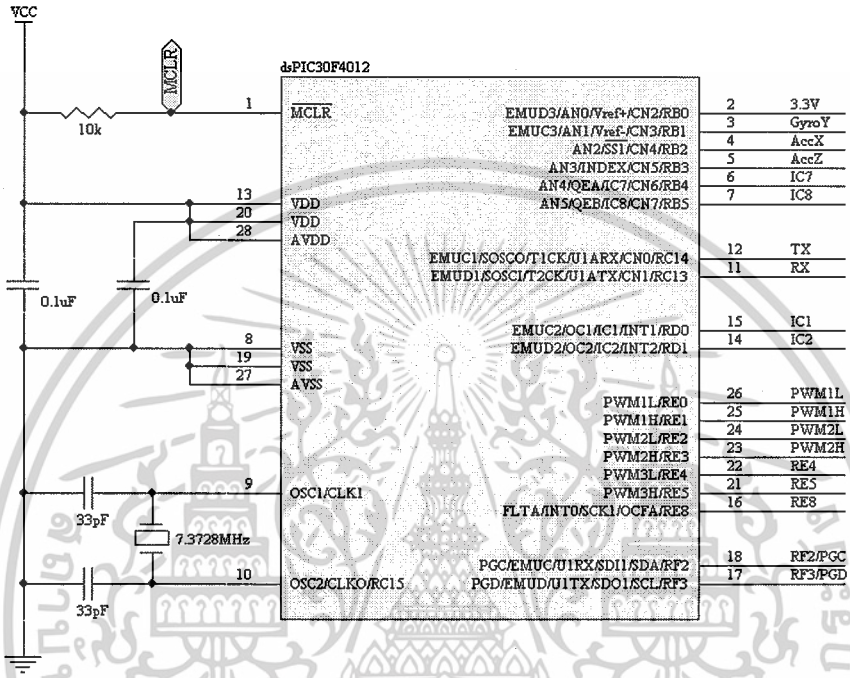
ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ประกอบด้วยวงจรอิเล็กทรอนิกส์ไฟฟ้า ดังแผนภาพวงจรรูปที่ 3.9 ได้แก่ วงจรไมโครคอนโทรลเลอร์ วงจรแสดงผลแอลซีดี วงจรเซนเซอร์ไจโรสโคป วงจรเซนเซอร์ความเร่ง วงจรจ่ายแรงดัน 3.3โวลต์ วงจรขับมอเตอร์ และวงจรรับอินพุทจากเอนโคดเดอร์ ในบทนี้จะอธิบายถึงการออกแบบวงจรต่างๆ ที่ใช้ในโครงงาน



รูปที่ 3.9 แผนภาพการเชื่อมต่อวงจรต่างๆ

3.2.1 วงจรไมโครคอนโทรลเลอร์

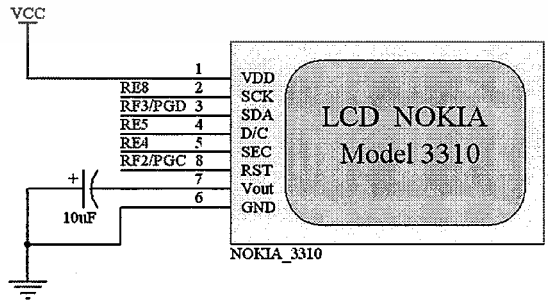
วงจรไมโครคอนโทรลเลอร์เป็นวงจรที่ออกแบบมาเพื่อให้รับข้อมูลจากเซนเซอร์แล้วนำมาประมวลผลส่งไปควบคุมมอเตอร์ให้รักษาสมดุลของหุ่นยนต์อินเวิร์ทเพนดูลัม รวมถึงควบคุมการแสดงผลบนจอแอลซีดีกราฟฟิกด้วย ในโครงงานนี้ใช้ไมโครคอนโทรลเลอร์ของบริษัทไมโครชิพ (Microchips) เบอร์ dsPIC4012 ประกอบไปด้วยอุปกรณ์ต่างๆ ดังรูปที่ 3.10



รูปที่ 3.10 วงจรไมโครคอนโทรลเลอร์

3.2.2 วงจรจอแสดงผลแอลซีดี

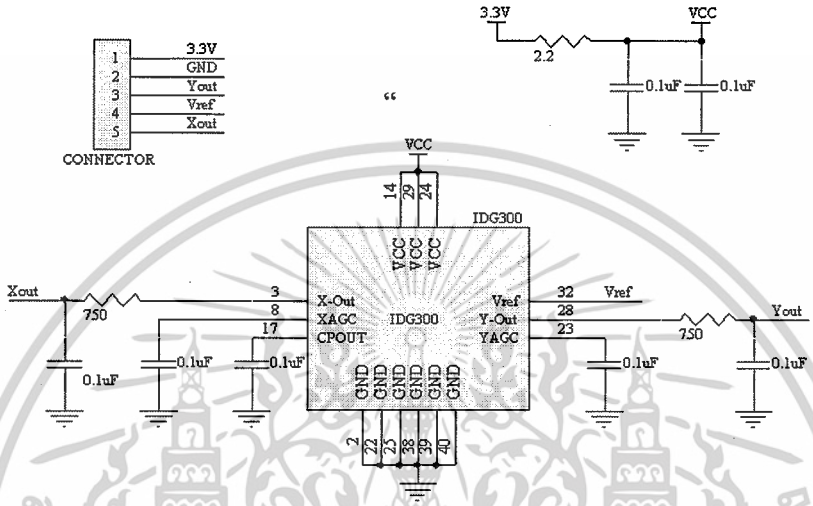
ในโครงงานนี้ ได้นำจอแอลซีดีมาใช้เพื่อแสดงผลค่าต่างๆ ที่วัดได้จากเซนเซอร์ จอแอลซีดีที่นำมาใช้เป็นจอแอลซีดีกราฟฟิก โดยมีวงจรดังรูปที่ 3.11 แต่นำมาพัฒนาให้สามารถแสดงผลข้อความได้ 6 บรรทัด บรรทัดละ 16 ตัวอักษร โดยติดต่อกับไมโครคอนโทรลเลอร์ผ่านระบบสื่อสารแบบ SPI



รูปที่ 3.11 วงจรจอแสดงแอลซีดี

3.2.3 วงจรเซนเซอร์ไจโรสโคป

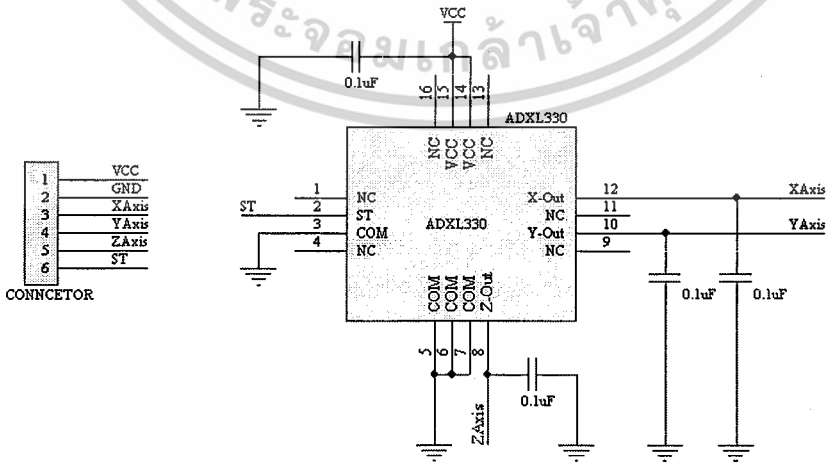
เซนเซอร์ไจโรสโคปเป็นเซนเซอร์ที่วัดอัตราเร็วเชิงมุม นำมาใช้ในการควบคุมสมดุลให้หุ่นยนต์อินเวิร์ทเพนดูลัม เซนเซอร์ที่ใช้คือเบอร์ IDG300 ให้เอาท์พุทเป็นสัญญาณอนาลอก และใช้ไฟเลี้ยง 3.3โวลต์ โดยที่ขาเอาท์พุทของแกน X และแกน Y ได้ต่อตัวต้านทานและตัวเก็บประจุไว้เพื่อเป็นวงจรกรองความถี่ต่ำผ่าน (Low pass filter) ดังรูปที่ 3.12



รูปที่ 3.12 วงจรเซนเซอร์ไจโรสโคป

3.2.4 วงจรเซนเซอร์ความเร่ง

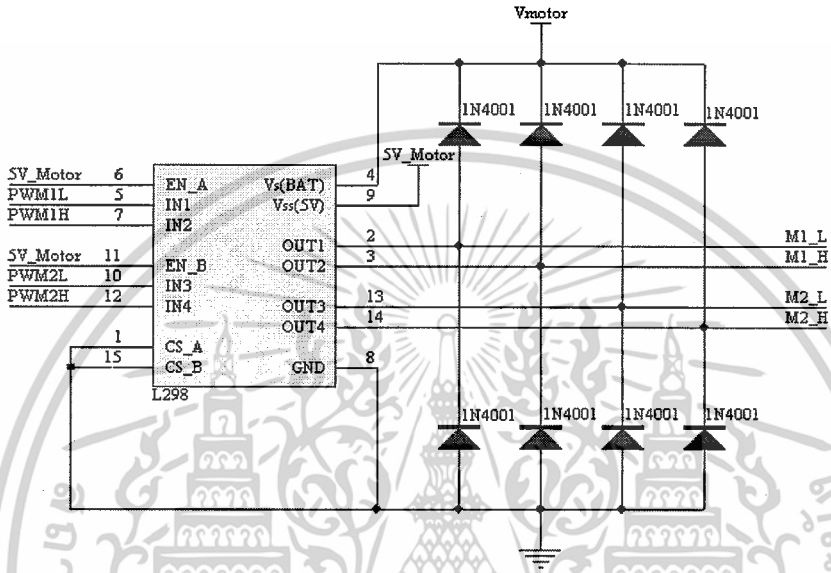
สำหรับเซนเซอร์ความเร่งถูกนำมาประยุกต์ใช้วัดค่ามุมเบี่ยงเบนของหุ่นยนต์ โดยในโครงงานนี้เลือกใช้เซนเซอร์เบอร์ ADXL330 ซึ่งให้เอาท์พุทเป็นสัญญาณอนาลอก และใช้ไฟเลี้ยง 3.3โวลต์ มีวงจรตามรูปที่ 3.13



รูปที่ 3.13 วงจรเซนเซอร์ความเร่ง

3.3.5 วงจรขับมอเตอร์

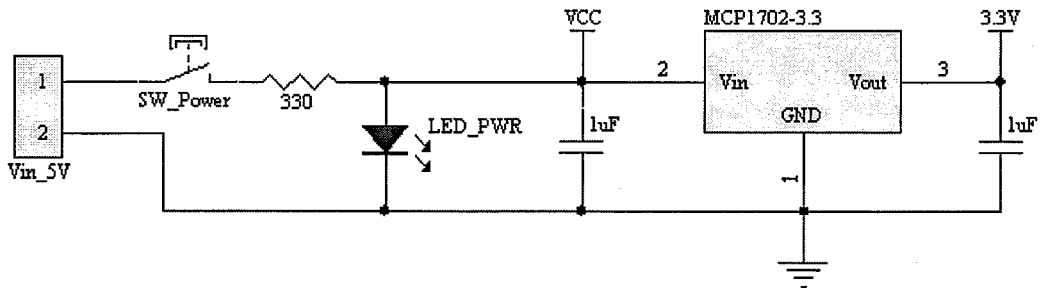
ในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ต้องควบคุมมอเตอร์ไฟฟ้ากระแสตรง สองตัว และมีพื้นที่สำหรับติดตั้งวงจรจำกัด จึงเลือกวงจรขับมอเตอร์ เป็นไอซีขับมอเตอร์สำเร็จรูป เบอร์ L298N โดยมีวงจรดังรูปที่ 3.14 ซึ่งสามารถขับมอเตอร์ได้ 2 ตัว และจ่ายกระแสได้ 2 แอมแปร์ เพียงพอสำหรับการใช้งานในโครงงาน



รูปที่ 3.14 วงจรขับมอเตอร์

3.2.6 วงจรจ่ายแรงดัน 3.3 โวลต์

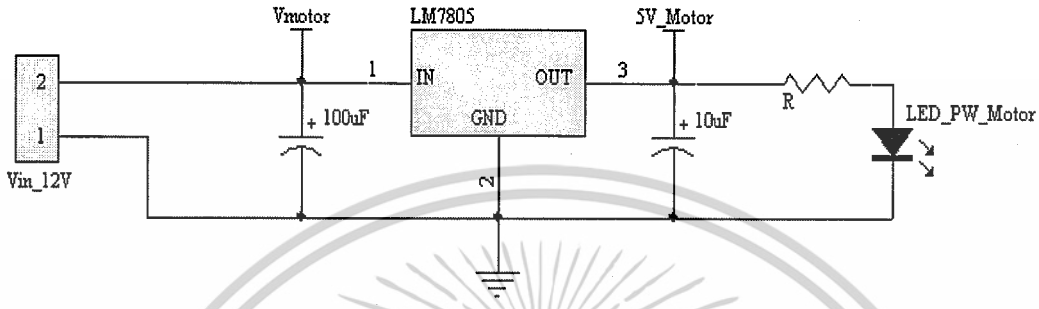
เนื่องจากเซนเซอร์ไจโรสโคป และเซนเซอร์ความเร่งที่ใช้ในโครงงานระบบควบคุม อินเวิร์ทเพนดูลัมใช้ไฟเลี้ยง 3.3 โวลต์ จึงต้องมีแหล่งวงจรจ่ายแรงดัน 3.3 โวลต์ ที่แปลงแรงดันจาก 5 โวลต์ เป็น 3.3 โวลต์ โดยในโครงงานเลือกใช้ไอซีของบริษัทไมโครชิพ เบอร์ MCP1702–3.3 ซึ่งเป็น ไอซีเรกกูเลเตอร์ 3.3 โวลต์ มีวงจรดังรูปที่ 3.15



รูปที่ 3.15 วงจรจ่ายแรงดัน 3.3 โวลต์

3.2.7 วงจรจ่ายแรงดัน 5 โวลต์

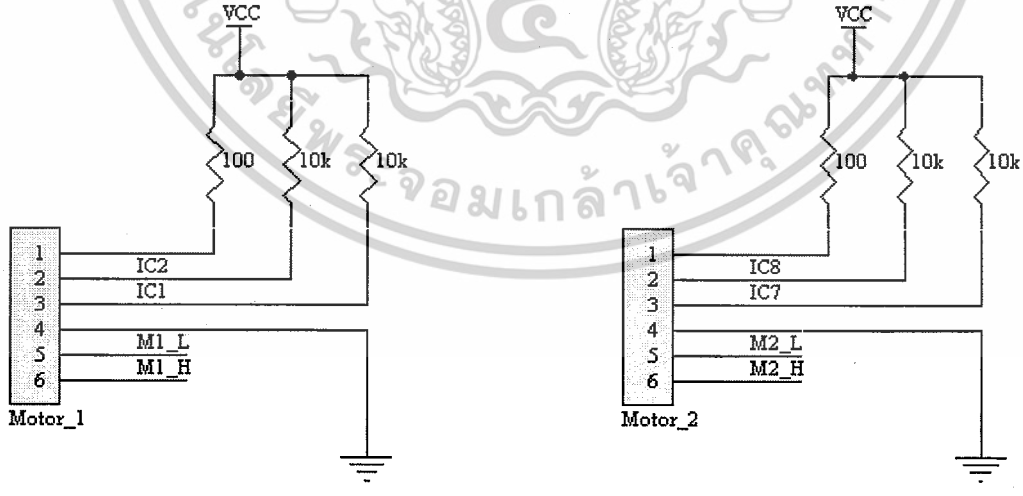
เนื่องจากต้องการแยกไฟเลี้ยงวงขับมอเตอร์ และวงจรไมโครคอนโทรลเลอร์ออกจากกัน ในวงจรขับมอเตอร์ต้องการใช้ไฟเลี้ยง 5 โวลต์ เพื่อเป็นไฟเลี้ยงสำหรับไอซี L298N โดยการแปลงไฟจาก 12 โวลต์ เป็น 5 โวลต์ โดยใช้ไอซีเรกกูเลเตอร์ 5 โวลต์ เบอร์ LM7805 มีวงจรดังรูปที่ 3.16



รูปที่ 3.16 วงจรจ่ายแรงดัน 5 โวลต์

3.2.8 วงจรรับอินพุทจากเอนโคดเดอร์มอเตอร์

เอนโคดเดอร์ที่ใช้ในโครงงานระบบควบคุมอินเวิร์ทเพนดูลัม เป็นเซนเซอร์ออปติคอล ซึ่งขาเอาท์พุทเป็นแบบ Open Collector ต้องต่อตัวต้านทานพูลอัพ (Pullup) เพื่อให้สามารถนำสัญญาณเอาท์พุทไปเข้าไมโครคอนโทรลเลอร์เพื่อใช้นับพัลส์ได้ โดยมีวงจรดังรูปที่ 3.17



รูปที่ 3.17 วงจรรับอินพุทจากเอนโคดเดอร์

3.3 การออกแบบระบบควบคุม

เนื่องจากระบบอินเวิร์ทเพนดูลัมเป็นระบบที่ไม่เป็นเชิงเส้น และไม่มีเสถียรภาพทางพลวัต จึงเป็นปัญหาทางระบบควบคุม ดังนั้นต้องนำทฤษฎีการออกแบบระบบควบคุมของหุ่นยนต์ ที่ได้อธิบายในหัวข้อที่ 2.2 มาใช้ในการจำลองระบบ

สำหรับจุดมุ่งหมายหัวข้อนี้จะแสดงถึงวิธีการควบคุมระบบ ด้วยตัวควบคุมปริภูมิสเตทเชิงเส้น โดยตัวควบคุมจะออกแบบจากแบบจำลองทางพลวัตของหุ่นยนต์อินเวิร์ทเพนดูลัม ในหัวข้อที่ 2.1 ซึ่งในการออกแบบระบบนั้น มีค่าพารามิเตอร์ต่างๆ ที่ต้องใช้ดังตารางที่ 3.1

ตัวพารามิเตอร์	ค่าพารามิเตอร์	หน่วย
ค่าแรงโน้มถ่วง (g)	9.81	เมตร / วินาที^{2}
รัศมีของล้อ (r)	0.01	เมตร
มวลของฐานหุ่นยนต์ที่ติดล้อ (M_w)	0.175	กิโลกรัม
มวลของก้านเพนดูลัม (M_p)	0.81217	กิโลกรัม
โมเมนต์ความเฉื่อยของล้อ (I_w)	0.0000015312	กิโลกรัม · เมตร^2
โมเมนต์ความเฉื่อยของหุ่นยนต์ (I_p)	0.00000085375	กิโลกรัม · เมตร^2
ระยะระหว่างจุดศูนย์กลางล้อถึงจุดศูนย์ถ่วง (l)	0.15	เมตร
ค่าคงที่ของแรงบิด (k_m)	0.0134	โวลต์ · วินาที / เรเดียน
ค่าคงที่ของแรงดันไฟฟ้าต้านกลับ (k_e)	0.0134	โวลต์ · วินาที / เรเดียน
ความต้านทาน (R)	1.9	โอห์ม

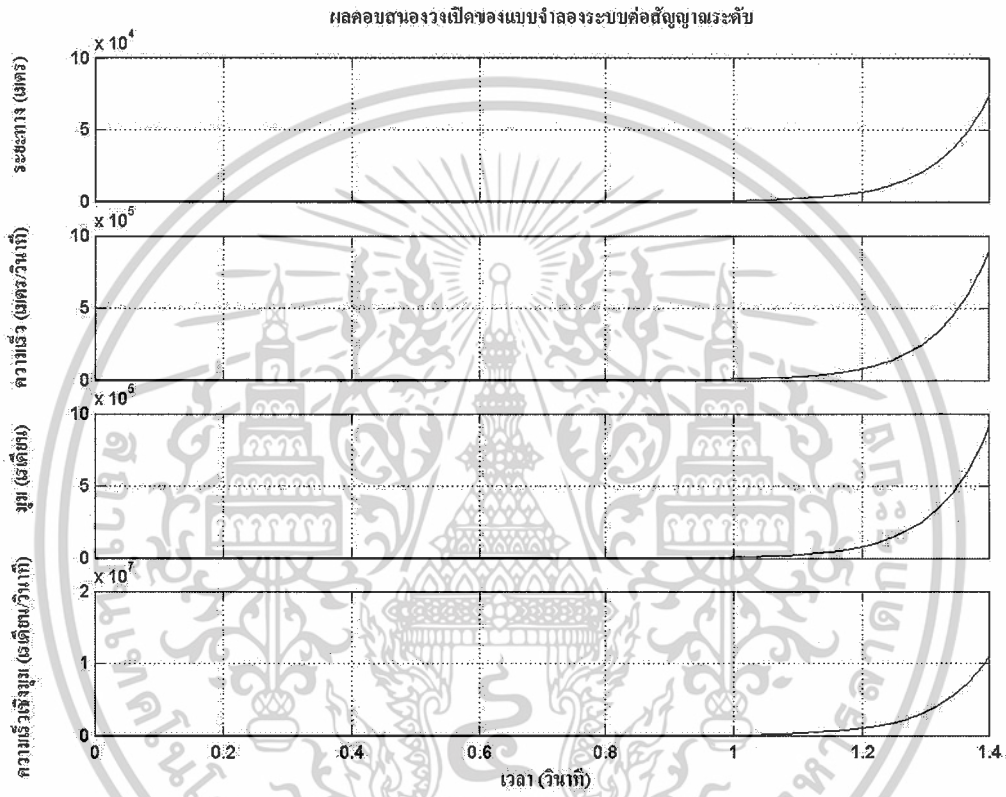
ตารางที่ 3.1 ค่าพารามิเตอร์ของแบบจำลองของหุ่นยนต์อินเวิร์ทเพนดูลัม

เมื่อแทนค่า ในตารางที่ 3.1 ลงในสมการแบบจำลองทางคณิตศาสตร์ในหัวข้อที่ 2.1 จะได้เมตริกซ์ A B และ C ของแบบจำลองหุ่นยนต์อินเวิร์ทเพนดูลัมเพื่อใช้ในการออกแบบระบบควบคุม ดังนี้

$$A=\begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ 0 & -3.3293 & 14.9955 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & -21.0640 & 164.5926 & 0 \end{bmatrix} \quad B=\begin{bmatrix} 0 \\ 2.4903 \\ 0 \\ 15.7559 \end{bmatrix} \quad C=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

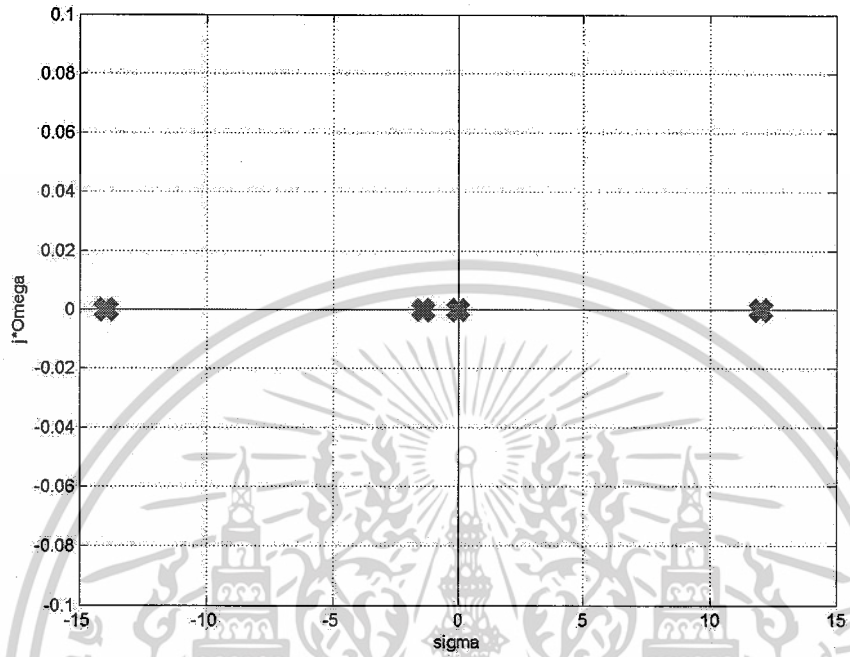
3.3.1 ปัญหาทางระบบควบคุม

เนื่องจากระบบอินเวิร์ทเพนดูลัมเป็นระบบที่ไม่มีเสถียรภาพ ผลตอบสนองของระบบต่อสัญญาณระดับ ทำให้มุมและตำแหน่งของหุ่นยนต์เปลี่ยนแปลงไปอย่างไม่มีขอบเขต ส่งผลให้หุ่นยนต์ไม่สามารถทรงตัวเพื่อรักษาสมดุลอยู่ได้



รูปที่ 3.18 ผลตอบสนองวงเปิดของแบบจำลองระบบต่อสัญญาณระดับ

จากแบบจำลองระบบสามารถนำมาหาโพลวงเปิดของระบบได้เป็นดังนี้ 0, -13.9412, -1.3875, 11.9994 แสดงดังรูปที่ 3.19 ซึ่งจะเห็นว่าระบบไม่มีเสถียรภาพ เนื่องจากมีโพลบางตัวอยู่ทางขวาของระนาบ เมื่อพิจารณาเมตริกซ์ $Q_c = \begin{bmatrix} 0 & 3 & -16 & 540 \\ 3 & -16 & 540 & -4654 \\ 0 & 22 & -193 & 5043 \\ 22 & -103 & 5043 & 37208 \end{bmatrix}$ จะเห็นว่ามีลำดับชั้นเต็ม ดังนั้นสามารถออกแบบระบบควบคุมให้สมบูรณ์ต่อไปได้



รูปที่ 3.19 โพลวงเปิดของระบบ

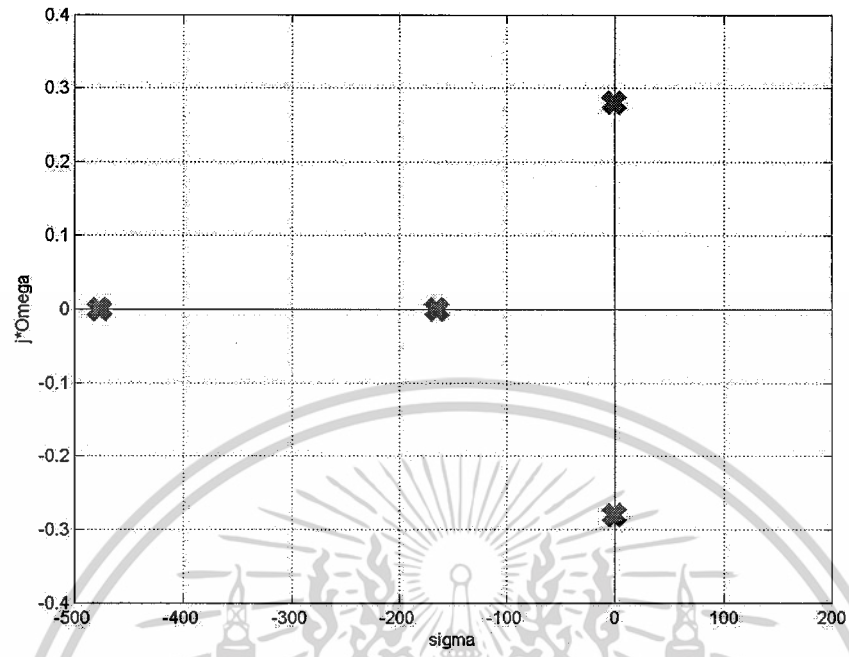
3.3.2 การออกแบบระบบควบคุมด้วยตัวคุมค่ากำลังสองเชิงเส้น

จากทฤษฎีตัวคุมค่ากำลังสองเชิงเส้นซึ่งกล่าวไว้ในหัวข้อที่ 2.2.3 ได้กำหนดค่าในเมตริกซ์ Q และ R ให้มีค่าเป็นนี้

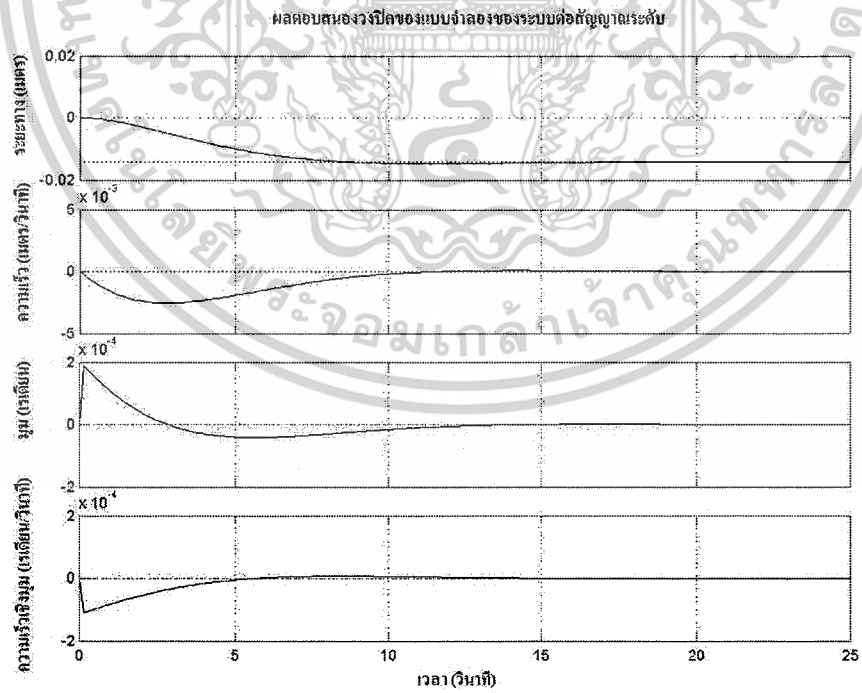
$$Q=\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & y & 0 \\ 0 & 0 & 0 & z \end{bmatrix} \qquad R = 0.01$$

$$;w = 50$$
$$;x = 1$$
$$;y = 25000$$
$$;z = 1$$

เมื่อออกแบบระบบด้วยวิธีตัวคุมค่ากำลังสองเชิงเส้นแล้วจะได้ค่าอัตราขยาย ที่ใช้ในการควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัมซึ่งมีค่า $-316.2, -517.8, 4578.3, 118$ และจะได้โพลวงปิดของระบบเป็น $-476.98, -165.18, -0.28+0.28i, -0.28-0.28i$ แสดงดังรูปที่ 3.20 และผลตอบสนองของการจำลองระบบด้วยอัตราขยายนี้ ดังรูปที่ 3.21



รูปที่ 3.20 โพลวงปิดของระบบเมื่อออกแบบระบบด้วยตัวคุมค่ากำลังสองเชิงเส้น



รูปที่ 3.21 ผลตอบสนองของระบบต่อสัญญาณระดับ เมื่อออกแบบด้วยตัวคุมค่ากำลังสองเชิงเส้น

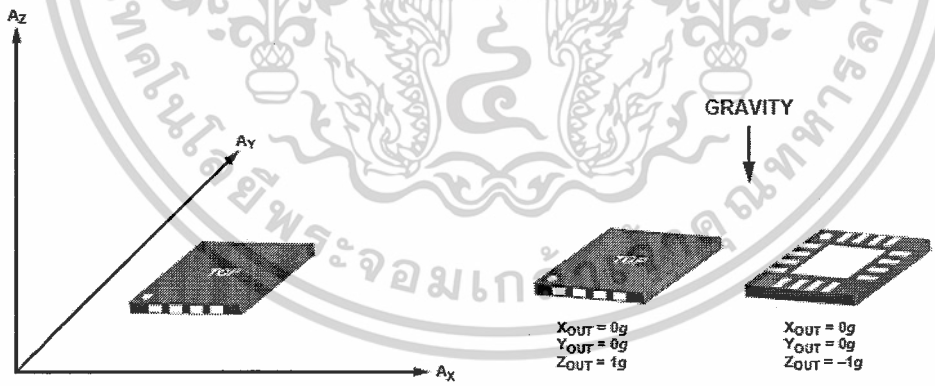
3.4 การออกแบบโปรแกรม

เนื่องจากในโครงงานระบบควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม ใช้ไมโครคอนโทรลเลอร์ dsPIC30F4012 เป็นหน่วยประมวลผลหลัก สามารถพัฒนาโปรแกรมด้วยภาษาซี ได้โดยใช้คอมไพล์เลอร์ MPLAB C30 ของบริษัทไมโครชิพ โดยการพัฒนาโปรแกรมเพื่อให้ง่ายต่อการพัฒนาได้แยกโปรแกรมออกเป็นส่วนต่างๆ ดังต่อไปนี้

- ส่วนการอ่านค่าจากเซนเซอร์ความเร่งและเซนเซอร์ไจโรสโคป
- ส่วนการประมวลผลเซนเซอร์ร่วมกันด้วยแคลแมนฟิลเตอร์
- ส่วนการอ่านค่าจากเอนโคดเดอร์
- ส่วนตัวควบคุมสมดุลหุ่นยนต์อินเวิร์ทเพนดูลัม
- ส่วนการควบคุมมอเตอร์
- ส่วนการแสดงผลบนจอแสดงผลแอลซีดี

3.4.1 ส่วนการอ่านค่าจากเซนเซอร์ความเร่งและเซนเซอร์ไจโรสโคป

การอ่านค่าจากเซนเซอร์ความเร่งและเซนเซอร์ไจโรสโคป สามารถทำได้โดยใช้หน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลบนไมโครคอนโทรลเลอร์ ความละเอียด 10 บิต ซึ่งกำหนดให้ใช้ แรงดันอ้างอิงบวก ที่ 3.3 โวลต์ และแรงดันอ้างอิงลบ ที่ 0 โวลต์หรือระดับกราวด์ โดยการอ่านค่าจากเซนเซอร์ทำได้โดย

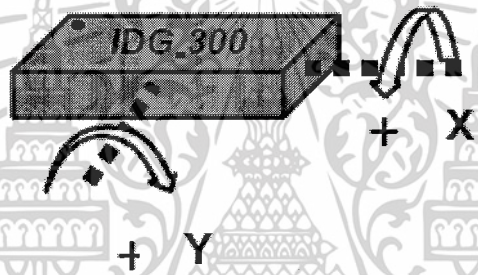


รูปที่ 3.22 ทิศทางแกนของเซนเซอร์ความเร่ง

จากรูปที่ 3.22 เมื่ออ่านค่าของเซนเซอร์ความเร่ง แกน X ที่มีค่าความเร่งเท่ากับ 0 จี สัญญาณเอาท์พุทมีค่าเท่ากับ 1.65 โวลต์ เมื่อแปลงค่าโดยใช้หน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลจะมีค่าเท่ากับ 512 หน่วย และแกน Z มีค่าความเร่งเท่ากับ 1 จี หรือเท่ากับ 1.95 โวลต์ หรือ 605 หน่วยเพื่อให้ง่ายต่อการคำนวณจึงนำค่า 512 หน่วยลบออกจากค่าที่อ่านได้ เพื่อให้ความเร่งที่ 0

จี มีค่าเท่ากับ 0 หน่วย โดยค่าแรงดันจะเปลี่ยนแปลงค่าเป็นบวกหรือลบ ขึ้นกับทิศของความเร่ง ดังนั้นค่าที่แกน X มีค่าเท่ากับหน่วย และ แกน Z มีค่าเท่ากับ – 93 หน่วย เมื่อคำนวณด้วยฟังก์ชัน atan2(-z, x) ซึ่งฟังก์ชัน atan2() ในไลบารี่ math.h มาตรฐานของคอมไพล์เลอร์ภาษาซี ซึ่งให้ค่าออกมามีหน่วยเป็น เรเดียน ตั้งแต่ π ถึง $-\pi$ มุมที่ได้มีค่าเท่ากับ 0 เรเดียน

เซนเซอร์ไจโรสโคปเมื่อไม่มีการหมุน สามารถอ่านค่าแรงดันเอาท์พุท ได้ 1.65 โวลต์ หรือเมื่อใช้หน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอลอ่านค่าจะมีค่าเท่ากับ 512 หน่วย คุณสมบัติของเซนเซอร์ไจโรสโคปเบอร์ IDG300 มีค่าความแม่นยำ 2 มิลลิโวลต์/องศา/วินาที ค่าที่อ่านได้เป็นค่าความเร็วเชิงมุม เพื่อให้ง่ายกับการใช้งานจึงนำค่า 512 ลบออกจากค่าที่อ่านได้จากหน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอล โดยเมื่อไม่มีความเร็วเชิงมุม สามารถอ่านค่าได้เท่ากับ 0 หน่วย และจะมีค่าเป็นบวกหรือลบ ตามทิศทางการหมุนของเซนเซอร์ ดังรูปที่ 3.23



รูปที่ 3.23 ทิศทางการหมุนของเซนเซอร์ไจโรสโคป

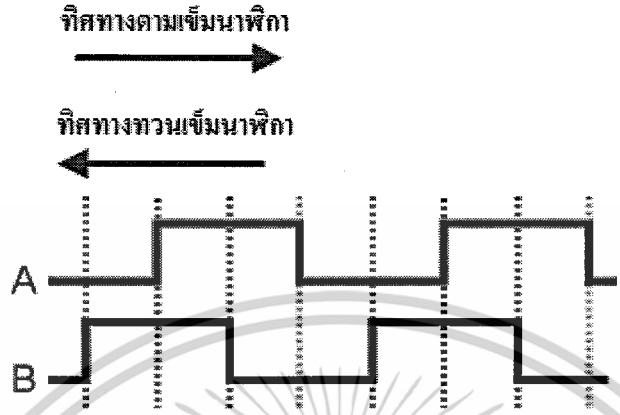
3.4.2 ส่วนการประมวลผลเซนเซอร์ร่วมกันด้วยแคลแมนฟิลเตอร์

ในส่วนนี้ได้นำค่าของเซนเซอร์ความเร่ง และเซนเซอร์ไจโรสโคปมาประมวลผลร่วมกันด้วยแคลแมนฟิลเตอร์ เพื่อประมาณสเตทล่วงหน้าให้กับระบบด้วยสมการทางคณิตศาสตร์ จากนั้นเมื่อระบบสามารถตรวจวัดและหาค่าสเตทปัจจุบันได้ จึงนำค่าที่ได้จากการประมาณ และค่าสเตทจริงของระบบมาเปรียบเทียบกัน เพื่อคำนวณอัตราขยายของแคลแมนฟิลเตอร์ และนำไปชดเชยให้กับการประมาณสเตทในครั้งต่อไป โดยอัตราขยายของแคลแมนฟิลเตอร์จะแปรผันไปตามค่าแตกต่างของการประมาณ และสเตทจริงของระบบ ทำให้การประมาณสเตทในครั้งต่อไปของระบบสามารถเข้าใกล้สเตทจริงของระบบมากขึ้น และทำให้ระบบมีเสถียรภาพที่ดีขึ้น

3.4.3 ส่วนการอ่านค่าจากเอนโคดเดอร์

เอนโคดเดอร์ในโครงงานระบบควบคุมอินเวิร์ทเพนดูลัม เป็นแบบอินคลีเมนท์เอนโคดเดอร์ มีเอาท์พุทเป็นสัญญาณพัลส์สี่เหลี่ยม 2 เฟสคือเฟส A และ เฟส B ต่างกัน 90 องศา

โดยเฟส A นำเฟส B เมื่อมอเตอร์หมุนตามเข็มนาฬิกา และเฟส B นำเฟส A เมื่อมอเตอร์หมุนทวนเข็มนาฬิกา ดังรูปที่ 3.24



รูปที่ 3.24 สัญญาณเอาท์พุทของเอนโคดเดอร์

ในโครงงานนี้ใช้หน่วยตรวจจับสัญญาณดิจิตอลของไมโครคอนโทรลเลอร์ในการอ่านค่าพัลส์ของเอนโคดเดอร์ โดยต่อเอนโคดเดอร์ เฟส A เข้ากับขาของหน่วยตรวจจับสัญญาณดิจิตอล และ เฟส B ต่อกับขาอินพุทดิจิตอลปกติ ด้วยโปรแกรมใช้งานหน่อยตรวจจับสัญญาณดิจิตอลในโหมดอินเตอร์รัปส์ขอบขาขึ้นของสัญญาณดิจิตอล เมื่อเกิดอินเตอร์รัปส์ขึ้น โปรแกรมจะตรวจสอบอินพุทเฟส B ถ้าเท่ากับ '0' แสดงว่า เฟส A นำ เฟส B แต่ถ้าอินพุทเท่ากับ '1' แสดงว่าเฟส B นำเฟส A และนับค่าพัลส์เก็บไว้เพื่อนำไปคำนวณหาค่าระยะทางจาก

$$x = n \times ppr \tag{3.4.1}$$

โดยที่ x คือระยะทาง n คือจำนวนพัลส์ และ ppr คือจำนวนพัลส์ต่อรอบ เมื่อได้ระยะทางมาแล้ว สามารถหาความเร็วได้จาก

$$\dot{x} = \frac{x - x\_old}{dt} \tag{3.4.2}$$

โดยที่ $\dot{x}$ คือความเร็ว x_old คือระยะทางครั้งที่แล้ว และ dt คือเวลา

3.4.4 ส่วนการควบคุมสมดุลหุ่นยนต์อินเวิร์ทเพนดูลัม

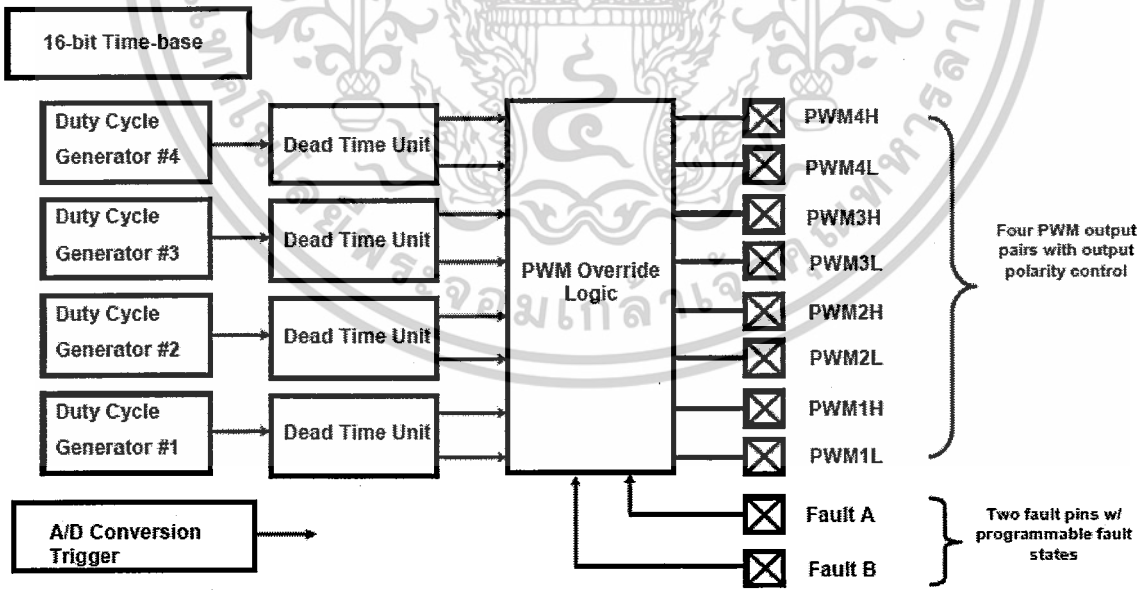
เป็นส่วนที่ประมวลผลหาสัญญาณพัลส์วิทมอดูเลชันที่ใช้กับมอเตอร์ เพื่อรักษาสมดุลของหุ่นยนต์อินเวิร์ทเพนดูลัม โดยการนำค่าอินพุทคือ ระยะทาง x ความเร็ว $\dot{x}$ มุม θ ความเร็วเชิงมุม $\dot{\theta}$ มาคูณกับค่าอัตราขยาย Kx $K\dot{x}$ $K\theta$ และ $K\dot{\theta}$ ดังนี้

$$command = (Kx * x) + (K\dot{x} * \dot{x}) + (K\theta * \theta) + (K\dot{\theta} * \dot{\theta}) \qquad (3.4.3)$$

โดยอินพุท ($Command$) คือค่าความกว้างพัลส์ของสัญญาณพัลส์วิทมอดูเลชัน สำหรับควบคุมความเร็วมอเตอร์ มีค่าทั้งเป็นค่าบวกและเป็นค่าลบ ซึ่งหมายถึงทิศทางการหมุนของมอเตอร์

3.4.5 ส่วนการควบคุมมอเตอร์

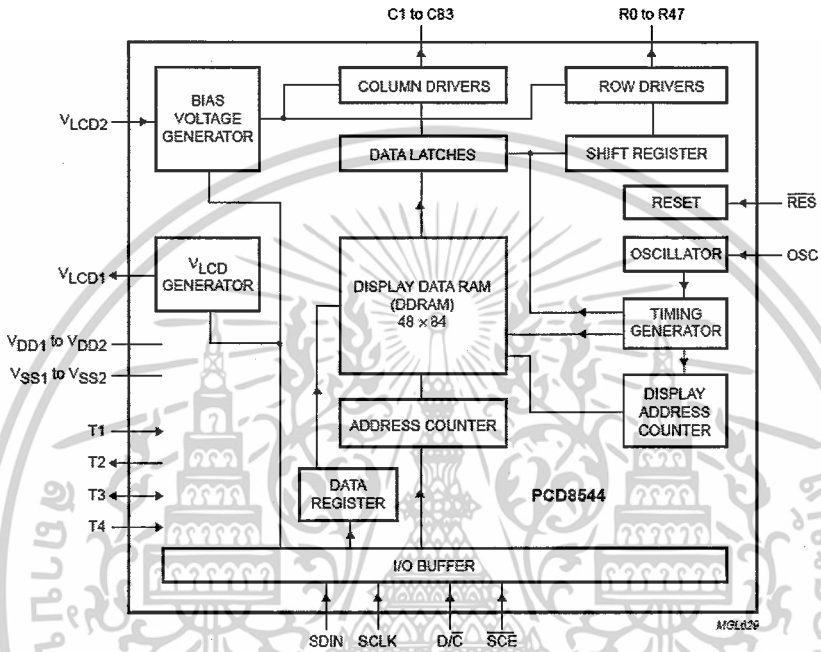
การควบคุมมอเตอร์ในโครงงานนี้ใช้หน่วยกำเนิดสัญญาณพัลส์วิทมอดูเลชันบนไมโครคอนโทรลเลอร์ ดังรูปที่ 3.25 โดยใช้หน่วยสร้างสัญญาณพัลส์วิทมอดูเลชันจำนวนสองชุดสำหรับควบคุมมอเตอร์ 2 ตัว คือ PWM1 และ PWM2 โดยโปรแกรมให้สร้างสัญญาณพัลส์วิทมอดูเลชันความถี่ 20 กิโลเฮิร์ทซ์ และควบคุมช่องสัญญาณออกของสัญญาณพัลส์วิทมอดูเลชัน ช่อง PWMxL และ PWMxH เพื่อควบคุมทิศทางการหมุนของมอเตอร์



รูปที่ 3.25 แผนผังหน่วยกำเนิดสัญญาณพัลส์วิทมอดูเลชัน

3.4.6 ส่วนการแสดงผลบนจอแสดงผลแอลซีดี

ในโครงงานนี้ต้องการแสดงผลค่าเอาท์พุทต่างๆ เพื่อความง่ายในการวิเคราะห์การทำงาน จึงทำให้มีการนำเอาจอแอลซีดีแบบกราฟฟิกมาใช้แสดงผล เนื่องจากว่าสามารถแสดงผลได้ครบถ้วน โดยจอแอลซีดีที่นำมาใช้ในโครงงานนี้เป็นจอแอลซีดีของมือถือ โดยใช้ ไอซีควบคุมเบอร์ PCD8544 ดังรูปที่ 3.26 ของบริษัทฟิลิปส์ มีการติดต่อแบบ SPI



รูปที่ 3.26 แผนผังภายในของไอซี PCD8544

ซึ่งจากโปรแกรมส่วนต่างๆ สามารถนำมารวมกันเป็นโปรแกรมควบคุมสมดุลอินเวิร์ทเพนดูลัมโดยการทำงานของโปรแกรมแสดงดังรูปที่ 5.5 เริ่มต้นการทำงานโปรแกรมจะตั้งค่าหน่วยการทำงานต่างๆ จากนั้นเข้าสู่รอบการทำงาน โดยใช้ไทเมอร์เป็นตัวกำหนดเวลาของรอบการประมวลผล โดยกำหนดไทเมอร์ทำงานในโหมดอินเตอร์รัปส์ให้อินเตอร์รัปส์ทุกๆ 10 มิลลิวินาที โดยเริ่มการประมวลผลจากไมโครคอนโทรลเลอร์อ่านค่าอัตราเร็วเชิงมุมจากเซนเซอร์ไจโรสโคป และมุมเบี่ยงเบนจากจุดสมดุลจากเซนเซอร์ความเร่ง จากนั้นนำค่าจากเซนเซอร์ทั้งสองมาประมวลผลร่วมกันด้วยแคลแมนฟิลเตอร์ เพื่อให้ได้ค่ามุมเบี่ยงเบนที่ใกล้เคียงมุมเบี่ยงเบนจริง หลังจากนั้นอ่านค่าเอนโคดเดอร์ เพื่อนำมาคำนวณหาค่าระยะทางและความเร็ว แล้วจึงนำค่าทั้งหมดที่ได้มาประมวล ผลโดยคูณกับค่าอัตราขยาย เพื่อหาค่าความเร็วมอเตอร์ สำหรับควบคุมสมดุลของหุ่นยนต์อินเวิร์ทเพนดูลัม โดยการสร้างสัญญาณพัลส์วิทมอดูเลชันที่มีขนาดความกว้างพัลส์ตามที่คำนวณได้จากส่วนของการควบคุมแล้วส่งไปยังวงจรขับมอเตอร์ และแสดงค่าผลต่างๆบนจอ

แอลซีดี เมื่อเสร็จสิ้นกระบวนการทั้งหมดแล้วจึงกลับไปเริ่มต้นรอบการทำงานใหม่อีกครั้ง โดยมีแผนผังการทำงานดังรูปที่ 3.27

รูปที่ 3.27 แผนผังการทำงานของโปรแกรม

บทที่ 4

การทดลองและผลการทดลอง

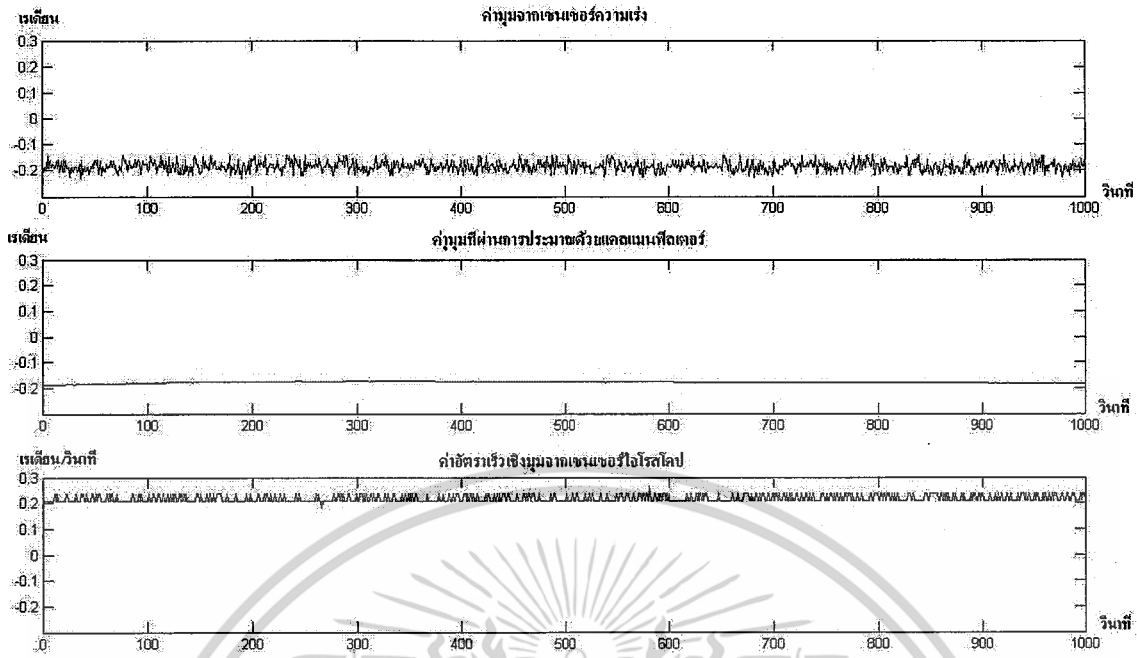
ในบทนี้นำเสนอการทดลองของโครงงานระบบควบคุมอินเวิร์ทเพนดูลัม ซึ่งแบ่งออกเป็นการทดลองหาค่าศูนย์ของเซนเซอร์ การทดลองหาโมเมนต์ความเฉื่อย และการทดลองควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม โดยในการทดลองใช้คาบการชักตัวอย่าง 10 มิลลิวินาที และเลือกใช้ค่า S_w และ S_v ของแคลแมนฟิลเตอร์เป็น $S_w = [0.00001 \quad 0.0003]$ และ $S_v = 0.69$ ในทุกๆ ตัวอย่างการทดลอง

4.1 การทดลองหาค่าศูนย์ของเซนเซอร์

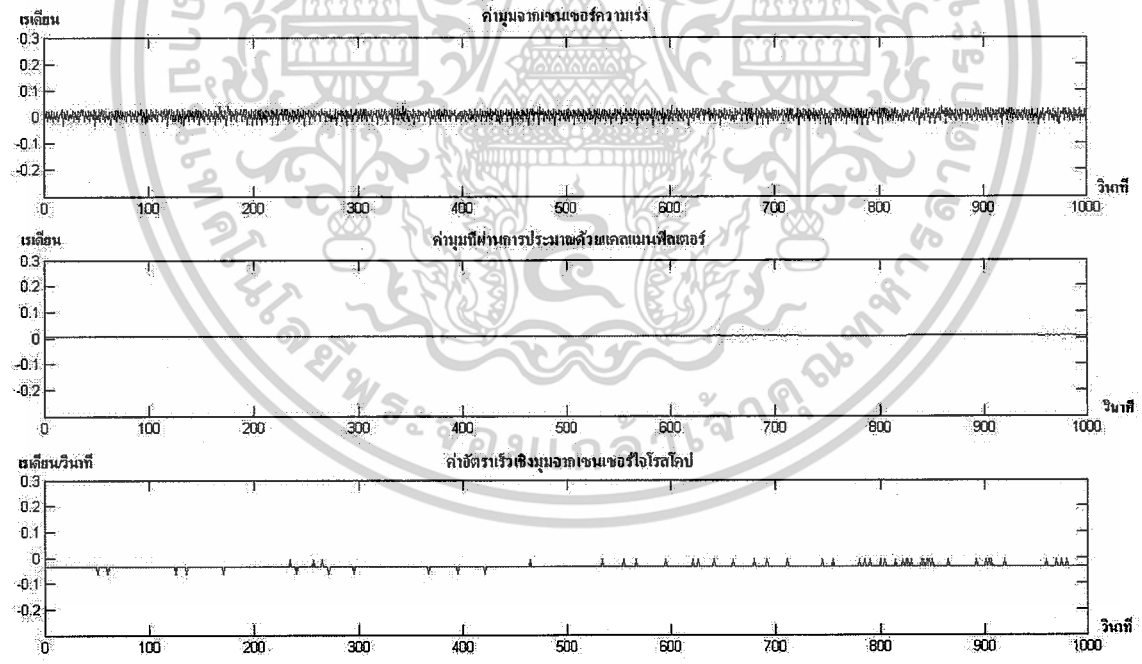
เนื่องจากในการทดลองควบคุมสมดุลของหุ่นยนต์อินเวิร์ทเพนดูลัม พบว่าขณะที่หุ่นยนต์ตั้งตรง และยังไม่มีการเคลื่อนที่เซนเซอร์ไจโรสโคป และเซนเซอร์ความเร่ง มีการแกว่งของค่าวัดและค่าวัดไม่เข้าสู่ศูนย์ ซึ่งเมื่อนำมาทดลอง จึงทำให้เกิดความผิดพลาด หุ่นยนต์อินเวิร์ทเพนดูลัมไม่สามารถทรงตัวได้ดีเท่าที่ควร ดังนั้นต้องทำการทดลองหาค่าศูนย์ของเซนเซอร์ เพื่อหาค่าชดเชยความผิดพลาดจากการวัดของเซนเซอร์ โดยทดลองเก็บค่าจากเซนเซอร์ทั้งสอง 5 ครั้งดังตารางที่ 4.1 เพื่อนำมาเขียนกราฟ และหาค่าเฉลี่ยด้วยโปรแกรม MATLAB โดยได้ค่าของตัวชดเชยของเซนเซอร์จากการเฉลี่ยทั้ง 5 ครั้ง เท่ากับ -0.1854 และ 0.2145 กับสำหรับเซนเซอร์ความเร่ง และเซนเซอร์ไจโรสโคป ตามลำดับ ซึ่งค่าที่ได้จะถูกนำมาบวก ลบกับค่าวัดทำให้ค่าวัดถูกต้องยิ่งขึ้น ตัวอย่างก่อนและหลังการปรับปรุงแสดงดังรูปที่ 4.1 และ 4.2

ครั้งที่	เซนเซอร์ไจโรสโคป (rad/s)	เซนเซอร์ความเร่ง (rad)	มุมจากการประมาณด้วยแคลแมนฟิลเตอร์ (rad)
1	0.2181	-0.1846	-0.1789
2	0.2178	-0.1859	-0.1822
3	0.2192	-0.1871	-0.1815
4	0.2127	-0.1846	-0.1814
5	0.2049	-0.1847	-0.1851
เฉลี่ย	0.2145	-0.1854	0.1818

ตารางที่ 4.1 ตัวอย่างผลการเก็บค่าวัดจากเซนเซอร์



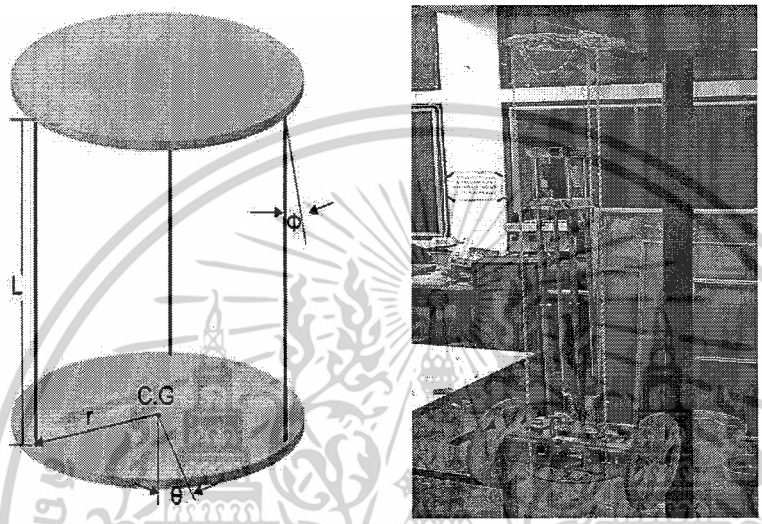
รูปที่ 4.1 ค่าวัดของเซนเซอร์ก่อนการชดเชย



รูปที่ 4.2 ค่าวัดของเซนเซอร์หลังการชดเชย

4.2 การทดลองเพื่อหาโมเมนต์ความเฉื่อย

ในการออกแบบเพื่อหาตัวควบคุมให้กับหุ่นยนต์อินเวิร์ทเพนดูลัม มีค่าพารามิเตอร์ที่ต้องแทนด้วยค่าคงที่ ซึ่งส่วนมากได้จากการวัด แต่สำหรับค่าโมเมนต์ความเฉื่อยเป็นค่าคงที่ ที่ไม่สามารถวัดได้โดยตรง จึงต้องทดลองหาค่าโมเมนต์ความเฉื่อย โดยการสร้างอุปกรณ์ที่เรียกว่า ทรีสตริงทอร์ชันแนลเพนดูลัม (Three-string torsional pendulum)



รูปที่ 4.3 แบบจำลองของฐานและตัวอย่างการทดลอง

กำหนดให้

m_{sc} คือ มวลของหุ่นยนต์อินเวิร์ทเพนดูลัม

m_p คือ มวลของแผ่นอุปกรณ์ทดลอง

I_{sc} คือ โมเมนต์ความเฉื่อยของหุ่นยนต์อินเวิร์ทเพนดูลัม

I_p คือ โมเมนต์ความเฉื่อยของแผ่นอุปกรณ์ทดลอง

r คือ ระยะระหว่างจุดศูนย์ถ่วงแผ่นอุปกรณ์ทดลองไปยังเชือก

θ คือ มุมของการเคลื่อนที่แผ่นอุปกรณ์ทดลอง

l คือ ความยาวเชือก

ϕ คือ มุมเชือกที่เคลื่อนที่

τ คือ ในการแกว่ง

จากสมการการหาโมเมนต์ความเฉื่อย

$$\sum M_z = -r(m_{sc} + m_p)g\sin\phi = (I_{sc} + I_p)\ddot{\theta} \tag{4.2.1}$$

เมื่อกำหนดให้มุมเชือกมีค่าน้อยเพื่อให้เป็นเชิงเส้น

$$\phi = \frac{r}{l}\theta \tag{4.2.2}$$

แทนค่า (4.2.2) ลงใน (4.2.1) จะได้

$$\ddot{\theta} + \frac{(m_o + m_p)gr^2}{(l_o + l_p)l}\theta = 0 \tag{4.2.3}$$

จะได้ความสัมพันธ์ของโมเมนต์ความเฉื่อย และคาบการแกว่งในสมการ

$$l_{sc} + l_p = \frac{(m_o + m_p)r^2 g}{l}\left(\frac{\tau}{2\pi}\right)^2 \tag{4.2.4}$$

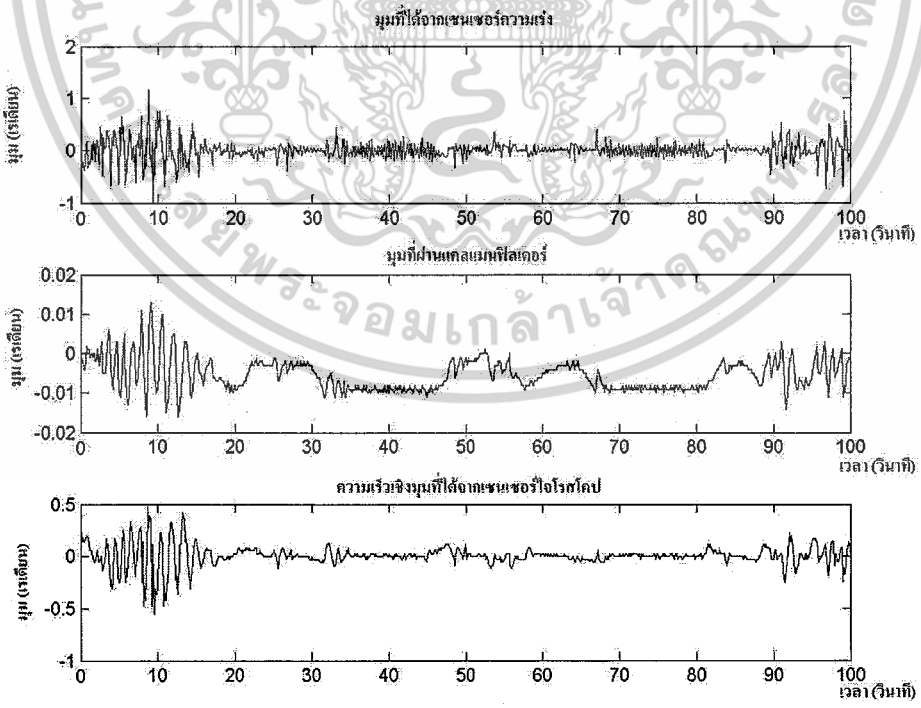
ในการทดลองหาค่าโมเมนต์ความเฉื่อยของหุ่นยนต์นั้น ทำโดยนำหุ่นยนต์ไปตั้งบนฐานที่แขวนด้วยเชือกยาว 92 เซนติเมตรดังรูปที่ 4.3 และออกแรงผลักเพื่อให้เกิดการแกว่งพร้อมทั้งจับเวลา เพื่อหาคาบการแกว่ง โดยให้มุม ϕ มีค่าน้อยๆ โดยทดลองทั้งหมด 5 ครั้ง ได้ผลดังตารางที่ 4.2 จากนั้นคำนวณตามสมการที่ (4.2.4) จะได้ค่า โมเมนต์ความเฉื่อยตามต้องการคือ 0.00000085375 kg•m^2

ครั้งที่	คาบการแกว่ง (ครั้ง / นาที)
1	31
2	30
3	28
4	31
5	31
เฉลี่ย	30.2

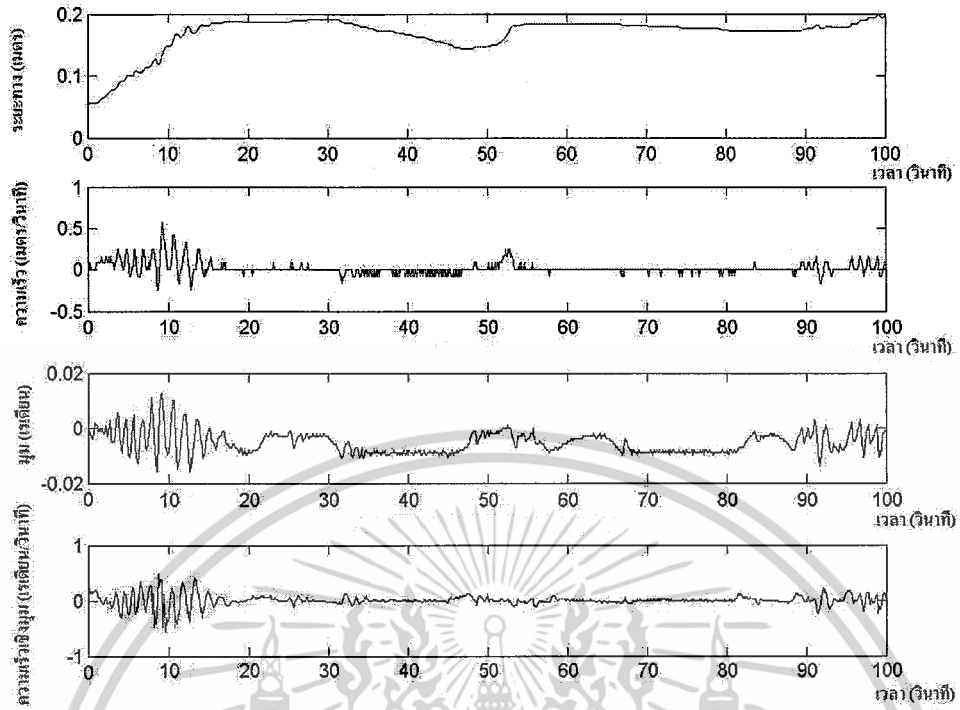
ตารางที่ 4.2 ผลการทดลองหาโมเมนต์ความเฉื่อย

4.3 การทดลองควบคุมระบบหุ่นยนต์อินเวิร์ทเพนดูลัม

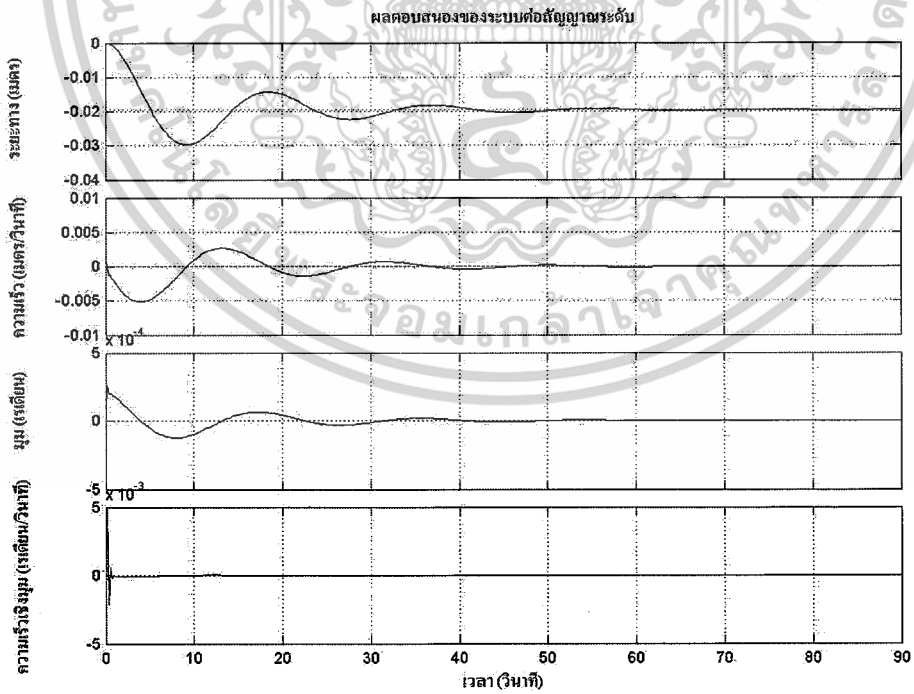
ในการทดลองหาผลตอบสนอง ได้ทดลองอ่านค่าจากเซนเซอร์ทั้งสองตัว คือ เซนเซอร์ไจโรสโคป และเซนเซอร์ความเร่ง โดยนำค่ามุมที่อ่านได้มาเปรียบเทียบกับค่ามุมที่ผ่านแคลแมนฟิลเตอร์ ซึ่งจะเห็นว่าค่ามุมที่ผ่านแคลแมนฟิลเตอร์จะมีความนิ่ง และเกิดการสั่นของสัญญาณที่น้อยกว่า ค่ามุมที่อ่านจากเซนเซอร์ความเร่ง ดังรูปที่ 4.4 ดังนั้นในการพิจารณาผลตอบสนองของระบบ จะนำค่ามุมที่ผ่านจากแคลแมนฟิลเตอร์ มาพิจารณา และในการทดลองหาผลตอบสนองของระบบจากการออกแบบด้วยตัวควบคุมแบบตัวคุมค่ากำลังสองเชิงเส้น พบว่าหลังจากนำค่าอัตราขยายจากการออกแบบในหัวข้อที่ 2.2.3 มาโปรแกรมให้กับหุ่นยนต์อินเวิร์ทเพนดูลัม หุ่นยนต์ไม่สามารถทรงตัวอยู่ได้ จึงทดลองปรับแต่งค่าอัตราขยายใหม่ เพื่อให้หุ่นยนต์สามารถทรงตัวอยู่ได้ พบว่าค่าอัตราขยายที่ทำให้หุ่นยนต์สามารถทรงตัวอยู่ได้นั้น ต่างไปจากค่าที่คำนวณได้จากโปรแกรม MATLAB ซึ่งเป็นผลมาจากค่าความผิดพลาดของเซนเซอร์ และแบบจำลองทางคณิตศาสตร์ของระบบ แต่ค่าอัตราขยายก็ยังมีแนวโน้มไปในทางเดียวกัน คือ ค่าอัตราขยายของค่ามุม จะมีค่ามากที่สุด ซึ่งเป็นไปตามหลักของการออกแบบ โดยค่าอัตราขยายนั้นคือ $K=[31,-51,4578,11]$ เพื่อเปรียบเทียบค่าอัตราขยายที่ได้จากการปรับแต่ง กับค่าอัตราขยายที่คำนวณได้จากโปรแกรม MATLAB มาป้อนให้กับระบบเพื่อหาผลตอบสนองของระบบด้วยโปรแกรม MATLAB จึงได้ผลตอบสนองดังรูปที่ 4.6 และผลตอบสนองของสเตทจริงจากระบบดังรูปที่ 4.5



รูปที่ 4.4 ค่าวัดจากเซนเซอร์ของหุ่นยนต์

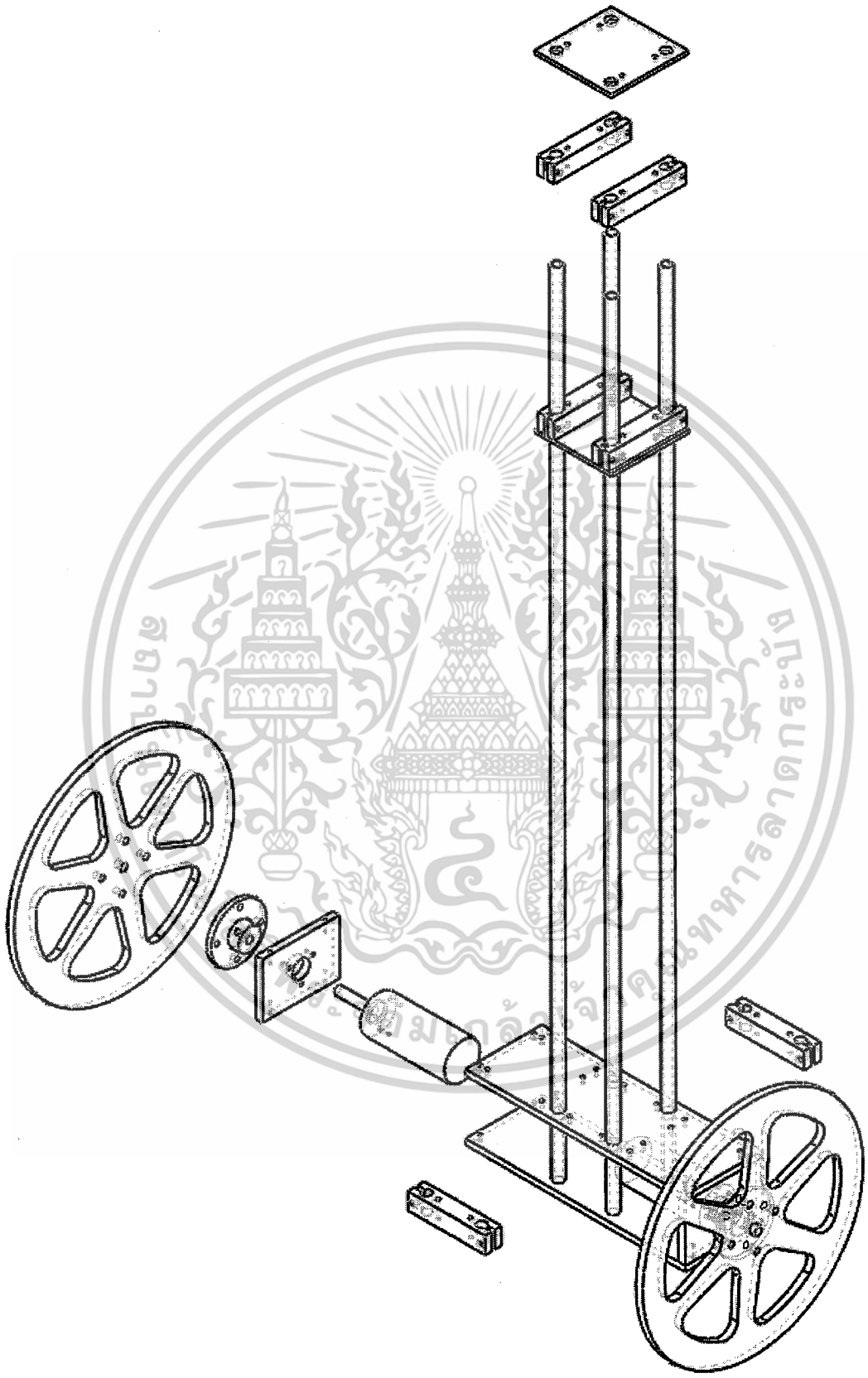


รูปที่ 4.5 ผลตอบสนองของตัวแปรสเตทจากหุ่นยนต์



รูปที่ 4.6 ผลตอบสนองของตัวแปรสเตทจากโปรแกรม MATLAB

ก.8 การประกอบชิ้นส่วนต่างๆ



รูปที่ ก.8 การประกอบชิ้นส่วนต่างๆ

ภาคผนวก ข

เอกสารคู่มืออุปกรณ์อิเล็กทรอนิกส์

ข.1 เอกสารคู่มือการใช้งาน dsPIC30F4012

ไอซี dsPIC30F4012 เป็นไอซีไมโครคอนโทรลเลอร์ มีรายละเอียดต่างๆดังนี้

MICROCHIP

dsPIC30F4011/4012

dsPIC30F4011/4012 Enhanced Flash 16-Bit Digital Signal Controller

Note: This data sheet summarizes features of this group of dsPIC30F devices and is not intended to be a complete reference source. For more information on the CPU, peripherals, register descriptions and general device functionality, refer to the *"dsPIC30F Family Reference Manual"* (DS70046). For more information on the device instruction set and programming, refer to the *"dsPIC30F/33F Programmer's Reference Manual"* (DS70157).

High-Performance, Modified RISC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set architecture with flexible addressing modes
- 83 base instructions
- 24-bit wide instructions, 16-bit wide data path
- 48 Kbytes on-chip Flash program space (16K instruction words)
- 2 Kbytes of on-chip data RAM
- 1 Kbyte of nonvolatile data EEPROM
- Up to 30 MIPS operation:
  - DC to 40 MHz external clock input
  - 4 MHz-10 MHz oscillator input with PLL active (4x, 8x, 16x)
- 30 interrupt sources:
  - 3 external interrupt sources
  - 8 user-selectable priority levels for each interrupt source
  - 4 processor trap sources
- 16 x 16-bit working register array

DSP Engine Features:

- Dual data fetch
- Accumulator write-back for DSP operations
- Modulo and Bit-Reversed Addressing modes
- Two, 40-bit wide accumulators with optional saturation logic
- 17-bit x 17-bit single-cycle hardware fractional/integer multiplier
- All DSP instructions are single cycle
- ±16-bit, single-cycle shift

Peripheral Features:

- High-current sink/source I/O pins: 25 mA/25 mA
- Timer module with programmable prescaler:
  - Five 16-bit timers/counters; optionally pair 16-bit timers into 32-bit timer modules
- 16-bit Capture input functions
- 16-bit Compare/PWM output functions
- 3-wire SPI modules (supports 4 Frame modes)
- I^2C™ module supports Multi-Master/Slave mode and 7-bit/10-bit addressing
- 2 UART modules with FIFO Buffers
- 1 CAN module, 2.0B compliant

Motor Control PWM Module Features:

- 6 PWM output channels:
  - Complementary or Independent Output modes
  - Edge and Center-Aligned modes
- 3 duty cycle generators
- Dedicated time base
- Programmable output polarity
- Dead-time control for Complementary mode
- Manual output control
- Trigger for A/D conversions

Quadrature Encoder Interface Module Features:

- Phase A, Phase B and Index Pulse input
- 16-bit up/down position counter
- Count direction status
- Position Measurement (x2 and x4) mode
- Programmable digital noise filters on inputs
- Alternate 16-Bit Timer/Counter mode
- Interrupt on position counter rollover/underflow

dsPIC30F4011/4012

Analog Features:

- 10-Bit Analog-to-Digital Converter (A/D) with 4 S/H inputs:
  - 1 Msps conversion rate
  - 9 input channels
  - Conversion available during Sleep and Idle
- Programmable Brown-out Reset

Special Digital Signal Controller Features:

- Enhanced Flash program memory:
  - 10,000 erase/write cycle (min.) for industrial temperature range, 100K (typical)
- Data EEPROM memory:
  - 100,000 erase/write cycle (min.) for industrial temperature range, 1M (typical)
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)

Special Digital Signal Controller Features (Cont.):

- Flexible Watchdog Timer (WDT) with on-chip, low-power RC oscillator for reliable operation
- Fail-Safe Clock Monitor operation detects clock failure and switches to on-chip, low-power RC oscillator
- Programmable code protection
- In-Circuit Serial Programming™ (ICSP™)
- Selectable Power Management modes:
  - Sleep, Idle and Alternate Clock modes

CMOS Technology:

- Low-power, high-speed Flash technology
- Wide operating voltage range (2.5V to 5.5V)
- Industrial and Extended temperature ranges
- Low-power consumption

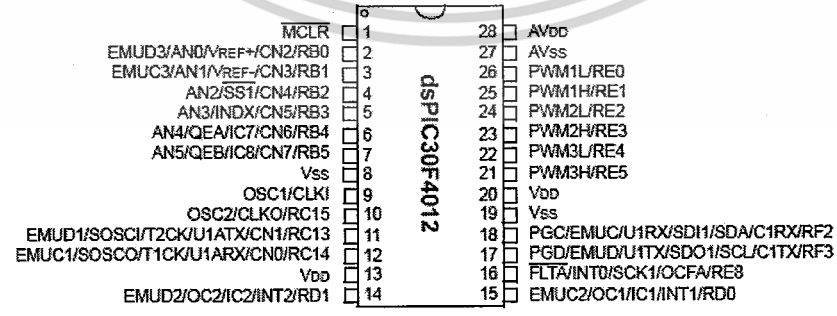
dsPIC30F Motor Control and Power Conversion Family*

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

* This table provides a summary of the dsPIC30F6010 peripheral features. Other available devices in the dsPIC30F Motor Control and Power Conversion Family are shown for feature comparison.

Pin Diagrams

28-Pin SPDIP and SOIC



ข.2 เอกสารคู่มือการใช้งาน IDG300

ไอซี IDG300 เป็น ไอซีเซนเซอร์ไจโรสโคป มีรายละเอียดต่างๆ ดังนี้

InvenSense

Integrated Dual-Axis Gyro

IDG-300

FEATURES

- Integrated X- and Y-axis gyro on a single chip
- Factory trimmed full scale range of ±500°/sec
- Integrated low-pass filters
- High vibration rejection over a wide frequency range
- High cross-axis isolation by design
- 3V single supply operation
- 5000 *g* shock tolerance
- RoHS compliant (completely lead free)
- 6 x 6 x 1.5mm QFN package

APPLICATIONS

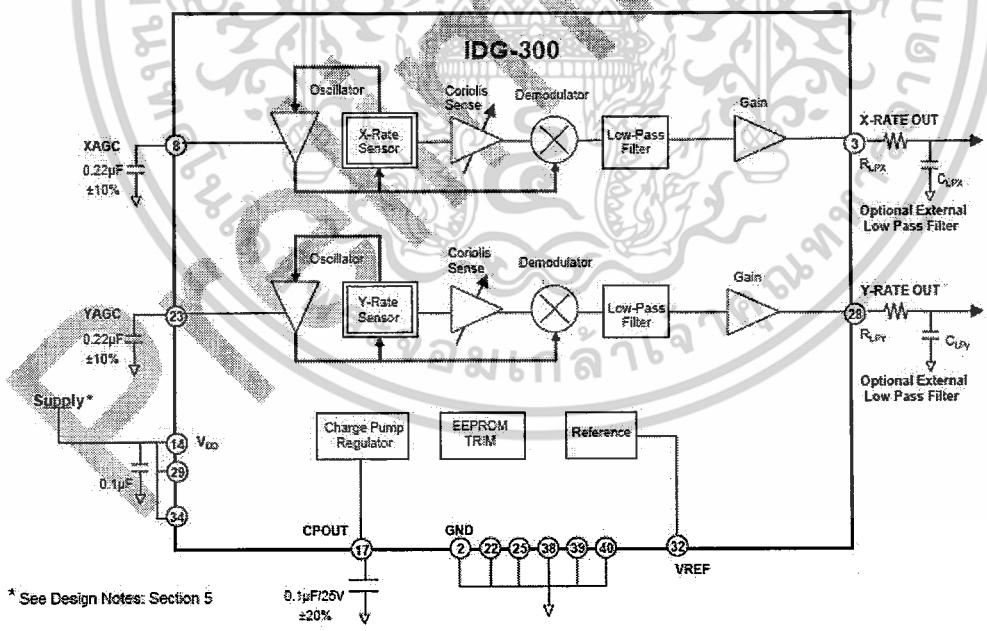
- Inertial measurement units (IMUs)
- Handheld GPS navigation devices
- Radio controlled helicopters
- Toys and game consoles
- Robotic and power tools
- Antenna positioning
- Remote control

GENERAL DESCRIPTION

The IDG-300 is an integrated dual-axis angular rate sensor (gyroscope). It uses InvenSense's proprietary and patented MEMS technology with vertically driven, vibrating masses to make a functionally complete, low-cost, dual-axis angular rate sensor. All required electronics are integrated onto a single chip with the sensor.

The IDG-300 gyro uses two sensor elements with novel vibrating dual-mass bulk silicon configurations that sense the rate of rotation about the X- and Y-axis (in-plane sensing). This results in a unique, integrated dual-axis gyro with guaranteed-by-design vibration rejection and high cross-axis isolation. It is specifically designed for demanding consumer applications requiring low cost, small size and high performance.

The IDG-300 gyro includes integrated electronics necessary for application-ready functionality. It incorporates X- and Y-axis low-pass filters and an EEPROM for on-chip factory calibration of the sensor. Factory trimmed scale factors eliminate the need for external active components and end-user calibration. This product is lead-free and Green Compliant.



Reference Application Diagram

InvenSense

IDG-300

SPECIFICATIONS

All parameters specified are @ VDD=3.0 V and Ta=25°C.

External LPF @ 2kHz. All specifications apply to both axes.

Parameters	Conditions	Min	Typical	Max	Unit
SENSITIVITY					
Full-Scale Range			±500		°/s
Sensitivity			2.0		mV/°/s
Initial Calibration Tolerance		-5		+5	%
Over Specified Temperature			±10		%
Nonlinearity	Best Fit Straight Line		<1		% of FS
Cross-axis Sensitivity			±2		%
ZERO-RATE OUTPUT					
Static Output (Bias)			1.5		V
Initial Calibration Tolerance		-300		+300	mV
Over Specified Temperature		-300		+300	mV
FREQUENCY RESPONSE					
High Frequency Cutoff	Internal LPF -90°		140		Hz
LPF Phase Delay	10Hz		-4.5		°
MECHANICAL FREQUENCIES					
Resonant Frequency	X-Axis Gyroscope	10	12	14	kHz
Resonant Frequency	Y-Axis Gyroscope	13	15	17	kHz
Frequency Separation	X and Y Gyroscopes		3		kHz
NOISE PERFORMANCE					
Rate Noise Density			0.014		°/s/√Hz
OUTPUT DRIVE CAPABILITY					
Output Voltage Swing	Load = 100kΩ to $V_{dd}/2$	0.05		Vdd-0.05	V
Capacitive Load Drive			100		pF
Output Impedance			100		Ω
REFERENCE					
Voltage Value			1.23		V
Load Drive			1		mA
Capacitive Load Drive	Load directly connected to VREF		100		pF
Power Supply Rejection	VDD= 3.0V to 3.3V		1		mV/V
Over Specified Temperature			±5		mV
POWER TIMING					
Zero-rate Output	Settling to ±3°/sec		200		ms
POWER SUPPLY					
Operating Voltage Range		3.0		3.3	V
Quiescent Supply Current				9.5	mA
Over Specified Temperature			±2		mA
TEMPERATURE RANGE					
Specified Temperature Range			0 to +70		°C
Extended Temperature Range	Performance parameters are not applicable beyond Specified Temperature Range		-20 to +85		°C

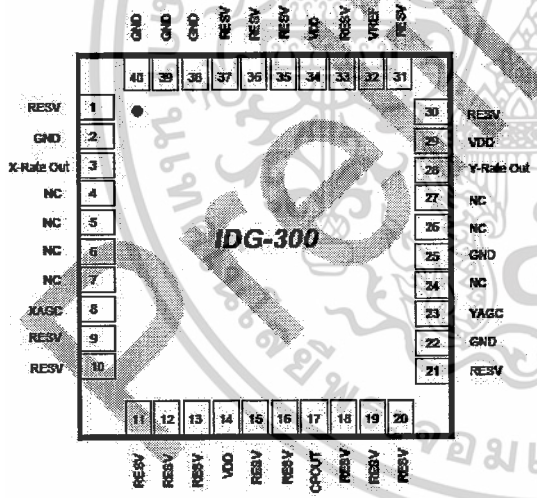
InvenSense

IDG-300

PIN DESCRIPTION

Number	Pin	Description
2, 22, 25, 38, 39, 40	GND	Ground
14, 29, 34	VDD	Positive supply voltage: +3.0V to +3.3V
3	X-Rate Out	X-Rate Out
8	XAGC	Amplitude control filter (See Design Notes: Section 2)
17	CPOUT	Charge pump capacitor
23	YAGC	Amplitude control filter (See Design Notes: Section 2)
28	Y-Rate Out	Y-Rate Out
32	VREF	1.23V precision reference output
1, 9, 10,11, 12, 13, 15,16, 18, 19, 20, 21, 30, 31, 33, 35, 36, 37	RESV	Reserved. Do not connect. Used for factory trimming
4 ,5, 6, 7, 24, 26, 27	NC	Not internally connected; may be used for PCB routing

PIN CONNECTION (TOP VIEW)



40 pin QFN Package
6 x 6 x 1.5mm

RATE SENSITIVE AXIS

This is a dual-axis rate sensing device. It produces a positive output voltage for rotation about the X- or Y-axis, as shown in the figure below.

ข.3 เอกสารคู่มือการใช้งาน AXDL330

ไอซี AXDL330 เป็น ไอซีเซนเซอร์ความเร่ง มีรายละเอียดต่างๆ ดังนี้

ANALOG DEVICES

Small, Low Power, 3-Axis ±3 *g* *i*MEMS® Accelerometer

ADXL330

FEATURES

- 3-axis sensing
- Small, low-profile package
  - 4 mm × 4 mm × 1.45 mm LFCSP
- Low power
  - 180 µA at V_S = 1.8 V (typical)
- Single-supply operation
  - 1.8 V to 3.6 V
- 10,000 *g* shock survival
- Excellent temperature stability
- BW adjustment with a single capacitor per axis
- RoHS/WEEE lead-free compliant

APPLICATIONS

- Cost-sensitive, low power, motion- and tilt-sensing applications
  - Mobile devices
  - Gaming systems
  - Disk drive protection
  - Image stabilization
  - Sports and health devices

GENERAL DESCRIPTION

The ADXL330 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs, all on a single monolithic IC. The product measures acceleration with a minimum full-scale range of ±3 *g*. It can measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration.

The user selects the bandwidth of the accelerometer using the C_X, C_Y, and C_Z capacitors at the X_{OUT}, Y_{OUT}, and Z_{OUT} pins. Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for X and Y axes, and a range of 0.5 Hz to 550 Hz for the Z axis.

The ADXL330 is available in a small, low profile, 4 mm × 4 mm × 1.45 mm, 16-lead, plastic lead frame chip scale package (LFCSP_LQ).

FUNCTIONAL BLOCK DIAGRAM

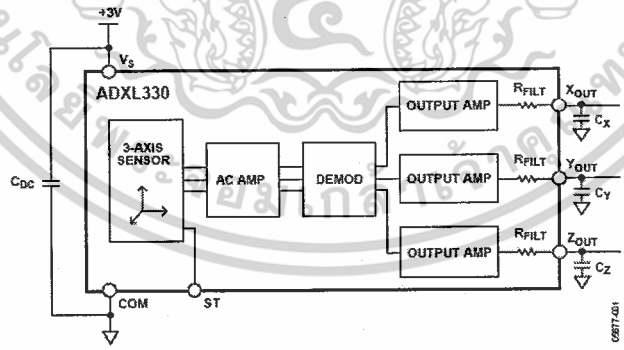


Figure 1.

SPECIFICATIONS

T_A = 25°C, V_S = 3 V, C_X = C_Y = C_Z = 0.1 µF, acceleration = 0 *g*, unless otherwise noted. All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.

Table 1.

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range		±3	±3.6		*g*
Nonlinearity	% of full scale		±0.3		%
Package Alignment Error			±1		Degrees
Interaxis Alignment Error			±0.1		Degrees
Cross Axis Sensitivity[1]			±1		%
SENSITIVITY (RATIOMETRIC)[2]	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 3 V	270	300	330	mV/*g*
Sensitivity Change Due to Temperature[3]	V_S = 3 V		±0.015		%/°C
ZERO *g* BIAS LEVEL (RATIOMETRIC)	Each axis				
0 *g* Voltage at X_{OUT}, Y_{OUT}, Z_{OUT}	V_S = 3 V	1.2	1.5	1.8	V
0 *g* Offset vs. Temperature			±1		mg/°C
NOISE PERFORMANCE					
Noise Density X_{OUT}, Y_{OUT}			280		µg/√Hz rms
Noise Density Z_{OUT}			350		µg/√Hz rms
FREQUENCY RESPONSE[4]					
Bandwidth X_{OUT}, Y_{OUT}[5]	No external filter		1600		Hz
Bandwidth Z_{OUT}[5]	No external filter		550		Hz
R_{FILT} Tolerance			32 ± 15%		kΩ
Sensor Resonant Frequency			5.5		kHz
SELF TEST[6]					
Logic Input Low			+0.6		V
Logic Input High			+2.4		V
ST Actuation Current			+60		µA
Output Change at X_{OUT}	Self test 0 to 1		−150		mV
Output Change at Y_{OUT}	Self test 0 to 1		+150		mV
Output Change at Z_{OUT}	Self test 0 to 1		−60		mV
OUTPUT AMPLIFIER					
Output Swing Low	No load		0.1		V
Output Swing High	No load		2.8		V
POWER SUPPLY					
Operating Voltage Range		1.8		3.6	V
Supply Current	V_S = 3 V		320		µA
Turn-On Time[7]	No external filter		1		ms
TEMPERATURE					
Operating Temperature Range		−25		+70	°C

[1] Defined as coupling between any two axes.
[2] Sensitivity is essentially ratiometric to V_S.
[3] Defined as the output change from ambient-to-maximum temperature or ambient-to-minimum temperature.
[4] Actual frequency response controlled by user-supplied external filter capacitors (C_X, C_Y, C_Z).
[5] Bandwidth with external capacitors = 1/(2 × π × 32 kΩ × *C*). For C_X, C_Y = 0.003 µF, bandwidth = 1.6 kHz. For C_Z = 0.01 µF, bandwidth = 500 Hz. For C_X, C_Y, C_Z = 10 µF, bandwidth = 0.5 Hz.
[6] Self-test response changes cubically with V_S.
[7] Turn-on time is dependent on C_X, C_Y, C_Z and is approximately 160 × C_X or C_Y or C_Z + 1 ms, where C_X, C_Y, C_Z are in µF.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

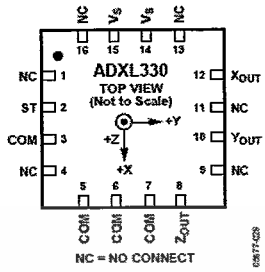


Figure 3. Pin Configuration

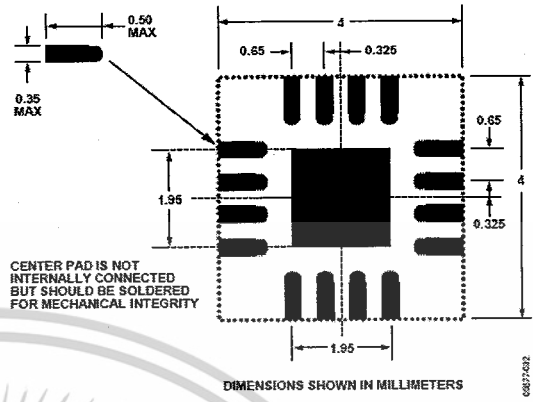


Figure 4. Recommended PCB Layout

Table 4. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	NC	No Connect
2	ST	Self Test
3	COM	Common
4	NC	No Connect
5	COM	Common
6	COM	Common
7	COM	Common
8	Z_{OUT}	Z Channel Output
9	NC	No Connect
10	Y_{OUT}	Y Channel Output
11	NC	No Connect
12	X_{OUT}	X Channel Output
13	NC	No Connect
14	V_S	Supply Voltage (1.8 V to 3.6 V)
15	V_S	Supply Voltage (1.8 V to 3.6 V)
16	NC	No Connect

The ADXL330 output is ratiometric, therefore, the output sensitivity (or scale factor) varies proportionally to the supply voltage. At V_S = 3.6 V, the output sensitivity is typically 360 mV/*g*. At V_S = 2 V, the output sensitivity is typically 195 mV/*g*.

The zero *g* bias output is also ratiometric, so the zero *g* output is nominally equal to $V_S/2$ at all supply voltages.

The output noise is not ratiometric but is absolute in volts; therefore, the noise density decreases as the supply voltage increases. This is because the scale factor (mV/*g*) increases while the noise voltage remains constant. At V_S = 3.6 V, the X- and Y-axis noise density is typically 230 μ*g*/√Hz, while at V_S = 2 V, the X- and Y-axis noise density is typically 350 μ*g*/√Hz.

Self test response in *g* is roughly proportional to the square of the supply voltage. However, when ratiometricity of sensitivity is factored in with supply voltage, the self test response in volts is roughly proportional to the cube of the supply voltage. For example, at V_S = 3.6 V, the self test response for the ADXL330 is approximately −275 mV for the X-axis, +275 mV for the Y-axis, and −100 mV for the Z-axis.

At V_S = 2 V, the self test response is approximately −60 mV for the X-axis, +60 mV for the Y-axis, and −25 mV for the Z-axis.

The supply current decreases as the supply voltage decreases. Typical current consumption at V_S = 3.6 V is 375 μA, and typical current consumption at V_S = 2 V is 200 μA.

AXES OF ACCELERATION SENSITIVITY

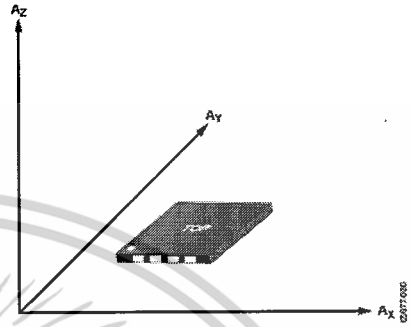


Figure 31. Axes of Acceleration Sensitivity, Corresponding Output Voltage Increases When Accelerated Along the Sensitive Axis

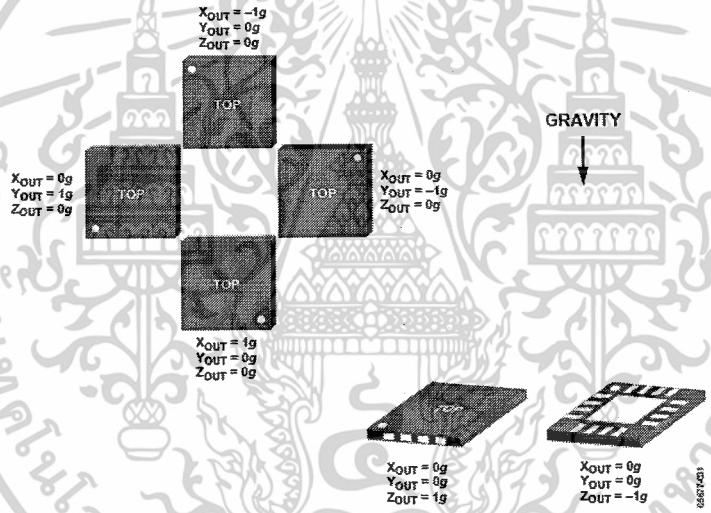


Figure 32. Output Response vs. Orientation to Gravity

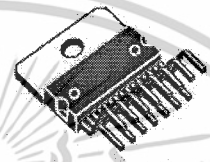
ข.4 เอกสารคู่มือการใช้งาน L298N

ไอซี L298N เป็น ไอซีขับมอเตอร์ สามารถขับมอเตอร์ได้ 2 ตัว มีรายละเอียดต่างๆ ดังนี้

L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15

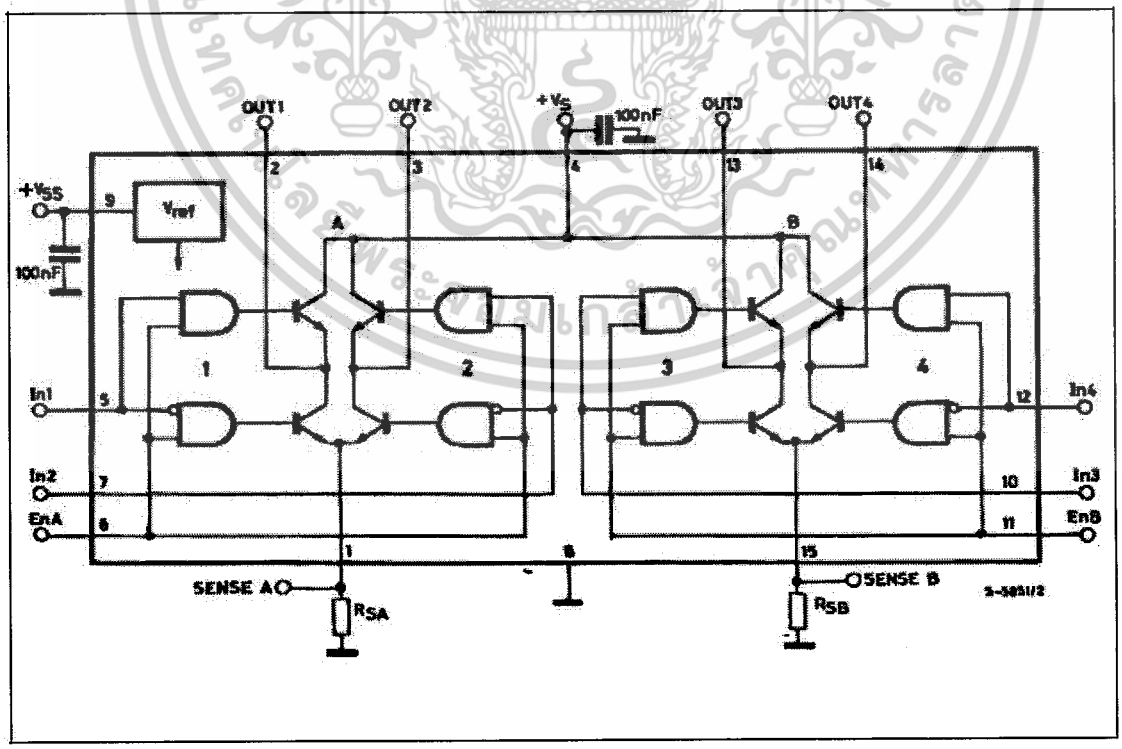
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

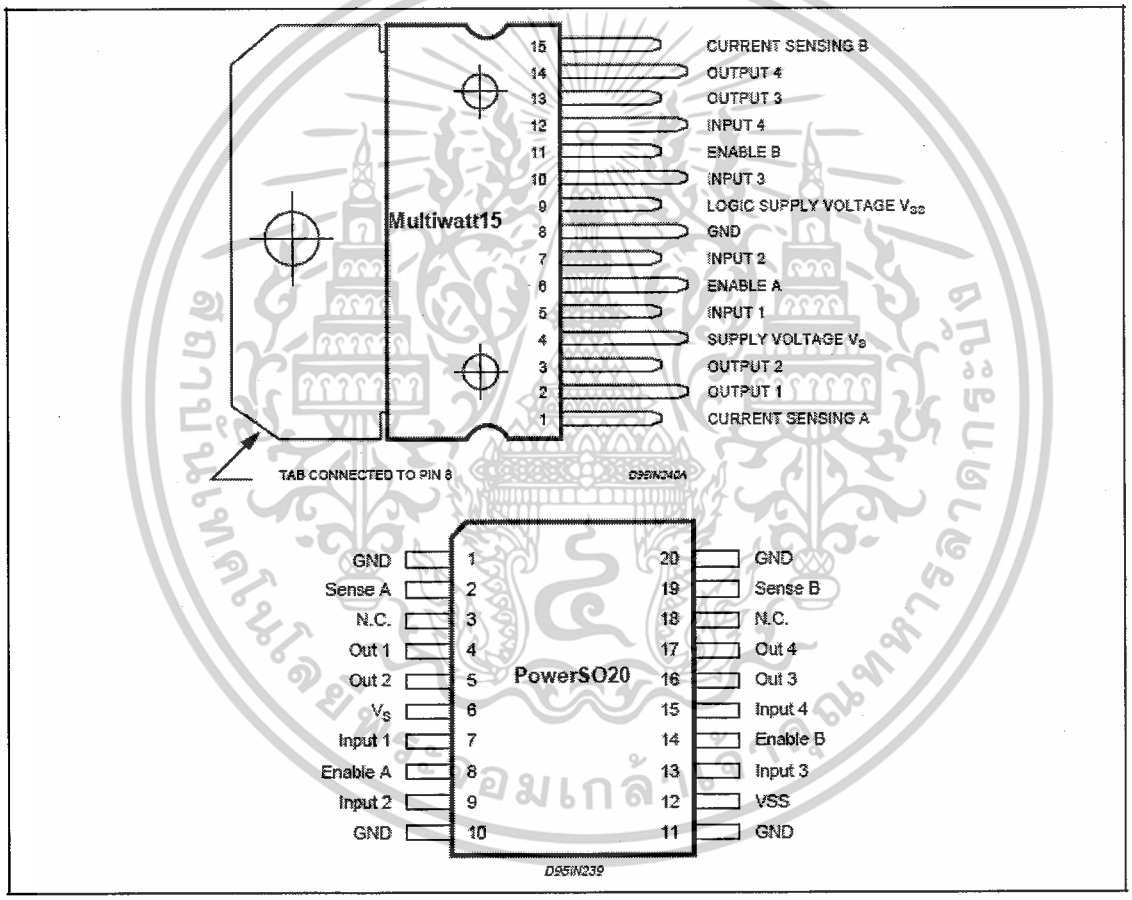
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I, V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\,j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\,j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	Vs	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ข.5 เอกสารคู่มือการใช้งาน MCP1702–3.3

ไอซี MCP1702–3.3 เป็น ไอซีเรกกูเรเตอร์ มีแรงดันเอาท์พุท 3.3 โวลต์ มีรายละเอียดต่างๆ เป็นดังนี้

MICROCHIP

MCP1702

250 mA Low Quiescent Current LDO Regulator

Features

- 2.0 µA Quiescent Current (typical)
- Input Operating Voltage Range: 2.7V to 13.2V
- 250 mA Output Current for Output Voltages ≥ 2.5V
- 200 mA Output Current for Output Voltages < 2.5V
- Low Dropout (LDO) voltage
  - 625 mV typical @ 250 mA (V_{OUT} = 2.8V)
- 0.4% Typical Output Voltage Tolerance
- Standard Output Voltage Options:
  - 1.2V, 1.5V, 1.8V, 2.5V, 2.8V, 3.0V, 3.3V, 4.0V, 5.0V
- Output voltage range 1.2V to 5.5V in 0.1V Increments (50 mV increments available upon request)
- Stable with 1.0 µF to 22 µF Output Capacitor
- Short-Circuit Protection
- Overtemperature Protection

Applications

- Battery-powered Devices
- Battery-powered Alarm Circuits
- Smoke Detectors
- CO^2 Detectors
- Pagers and Cellular Phones
- Smart Battery Packs
- Low Quiescent Current Voltage Reference
- PDAs
- Digital Cameras
- Microcontroller Power
- Solar-Powered Instruments
- Consumer Products
- Battery Powered Data Loggers

Related Literature

- AN765, *"Using Microchip's Micropower LDOs"*, DS00765, Microchip Technology Inc., 2002
- AN766, *"Pin-Compatible CMOS Upgrades to BiPolar LDOs"*, DS00766, Microchip Technology Inc., 2002
- AN792, *"A Method to Determine How Much Power a SOT-23 Can Dissipate in an Application"*, DS00792, Microchip Technology Inc., 2001

Description

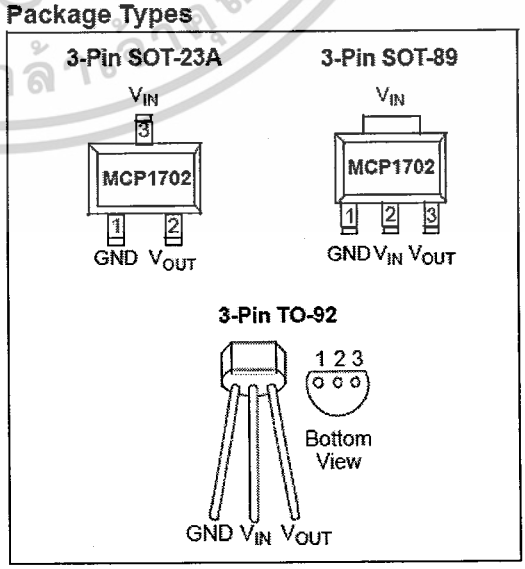
The MCP1702 is a family of CMOS low dropout (LDO) voltage regulators that can deliver up to 250 mA of current while consuming only 2.0 µA of quiescent current (typical). The input operating range is specified from 2.7V to 13.2V, making it an ideal choice for two to six primary cell battery-powered applications, 9V alkaline and one or two cell Li-Ion-powered applications.

The MCP1702 is capable of delivering 250 mA with only 625 mV (typical) of input to output voltage differential (V_{OUT} = 2.8V). The output voltage tolerance of the MCP1702 is typically ±0.4% at +25°C and ±3% maximum over the operating junction temperature range of -40°C to +125°C. Line regulation is ±0.1% typical at +25°C.

Output voltages available for the MCP1702 range from 1.2V to 5.0V. The LDO output is stable when using only 1 µF of output capacitance. Ceramic, tantalum or aluminum electrolytic capacitors can all be used for input and output. Overcurrent limit and overtemperature shutdown provide a robust solution for any application.

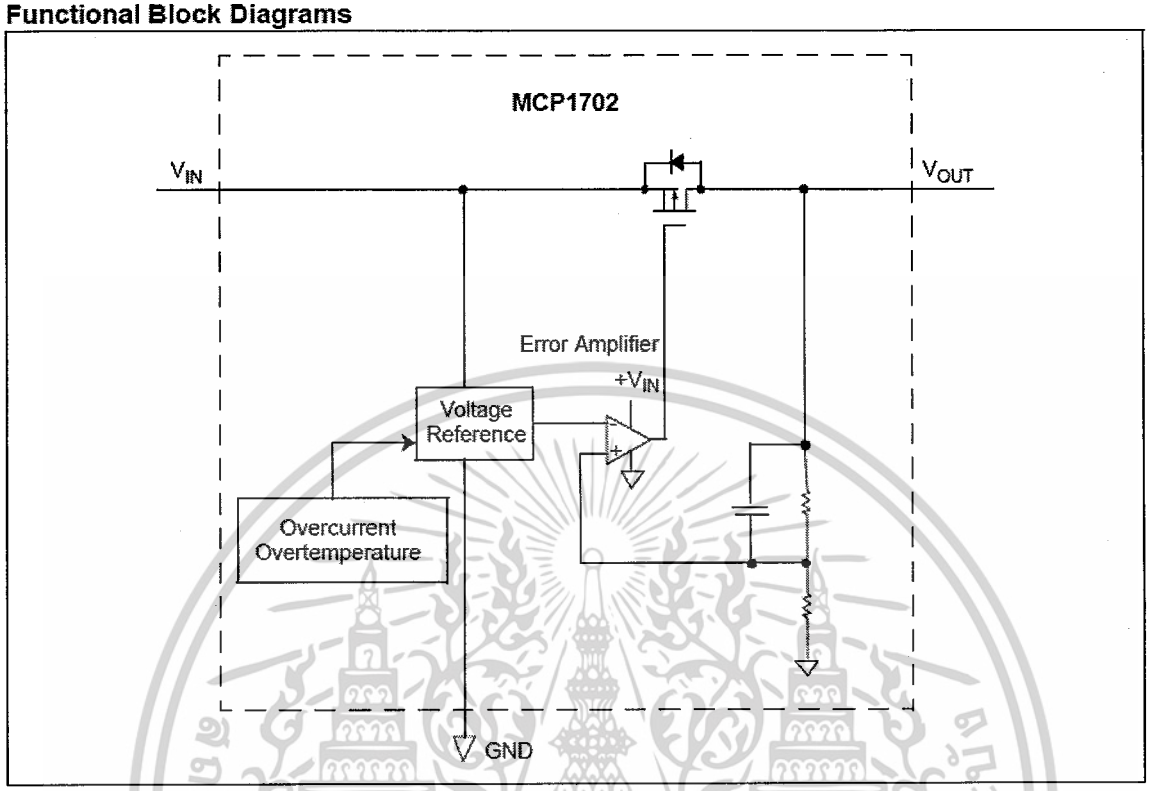
Package options include the SOT-23A, SOT-89-3, and TO-92.

Package Types

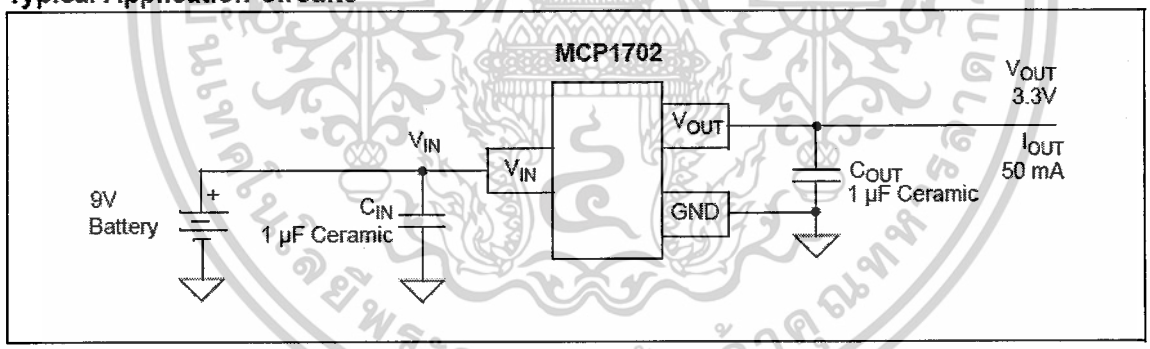


MCP1702

Functional Block Diagrams



Typical Application Circuits



ข.6 เอกสารคู่มือการใช้งาน MC 7805

ไอซี MC 7805 เป็น ไอซีเรกกูเรเตอร์ มีแรงดันเอาท์พุท 5 โวลต์ มีรายละเอียดต่างๆ ดังนี้

MC7800, MC7800A, NCV7805

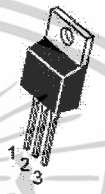
1.0 A Positive Voltage Regulators

ON Semiconductor®

http://onsemi.com

These voltage regulators are monolithic integrated circuits designed as fixed−voltage regulators for a wide variety of applications including local, on−card regulation. These regulators employ internal current limiting, thermal shutdown, and safe−area compensation. With adequate heatsinking they can deliver output currents in excess of 1.0 A. Although designed primarily as a fixed voltage regulator, these devices can be used with external components to obtain adjustable voltages and currents.

- Output Current in Excess of 1.0 A
- No External Components Required
- Internal Thermal Overload Protection
- Internal Short Circuit Current Limiting
- Output Transistor Safe−Area Compensation
- Output Voltage Offered in 2% and 4% Tolerance
- Available in Surface Mount D^2PAK−3, DPAK−3 and Standard 3−Lead Transistor Packages
- NCV Prefix for Automotive and Other Applications Requiring Site and Control Changes
- Pb−Free Packages are Available



TO−220−3
T SUFFIX
CASE 221A

Heatsink surface connected to Pin 2.

Pin 1. Input
2. Ground
3. Output

D^2PAK−3
D2T SUFFIX
CASE 936

Heatsink surface (shown as terminal 4 in case outline drawing) is connected to Pin 2.

DPAK−3
DT SUFFIX
CASE 369C

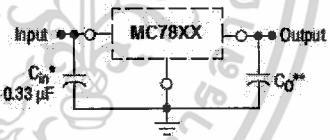
MAXIMUM RATINGS (T_A = 25°C, unless otherwise noted)

Rating	Symbol	Value 369C	Value 221A	Value 936	Unit
Input Voltage (5.0 − 18 V) (24 V)	V_I	35 40			Vdc
Power Dissipation	P_D	Internally Limited			W
Thermal Resistance, Junction−to−Ambient	$R_{\theta JA}$	92	65	Figure 14	°C/W
Thermal Resistance, Junction−to−Case	$R_{\theta JC}$	5.0	5.0	5.0	°C/W
Storage Junction Temperature Range	T_{stg}	−65 to +150			°C
Operating Junction Temperature	T_J	+150			°C

Maximum ratings are those values beyond which device damage can occur. Maximum ratings applied to the device are individual stress limit values (not normal operating conditions) and are not valid simultaneously. If these limits are exceeded, device functional operation is not implied, damage may occur and reliability may be affected.

NOTE: ESD data available upon request.

STANDARD APPLICATION



A common ground is required between the input and the output voltages. The input voltage must remain typically 2.0 V above the output voltage even during the low point on the input ripple voltage.

XX, These two digits of the type number indicate nominal voltage.

* C_{in} is required if regulator is located an appreciable distance from power supply filter.

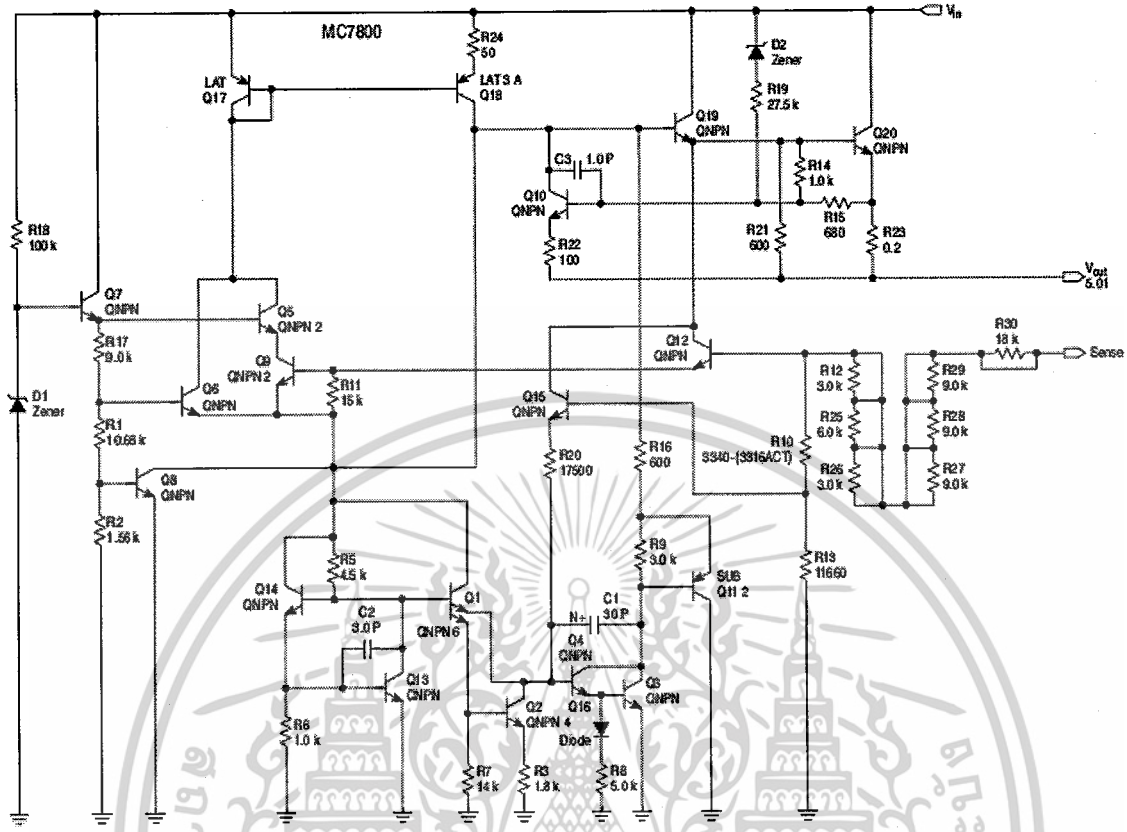
** C_O is not needed for stability; however, it does improve transient response. Values of less than 0.1 μF could cause instability.

ORDERING INFORMATION

See detailed ordering and shipping information in the package dimensions section on page 21 of this data sheet.

DEVICE MARKING INFORMATION

See general marking information in the device marking section on page 25 of this data sheet.



This device contains 22 active transistors.

Figure 1. Representative Schematic Diagram

ภาคผนวก ค

เอกสารคู่มือมอเตอร์

FAULHABER

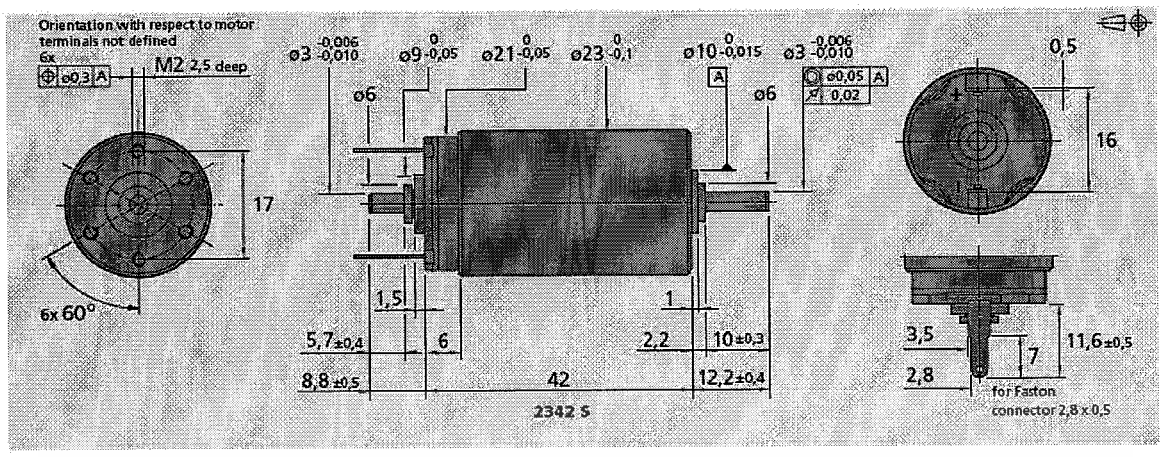
DC-Micromotors 16 mNm

Graphite Commutation

For combination with (overview on page 14-15)
Gearheads:
23/1, 26A, 26/1, 30/1, 38/3
Encoders:
IE2 – 16 ... 512, 5500, 5540

Series 2342 ... CR

		2342 S	006 CR	012 CR	018 CR	024 CR	036 CR	048 CR	
1	Nominal voltage	U_N	6	12	18	24	36	48	Volt
2	Terminal resistance	R	0,40	1,90	4,10	7,10	15,9	31,20	Ω
3	Output power	$P_{2\,max.}$	20,50	17,00	18,10	19,00	19,40	17,70	W
4	Efficiency	$\eta_{max.}$	81	80	81	81	81	81	%
5	No-load speed	n_0	9 000	8 100	8 000	8 500	8 100	8 000	rpm
6	No-load current (with shaft ø 3,0 mm)	I_0	0,170	0,075	0,048	0,038	0,024	0,017	A
7	Stall torque	M_H	87,2	80,0	86,5	85,4	91,4	84,4	mNm
8	Friction torque	M_R	0,98	1,00	0,99	0,99	0,99	0,95	mNm
9	Speed constant	k_n	1 650	713	462	366	231	170	rpm/V
10	Back-EMF constant	k_E	0,604	1,400	2,160	2,730	4,340	5,870	mV/rpm
11	Torque constant	k_M	5,77	13,40	20,70	26,10	41,40	56,10	mNm/A
12	Current constant	k_I	0,173	0,075	0,048	0,038	0,024	0,018	A/mNm
13	Slope of n-M curve	$\Delta n/\Delta M$	103	101	92,5	99,5	88,6	94,8	rpm/mNm
14	Rotor inductance	L	13,5	65	150	265	590	1 050	µH
15	Mechanical time constant	τ_m	6	6	6	6	6	6	ms
16	Rotor inertia	J	5,6	5,7	6,2	5,8	6,5	6,0	gcm²
17	Angular acceleration	$\alpha_{max.}$	160	140	140	150	140	140	10^3rad/s²
18	Thermal resistance	R_{th1} / R_{th2}	3 / 15						K/W
19	Thermal time constant	τ_{w1} / τ_{w2}	6,5 / 490						s
20	Operating temperature range:								
	– motor		– 30 ... + 100						°C
	– rotor, max. permissible		+ 125						°C
21	Shaft bearings		ball bearings, preloaded						
22	Shaft load max.:								
	– with shaft diameter		3,0						mm
	– radial at 3 000 rpm (3 mm from bearing)		20						N
	– axial at 3 000 rpm		2						N
	– axial at standstill		20						N
23	Shaft play:								
	– radial	≤	0,015						mm
	– axial	=	0						mm
24	Housing material		steel, black coated						
25	Weight		88						g
26	Direction of rotation		clockwise, viewed from the front face						
Recommended values - mathematically independent of each other									
27	Speed up to	$n_{max.}$	7 000	7 000	7 000	7 000	7 000	7 000	rpm
28	Torque up to	$M_{max.}$	16	16	16	16	16	16	mNm
29	Current up to (thermal limits)	$I_{max.}$	2,700	1,400	0,950	0,720	0,480	0,350	A



ภาคผนวก ง
โปรแกรมควบคุมหุ่นยนต์อินเวิร์ทเพนดูลัม

โปรแกรมในโครงงานระบบควบคุมอินเวิร์ทเพนดูลัม เขียนด้วยภาษาซี โดยใช้คอมไพเลอร์ MPLAB C30 เวอร์ชัน 3.02 ของบริษัทไมโครชิพ โดยได้แยกไฟล์ออกเป็นสำหรับหน่วยติดต่อต่างๆ

ง.1 โปรแกรมย่อยสำหรับควบคุมหน่วยแปลงสัญญาณอนาลอกเป็นดิจิตอล (adc.h)

```
//------------------------------------------------------------------------------//
//-------------------------------- ADC Module ----------------------------------//
//------------------------------------------------------------------------------//
#define adc_read(pin)          adc_buf[pin]
void init_adc();
unsigned int adc_buf[16];
volatile unsigned int* ADC16Ptr = &ADCBUF0; //Pointer to ADC register buffer,
unsigned int adc_filter_buffer[4][3] = { {0,0,0},{0,0,0},{0,0,0},{0,0,0} };
unsigned int current_buf = 1;
//------------------------------------------------------------------------------//
//-------------------------------- ADC Initial ---------------------------------//
//------------------------------------------------------------------------------//
void init_adc()
{
        ADCON1bits.ADON = 0;            // Turn off the A/D converter
        ADCON2bits.SMPI = 3;            // Interrupt on 4th sample
        ADCON2bits.CHPS = 0;            // Sample channel CH0
        ADCON1bits.SIMSAM = 0;          // n/a for single channel sample
        ADCON2bits.BUFM = 0;            // Single 16-word result buffer
        ADCON2bits.ALTS = 0;            // Always use MUX A input select
        // MUX A Input Select
        ADCHSbits.CH0SA = 0;            // Overridden by CSCNA
        ADCHSbits.CH0NA = 0;            // Select VREF- for CH0- input
        ADCON2bits.CSCNA = 1;           // Scan channel 0 inputs
        ADCSSL = 0x000E;                // Scan AN1..AN3
        ADCHSbits.CH123SA = 0;          // n/a
        ADCHSbits.CH123NA = 0;          // n/a
        // MUX B Input Select
        ADCHSbits.CH123SB = 0;          // n/a
        ADCHSbits.CH123NB = 0;          // n/a
        // SPECIFIC BITS
        ADCON1bits.FORM = 0b10;         // Use fractional (DOUT = dddd dddd dd00 0000)
                                        // for future compatibility with 12-bit ADC
        ADCON1bits.SSRC = 7;            // Use auto to start convertion
        ADCON1bits.ASAM = 1;            // Sampling begins immediately after last
                                        // conversion completes.
                                        //SAMP bit is auto set
        ADCON2bits.VCFG = 1;            // Use VREF+ at AN0 VREF- at AVSS
        ADCON3bits.ADCS = 0b011111;     // Timing for 8MHz
        ADPCFG = 0xFFF0;                // AN0..AN3 are analog
        TRISB = 0b11111111;             // Port B is input
        IFS0bits.ADIF = 0;              //Clear the A/D interrupt flag bit
        IEC0bits.ADIE = 1;              //Set the A/D interrupt enable bit
        ADCON1bits.ADON = 1;            //Turn on the A/D converter
}
//------------------------------------------------------------------------------//
//------------------------------ ADC Interrupt ---------------------------------//
//------------------------------------------------------------------------------//
extern void __attribute__((__interrupt__)) _ADCInterrupt(void)
{
    unsigned int buffer = 0;
    for (buffer=0; buffer < 3; buffer++)
        adc_buf[buffer] = ADC16Ptr[buffer];
        IFS0bits.ADIF = 0;
}
//------------------------------------------------------------------------------//
```

ง.2 โปรแกรมย่อยอ่านค่ามุมเบี่ยงเบนจากเซนเซอร์ความเร่ง (accelero.h)

```
//-------------------------------------------------------------------------------//
//--------------------------Accelerometer Module ---------------------------------//
//-------------------------------------------------------------------------------//

#include <math.h>
static const int ADC_X_NEUTRAL = 32500;         // Center of X axis
static const int ADC_Z_NEUTRAL = 32500;         // Center of Z axis
#define ACCEL_Z_ADC    2               //input ADC pin accelero axis Z
#define ACCEL_X_ADC    1               //input ADC pin accelero axis Y

//-------------------------read angle from accelerometre----------------------------//
float read_angle()
{
       int shifted_z = (int)adc_read(ACCEL_Z_ADC) - ADC_Z_NEUTRAL;
       int shifted_x = (int)adc_read(ACCEL_X_ADC) - ADC_X_NEUTRAL;
       //no need to scale
       float z = (float) shifted_z;
       float x = (float) shifted_x;
       return (atan2(-z, x));
}
//-------------------------------------------------------------------------------//
```

ง.3 โปรแกรมย่อยอ่านค่าอัตราเร็วเชิงมุมจากเซนเซอร์ไจโรสโคป (gyro.h)

```
//-------------------------------------------------------------------------------//
//-------------------------------Gyro Module -----------------------------------//
//-------------------------------------------------------------------------------//

#include "adc.h"
#define gyro_y  0
static const float gyro_y_degrees_per_second = 500.0*2.0;  // Specific of gyro (degree/s)
static const float gyro_y_rad_per_second = 8.72*2.0;       // Specific of gyro (rad/s)

// Returns degrees per second along the roll axis
// Our gyro (IDG-300) is non-radiometric, meaning we should
// treat our measurements as if the device was running on 3V
// and not 3V3.
// Centre (0/s) is at 1.5V
// 2mV per degrees/s
//-------------------------------------------------------------------------------//
//---------------------------Function read to degree -----------------------------//
//-------------------------------------------------------------------------------//

float gyro_y_degrees()
{
       return -(((float)adc_read(gyro_y) / 65500.0*3.3)-1.5)/0.002;
}

//-------------------------------------------------------------------------------//
//---------------------------Function read to rad --------------------------------//
//-------------------------------------------------------------------------------//

float gyro_y_rad()
{
       //return (((float)adc_read(gyro_y) / 65500.0*3.3)-1.5)/0.002/180.0*3.14159;
       return -(((float)adc_read(gyro_y) * 0.0004394279)-13.09);
}

//-------------------------------------------------------------------------------//
```

ง.4 โปรแกรมย่อยแคลแมนฟิลเตอร์ (kalman.h)

```
//-------------------------------------------------------------------------------------//
//-----------------------------Kalman Filter-------------------------------------------//
//-------------------------------------------------------------------------------------//

#include <math.h>
struct kalman
{
        //These variables represent our state matrix x //
        float x_angle,
              x_bias;
        //Our error covariance matrix //
        float P_00,
              P_01,
              P_10,
              P_11;

        // Q is a 2x2 matrix of the covariance. Because we
        // assuma the gyro and accelero noise to be independend
        // of eachother, the covariances on the / diagonal are 0.
        // Covariance Q, the process noise, from the assumption
        // x = F x + B u + w
        // with w having a normal distribution with covariance Q.
        // covariance = E[ (X - E[X])*(X - E[X])' ]
        // We assume is linair with dt

        float Q_angle, Q_gyro;

        //Covariance R, our observation noise (from the accelerometer)
        //Also assumed to be linair with dt

        float R_angle;
};

// Initializing the struct
void init_kalman(struct kalman *filterdata, float Q_angle,
                 float Q_gyro, float R_angle);

// Kalman predict
void kalman_predict(struct kalman *filterdata, const float gyro, const float dt);

//Kalman update
float kalman_update(struct kalman *filterdata, const float angle_m);

void init_kalman(struct kalman *filterdata, float Q_angle,
                 float Q_gyro, float R_angle)
{
        filterdata->Q_angle = Q_angle;
        filterdata->Q_gyro  = Q_gyro;
        filterdata->R_angle = R_angle;
}

// The predict function. Updates 2 variables:
// our model-state x and the 2x2 matrix P
// x = [ angle, bias ]'
//   = F x + B u
//   = [1-dt, 0 1][angle, bias ] + [ dt, 0][dotAngle 0]
//   => angle = angle + dt (dotAngle - bias)
// bias  = bias
// P = F P transpose(F) + Q
//     = [1-dt, 0 1]* P * [ 1 0, -dt 1]+Q
// P(0,0)=P(0,0)-dt * ( P(1,0)+P(0,1))+dt * P(1,1)+Q(0,0)
// P(0,1)=P(0,1)-dt * P(1,1)+Q(0,1)
// P(1,0)=P(1,0)-dt * P(1,1)+Q(1,0)
// P(1,1)=P(1,1)+Q(1,1)


void kalman_predict(struct kalman *filterdata, const float dotAngle, const float dt)
{
```

```
        filterdata->x_angle += dt * (dotAngle - filterdata->x_bias);
        filterdata->P_00 += -dt *(filterdata->P_10+filterdata->P_01)+filterdata->Q_angle*dt;
        filterdata->P_01 += -dt * filterdata->P_11;
        filterdata->P_10 += -dt * filterdata->P_11;
        filterdata->P_11 += +filterdata->Q_gyro * dt;
}

// The update function updates our model using
// the information from a 2nd measurement.
// Input angle_m is the angle measured by the accelerometer.
// y = z - H x

// S = H P transpose(H) + R
//   = [1 0]P [1,0]+R
//   = P(0,0)+R

// K = P transpose(H) S^-1
// [P(0,0), P(1,0)]/S
// x = x + K y
// P = (I - K H) P
// ([1 0,     [ K(0),
//   0 1]- K(1)]*[1 0])P
// =[P(0,0)-P(0,0)*K(0) P(0,1)-P(0,1)*K(0),
// =P(1,0)-P(0,0)*K(1) P(1,1)-P(0,1)*K(1)]

float kalman_update(struct kalman *filterdata, const float angle_m)
{
        const float y = angle_m - filterdata->x_angle;
        const float S = filterdata->P_00+filterdata->R_angle;
        const float K_0=filterdata->P_00/S;
        const float K_1=filterdata->P_10/S;
        filterdata->x_angle +=  K_0*y;
        filterdata->x_bias  +=  K_1*y;
        filterdata->P_00=K_0*filterdata->P_00;
        filterdata->P_01=K_0*filterdata->P_01;
        filterdata->P_10=K_1*filterdata->P_00;
        filterdata->P_11=K_1*filterdata->P_01;
        return filterdata->x_angle;
}
//-------------------------------------------------------------------------------//
```

ง.5 โปรแกรมย่อยควบคุมการอ่านค่าจากเอนโคดเดอร์ (incap.h)

```
//-------------------------------------------------------------------------------//
//-----------------------Function for Input Capture  -----------------------------//
//-------------------------------------------------------------------------------//

#include <incap.h>
int pulse_l = 0,pulse_r = 0;
//-------------------------------------------------------------------------------//
//----------------------Function for Initial Input Capture  ----------------------//
//-------------------------------------------------------------------------------//

void init_incap()
{
        //Config InputCapture 1
        ConfigIntCapture1(IC_INT_PRIOR_2 & IC_INT_ON); // Enable IC1 interrupt
        OpenCapture1(IC_IDLE_STOP &                    // Capture disable in idle mode
                IC_TIMER3_SRC &                        // Timer 3 count for IC1
                IC_INT_1CAPTURE &                      // Capture 1 count for interrupt
                IC_EVERY_RISE_EDGE);                   // Capture every falling edge only

        //Config InputCapture 7
        ConfigIntCapture7(IC_INT_PRIOR_2 & IC_INT_ON); // Enable IC7 interrupt


OpenCapture7(IC_IDLE_STOP &                    // Capture disable in idle mode
        IC_TIMER3_SRC &                        // Timer 3 count for IC7
```

```
        IC_INT_1CAPTURE &                       // Capture per 1 count for interrupt
        IC_EVERY_RISE_EDGE);                    // Capture every falling edge only

        TRISDbits.TRISD1=1;
        TRISBbits.TRISB5=1;
}
//-------------------------------------------------------------------------------//
//-----------Interrupt service routine Capture every Falling  edge ---------------//
//-------------------------------------------------------------------------------//
//Input Capture 1
void _ISR _IC1Interrupt(void)
{
        IFS0bits.IC1IF = 0;     // Clear IC1 interrupt flag
        if(PORTDbits.RD1)
                pulse_l--;
        else
                pulse_l++;
}
//Input Capture 7
void _ISR _IC7Interrupt(void)
{
        IFS1bits.IC7IF = 0;     // Clear IC7 interrupt flag

        if(PORTBbits.RB5)
                pulse_r++;
        else
                pulse_r--;
}
//-------------------------------------------------------------------------------//
//-----------------------Function Read Pulse from Encoder ----------------------//
//-------------------------------------------------------------------------------//
//Read encoder from wheel Left
int read_encoder_L(void)
{
        int tmp;
        tmp = pulse_l;
        pulse_l = 0;
        return(tmp);
}
//Read encoder from wheel Right
int read_encoder_R(void)
{
        int tmp;
        tmp = pulse_r;
        pulse_r = 0;
        return(tmp);
}
//-------------------------------------------------------------------------------//
```

ง.6 โปรแกรมย่อยความคุมหน่วยสร้างสัญญาณ PWM (pwm.h)

```
//-------------------------------------------------------------------------------//
//-----------------------------------PWM Module----------------------------------//
//      Motor1=PWM1L(RE0)& PWM1H(RE1)
//      Motor2=PWM2L(RE2)& PWM2H(RE3)
//-------------------------------------------------------------------------------//
#define FW      0
#define BW      1
#define SP      2
//-------------------------------------------------------------------------------//
//-----------------------------------PWM Initial---------------------------------//
//-------------------------------------------------------------------------------//

void init_pwm()
{
        PTPER = 1473;                   // setup PWM Time base period 20 kHz
                                        // for Free running mode
                                        // PTPER = (Fcy/(Fpwm*PTMR prescaler)) - 1
                                        // PWM 12 bit 0-4096
        OVDCON = 0X0000;                // setup output for PIN PWNxL & PWNxH all not PWM
        DTCON1=1;                       // setup Dead Time
```

```
        PWMCON1=0x0333;                 // setup PWMCON1(0b0011 0011 0011)
                                        // independant output mode
                                        // Enable PEN1H & PEN2H for PWM [Motor 1]
                                        // Enable PEN1L & PEN2L for PWM [Motor 2]
        PDC1=0;                         // setup duty cycle for PWM1
        PDC2=0;                         // setup duty cycle for PWM2

        IFS2bits.PWMIF = 0;             // Clear PWM interrupt flag
        IEC2bits.PWMIE = 0;             // Disable PWM interrupt
        PTCONbits.PTMOD = 0;            // PWM time mode select [Free running mode ]
        PTCONbits.PTEN = 1;             // PWM time base Enable bit
}

//-----------------------------------------------------------------------------//
//-------------------------------------Motor1----------------------------------//
//-----------------------------------------------------------------------------//

void motor_L(char tris,unsigned int pwm)
{
        if(tris == FW)
        {
                OVDCONbits.POVD1H = 0;
                OVDCONbits.POVD1L = 1;
        }
        else if(tris == BW)
        {
                OVDCONbits.POVD1H = 1;
                OVDCONbits.POVD1L = 0;
        }
        else if(tris == SP)
        {
                OVDCONbits.POVD1H = 0;
                OVDCONbits.POVD1L = 0;
        }
        PDC1=pwm*29;
}

//-----------------------------------------------------------------------------//
//-------------------------------------Motor2----------------------------------//
//-----------------------------------------------------------------------------//

void motor_R(char tris,unsigned int pwm)
{
        if(tris == FW)
        {
                OVDCONbits.POVD2H = 1;
                OVDCONbits.POVD2L = 0;
        }
        else if(tris == BW)
        {
                OVDCONbits.POVD2H = 0;
                OVDCONbits.POVD2L = 1;
        }
        else if(tris == SP)
        {
                OVDCONbits.POVD2H = 0;
                OVDCONbits.POVD2L = 0;
        }
        PDC2=pwm*29;
}

//-----------------------------------------------------------------------------//
```

ง.7 โปรแกรมย่อยควบคุมหน่วยสื่อสารข้อมูลอนุกรม (uart1.h)

```
//-------------------------------------------------------------------------------//
//---------------------------------UART1 Function--------------------------------//
//-------------------------------------------------------------------------------//
#include "common.h"
#define TRUE                    1
#define FALSE                   0
#define BAUDRATE                115200//19200
#define BRGVAL                  ((FCY/BAUDRATE)/16)-1
#define RECV_FIFO_SIZE          96

//static unsigned char tempchar[20];
static int sFTXREG;
volatile unsigned  char       recv_fifo[RECV_FIFO_SIZE];
volatile unsigned  char       *recv_fifo_head;
volatile unsigned  char       *recv_fifo_tail;
volatile unsigned  char       recv_fifo_byte_count;
void init_uart10;
void init_uart1_buffer(void);
unsigned char uart1_getc(void);
int is_uart1_char_ready(void);
int uart1_putc(char c);
int uart1_puts(const char *s );

extern void __attribute__ ((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{
        unsigned char temp;
    IFS0bits.U1RXIF = 0;
    if(recv_fifo_byte_count < RECV_FIFO_SIZE)
    {
        IEC0bits.U1RXIE = 0;
                temp = U1RXREG;
                *recv_fifo_head = temp;
                recv_fifo_head++;
                if(recv_fifo_head == (recv_fifo + RECV_FIFO_SIZE))
                recv_fifo_head = recv_fifo;
                recv_fifo_byte_count++;
        IEC0bits.U1RXIE = 1;
        }
    else{
        IEC0bits.U1RXIE = 0;
        IFS0bits.U1RXIF = 0;
    }
}

extern void __attribute__ ((interrupt, no_auto_psv)) _U1TXInterrupt(void)
{
        sFTXREG=0;
        IFS0bits.U1TXIF = 0;
        while(U1STAbits.UTXBF);
}

void init_uart10
{
        /*** configure U1MODE ***/
        //U1MODEbits.UARTEN = 1;        // Bit15 TX, RX DISABLED, ENABLE at end of func
        //U1MODEbits.notimplemented;    // Bit14 Unimplemented: Read as '0'
        U1MODEbits.USIDL = 0;           // Bit13 Continue in Idle
        U1MODEbits.ALTIO = 1;           // Bit10 ALTIO: UART Alternate I/O Selection bit
        //U1MODEbits.notimplemented;    // Bit9-8 Unimplemented: Read as '0'
        U1MODEbits.WAKE = 0;            // Bit7 No Wake up (since we don't sleep here)
        U1MODEbits.LPBACK = 0;          // Bit6 No Loop Back
        U1MODEbits.ABAUD = 0;           // Bit5 No Auto  baud (would require sending '55')
        //U1MODEbits.notimplemented;    // Bit4-3 Unimplemented: Read as '0'
        U1MODEbits.PDSEL = 0;           // Bit1-28bit, No Parity
        U1MODEbits.STSEL = 0;           // Bit0   One Stop Bit
        U1BRG = BRGVAL;
```

```
        /***configure U2Status ***/
        //U1STA  = 0X0440;              // Reset status register and enable TX & RX
        //Load all values in for U1STA SFR
        U1STAbits.UTXISEL = 1;          //Bit15 Int when Char is transferred (1/2 config!)
        //U1STAbits.notimplemented;     //Bit14-12
        U1STAbits.UTXBRK = 0;           //Bit11 Transmit Break bit
        //U1STAbits.UTXEN = 1;          //Bit10 Transmit Enable bit
        U1STAbits.UTXBF = 0;            //Bit9 Read Only Bit
                                        //Transmit Buffer Full Status bit
        U1STAbits.TRMT = 0;             //Bit8 Read Only bit* Transmit Shift Register
        U1STAbits.URXISEL = 0;          //Bits6,7 Receive Interrupt Mode Selection bit
        U1STAbits.ADDEN = 0;            //Bit5  Address Detect Disabled
        U1STAbits.RIDLE = 0;            //Bit4*Read Only Bit* Receiver Idle
        U1STAbits.PERR = 0;             //Bit3*Read Only Bit* Parity Error Status
        U1STAbits.FERR = 0;             //Bit2*Read Only Bit* Framing Error Status
        U1STAbits.OERR = 0;             //Bit1*Read Only Bit* Receive Buffer
                                        //Overrun Error Status
        U1STAbits.URXDA = 0;            //Bit0*Read Only Bit* Receive
                                        //Buffer Data Available bit
        //IPC6=0x4400;                  // Mid Range Interrupt Priority level,
                                        //no urgent reason
        IFS0bits.U1TXIF = 0;            // Clear the Transmit Interrupt Flag
        IEC0bits.U1TXIE = 1;            // Enable Transmit Interrupts
        IFS0bits.U1RXIF = 0;            // Clear the Recieve Interrupt Flag
        IEC0bits.U1RXIE = 1;            // Enable Recieve Interrupts
        U1MODEbits.UARTEN = 1;          // And turn the peripheral on
        U1STAbits.UTXEN = 1;
        init_uart1_buffer();
}

void init_uart1_buffer(void)
{
        recv_fifo_head = recv_fifo;
        recv_fifo_tail = recv_fifo;
        recv_fifo_byte_count = 0;
}

int is_uart1_char_ready(void)
{
        if(recv_fifo_byte_count != 0)
                return(TRUE);
        else
                return(FALSE);
}

int uart1_putc(char c)
{
        while (U1STAbits.UTXBF);        // wait for room in the transmit buffer
                U1TXREG = c;
        return 0;
}

int uart1_puts(const char *s)
{
        while (*s)
                uart1_putc(*s++);
        return 0;
}

unsigned char uart1_getc(void)
{
        unsigned char value;
        value = 0;
        if(recv_fifo_byte_count > 0)
        {
                value = *recv_fifo_tail++;
                if(recv_fifo_tail == (recv_fifo + RECV_FIFO_SIZE))
                        recv_fifo_tail = recv_fifo;
                recv_fifo_byte_count--;
```

```
        }
        return(value);
}

int uart1_gets(char *s)
{
        char tmp;
        while(1)
        {
                if(is_uart1_char_ready())
                {
                tmp = uart1_getc();
                        *s++ = tmp;
                }
                if(*s == '\n')
                        break;
        }
    return 0;
}

int uart1_putcommand(char *s )
{
    while (*s!='\n')
         uart1_putc(*s++);
    return 0;
}

//-------------------------------------------------------------------------------//
```

ง.8 โปรแกรมย่อยควบคุมจอแสดงผลแอลซีดี (lcd3310.h)

```
//-------------------------------------------------------------------------------//
//---------------------------- NOKIA LCD3310 Module -----------------------------//
//-------------------------------------------------------------------------------//
#ifndef lcd3310_h
#define lcd3310_h
//-------------------------------------------------------------------------------//
//-------------------------------Config PIN LCD----------------------------------//
//-------------------------------------------------------------------------------//
#define LCD3310_RST             LATFbits.LATF2          // LCD3310 Reset LCD
#define TRIS_RST                TRISFbits.TRISF2
#define LCD3310_CS              LATEbits.LATE4          // LCD3310 Chip select
#define TRIS_CS                 TRISEbits.TRISE4
#define LCD3310_DC              LATEbits.LATE5          // LCD3310 Data /Command mode
select
#define TRIS_DC                 TRISEbits.TRISE5
#define LCD3310_SDIN            LATFbits.LATF3          // LCD3310 Data in
#define TRIS_SDIN               TRISFbits.TRISF3
#define LCD3310_SCLK            LATEbits.LATE8          // LCD3310 CLK
#define TRIS_SCLK               TRISEbits.TRISE8
//-------------------------------------------------------------------------------
----------//
//-------------------------------------Font table--------------------------------
----------//
//-------------------------------------------------------------------------------
----------//

static const unsigned short FontLookup [][5] =
{
   { 0x00, 0x00, 0x00, 0x00, 0x00 },  // sp
   { 0x00, 0x00, 0x2f, 0x00, 0x00 },  // !
   { 0x00, 0x07, 0x00, 0x07, 0x00 },  // "
   { 0x14, 0x7f, 0x14, 0x7f, 0x14 },  // #
   { 0x24, 0x2a, 0x7f, 0x2a, 0x12 },  // $
   { 0xc4, 0xc8, 0x10, 0x26, 0x46 },  // %
   { 0x36, 0x49, 0x55, 0x22, 0x50 },  // &
   { 0x00, 0x05, 0x03, 0x00, 0x00 },  // '
   { 0x00, 0x1c, 0x22, 0x41, 0x00 },  // (
   { 0x00, 0x41, 0x22, 0x1c, 0x00 },  // )
   { 0x14, 0x08, 0x3E, 0x08, 0x14 },  // *
   { 0x08, 0x08, 0x3E, 0x08, 0x08 },  // +
   { 0x00, 0x00, 0x50, 0x30, 0x00 },  // ,
   { 0x10, 0x10, 0x10, 0x10, 0x10 },  // -
```

```
{ 0x00, 0x60, 0x60, 0x00, 0x00 },  // .
{ 0x20, 0x10, 0x08, 0x04, 0x02 },  // /
{ 0x3E, 0x51, 0x49, 0x45, 0x3E },  // 0
{ 0x00, 0x42, 0x7F, 0x40, 0x00 },  // 1
{ 0x42, 0x61, 0x51, 0x49, 0x46 },  // 2
{ 0x21, 0x41, 0x45, 0x4B, 0x31 },  // 3
{ 0x18, 0x14, 0x12, 0x7F, 0x10 },  // 4
{ 0x27, 0x45, 0x45, 0x45, 0x39 },  // 5
{ 0x3C, 0x4A, 0x49, 0x49, 0x30 },  // 6
{ 0x01, 0x71, 0x09, 0x05, 0x03 },  // 7
{ 0x36, 0x49, 0x49, 0x49, 0x36 },  // 8
{ 0x06, 0x49, 0x49, 0x29, 0x1E },  // 9
{ 0x00, 0x36, 0x36, 0x00, 0x00 },  // :
{ 0x00, 0x56, 0x36, 0x00, 0x00 },  // ;
{ 0x08, 0x14, 0x22, 0x41, 0x00 },  // <
{ 0x14, 0x14, 0x14, 0x14, 0x14 },  // =
{ 0x00, 0x41, 0x22, 0x14, 0x08 },  // >
{ 0x02, 0x01, 0x51, 0x09, 0x06 },  // ?
{ 0x32, 0x49, 0x59, 0x51, 0x3E },  // @
{ 0x7E, 0x11, 0x11, 0x11, 0x7E },  // A
{ 0x7F, 0x49, 0x49, 0x49, 0x36 },  // B
{ 0x3E, 0x41, 0x41, 0x41, 0x22 },  // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C },  // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 },  // E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 },  // F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A },  // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F },  // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 },  // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 },  // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 },  // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 },  // L
{ 0x7F, 0x02, 0x0C, 0x02, 0x7F },  // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F },  // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E },  // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 },  // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E },  // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 },  // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 },  // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 },  // T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F },  // U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F },  // V
{ 0x3F, 0x40, 0x38, 0x40, 0x3F },  // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 },  // X
{ 0x07, 0x08, 0x70, 0x08, 0x07 },  // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 },  // Z
{ 0x00, 0x7F, 0x41, 0x41, 0x00 },  // [
{ 0x55, 0x2A, 0x55, 0x2A, 0x55 },  // 55
{ 0x00, 0x41, 0x41, 0x7F, 0x00 },  // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 },  // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 },  // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 },  // '
{ 0x20, 0x54, 0x54, 0x54, 0x78 },  // a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 },  // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 },  // c
{ 0x38, 0x44, 0x44, 0x48, 0x7F },  // d
{ 0x38, 0x54, 0x54, 0x54, 0x18 },  // e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 },  // f
{ 0x0C, 0x52, 0x52, 0x52, 0x3E },  // g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 },  // h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 },  // i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 },  // j
{ 0x7F, 0x10, 0x28, 0x44, 0x00 },  // k
{ 0x00, 0x41, 0x7F, 0x40, 0x00 },  // l
{ 0x7C, 0x04, 0x18, 0x04, 0x78 },  // m
{ 0x7C, 0x08, 0x04, 0x04, 0x78 },  // n
{ 0x38, 0x44, 0x44, 0x44, 0x38 },  // o
{ 0x7C, 0x14, 0x14, 0x14, 0x08 },  // p
{ 0x08, 0x14, 0x14, 0x18, 0x7C },  // q
{ 0x7C, 0x08, 0x04, 0x04, 0x08 },  // r
{ 0x48, 0x54, 0x54, 0x54, 0x20 },  // s
{ 0x04, 0x3F, 0x44, 0x40, 0x20 },  // t
{ 0x3C, 0x40, 0x40, 0x20, 0x7C },  // u
{ 0x1C, 0x20, 0x40, 0x20, 0x1C },  // v
{ 0x3C, 0x40, 0x30, 0x40, 0x3C },  // w
{ 0x44, 0x28, 0x10, 0x28, 0x44 },  // x
{ 0x0C, 0x50, 0x50, 0x50, 0x3C },  // y
{ 0x44, 0x64, 0x54, 0x4C, 0x44 }   // z
```

```
};

//-------------------------------------------------------------------------------//
typedef enum
{
        LCD_CMD  = 0,
        LCD_DATA = 1
} LcdCmdData;
//-------------------------------------------------------------------------------//
//--------------------------------Function Phototype-----------------------------//
//-------------------------------------------------------------------------------//
#define lcd_print(x,y,data)   lcd_gotoxy(x,y); lcd_puts(data);

void init_lcd();                                        // Init SPI LCD
void lcd_clear();                                       // Clear screen
void lcd_gotoxy(unsigned short xnokia,                  // Set the position
                unsigned short ynokia);                 //for data (x = 0->83, y = 0->5)
void lcd_puts(char *dataPtr );                          // Print Mesg to LCD
void lcd_putc(char ascii_code);                         // Print char to LCD
void lcd_send(unsigned short data, unsigned short cd );     // Send Command/Data
//-------------------------------------------------------------------------------//
//-----------------------------------Init LCD------------------------------------//
//-------------------------------------------------------------------------------//

void init_lcd()
{
        TRIS_RST = 0;
        TRIS_CS = 0;
        TRIS_DC = 0;
        TRIS_SDIN = 0;
        TRIS_SCLK = 0;
        LCD3310_RST = 1;                        // RST LCD
        LCD3310_CS = 1;                         // Set SPI CS pin du LCD 3310
        LCD3310_DC = 0;                         // Enter Data Mode
        lcd_send( 0x21, LCD_CMD );  // LCD Extended Commands.
        lcd_send( 0xC8, LCD_CMD );  // Set LCD Vop (Contrast).
        lcd_send( 0x06, LCD_CMD );  // Set Temp coefficent.
        lcd_send( 0x13, LCD_CMD );  // LCD bias mode 1:48.
        lcd_send( 0x20, LCD_CMD );  // LCD Standard Commands,Horizontal address mode.
        lcd_send( 0x0C, LCD_CMD );  // LCD in normal mode.
        lcd_clear();
}

//-------------------------------------------------------------------------------//
//-----------------------------------LCD Clear-----------------------------------//
//-------------------------------------------------------------------------------//

void lcd_clear()
{
        unsigned int x, y;
        for (y=0; y<6; y++)
                for (x=0; x<84; x++)
                        lcd_send(0x00, LCD_DATA);
}

//-------------------------------------------------------------------------------//
//--------------------------------LCD Go to X Y----------------------------------//
//-------------------------------------------------------------------------------//

void lcd_gotoxy(unsigned short xnokia, unsigned short ynokia)
{
        lcd_send(0x40|(ynokia & 0x07),LCD_CMD);             // new y cursor position
        lcd_send(0x80|(xnokia & 0x7f),LCD_CMD);             // new x cursor position
}

//-------------------------------------------------------------------------------//
//--------------------------------LCD Prinrt String------------------------------//
//-------------------------------------------------------------------------------//
void lcd_puts(char *dataPtr )
{
        while (*dataPtr)
                lcd_putc( *dataPtr++ );
}

//-------------------------------------------------------------------------------//
```

```
//---------------------------------LCD Print Char-------------------------------//
//------------------------------------------------------------------------------//

void lcd_putc(char ascii_code)
{
        if (ascii_code<0x20)
                return;
        lcd_send( FontLookup [ascii_code-0x20][0], LCD_DATA );      // bout de char
        lcd_send( FontLookup [ascii_code-0x20][1], LCD_DATA );      // bout de char
        lcd_send( FontLookup [ascii_code-0x20][2], LCD_DATA );      // bout de char
        lcd_send( FontLookup [ascii_code-0x20][3], LCD_DATA );      // bout de char
        lcd_send( FontLookup [ascii_code-0x20][4], LCD_DATA );      // bout de char
}

//------------------------------------------------------------------------------//
//---------------------------------LCD Send data--------------------------------//
//------------------------------------------------------------------------------//

void lcd_send(unsigned short data, unsigned short cd )
{
        char caa;
        if (cd == LCD_DATA)
                LCD3310_DC = 1;                         // Enter Data Mode
        else
                LCD3310_DC = 0;                         // Enter Command Mode

        LCD3310_CS = 0;                                 // Set SPI CS PIN LCD 3310
        for (caa=8;caa>0;caa--)
        {
                LCD3310_SCLK = 0;
                 if ((data & 0x80)==0)
                {
                        LCD3310_SDIN = 0;
                }
                else
                {
                        LCD3310_SDIN = 1;
                }
                LCD3310_SCLK = 1;
                data = data<<1;
  }
        LCD3310_CS = 1;                         // Set SPI CS pin du LCD 3310 non actif
}
#endif
//------------------------------------------------------------------------------//
```

ง.9 โปรแกรมย่อยกำหนดค่าของไมโครคอนโทรลเลอร์ (common.h)

```
//------------------------------------------------------------------------------//
//we're using the external oscillator so it's running at 7.3728MHz
#define XTAL    7372800
#define PLL             16
#define FCY             XTAL*PLL/4
//------------------------------------------------------------------------------//
```

ง.10 โปรแกรมหลักของระบบควบคุมอินเวิร์ทเพนดูลัม (main.c)

```
//------------------------------------------------------------------------------//
//---------------------------------Main Program --------------------------------//
//------------------------------------------------------------------------------//
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <timer.h>
#include "gyro.h"
#include "accelero.h"
#include "timer.h"
#include "pwm.h"
#include "incap.h"
#include "kalman.h"
#include "lcd3310.h"
#include "uart1.h"
```

```
//-------------------------------------------------------------------------------------//
//-----------------------------------Configuration bits --------------------------------//
//-------------------------------------------------------------------------------------//

_FOSC(CSW_FSCM_OFF & XT_PLL16); //7.3728Mhz *16=117.9648 MHz /4=29.4912 MIPS
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & BORV_27 & PWRT_16 & MCLR_EN);
_FGS(CODE_PROT_OFF);
//-------------------------------------------------------------------------------------//
//-----------------------------------Define Constant -----------------------------------//
//-------------------------------------------------------------------------------------//
#define MPP                                     0.000817            //metre per pulse
#define angle_acc_offset        1.7562//1.5708//1.770
#define angle_rate_offset       -0.2145//0.125
#define angle_kal_offset        0.007
//Gain K
#define K_x             -31
#define K_xdot  -51
#define K_a     4578
#define K_adot  11
#define K_servo 10
//-------------------------------------------------------------------------------------//
//-----------------------------------Function Prototype --------------------------------//
//-------------------------------------------------------------------------------------//

void init_timer20;
void delay_ms(unsigned int N);
void balancing_control();

//-------------------------------------------------------------------------------------//
//-----------------------------------Define Variable -----------------------------------//
//-------------------------------------------------------------------------------------//

char charIn,i;
char buffer[50];
char tempString[100];
char *tempPointer;
char count = 0;
char flag = 0;
int pwm = 0;
int send = 0;
double angle_kal = 0;
double angle_acc = 0;
double angle_rate = 0;
double angle_rate1=0;
double angle_rate2=0;
double X = 0;
double X_dot = 0;
double X_old = 0;
double command_motor = 0;
double command = 0;
double X_ref = 0;
double X_ref_old = 0;
double X_integral = 0;
double command_servo = 0;
double setpoint = 0;
float dt;
struct kalman filter_angle;

//-------------------------------------------------------------------------------------//
//-----------------------------------Main Function -------------------------------------//
//-------------------------------------------------------------------------------------//

int main()
{
        tempPointer = tempString;
        delay_ms(100);
        init_pwm();
```

```
                motor_L(SP,0);
                motor_R(SP,0);

        init_adc();
        init_incap();
        init_uart1();
        init_lcd();
        init_timer2();
//      timer_init_ms();
        init_kalman(&filter_angle, 0.0001, 0.0003, 0.69);//q_angle,q_gyro,r_angle
        delay_ms(200);
        lcd_print(1,1,"adot  =");
        lcd_print(1,2,"angle =");
        lcd_print(1,3,"kal   =");
        lcd_print(1,4,"V:      X:");
        lcd_print(1,5,"com:     PWM:");
        //timer_reset();
                motor_L(BW,25);
                motor_R(BW,25);
        while(1)
        {
                if(is_uart1_char_ready())
                {
                        do{
                            while(!is_uart1_char_ready());
                        charIn =  uart1_getc();
                                *tempPointer = charIn;
                                tempPointer++;
                                count++;
                        }while((charIn != 0x0A) && (charIn != 0x0D));
                        tempPointer = tempPointer - count;
                        count =0;
                        command = atof(tempString);
                        sprintf(buffer,"command = %.2f ",command);
                        lcd_print(0,0,buffer);
                        //uart1_puts(buffer);
                        flag = 1;
                }

        }
        return 0;
}

//-------------------------------------------------------------------------------//
//-----------------------------------Fanction Delay------------------------------//
//-------------------------------------------------------------------------------//

void delay_ms(unsigned int N)
{
        unsigned int j;
        while(N--)
        for(j=0;j<5600;j++);
}

//-------------------------------------------------------------------------------//
//-----------------------------------Timer2 Initial------------------------------//
//-------------------------------------------------------------------------------//

void init_timer2()
{
        unsigned int match_value;
        ConfigIntTimer2(T2_INT_PRIOR_1 & T2_INT_ON);    // Timer 2 Enable interrupt
        //Configuration Timer 2 for interrupt
        WriteTimer2(0);                         // Timer 2 clear period
        match_value = 1152;                     // Timer 2 interval 10 ms
        OpenTimer2(T2_ON &T2_GATE_OFF & T2_IDLE_STOP &
                                T2_PS_1_256 & T2_SOURCE_INT, match_value);
}
```

```
//-------------------------------------------------------------------------------//

//--------------------Interrupt service routine  Timer 2 every 10 ms --------------//
//-------------------------------------------------------------------------------//

void __attribute__((__interrupt__)) _T2Interrupt(void)
{
        IFS0bits.T2IF = 0;                      // Clear Timer interrupt flag
        WriteTimer2(0);                         // Clear count value at TMR2 register

        balancing_control();
}
//-------------------------------------------------------------------------------//
//--------------------------------Balancing Control ------------------------------//
//-------------------------------------------------------------------------------//


void balancing_control()
{
        dt = 0.01;                                      //simpling Time

        sprintf(buffer,"dt = %.5f ",dt);
        lcd_print(0,0,buffer);

        //---------------------------Find Angle Rate-----------------------------//

        angle_rate = gyro_y_rad() + angle_rate_offset;
        //debug angle rate
        sprintf(buffer,"%.3f ",angle_rate);
        lcd_print(40,1,buffer);

        //-----------------------------Find Angle--------------------------------//
        //execute kalman filter
        //read angle from accelerometer
                angle_acc = read_angle() + angle_acc_offset;
        //Kalman predict
                kalman_predict(&filter_angle, angle_rate, dt);

        //Kalman update + result (angle)
                angle_kal = kalman_update(&filter_angle, angle_acc);


        angle_rate1=(angle_kal-angle_rate2)/dt;
        angle_rate2=angle_kal;

        //debug angle
        sprintf(buffer,"%.5f ",angle_acc);
        lcd_print(40,2,buffer);
        sprintf(buffer,"%.5f ",angle_kal);
        lcd_print(40,3,buffer);

        //-------------------------Find Distance & velocity----------------------//
        X += ((read_encoder_L()+read_encoder_R())/2)*MPP;

        X_dot = (X - X_old)/dt;

        X_old = X;
        //debug distance , velocity
        sprintf(buffer,"%.2f ",X_dot);
        lcd_print(15,4,buffer);
        sprintf(buffer,"%.2f ",X);
        lcd_print(55,4,buffer);

        //-----------------------------Servo Control-----------------------------//
        /*
        X_ref = setpoint - X;
        //integral
        X_integral += 0.5*(X_ref+X_ref_old)*dt;
        X_ref_old = X_ref;

        command_servo = X_integral*K_servo;
```

```
        //sprintf(buffer,"X_ref = %.3f ",X_ref);
        //lcd_print(0,0,buffer);
        //sprintf(buffer,"servo = %.3f ",output_servo);
        //lcd_print(0,1,buffer);
        */
        //------------------------State feedback Control-----------------------//

        command_motor = (K_x*X)+(K_xdot*X_dot)+(K_a*angle_kal)
                              +(K_adot*angle_rate);//+command_servo;

        if(flag == 1)
        {
                command_motor += command;
                send = 1;
        }

        //debug output
        //sprintf(buffer,"C=%.0f ",output);
        //lcd_print(0,5,buffer);

        //---------------------------Driver Motor PWM---------------------------//
        pwm = command_motor;

        if(command_motor > 0)
        {
                if(pwm > 100)
                        pwm = 100;
                motor_L(BW,pwm);
                motor_R(BW,pwm);
        }
        else if(command_motor < 0)
        {
                if(pwm < -100)
                        pwm = -100;
                motor_L(FW,-pwm);
                motor_R(FW,-pwm);
        }
        else
        {
                motor_L(SP,0);
                motor_R(SP,0);
        }

        //debug pwm
        sprintf(buffer,"%.0f ",command_motor);
        lcd_print(20,5,buffer);
        sprintf(buffer,"%d ",pwm);
        lcd_print(65,5,buffer);

        if(send > 0)
        {

        sprintf(buffer,"%d\n\r%.3f\n\r%.3f\n\r%.3f\n\r%.3f\n\r%.3f\n\r",pwm,X_dot,X,
                                                angle_acc,angle_kal,angle_rate);
                uart1_puts(buffer);

                send++;

                if(send >=1010)
                {
                        flag = 0;
                        send = 0;
                }
        }
}
//------------------------------------------------------------------------------//
```

ภาคผนวก จ

โปรแกรมออกแบบตัวควบคุมด้วย MATLAB

```
%%%%%%%%%%%%%%%%%%%%%%%%
%%    LQR Method     %%
%%%%%%%%%%%%%%%%%%%%%%%%
%% Moment of Inertia %%
%%%%%%%%%%%%%%%%%%%%%%%%

Mr  =  0.81217 ;              % Mass of Robot (kg)
Mp  =  0.1     ;              % Mass of Plate (kg)
Ir                            % Moment of inertia  ( robot )
Ip                            % Moment of inertia  ( Plate )
rr   =  0.072   ;             % Radius of plate (m)
L   =  0.92    ;              % Height
Tpp  =  0.02    ;             % Period of plate
Tp  =  0.00053763;            % Period of Total
g   =  9.81    ;              % Gravity
%% Calculate %%
Ipp = (Mp*(rr^2)*g*(Tpp^2))/(L*((2*pi)^2));
%% Variable of initial %%

g  = 9.81 ;                   % Gravity (m/s^2)
r  = 0.01 ;                   % Radius of wheel (m)
Mw = 0.175 ;                  % Mass of wheel (kg)
Mp = 0.81217  ;               % Mass of robot's chassis (kg)
Iw =((Mw*r)^2)/2 ;            % Inertia of wheel (kg*m^2)

Ip =((Mp+Mr)*(rr^2)*g*(Tpp^2))/(L*((2*pi)^2))-Ipp;
% Inertia of robot's chassis (kg*m^2)

l  = 0.15  ;                  % Length to the body's centre of mass (m)

%% Variable of Motor %%
Km  = 0.0134    ;             % Motor torque constant (Nm/A)
Ke  = 0.013369 ;              % Back emf constant (Vs/rad)
R   = 1.9 ;                   % Nominal Terminal Resistance (ohm)

%%  Matrix System  %%
beta   = ( 2*Mw + (2*Iw/r^2) + Mp ) ;
alpha  = ( Ip*beta + 2*Mp*l^2*(Mw+(Iw/r^2))) ;
A = [0                          1                                   0                    0 ;
     0  (2*Km*Ke*((Mp*l*r)-Ip-(Mp*l^2)))/(R*r^2*alpha) (Mp^2*g*l^2)/alpha    0 ;
     0                          0                                   0                    1 ;
     0    (2*Km*Ke*((r*beta)-(Mp*l)))/(R*r^2*alpha)     (Mp*g*l*beta)/alpha  0 ]

B = [          0                                  ;
       (2*Km*(Ip+(Mp*l^2)-(Mp*l*r)))/(R*r*alpha)   ;
               0                                  ;
       (2*Km*((Mp*l)-(r*beta))/(R*r*alpha ))      ]

C = [  1   0  0  0  ;
       0  0  1  0  ]

D = [ 0 ;
      0 ]

%%   Transfer function of state space system   %%

disp('Transfer Function of the system')
[num,den] = ss2tf(A,B,C,D)
printsys(num,den)

%%   The Eigenvalues of the system matrix  %%

disp('The Eigenvalue of the system matrix A')
disp('A positive value will indicate an unstable system')
p = eig(A)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  LQR Control Design   %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Designing the LQR controller')
```

```
disp('Q = C"*C is a 4x4 weighting matrix for the outputs')
disp('R = is a 1x1 weighting matrix for the input')

% x is the weighting for the cart position
% y is the weighting for the pendulum position
w = 100;
x = 1 ;
y = 20000 ;
z = 1 ;

Q = [w 0 0 0 ;
   0 x 0 0 ;
   0 0 y 0 ;
   0 0 0 z] ;
R = 0.001;

%% Feedback Gain
disp('Feedback Gains for the system')
K = LQR(A,B,Q,R)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Simulate the system   %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
%% Simutation time step
t = 0:0.01:10;
%% System matrices with feedback
Ac = [A-(B*K)] ;    % xDot = (A-BK)x , u = -Kx
Bc = [B] ;
Cc = [C] ;
Dc = [D] ;

New_pole = eig(Ac)

%% Plot Step Response
figure(1) ,  step(Ac,B,eye(4),[0;0;0;0]),  %% Response of step input
grid
title ('Step Response of LQR Controller')
xlabel('time[s]') ,
ylabel('Angular velocity(rad/s)                    Angle[rad]                Velocity[m/s]
Position[m]'),

%%%%%%%%%%%%%%%%%%
%% Define Gain %%
%%%%%%%%%%%%%%%%%%
k1= -51;
k2= -61;
k3= 4578;
k4= 10;
K = [k1 k2 k3 k4] ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Simulate the system  (define gain)  %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all
%% Simutation time step
t = 0:0.01:10;
%% System matrices with feedback
Acc = [A-(B*K)] ;       % xDot = (A-BK)x , u = -Kx
Bc  = [B] ;
Cc  = [C] ;
Dc  = [D] ;

New_pole = eig(Acc)

%% Plot Step Response

figure(1) ,  step(Acc,B,eye(4),[0;0;0;0]),  %% Response of step input
grid
title ('Step Response of LQR Controller')
xlabel('time[s]') ,
ylabel('Angular velocity(rad/s)  Angle[rad] Velocity[m/s] Position[m]'),
```