การออกแบบและการสร้างหุ่นยนต์แขนกล

SCARA ROBOT ARM

โดย

นายดำรงเกียรติ ปียะบงการ

นายเอกรัฐ อินทกาญจน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมเครื่องกล

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปีการศึกษา2542

การออกแบบและการสร้างหุ่นยนต์แขนกล

SCARA ROBOT ARM

โดย

นาย ดำรงเกียรติ ปียะบงการ

นาย เอกรัฐ อินทกาญจน์

อาจารย์ที่ปรึกษา

อ. เอกพจน์ ตันตราภิวัฒน์

อ.วิภู ศรีสืบสาย

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมเครื่องกล

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การออกแบบและการสร้างหุ่นยนต์แขนกล

ผู้จัดทำ

1. นายดำรงเกียรติ ปิยะบงการ
2. นายเอกรัฐ อินทกาญจน์

อาจารย์ที่ปรึกษา
(อ. เอกพจน์ ตันตราภิวัฒน์)

อาจารย์ที่ปรึกษา
(อ . วิภู ศรีสืบสาย)

หัวข้อปริญญานิพนธ์	การออกแบบและการสร้างหุ่นยนต์แขนกล
นักศึกษา	ดำรงเกียรติ ปียะบงการ
	เอกรัฐ อินทกาญจน์
ระดับการศึกษา	วิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมเครื่องกล
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.	2542
อาจารย์ผู้ควบคุมวิทยานิพนธ์	อ. เอกพจน์ ตันตราภิวัฒน์
	อ. วิภู ศรีสืบสาย

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการศึกษาและการออกแบบรวมทั้งการสร้างแขนกลแบบ SCARA ROBOT ARM โดยรวมทั้งระบบควบคุมการทำงาน โดยใช้ DC Motor เป็นตัวต้นกำลังในการขับเคลื่อนแขนของหุ่นยนต์ ซึ่งมีสอง ส่วนด้วยกันคือ แขนท่อนบนและล่าง ซึ่งลักษณะของตัวแขนจะสามารถเคลื่อนที่ได้ใน สองแกน โดยการหมุนที่สัมพันธ์กันของมอเตอร์ทั้งสองไปยังจุดที่เป็นเป้าหมายตามค่าที่เราได้ป้อนให้ทำงาน ซึ่งเราจะสามารถตรวจสอบตำแหน่งโดยใช้ตัวเอนโคดเดอร์ (encoder) ซึ่งเป็นตัวนับรอบของการหมุนของมอเตอร์เพื่อถ่ายทอดไปยังโปรแกรมเพื่อตรวจสอบว่าถึงจุดที่เป็นเป้าหมายหรือยังและป้อนเอาท์พุทไปยังวงจรเพื่อส่งกระแสให้ต่อตัวมอเตอร์ต่อไป

Thesis Title	Scara Robot Arm
Student	Dumrongkiat Piyabongkan
	Ek-karat Intakarn
Level of Study	Bachelor of Engineering in Mechanical Engineering
	King Mongkut 's Institute of Technology Ladkrabang
Year	1999
Thesis Advisor	Eakkapoj Tontrapiwat
	Wipoo Sriseubsai

ABSTRACT

This thesis is the study and design of SCARA ROBOT ARM and motion control system. This model uses DC motor to generate movement of the arm in x (horizontal) and y (vertical) axis driven by DC motor running in correspondence direction (clockwise or counterclockwise). We can check the position of robot arm by encoder which is counting the cycle of running motor and then ,send result to program for checking the robot arm position subsequently, the program will send output to electronic circuit in order to generate electrical current for motor.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จไปได้ด้วยดีเพราะได้รับความกรุณาจาก อาจารย์เอกพจน์ ตันตราภิวัฒน์ อาจารย์ที่ปรึกษา ที่ให้คำปรึกษาและชี้แนะแนวทางตลอดการทำปริญญานิพนธ์ทั้ง ขอขอบคุณ พี่มณฑา เทียมเมือง ที่ให้คำแนะนำ และช่วยเหลือในการเชื่อมชิ้นงานให้ออกมาเรียบร้อยสวยงาม

ขอขอบคุณน้องๆเพื่อนๆชมรม รักบี้ ที่ช่วยในการพิมพ์งานคอมพิวเตอร์ ขอบคุณ นาย อนุสรณ์ ดวงรัตน์ 4H สำหรับคำแนะนำด้านคอมพิวเตอร์เมื่อเครื่องคอมพิวเตอร์มีปัญหา และ ขอขอบคุณทุกคนที่เกี่ยวข้องแต่ไม่สามารถลงนามที่นี้ได้หมดทุกคน

คำรงเกียรติ ปิยะบงการ
เอกรัฐ อินทกาญจน์

สารบัญ

สารบัญภาพ

สารบัญตาราง

บทที่ 1

บทนำ

คำว่า " หุ่นยนต์ " นี้ค้นกำเนิดจริงๆแล้ว เกิดขึ้นจากนิยายวิทยาศาสตร์แบบเพ้อฝันหรือที่เราเรียกว่า แฟนตาซี (fantasy) ของชาวเชคโกสโลวาเกีย ในปี 1920 โดยทั่วไปแล้วพอพูดได้ว่า " หุ่นยนต์ " คนทั่วไปมักจะคิดถึงหุ่นยนต์ที่มีรูปร่างเหมือนมนุษย์สามารถที่จะคิดและทำงานบางอย่างแทนมนุษย์ สามารถควบคุมได้โดยใช้โปรแกรมทางคอมพิวเตอร์ และสามารถที่จะเปลี่ยนการทำงานให้ทำงานอย่างอื่นได้ (ตามนิยามของสมาคมหุ่นยนต์โรงงาน (Robotics Industrial Association, RIA) และสถาบันเทคโนโลยีเกี่ยวกับหุ่นยนต์ของอเมริกา (Robotics Institute of America , RIA)

จากคำนิยามนี้จะเห็นว่า หุ่นยนต์ก็คือเครื่องยนต์อัตโนมัติชนิดหนึ่งที่สามารถเปลี่ยนโปรแกรมการทำงานได้เท่านั้นเอง เครื่องยนต์อัตโนมัติ (Automatics) นี้แบ่งออกได้เป็น 3 ชนิดคือ

1. Fixed Automation คือเครื่องยนต์อัตโนมัติที่ทำงานอยู่กับที่และทำงานอย่างเดิมตลอด
2. Programmable Automation คือเครื่องยนต์อัตโนมัติที่สามารถเปลี่ยนการทำงานได้หลายอย่างแต่ต้องรองานที่ทำในตอนนั้นให้เสร็จก่อนจึงสามารถเปลี่ยนได้
3. Flexible Automation คือเครื่องยนต์อัตโนมัติที่สามารถเปลี่ยนการทำงานได้และสามารถทำงานได้หลายอย่างได้ในวงรอบการทำงานเดียวกัน

ปัจจุบันหุ่นยนต์ ได้ถูกมาประยุกต์ใช้ในงานอุตสาหกรรมอย่างกว้างขวาง หุ่นยนต์จะถูกนำมารวมเข้ากับระบบการผลิต โดยทำหน้าที่เป็นเครื่องจักรกลหลัก เช่น งานประกอบชิ้นส่วน งานเชื่อมโลหะ งานพ่นสี นอกจากนี้ ในระบบการผลิต หุ่นยนต์ยังถูกนำไปใช้เป็นเครื่องจักรช่วยงาน เช่น งานหยิบจับเพื่อเปลี่ยนชิ้นงานระหว่างขั้นตอนการผลิต งานเปลี่ยนเครื่องมือ เป็นต้น

ระบบการผลิตที่มีการนำเอาหุ่นยนต์เข้ามาทำงานร่วมด้วยจะเป็นระบบการผลิตที่มีการยืดหยุ่นสูง สามารถผลิตชิ้นงานหรือผลิตภัณฑ์ได้จำนวนมาก มีความต่อเนื่องและผลผลิตมีคุณภาพใกล้เคียงกัน ความสามารถและความยืดหยุ่นในการทำงานของหุ่นยนต์ จะขึ้นอยู่กับลักษณะโครงสร้างทางกล ชนิดของตัวขับเคลื่อน (Actuator) ที่ใช้ในการขับเคลื่อนแกนของหุ่นยนต์ ชุดควบคุมการขับเคลื่อน และโปรแกรมควบคุมฟังก์ชันการทำงาน

บทที่ 2

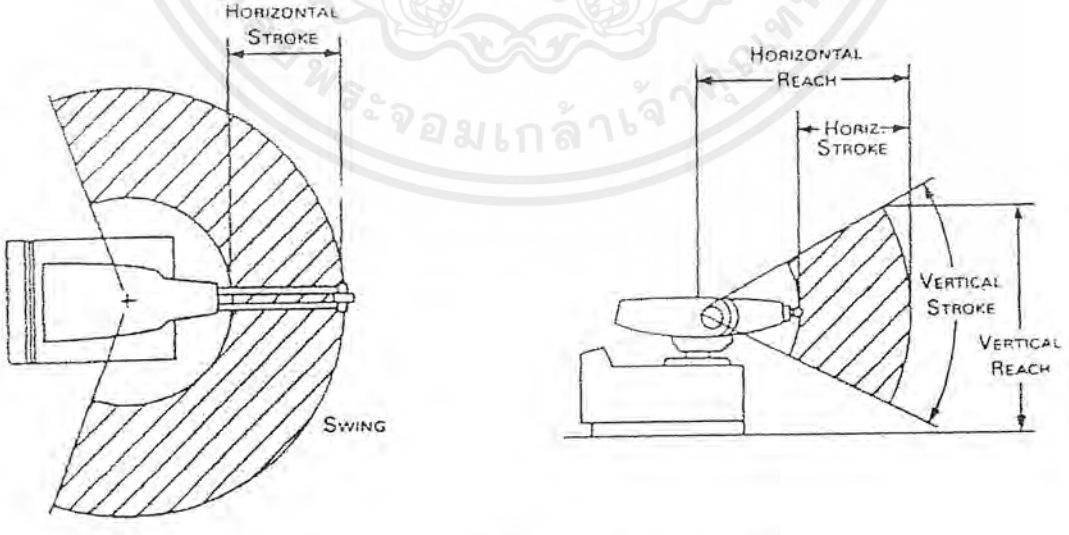
หุ่นยนต์แขนกล

2.1 ชนิดของหุ่นยนต์

หุ่นยนต์แบ่งตามลักษณะการเคลื่อนที่และลักษณะของแขนแบ่งได้ เป็น 4 ชนิดใหญ่ ๆ คือ

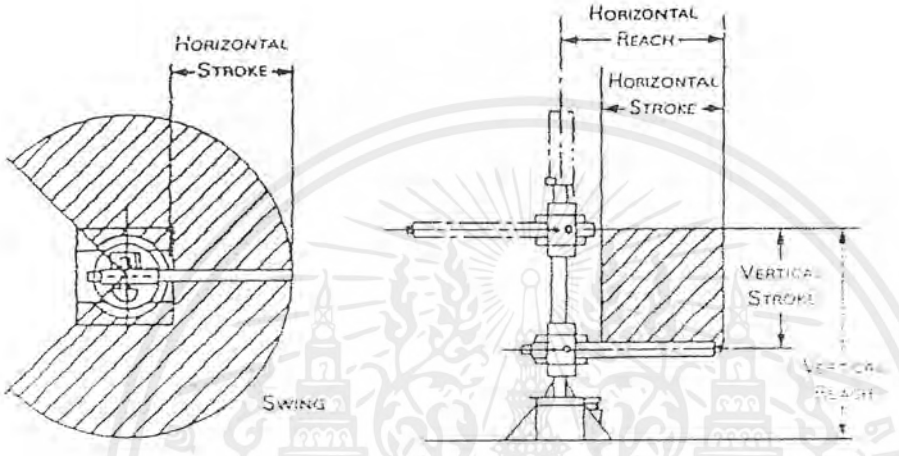
1. Polar Configuration.
2. Cylindrical Configuration.
3. Cartesian Configuration.
4. Articulated Configuration.
5. Scara

2.1.1 หุ่นยนต์ที่มีลักษณะเป็นแบบโพลาร์ (Polar) นั้นลักษณะการเคลื่อนที่ของแขนจะสามารถยกขึ้นลงได้ในแนวดิ่งโดยยกทำมุมกับฐาน สามารถหมุนได้รอบตัว (รูปที่ 2.1) พื้นที่การทำงานจะเป็นแบบทรงกลม ดังนั้นในบางครั้งจึงเรียกว่า "Spherical coordinate"



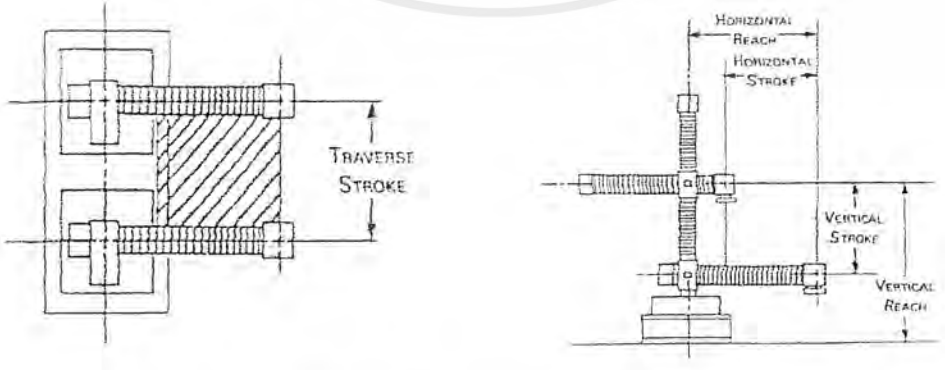
รูปที่ 2.1 หุ่นยนต์ที่มีลักษณะเป็นแบบโพลาร์ (Polar)

2.1.2 หุ่นยนต์ที่ลักษณะเป็นแบบไซลนดริคัล (*Cylindrical*) (รูปที่ 2.2) เคลื่อนที่ขึ้นลงได้ตามแกนตั้งที่เป็นแกนหลัก และสามารถเคลื่อนที่ไปมาได้โดยใช้แกนนอน แกนตั้งสามารถที่จะหมุนได้พื้นที่การทำงานจะเป็นแบบทรงกระบอก



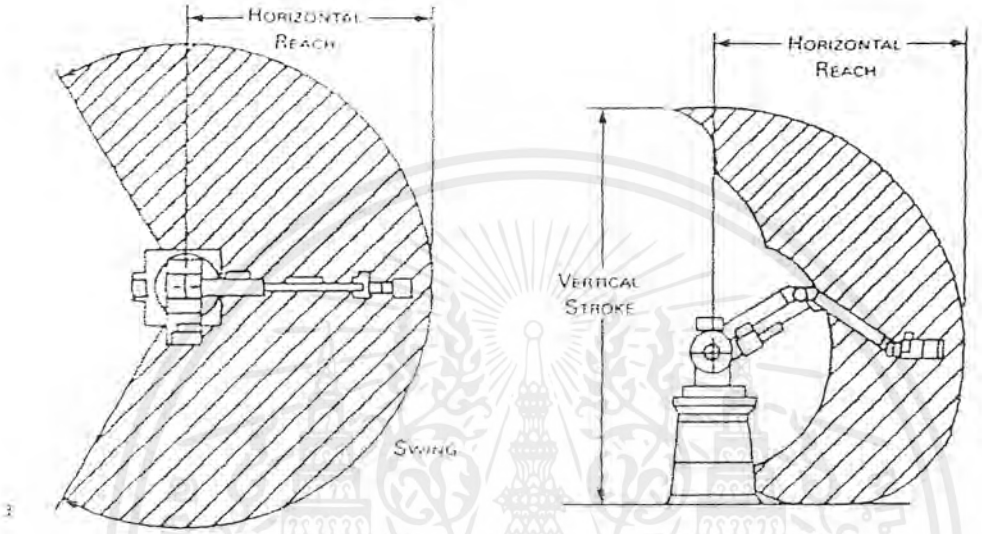
รูปที่ 2.2 หุ่นยนต์ที่ลักษณะเป็นแบบไซเลนดริคัล (Cylindrical)

2.1.3 หุ่นยนต์ที่มีลักษณะเป็นแบบคาร์ทีเซียน (Cartesian) (รูปที่ 2.3) ลักษณะการเคลื่อนที่จะมีแกน 3 แกนเป็นเหมือนแกน X,Y,Z ดังนั้นในบางครั้งจึงเรียกว่าหุ่นยนต์เรคติลิเนียร์ (Rectilinear Robot) พื้นที่การทำงานสามารถที่จะทำงานได้ในส่วนที่เป็นด้านของมือเพียงด้านเดียวเท่านั้นเพราะไม่มีการหมุนของฐาน



รูปที่ 2.3 หุ่นยนต์ที่มีลักษณะเป็นแบบคาร์ทีเซียน (Cartesian)

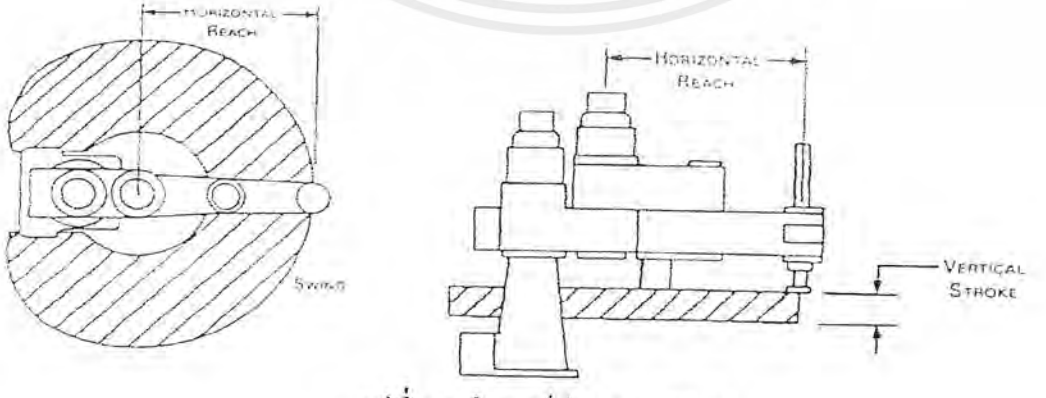
2.1.4 หุ่นยนต์ที่มีลักษณะเป็นอาร์ติคคูเลต (*Articulated*) หุ่นยนต์แบบนี้จะมีลักษณะใกล้เคียงกับแขนของมนุษย์มีข้อหมุนต่าง ๆ เหมือนกัน (รูปที่ 2.4) ดังนั้นพื้นที่การทำงานจึงสามารถที่จะทำงานได้ในทุกตำแหน่งในระยะความยาวของแขน



รูปที่ 2.4 หุ่นยนต์ที่มีลักษณะเป็นอาร์ติคคูเลต (Articulated)

2.1.5 หุ่นยนต์ประเภท สการา (Selective Compliance Automative Robot Arm,SCARA)

เป็นหุ่นยนต์แขนกลที่พัฒนามาจากหุ่นยนต์แขนกลแบบ อาร์ติคูเลต หรือทั่วไปที่เรียกว่าแขนกลแบบจอยท์ (Joint Arm) กับหุ่นยนต์แขนกลแบบ ไซเลนดิคอล ซึ่งทำให้การเคลื่อนที่ของหุ่นยนต์ประเภทนี้เป็นไปอย่างรวดเร็วและแม่นยำมีลักษณะการเคลื่อนที่ดังรูปที่ 2.5



รูปที่ 2.5 หุ่นยนต์ประเภท สการา

ข้อดีข้อเสียของแต่ละชนิดนี้แตกต่างกันออกไป เพราะว่าลักษณะทางกายภาพแตกต่างกัน แต่ถ้ามองในแง่ของการทำงานที่เป็นแบบซ้ำ ๆ ที่เดิมตลอด ชนิดคาร์ทีเซียน จะสามารถทำงานได้ดีกว่า (สามารถเคลื่อนที่ไปหาเป้าหมายโดยมีความผิดพลาดน้อยที่สุด) แต่ถ้ามองในแง่การเข้าถึงวัตถุชนิดที่เป็นแบบโพลาร์ และอาร์ติคูเลต จะสามารถเข้าถึงวัตถุได้ดีกว่าชนิดอื่น และชนิดที่เป็นแบบไซเลนดริคัล จะมีข้อดีในแง่ที่สามารถยกวัตถุได้มากกว่าในงานทั่วไปแล้วใช้แบบโพลาร์ และ ไซเลนดริคัล เพราะ 2 ชนิดนี้สามารถที่จะทำงานเป็นแบบ Load และ Unload โดยมีการเคลื่อนที่ของแขนออกไปในด้านข้างได้ดีกว่าชนิดอื่น ๆ

2.2 มือของหุ่นยนต์ (Robot end effector)

มือจับของหุ่นยนต์คือ ตัวที่ทำหน้าที่ขั้นสุดท้ายของหุ่นยนต์ มือจับของหุ่นยนต์สามารถแบ่งตามลักษณะของการทำงานได้ดังนี้

1. ทำงานเป็นตัวจับ (Gripper)
2. ทำหน้าที่เป็นเครื่องมือ (Tool)

มือของหุ่นยนต์ชนิดที่เป็นตัวจับมีหน้าที่โดยทั่วไปคือ การจับและการยกวัตถุสามารถแบ่งออกอีกเป็น 2 ชนิด คือ Single Gripper และ Double Gripper. Single gripper หมายถึงมือจับที่มีการเคลื่อนที่ของนิ้วเพียงข้างเดียวอีกข้างหนึ่งอยู่กับที่ และ Double gripper มีการเคลื่อนที่ของนิ้วทั้ง 2 ข้างเข้าหากันมือของหุ่นยนต์ชนิดที่เป็นตัวจับ สามารถแบ่งออกเป็นหลาย ๆ ชนิดตามลักษณะของตัวควบคุมได้ดังนี้

2.2.1 เมคคานิคกริปเปอร์ (Mechanical gripper)

เมคคานิคกริปเปอร์ คือมือจับที่มีกลไกทางเมคคานิคเป็นตัวควบคุมนิ้วบางครั้งเรียกว่าจอว์ส

(jaws) หน้าที่พื้นฐานของมือจับแบบเมคคานิค คือการส่งถ่ายกำลังจากมือไปสู่วัตถุต้นกำลังอาจใช้ไฟฟ้า, นิวเมติกส์, เชิงกลหรือไฮโดรลิค ลักษณะการจับของมือจับนี้สามารถที่จะออกแบบการจับได้ 2 อย่างคือ

1. ใช้ลักษณะทางกายภาพ เช่น ต้องการจับวัตถุเป็นทรงกลมตรงปลายมือก็ทำส่วนโค้งที่มีขนาดพอเหมาะกับวัตถุนั้น
2. ใช้ความเสียดทาน นิ้วไม่ได้ออกแบบให้มีลักษณะเหมือนกับรูปทรงของวัตถุแต่อาศัยแรงเสียดทานของนิ้วกับวัตถุโดยมีสมการของแรงเสียดทานคือ

$$\mu n_f F_g = w \tag{2.1}$$

เมื่อ μ = สัมประสิทธิแรงเสียดทานของนิ้วมือกับวัตถุ

n_f = จำนวนนิ้วของมือจับ

F_g = แรงจับ

w = น้ำหนักของวัตถุ

สมการ 2.1 นี้ใช้ เมื่อแรงโน้มถ่วงมีทิศทางขนานกับพื้นที่สัมผัส ถ้าทิศทางของแรงโน้มถ่วงเปลี่ยนไปจะใช้สมการ 2.2 คือ

$$\mu n_f F_g = mg \tag{2.2}$$

เมื่อ m = มวล

g คือ gravity factor และความเร่ง

$g = 1$ เมื่อความเร่งมีทิศทางตรงข้ามกับแรงโน้มถ่วง

$g = 2$ เมื่อความเร่งมีทิศทางในแนวราบ (ตั้งฉากกับแรงโน้มถ่วง)

$g = 3$ เมื่อความเร่งมีทิศทางเดียวกันกับแรงโน้มถ่วง

2.3 ความแม่นยำของการเคลื่อนที่

ความสำคัญอย่างหนึ่งของการสร้างหุ่นยนต์คือ ต้องการความแม่นยำในการทำงาน ความแม่นยำของหุ่นยนต์แต่ละตัวขึ้นกับตัวแปร 3 ประการคือ

1. สปาเชียนรีโซลูชั่น (Spatial resolution)
2. แอคคูเรซี (Accuracy)
3. รีพีททีบิลิตี้ (Repeatability)

2.3.1 สปาเชียนรีโซลูชั่น

คือช่วงการเคลื่อนที่ที่มีระยะทางสั้นที่สุดที่หุ่นยนต์แต่ละตัวสามารถที่จะทำได้ สปาเชี่ยนรีโซลูชั่น ขึ้นอยู่กับองค์ประกอบสำคัญ 2 ประการคือ

2.3.1.1 ระบบควบคุม (Control system) ระบบการควบคุมนี้จะรวมถึงการวัดสัญญาณป้อนกลับของหุ่นยนต์ด้วย ช่วงของการเคลื่อนที่ที่ระบบควบคุมสามารถที่จะทำได้ขึ้นอยู่กับหน่วยความจำหลักของเครื่องคอมพิวเตอร์เช่น เครื่องคอมพิวเตอร์ที่มีหน่วยความจำ 8 บิต จะสามารถที่จะแบ่ง

ช่วงการเคลื่อนที่ออกได้เป็น 256 ช่วง คือ ช่วงของการเคลื่อนที่ที่คอมพิวเตอร์แบ่งได้มีค่าเท่ากับ 2n เมื่อ n คือหน่วยความจำหลักของเครื่อง

2.3.1.2 ความคลาดเคลื่อนเชิงกล (Mechanical inaccuracy) ความคลาดเคลื่อนเชิงกลของหุ่นยนต์แต่ละตัวขึ้นอยู่กับลักษณะของข้อหมุน (joint) และข้อต่อ (link) และระบบต้นกำลังของหุ่นยนต์ตัวนั้นด้วย

2.3.2 แอคคูเรซี

แอคคูเรซี คือ ตัวที่แสดงถึงความสามารถของหุ่นยนต์ในการเคลื่อนที่เข้าใกล้จุดเป้าหมายตามที่เราสั่ง แอคคูเรซี สามารถที่จะกำหนดให้อยู่ในเทอมของ สปาเชียนรีโซลูชั่น ได้ทั้งนี้เพราะว่าการเคลื่อนที่ให้เข้าใกล้จุดเป้าหมายก็ต้องขึ้นอยู่กับช่วงของการเคลื่อนที่ว่ามีความละเอียดมากน้อยเพียงใด ในการทำงานเราต้องวางจุดที่เราต้องการให้หุ่นยนต์ทำงานอยู่ระหว่างกลางของตำแหน่งการเคลื่อนที่ของหุ่นยนต์ทั้งนี้ เพราะว่าความคลาดเคลื่อนเชิงกลมีผลต่อความแม่นยำของหุ่นยนต์

ความแม่นยำของหุ่นยนต์กำหนดให้เท่ากับครึ่งหนึ่งของระยะทางการเคลื่อนที่ที่สั้นที่สุดของหุ่นยนต์ที่สามารถทำได้ ความแม่นยำของหุ่นยนต์ขึ้นอยู่กับองค์ประกอบเหล่านี้คือ

1. พื้นที่การทำงานของหุ่นยนต์ ถ้าแขนทำงานในพื้นที่การทำงานจะมีความแม่นยำมากกว่าเมื่อแขนออกนอกพื้นที่การทำงาน
2. วงรอบการทำงาน ถ้าวงรอบการทำงานเป็นวงรอบที่แน่นอนความแม่นยำจะมีมากขึ้น
3. น้ำหนักที่ได้รับถ้าหุ่นยนต์ทำงานโดยการรับน้ำหนักมาก ๆ ความแม่นยำจะลดลง

2.3.3 รีพีททีบิลิตี้

รีพีททีบิลิตี้ คือ ความสามารถของหุ่นยนต์ในการกลับมาทำงานซ้ำที่เดิมจากรูป 2.1 จุดเป้าหมายคือ จุด T แต่เนื่องจากข้อจำกัดของหุ่นยนต์ หุ่นยนต์ไม่สามารถที่จะทำงานที่จุด T ได้ดังนั้นจึงทำงานที่จุด P ระยะห่างระหว่างจุด P กับจุด T คือตัวแสดงถึงความแม่นยำของหุ่นยนต์แต่เมื่อสั่งให้หุ่นยนต์กลับมาทำงานที่จุด P

อีกครั้งหนึ่งหุ่นยนต์ไม่สามารถกลับมาทำงานที่จุด P ได้จะทำงานที่จุด R แทนระยะห่างระหว่างจุด P และจุด R คือ ค่าแสดงถึงรีพีททีบิลิตี้

2.4 การเคลื่อนที่ของหุ่นยนต์

2.4 การเคลื่อนที่ของหุ่นยนต์สามารถแบ่งการเคลื่อนที่ออกได้ดังนี้ คือ

2.4.1 การเคลื่อนที่แบบเส้นตรง (Linear)

2.4.2 การเคลื่อนที่แบบหมุนรอบจุดหมุน (Rotational)

2.4.3 การเคลื่อนที่แบบบิดรอบจุดหมุน (Twisting)

2.4.4 การเคลื่อนที่แบบหมุนตั้งฉาก (Revolving)

2.5 การเคลื่อนที่ของมือ

2.5.1 หมุน (Rool) บางครั้งเรียกว่า Swivasl ข้อมือหมุนรอบแกนของแขน

2.5.2 บิด (Pitch) บางครั้งเรียกว่า Bend ข้อมือจะยกขึ้นลงในแนวดิ่ง

2.5.3 ส่าย (Yaw) หมายถึงการบิดไปมาทางซ้ายและขวาของแกนมือ

บทที่ 3
มอเตอร์ไฟฟ้ากระแสตรง

เนื่องจากในโครงการนี้ใช้มอเตอร์กระแสตรง 2 ตัว คือ มอเตอร์ขนาด 28 volts 18 amp 175 rpm และ มอเตอร์ขนาด 18 volts 24 amp 89 rpm เป็นตัวต้นกำลังของแขนกน ดังนั้นในบทที่จะกล่าวถึงต่อไปนี้ก็จะแสดงถึง หลักการทั่วไปของมอเตอร์ไฟฟ้ากระแสตรง คุณสมบัติของมอเตอร์ไฟฟ้ากระแสตรงและการทำงานของมอเตอร์ไฟฟ้ากระแสตรง [6]

3.1 หลักการทั่วไปของมอเตอร์ไฟฟ้ากระแสตรง

มอเตอร์ คือ เครื่องจักรที่เปลี่ยนพลังงานไฟฟ้าให้เป็นพลังงานกล เพื่อนำพลังงานที่ได้ไปขับเคลื่อนสิ่งต่างๆตามที่ต้องการ ซึ่งการเรียกว่ามอเตอร์ไฟฟ้ากระแสตรงเพราะว่าใช้ไฟฟ้ากระแสตรงนั่นเอง โดยพลังงานกลที่ได้นี้อาศัยหลักการดังนี้คือ เมื่อมีกระแสไหลในตัวนำซึ่งอยู่ในสนามแม่เหล็ก ย่อมทำให้เกิดแรงขึ้นในทิศทางที่แน่นอนได้จากกฎมือซ้ายของเฟลมมิ่ง โดยขนาดของแรงที่เกิดขึ้นหาได้จาก สมการ

$$F = B*I*L$$

โดย

F = แรงที่เกิดขึ้นบนตัวนำ (นิวตัน)

B = ความหนาแน่นสนามแม่เหล็ก (เวเบอร์/ตารางเมตร)

I = กระแสที่ไหลผ่านตัวนำ (แอมป์)

L = ความยาวของตัวนำ (เมตร)

แรงที่เกิดขึ้นจะอยู่ในแนวตั้งฉากกับสนามแม่เหล็กและกระแสที่ไหลผ่านในตัวนำเมื่อขดอาเมเจอร์ของสนามแม่เหล็กหลัก ซึ่งจะทำให้เกิดตัวนำที่อยู่ในขดลวดอาร์เมเจอร์ตัดเส้นแรงแม่เหล็กที่มาจากสนามแม่เหล็กหลัก ซึ่งจะทำให้เกิดการเหนี่ยวนำแรงเคลื่อนไฟฟ้าขึ้นบนตัวนำภายในบนตัวนำภายในอาร์มาเจอร์ซึ่งเป็นไปตามกฎการเหนี่วยนำแรงเคลื่อนแม่เหล็กไฟฟ้า ซึ่งแรงเคลื่อนไฟฟ้าที่เกิดขึ้นนี้จะมีทิศทางสวนกับแรงเคลื่อนไฟฟ้าที่ใส่เข้าไปให้กับมอเตอร์ จึงเรียกว่าแรงเคลื่อนไฟฟ้าสวนกับ E_b (back emf)

การหา กระแสที่ไหลในขดอาเมเจอร์ I_a หาได้จากวงจรสมมูลย์ของมอเตอร์ได้ดังรูปที่ 3.1 จะได้ว่า

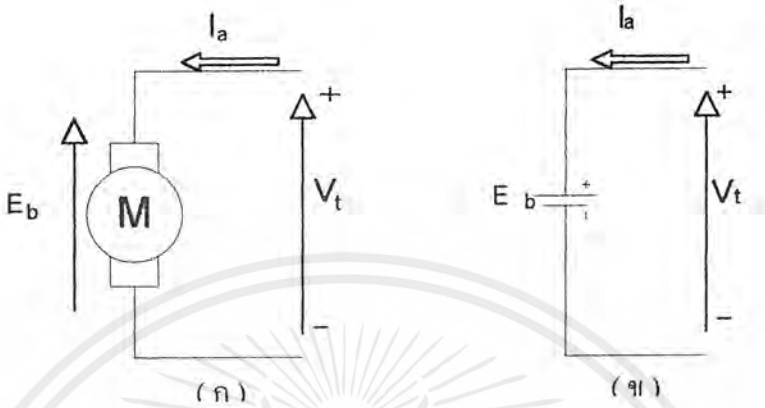
$$I_a = (V-E_b)/R$$

โดย V คือแรงเคลื่อนไฟฟ้าที่จ่ายให้กับมอเตอร์

$$T*2\pi S = E_b*I_b$$

โดย S = ความเร็วรอบของมอเตอร์ (รอบ/วินาที)

T = แรงบิดที่เกิดขึ้นใจขดอาร์เมเจอร์ (นิวตัน)

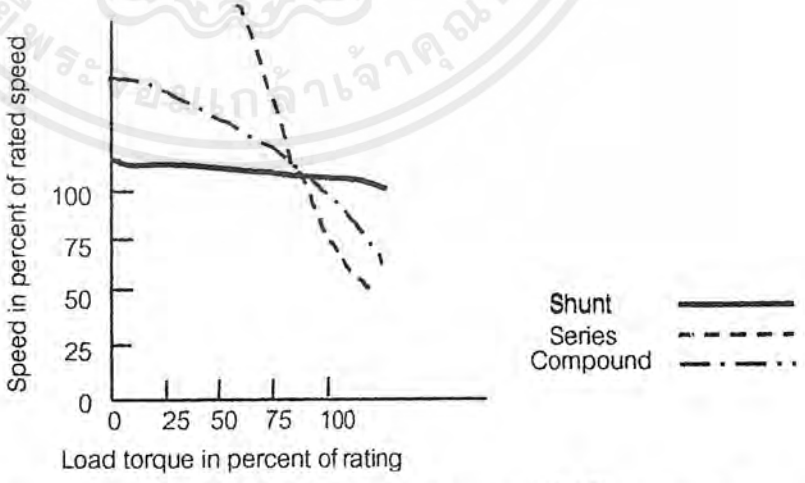


รูปที่ 3.1 (ก) วงจรสมมูลย์ของมอเตอร์ไฟฟ้ากระแสตรง

(ข) วงจรสมมูลย์ของมอเตอร์ที่เขียนแทนแรงเคลื่อนไฟฟ้าสวนกลับได้ด้วยแบตเตอร์รี่ E_b

3.2 คุณสมบัติของมอเตอร์ไฟฟ้ากระแสตรง

คุณสมบัติของมอเตอร์ไฟฟ้ากระแสตรงพิจารณาได้จากกราฟแสดงคุณสมบัติระหว่างความเร็วกับแรงบิดของมอเตอร์ไฟฟ้ากระแสตรง ดังรูปที่ 3.2



รูปที่ 3.2 กราฟแสดงคุณสมบัติระหว่างความเร็วกับแรงบิดของมอเตอร์ไฟฟ้ากระแสตรง

จากกราฟแสดงคุณสมบัติระหว่างความเร็วกับแรงบิดของมอเตอร์ไฟฟ้ากระแสตรงแบบขนานจะมีลักษณะค่อนข้างเป็นเส้นตรง โดยขณะที่ความเร็วลดลง แรงบิดจะเพิ่มขึ้น เหมาะกับงานที่ต้องการ สปีดเร็กกูเรชัน (speed regulation) ดีๆ

จากกราฟแสดงคุณสมบัติระหว่างความเร็วกับแรงบิดของมอเตอร์ไฟฟ้ากระแสตรงแบบอนุกรม นะพบว่า เราสามารถเพิ่มแรงบิดได้โดยการเพิ่มกระแสอาเมเจอร์ซึ่งจะส่งผลให้ฟลักซ์แม่เหล็กเพิ่มขึ้นด้วย เมื่อฟลักซ์แม่เล็กเพิ่มขึ้นความเร็วจะต้องตกลงจนถึงจุดสมดุลย์ ระหว่างแรงดันไฟฟ้าเหนี่ยวนำกับแรงดันไฟฟ้าของแหล่งจ่าย จากคุณสมบัตินี้ทำให้ เส้นกราฟของมอเตอร์ไฟฟ้ากระแสตรงแบบอนุกรม ดังรูปมีความชันสูง ตามมาตรฐานแล้ว มอเตอร์ชนิดนี้จะถูกออกแบบให้ทำงานที่จุดหักมุมของ magnetization characteristic ที่แรงบิดที่พิกัด ฉะนั้นเมื่อแรงบิดโอเวอร์โหลอมากๆจะทำให้แกนเหล็กอิ่มตัวได้

มอเตอร์ไฟฟ้ากระแสตรงอนุกรมเหมาะสมกับการนำไปใช้งานที่ต้องการ แรงบิดขณะสตาร์ทสูงๆ และมีแรงบิดที่โอเวอร์โหลดมากๆ โดยความเร็วจะเพิ่มขึ้นเมื่อแรงบิดลดลง มอเตอร์ไฟฟ้ากระแสตรงแบบอนุกรมสามารถที่จะนำไปประยุกต์ใช้ขับโหลดที่ต้องออกตัวบ่อยๆและแรงบิดโอเวอร์โหลดบ่อยๆ

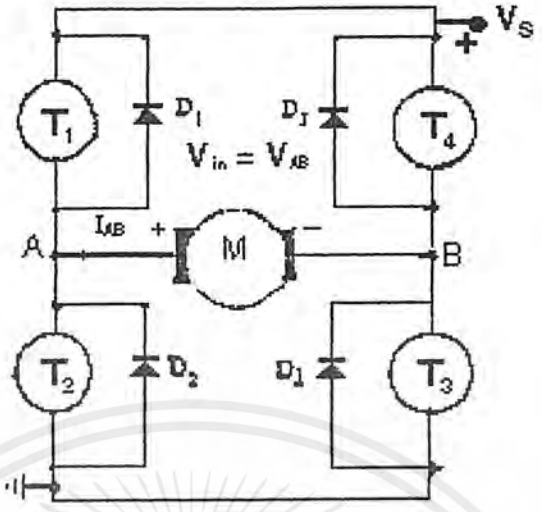
จากกราฟแสดงคุณสมบัติระหว่างความเร็วกับแรงบิดของมอเตอร์ไฟฟ้ากระแสตรงแบบ compound ซึ่งความเร็วขณะไม่มีโหลด (no-load) จะขึ้นอยู่กับ shunt field และความเร็วที่ตกลงมาจะขึ้นอยู่กับ series field ดังนั้นมอเตอร์ชนิดนี้จึงนำมาประยุกต์ใช้งานที่มีคุณสมบัติคล้ายคลึงกับมอเตอร์ไฟฟ้ากระแสตรงแบบอนุกรม โดยความเร็วขณะไม่มีโหลด จะถูกจำกัดไว้ในค่าที่ปลอดภัยโดย shunt field นอกจากนี้ยังสามารถนำไปประยุกต์ใช้งานซึ่งโหลดมีค่าเปลี่ยนแปลงมากๆ

3.3 การทำงานของมอเตอร์กระแสตรง

ระบบ PWM ปกติจะใช้แหล่งจ่ายไฟกระแสตรงและ amplifier เป็นตัวสวิตชิ่งซัพพลายโวลท์เตจให้ on และ off ที่ความถี่คงที่และมีช่วงเวลาของการ on ที่ปรับค่าได้ในหนึ่งคาบหรือที่เรียกว่าการปรับ duty cycle การปรับ duty cycle นั้นเป็นการปรับค่าเฉลี่ยของ voltage ที่จ่ายให้แก่โหลดได้ ในที่นี้ก็คือมอเตอร์กระแสตรงนั้นเอง เมื่อโหลดเพิ่มขึ้นความถี่ของ PWM amplifier จะคงที่แต่ละเปลี่ยนค่า duty cycle ตามโหลด สวิตชิ่ง amplifier สามารถควบคุมความเร็วต่ำ ๆ โดยมีแรงบิดสูงอยู่โดยไม่สิ้นเปลืองพลังงานเหมือนพวก amplifier เชิงเส้น amplifier แบบ PWM สามารถทำงานได้ 3 ลักษณะ คือ

1. ไบโพล่าร์
2. ยูนิโพล่าร์
3. ลิมิตยูนิโพล่าร์

สำหรับแบบ ไบโพล่าร์เป็นแบบที่ง่ายที่สุดดูการทำงานตามรูปที่ 3.3



รูปที่ 3.3 แสดง PWM amplifier สำหรับมอเตอร์กระแสตรง

ซึ่งส่วนประกอบของรูปที่ 3.3 นั้นจะประกอบอยู่ในส่วนวงจรขับมอเตอร์ที่ใช้ใน โครงงาน

โดยที่เราจะกำหนดให้มีความถี่การสวิตช์ให้เป็น f_s โดยให้ t_{on} เกิดขึ้นในช่วงแรกของคาบและ t_{off} เกิดขึ้นในช่วงหลัง

t_{on} เมื่อ 0

t_{off} เมื่อ

ไบโพล่าร์จะมี T_1 และ T_4 นำกระแสระหว่างเฟส on และส่วน T_2 และ T_3 จะนำกระแสขณะเฟส off จะได้ function ตกคร่อมมอเตอร์เป็น

$$V_{MOTOR} = V_{AB} = \begin{cases} V_s; 0 > t > t_1 \\ V_s; t_1 > t > t_f \end{cases}$$

ยูนิโพล่าร์จะลดจำนวนทรานซิสเตอร์ในการสวิตช์ลง การสวิตช์ขึ้นกับ V_{in} เป็นบวก หรือ ลบ เมื่อ V_{in} เป็นบวก T_4 จะนำกระแสตลอดคาบในขณะที่ T_1 นำกระแสในช่วงเฟส on และ T_2 จะนำกระแสในช่วงเฟส off เมื่อ V_{in} เป็นลบ T_2 จะนำกระแสตลอดโดยมี T_3 และ T_4 สลับกันทำงาน ถ้า V_{in} เป็นบวกจะได้

$$V_{in} = \begin{cases} V_s & \text{เมื่อ } t_{on} \\ 0 & \text{เมื่อ } t_{off} \end{cases}$$

การแสดงค่าในทางลบจะเหมือนกันเพียงแต่เป็นลบเท่านั้น

จากลักษณะของ 2 แบบ ที่กล่าวมานั้นเป็นประโยชน์เหมือนกันซึ่งในแต่ละกรณีจะมีทรานซิสเตอร์คู่หนึ่ง T_1 , T_2 หรือ T_3 , T_4 จะหยุดนำกระแสขณะที่อีกคู่หนึ่งนำกระแส ซึ่งมีเวลาเก็บสะสมและเวลาที่ปล่อยออกของทรานซิสเตอร์เกิดขึ้นและมันอาจเป็นไปได้ที่ทรานซิสเตอร์ทั้งหมดนำกระแสในเวลาเดียวกัน ซึ่งจะทำให้เกิดการลัดวงจรของแหล่งจ่ายไฟ เราจำเป็นต้องหลีกเลี่ยงสภาวะดังกล่าวซึ่งสามารถทำได้โดยการสร้างช่วงเวลาหน่วง delay time ระหว่างการหยุดและนำกระแสของทรานซิสเตอร์และด้วยเหตุผลดังกล่าวความถี่ของการสวิตช์จะถูกจำกัดในวงที่แคบลง

ลิมิตยูนิโพล่าร์ จะแสดงให้เห็นคือมีความจำเป็นต้องมีช่วงเวลาหน่วงซึ่งการสวิตช์ขึ้นกับค่า V_{in} เมื่อ Vin เป็นบวก T_4 จะนำกระแสตลอด โดย T_1 จะเป็นสวิตช์ on ในช่วงเฟส on ดังนั้นในช่วงเฟส on ทั้ง T_1 และ T_4 จะ on ยังผงแก่โวลท์เตจของมอเตอร์ V_{in} คือ

$$V_{motor} = V_s \quad \text{เมื่อ} \quad t_{on}$$

ระหว่างเฟส off จะมี T_4 นำกระแสเพียงตัวเดียวเป็นผลให้ V_{in} ขึ้นกับ I_{AB} ตราบใดที่ $I_{AB} > 0$ ซึ่งเป็นสภาวะปกติเมื่อ $V_{AB} > 0$ กระแส I_{AB} จะไหลผ่าน D_2 และ T_4 เป็นผลให้ $V_A = 0$ และ

$$V_{motor} = V_{AB} \quad \text{เมื่อ} \quad t_{off} \text{ และ } I_{AB} > 0$$

ในกรณีที่ I_{AB} เป็นลบกระแสจะไหลผ่าน D_1 และ D_4 เป็นผลให้ $V_A = V_S$ และ

$$V_{motor} = V_{AB} \quad \text{เมื่อ} \quad t_{off} \text{ และ } I_{AB} < 0$$

ซึ่งจะเกิดขึ้นภายหลังเปลี่ยนขั้ว V_{in}

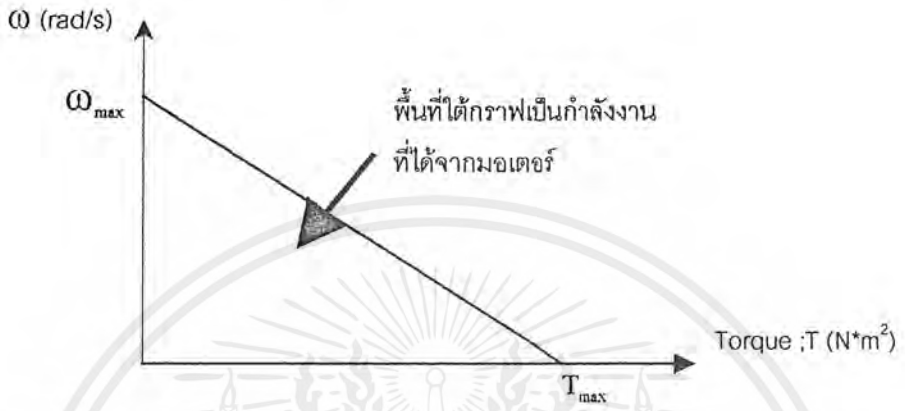
ในที่สุด ถ้าเราสามารถทำให้ $I_{AB} = 0$ (เข้าใกล้ศูนย์จนถือว่าเป็นศูนย์) จะทำให้ ทั้ง D_1 และ D_4 ไม่นำกระแสและโวลท์เตจ V_{in} จะอยู่ระหว่างค่าศูนย์และ V_S ดังต่อไปนี้

$$0 < V_{motor} \quad \text{เมื่อ} \quad t_{off} \text{ และ } I_{AB} = 0$$

อย่างไรก็ตาม ถ้า $I_{AB} > 0$ เป็นสภาวะปกติ เมื่อ $V_{in} > 0$ แบบยูนิโพล่าร์และแบบลิมิตยูนิโพล่าร์จะแสดงคุณสมบัติคล้ายกันมาก ซึ่ง เราสามารถสรุปโหมดการทำงานและผลของโวลท์เตจ

บทที่ 4
การออกแบบและการคำนวณ

4.1. คุณสมบัติของมอเตอร์



รูปที่ 4.1 กราฟความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิดของมอเตอร์

ในกราฟข้างบนนี้จะเป็นกราฟที่ทางบริษัทผลิต DC motor จะต้องทำการทดลองและสร้างกราฟขึ้นมาเพื่อแสดงคุณสมบัติ และประสิทธิภาพของ DC motor ที่ได้ทำการผลิต

- ถ้ามีการทดอัตราทดเฟืองโดยเพิ่มทอร์ก (T) เพิ่มขึ้น ค่าทอร์กสูงสุด (T_{max}) จะเพิ่มขึ้น และจะทำให้ ความเร็วรอบสูงสุด (ω_{max}) ลดลง
- ถ้ามีการทดอัตราทดเฟืองโดยลดทอร์ก (T) น้อยลง ค่าทอร์กสูงสุด (T_{max}) จะลดลง และจะทำให้ความเร็วรอบสูงสุด (ω_{max}) เพิ่มขึ้น

โดยที่พื้นที่ใต้กราฟจะเป็น กำลังงานทั้งหมดที่ได้จากมอเตอร์[6]

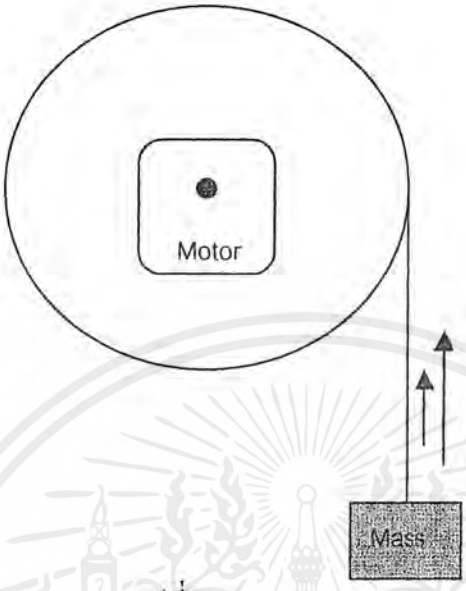
$$P = \int \omega dT \quad \text{(Watt.)}$$

การทดอัตราทดเฟืองโดยเพิ่มทอร์ก (T) ให้สูงขึ้นจะทำให้ความเร่งเชิงมุม (α) มากขึ้นด้วย ซึ่งมอเตอร์จะสามารถเพิ่มความเร็วให้ถึงจุดที่ความเร็วคงที่ หรือ ความเร็วสูงสุดได้เร็วขึ้น โดยสามารถพิจารณาความสัมพันธ์ของค่าทอร์ก (T) กับค่าความเร่งเชิงมุม (α)[7] ได้จากสมการ

$$T = J\alpha \quad (N*m^2)$$

ซึ่งคุณสมบัติขั้นต้นของ DC motor ที่ได้กล่าวมาแล้ว จะต้องทำการทดลองเนื่องจากมอเตอร์ที่หาได้ ไม่ได้เป็นมอเตอร์ใหม่ที่มีคุณภาพที่ได้ถูกรับรองมาจากบริษัทที่ได้ทำการผลิตและได้ทำการทดลองมาแล้ว แต่ DC motor ที่ใช้เป็นมอเตอร์สภาพเก่าผ่านการใช้งานมานานพอสมควร ทำให้จะต้องมีการทำการทดลอง ซึ่งการทดลองที่จะใช้ในการทดลอง DC motor ที่ใช้นี้จะเป็นการทดลองแบบง่ายๆ โดยจะอธิบายขั้นตอนการทดลอง และวิธีการทดลองได้ดังต่อไปนี้

4.2 การทดลองและคำนวณเพื่อหาทอร์กสูงสุด (T_{max})ของDC motor ที่ใช้



รูปที่ 4.2 แสดงการทดลองมอเตอร์

1. ทำการประดิษฐ์อุปกรณ์ทดลองแบบง่ายๆ โดยใช้รอกหมุน ซึ่งมีรัศมี (r) เท่ากับ 210 mm. โดยมีเชือกเป็นตัวยกมวล
2. นำ DC motor ที่ต้องการทดลองมาติดกับชุดทดลอง
3. นำมวลที่มีน้ำหนักมากที่สุด (10 kg.) มาติดไว้ที่ชุดทดลอง
4. ทำการจ่ายกระแสไฟฟ้าให้กับมอเตอร์โดยใช้ DC power supply ทำการเพิ่มค่า voltage ขึ้นอย่างต่อเนื่องอย่างช้าๆ จนถึงค่า voltage ที่มอเตอร์สามารถจะรับได้ และถ้า DC มอเตอร์ ยังไม่สามารถยกมวลที่ใช้ให้ยกขึ้นได้ จะต้องลดมวลลงมาทีละน้อย จน DC motor จะสามารถยกมวลขึ้นได้ บันทึกผลการทดลอง
5. ทำการทดลอง DC motor ตัวอื่นๆต่อไป แล้วทำผลการทดลองที่ได้ไปคำนวณหาค่าทอร์ก (T) ซึ่งค่าทอร์ก (T) ที่ได้จะเป็นค่าทอร์กสูงสุด (T_{max}) ที่ DC motor ตัวนั้นจะสามารถให้ได้ ซึ่งจะทำให้สามารถนำไปสร้างกราฟความสัมพันธ์ระหว่างความเร็วเชิงมุมกับทอร์กได้

ข้อควรระวัง ในการจ่ายกระแสไฟฟ้าให้กับ DC motor จะต้องระวังไม่ให้ค่ากระแส (I) เกินจากข้อกำหนดของขนาดมอเตอร์แต่ละตัว เพราะเมื่อมีโหลดเพิ่มมากขึ้นเท่าไร ค่ากระแสที่ต้องจ่ายให้ก็จะต้องจ่ายให้มากขึ้นด้วย เพราะฉะนั้น การเพิ่มกระแส (I) ควรจะเพิ่มขึ้นทีละน้อยแต่จะต้องเพิ่มอย่างต่อเนื่องด้วย

4.3 ตารางผลการทดลอง

DC motor 25 v. 175 rpm 18 A.

ครั้งที่	น้ำหนัก (kg)	ทอร์ก (T) ($N*m^2$)	ยกได้หรือไม่	กระแส (I) ที่จะต้องให้ (A)	กำลังงานที่ให้ (Watt)
1	10	20.601	ไม่ได้	-	-
2	9	18.54	ไม่ได้	-	-
3	8	16.48	ไม่ได้	-	-
4	7	14.42	ไม่ได้	-	-
5	6	12.36	ได้	7	168
6	6.5	13.39	ได้	8	192
7	6.7	13.8	ได้	8	192

ตารางที่ 4.1 ตารางผลการทดลองมอเตอร์กระแสตรง 25 v. 175 rpm 18 A.

DC motor 18 v. 59 rpm 26 A.

ครั้งที่	น้ำหนัก (kg)	ทอร์ก (T) ($N*m^2$)	ยกได้หรือไม่	กระแส (I) ที่จะต้องให้ (A)	กำลังงานที่ให้ (Watt)
1	10	20.601	ไม่ได้	-	-
2	9	18.54	ไม่ได้	-	-
3	8	16.48	ไม่ได้	-	-
4	7	14.42	ได้	5	90
5	7.5	15.45	ได้	5	90
6	7.8	16.07	ได้	5	90

ตารางที่ 4.2 ตารางผลการทดลองมอเตอร์กระแสตรง 18 v. 59 rpm 26 A.

4.4 แสดงการคำนวณ

DC motor 28 v. 175 rpm 18 A.

จาก $\omega_{max} = 2\pi f$

$= 2\pi * 175/60$ rpm

$= 18.326$ rad/s

พลังงานที่ได้ $P = \int \omega dT$

$\approx 0.5\omega T$

$\approx 0.5 * 18.326 * 13.8$

≈ 126.449 w.

$\therefore$ ประสิทธิภาพ ของ DC motor $\eta = 126.449/192 = 0.66 = 66\%$

DC motor 18 v. 59 rpm 26 A.

จาก $\omega_{max} = 2\pi f$

$= 2\pi * 59/60$ rpm

$= 6.178$ rad/s

พลังงานที่ได้ $P = \int \omega dT$

$\approx 0.5\omega T$

$\approx 0.5 * 6.178 * 16.07$

≈ 49.6365 w.

$\therefore$ ประสิทธิภาพ ของ DC motor $\eta = 49.6365/90 = 0.55 = 55\%$

4.5 สมการการเคลื่อนที่

กฎการเคลื่อนที่ข้อที่ 2 ของนิวตันกล่าวไว้ว่า " ความเร่งของอนุภาคเป็นสัดส่วนโดยตรงกับแรงลัพท์ที่กระทำ และมีทิศทางเดียวกันกับแรงลัพธ์นั้น " [7]

$$\sum F = m*a$$

a คือ ความเร่งของอนุภาคซึ่งมีค่าคงที่ หรือแปรผันขึ้นอยู่กับแรงลัพธ์ $\sum F$

เป็นระบบการเคลื่อนที่ระบบแกน n-t ; $\sum F_n = m*a_n$

$$\sum F_t = m^* a_t$$

เมื่อ $a_n = \rho\dot{\theta}^2 = v^2/\rho = v\dot{\theta}$

$a_t = \dot{v} = \dot{s} = \rho\ddot{\theta}$

$a = (a_n^2 + a_t^2)^{0.5}$

โดยที่ $v = \dot{s}$ $ds/dt = \bar{\rho}d\theta/dt = \bar{\rho}\dot{\theta}$

เมื่อ $\bar{\rho}$ คือ รัศมีความโค้งที่ตำแหน่งพิจารณา

a_n คือความเร่งในแนว n

a_t คือความเร่งในแนว t

v คือ ความเร็วในแนวสัมผัสหรือในแนว t

$\dot{\theta}$ คือ ความเร็วเชิงมุมหรือ ω

$\ddot{\theta}$ คือ ความเร่งเชิงมุมหรือ α

$v = r\dot{\theta}$

$a_n = v^2/r = r\dot{\theta}^2 = v\dot{\theta}$

$a_t = \dot{v} = r\ddot{\theta}$

$[\sum F_t = ma_t]$; $F_t = mr\alpha$

$[\sum F_n = ma_n]$; $F_n = mr\omega^2$

การหมุนรอบแกนคงที่

สมบัติของการเคลื่อนที่เชิงมุมของเส้นจะได้เป็น

$$\omega = d\theta/dt = \dot{\theta}$$

$$\alpha = d\omega/dt = \dot{\omega}$$

หรือ $\alpha = d^2\theta/dt^2 = \ddot{\theta}$

$$\omega d\omega = \alpha d\theta$$

หรือ $\dot{\theta}d\dot{\theta} = \theta d\ddot{\theta}$

เมื่อ θ คือ การขจัดเชิงมุม

ω คือ ความเร็วเชิงมุม

α คือ ความเร่งเชิงมุม

การเคลื่อนที่เชิงมุมจะได้เป็น

$$v = \omega r$$

$$a_n = v\omega^2 = v^2/r = v\omega$$

$$a_t = r\alpha$$

หรือ เขียนในรูปเวกเตอร์ ได้เป็น

$$\overline{v} = \omega * \overline{r}$$

$$a_n = \omega * (\omega * r) = \omega * v$$

$$a_t = \alpha * r$$

$$a = a_n + a_t$$

Polar Moment of Inertia of a Plan Area

Polar Moment of Inertia ของพื้นที่ราบคือ Moment of Inertia ของพื้นที่ราบ รอบแกน ซึ่งตั้งฉากกับพื้นที่ราบ ณ จุดที่แกนต่อกับพื้นที่ราบนั้น[7]

$$J = \int r^2 dA$$

เมื่อ $r^2 = x^2 + y^2$

$$J = \int r^2 dA$$

$$= \int (x^2 + y^2) dA$$

$$= \int y^2 dA + \int x^2 dA$$

$$\therefore \quad J = I_x + I_y$$

4.6 Moment of Inertia of masses

$$I_2 = \int r^2 dm$$

$$= \int (y^2 + x^2) dm$$

$$= \int y^2 dm + \int x^2 dm$$

4.7 อัตราการทดเฟือง

<u>อัตราการทดเฟือง ที่ DC motor 28 v. 195 rpm 18 A</u>

(ควบคุมการหมุนของแกนเพลาหลัก)

จะทดในอัตรา 1:6.4

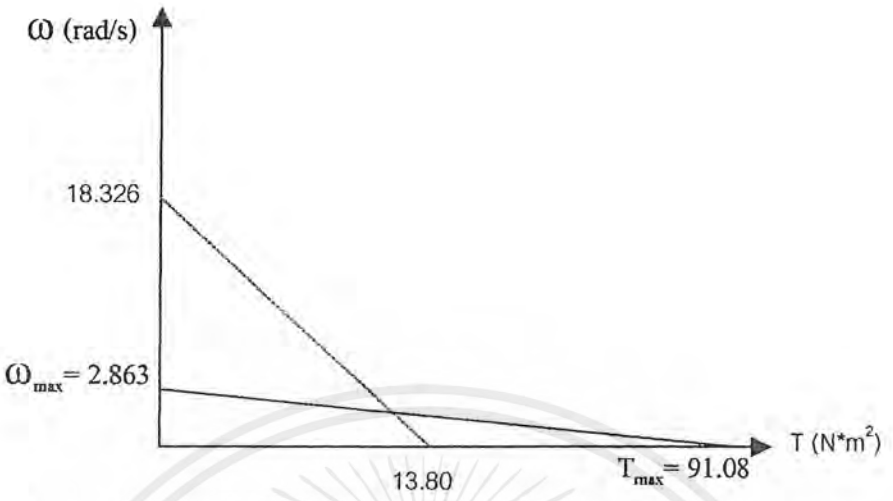
ซึ่งจะทำให้ทอร์กสูงสุด (T_{max}) จะเพิ่มขึ้น $= 6.4 * 13.8$ N*m

$= 91.08$ N*m

ความเร็วเชิงมุมสูงสุด(ω_{max}) จะลดลงเป็น $= (1/0.64) * 18.326$ rad/s

$= 2.863$ rad/s หรือ 27.3438 rpm

ความเร่งสูงสุด $\alpha_{max} = T_{max}/J_{รวม} = ?$ (จะทำการคำนวณในช่วงต่อไป)



รูปที่ 4.3 กราฟความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิดของ DC Motor 28 V

ค่าทอร์กสูงสุด (T_{max}) ใหม่ที่ได้จากการทดเฟืองจะเห็นได้ว่าเพลาหลักที่ได้รับการทดเฟืองแล้วจะสามารถหมุนวัตถุมวล 9.1 kg ที่ระยะ (r) 1 m. ได้แสดงว่าสามารถหมุนแขนช่วงที่ 1 และช่วยที่ 2 ที่มีน้ำหนัก 6 kg ให้เคลื่อนที่ไปได้ ซึ่งความเร็วในการเคลื่อนที่เชิงมุมนั้นจะสามารถควบคุมได้จากโปรแกรมควบคุมโดยผ่านระบบควบคุมได้

<u>อัตราการทดเฟือง ที่ DC motor 18 v. 59 rpm 18A</u>

(ควบคุมการเคลื่อนที่ของแขนท่อนที่ 2)

จะทดในอัตรา 1:2.4

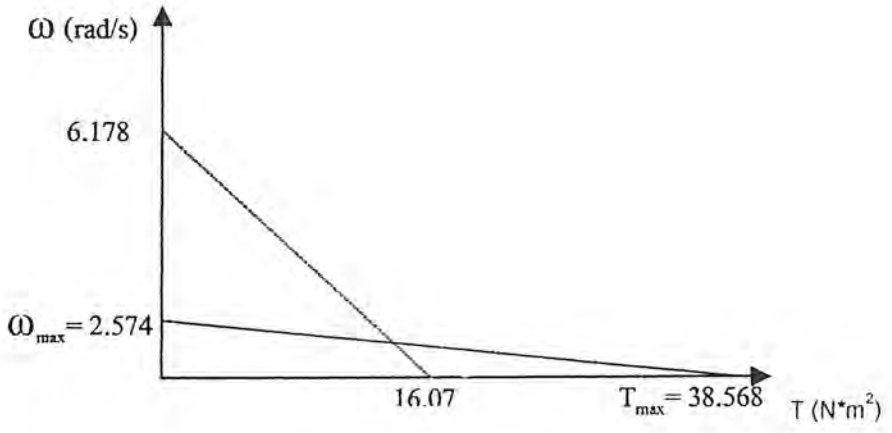
ซึ่งจะทำให้ทอร์กสูงสุด (T_{max}) จะเพิ่มขึ้น = 2.4*16.07 N*m

= 38.568 N*m

ความเร็วเชิงมุมสูงสุด(ω_{max}) จะลดลงเป็น = (1/2.4)*6.178 rad/s

= 2.574 rad/s หรือ 24.58 rpm

ความเร่งสูงสุด $\alpha_{max} = T_{max}/J_{ท่อนที่ 2} = ?$ (จะทำการคำนวณในช่วงต่อไป)



รูปที่ 4.4 กราฟความสัมพันธ์ระหว่างความเร็วรอบกับแรงบิดของ DC Motor 18 V

ค่าทอร์กสูงสุด (T_{max}) ใหม่ที่ได้จากการทดเฟือง 1:2 จะเห็นได้ว่าจะสามารถเคลื่อนที่วัตถุในการหมุนเชิงมุมโดยวัตถุมวล 3.8 kg ที่ระยะ (r) 1 m. ได้
ในแขนท่อนที่ 2 มีระยะ ® = 0.3 m. มีน้ำหนัก = 1.3 kg.

เพราะฉะนั้น DC motor ตัวนี้สามารถทำให้แขนท่อนที่ 2 เคลื่อนที่ได้อย่างแน่นอน ซึ่งความเร็วแสดงความเร่งที่แขนจะเคลื่อนที่เชิงมุมจะถูกควบคุมโดยโปรแกรมควบคุม

4.8 การคำนวณค่า Moment of Inertia (J) เพื่อนำไปหาความเร่งเชิงมุม (α) ได้ต่อไป

<u>พิจารณาที่แขนท่อนที่ 1</u> เนื่องจากมวลที่ได้จากการชั่งน้ำหนักเป็นมวลรวมทั้งหมดของชิ้นส่วนที่นำมาประกอบรวมกัน ทำให้ต้องพิจารณารูปทรงเป็นทรงสี่เหลี่ยมปริซึม กลวงตลอดความยาวของแขนท่อนที่ 1

$$J_{1(\text{ท่อนที่1})} = (m/6)*(b^2+l^2)+ md^2$$
$$= [(1/6)*3.25*(0.1^2+0.59^2)]+[3.25*(0.8)^2]$$
$$= 0.1634 + 0.0208$$
$$= 0.1842\ kg*m^2$$

<u>มอเตอร์</u>

$$J_{2(\text{ท่อนที่1})} = 0.5mr^2+md^2$$
$$= [0.5*1.45*(0.04)^2]+[1.45*(0.1)^2]$$
$$= 1.16^{-0.3}+0.0145$$
$$= 0.01566\ kg*m^2$$

พิจารณาแขนท่อนที่ 2

ค่า Moment of inertia ที่ได้จากแขนท่อนที่ 2 ที่กระทำต่อแกนเพลาหลักซึ่งส่งผ่านแรงโดย DC motor 28 v.175 rpm นั้นจะสามารถเปลี่ยนแปลงไปได้โดยจะขึ้นอยู่กับมุมองศาที่แขนท่อนที่ 2 หมุนเคลื่อนที่ไป เนื่องจากเมื่อแขนท่อนที่ 2 หมุนไปจะทำให้จุดศูนย์กลางมวล (จุด centroid) ของแขนท่อนที่ 2 มีระยะเปลี่ยนแปลงไปเมื่อเทียบกับแกนเพลาหลักซึ่งทำหน้าที่เป็นแกนหมุน ค่า moment of Inertia ที่ได้ซึ่งขึ้นอยู่กับระยะนี้ด้วยก็จะเปลี่ยนแปลงไปเรื่อยๆ เพราะฉะนั้นเราจะติดค่า moment of Inertia ที่ได้จากแขนท่อนที่ 2 ในรูปของฟังก์ชั่นของมุม (θ_2) จะเขียนได้เป็น

$$J_{\text{ท่อนที่ 2}} = J(\theta_2) \quad kg*m^2$$

สรุปจะได้ Moment of inertia ที่กระทำรอบแกนเพลาหลักจะได้เป็น

$$\begin{aligned} J &= J_{1(\text{ท่อนที่ 1})} + J_{2(\text{ท่อนที่ 1})} + J_{\text{ท่อนที่ 1}} \\ &= 0.1892 + 0.01566 + J(\theta_2) \quad kg*m^2 \\ &= 0.19986 + J(\theta_2) \quad kg*m^2 \end{aligned}$$

ซึ่งมีค่า Moment of inertia รอบแกนเพลาหลักก็จะอยู่ในฟังก์ชั่นของ θ_2

จาก $T = J\alpha$

$$\alpha_{max} = T_{max} / J_{\text{รวม}} = 85.98/[\,0.19986 + J(\theta_2)\,] \quad rad/s^2$$

DC Motor 18V 59 rpm

$$\begin{aligned} J &= 1/6[m(b^2 + l^2)] + md^2 \\ &= [\,1/6 * 1.3*(0.1^2 + 0.3^2)\,] + [\,1.3*(0.1)^2\,] \\ &= 0.03467 \quad kg*m^2 \end{aligned}$$

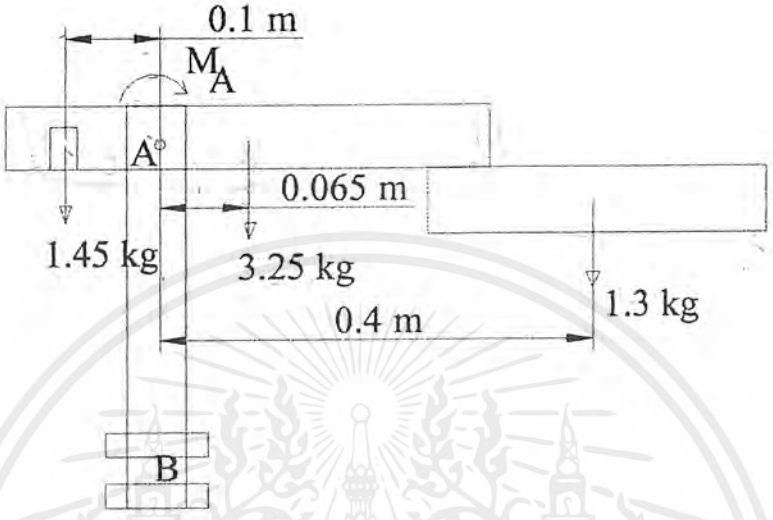
จากความสัมพันธ์ $T = J\alpha$

$$\alpha = T/J$$

$$\begin{aligned} \alpha_{max} &= T_{max}/J \\ &= 39.28 /0.03467 \quad N*M/kg*m^2 \\ &= 1132.97 \quad rad/s^2 \end{aligned}$$

4.9 การคำนวณเกี่ยวกับโครงสร้าง

4.9.1 พิจารณาที่เพลาใหญ่ จากรูปtake moment ที่จุด A โดยพิจารณาเป็น fix support เพื่อหา โมเมนต์ดัดที่กระทำต่อเพลา



รูปที่ 4.5 รูปประกอบการคำนวณหา M_A และแรงในแนวดิ่ง

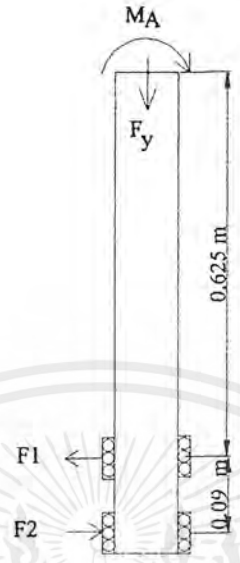
$$M_A + (3.25*9.81*0.065) + (1.3*9.81*0.4) - (1.45*9.81*0.1) = 0$$

$$M_A = -5.75\ N*m$$

เพราะฉะนั้น M_A จะมีทิศ ทวนเข็มนาฬิกา

$$\sum F_y = 0, \qquad F_y = (3.25+1.3+1.45)*9.81 = 58.86\ N$$

4.9.2 คิดที่เพลาแรงปฏิกิริยาจะมีทิศทาง- y และ โมเมนต์เป็น ตามเข็ม take moment ที่แบริ่ง B



รูปที่ 4.6 แรงที่กระทำต่อเพลาหลัก

$$F_1 * 0.09 = 5.75$$

$$F_1 = 63.88 \text{ N}$$

$$\sum F_x = 0, F_1 = F_2 = 63.88 \text{ N} \quad \text{ทิศตรงกันข้ามกัน}$$

F_A และ F_B คือแรงกระทำที่แบริ่ง

เราจะได้ความเค้นที่เพลา คือ

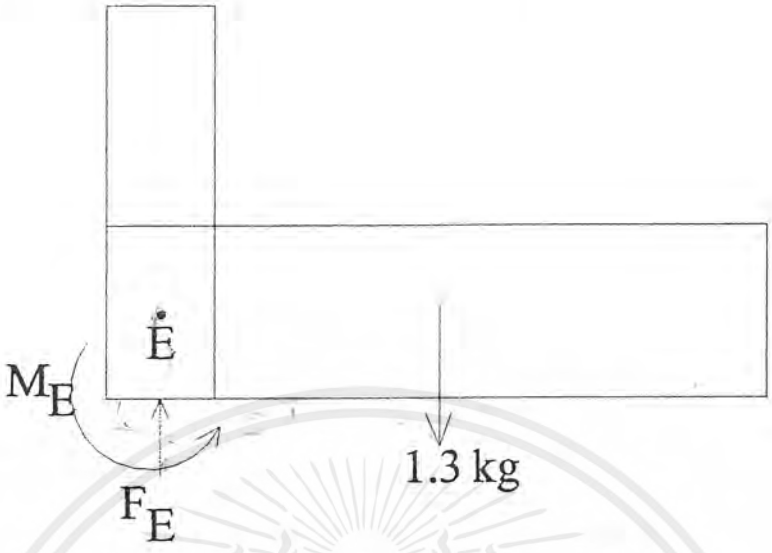
$$\sigma_{max} = F/A + Mc/I$$

โดย A คือ พื้นที่หน้าตัดของเพลากลวงที่วิเคราะห์ $= 4*10^{-4} \text{ m}^2$

$$= [(58.56)/(4*10^{-4})] + [(5.75*0.025)/(1.143*10^{-7})]$$

$$= 12.725 \text{ Mpa}$$

4.9.3 พิจารณาหาความเค้นที่เพลาเล็กตรงกลางของแขนกลและแรงกระทำที่แบริ่ง



รูปที่ 4.7 ประกอบการคำนวณหา F_E และ M_E

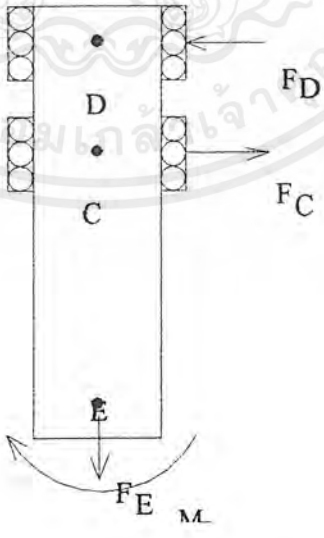
หาแรงดึงและโมเมนต์ดัดในเพลา

$\Sigma F_y = 0$, $F_E = 1.3 * 9.81$

$= 12.75$ N

$M_E = 12.75 * 0.1 = 1.275$ N*m

Take moment ที่จุด C เราจะได้แรงที่ แบริ่ง D



รูปที่ 4.8 รูปประกอบการคำนวณหาแรงที่แบริ่ง

$$F_D * 0.07 = 1.275$$

$$F_D = 18.24 \text{ N}$$

เราจะได้ความเค้นที่เพลาคือ

$$\sigma_{max} = F_D/A + M_D c/I$$

โดย A คือ พื้นที่หน้าตัดของเพลากลวงที่วิเคราะห์ $= 1.3*10^{-4}\ m^2$

$$= [(12.75)/(1.3*10^{-4})] + [(1.275 * 0.015)/(1.4 * 10^{-8})]$$

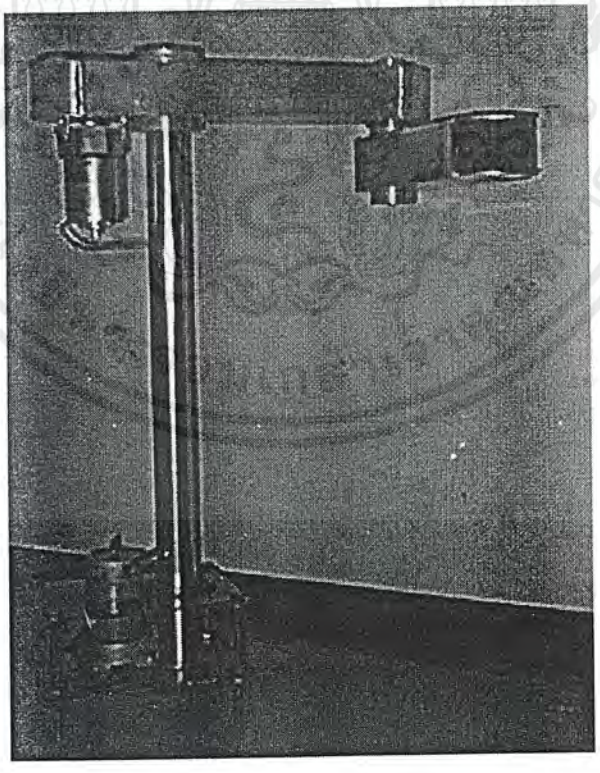
$$= 1.464 \text{ MPa}$$

บทที่ 5
การประกอบแขนกล

แขนกลที่ประกอบขึ้นเป็นแขนกลแบบสการา ซึ่งมีการเคลื่อนที่ 2 แกนอิสระใช้ DC มอเตอร์ 2 ตัว คือ มอเตอร์ขนาด 28 volts 18 amp 175 rpm และ มอเตอร์ขนาด 18 volts 24 amp 89 rpm ตัว ในการขับในแต่ละส่วนของแขนกล

วัสดุที่ใช้ในการประกอบแขนกลนี้ใช้อะลูมิเนียมแผ่นที่มีความหนา 3 มิลลิเมตร เพื่อความแข็งแรงของโครงสร้างของแขนกลเอง และขนาดในแต่ละส่วนได้ออกแบบตามความเหมาะสมของโครงสร้าง

การออกแบบแขนกลเน้นที่ความแข็งแรงของโครงสร้างโดยจะต้องทำให้เกิดการกระตุกหรือการสั่นในระหว่างที่มีการเคลื่อนที่ให้น้อยที่สุดซึ่งจะมีผลต่อความแม่นยำในการเคลื่อนที่ และเกิดความรวดเร็วในการเคลื่อนที่ ชิ้นส่วนแต่ละชิ้นจะต้องทนทานต่อการใช้งานและสามารถนำแขนกลนี้ไปปรับปรุงและพัฒนาต่อได้โดยการส่งกำลังระหว่างมอเตอร์กับข้อต่อส่วนต่างๆของแขนกลจะใช้สายพานและเฟืองเป็นตัวส่งผ่านกำลังไปยังส่วนต่างๆ ของข้อต่อแต่ละที่

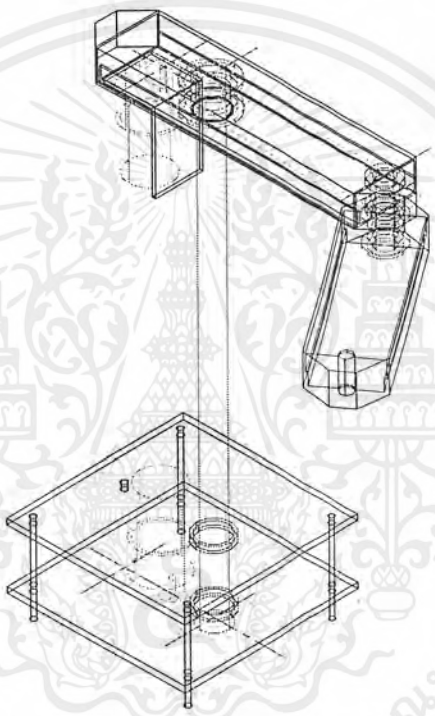


รูปที่ 5.1 แสดงโครงสร้างของแขนกลสการา

จากรูปที่ 5.1 แสดงโครงสร้างของแขนกลซึ่งประกอบด้วยส่วนต่าง ๆ ดังนี้

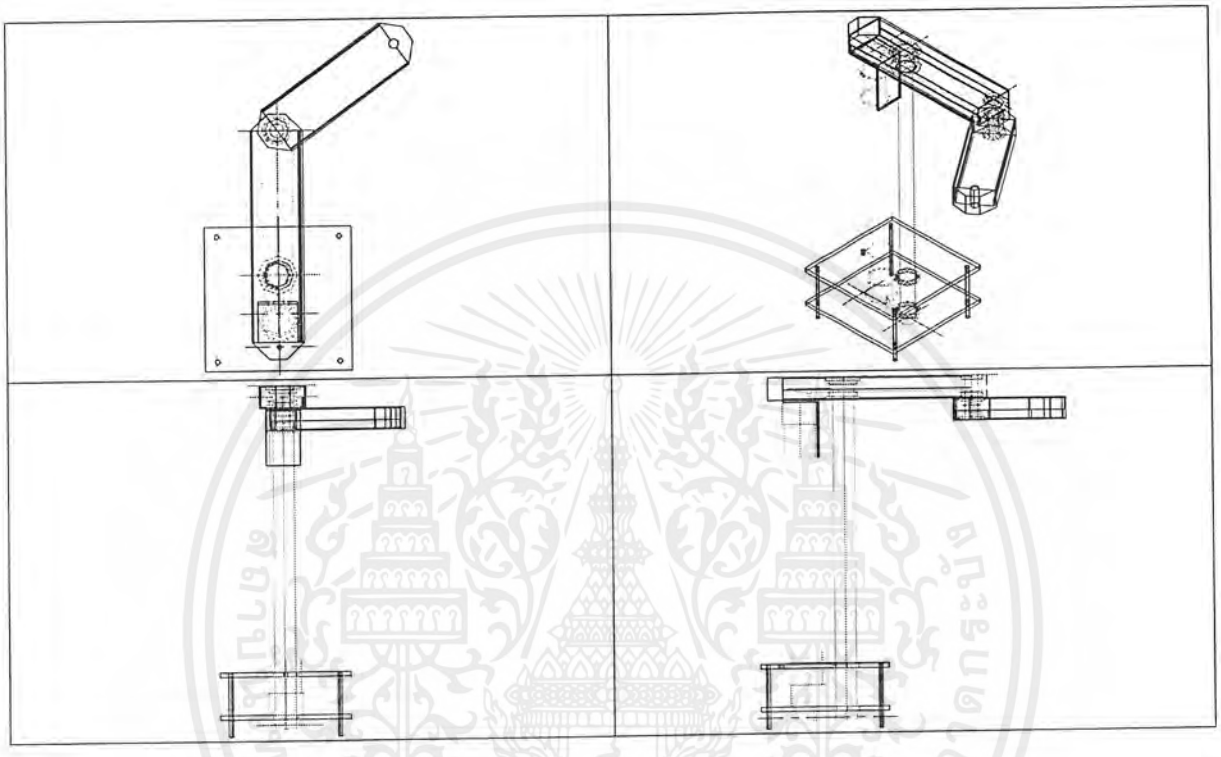
1. ส่วนฐาน (base part)
2. ส่วนแขนช่วงแรก (arm part-1)
3. ส่วนแขนช่วงที่สอง (arm part-2)

โดยทั้ง 3 ส่วนจะต้องทำงานสัมพันธ์กันเพื่อที่จะทำให้แขนกลทำงานได้อย่างมีประสิทธิภาพ



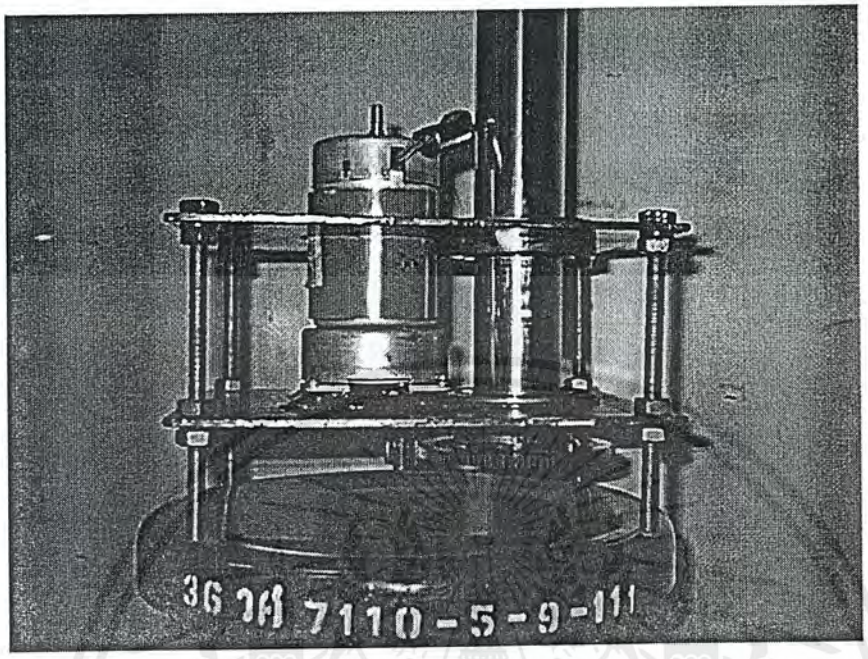
รูปที่ 5.2 ภาพ Drawing ของแขนกล

แสดงภาพในมุมมองต่างๆได้แก่ มุมด้านบน , มุมด้านหน้า , มุมด้านข้าง และภาพสามมิติ

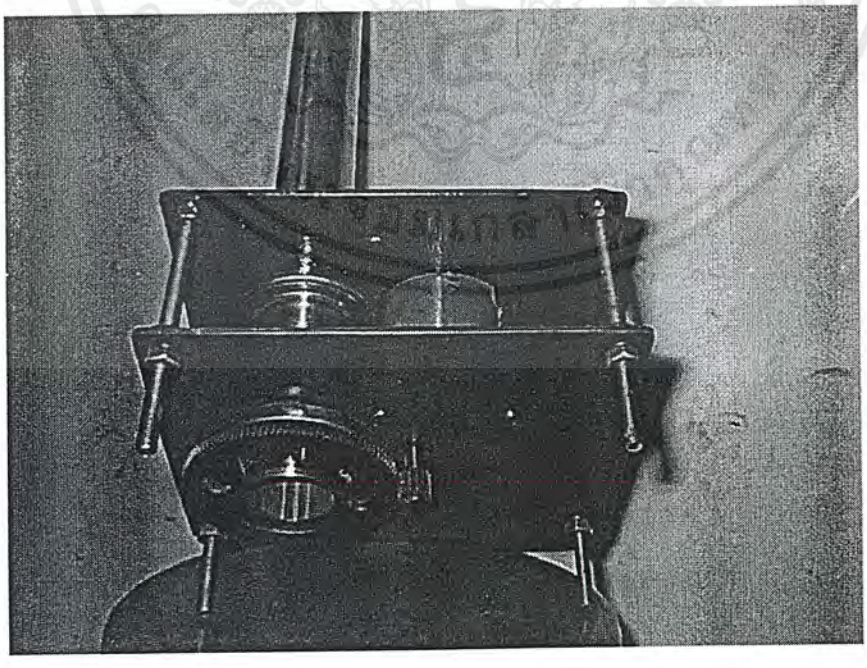


รูปที่ 5.3 ภาพฉายของแขนกล

5.1 ส่วนฐาน (base part)

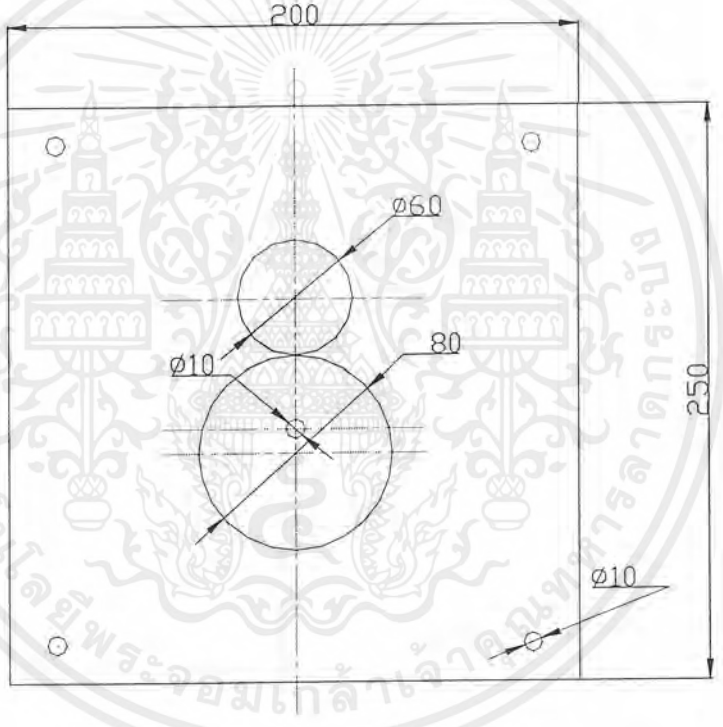


รูปที่ 5.4 ภาพแสดงโครงสร้างส่วนฐานของแขนกล

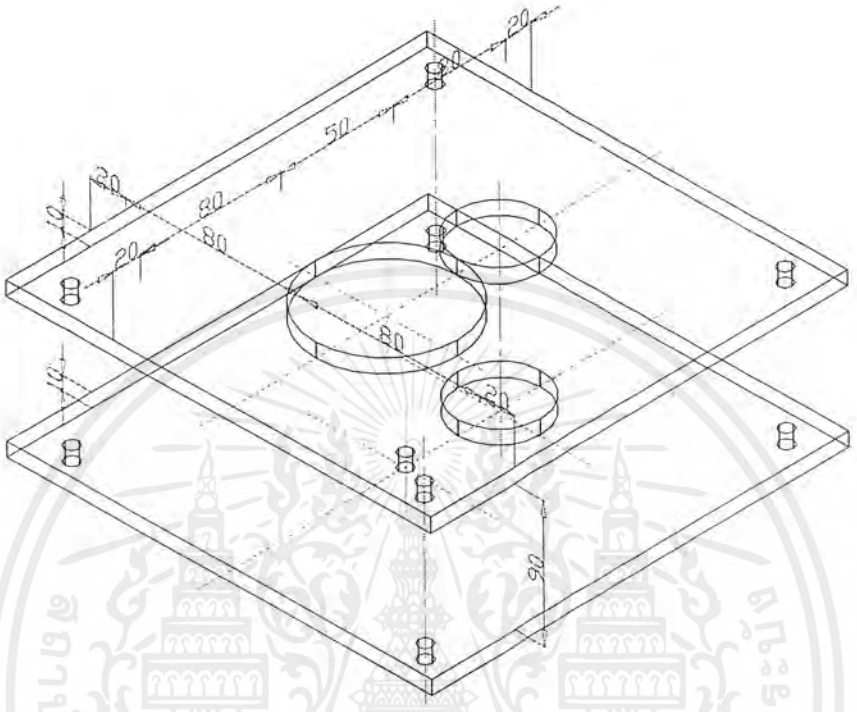


รูปที่ 5.5 ภาพแสดงการทดเฟืองในส่วนฐานของแขนกล

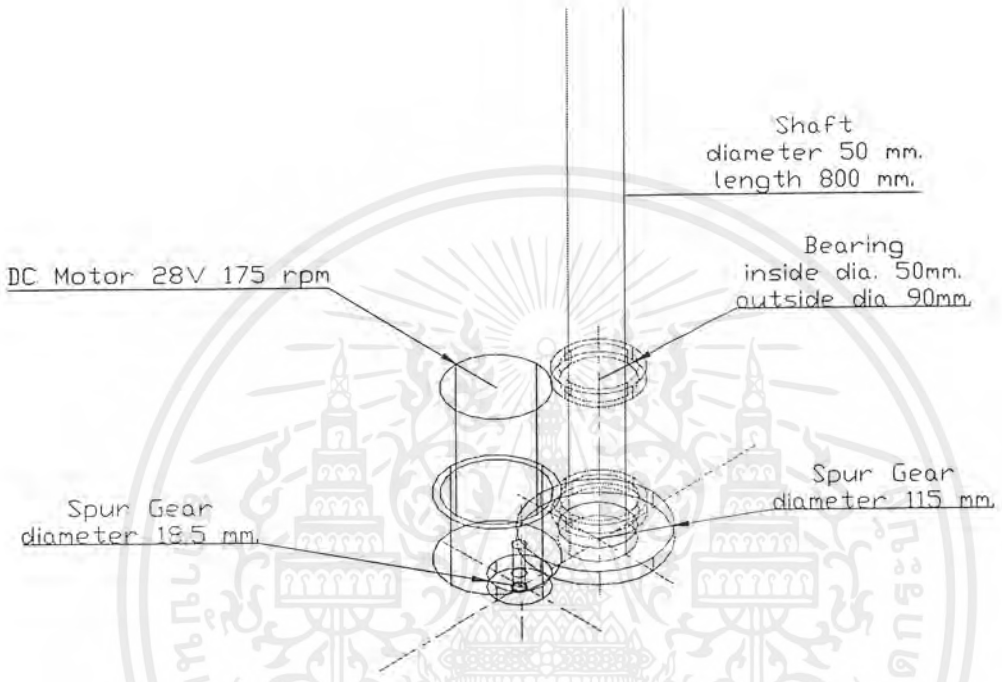
จากรูปที่ 5.4 และรูปที่ 5.5 จะประกอบด้วย DC มอเตอร์ 1 ตัว ซึ่งมีขนาด 28 โวลต์ 18 แอมป์ 175 RPM DC มอเตอร์ตัวนี้จะมีเฟืองขนาดเล็กติดอยู่เพื่อทำการทดกำลังไปยังเฟืองตัวใหญ่ โดยอัตราทดเฟืองที่ได้ทำการทดเฟืองเท่ากับ 1:6.6 โดยเฟืองเล็กมีจำนวนฟัน 17 ฟัน และเฟืองใหญ่มีจำนวนฟัน 112 ฟัน ทำให้เกิดค่าทอร์ก (torque) สูงขึ้นดังที่ได้ทำการคำนวณแล้วก่อนหน้านี้ การทดเฟืองนี้เพื่อทำให้เกิดแรงที่จะต้องในการหมุนแขนกลทั้งชุดโดยสามารถหมุนได้ 360 องศา โดยมีความเร็วเชิงมุมสูงสุดและความเร่งเชิงมุมสูงสุดตามที่ได้แสดงการคำนวณไว้แล้ว และส่วนด้านความแข็งแรงของโครงสร้างจะมีแบริ่งที่ใช้ในการยึดแกนเพลาหลักให้สามารถตั้งอยู่ได้และจะต้องสามารถรับน้ำหนักของแขนกลทั้งชุดได้ ขนาดของโครงสร้างได้แสดงไว้ดังรูปที่ 5.4



รูปที่ 5.6 แสดงขนาดของโครงสร้างส่วนฐานโดยเป็นภาพฉายด้านบน

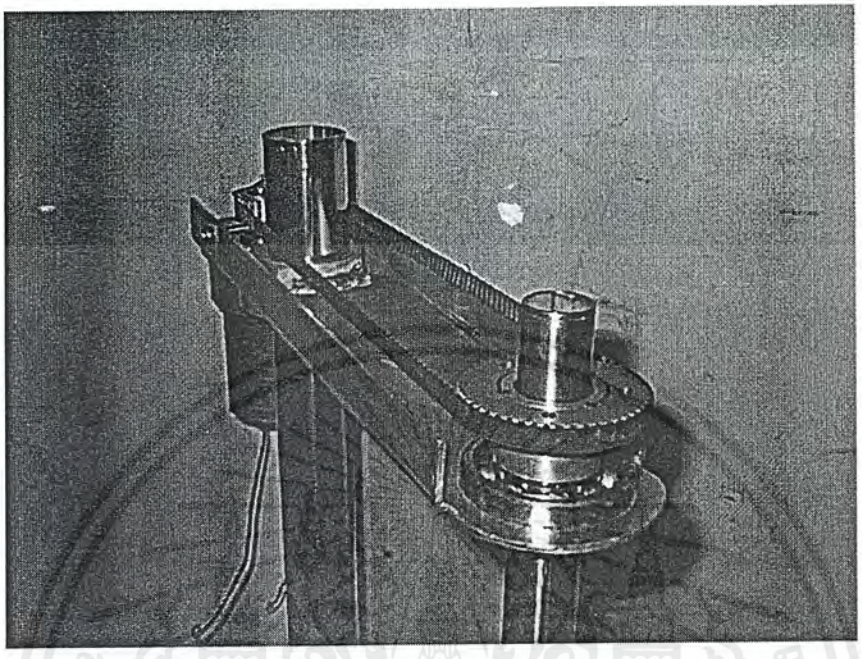


รูปที่ 5.7 แสดงขนาดของด้านต่างๆในมุมมองสามมิติของส่วนฐาน

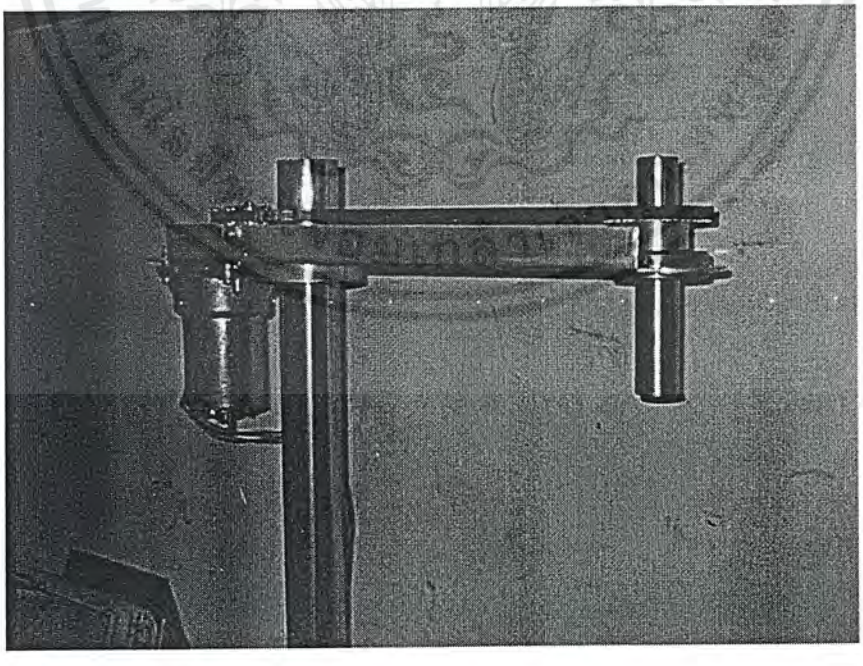


รูปที่ 5.8 แสดงส่วนประกอบของส่วนส่งกำลังในส่วนฐาน

5.2 ส่วนแขนช่วงแรก (arm part-1)

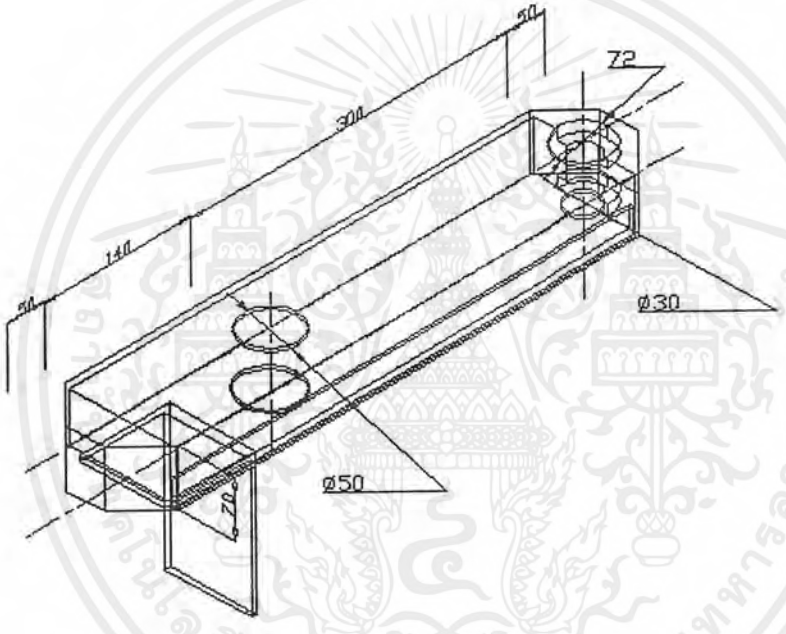


รูปที่ 5.9 ภาพแสดงโครงสร้างส่วนแขนช่วงแรกจากด้านหน้า

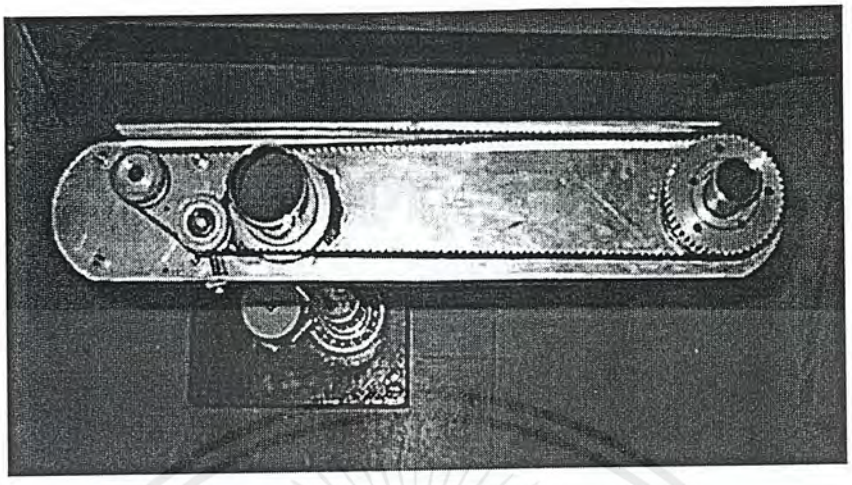


รูปที่ 5.10 ภาพแสดงโครงสร้างส่วนแขนช่วงแรกจากด้านข้าง

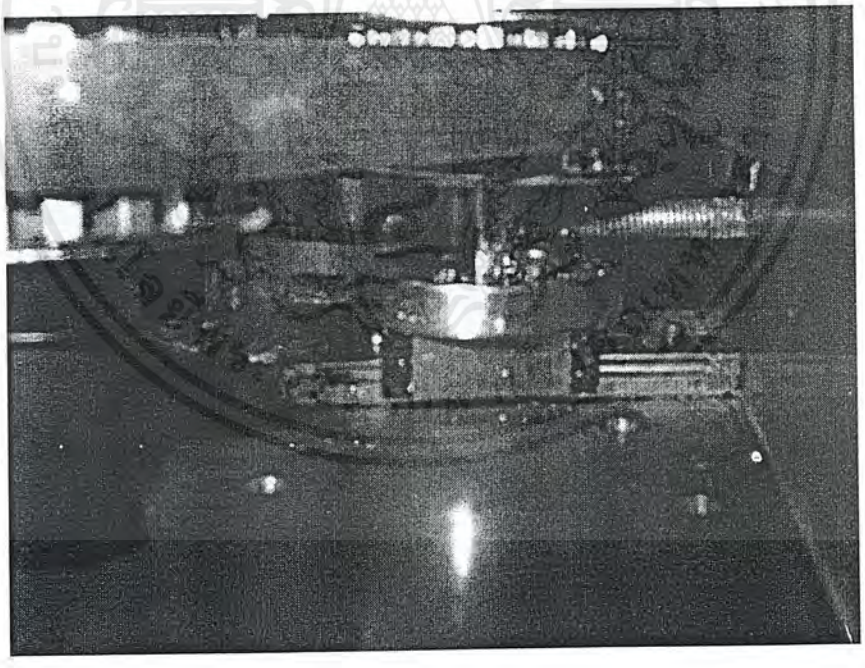
จากรูปที่ 5. 10ประกอบด้วย DC มอเตอร์ ขนาด 18 โวลต์ 26 แอมป์ 59 RPM ส่งผ่านกำรังด้วยสายพานแบบมีฟัน โดยจะมีตัวปรับความตึงของสายพานเพราะเมื่อสายพานถูกใช้ไปช่วงระยะเวลาพอสมควรสายพานก็จะเกิดการหย่อนจึงจำเป็นที่จะต้องมีตัวไว้คอยปรับความตึงให้เหมาะสม และมีการทดเฟืองมู่เล่ด้วยอัตราการทดเท่ากับ 1:2.4 โดยที่มู่เล่ตัวเล็กมีขนาด 20 ฟัน และมู่เล่ตัวใหญ่มีขนาด 48 ฟัน เพื่อทำหน้าที่ในการขับแขนท่อนที่สอง ในลักษณะหมุนเป็นวงกลมในแนวระดับ มีแบริ่งที่ใช้ในการยึดแกนเพลาท่อนที่สองให้สามารถอยู่ได้และจะต้องสามารถรับน้ำหนักของแขนกลท่อนที่สองได้ด้วย



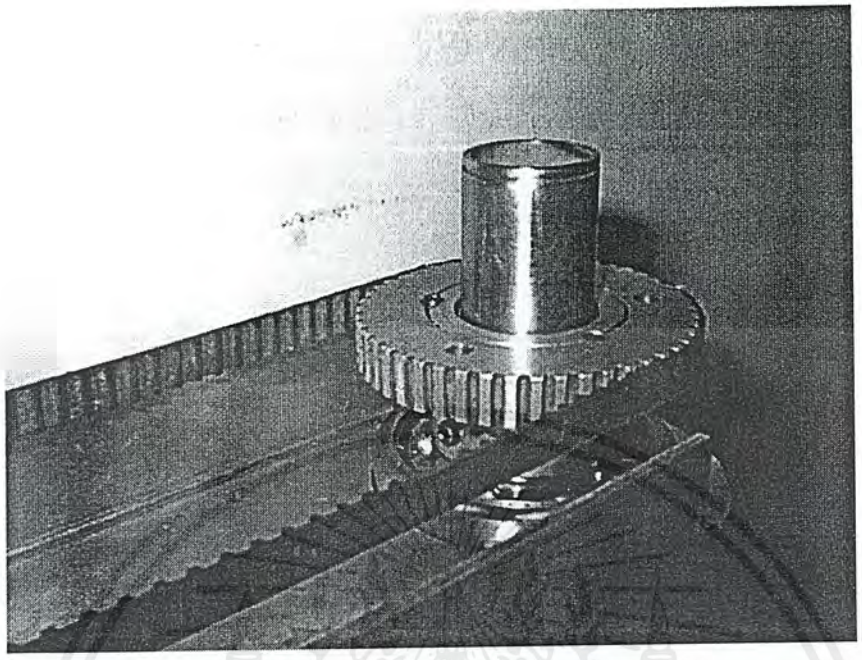
รูปที่ 5. 11 ภาพแสดงขนาดของด้านต่างๆในมุมมองสามมิติ



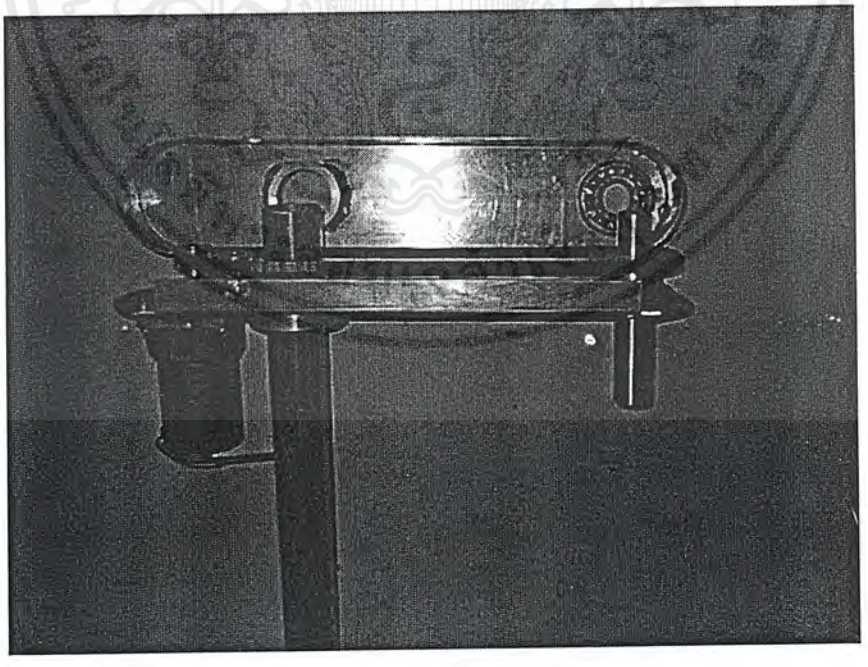
รูปที่ 5.12 ภาพแสดงการส่งกำลังด้วยสายพาน



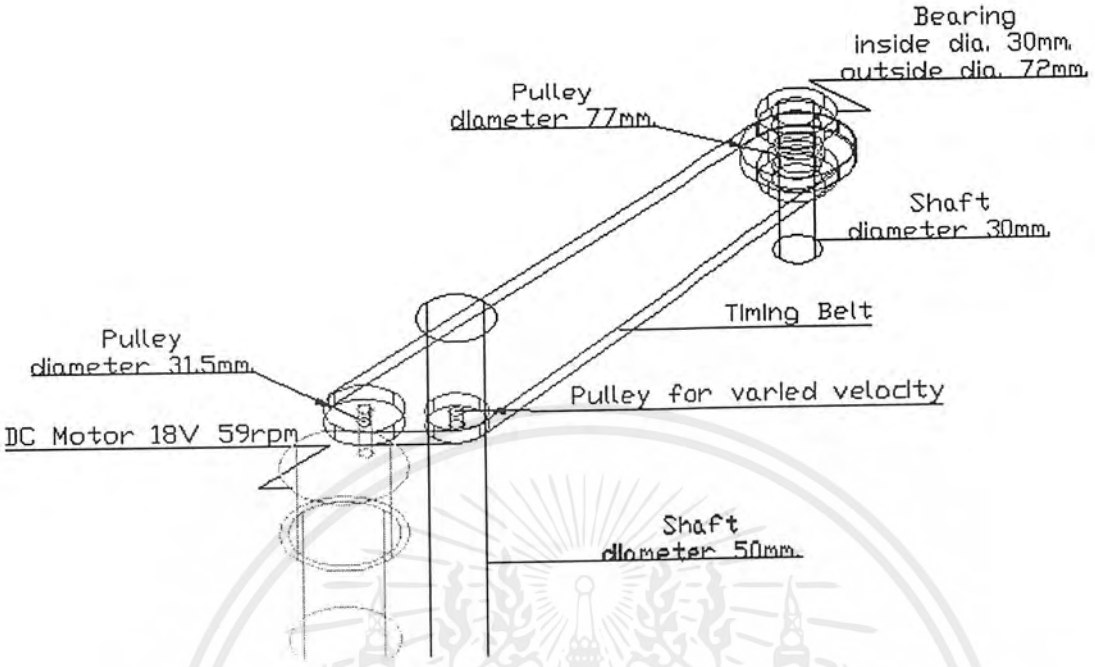
รูปที่ 5.13 ภาพแสดงชุดปรับระดับความตึงของสายพาน



รูปที่ 5.14 ภาพแสดงแกนเพลาที่สองที่ใช้ส่งกำลังต่อให้กับแขนกลท่อนที่สองที่อยู่ด้านล่าง

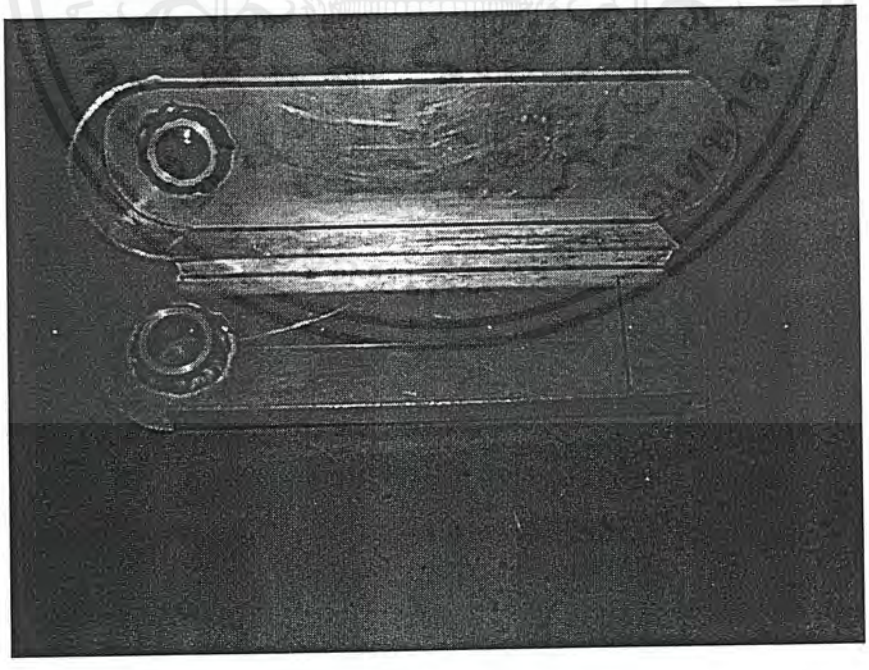


รูปที่ 5.15 ภาพแสดงแขนท่อนแรกโดยมีฝาปิดที่มีแบริ่งเป็นตัวช่วยผยุงแกนเพลาที่สอง

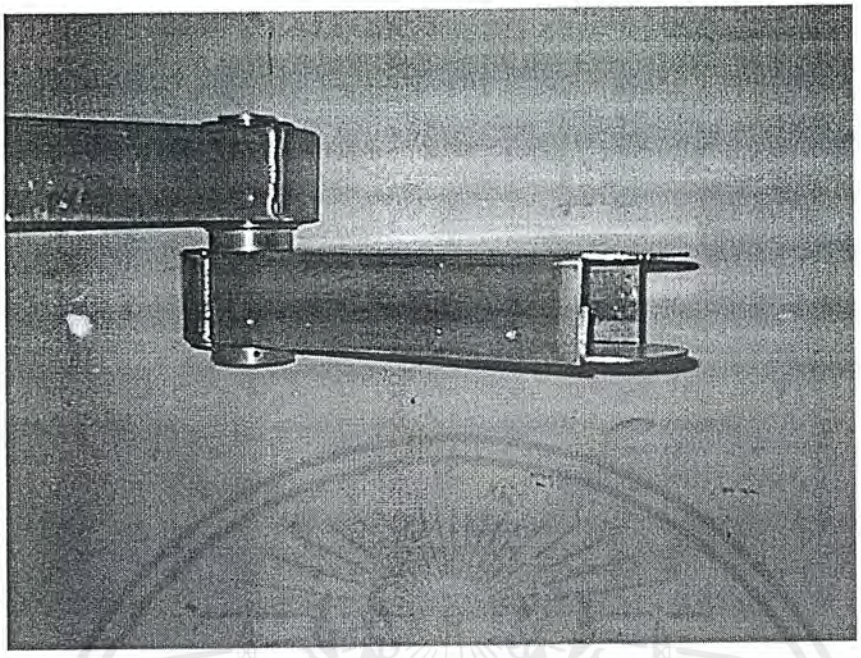


รูปที่ 5.16 แสดงส่วนประกอบของส่วนส่งกำลังในส่วนส่งกำให้แขนท่อนที่สอง

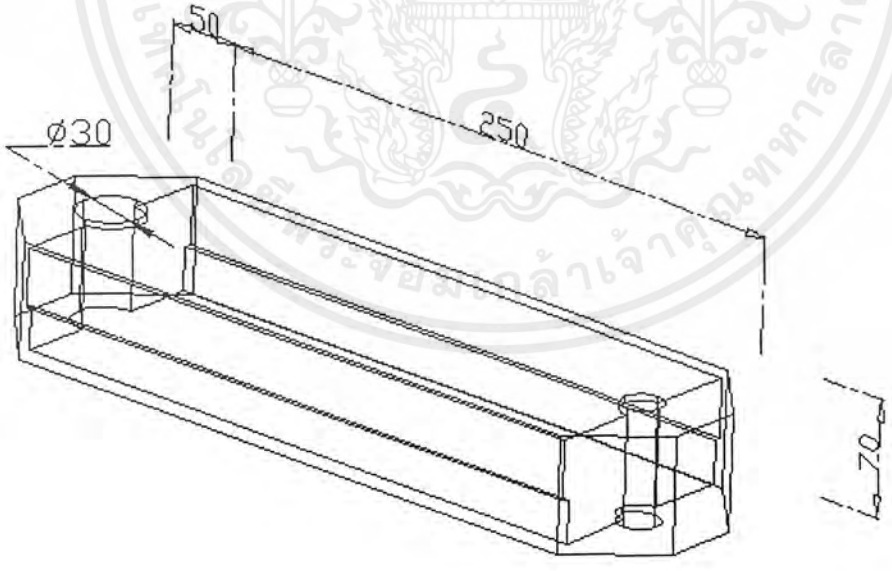
5.3 ส่วนแขนช่วงที่สอง (arm part-2)



รูปที่ 5.17 ภาพแสดงโครงสร้างของแขนกลท่อนที่สอง



รูปที่ 5.18 ภาพแสดงแขนกลท่อนที่สองเมื่อประกอบแล้ว



รูปที่ 5.19 ภาพแสดงขนาดของด้านต่างๆในมุมมองสามมิติ

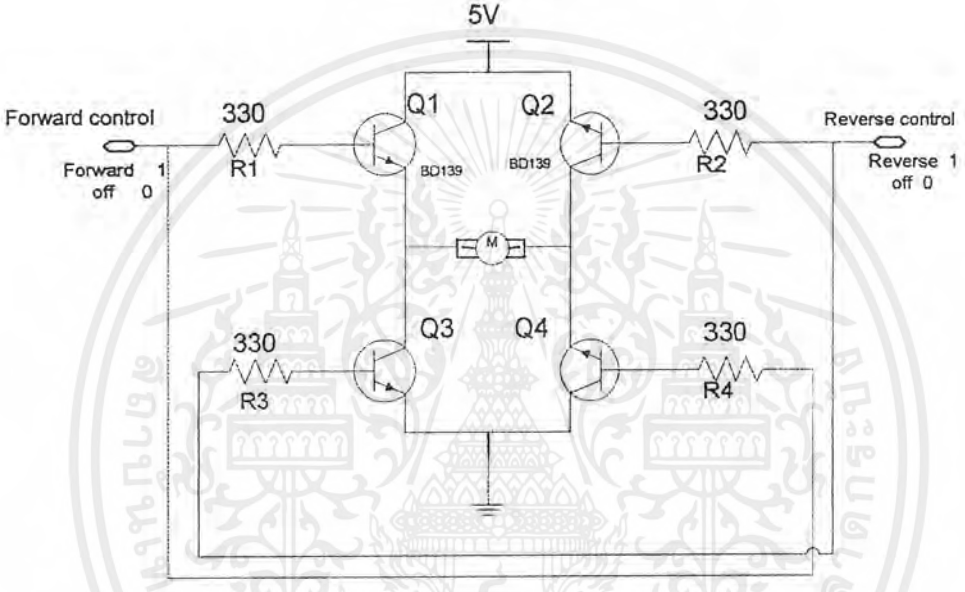
จากรูปที่ 5.19 เป็นรูปโครงสร้างของแขนท่อนที่สองซึ่งสามารถที่จะนำไปพัฒนาเพื่อจุดประสงค์ต่างๆที่ต้องการ

บทที่ 6
วงจรและอุปกรณ์ควบคุมมอเตอร์

6.1 วงจรขับเคลื่อนมอเตอร์กระแสตรง

เป็นวงจรที่ใช้ในการควบคุมมอเตอร์ให้มีทิศทางการหมุนได้ตามต้องการหมุนทวนเข็มหรือตามเข็มนาฬิกาโดยสามารถควบคุมได้ด้วยไมโครโปรเซสเซอร์ มีหลายวงจรดังนี้

6.1.1 ทรานซิสเตอร์ในการควบคุมมอเตอร์

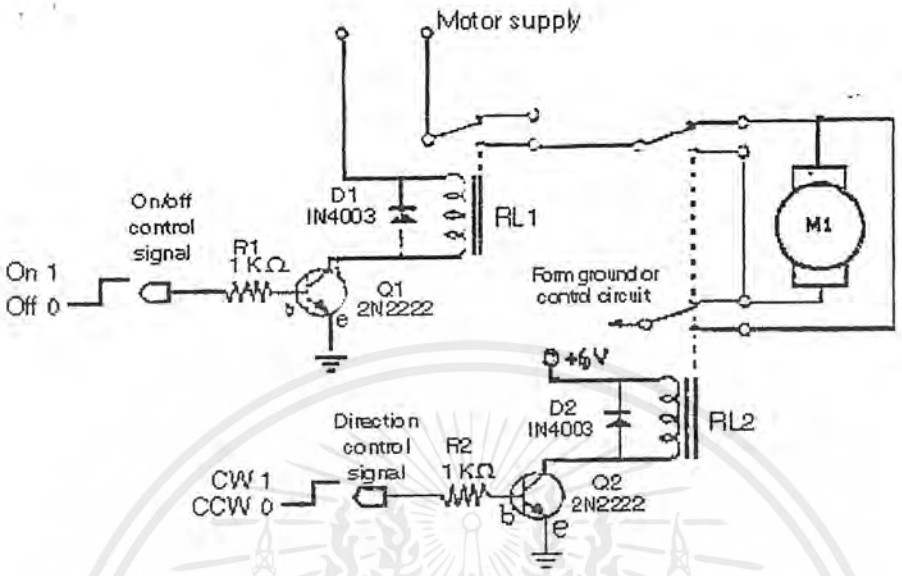


รูปที่ 6.1 วงจรขับมอเตอร์โดยใช้ทรานซิสเตอร์

การใช้วงจรในการควบคุมมอเตอร์นั้นเป็นการนำทรานซิสเตอร์ใช้งานในลักษณะของสวิทช์ โดยการทำการต่อดังรูปที่ 6.1 โดยจะเห็นว่า ในขณะใดขณะหนึ่งที่มอเตอร์หมุน เราจะให้ทรานซิสเตอร์ทำงานเพียง 2 ตัว นั่นคือ เมื่อมีการป้อนสัญญาณ logic "1" เข้าที่ I/P_1 และที่ I/P_2 เป็น logic "0" จะทำให้ได้ทรานซิสเตอร์ Q1 และ Q4 ทำงาน ทำให้มอเตอร์หมุนในทิศทางหนึ่ง แต่เมื่อเราป้อนสัญญาณที่ I/P_1 เป็น logic "0" และที่ I/P_2 เป็น logic "1" จะทำให้ทรานซิสเตอร์ Q2 และ Q3 ทำงานทำให้มอเตอร์หมุนในอีกทิศทางหนึ่ง

วงจรลักษณะนี้เราสามารถที่จะใช้แหล่งจ่ายไฟขั้วเดียวและสามารถแยกเป็นแหล่งจ่ายไฟจากวงจรอิเลคทรอนิกส์ได้ ทำให้เราสามารถใช้เอาท์พุทจากวงจรดิจิตอลมาใช้ใน การเปิด-ปิดทรานซิสเตอร์ได้โดยตรง

6.1.2 การใช้รีเลย์ควบคุมมอเตอร์



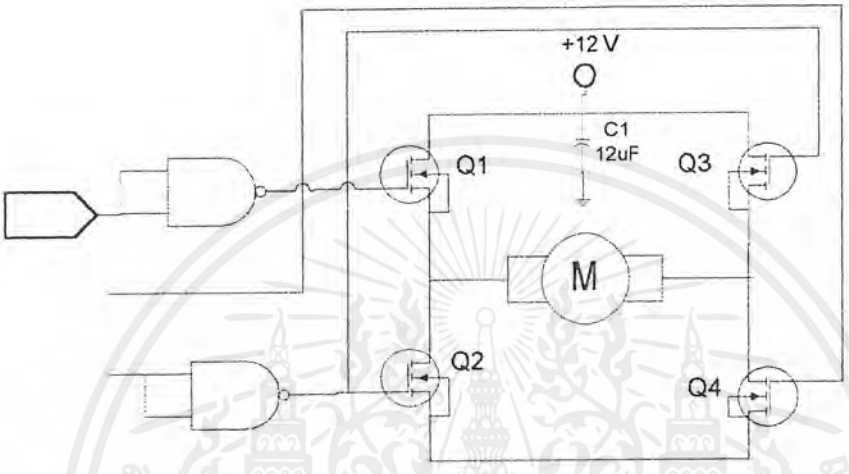
รูปที่ 6.2 วงจรขับมอเตอร์โดยใช้รีเลย์

จากรูปที่ 6.2 เป็นวงจรที่ใช้รีเลย์เป็นสวิทช์ควบคุม โดยรีเลย์RL1 เป็นแบบที่เรียกว่าซิงเกิลโพล-ดับเบิ้ลโธรล (Single-Pole Double-Throw : SPDT) คือ เป็นรีเลย์ที่มีขั้วเดียว ซึ่งจะเป็นรีเลย์ที่ใช้ในการควบคุมการเปิด-ปิด ของมอเตอร์โดยการป้อนสัญญาณควบคุมเข้าที่ขาเบสของทรานซิสเตอร์ Q1 นั่นคือ เมื่อป้อน logic "1" (5V)จะทำให้ทรานซิสเตอร์ Q1 ทำงานทำให้มีกระแสไหลผ่านขดลวดของรีเลย์ทำให้รีเลย์ทำงาน ทำให้มอเตอร์ได้รับการไบอัส มอเตอร์จึงเริ่มหมุน แต่เมื่อป้อน logic "0" (0V) จะทำให้มอเตอร์ไม่ทำงาน

ส่วนการควบคุมทิศทางการหมุนของมอเตอร์สามารถทำได้โดย การป้อนสัญญาณควบคุมเข้าที่ขาเบสของทรานซิสเตอร์ Q2 ที่ เป็นรีเลย์แบบ ดับเบิ้ล-โพล ดับเบิ้ลโธรล ที่มีสองขั้ว นั่นคือ การป้อน logic "1" หรือ logic "0" เพื่อให้ทรานซิสเตอร์ Q2 ทำงานหรือไม่ทำงาน ตามลำดับ ซึ่งส่งผลให้มอเตอร์ได้รับการไบอัส ที่กลับทิศทางกันได้ อีกทั้งวงจรนี้สามารถที่จะใช้เอาท์พุทจากวงจรดิจิตอลได้โดยตรงและยังสามารถแยกไฟเลี้ยงของวงจรอิเลคทรอนิกส์และไฟเลี้ยงที่ป้อนให้กับมอเตอร์ออกจากกันได้อีกด้วย

6.1.3 การใช้เพาเวอร์มอสเฟสในการควบคุมมอเตอร์

วงจรที่ใช้งานทั่วไปแสดงได้ดังรูปที่ 6.3 ซึ่งการต่อมอสเฟสจะมีการต่อเหมือนกับวงจรที่ใช้ทรานซิสเตอร์ แต่ได้มีการเพิ่มอินเวอร์ทเตอร์ซึ่งทำให้สามารถลดอินพุทที่ต้องใช้ในการควบคุมทิศทางการหมุนของมอเตอร์ได้ แต่เราต้องเพิ่มเติมมอสเฟตที่จะใช้ในการควบคุมให้มอเตอร์หมุนหรือหยุดหมุนซึ่งทำให้เราสามารถที่จะควบคุมความเร็วมอเตอร์ได้



รูปที่ 6.3 วงจรขับมอเตอร์โดยใช้เพาเวอร์มอสเฟต

6.2 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ที่ใช้ในโครงงานนี้คือ LM629 ซึ่งมีลักษณะดังต่อไปนี้

รูปที่ 6.4 ไมโครคอนโทรลเลอร์ LM 629

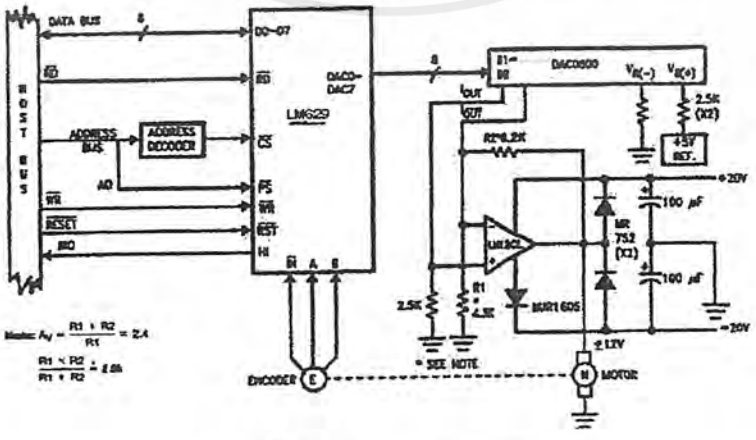
6.2.1 ลักษณะการทำงานของไมโครคอนโทรลเลอร์ LM629

LM629 เป็นไมโครคอนโทรลเลอร์ที่ใช้ควบคุมการเคลื่อนที่ของมอเตอร์แบบความเที่ยงตรงสูง ซึ่งสามารถใช้ได้กับมอเตอร์กระแสตรงหลายๆแบบ และยังสามารถใช้ได้กับเซอร์โวมอเตอร์ชนิดแปลงถ่าน ซึ่งจะมีสัญญาณป้อนกลับมาเพื่อบอกตำแหน่งของมอเตอร์ขณะนั้นได้อีกด้วย การทำงานของ LM629สามารถทำการสั่งงานและให้ผลตอบกลับได้ทันที (Real Time) ซึ่งในการทำงานลักษณะนี้ต้องการการควบคุมที่มีความเร็วสูง และในส่วนของโปรแกรม (Software) ที่ใช้ควบคุมถูกทำให้ง่ายยิ่งขึ้นด้วยชุดของคำสั่งระดับสูง

6.2.2 คุณสมบัติของไมโครคอนโทรลเลอร์ LM629

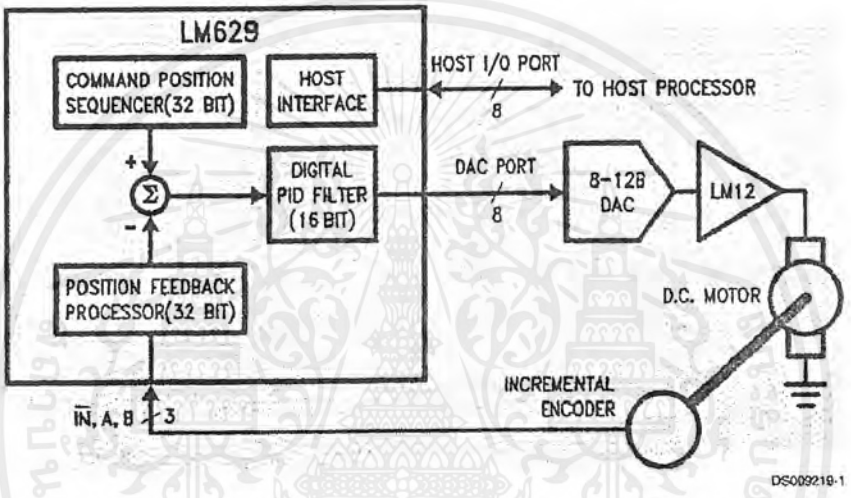
- เป็นแบบ 32 บิต เป็นรีจิสเตอร์ควบคุมความเร็วและความเร่งได้อย่างแม่นยำ
- สามารถโปรแกรม PID filter ด้วยตัวเลข 16 บิต
- สามารถโปรแกรมด้วยการสุ่มค่าเข้ามาได้
- ข้อมูลที่ออกมาสามารถเป็น 8 หรือ 12 บิต DAC
- เป็นสัญญาณขนาด 8 บิตของ PWM output
- สามารถควบคุมให้พื้นที่ระหว่างความเร็วและเวลาให้เป็นรูปสี่เหลี่ยมคางหมูได้
- ความเร็ว ,ตำแหน่งเป้าหมาย, และตัวแปรฟิลเตอร์(filter)อาจจะถูกเปลี่ยนระหว่างการเคลื่อนที่
- มีระบบของการคำนวณตำแหน่งและความเร็ว
- สามารถโปรแกรมให้มีการ interrupts ได้ทันที
- ส่วนติดต่อกับไมโครคอมพิวเตอร์เป็น asynchronous แบบ ขนาน

6.3 วงจรควบคุมมอเตอร์ที่ใช้ในโครงงาน



รูปที่ 6.4 วงจรควบคุมมอเตอร์

วงจรควบคุมมอเตอร์ที่ใช้แสดงได้ดังรูปที่ 6.4 ซึ่งมีส่วนประกอบและการทำงานดังนี้ คือ เมื่อป้อนค่าตำแหน่งที่จะเคลื่อนที่เข้าที่ตัวโปรแกรมจากไมโครคอมพิวเตอร์ ไมโครคอมพิวเตอร์จะส่งข้อมูลไปที่ไมโครคอนโทรลเลอร์ LM629 เพื่อทำการแปลงข้อมูลเป็นสัญญาณส่งไปที่วงจรขับมอเตอร์ให้ส่งกระแสให้กับตัวมอเตอร์ในปริมาณที่จะทำให้มอเตอร์มีความเร็วในการหมุนตามที่กำหนดในโปรแกรมและเมื่อมอเตอร์หมุนก็จะทำให้เอนโคดเดอร์หมุนไปด้วย และเอนโคดเดอร์ก็จะส่งสัญญาณไปที่ไมโครคอนโทรลเลอร์เพื่อทำการคำนวณและป้อนสัญญาณให้กับวงจรขับมอเตอร์ต่อไปจนกระทั่งตัวแขนกลเคลื่อนที่ไปถึงจุดเป้าหมายจึงเสร็จสิ้นการทำงาน



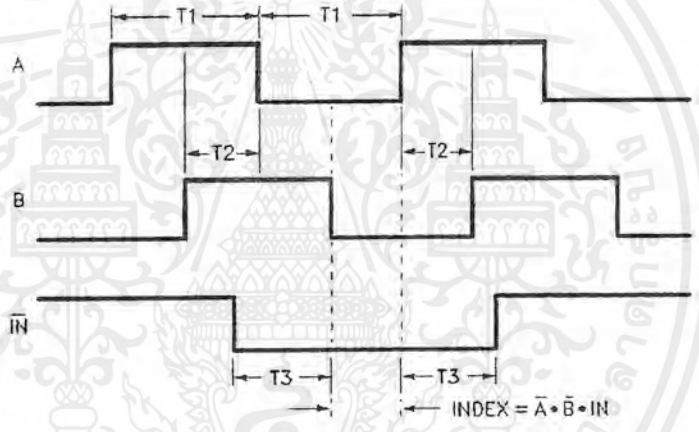
รูปที่ 6.5 บล็อกไดอะแกรมของระบบทั่วไปของ LM629

รูปที่ 6.5 แสดงบล็อกไดอะแกรมของระบบการทำงานของ LM629 ซึ่งสามารถอธิบายได้ดังนี้

การสร้างระบบควบคุมเซอร์โวมอเตอร์ โดยใช้ LM629 จะมีการติดต่อระหว่างตัวประมวลผลกับ LM629 โดยผ่านทางอินพุท/เอาท์พุทพอร์ท (I/O Port) เพื่อความสะดวกต่อการกำหนดความสัมพันธ์ระหว่างความเร็วกับเวลาในรูปแบบของสี่เหลี่ยมคางหมู (Trapezoidal velocity profile) และตัวกรองที่ชดเชยค่าแบบดิจิตอล ตัวสร้างความสัมพันธ์ระหว่างความเร็วกับเวลาในรูปแบบของสี่เหลี่ยมคางหมู (Trapezoidal velocity profile generator) จะทำการคำนวณวิถีทางที่ต้องการใช้สำหรับแต่ละตำแหน่ง หรือโหมดความเร็วที่ได้กำหนดไว้ โดย LM629 จะทำการหักลบค่าของตำแหน่งจริง (ที่ได้มาจาก สัญญาณป้อนกลับจาก Encoder) ออกจากตำแหน่งที่ได้ออกแบบไว้ และผลที่ได้ซึ่งเป็นความแตกต่างของตำแหน่ง (position error) นี้จะถูกนำไปประมวลผลโดยตัวกรองดิจิตอล เพื่อที่จะนำไปควบคุมมอเตอร์ให้ได้ตำแหน่งตามต้องการ

หลังจากที่ได้ค่าเอาท์พุทเป็นดิจิตอล 8 บิต ค่าเอาท์พุทที่ได้นี้จะถูกส่งต่อไปยังส่วนของวงจรแปลงสัญญาณดิจิตอลเป็นอนาลอก (Digital to Analog Converter : DAC) ซึ่งในวงจรนี้จะใช้ไอซี DAC เบอร์ DAC0800 และเมื่อทำการแปลงค่าสัญญาณดิจิตอลที่เข้ามาเป็นค่าสัญญาณอนาลอกแล้ว ค่าเอาท์พุทที่ได้จากส่วนนี้จะถูกส่งต่อไปยังวงจรขยายกำลัง (Power Amplifier) โดยใช้ออปแอมป์เบอร์ LM12CL เป็นตัวขยายกำลังจากนั้นจึงนำค่าเอาท์พุทไปทำการควบคุมมอเตอร์ต่อไป

LM629 จะสามารถรู้ตำแหน่งในขณะใดๆ ของมอเตอร์ได้จาก Encoder ซึ่งจะส่งสัญญาณป้อนกลับมายัง LM629 , 3 สัญญาณ ได้แก่ สัญญาณที่มีลักษณะเป็นพัลส์สี่เหลี่ยม A , B ซึ่งสามารถใช้บอกทิศทางในการเคลื่อนที่ของมอเตอร์ในขณะนั้นได้ ซึ่งในแต่ละเส้นของสัญญาณจาก Encoder จะถูกแบ่งเป็น 4 states แต่ละ state จะถูกกำหนดตามสัญญาณ A และ B ตามตารางในรูปที่ 6.3 ส่วนอีกสัญญาณหนึ่ง เป็นสัญญาณบ่งชี้ (Index Pulse) ซึ่งสัญญาณนี้จะเป็นลอจิกต่ำเมื่อมอเตอร์มีการหมุนกลับทิศจากทิศทางเดิม แสดงได้ดังรูปที่ 6.6



รูปที่ 6.6 สัญญาณป้อนกลับจาก Encoder

บทที่ 7
สรุป

7.1 ปัญหาสำหรับการทำโครงงานชิ้นนี้

1. ไม่สามารถพับแผ่นอะลูมิเนียมที่มีความหนา 3 มิลลิเมตรให้ได้รูปแบบและขนาดที่ต้องการได้โดยใช้เครื่องพับที่มีใช้อยู่ในคณะได้
2. แกนเพลาหลักที่เป็นท่อสแตนเลสซึ่งมีเส้นผ่านศูนย์กลางใหญ่กว่าที่เครื่องกลึงที่มีใช้อยู่ในคณะไม่สามารถงานได้อย่างสะดวกซึ่งจะต้องชิ้นส่วนเพิ่มเติมเพื่อเป็นตัวกำหนดศูนย์
3. มอเตอร์ที่ต้องการไม่สามารถซื้อมอเตอร์ใหม่ได้เพราะงบประมาณไม่เพียงพอ
4. เนื่องจาก เอนโคดเดอร์ หาซื้อได้ยากและมีราคาแพง ทำให้กว่าจะได้ตัวอุปกรณ์มาล่าช้า
5. เนื่องจากมอเตอร์เป็นของเก่าจึงอาจได้อุปกรณ์ที่มีคุณลักษณะไม่เท่ากับที่ระบบไว้ที่ตัว มอเตอร์จึงต้องเสียเวลาในการทดสอบเพื่อหาคุณสมบัติของตัวมอเตอร์สียก่อน
6. ปัญหาที่อาจทำให้แขนกลเอียงหรือทำให้ไม่ได้ฉาก
   - 6.1 อาจจะมีการเยื้องศูนย์เล็กน้อยของแบริ่งที่รองรับตรงด้านล่าง
   - 6.2 ความร้อนเนื่องจากการเชื่อมระหว่างแผ่นอะลูมิเนียมที่ใช้ทำโครงของแขนกลกับคัปปลิ้ง อาจมีการบิดเบี้ยว ทำให้คัปปลิ้ง อาจจะเอียงไปด้านใดด้านหนึ่ง
   - 6.3 หากการขันนอต ยึดระหว่างแกนหลักของแขนกลกับตัวโครงแขนกล หลอมเกินไปหรือแน่นเกินไป อาจจะทำให้เอียงได้
   - 6.4 แบริ่งตรงที่อยู่ที่จุดต่อระหว่างแขนช่วงต้นและแขนช่วงปลาย อาจมีการขยับเล็กน้อย เนื่องจากเจาะรูที่แผ่นอะลูมิเนียมหลวมเกินไป
   - 6.5 การประกอบแบริ่ง เข้ากับโครงของแขนกล เนื่องจากในบางครั้งในการประกอบต้องใช้แรงในการยัดแบริ่งเข้าไปในรูที่เจาะไว้ ทำให้แผ่นอะลูมิเนียมอาจจะงอได้
   - 6.6 การกลึงเพลาแกนของตัวแขนกลทั้ง เพลาแกนหลักและเพลาตรงจุดต่อระหว่างแขนช่วงต้นและแขนช่วงปลาย หากกลึงแล้วหลอมเกินไปหรือไม่ได้ศูนย์ก็อาจจะทำให้จุดต่อไม่แข็งแรง มีการสั่นของแขนกลขณะหมุนได้
   - 6.7 โครงอะลูมิเนียมอาจมีการงอ เนื่องจากการรับน้ำหนักหากมีอายุการใช้งานนานๆ

7.2 แนวทางแก้ไข

1. จะต้องนำแผ่นอะลูมิเนียมที่ตัดได้ขนาดไปจ้างโรงงานที่รับงานพับโลหะที่มีเครื่องจักรที่ที่ทันสมัย
2. ต้องนำท่อสแตนเลสที่ต้องการจะปลอกผิวนอกไปจ้างโรงกลึงที่มีเครื่องจักรที่จะสามารถรับง นได้
3. จะต้องหาซื้อมอเตอร์เก่าตามขนาดที่ต้องการตามร้านขายชิ้นส่วนเครื่องจักรเก่า
4. จะต้องมีงบประมาณสำหรับอุปกรณ์เอนโคดเดอร์ที่เพียงพอหรือค้นหาจากอุปกรณ์ที่เคยสร้างมาและมี เอนโคดเดอร์ เป็นส่วนประกอบ
5. ซื้อมอเตอร์ใหม่ หากมีงบประมาณเพียงพอ เพราะ จะได้ไม่เสียเวลาในการทดสอบมากนัก
6. การแก้ไขปัญหาตัวแขนกลให้ได้ฉาก หรือไม่ให้มีการเอียง
   6.1 ประกอบแบริ่ง ให้มีการเยื้องศูนย์ให้น้อยที่สุด
   6.2 ควรยึดคัปปลิ้งให้แน่นที่สุดก่อนที่จะเชื่อม หรือเชื่อมไปทีละน้อยเพื่อไม่ให้แผ่นอะลูมิเนียมต้องรับความร้อนมากจนบิดงอ หรือเปลี่ยนจากการเชื่อมมาเป็นการใช้นอตยึดคัปปลิ้งกับแผ่นอะลูมิเนียมแทนการเชื่อม
   6.3 ขันนอตให้พอดี และตรวจสอบโดยการใช้ไม้ฉากให้ตัวเพลาแกนหลักกับแขนกลฉากกัน
   6.4 ควรเจาะรูด้วยเครื่อง CNC เพื่อความเที่ยงตรงและ และขนาดพอดี ไม่ขับหรือหลวมจนเกินไป
   6.5 เจาะรูให้พอดีโดยไม่ให้ยากแก่การประกอบ และเราก็ไม่ต้องออกแรงมากเกินไปซึ่งจะทำให้โครงแขนกลเสียหายได้
   6.6 การกลึงเพลา ควรกลึงให้พอดี เพื่อไม่ให้มีช่องว่างระหว่างเพล เกับแบริ่งมากเกินไป
   6.7 อาจจะเปลี่ยนจากโครงแขนกลที่ใช้แผ่นอะลูมิเนียมเป็นแผ่นสแตนเลส เพราะ มีความแข็งแรงกว่า

7.3 ผลการทดลอง

1. การกำหนดทิศทางการหมุนและระยะทาง

จากการป้อนค่าลงในโปรแกรม ตัวแขนกลสามารถเคลื่อนที่ได้ตามระยะทางที่กำหนดซึ่งก็เป็นไปตามเป้าหมายของการทำโปรเจทนี้

2. การทดลองการเคลื่อนที่ของส่วนของแขนกล

ผลปรากฏว่า สามารถหมุนไปด้วยดีไม่มีอาการสั่นของแขน และไปยังจุดเป้าหมายได้ตามที่ต้องการ อัตราความเร็วในการหมุนต่างๆขึ้นอยู่กับการตั้งค่าในโปรแกรมซึ่งจากการการทดลองจึงได้ความเร็วและความเร่งที่เหมาะสมที่ไม่ทำให้แขนกลไม่มีอาการสั่น หรือมีการกระชากอย่างรุนแรง

3. การเคลื่อนที่ในลักษณะต่างๆ

จากการทดลองขับเคลื่อนแขนกล เราจะสามารถขับแขนกลได้ในหลายลักษณะโดยที่ขึ้นอยู่กับโปรแกรมที่เขียนไว้ ซึ่งในโครงงานนี้มีลักษณะการเคลื่อนที่ต่างๆดังต่อไปนี้

3.1 การเคลื่อนที่เพื่อปรับค่าจุดอ้างอิง (jog) โดยจะรับค่าบนคีย์บอร์ดเพื่อเคลื่อนที่ โดยจะไปอย่างช้าๆทีละน้อย เราสามารถปรับค่าการเคลื่อนที่ในการ jog แต่ละครั้งได้โดยการปรับค่าที่ตัวโปรแกรม

3.2 การเคลื่อนที่ที่สามารถเคลื่อนที่ไปได้หลายจุดในการใส่ค่าเพียงครั้งเดียว

3.3 การเคลื่อนที่ที่เป็นการเคลื่อนที่ซ้ำได้ (loop moving)

7.4 แนวทางการพัฒนา

1. เพิ่มแกนการเคลื่อนที่ของตัวแขนกล

เพื่อให้มีพื้นที่ปฏิบัติงานของแขนกลเพิ่มขึ้นเราควรจะเพิ่มแกนของแขนกลที่ส่วนฐานของแขนกล โดยให้แขนกลที่สร้างเสร็จแล้วใน โครงงานนี้ เคลื่อนที่บน ball screwโดยการวางบนตัว ball Bearing ก็จะทำทำให้แขนกลนี้สามารถมีพื้นที่ปฏิบัติงานได้เพิ่มขึ้นอีกมากทีเดียว โดยอาจจะประกอบโต๊ะขึ้นมาเพื่อวางส่วนของแขนกลโดยเฉพาะ เพื่อให้เป็นฐานรองรับที่มั่นคงขึ้นกว่าฐานของแขนกลที่ได้ทำในโปรเจทนี้ ทำให้แขนกลมีอาการสั่นน้อยลงด้วยเพราะมีฐานที่รองรับที่ดี

2. เพิ่มส่วนการประยุกต์ใช้งาน

จากงานที่ได้ทำไปแล้วสามารถพัฒนาโดยการเพิ่มส่วนประยุกต์ใช้งานในด้านต่างๆได้ หลายแบบด้วยกันตรงส่วนปลายของแขนซึ่งมีพื้นที่ว่างในการเพิ่มส่วนการประยุกต์ใช้งาน เช่น มือหยิบจับชิ้นงานหรืออาจจะเป็นเครื่องเจาะขนาดเล็กซึ่งทำงานชนิดเบา เช่น การเจาะแผ่นปริ้น เป็นต้น

3. พัฒนาโปรแกรมที่ใช้ในการขับมอเตอร์และกำหนดระยะทาง

จากโปรแกรมที่ได้ทำในโครงงานชิ้นนี้ การใส่ค่าระยะทางลงไปในโปรแกรมเพื่อให้มอเตอร์ขับยังไม่สะดวก เพราะต้องนำจุดที่จะชี้มาคำนวณก่อน แล้วจึงใส่ค่าที่ได้ลงไปโปรแกรมที่พัฒนาต่อไปต้องสามารถใส่ค่าจุดไปได้เลยในทีเดียวและหากมีการเพิ่มแกนขึ้นอีก ก็ต้องพัฒนาโปรแกรมเพื่อรับค่า จุดที่จะให้ตัวแขนกลเคลื่อนที่ไปมาคำนวณเพื่อขับให้ให้มอเตอร์หมุนได้จำนวนรอบที่พอดีจะไปอยู่ที่จุดนั้นได้

ภาคผนวก

ภาคผนวก ก โปรแกรมสำหรับการขับและควบคุมตำแหน่งของมอเตอร์

```
program output;
uses crt;
var a1,a2 : byte;
   b1,b2,JogKey,Home1,Home2 : char;
   i,j,n,m,JogAx:integer;
   pos1,pos2:array[1..100] of longint;
   position1,position2,JogPos1,JogPos2,gap11,gap12,gap21,gap22,GoPos1,GoPos2: longint;
label jog;
const   Pa1=$0280;
        Pb1=$0285;
        Pc1=$0286;
        Pcont1=$0283;
        Pcont21=$0287;
        Pa2=$0290;
        Pb2=$0295;
        Pc2=$0296;
        Pcont2=$0293;
        Pcont22=$0297;
        Er1=100;
        Er2=20;
        JogFac1=240;
        JogFac2=100;
Function Status1: Byte; var tmp: byte;
begin
     port[Pb1]:=$06;
     port[Pb1]:=$04;
     tmp:=port[Pa1];
     port[Pb1]:=$06;
     port[Pb1]:=$0F;
     writeln(tmp);
     Status1 := tmp;
end;
```

```
Function Status2: Byte;
var tmp: byte;
begin
    port[Pb2]:=$06;
    port[Pb2]:=$04;
    tmp:=port[Pa2];
    port[Pb2]:=$06;
    port[Pb2]:=$0F;
    writeln(tmp);
    Status2 := tmp;
end;

Function Busy1: Boolean;
var tmp: byte;
begin
    port[Pb1]:=$06;
    port[Pb1]:=$04;
    tmp:=port[Pa1];
    port[Pb1]:=$06;
    port[Pb1]:=$0F;
    if (tmp and $01) = 1 then
      Busy1 := True
    else Busy1 := False;
end;

Function Busy2: Boolean;
var tmp: byte;
begin
    port[Pb2]:=$06;
    port[Pb2]:=$04;
    tmp:=port[Pa2];
    port[Pb2]:=$06;
    port[Pb2]:=$0F;
    if (tmp and $01) = 1 then
      Busy2 := True
    else Busy2 := False;
```

```
end;

Function Traj_Cmpt1: Boolean;
var tmp: byte;
begin
    port[Pb1]:=$06;
    port[Pb1]:=$04;
    tmp:=port[Pa1];
    port[Pb1]:=$06;
    port[Pb1]:=$0F;
    if (tmp and $02) = 2 then
     Traj_Cmpt1 := True
    else Traj_Cmpt1 := False;
end;

Function Traj_Cmpt2: Boolean;
var tmp: byte;
begin
    port[Pb2]:=$06;
    port[Pb2]:=$04;
    tmp:=port[Pa2];
    port[Pb2]:=$06;
    port[Pb2]:=$0F;
    if (tmp and $02) = 2 then
     Traj_Cmpt2 := True
    else Traj_Cmpt2 := False;
end;
procedure Wrt_Cmd1(Command: byte);
begin
    {set Port A to Output port}
    port[Pcont1]:=$89;
    port[Pb1]:=$0F;
    port[Pb1]:=$06;
    port[Pb1]:=$02;
    port[Pa1]:=Command;
    port[Pb1]:=$06;
```

```
    port[Pb1]:=$0F;
    {set Port A to Output port}
    port[Pcont1]:=$99;
    port[Pb1]:=$0F;
end;

procedure Wrt_Cmd2(Command: byte);
begin
    {set Port A to Output port}
    port[Pcont2]:=$89;
    port[Pb2]:=$0F;
    port[Pb2]:=$06;
    port[Pb2]:=$02;
    port[Pa2]:=Command;
    port[Pb2]:=$06;
    port[Pb2]:=$0F;
    {set Port A to Output port}
    port[Pcont2]:=$99;
    port[Pb2]:=$0F;
end;
procedure Wrt_Data1(Data: byte);
begin
    {set Port A to Output port}
    port[Pcont1]:=$89;
    port[Pb1]:=$0F;
    port[Pb1]:=$0E;
    port[Pb1]:=$0A;
    port[Pa1]:=Data;
    port[Pb1]:=$0E;
    port[Pb1]:=$0F;
    {set Port A to Output port}
    port[Pcont1]:=$99;
    port[Pb1]:=$0F;
end;

procedure Wrt_Data2(Data: byte);
```

```
begin
    {set Port A to Output port}
    port[Pcont2]:=$89;
    port[Pb2]:=$0F;
    port[Pb2]:=$0E;
    port[Pb2]:=$0A;
    port[Pa2]:=Data;
    port[Pb2]:=$0E;
    port[Pb2]:=$0F;
    {set Port A to Output port}
    port[Pcont2]:=$99;
    port[Pb2]:=$0F;
end;


Function Read_Data1: byte;
var tmp: byte;
begin
    port[Pb1]:=$0E;
    port[Pb1]:=$0C;
    tmp:= port[Pa1];
    port[Pb1]:=$0E;
    port[Pb1]:=$0F;
    Read_Data1:=tmp;
end;

Function Read_Data2: byte;
var tmp: byte;
begin
    port[Pb2]:=$0E;
    port[Pb2]:=$0C;
    tmp:= port[Pa2];
    port[Pb2]:=$0E;
    port[Pb2]:=$0F;
    Read_Data2:=tmp;
end;
```

```
Function Read_Pos1:LongInt;
var t1,t2,t3,t4:byte;
   tmp: longint;
begin
    t1:=Read_Data1;
    t2:=Read_Data1;
    repeat until not Busy1;
    t3:=Read_Data1;
    t4:=Read_Data1;
    tmp:=t1;
    tmp:= (tmp shl 8) or t2;
    tmp:= (tmp shl 8) or t3;
    tmp:= (tmp shl 8) or t4;
    GoPos1:=tmp;
    Read_Pos1 := tmp;
{    Read_Pos := t1*16777216 + t2*65536 + t3*256+t4;}
end;

Function Read_Pos2:LongInt;
var t1,t2,t3,t4:byte;
   tmp: longint;
begin
    t1:=Read_Data2;
    t2:=Read_Data2;
    repeat until not Busy2;
    t3:=Read_Data2;
    t4:=Read_Data2;
    tmp:=t1;
    tmp:= (tmp shl 8) or t2;
    tmp:= (tmp shl 8) or t3;
    tmp:= (tmp shl 8) or t4;
    GoPos2 := tmp;
    Read_Pos2 := tmp;
{    Read_Pos2 := t1*16777216 + t2*65536 + t3*256+t4;}
```

```
end;


begin


  {set Port A to Output port}
  port[Pcont21]:=$99;
  port[Pb1]:=$0F;
  port[Pcont1]:=$99;
  writeln('Please RESET Axis 1 now!!!!');

  port[Pcont22]:=$99;
  port[Pb2]:=$0F;
  port[Pcont2]:=$99;
  writeln('Please RESET Axis 2 now!!!!');

  {Reset}
  repeat until not Busy1;
  Wrt_Cmd1($00);

  repeat until not Busy2;
  Wrt_Cmd2($00);
  delay(4);

  Repeat Until (Status1 = $C4);
  if Busy1 then writeln('Axis 1 Busy')
   else writeln('Axis 1 Free.');

  Repeat Until (Status2 = $C4);
  if Busy2 then writeln('Axis 2 Busy')
   else writeln('Axis 2 Free.');

  { Mask interrupts }
  Wrt_Cmd1($1C);
  repeat until not Busy1;
```

```
Wrt_Data1($00);
Wrt_Data1($00);
repeat until not Busy1;

Wrt_Cmd1($1D);
repeat until not Busy1;
Wrt_Data1($00);
Wrt_Data1($00);

writeln('check status');
Repeat
 a1:= Status1;
writeln(a1);
Until (a1=$C0) or (a1=$80);
writeln('check status OK');

repeat until not Busy1;
Wrt_Cmd1($1D);
repeat until not Busy1;
Wrt_Data1($00);
Wrt_Data1($00);

writeln('Load Axis 1 filter Parameter.');
repeat until not Busy1;
Wrt_Cmd1($1E);
repeat until not Busy1;
Wrt_Data1($00);
Wrt_Data1($0f);
repeat until not Busy1;
{ kp }
Wrt_Data1($01);
Wrt_Data1($00);
repeat until not Busy1;
{ ki }
Wrt_Data1($08);
Wrt_Data1($00);
```

```
repeat until not Busy1;
{ kd }
Wrt_Data1($10);
Wrt_Data1($00);
repeat until not Busy1;
{ il }
Wrt_Data1($00);
Wrt_Data1($00);
repeat until not Busy1;

writeln('Update Axis 1 filter.');
Wrt_Cmd1($04);

{Read position1}
repeat until not Busy1;
Wrt_Cmd1($0A);
repeat until not Busy1;
writeln(Read_Pos1);

Wrt_Cmd2($1C);
repeat until not Busy2;
Wrt_Data2($00);
Wrt_Data2($00);
repeat until not Busy2;

Wrt_Cmd2($1D);
repeat until not Busy2;
Wrt_Data2($00);
Wrt_Data2($00);

writeln('check Status2');
Repeat
a2:= Status2;
writeln(a2);
Until (a2=$C0) or (a2=$80);
writeln('check Status2 OK');
```

```
repeat until not Busy2;
Wrt_Cmd2($1D);
repeat until not Busy2;
Wrt_Data2($00);
Wrt_Data2($00);

writeln('Load Axis 2 filter Parameter.');
repeat until not Busy2;
Wrt_Cmd2($1E);
repeat until not Busy2;
Wrt_Data2($00);
Wrt_Data2($0f);
repeat until not Busy2;
{ kp }
Wrt_Data2($04);
Wrt_Data2($00);
repeat until not Busy2;
{ ki }
Wrt_Data2($08);
Wrt_Data2($00);
repeat until not Busy2;
{ kd }
Wrt_Data2($40);
Wrt_Data2($00);
repeat until not Busy2;
{ il }
Wrt_Data2($00);
Wrt_Data2($00);
repeat until not Busy2;

writeln('Update Axis 2 filter.');
Wrt_Cmd2($04);

{Read position2}
repeat until not Busy2;
```

```
    Wrt_Cmd2($0A);
    repeat until not Busy2;
    writeln(Read_Pos2);
Jog:  repeat
  writeln('Do you like to jog the robot ? (y/n) ');
  b2:= readkey;
  until (b2='y') or (b2='n');

  if b2='y' then
  begin
  write('Choose Axis ( 1 or 2 ) ');
  readln(JogAx);
  writeln;
  writeln(' Press "a" to Turn Clockwise or "d" to Turn Counter Clockwise ');
  writeln;
  writeln(' Press "q" to Quite from jog ');
  JogPos1:=0;
  JogPos2:=0;


  if JogAx=1 then
  begin

  repeat
   JogKey:=readkey;
   if JogKey='d'then JogPos1:=JogPos1+JogFac1;
   if JogKey='a'then JogPos1:=JogPos1-JogFac1;

   Wrt_Cmd1($1F);
   repeat until not Busy1;
   Wrt_Data1($00);
   Wrt_Data1($2a);

   repeat until not Busy1;
   { Aceleration1}
   Wrt_Data1($0);
```

```
Wrt_Data1($0);
repeat until not Busy1;
Wrt_Data1($02);
Wrt_Data1($8);

repeat until not Busy1;
{ Velocity1 }
Wrt_Data1($3^`;
Wrt_Data1($00);
repeat until not Busy1;
Wrt_Data1($01);
Wrt_Data1($00);

repeat until not Busy1;
{ position1 }
Wrt_Data1($ff and (JogPos1 shr 24));
Wrt_Data1($ff and (JogPos1 shr 16));
repeat until not Busy1;
Wrt_Data1($ff and (JogPos1 shr 8));
Wrt_Data1($ff and JogPos1);

repeat until not Busy1;
Wrt_Cmd1($01);

until JogKey='q';
writeln('Do you like to Define this Position is Home (y/n)');
repeat
Home1:=readkey;
until (Home1='y')or(Home1='n');
if Home1='y'then
begin
   repeat until not Busy1;
   Wrt_Cmd1($02);
end;

end;
```

```
if JogAx=2 then
begin

repeat
JogKey:=readkey;
if JogKey='d'then JogPos2:=JogPos2+JogFac2;
if JogKey='a'then JogPos2:=JogPos2-JogFac2;

Wrt_Cmd2($1F);
repeat until not Busy2;
Wrt_Data2($00);
Wrt_Data2($2a);

repeat until not Busy2;
{ Aceleration2 }
Wrt_Data2($0);
Wrt_Data2($0);
repeat until not Busy2;
Wrt_Data2($03);
Wrt_Data2($8);

repeat until not Busy2;
{ Velocity2 }
Wrt_Data2($30);
Wrt_Data2($00);
repeat until not Busy2;
Wrt_Data2($01);
Wrt_Data2($00);

repeat until not Busy2;
{ position2 }
Wrt_Data2($ff and (JogPos2 shr 24));
Wrt_Data2($ff and (JogPos2 shr 16));
repeat until not Busy2;
```

```
  Wrt_Data2($ff and (JogPos2 shr 8));
  Wrt_Data2($ff and JogPos2);

  repeat until not Busy2;
  Wrt_Cmd2($01);

  until JogKey='q';

  writeln('Do you like to Define this Position is Home (y/n)');
  repeat
  Home2:=readkey;
  until (Home2='y')or(Home2='n');
  if Home2='y'then
  begin
     repeat until not Busy2;
     Wrt_Cmd2($02);
  end;
  end;
  goto jog;
end;

writeln('Enter the number of Loop  ');
readln(m);
writeln;
write('Enter the number of Positioning  ');
readln(n);
writeln;
for i:=1 to n do
  begin
  write('Enter Axis 1 Position ',(i),'>> ');
    read(pos1[i]);
  write('Enter Axis 2 Position ',(i),'>> ');
    read(pos2[i]);
  end;
for j := 1 to m do
begin
```

```
for i:=1 to n do
begin
{
repeat until not Busy1;
write('Enter Position Axis1 :...');

repeat until not Busy1;
write('Enter Position Axis 2 :...');
  read(position2);
repeat until not Busy2;
}
writeln('Load Axis 1 trajectory parameter.');
Wrt_Cmd1($1F);
repeat until not Busy1;
Wrt_Data1($00);
Wrt_Data1($2a);

repeat until not Busy1;
{ Acceleration1 }
Wrt_Data1($0);
Wrt_Data1($0);
repeat until not Busy1;
Wrt_Data1($06);
Wrt_Data1($8);

repeat until not Busy1;
{ Velocity1 }
Wrt_Data1($30);
Wrt_Data1($00);
repeat until not Busy1;
Wrt_Data1($01);
Wrt_Data1($00);

repeat until not Busy1;
{ Position1 }
```

```
Wrt_Data1($ff and (pos1[i] shr 24));
Wrt_Data1($ff and (pos1[i] shr 16));
repeat until not Busy1;
Wrt_Data1($ff and (pos1[i] shr 8));
Wrt_Data1($ff and pos1[i]);

writeln('Load Axis 2 trajectory parameter.');
Wrt_Cmd2($1F);
repeat until not Busy2;
Wrt_Data2($00);
Wrt_Data2($2a);

repeat until not Busy2;
{ Acceleration2 }
Wrt_Data2($0);
Wrt_Data2($0);
repeat until not Busy2;
Wrt_Data2($08);
Wrt_Data2($8);

repeat until not Busy2;
{ Velocity2 }
Wrt_Data2($30);
Wrt_Data2($00);
repeat until not Busy2;
Wrt_Data2($01);
Wrt_Data2($00);

repeat until not Busy2;
{ position2 }
Wrt_Data2($ff and (pos2[i] shr 24));
Wrt_Data2($ff and (pos2[i] shr 16));
repeat until not Busy2;
Wrt_Data2($ff and (pos2[i] shr 8));
Wrt_Data2($ff and pos2[i]);
```

```
repeat until not Busy1;
Wrt_Cmd1($01);

repeat until not Busy2;
Wrt_Cmd2($01);

{ Read Position1 }
repeat
 delay(500);
repeat until not Busy1;
 Wrt_Cmd1($0A);

repeat until not Busy1;
 writeln(Read_Pos1);

repeat until not Busy2;
 Wrt_Cmd2($0A);

repeat until not Busy2;
 writeln('          ',Read_Pos2);
repeat until not Busy1;
 gap11:=(Pos1[i]-Er1);
repeat until not Busy1;
 gap12:=(Pos1[i]+Er1);
repeat until not Busy2;
 gap21:=(Pos2[i]-Er2);
repeat until not Busy2;
 gap22:=(Pos2[i]+Er2);

 writeln((gap11),('  '),(gap12),('  '),(gap21),('  '),(gap22));

until (((GoPos1 > gap11)and(GoPos1 < gap12))and((GoPos2 > gap21)and(GoPos2 < gap22))) ;

if Traj_Cmpt1 then writeln('Trajectory Axis 1 Complete.')
 else writeln('Trajectory Axis 1 not Complete.');
```

```
    if Traj_Cmpt2 then writeln('Trajectory Axis 2 Complete.')
     else writeln('Trajectory Axis 2 not Complete.');
    end;
    end;
end.
```

ภาคผนวก ข

รายละเอียดของไมโครคอนโทรลเลอร์ LM629

National Semiconductor

November 1999

LM628/LM629
Precision Motion Controller

General Description

The LM628/LM629 are dedicated motion-control processors designed for use with a variety of DC and brushless DC servo motors, and other servomechanisms which provide a quadrature incremental position feedback signal. The parts perform the intensive, real-time computational tasks required for high performance digital motion control. The host control software interface is facilitated by a high-level command set. The LM628 has an 8-bit output which can drive either an 8-bit or a 12-bit DAC. The components required to build a servo system are reduced to the DC motor/actuator, an incremental encoder, a DAC, a power amplifier, and the LM628. An LM629-based system is similar, except that it provides an 8-bit PWM output for directly driving H-switches. The parts are fabricated in NMOS and packaged in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only). Both 6 MHz and 8 MHz maximum frequency versions are available with the suffixes -6 and -8, respectively, used to designate the versions. They incorporate an SDA core processor and cells designed by SDA.

Features

- 32-bit position, velocity, and acceleration registers
- Programmable digital PID filter with 16-bit coefficients
- Programmable derivative sampling interval
- 8- or 12-bit DAC output data (LM628)
- 8-bit sign-magnitude PWM output data (LM629)
- Internal trapezoidal velocity profile generator
- Velocity, target position, and filter parameters may be changed during motion
- Position and velocity modes of operation
- Real-time programmable host interrupts
- 8-bit parallel asynchronous host interface
- Quadrature incremental encoder interface with index pulse input
- Available in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only)

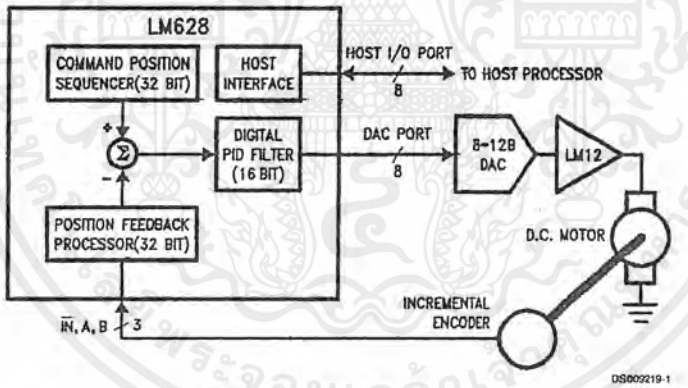
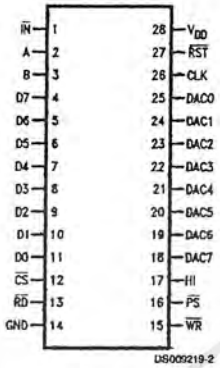


FIGURE 1. Block Diagram

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

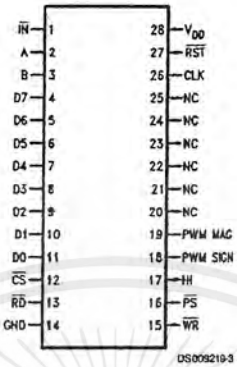
Connection Diagrams

LM628N



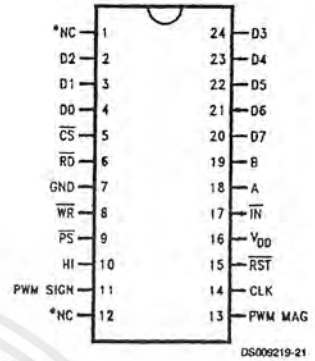
DS009219-2

LM629N



DS009219-3

LM629M



DS009219-21

*Do not connect.

Order Number LM629M-6, LM629M-8, LM628N-6, LM628N-8, LM629N-6 or LM629N-8
See NS Package Number M24B or N28B

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Voltage at Any Pin with Respect to GND	−0.3V to +7.0V
Ambient Storage Temperature	−65°C to +150°C
Lead Temperature	
28-pin Dual In-Line Package (Soldering, 4 sec.)	260°C
24-pin Surface Mount Package (Soldering, 10 sec.)	300°C
Maximum Power Dissipation ($T_A \leq 85°C$, (Note 2)	605 mW
ESD Tolerance (C_{ZAP} = 120 pF, R_{ZAP} = 1.5k)	2000V

Operating Ratings

Temperature Range	$-40°C < T_A < +85°C$
Clock Frequency:	
LM628N-6, LM629N-6, LM629M-6	$1.0\ MHz < f_{CLK} < 6.0\ MHz$
LM628N-8, LM629N-8, LM629M-8	$1.0\ MHz < f_{CLK} < 8.0\ MHz$
V_{DD} Range	$4.5V < V_{DD} < 5.5V$

DC Electrical Characteristics

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz)

Symbol	Parameter	Conditions	Tested Limits		Units
			Min	Max	
I_{DD}	Supply Current	Outputs Open		110	mA
INPUT VOLTAGES					
V_{IH}	Logic 1 Input Voltage		2.0		V
V_{IL}	Logic 0 Input Voltage			0.8	V
I_{IN}	Input Currents	$0 \leq V_{IN} \leq V_{DD}$	−10	10	µA
OUTPUT VOLTAGES					
V_{OH}	Logic 1	I_{OH} = −1.6 mA	2.4		V
V_{OL}	Logic 0	I_{OL} = 1.6 mA		0.4	V
I_{OUT}	TRI-STATE® Output Leakage Current	$0 \leq V_{OUT} \leq V_{DD}$	−10	10	µA

AC Electrical Characteristics

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz; C_{LOAD} = 50 pF; Input Test Signal $t_r = t_f$ = 10 ns)

Timing Interval	T#	Tested Limits		Units
		Min	Max	
ENCODER AND INDEX TIMING (See *Figure 2*)				
Motor-Phase Pulse Width	T1	$\frac{16}{f_{CLK}}$		µs
Dwell-Time per State	T2	$\frac{8}{f_{CLK}}$		µs
Index Pulse Setup and Hold (Relative to A and B Low)	T3	0		µs
CLOCK AND RESET TIMING (See *Figure 3*)				
Clock Pulse Width				
LM628N-6, LM629N-6, LM629M-6	T4	78		ns
LM628N-8, LM629N-8, LM629M-8	T4	57		ns
Clock Period				
LM628N-6, LM629N-6, LM629M-6	T5	166		ns
LM628N-8, LM629N-8, LM629M-8	T5	125		ns
Reset Pulse Width	T6	$\frac{8}{f_{CLK}}$		µs

AC Electrical Characteristics (Continued)

(V_{DD} and T_A per Operating Ratings; f_{CLK} = 6 MHz; C_{LOAD} = 50 pF; Input Test Signal $t_r = t_f$ = 10 ns)

Timing Interval	T#	Tested Limits		Units
		Min	Max	
STATUS BYTE READ TIMING (See *Figure 4*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Read Data Access Time	T10		180	ns
Read Data Hold Time	T11	0		ns
$\overline{RD}$ High to Hi-Z Time	T12		180	ns
COMMAND BYTE WRITE TIMING (See *Figure 5*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Busy Bit Delay	T13		(Note 3)	ns
$\overline{WR}$ Pulse Width	T14	100		ns
Write Data Setup Time	T15	50		ns
Write Data Hold Time	T16	120		ns
DATA WORD READ TIMING (See *Figure 6*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Read Data Access Time	T10		180	ns
Read Data Hold Time	T11	0		ns
$\overline{RD}$ High to Hi-Z Time	T12		180	ns
Busy Bit Delay	T13		(Note 3)	ns
Read Recovery Time	T17	120		ns
DATA WORD WRITE TIMING (See *Figure 7*)				
Chip-Select Setup/Hold Time	T7	0		ns
Port-Select Setup Time	T8	30		ns
Port-Select Hold Time	T9	30		ns
Busy Bit Delay	T13		(Note 3)	ns
$\overline{WR}$ Pulse Width	T14	100		ns
Write Data Setup Time	T15	50		ns
Write Data Hold Time	T16	120		ns
Write Recovery Time	T18	120		ns

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond the above Operating Ratings.

Note 2: When operating at ambient temperatures above 70°C, the device must be protected against excessive junction temperatures. Mounting the package on a printed circuit board having an area greater than three square inches and surrounding the leads and body with wide copper traces and large, uninterrupted areas of copper, such as a ground plane, suffices. The 28-pin DIP (N) and the 24-pin surface mount package (M) are molded plastic packages with solid copper lead frames. Most of the heat generated at the die flows from the die, through the copper lead frame, and into copper traces on the printed circuit board. The copper traces act as a heat sink. Double-sided or multi-layer boards provide heat transfer characteristics superior to those of single-sided boards.

Note 3: In order to read the busy bit, the status byte must first be read. The time required to read the busy bit far exceeds the time the chip requires to set the busy bit. It is, therefore, impossible to test actual busy bit delay. The busy bit is guaranteed to be valid as soon as the user is able to read it.

AC Electrical Characteristics (Continued)

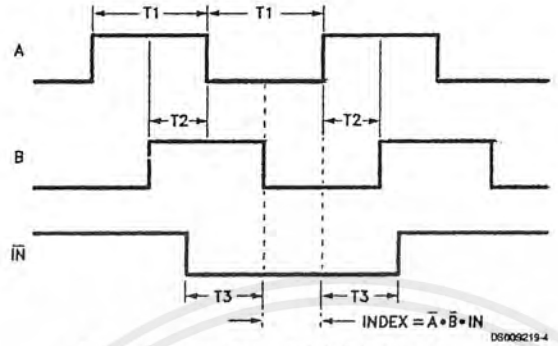


FIGURE 2. Quadrature Encoder Input Timing

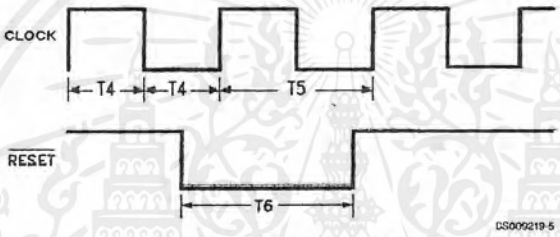


FIGURE 3. Clock and Reset Timing

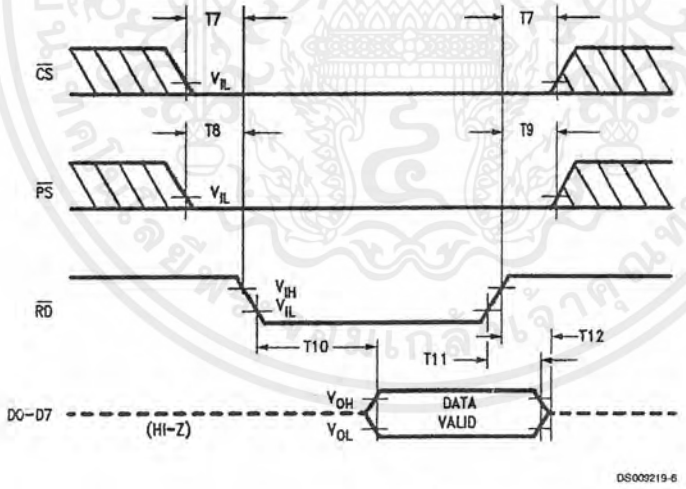


FIGURE 4. Status Byte Read Timing

AC Electrical Characteristics (Continued)

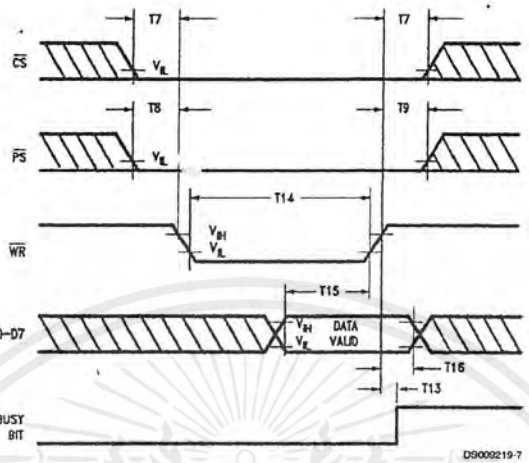


FIGURE 5. Command Byte Write Timing

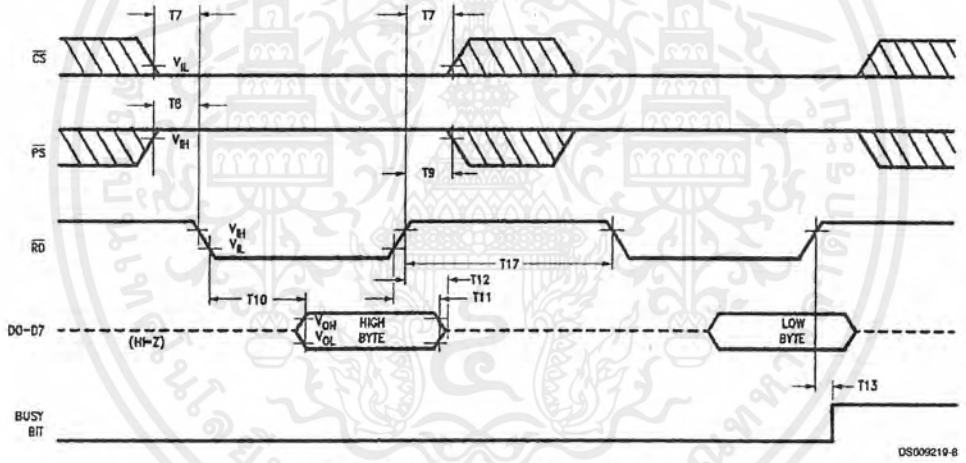


FIGURE 6. Data Word Read Timing

AC Electrical Characteristics (Continued)

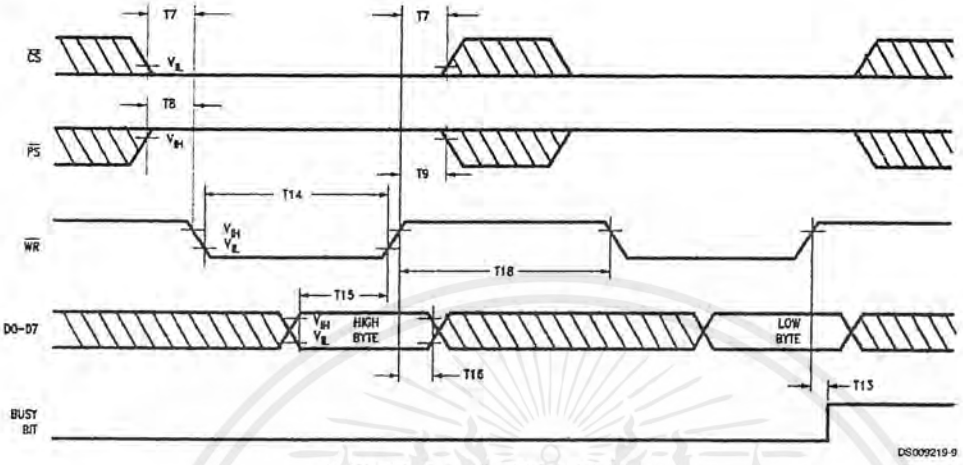


FIGURE 7. Data Word Write Timing

Pinout Description

(See Connection Diagrams) Pin numbers for the 24-pin surface mount package are indicated in parentheses.

Pin 1 (17), Index ($\overline{IN}$) Input: Receives optional index pulse from the encoder. Must be tied high if not used. The index position is read when Pins 1, 2, and 3 are low.

Pins 2 and 3 (18 and 19), Encoder Signal (A, B) Inputs: Receive the two-phase quadrature signals provided by the incremental encoder. When the motor is rotating in the positive ("forward") direction, the signal at Pin 2 leads the signal at Pin 3 by 90 degrees. Note that the signals at Pins 2 and 3 must remain at each encoder state (See *Figure 9*) for a minimum of 8 clock periods in order to be recognized. Because of a four-to-one resolution advantage gained by the method of decoding the quadrature encoder signals, this corresponds to a maximum encoder-state capture rate of 1.0 MHz (f_{CLK} = 8.0 MHz) or 750 kHz (f_{CLK} = 6.0 MHz). For other clock frequencies the encoder signals must also remain at each state a minimum of 8 clock periods.

Pins 4 to 11 (20 to 24 and 2 to 4), Host I/O Port (D0 to D7): Bi-directional data port which connects to host computer/processor. Used for writing commands and data to the LM628, and for reading the status byte and data from the LM628, as controlled by $\overline{CS}$ (Pin 12), $\overline{PS}$ (Pin 16), $\overline{RD}$ (Pin 13), and $\overline{WR}$ (Pin 15).

Pin 12 (5), Chip Select ($\overline{CS}$) Input: Used to select the LM628 for writing and reading operations.

Pin 13 (6), Read ($\overline{RD}$) Input: Used to read status and data.

Pin 14 (7), Ground (GND): Power-supply return pin.

Pin 15 (8), Write ($\overline{WR}$) Input: Used to write commands and data.

Pin 16 (9), Port Select ($\overline{PS}$) Input: Used to select command or data port. Selects command port when low, data port when high. The following modes are controlled by Pin 16:

1. Commands are written to the command port (Pin 16 low),
2. Status byte is read from command port (Pin 16 low), and
3. Data is written and read via the data port (Pin 16 high).

Pin 17 (10), Host Interrupt (HI) Output: This active-high signal alerts the host (via a host interrupt service routine) that an interrupt condition has occurred.

Pins 18 to 25, DAC Port (DAC0 to DAC7): Output port which is used in three different modes:

1. LM628 (8-bit output mode): Outputs latched data to the DAC. The MSB is Pin 18 and the LSB is Pin 25.
2. LM628 (12-bit output mode): Outputs two, multiplexed 6-bit words. The less-significant word is output first. The MSB is on Pin 18 and the LSB is on Pin 23. Pin 24 is used to demultiplex the words; Pin 24 is low for the less-significant word. The positive-going edge of the signal on Pin 25 is used to strobe the output data. *Figure 8* shows the timing of the multiplexed signals.
3. LM629 (sign/magnitude outputs): Outputs a PWM sign signal on Pin 18 (11 for surface mount), and a PWM magnitude signal on Pin 19 (13 for surface mount). Pins 20 to 25 are not used in the LM629. *Figure 11* shows the PWM output signal format.

Pin 26 (14), Clock (CLK) Input: Receives system clock.

Pin 27 (15), Reset ($\overline{RST}$) Input: Active-low, positive-edge triggered, resets the LM628 to the internal conditions shown below. Note that the reset pulse must be logic low for a minimum of 8 clock periods. Reset does the following:

1. Filter coefficient and trajectory parameters are zeroed.
2. Sets position error threshold to maximum value (7FFF hex), and effectively executes command LPEI.
3. The SBPA/SBPR interrupt is masked (disabled).
4. The five other interrupts are unmasked (enabled).
5. Initializes current position to zero, or "home" position.
6. Sets derivative sampling interval to $2048/f_{CLK}$ or 256 µs for an 8.0 MHz clock.
7. DAC port outputs 800 hex to "zero" a 12-bit DAC and then reverts to 80 hex to "zero" an 8-bit DAC.

Pinout Description (Continued)

Immediately after releasing the reset pin from the LM628, the status port should read "00". If the reset is successfully completed, the status word will change to hex "84" or "C4" within 1.5 ms. If the status word has not changed from hex "00" to "84" or "C4" within 1.5 ms, perform another reset and repeat the above steps. To be certain that the reset was properly performed, execute a **RSTI** command. If the chip has reset properly, the status byte will change from hex "84" or "C4" to hex "80" or "C0". If this does not occur, perform another reset and repeat the above steps.

Pin 28 (16), Supply Voltage (V_{DD}): Power supply voltage (+5V).

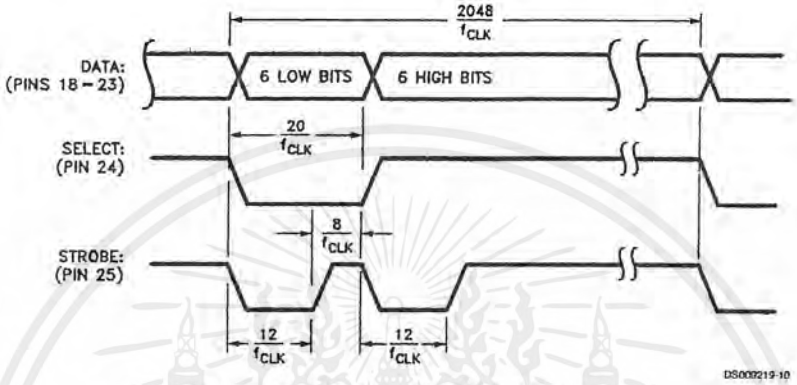


FIGURE 8. 12-Bit Multiplexed Output Timing

Theory of Operation

INTRODUCTION

The typical system block diagram (See *Figure 1*) illustrates a servo system built using the LM628. The host processor communicates with the LM628 through an I/O port to facilitate programming a trapezoidal velocity profile and a digital compensation filter. The DAC output interfaces to an external digital-to-analog converter to produce the signal that is power amplified and applied to the motor. An incremental encoder provides feedback for closing the position servo loop. The trapezoidal velocity profile generator calculates the required trajectory for either position or velocity mode of operation. In operation, the LM628 subtracts the actual position (feedback position) from the desired position (profile generator position), and the resulting position error is processed by the digital filter to drive the motor to the desired position. *Table 1* provides a brief summary of specifications offered by the LM628/LM629:

POSITION FEEDBACK INTERFACE

The LM628 interfaces to a motor via an incremental encoder. Three inputs are provided: two quadrature signal inputs, and an index pulse input. The quadrature signals are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the LM628 internal position register is incremented or decremented accordingly. This provides four times the resolution over the number of lines provided by the encoder. See *Figure 9*. Each of the encoder signal inputs is synchronized with the LM628 clock.

The optional index pulse output provided by some encoders assumes the logic-low state once per revolution. If the LM628 is so programmed by the user, it will record the absolute motor position in a dedicated register (the index register) at the time when all three encoder inputs are logic low.

If the encoder does not provide an index output, the LM628 index input can also be used to record the home position of the motor. In this case, typically, the motor will close a switch which is arranged to cause a logic-low level at the index input, and the LM628 will record motor position in the index register and alert (interrupt) the host processor. Permanently grounding the index input will cause the LM628 to malfunction.

TABLE 1. System Specifications Summary

Position Range	–1,073,741,824 to 1,073,741,823 counts
Velocity Range	0 to 1,073,741,823/2^{16} counts/sample; ie, 0 to 16,383 counts/sample, with a resolution of $1/2^{16}$ counts/sample
Acceleration Range	0 to 1,073,741,823/2^{16} counts/sample/sample; ie, 0 to 16,383 counts/sample/sample, with a resolution of $1/2^{16}$ counts/sample/sample
Motor Drive Output	LM628: 8-bit parallel output to DAC, or 12-bit multiplexed output to DAC LM629: 8-bit PWM sign/magnitude signals
Operating Modes	Position and Velocity

Theory of Operation (Continued)

TABLE 1. System Specifications Summary (Continued)

Feedback Device	Incremental Encoder (quadrature signals; support for index pulse)
Control Algorithm	Proportional Integral Derivative (PID) (plus programmable integration limit)
Sample Intervals	Derivative Term: Programmable from $2048/f_{CLK}$ to $(2048 * 256)/f_{CLK}$ in steps of $2048/f_{CLK}$ (256 to 65,536 µs for an 8.0 MHz clock). Proportional and Integral: $2048/f_{CLK}$

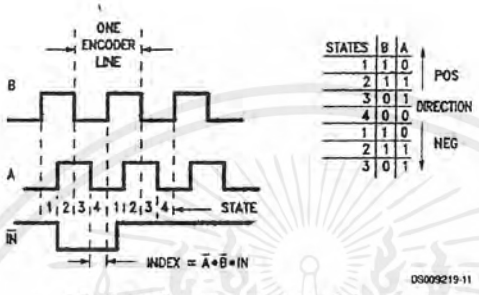


FIGURE 9. Quadrature Encoder Signals

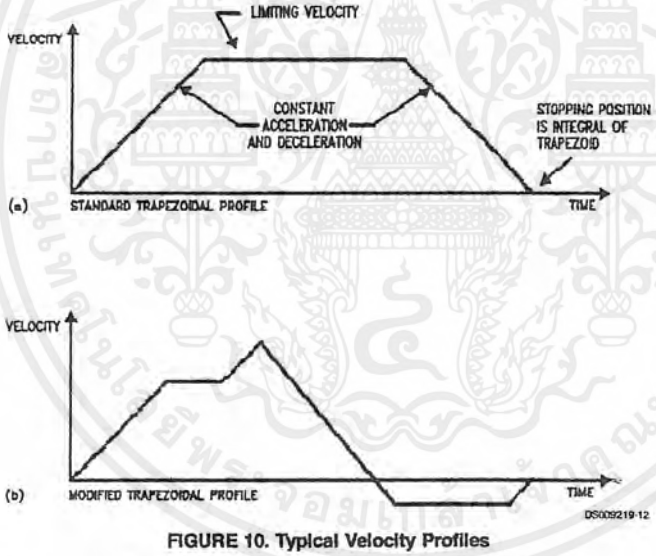


FIGURE 10. Typical Velocity Profiles

VELOCITY PROFILE (TRAJECTORY) GENERATION

The trapezoidal velocity profile generator computes the desired position of the motor versus time. In the position mode of operation, the host processor specifies acceleration, maximum velocity, and final position. The LM628 uses this information to affect the move by accelerating as specified until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. At any time during the move the maximum velocity and/or the target position may be changed, and the motor will accelerate or decelerate accordingly. *Figure 10* illustrates two typical trapezoidal velocity profiles. *Figure 10*(a) shows a simple trapezoid, while *Figure 10*(b) is an example of what the trajectory looks like when velocity and position are changed at different times during the move.

When operating in the velocity mode, the motor accelerates to the specified velocity at the specified acceleration rate and maintains the specified velocity until commanded to stop. The velocity is maintained by advancing the desired position at a constant rate. If there are disturbances to the motion during velocity mode operation, the long-time average velocity remains constant. If the motor is unable to maintain the specified velocity (which could be caused by a locked rotor, for example), the desired position will continue to be increased, resulting in a very large position error. If this condi-

Theory of Operation (Continued)

tion goes undetected, and the impeding force on the motor is subsequently released, the motor could reach a very high velocity in order to catch up to the desired position (which is still advancing as specified). This condition is easily detected; see commands LPEI and LPES.

All trajectory parameters are 32-bit values. Position is a signed quantity. Acceleration and velocity are specified as 16-bit, positive-only integers having 16-bit fractions. The integer portion of velocity specifies how many counts per sampling interval the motor will traverse. The fractional portion designates an additional fractional count per sampling interval. Although the position resolution of the LM628 is limited to integer counts, the fractional counts provide increased average velocity resolution. Acceleration is treated in the same manner. Each sampling interval the commanded acceleration value is added to the current desired velocity to generate a new desired velocity (unless the command velocity has been reached).

One determines the trajectory parameters for a desired move as follows. If, for example, one has a 500-line shaft encoder, desires that the motor accelerate at one revolution per second per second until it is moving at 600 rpm, and then decelerate to a stop at a position exactly 100 revolutions from the start, one would calculate the trajectory parameters as follows:

let P = target position (units = encoder counts)
let R = encoder lines * 4 (system resolution)
then R = 500 * 4 = 2000
and P = 2000 * desired number of revolutions
P = 2000 * 100 revs = 200,000 counts (value to load)
P (coding) = 00030D40 (hex code written to LM628)
let V = velocity (units = counts/sample)
let T = sample time (seconds) = 341 µs (with 6 MHz clock)
let C = conversion factor = 1 minute/60 seconds
then V = R * T * C * desired rpm
and V = 2000 * 341E–6 * 1/60 * 600 rpm
V = 6.82 counts/sample
V (scaled) = 6.82 * 65,536 = 446,955.52
V (rounded) = 446,956 (value to load)
V (coding) = 0006D1EC (hex code written to LM628)
let A = acceleration (units = counts/sample/sample)
A = R * T * T * desired acceleration (rev/sec/sec)
then A = 2000 * 341E–6 * 341E-6 * 1 rev/sec/sec
and A = 2.33E–4 counts/sample/sample
A (scaled) = 2.33E–4 * 65,536 = 15.24
A (rounded) = 15 (value to load)
A (coding) = 0000000F (hex code written to LM628)

The above position, velocity, and acceleration values must be converted to binary codes to be loaded into the LM628. The values shown for velocity and acceleration must be multiplied by 65,536 (as shown) to adjust for the required integer/fraction format of the input data. Note that after scaling the velocity and acceleration values, literal fractional data cannot be loaded; the data must be rounded and converted to binary. The factor of four increase in system resolution is due to the method used to decode the quadrature encoder signals, see *Figure 9*.

PID COMPENSATION FILTER

The LM628 uses a digital Proportional Integral Derivative (PID) filter to compensate the control loop. The motor is held at the desired position by applying a restoring force to the motor that is proportional to the position error, plus the integral of the error, plus the derivative of the error. The following discrete-time equation illustrates the control performed by the LM628:

$$u(n) = kp*e(n) + ki\sum_{N=0}^{n} e(n) + kd[e(n') - e(n'-1)]; \quad \text{(Eq. 1)} \qquad (1)$$

where u(n) is the motor control signal output at sample time n, e(n) is the position error at sample time n, n' indicates sampling at the derivative sampling rate, and kp, ki, and kd are the discrete-time filter parameters loaded by the users.

The first term, the proportional term, provides a restoring force porportional to the position error, just as does a spring obeying Hooke's law. The second term, the integration term, provides a restoring force that grows with time, and thus ensures that the static position error is zero. If there is a constant torque loading, the motor will still be able to achieve zero position error.

The third term, the derivative term, provides a force proportional to the rate of change of position error. It acts just like viscous damping in a damped spring and mass system (like a shock absorber in an automobile). The sampling interval associated with the derivative term is user-selectable; this capability enables the LM628 to control a wider range of inertial loads (system mechanical time constants) by providing a better approximation of the continuous derivative. In general, longer sampling intervals are useful for low-velocity operations.

In operation, the filter algorithm receives a 16-bit error signal from the loop summing-junction. The error signal is saturated at 16 bits to ensure predictable behavior. In addition to being multiplied by filter coefficient kp, the error signal is added to an accumulation of previous errors (to form the integral signal) and, at a rate determined by the chosen *derivative* sampling interval, the previous error is subtracted from it (to form the derivative signal). All filter multiplications are 16-bit operations; only the bottom 16 bits of the product are used.

The integral signal is maintained to 24 bits, but only the top 16 bits are used. This scaling technique results in a more usable (less sensitive) range of coefficient ki values. The 16 bits are right-shifted eight positions and multiplied by filter coefficient ki to form the term which contributes to the motor control output. The absolute magnitude of this product is compared to coefficient il, and the lesser, appropriately signed magnitude then contributes to the motor control signal.

The derivative signal is multiplied by coefficient kd each *derivative* sampling interval. This product contributes to the motor control output *every* sample interval, independent of the user-chosen *derivative* sampling interval.

The kp, limited ki, and kd product terms are summed to form a 16-bit quantity. Depending on the output mode (wordsize), either the top 8 or top 12 bits become the motor control output signal.

Theory of Operation (Continued)

LM628 READING AND WRITING OPERATIONS

The host processor writes commands to the LM628 via the host I/O port when Port Select ($\overline{PS}$) input (Pin 16) is logic low. The desired command code is applied to the parallel port line and the Write ($\overline{WR}$) input (Pin 15) is strobed. The command byte is latched into the LM628 on the rising edge of the $\overline{WR}$ input. When writing command bytes it is necessary to first read the status byte and check the state of a flag called the "busy bit" (Bit 0). If the busy bit is logic high, no command write may take place. The busy bit is never high longer than 100 µs, and typically falls within 15 µs to 25 µs.

The host processor reads the LM628 status byte in a similar manner: by strobing the Read ($\overline{RD}$) input (Pin 13) when $\overline{PS}$ (Pin 16) is low; status information remains valid as long as $\overline{RD}$ is low.

Writing and reading data to/from the LM628 (as opposed to writing commands and reading status) are done with $\overline{PS}$ (Pin 16) logic high. These writes and reads are always an integral number (from one to seven) of two-byte words, with the first byte of each word being the more significant. Each byte requires a write ($\overline{WR}$) or read ($\overline{RD}$) strobe. When transferring data words (byte-pairs), it is necessary to first read the status byte and check the state of the busy bit. When the busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). Data transfers are accomplished via LM628-internal interrupts (which are not nested); the busy bit informs the host processor when the LM628 may not be interrupted for data transfer (or a command byte). If a command is written when the busy bit is high, the command will be ignored.

The busy bit goes high immediately after writing a command byte, or reading or writing a second byte of data (See *Figure 5* thru *Figure 7*).

MOTOR OUTPUTS

The LM628 DAC output port can be configured to provide either a latched eight-bit parallel output or a multiplexed 12-bit output. The 8-bit output can be directly connected to a flow-through (non-input-latching) D/A converter; the 12-bit output can be easily demultiplexed using an external 6-bit latch and an input-latching 12-bit D/A converter. The DAC output data is offset-binary coded; the 8-bit code for zero is 80 hex and the 12-bit code for zero is 800 hex. Values less than these cause a negative torque to be applied to the motor and, conversely, larger values cause positive motor torque. The LM628, when configured for 12-bit output, provides signals which control the demultiplexing process. See for details.

The LM629 provides 8-bit, sign and magnitude PWM output signals for directly driving switch-mode motor-drive amplifiers. *Figure 11* shows the format of the PWM magnitude output signal.

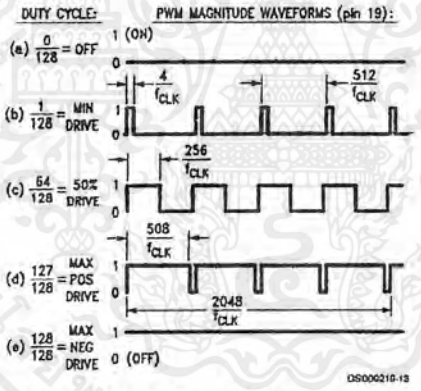


FIGURE 11. PWM Output Signal Format (Sign output (pin 18) not shown)

TABLE 2. LM628 User Command Set

Command	Type	Description	Hex	Data Bytes	Note
RESET	Initialize	Reset LM628	00	0	1
PORT8	Initialize	Select 8-Bit Output	05	0	2
PORT12	Initialize	Select 12-Bit Output	06	0	2
DFH	Initialize	Define Home	02	0	1
SIP	Interrupt	Set Index Position	03	0	1
LPEI	Interrupt	Interrupt on Error	1B	2	1
LPES	Interrupt	Stop on Error	1A	2	1
SBPA	Interrupt	Set Breakpoint, Absolute	20	4	1
SBPR	Interrupt	Set Breakpoint, Relative	21	4	1

Theory of Operation (Continued)

TABLE 2. LM628 User Command Set (Continued)

Command	Type	Description	Hex	Data Bytes	Note
MSKI	Interrupt	Mask Interrupts	1C	2	1
RSTI	Interrupt	Reset Interrupts	1D	2	1
LFIL	Filter	Load Filter Parameters	1E	2 to 10	1
UDF	Filter	Update Filter	04	0	1
LTRJ	Trajectory	Load Trajectory	1F	2 to 14	1
STT	Trajectory	Start Motion	01	0	3
RDSTAT	Report	Read Status Byte	None	1	1, 4
RDSIGS	Report	Read Signals Register	0C	2	1
RDIP	Report	Read Index Position	09	4	1
RDDP	Report	Read Desired Position	08	4	1
RDRP	Report	Read Real Position	0A	4	1
RDDV	Report	Read Desired Velocity	07	4	1
RDRV	Report	Read Real Velocity	0B	2	1
RDSUM	Report	Read Integration Sum	0D	2	1

Note 4: Commands may be executed "On the Fly" during motion.

Note 5: Commands not applicable to execution during motion.

Note 6: Command may be executed during motion if acceleration parameter was not changed.

Note 7: Command needs no code because the command port status-byte read is totally supported by hardware.

User Command Set

GENERAL

The following paragraphs describe the user command set of the LM628. Some of the commands can be issued alone and some require a supporting data structure. As examples, the command STT (STarT motion) does not require additional data; command LFIL (Load FILter parameters) requires additional data (derivative-term sampling interval and/or filter parameters).

Commands are categorized by function: initialization, interrupt control, filter control, trajectory control, and data reporting. The commands are listed in *Table 2* and described in the following paragraphs. Along with each command name is its command-byte code, the number of accompanying data bytes that are to be written (or read), and a comment as to whether the command is executable during motion.

Initialization Commands

The following four LM628 user commands are used primarily to initialize the system for use.

RESET COMMAND: RESET the LM628

Command Code: 00 Hex
Data Bytes: None
Executable During Motion: Yes

This command (and the hardware reset input, Pin 27) results in setting the following data items to zero: filter coefficients and their input buffers, trajectory parameters and their input buffers, and the motor control output. A zero motor control output is a half-scale, offset-binary code: (80 hex for the 8-bit output mode; 800 hex for 12-bit mode). During reset, the DAC port outputs 800 hex to "zero" a 12-bit DAC and reverts to 80 hex to "zero" an 8-bit DAC. The command also clears five of the six interrupt masks (only the SBPA/SBPR interrupt is masked), sets the output port size to 8 bits, and defines the current absolute position as home. Reset, which may be executed at any time, will be completed in less than 1.5 ms. Also see commands PORT8 and PORT12.

PORT8 COMMAND: Set Output PORT Size to 8 Bits

Command Code: 05 Hex
Data Bytes: None
Executable During Motion: Not Applicable

The default output port size of the LM628 is 8 bits; so the PORT8 command need not be executed when using an 8-bit DAC. This command must not be executed when using a 12-bit converter; it will result in erratic, unpredictable motor behavior. The 8-bit output port size is the required selection when using the LM629, the PWM-output version of the LM628.

PORT12 COMMAND: Set Output PORT Size to 12 Bits

Command Code: 06 Hex
Data Bytes: None
Executable During Motion: Not Applicable

When a 12-bit DAC is used, command PORT12 should be issued very early in the initialization process. Because use of this command is determined by system hardware, there is only one foreseen reason to execute it later: if the RESET command is issued (because an 8-bit output would then be selected as the default) command PORT12 should be immediately executed. This command must not be issued when using an 8-bit converter or the LM629, the PWM-output version of the LM628.

DFH COMMAND: DeFine Home

Command Code: 02 Hex
Data Bytes: None
Executable During Motion: Yes

Initialization Commands (Continued)

This command declares the current position as "home", or absolute position 0 (Zero). If DFH is executed during motion it will not affect the stopping position of the on-going move unless command STT is also executed.

Interrupt Control Commands

The following seven LM628 user commands are associated with conditions which can be used to interrupt the host computer. In order for any of the potential interrupt conditions to actually interrupt the host via Pin 17, the corresponding bit in the interrupt mask data associated with command MSKI must have been set to logic high (the non-masked state).

The identity of all interrupts is made known to the host via reading and parsing the status byte. Even if all interrupts are masked off via command MSKI, the state of each condition is still reflected in the status byte. This feature facilitates polling the LM628 for status information, as opposed to interrupt driven operation.

SIP COMMAND: Set Index Position

Command Code:	03 Hex
Data Bytes:	None
Executable During Motion:	Yes

After this command is executed, the absolute position which corresponds to the occurrence of the next index pulse input will be recorded in the index register, and bit 3 of the status byte will be set to logic high. The position is recorded when both encoder-phase inputs and the index pulse input are logic low. This register can then be read by the user (see description for command RDIP) to facilitate aligning the definition of home position (see description of command DFH) with an index pulse. The user can also arrange to have the LM628 interrupt the host to signify that an index pulse has occurred. See the descriptions for commands MSKI and RSTI.

LPEI COMMAND: Load Position Error for Interrupt

Command Code:	1B Hex
Data Bytes:	Two
Data Range:	0000 to 7FFF Hex
Executable During Motion:	Yes

An excessive position error (the output of the loop summing junction) can indicate a serious system problem; e.g., a stalled rotor. Instruction LPEI allows the user to input a threshold for position error detection. Error detection occurs when the absolute magnitude of the position error exceeds the threshold, which results in bit 5 of the status byte being set to logic high. If it is desired to also stop (turn off) the motor upon detecting excessive position error, see command LPES, below. The first byte of threshold data written with command LPEI is the more significant. The user can have the LM628 interrupt the host to signify that an excessive position error has occurred. See the descriptions for commands MSKI and RSTI.

LPES COMMAND: Load Position Error for Stopping

Command Code:	1A Hex
Data Bytes:	Two
Data Range:	0000 to 7FFF Hex
Executable During Motion:	Yes

Instruction LPES is essentially the same as command LPEI above, but adds the feature of turning off the motor upon detecting excessive position error. The motor drive is not actually switched off, it is set to half-scale, the offset-binary code for zero. As with command LPEI, bit 5 of the status byte is also set to logic high. The first byte of threshold data written with command LPES is the more significant. The user can have the LM628 interrupt the host to signify that an excessive position error has occurred. See the descriptions for commands MSKI and RSTI.

SBPA COMMAND:

Command Code:	20 Hex
Data Bytes:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command enables the user to set a breakpoint in terms of absolute position. Bit 6 of the status byte is set to logic high when the breakpoint position is reached. This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM628 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

SBPR COMMAND:

Command Code:	21 Hex
Data Bytes:	Four
Data Range:	See Text
Executable During Motion:	Yes

This command enables the user to set a breakpoint in terms of relative position. As with command SBPA, bit 6 of the status byte is set to logic high when the breakpoint position (relative to the current commanded target position) is reached. The relative breakpoint input value must be such that when this value is added to the target position the result remains within the absolute position range of the system (C0000000 to 3FFFFFFF hex). This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM628 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

MSKI COMMAND: MaSK Interrupts

Command Code:	1C Hex
Data Bytes:	Two
Data Range:	See Text
Executable During Motion:	Yes

The MSKI command lets the user determine which potential interrupt condition(s) will interrupt the host. Bits 1 through 6 of the status byte are indicators of the six conditions which are candidates for host interrupt(s). When interrupted, the host then reads the status byte to learn which condition(s) occurred. Note that the MSKI command is immediately followed by two data bytes. Bits 1 through 6 of the second (less significant) byte written determine the masked/unmasked status of each potential interrupt. Any zero(s) in this 6-bit field will mask the corresponding interrupt(s); any one(s) enable the interrupt(s). Other bits comprising the two bytes have no effect. The mask controls only the host interrupt process; reading the status byte will still reflect the actual conditions independent of the mask byte. See *Table 3*.

Interrupt Control Commands

(Continued)

TABLE 3. Mask and Reset Bit Allocations for Interrupts

Bit Position	Function
Bits 15 thru 7	Not Used
Bit 6	Breakpoint Interrupt
Bit 5	Position-Error Interrupt
Bit 4	Wrap-Around Interrupt
Bit 3	Index-Pulse Interrupt
Bit 2	Trajectory-Complete Interrupt
Bit 1	Command-Error Interrupt
Bit 0	Not Used

RSTI COMMAND: ReSeT Interrupts

Command Code: 1D Hex
Data Bytes: Two
Data Range: See Text
Executable During Motion: Yes

When one of the potential interrupt conditions of *Table 3* occurs, command RSTI is used to reset the corresponding interrupt flag bit in the status byte. The host may reset one or all flag bits. Resetting them one at a time allows the host to service them one at a time according to a priority programmed by the user. As in the MSKI command, bits 1 through 6 of the second (less significant) byte correspond to the potential interrupt conditions shown in *Table 3*. Also see description of RDSTAT command. Any zero(s) in this 6-bit field reset the corresponding interrupt(s). The remaining bits have no effect.

Filter Control Commands

The following two LM628 user commands are used for setting the derivative-term sampling interval, for adjusting the filter parameters as required to tune the system, and to control the timing of these system changes.

LFIL COMMAND: Load FILter Parameters

Command Code: 1E Hex
Data Bytes: Two to Ten
Data Ranges...
Filter Control Word: See Text
Filter Coefficients: 0000 to 7FFF Hex (Pos Only)
Integration Limit: 0000 to 7FFF Hex (Pos Only)
Executable During Motion: Yes

The filter parameters (coefficients) which are written to the LM628 to control loop compensation are: kp, ki, kd, and il (integration limit). The integration limit (il) constrains the contribution of the integration term

$$\left[ki * \sum_{N=0}^{n} e(n) \right]$$

(see Eq. 1) to values equal to or less than a user-defined maximum value; this capability minimizes integral or reset "wind-up" (an overshooting effect of the integral action). The positive-only input value is compared to the absolute magnitude of the integration term; when the magnitude of integration term value exceeds il, the il value (with appropriate sign) is substituted for the integration term value.

The derivative-term sampling interval is also programmable via this command. After writing the command code, the first two data bytes that are written specify the derivative-term sampling interval and which of the four filter parameters is/are to be written via any forthcoming data bytes. The first byte written is the more significant. Thus the two data bytes constitute a filter control word that informs the LM628 as to the nature and number of any following data bytes. See *Table 4*.

TABLE 4. Filter Control word Bit Allocation

Bit Position	Function
Bit 15	Derivative Sampling Interval Bit 7
Bit 14	Derivative Sampling Interval Bit 6
Bit 13	Derivative Sampling Interval Bit 5
Bit 12	Derivative Sampling Interval Bit 4
Bit 11	Derivative Sampling Interval Bit 3
Bit 10	Derivative Sampling Interval Bit 2
Bit 9	Derivative Sampling Interval Bit 1
Bit 8	Derivative Sampling Interval Bit 0
Bit 7	Not Used
Bit 6	Not Used
Bit 5	Not Used
Bit 4	Not Used
Bit 3	Loading kp Data
Bit 2	Loading ki Data
Bit 1	Loading kd Data
Bit 0	Loading il Data

Bits 8 through 15 select the derivative-term sampling interval. See *Table 5*. The user must locally save and restore these bits during successive writes of the filter control word.

Bits 4 through 7 of the filter control word are not used.

Bits 0 to 3 inform the LM628 as to whether any or all of the filter parameters are about to be written. The user may choose to update any or all (or none) of the filter parameters. Those chosen for updating are so indicated by logic one(s) in the corresponding bit position(s) of the filter control word.

The data bytes specified by and immediately following the filter control word are written in pairs to comprise 16-bit words. The order of sending the data words to the LM628 corresponds to the descending order shown in the above description of the filter control word; i.e., beginning with kp, then ki, kd and il. The first byte of each word is the more-significant byte. Prior to writing a word (byte pair) it is necessary to check the busy bit in the status byte for readiness. The required data is written to the primary buffers of a double-buffered scheme by the above described operations; it is not transferred to the secondary (working) registers until the UDF command is executed. This fact can be used advantageously; the user can input numerous data ahead of their actual use. This simple pipeline effect can relieve potential host computer data communications bottlenecks, and facilitates easier synchronization of multiple-axis controls.

UDF COMMAND: UpDate Filter

Command Code: 04 Hex
Data Bytes: None

Filter Control Commands (Continued)

Executable During Motion: Yes

The UDF command is used to update the filter parameters, the specifics of which have been programmed via the LFIL command. Any or all parameters (derivative-term sampling interval, kp, ki, kd, and/or il) may be changed by the appropriate command(s), but command UDF must be executed to affect the change in filter tuning. Filter updating is synchronized with the calculations to eliminate erratic or spurious behavior.

Trajectory Control Commands

The following two LM628 user commands are used for setting the trajectory control parameters (position, velocity, acceleration), mode of operation (position or velocity), and direction (velocity mode only) as required to describe a desired motion or to select the mode of a manually directed stop, and to control the timing of these system changes.

LTRJ COMMAND: Load TRaJectory Parameters

Command Code:	1F Hex
Data Bytes:	Two to Fourteen
Data Ranges...	
Trajectory Control Word:	See Text
Position:	C0000000 to 3FFFFFFF Hex
Velocity:	00000000 to 3FFFFFFF Hex (Pos Only)
Acceleration:	00000000 to 3FFFFFFF Hex (Pos Only)
Executable During Motion:	Conditionally, See Text

TABLE 5. Derivative-Term Sampling Interval Selection Codes

	Bit Position								Selected Derivative
	15	14	13	12	11	10	9	8	Sampling Interval
	0	0	0	0	0	0	0	0	256 µs
	0	0	0	0	0	0	0	1	512 µs
	0	0	0	0	0	0	1	0	768 µs
	0	0	0	0	0	0	1	1	1024 µs, etc...
thru	1	1	1	1	1	1	1	1	65,536 µs

Note 8: Sampling intervals shown are when using an 8.0 MHz clock. The 256 corresponds to 2048/8 MHz; sample intervals must be scaled for other clock frequencies.

The trajectory control parameters which are written to the LM628 to control motion are: acceleration, velocity, and position. In addition, indications as to whether these three parameters are to be considered as absolute or relative inputs, selection of velocity mode and direction, and manual stopping mode selection and execution are programmable via this command. After writing the command code, the first two data bytes that are written specify which parameter(s) is/are being changed. The first byte written is the more significant. Thus the two data bytes constitute a trajectory control word that informs the LM628 as to the nature and number of any following data bytes. See *Table 6*.

TABLE 6. Trajectory Control Word Bit Allocation

Bit Position	Function
Bit 15	Not Used
Bit 14	Not Used
Bit 13	Not Used
Bit 12	Forward Direction (Velocity Mode Only)
Bit 11	Velocity Mode
Bit 10	Stop Smoothly (Decelerate as Programmed)
Bit 9	Stop Abruptly (Maximum Deceleration)
Bit 8	Turn Off Motor (Output Zero Drive)
Bit 7	Not Used
Bit 6	Not Used
Bit 5	Acceleration Will Be Loaded
Bit 4	Acceleration Data Is Relative
Bit 3	Velocity Will Be Loaded
Bit 2	Velocity Data Is Relative
Bit 1	Position Will Be Loaded
Bit 0	Position Data Is Relative

Bit 12 determines the motor direction when in the velocity mode. A logic one indicates forward direction. This bit has no effect when in position mode.

Bit 11 determines whether the LM628 operates in velocity mode (Bit 11 logic one) or position mode (Bit 11 logic zero).

Bits 8 through 10 are used to select the method of *manually stopping* the motor. These bits are *not* provided for one to merely specify the desired *mode* of stopping, in position mode operations, normal stopping is always smooth and occurs automatically at the end of the specified trajectory. Under exceptional circumstances it may be desired to manually intervene with the trajectory generation process to affect a premature stop. In velocity mode operations, however, the normal means of stopping *is* via bits 8 through 10 (usually bit 10). Bit 8 is set to logic one to stop the motor by turning off motor drive output (outputting the appropriate offset-binary code to apply zero drive to the motor); bit 9 is set to one to stop the motor abruptly (at maximum available acceleration, by setting the target position equal to the current position); and bit 10 is set to one to stop the motor smoothly by using

Trajectory Control Commands
(Continued)

the current user-programmed acceleration value. Bits 8 through 10 are to be used *exclusively;* only one bit should be a logic one at any time.

Bits 0 through 5 inform the LM628 as to whether any or all of the trajectory controlling parameters are about to be written, and whether the data should be interpreted as absolute or relative. The user may choose to update any or all (or none) of the trajectory parameters. Those chosen for updating are so indicated by logic one(s) in the corresponding bit position(s). Any parameter may be changed while the motor is in motion; however, if acceleration is changed then the next STT command must not be issued until the LM628 has completed the current move or has been manually stopped.

The data bytes specified by and immediately following the trajectory control word are written in pairs which comprise 16-bit words. Each data item (parameter) requires two 16-bit words; the word and byte order is most-to-least significant. The order of sending the parameters to the LM628 corresponds to the descending order shown in the above description of the trajectory control word; i.e., beginning with acceleration, then velocity, and finally position.

Acceleration and velocity are 32 bits, positive only, but range only from 0 (00000000 hex) to $[2^{30}]-1$ (3FFFFFFF hex). The bottom 16 bits of both acceleration and velocity are scaled as fractional data; therefore, the least-significant integer data bit for these parameters is bit 16 (where the bits are numbered 0 through 31). To determine the coding for a given velocity, for example, one multiplies the desired velocity (in counts per sample interval) times 65,536 and converts the result to binary. The units of acceleration are counts per sample per sample. The value loaded for acceleration must not exceed the value loaded for velocity. Position is a signed, 32-bit integer, but ranges only from $-[2^{30}]$ (C0000000 hex) to $[2^{30}]-1$ (3FFFFFFF Hex).

The required data is written to the primary buffers of a double-buffered scheme by the above described operations; it is not transferred to the secondary (working) registers until the STT command is executed. This fact can be used advantageously; the user can input numerous data ahead of their actual use. This simple pipeline effect can relieve potential host computer data communications bottlenecks, and facilitates easier synchronization of multiple-axis controls.

STT COMMAND: STarT Motion Control

Command Code:	01 Hex
Data Bytes:	None
Executable During Motion:	Yes, if acceleration has not been changed

The STT command is used to execute the desired trajectory, the specifics of which have been programmed via the LTRJ command. Synchronization of multi-axis control (to within one sample interval) can be arranged by loading the required trajectory parameters for each (and every) axis and then simultaneously issuing a single STT command to all axes. This command may be executed at any time, unless the acceleration value has been changed and a trajectory has not been completed or the motor has not been manually stopped. If STT is issued during motion and acceleration has been changed, a command error interrupt will be generated and the command will be ignored.

Data Reporting Commands

The following seven LM628 user commands are used to obtain data from various registers in the LM628. Status, position, and velocity information are reported. With the exception of RDSTAT, the data is read from the LM628 data port after first writing the corresponding command to the command port.

RDSTAT COMMAND: ReaD STATus Byte

Command Code:	None
Byte Read:	One
Data Range:	See Text
Executable During Motion:	Yes

The RDSTAT command is really not a command, but is listed with the other commands because it is used very frequently to control communications with the host computer. There is no identification code; it is directly supported by the hardware and may be executed at any time. The single-byte status read is selected by placing $\overline{CS}$, $\overline{PS}$ and $\overline{RD}$ at logic zero. See *Table 7.*

TABLE 7. Status Byte Bit Allocation

Bit Position	Function
Bit 7	Motor Off
Bit 6	Breakpoint Reached [Interrupt]
Bit 5	Excessive Position Error [Interrupt]
Bit 4	Wraparound Occurred [Interrupt]
Bit 3	Index Pulse Observed [Interrupt]
Bit 2	Trajectory Complete [Interrupt]
Bit 1	Command Error [Interrupt]
Bit 0	Busy Bit

Bit 7, the motor-off flag, is set to logic one when the motor drive output is off (at the half-scale, offset-binary code for zero). The motor is turned off by any of the following conditions: power-up reset, command RESET, excessive position error (if command LPES had been executed), or when command LTRJ is used to manually stop the motor via turning the motor off. Note that when bit 7 is set in conjunction with command LTRJ for producing a manual, motor-off stop, the actual setting of bit 7 does not occur until command STT is issued to affect the stop. Bit 7 is cleared by command STT, except as described in the previous sentence.

Bit 6, the breakpoint-reached interrupt flag, is set to logic one when the position breakpoint loaded via command SBPA or SBPR has been exceeded. The flag is functional independent of the host interrupt mask status. Bit 6 is cleared via command RSTI.

Bit 5, the excessive-position-error interrupt flag, is set to logic one when a position-error interrupt condition exists. This occurs when the error threshold loaded via command LPEI or LPES has been exceeded. The flag is functional independent of the host interrupt mask status. Bit 5 is cleared via command RSTI.

Bit 4, the wraparound interrupt flag, is set to logic one when a numerical "wraparound" has occurred. To "wraparound" means to exceed the position address space of the LM628, which could occur during velocity mode operation. If a wraparound has occurred, then position information will be in error and this interrupt helps the user to ensure position data integrity. The flag is functional independent of the host interrupt mask status. Bit 4 is cleared via command RSTI.

Data Reporting Commands (Continued)

Bit 3, the index-pulse acquired interrupt flag, is set to logic one when an index pulse has occurred (if command SIP had been executed) and indicates that the index position register has been updated. The flag is functional independent of the host interrupt mask status. Bit 3 is cleared by command RSTI.

Bit 2, the trajectory complete interrupt flag, is set to logic one when the trajectory programmed by the LTRJ command and initiated by the STT command has been completed. Because of overshoot or a limiting condition (such as commanding the velocity to be higher than the motor can achieve), the motor may not yet be at the final commanded position. This bit is the logical OR of bits 7 and 10 of the Signals Register, see command RDSIGS below. The flag functions independently of the host interrupt mask status. Bit 2 is cleared via command RSTI.

Bit 1, the command-error interrupt flag, is set to logic one when the user attempts to read data when a write was appropriate (or vice versa). The flag is functional independent of the host interrupt mask status. Bit 1 is cleared via command RSTI.

Bit 0, the busy flag, is frequently tested by the user (via the host computer program) to determine the busy/ready status prior to writing and reading any data. Such writes and reads may be executed only when bit 0 is logic zero (not busy). Any command or data writes when the busy bit is high will be ignored. Any data reads when the busy bit is high will read the current contents of the I/O port buffers, not the data expected by the host. Such reads or writes (with the busy bit high) will not generate a command-error interrupt.

RDSIGS COMMAND: ReaD SIGnalS Register

Command Code:	0C Hex
Bytes Read:	Two
Data Range:	See Text
Executable During Motion:	Yes

The LM628 internal "signals" register may be read using this command. The first byte read is the more significant. The less significant byte of this register (with the exception of bit 0) duplicates the status byte. See *Table 8*.

TABLE 8. Signals Register Bit Allocation

Bit Position	Function
Bit 15	Host Interrupt
Bit 14	Acceleration Loaded (But Not Updated)
Bit 13	UDF Executed (But Filter Not yet Updated)
Bit 12	Forward Direction
Bit 11	Velocity Mode
Bit 10	On Target
Bit 9	Turn Off upon Excessive Position Error
Bit 8	Eight-Bit Output Mode
Bit 7	Motor Off
Bit 6	Breakpoint Reached [Interrupt]
Bit 5	Excessive Position Error [Interrupt]
Bit 4	Wraparound Occurred [Interrupt]
Bit 3	Index Pulse Acquired [Interrupt]
Bit 2	Trajectory Complete [Interrupt]
Bit 1	Command Error [Interrupt]
Bit 0	Acquire Next Index (SIP Executed)

Bit 15, the host interrupt flag, is set to logic one when the host interrupt output (Pin 17) is logic one. Pin 17 is set to logic one when any of the six host interrupt conditions occur (if the corresponding interrupt has not been masked). Bit 15 (and Pin 17) are cleared via command RSTI.

Bit 14, the acceleration-loaded flag, is set to logic one when acceleration data is written to the LM628. Bit 14 is cleared by the STT command.

Bit 13, the UDF-executed flag, is set to logic one when the UDF command is executed. Because bit 13 is cleared at the end of the sampling interval in which it has been set, this signal is very short-lived and probably not very profitable for monitoring.

Bit 12, the forward direction flag, is meaningful only when the LM628 is in velocity mode. The bit is set to logic one to indicate that the desired direction of motion is "forward"; zero indicates "reverse" direction. Bit 12 is set and cleared via command LTRJ. The actual setting and clearing of bit 12 does not occur until command STT is executed.

Bit 11, the velocity mode flag, is set to logic one to indicate that the user has selected (via command LTRJ) velocity mode. Bit 11 is cleared when position mode is selected (via command LTRJ). The actual setting and clearing of bit 11 does not occur until command STT is executed.

Bit 10, the on-target flag, is set to logic one when the trajectory generator has completed its functions for the last-issued STT command. Bit 10 is cleared by the next STT command.

Bit 9, the turn-off on-error flag, is set to logic one when command LPES is executed. Bit 9 is cleared by command LPEI.

Bit 8, the 8-bit output flag, is set to logic one when the LM628 is reset, or when command PORT8 is executed. Bit 8 is cleared by command PORT12.

Bits 0 through 7 replicate the status byte (see), with the exception of bit 0. Bit 0, the acquire next index flag, is set to logic one when command SIP is executed; it then remains set until the next index pulse occurs.

RDIP COMMAND: ReaD Index Position

Command Code:	09 Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the position recorded in the index register. Reading the index register can be part of a system error checking scheme. Whenever the SIP command is executed, the new index position minus the old index position, divided by the incremental encoder resolution (encoder lines times four), should always be an integral number. The RDIP command facilitates acquiring these data for host-based calculations. The command can also be used to identify/verify home or some other special position. The bytes are read in most-to-least significant order.

RDDP COMMAND: ReaD Desired Position

Command Code:	08 Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

Data Reporting Commands (Continued)

This command reads the instantaneous desired (current *temporal*) position output of the profile generator. This is the "setpoint" input to the position-loop summing junction. The bytes are read in most-to-least significant order.

RDRP COMMAND: ReaD Real Position

Command Code:	0A Hex
Bytes Read:	Four
Data Range:	C0000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

This command reads the current actual position of the motor. This is the feedback input to the loop summing junction. The bytes are read in most-to-least significant order.

RDDV COMMAND: ReaD Desired Velocity

Command Code:	07 Hex
Bytes Read:	Four
Data Range:	C0000001 to 3FFFFFFF
Executable During Motion:	Yes

This command reads the integer and fractional portions of the instantaneous desired (current *temporal*) velocity, as used to generate the desired position profile. The bytes are read in most-to-least significant order. The value read is properly scaled for numerical comparison with the user-supplied (commanded) velocity; however, because the two least-significant bytes represent *fractional* velocity, only the two most-significant bytes are appropriate for comparison with the data obtained via command RDRV (see below). Also note that, although the velocity *input* data is constrained to positive numbers (see command LTRJ), the data returned by command RDDV represents a *signed* quantity where negative numbers represent operation in the reverse direction.

RDRV COMMAND: ReaD Real Velocity

Command Code:	0B Hex
Bytes Read:	Two
Data Range:	C000 to 3FFF Hex, See Text
Executable During Motion:	Yes

This command reads the *integer* portion of the instantaneous actual velocity of the motor. The internally maintained fractional portion of velocity is not reported because the reported data is derived by reading the incremental encoder, which produces only integer data. For comparison with the result obtained by executing command RDDV (or the user-supplied input value), the value returned by command RDRV must be multiplied by 2^{16} (shifted left 16 bit positions). Also, as with command RDDV above, data returned by command RDRV is a *signed* quantity, with negative values representing reverse-direction motion.

RDSUM COMMAND: ReaD Integration-Term SUMmation Value

Command Code:	0D Hex
Bytes Read:	Two
Data Range:	00000 Hex to ± the Current Value of the Integration Limit
Executable During Motion:	Yes

This command reads the value to which the integration term has accumulated. The ability to read this value may be helpful in initially or adaptively tuning the system.

Typical Applications

Programming LM628 Host Handshaking (Interrupts)

A few words regarding the LM628 host handshaking will be helpful to the system programmer. As indicated in various portions of the above text, the LM628 handshakes with the host computer in two ways: via the host interrupt output (Pin 17), or via polling the status byte for "interrupt" conditions. When the hardwired interrupt is used, the status byte is also read and parsed to determine which of six possible conditions caused the interrupt.

When using the hardwired interrupt it is very important that the host interrupt service routine does not interfere with a command sequence which might have been in progress when the interrupt occurred. If the host interrupt service routine were to issue a command to the LM628 while it is in the middle of an ongoing command sequence, the ongoing command will be aborted (which could be detrimental to the application).

Two approaches exist for avoiding this problem. If one is using hardwired interrupts, they should be disabled at the host prior to issuing any LM628 command sequence, and re-enabled after each command sequence. The second approach is to avoid hardwired interrupts and poll the LM628 status byte for "interrupt" status. The status byte always reflects the interrupt-condition status, independent of whether or not the interrupts have been masked.

Typical Host Computer/Processor Interface

The LM628 is interfaced with the host computer/processor via an 8-bit parallel bus. *Figure 12* shows such an interface and a minimum system configuration.

As shown in *Figure 12*, the LM628 interfaces with the host data, address and control lines. The address lines are decoded to generate the LM628 $\overline{CS}$ input; the host address LSB directly drives the LM628 $\overline{PS}$ input. *Figure 12* also shows an 8-bit DAC and an LM12 Power Op Amp interfaced to the LM628.

LM628 and High Performance Controller (HPC) Interface

Figure 13 shows the LM628 interfaced to a National HPC High Performance Controller. The delay and logic associated with the $\overline{WR}$ line is used to effectively increase the write-data hold time of the HPC (as seen at the LM628) by causing the $\overline{WR}$ pulse to rise early. Note that the HPC CK2 output provides the clock for the LM628. The 74LS245 is used to decrease the read-data hold time, which is necessary when interfacing to fast host busses.

Interfacing a 12-Bit DAC

Figure 14 illustrates use of a 12-bit DAC with the LM628. The 74LS378 hex gated-D flip-flop and an inverter demultiplex the 12-bit output. DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. Two methods exist for making this adjustment. If the DAC1210 has been socketed, remove it and temporarily connect a 15 kΩ resistor between Pins 11 and 13 of the DAC socket (Pins 2 and 6 of the LF356) and adjust the 25 kΩ potentiometer for 0V at Pin 6 of the LF356.

If the DAC is not removable, the second method of adjustment requires that the DAC1210 inputs be presented an all-zeros code. This can be arranged by commanding the ap-

Typical Applications (Continued)

propriate move via the LM628, but with no feedback from the system encoder. When the all-zeros code is present, adjust the pot for 0V at Pin 6 of the LF356.

A Monolithic Linear Drive Using LM12 Power Op Amp

Figure 15 shows a motor-drive amplifier built using the LM12 Power Operational Amplifier. This circuit is very simple and can deliver up to 8A at 30V (using the LM12L/LM12CL). Resistors R1 and R2 should be chosen to set the gain to provide maximum output voltage consistent with maximum input voltage. This example provides a gain of 2.2, which allows for amplifier output saturation at ±22V with a ±10V input, assuming power supply voltages of ±30V. The amplifier gain should not be higher than necessary because the system is non-linear when saturated, and because gain should be controlled by the LM628. The LM12 can also be configured as a current driver, see 1987 Linear Databook, Vol. 1, p. 2-280.

Typical PWM Motor Drive Interfaces

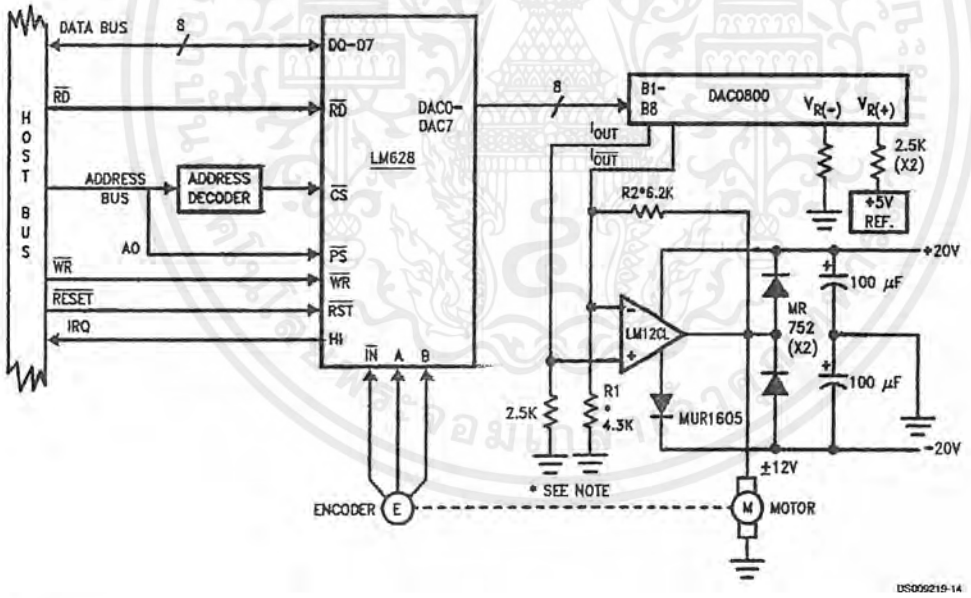
Figure 16 shows an LM18298 dual full-bridge driver interfaced to the LM629 PWM outputs to provide a switch-mode power amplifier for driving small brush/commutator motors.

Incremental Encoder Interface

The incremental (position feedback) encoder interface consists of three lines: Phase A (Pin 2), Phase B (Pin 3), and Index (Pin 1). The index pulse output is not available on some encoders. The LM628 will work with both encoder types, but commands SIP and RDIP will not be meaningful without an index pulse (or alternative input for this input ... be sure to tie Pin 1 high if not used).

Some consideration is merited relative to use in high Gaussian-noise environments. If noise is added to the encoder inputs (either or both inputs) and is such that it is not sustained until the next encoder transition, the LM628 decoder logic will reject it. Noise that mimics quadrature counts or persists through encoder transitions must be eliminated by appropriate EMI design.

Simple digital "filtering" schemes merely reduce susceptibility to noise (there will always be noise pulses longer than the filter can eliminate). Further, any noise filtering scheme reduces decoder bandwidth. In the LM628 it was decided (since simple filtering does not eliminate the noise problem) to not include a noise filter in favor of offering maximum possible decoder bandwidth. Attempting to drive encoder signals too long a distance with simple TTL lines can also be a source of "noise" in the form of signal degradation (poor rise-time and/or ringing). This can also cause a system to lose positional integrity. Probably the most effective countermeasure to noise induction can be had by using balanced-line drivers and receivers on the encoder inputs. *Figure 17* shows circuitry using the DS26LS31 and DS26LS32.



Note:

$$A_V = \frac{R1 + R2}{R1} \simeq 2.4$$

$$\frac{R1 \times R2}{R1 + R2} \triangleq 2.5k$$

FIGURE 12. Host Interface and Minimum System Configuration

Typical Applications (Continued)

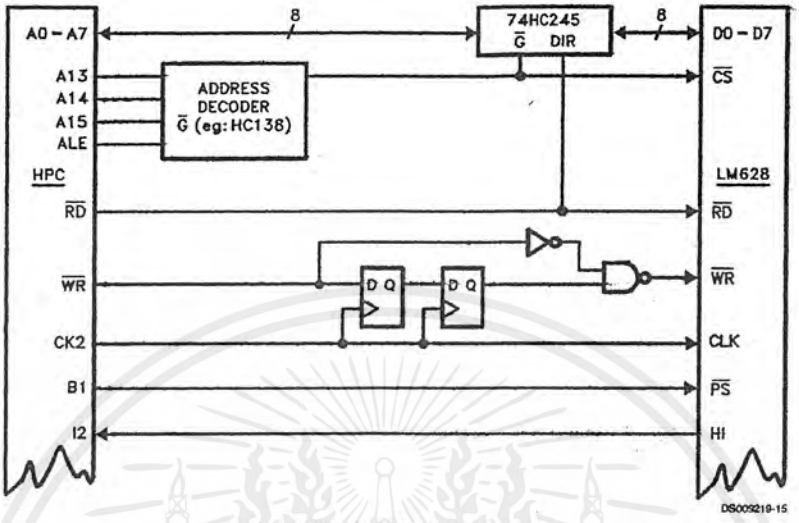
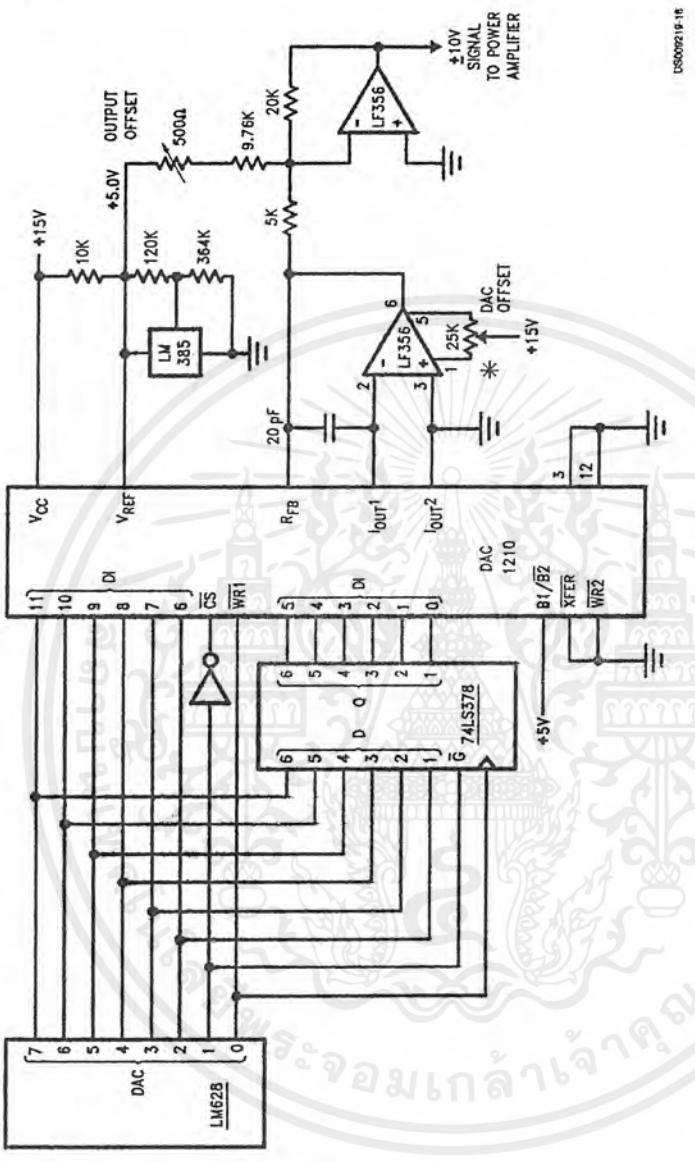


FIGURE 13. LM628 and HPC Interface

Typical Applications (Continued)



*DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. See text.

FIGURE 14. Interfacing a 12-Bit DAC and LM628

Typical Applications (Continued)

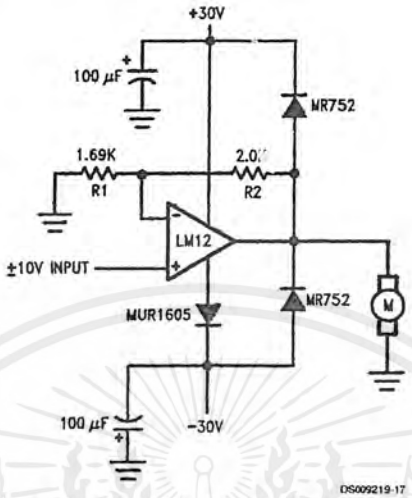


FIGURE 15. Driving a Motor with the LM12 Power Op Amp

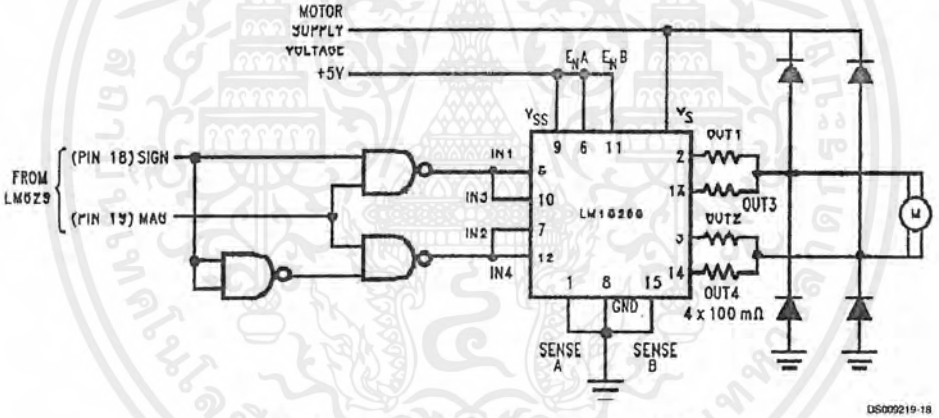


FIGURE 16. PWM Drive for Brush/Commutator Motors

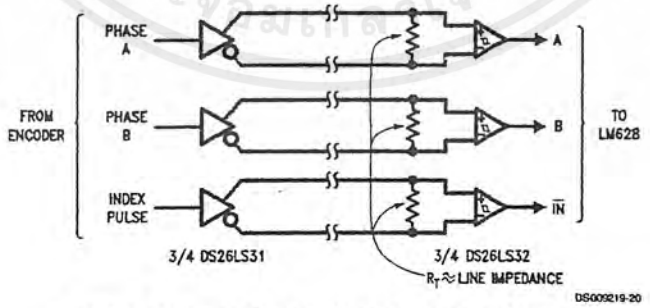
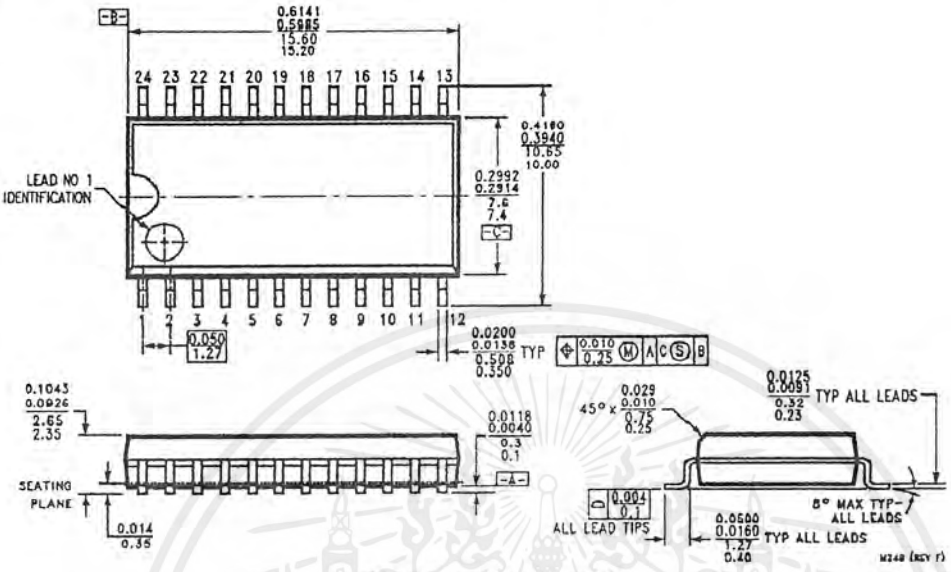
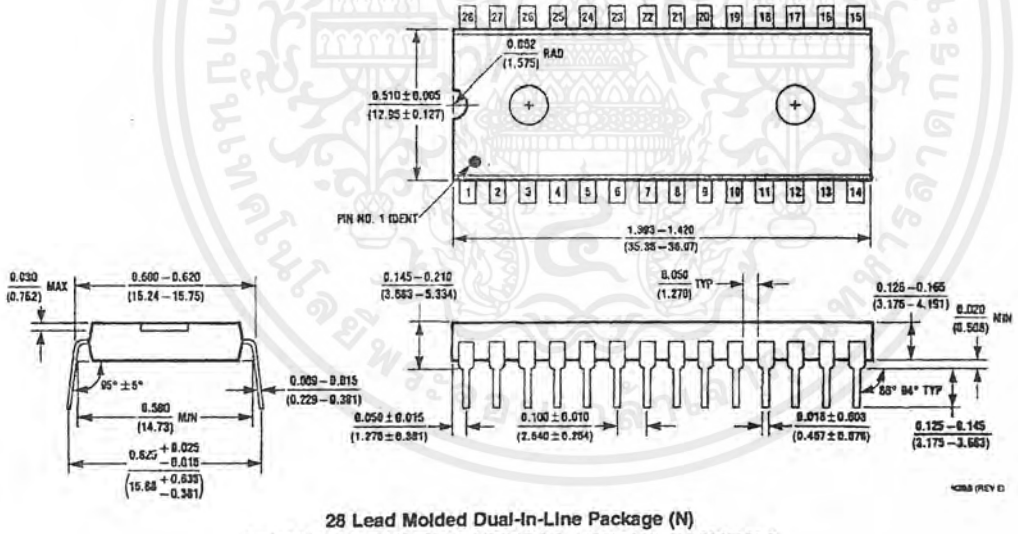


FIGURE 17. Typical Balanced-Line Encoder Input Circuit

Physical Dimensions inches (millimeters) unless otherwise noted



24-Lead Small Outline Package (M)
Order Number LM629M-6 or LM629M-8
NS Package Number M24B



28 Lead Molded Dual-In-Line Package (N)
Order Number LM628N-6, LM628N-8, LM629N-6 or LM629N-8
NS Package Number N28B

Notes

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

www.national.com

National Semiconductor Europe
Fax: +49 (0) 1 80-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 1 80-530 85 85
English Tel: +49 (0) 1 80-532 78 32
Français Tel: +49 (0) 1 80-532 93 58
Italiano Tel: +49 (0) 1 80-534 16 80

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: sea.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

เอกสารอ้างอิง

1. Introduction to Robotics , Phillip John Mckerrow
2. Understanding Robotics , V.Daniel Hunt
3. Robotics Research , Georges Giralt and Gerhard Hirzinger (Eds)
4. Robotics A Manager 's Guide ,Rax Maus , Randall Allsup
5. http://www.national.com/pf/LM/LM629.html#GeneralDescription
6. เครื่องจักรกลไฟฟ้า 1 , พิชิต ลำยอง
7. กลศาสตร์ของแข็งชั้นสูง , สมชัย นรเศรษฐโสภณ