ระบบนำร่องอัตโนมัติ

MOBILE ROBOT

นายภักดี อัญญะกมล

นายอรรถพล วิทยกฤตศิริกุล

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

ระบบนำร่องอัตโนมัติ

MOBILE ROBOT

โดย

นายภักดี อัญญะกมล

นายอรรถพล วิทยกฤตศิริกุล

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร. โยธิน เปรมปราณีรัชต์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

ปริญญานิพนธ์

ภาคการเรียนที่ 2 ปีการศึกษา 2538

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบน~

MOBILE ROBOT

โดย นายภักดี อัญญะกมล 35104312

นายอรรถพล วิทยกฤตศิริกุล 35104540

.....

(รองศาสตราจารย์ ดร. โยธิน เปรมปราณีรัชต์) อาจารย์ที่ปรึกษา

สารบัญ

หน้า

ระบบนำร่องอัตโนมัติ

MOBILE ROBOT

โดย

นายภักดี อัญญะกมล

นายอรรถพล วิทยกฤตศิริกุล

อาจารย์ที่ปรึกษา

- รองศาสตราจารย์ ดร. โยธิน เปรมปราณีรัชต์

บทคัดย่อ

ในปริญญานิพนธ์ฉบับนี้กล่าวถึงการทำงานของระบบนำร่องอัตโนมัติ ตัวยานขนส่งจะวิ่งไปตามเส้นแถบสีเงินที่ติดอยู่กับพื้นโดยมีเซ็นเซอร์อินฟราเรดที่ติดอยู่กับตัวยานเป็นตัวตรวจจับใช้แบตเตอรี่ชนิดตะกั่ว-กรดและขับเคลื่อนด้วยมอเตอร์กระแสตรง การประมวลผลโปรแกรมการควบคุมการเคลื่อนที่ทำได้โดยผ่านบอร์ด 8032 และชิพ LM 629 ที่ติดอยู่บนตัวยาน

ABSTRACT

In this thesis it 's told about how this automatic guided vehicle works. The automatic guided vehicle is led by the bright gray strip which lay down on the floor. It can be detected by the infrared sensor. We use lead - acid battery and drive this vehicle by DC motor. ETT board 8032 and LM 629 are our tools for controlling the motion of this automatic guided vehicle.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดี ก็เพราะได้รับความเมตตาจาก รองศาสตราจารย์ ดร. โยธิน เปรมปราณีรัชต์ และอาจารย์สุมิตร ที่ได้ให้ความกรุณาแนะนำพร้อมทั้งให้ความช่วยเหลือแก่คณะผู้จัดทำตลอดมา ทางคณะผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ และขอขอบคุณครอบครัวของคณะผู้จัดทำที่ได้ให้กำลังใจมาตลอด นอกจากนี้ขอขอบคุณ คุณนิรุดต์ นาคสุข ที่ให้คำแนะนำมาตลอด สุดท้ายนี้ขอขอบคุณเพื่อน ๆ ทุกคนทั้งในภาควิชาและนอกภาควิชาที่ให้ความช่วยเหลือตลอดมา

คณะผู้จัดทำ

สารบัญรูปภาพ

รูปที่ หน้า

บทนำ

ยานขนส่งระบบนำร่องอัตโนมัติ หรือ AGV จัดเป็นหุ่นยนต์เคลื่อนที่ชนิดหนึ่งซึ่งมีใช้อย่างแพร่หลายในอุตสาหกรรมมากกว่า 30 ปี AGV เป็นยานพาหนะที่ไม่ต้องใช้คนขับที่สามารถวิ่งไปตามเส้นทางที่กำหนดได้เองโดยอัตโนมัติ AGV สามารถใช้แทนยานพาหนะอาทิ เช่น รถบรรทุก หรือใช้แทนระบบสายพานลำเลียง AGV มีหน้าที่ขนส่งวัตถุดิบ ชิ้นส่วน ผลิตภัณฑ์หรือสินค้าจากสถานที่หนึ่งไปยังอีกที่หนึ่ง

AGV ในปัจจุบันได้รับการออกแบบเพื่อให้สามารถทำงานภายใต้การควบคุมจากคอมพิวเตอร์ส่วนกลาง โดยสามารถโปรแกรมให้ AGV วิ่งไปตามเส้นทางและหยุดยังตำแหน่งเป้าหมายได้อย่างถูกต้องแม่นยำ AGV ส่วนใหญ่จะถูกกำหนดให้เคลื่อนที่ไปตามทางเดินนำร่องซึ่งติดตั้งอยู่ตามพื้น ทางเดินนำร่องที่นิยมใช้อาทิเช่น การฝังสายตัวนำใต้พื้น , การใช้แถบสีติดบนพื้น เป็นต้น โครงสร้างภายนอกสำหรับ AGV ส่วนมากจะทำเป็นแท่นไว้วางภาชนะที่ใส่วัสดุอาทิเช่น ถาด , กล่อง หรือชั้นวางของ

ตามปกติด้านหน้าและด้านหลังของ AGV จะติดบัมเปอร์เพื่อความปลอดภัย AGV ส่วนมากจะใช้แบตเตอรี่ชนิดตะกั่ว - กรด และขับเคลื่อนด้วยมอเตอร์กระแสตรง

ข้อดีของระบบ AGV คือไม่เกะกะ , การติดตั้งทางเดินนำร่องมีความยืดหยุ่นสามารถดัดแปลงแก้ไขได้ง่าย ส่วนข้อเสียจะเกิดขึ้นในกรณีที่ทิศทางการขนส่งวัสดุมีความแน่นอน , ปริมาณการขนส่งมีจำนวนมาก ตลอดจนการขนส่งเป็นไปอย่างต่อเนื่องตลอดเวลา ทำให้ต้องใช้ AGV จำนวนมากไปทำในงานลักษณะเดียวกัน ซึ่งเป็นการสิ้นเปลืองเกินไป

นอกจากในโรงงานอุตสาหกรรมแล้ว AGV ยังสามารถนำไปประยุกต์ใช้ในงานด้านอื่น ๆ อาทิเช่น การขนสัมภาระไปยังห้องต่าง ๆ ในโรงแรม , การรับส่งเอกสารในสำนักงาน , การบริการอาหารตามโรงแรมและภัตตาคาร หรือการขนส่งวัสดุในสถานที่ที่เป็นความลับทางราชการ

ทฤษฎีการควบคุมการทำงานของ AGV นั้นจะแบ่งเป็นส่วนต่าง ๆ ได้ดังนี้

- ทฤษฎีการสื่อสารข้อมูลระหว่าง AGV กับคอมพิวเตอร์ส่วนกลาง
- ทฤษฎีการนำร่อง
- ทฤษฎีการควบคุมการเคลื่อนที่และการเลี้ยว
- ทฤษฎีอื่น ๆ เช่น ใช้คลื่นอุลตร้าโซนิคตรวจจับหาสิ่งกีดขวาง หรือใช้วงจรตรวจจับแรงดันแบตเตอรี่เพื่อเพิ่มเสถียรภาพการทำงาน

ทฤษฎีการสื่อสารข้อมูลระหว่าง AGV กับคอมพิวเตอร์ส่วนกลาง

จะทำการส่งข้อมูลผ่านแสงอินฟราเรดด้วยความเร็วค่อนข้างสูง แต่มีข้อจำกัดตรงที่การส่งต้องเป็นแนวตรงเท่านั้น ต่อมาได้มีการพัฒนาการสื่อสารข้อมูลด้วยคลื่นวิทยุ ซึ่งมีปัญหาตรงที่อาจมีสัญญาณรบกวนได้ นอกจากนั้นยังมีข้อจำกัดทางด้านกฎหมายอีกด้วย ตามปกติลักษณะข้อมูลที่คอมพิวเตอร์ส่วนกลางกับ AGV สื่อสารกันประกอบด้วย

- คำสั่งการขับเคลื่อน
- ข้อมูลทางตำแหน่ง

ทฤษฎีการนำร่อง

มีหลายวิธี ดังนี้

1) การนำร่องโดยใช้ทางเดินนำร่อง

1.1) การฝังสายตัวนำใต้พื้น นิยมใช้กันมาก ในการติดตั้งต้องมีการเซาะร่องตามพื้นแล้วฝังสายตัวนำลึกประมาณ 0.5 นิ้ว แล้วปูกระเบื้องยางพร้อมลงอีพอกซี การใช้งานจะเริ่มโดยป้อนไฟสลับความถี่สูงแก่ตัวนำเพื่อให้เกิดการเหนี่ยวนำสนามแม่เหล็ก แรงดันที่ใช้ประมาณ 40 โวลต์ กระแสประมาณ 400 มิลลิแอมป์ และความถี่ที่ใช้อยู่ในช่วง 1 - 15 กิโลเฮิรตซ์ AGV จะมีขดลวด 2 ขดคอยตรวจจับสนามแม่เหล็กที่เหนี่ยวนำจากสายตัวนำ กรณีที่ AGV วิ่งอยู่ตรงแนวฝังสายตัวนำพอดี ค่าสนามแม่เหล็กที่วัดได้จากขดทั้งสองจะมีค่าเท่ากัน แต่ถ้า AGV วิ่งเบี่ยงเบนไปจากแนวสายตัวนำ สนามแม่เหล็กที่เกิดขึ้นที่ขดลวดทั้งสองจะมีค่าไม่เท่ากัน ระบบควบคุมก็จะส่งสัญญาณไปปรับแต่งทิศทางการวิ่งของ AGV ให้ตรงแนวสายตัวนำ ข้อดีของวิธีนี้คือมีความเชื่อถือได้สูง ทนทาน ส่วนข้อเสียคือ ระบบไม่ยืดหยุ่น การดัดแปลงแก้ไขทำได้ยาก และค่าใช้จ่ายในการติดตั้งสูงโดยเฉพาะถ้าต้องขนส่งในระยะไกล

1.2) ใช้แถบสีหรือเทปสะท้อนแสงติดบนพื้น AGV จะคอยตรวจจับหาแนวเส้นทางของแถบสี ข้อดีของวิธีนี้คือระบบนำร่องไม่ซับซ้อน การเปลี่ยนแปลงแก้ไขทำได้ง่าย ตลอดจนค่าใช้จ่ายในการติดตั้งถูก เทคนิคในการตรวจจับแถบสีมี 2 วิธี ได้แก่

ก) ใช้โฟโต้เซนเซอร์ ตรวจจับความเข้มแสงที่สะท้อนกลับขึ้นมา ข้อดีของการใช้โฟโต้เซนเซอร์คือสร้างง่ายและราคาถูก ส่วนข้อเสียคือการใช้โฟโต้เซนเซอร์นั้นอ่อนไหวต่อคุณภาพของแถบสีมาก อีกทั้งแถบสีเลอะเลือนและชำรุดเสียหายได้ง่าย นอกจากนี้ความสะอาดของพื้นก็มีผลต่อการตรวจจับเช่นกัน

ข) ใช้กล้องทีวี โดยนำสัญญาณภาพที่ได้รับมาทำการประมวลผลเพื่อหาทางเดินของแถบสีบนพื้น มีข้อดีคือสามารถหาตำแหน่งแถบสีได้ แม้ว่าแถบสีนั้นจะมีการเลอะเลือนหรือชำรุดเสียหายไปบ้างก็ตาม ทำให้สามารถนำไปใช้ขนส่งภายนอกอาคารได้ แต่ข้อเสียก็คือความเร็วในการประมวลผลภาพช้าเกินไป

1.3) ใช้แถบโลหะ วางเป็นแนวแบบเดียวกับแถบสี แล้วใช้พร็อคซิมิตี้เซ็นเซอร์เป็นตัวตรวจจับตำแหน่งแถบโลหะนั้น

1.4) ใช้แถบแม่เหล็ก ฝังลงในพื้นเป็นลักษณะตารางทั่ว ๆ พื้นที่ AGV จะตรวจจับแถบแม่เหล็กด้วยแมกเนติคเซ็นเซอร์ โดย AGV จะเคลื่อนไปตามแนวขอบตาราง จุดตัดของเส้นตารางจะเป็นตัวนับตำแหน่งในการเคลื่อนที่ ลักษณะแมกเนติคเซ็นเซอร์ประกอบด้วย exciting coil 1 ชุด , detecting coil 2 ชุด exciting coil จะผลิตสนามแม่เหล็กจากไฟสลับโดยมี detecting coil คอยตรวจจับสนามแม่เหล็ก กรณีที่ AGV อยู่ตรงแนวแถบแม่เหล็ก สนามแม่เหล็กที่ตรวจจับได้จะมีค่ามากที่สุด

2) การนำร่องแบบไร้สาย เป็นการนำร่องโดยไม่ต้องติดตั้งทางเดินนำร่องตามพื้น AGV สามารถเคลื่อนที่ได้อย่างอิสระ ซึ่งมีได้หลายวิธี ดังนี้

2.1) การนำทางด้วยคลื่นวิทยุ วิธีนี้สามารถหาตำแหน่งของวัตถุที่เคลื่อนที่ในห้องโดยประมาณ โดยมีขอบเขตของการตรวจจับตั้งแต่หลายร้อยเมตรจนถึงหลายกิโลเมตร วิธีมีข้อเสียคือเหล็กจะทำให้คลื่นวิทยุเกิดการสะท้อนและเบี่ยงเบนไป ทำให้การหาตำแหน่งขาดความถูกต้องแม่นยำ โดยเฉพาะหากขอบเขตของการตรวจจับต่ำกว่า 100 เมตร

2.2) การนำทางโดยอาศัยระบบดาวเทียม สามารถหาตำแหน่งบนพื้นโลกได้โดยการวัดรันนิ่งไทม์ของสัญญาณดาวเทียม แต่ความละเอียดสูงสุดของเทคนิคนี้คือ 10 เมตร

2.3) การใช้เลเซอร์วัดระยะทาง ใช้หลักการวัดรันนิ่งไทม์ของสัญญาณและวัดการรบกวนผลลัพธ์และความละเอียดของการวัดเวลาเดินทางจะอยู่ในหน่วยเซนติเมตร ส่วนการวัดการรบกวนจะอยู่ในหน่วยมิลลิเมตร ข้อเสียของวิธีนี้คือใช้ได้ในบางกรณีเท่านั้น เพราะระบบการขนส่งหลายแห่งเป็นบริเวณเปิดโล่งหรือมีกำแพงซึ่งอยู่ไกลเกินกว่าจะตรวจจับได้ นอกจากนี้เลเซอร์จะใช้ไม่ได้กับบริเวณที่มีผู้คนและยานพาหนะหนาแน่น

2.4) การประมวลผลภาพ หาตำแหน่ง AGV โดยใช้กล้องติดบนเพดานของห้องโถง กล้องที่ใช้อาจจะเป็นแบบ tube หรือ ccd ในทางการค้ากล้อง ccd มีความละเอียดประมาณ 500 pixel สัญญาณภาพที่ได้จะถูกนำไปประมวลผล

ทฤษฎีการขับเคลื่อนและการเลี้ยว

AGV ส่วนใหญ่ใช้พลังงานในการขับเคลื่อนจากแบตเตอรี่ชนิดตะกั่ว - กรด และขับเคลื่อนโดยใช้มอเตอร์กระแสตรงซึ่งมีการควบคุมทิศทางการหมุน , ความเร็วและตำแหน่งโดยใช้คอมพิวเตอร์ซึ่งติดตั้งอยู่บน AGV สำหรับเทคนิคการเลี้ยวของ AGV นั้นจะขึ้นกับโครงสร้างของระบบล้อ ระบบล้อตามปกติที่ใช้กันอาจเป็นระบบ 3 ล้อ หรือระบบ 4 ล้อก็ได้ เมื่อเปรียบเทียบระบบ 4 ล้อ กับระบบ 3 ล้อแล้วระบบ 4 ล้อจะมีเสถียรภาพดีกว่า , กำลังลากจูงดีกว่า , ช่องว่างทางกลน้อยกว่า แต่ระบบ 3 ล้อ มีข้อดีตรงที่ระบบเลี้ยวไม่ยุ่งยากซับซ้อน ,โครงรถมีน้ำหนักเบากว่าซึ่งทำให้ประหยัดพลังงานจากแบตเตอรี่ได้มากกว่า โครงสร้างของระบบล้อของ AGV มีหลายแบบ เช่น

- ระบบ 3 ล้อ เลี้ยวโดยล้อหน้า 1 ล้อ โดยให้ล้อหลัง 2 ล้อเป็นล้อขับ
- ระบบ 3 ล้อ เลี้ยวโดย 2 ล้อหลัง ส่วนล้อหน้าเป็นล้อขับ
- ระบบ 3 ล้อ ล้อหน้าเป็นทั้งล้อขับและเลี้ยว ส่วน 2 ล้อหลังเป็นล้อช่วย
- ระบบ 3 ล้อ เลี้ยวโดยใช้หลักการหมุนในทิศตรงข้ามกันของล้อหลัง 2 ล้อ ส่วนล้อหน้าก็สามารถเลี้ยวได้ด้วย
- ระบบ 3 ล้อ ล้อทุกล้อเป็นล้อขับที่สามารถเลี้ยวได้โดยอิสระ
- ระบบ 4 ล้อ เลี้ยวโดยใช้หลักการหมุนในทิศตรงกันข้ามของ 2 ล้อขับ อีก 2 ล้อเป็นล้อช่วย
- ระบบ 4 ล้อ ขับเคลื่อนได้ทั้ง 4 ล้อ และล้อแต่ละคู่ก็สามารถเลี้ยวได้ด้วย
- ระบบ 4 ล้อ 2 ล้อหน้าเป็นทั้งล้อขับและเลี้ยว ส่วน 2 ล้อหลังเป็นล้อช่วย
- ระบบ 4 ล้อ 2 ล้อหน้าสำหรับเลี้ยว ส่วน 2 ล้อหลังเป็นล้อขับ
- ระบบ 4 ล้อ 2 ล้อหลังเป็นทั้งล้อขับและเลี้ยว ส่วน 2 ล้อหน้าเป็นล้อช่วย ในปริญญานิพนธ์ฉบับนี้เลือกใช้ระบบนี้เป็นกรณีศึกษาสำหรับระบบ AGV

เทคโนโลยีอื่น ๆ

ในการใช้ AGV ตามโรงงานหรือสถานที่ต่าง ๆ ปัญหาที่ต้องพบคือ วัตถุกีดขวาง จึงได้มีการพัฒนาเทคโนโลยีในการตรวจจับวัตถุกีดขวาง โดยมีหลายวิธีเช่น การใช้คลื่นอุลตร้าโซนิค , การใช้แสงอินฟราเรด และการใช้กล้องทีวี เป็นต้น

นอกจากนี้เนื่องจาก AGV ใช้พลังงานจากแบตเตอรี่ จึงต้องมีระบบตรวจจับแรงดันแบตเตอรี่เพื่อให้การทำงานเป็นไปอย่างต่อเนื่อง

จุดประสงค์ของปริญญานิพนธ์ฉบับนี้ ก็เพื่อให้เป็นพื้นฐานสำหรับคนที่ต้องการเรียนรู้และนำไปพัฒนาประยุกต์ใช้ให้เหมาะสมกับงานที่จะนำ AGV ไปใช้

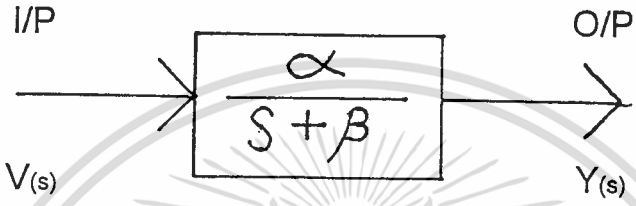
เนื้อหาของปริญญานิพนธ์ฉบับนี้จะกล่าวถึง

- ลักษณะการติดต่อกับบอร์ด 8032
- วงจรขับมอเตอร์โดยใช้ชิพ LM 629
- วิธีการหาสมการมอเตอร์ รวมทั้งหาค่า PI-Controller
- ระบบเซ็นเซอร์อินฟราเรดที่ใช้งาน
- หลักการเขียนโปรแกรมการทำงาน
- ผลการทดลอง
- ลักษณะของปัญหาที่เกิดขึ้น
- ข้อเสนอแนะ

บทที่ 2
วิธี ประมาณ Transfer Function ของระบบใดๆให้เป็นระบบอันดับที่หนึ่ง

วิธีประมาณ transfer function ของระบบใดๆให้เป็นระบบอันดับที่หนึ่งใช้ประมาณค่า transfer function ของ inner loop ในระบบให้เป็นระบบอันดับที่หนึ่ง เพื่อความสะดวกในการวิเคราะห์ outer loop ของระบบ

จากระบบอันดับที่หนึ่งจะได้ block diagram ของระบบคือ



รูปที่ 2.1

เมื่อ α = ค่าคงที่ใดๆ

β = ค่าคงที่ใดๆ

ถ้าให้ อินพุต เป็น step input มีแอมปลิจูดขนาด V_0

นั่นคือ $$V(s) = V_0 / s$$

จาก block diagram จะได้ response ของระบบดังรูปที่ (1.1) และได้ความสัมพันธ์ระหว่าง อินพุต V(s) กับ เอาต์พุต Y(s) ดังนี้

$$Y(s) = V(s) \times [\alpha / (s+\beta)]$$

$$Y(s) = (V_0 \times \alpha) / [s(s+\beta)]$$

$$Y(s) = (V_0 \times \alpha / \beta)[\beta / s(s+\beta)]$$

ทำการ $\mathcal{L}^{-1}$ [Y(s)] ได้

$$y(t) = [V_0 \times \alpha / \beta] \times \mathcal{L}^{-1} [\beta / s(s+\beta)]$$

$$y(t) = (\alpha / \beta)(1 - e^{-\beta t})V_0 \qquad (1)$$

จากสมการที่ 1 พิจารณาที่สถานะคงตัว (steady state) หรือที่ $t \longrightarrow \infty$

และให้ y_{ss} คือ y(t) ที่สถานะคงตัว จะได้

$$y_{ss} = (\alpha / \beta)(1 - 0)V_0$$

นั่นคือ

$$\alpha = y_{ss}\beta / V_0 \qquad (2)$$

จากสมการที่ 1 และ 2 ถ้าพิจารณาขณะที่เวลาใดๆ

ให้ T_k เป็นเวลาที่ y(t) มีค่าเป็น k เท่าของ y_{ss}

เพราะฉะนั้นจะได้

$$y(T_k) = (\alpha / \beta)(1 - e^{-\beta T_k})V_0$$

$$k\, y_{ss} = (y_{ss}\beta / V_0)(1 - e^{-\beta T_k})V_0$$

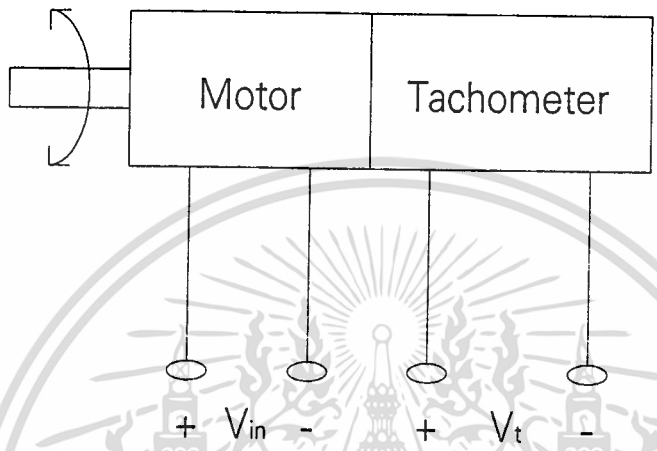
$$\beta = [-\ln(1-k)] / T_k \qquad (3)$$

การ Test Motor

ทาโคมิเตอร์ (tachometer)

เป็นเครื่องมือที่สามารถเปลี่ยนพลังงานกลให้เป็นพลังงานไฟฟ้า และให้กำเนิด เอาต์พุต Voltage ที่เป็นสัดส่วนกับขนาดของความเร็วเชิงมุม

เมื่อจ่าย Voltage input เป็น step input ขนาด 21.8 โวลท์



รูปที่ 2.2

เมื่อ V_{in} = Voltage Input

V_t = Voltage จาก Tachometer

จะได้ Response ของ V_t และ Response ของ $d\theta/dt$ ดังรูป (1.2)

เมื่อจ่าย step input ขนาด 21.8 โวลท์ จะทำให้ได้ Max speed ($d\theta/dt_{max}$) = 371 rad/sec หรือ $V_{t,max}$ = 50.5 โวลท์

เพราะฉะนั้น จะได้ความสัมพันธ์ของ V_t กับ $d\theta/dt$ เป็น

$$V_t = (50.5 / 371)(d\theta/dt)$$

$$V_t = 0.136 (d\theta/dt)$$

เพราะฉะนั้นได้ transfer function ของ tachometer เป็น

transfer function ของ transfer function (K_t)= 0.136

สมการของมอเตอร์ที่ใช้งานจริง

จากผลการทดลองตามทฤษฎีข้างต้น โดยการป้อนไฟฟ้ากระแสตรงขนาด 24 โวลท์ และทาโคมิเตอร์ที่ใช้มีขนาด 70 รอบ/นาที/โวลท์ จะได้ผลดังนี้

การวัดครั้งที่ 1

ได้ โวลท์เตจที่สเตดดีสเตจ = 64.37 โวลท์

โวลท์เตจที่ 63.2 % = 40.68 โวลท์

ค่า Time Constant = 0.154 วินาที

จากสมการข้างต้น

ได้ β = 6.49

α = 1218.47

การวัดครั้งที่ 2

ได้ โวลท์เตจที่สเตดดีสเตจ = 65.00 โวลท์

โวลท์เตจที่ 63.2 % = 41.08 โวลท์

ค่า Time Constant = 0.148 วินาที

จากสมการข้างต้น

ได้ β = 6.75

α = 1279.55

การวัดครั้งที่ 3

ได้ โวลท์เตจที่สเตดดีสเตจ = 65.62 โวลท์

โวลท์เตจที่ 63.2 % = 41.47 โวลท์

ค่า Time Constant = 0.146 วินาที

จากสมการข้างต้น

ได้ β = 6.847

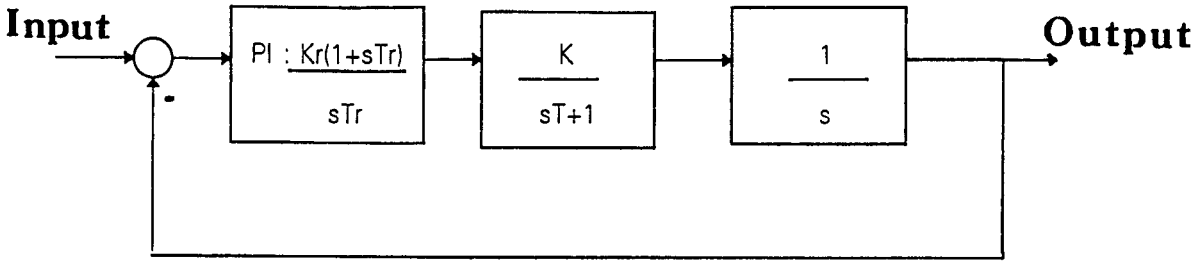
α = 1310.52

สรุปค่าที่ใช้มีดังนี้

ได้ β = 6.7

α = 1270.20

มอเตอร์มีสมการเป็นดังนี้ = $\frac{189.583}{1+0.149S}$



รูปที่ 2.3 บล็อกไดอะแกรมแสดงคอนโทรลเลอร์แบบ PI
และเอนโคดเดอร์ในส่วนป้อนกลับ

ในการหาค่าตัวแปรเราจะใช้วิธี Symmetrical Optimum แบบมี I - Element ซึ่งมีวิธีหาดังนี้

1) กำหนดให้ Tr = 4*(Time Constant ของมอเตอร์)

2) กำหนดให้ Kr = (Time Constant ของ I - Element) / 2*(Time Constant ของมอเตอร์)

3) เนื่องจากวิธีการประมาณค่าแบบนี้จะทำให้เกิดโอเวอร์ชู๊ตสูงเกินกว่า 5 % ดังนั้นเราจะทำการใส่วงจร Smoothing เข้าไป

มีสมการดังนี้ 1 / (1+ STsm) โดย Tsm = Tr

ตำแหน่งที่ใส่เข้าไปจะอยู่ที่ส่วนอินพุท (ตรง Summing Point) ก่อนที่จะเข้าลูป

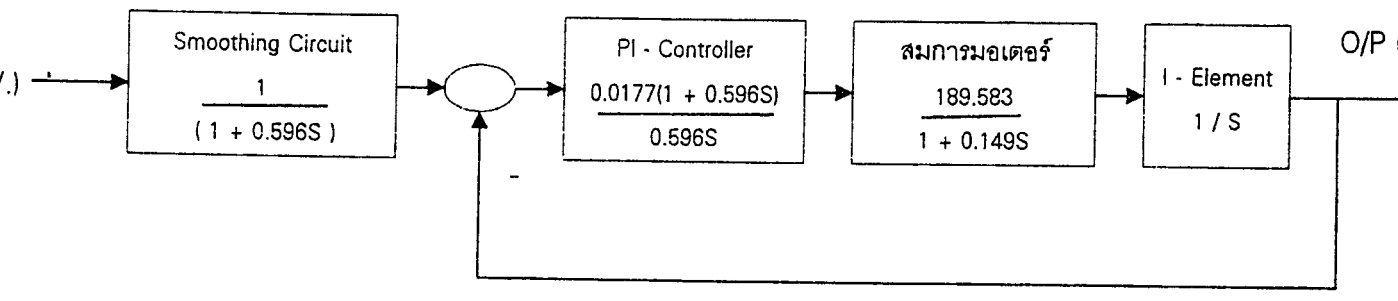
** I - Element ใส่เข้าไปเพื่อแปลงความเร็วเป็นระยะทาง **

จากสมการมอเตอร์ที่หาได้ก่อนหน้านี้แทนค่าลงไป

ได้ Tr = 0.596 วินาที

Kr = 0.0177

Tsm = 0.596 วินาที



รูปที่ 2.4 แสดงบล็อกไดอะแกรมที่สมบูรณ์ของการควบคุมมอเตอร์

บทที่ 3

ส่วนประกอบต่าง ๆ ของ AGV

1. ETT บอร์ด 8032

ทำหน้าที่ประมวลผลค่าที่ได้จากเซนเซอร์ , ตัดสินใจ, ส่งค่าคำสั่งและข้อมูลให้ LM 629 ต่อไป

2. LM 629

ทำหน้าที่รับคำสั่งและข้อมูลจากซีพียูแล้ว ส่งเอาท์พุทเป็นสัญญาณ PWM magnitude และ sign

3. L 298

ทำหน้าที่แปลงสัญญาณจาก PWM magnitude, PWM sign เป็นสัญญาณ PWM1 & PWM2 ซึ่งมีค่า Duty Cycle เป็นสัดส่วนตรงข้ามกัน เช่น PWM1 มี Duty Cycle = 70% , PWM2 จะมี Duty Cycle = 30%

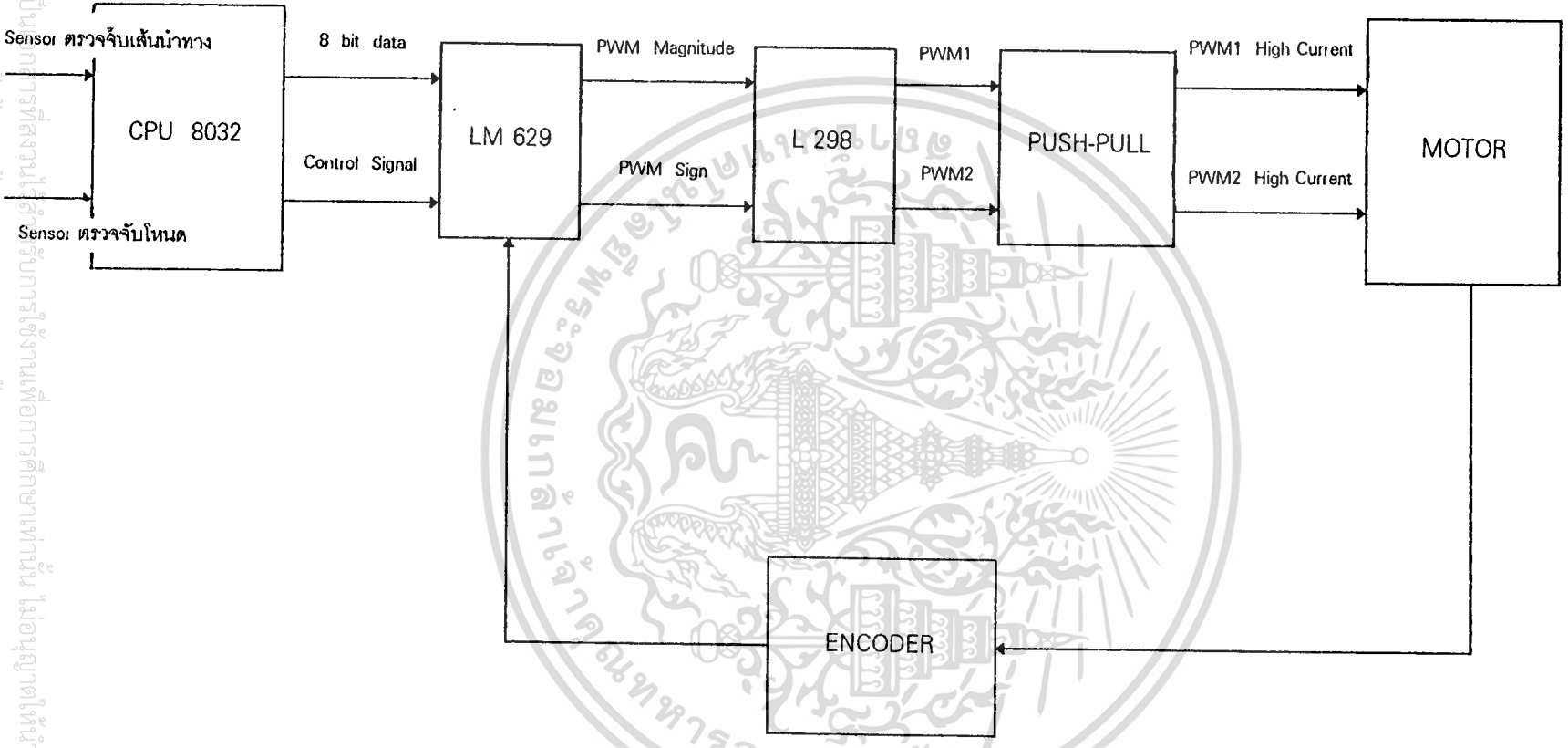
4. PUSH-PULL

เป็น ทรานซิสเตอร์ 4 ตัว ทำหน้าที่ขับกระแสให้เพิ่มขึ้น แล้วส่งต่อให้ มอเตอร์

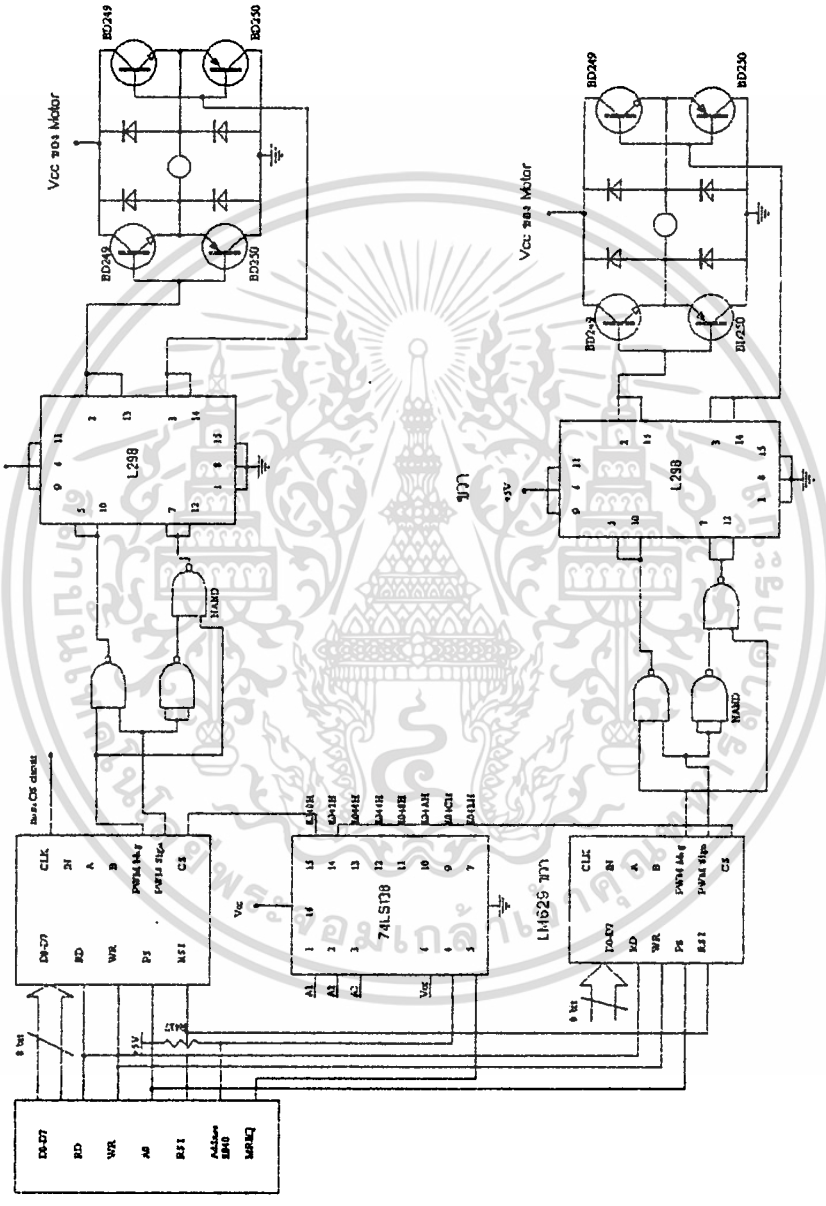
5. แบตเตอรี่ตะกั่ว-กรด

DC 12 V ขนาด 2 Ah (Amp-hour) จำนวน 2 ลูก โดยลูกหนึ่งจ่ายให้กับบอร์ด 8032, แผงวงจร และแผงเซ็นเซอร์ ส่วนอีกลูกหนึ่งจ่ายให้กับมอเตอร์

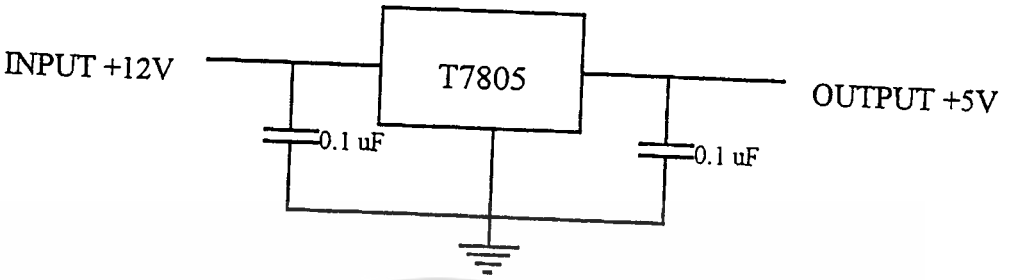
****



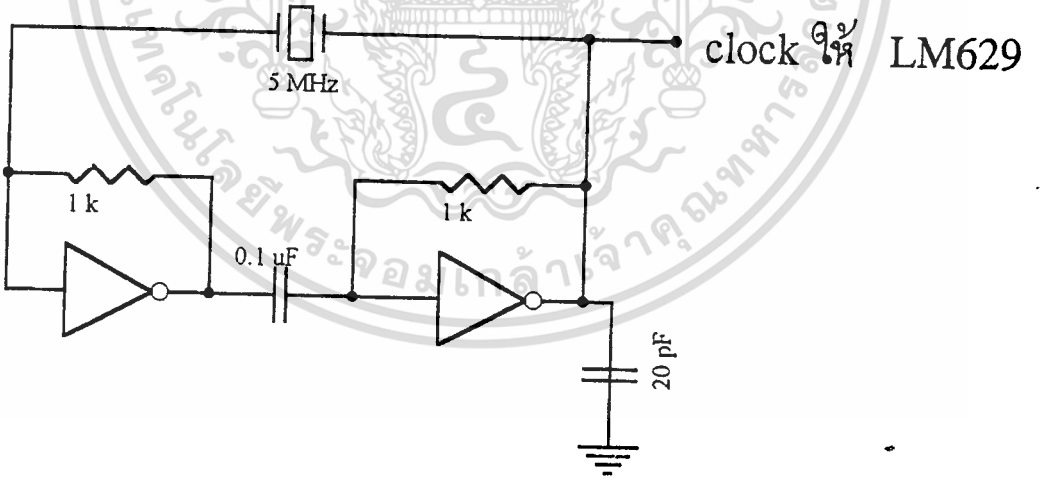
รูปที่ 3.1 บล็อกไดอะแกรมแสดงการทำงานของทั้งระบบ



รูปที่ 3.2 แผนผังวงจรขับมอเตอร์



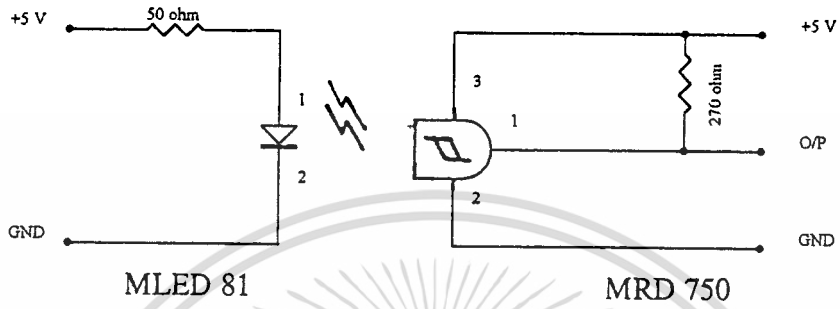
รูปที่ 3.3 วงจรแปลงไฟ DC 12 V. เป็น 5 V.



รูปที่ 3.4 วงจรป้อน clock ให้กับ LM 629

ระบบ SENSOR

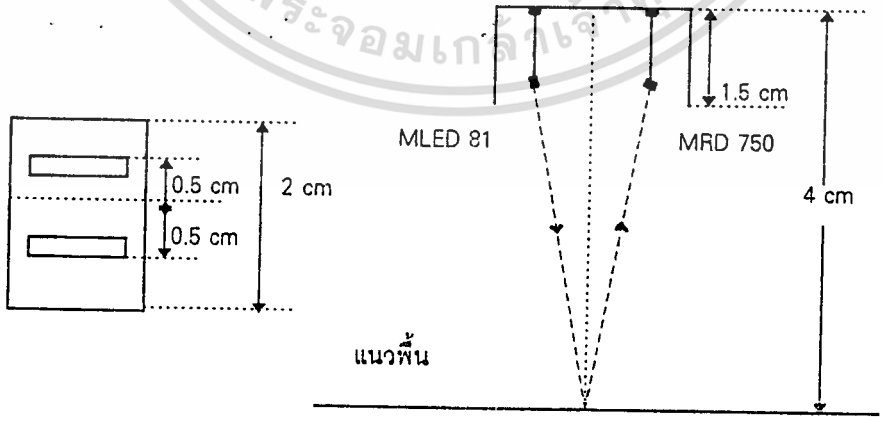
ระบบ sensor ที่ใช้ในวงจรนี้ ใช้ตัวส่งอินฟราเรดเบอร์ MLED 81 ส่วนตัวรับใช้เบอร์ MRD 750 โดยมีลักษณะการต่อวงจรใช้งานดังนี้



รูปที่ 3.5 แสดงลักษณะเซ็นเซอร์ตรวจจับเส้น

ลักษณะที่ต่อใช้งานใน 1 จุด

MLED 81 จะส่งแสงอินฟราเรดเป็นเส้นตรง พุ่งลงไปกระทบกับพื้น ซึ่งจะติดแทบสีไว้เป็นเส้นทางเดินของตัว ROBOT โดยจะติดเป็นแทบสีเงินไว้ เมื่อ MLED 81 ส่งแสงออกมากระทบแทบสีนี้ก็จะสะท้อนกลับขึ้นมายัง MRD 750 เมื่อ MRD 750 จับสัญญาณแสงนี้ได้ ก็จะให้สัญญาณเอาท์พุทออกมาเป็นพัลซ์ dc ขนาด 0 V.



รูปที่ 3.6 แสดงระยะการทำงาน

การทำงานของระบบเซ็นเซอร์บนตัวรถ

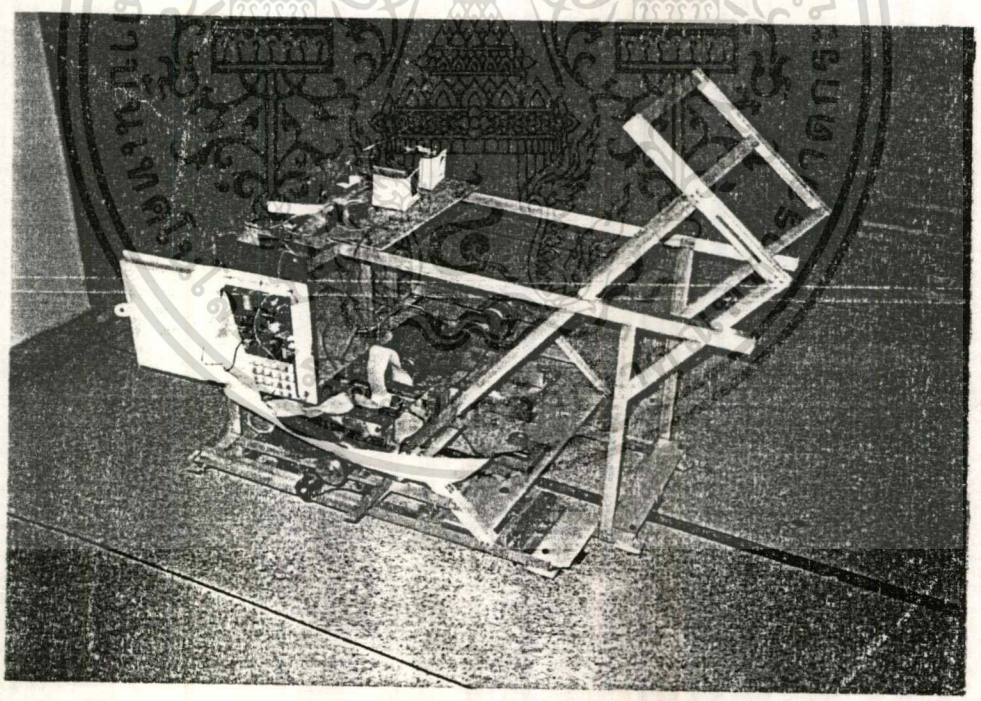
จะมีเซ็นเซอร์ 8 ชุด ที่ต่อเข้ากับพอร์ต A ของบอร์ด 8032 ตามที่ได้แสดงในรูปข้างต้น โดยที่จุด PA 2 ใช้เป็นตัวจับแถบสีในการเคลื่อนที่ตามปกติ ส่วน PA0 , PA1 ,PA3 และ PA4 ใช้เป็นตัวเช็คว่า ROBOT มีการเคลื่อนที่ออกนอกแถบสีหรือเปล่า ส่วน PA5 , PA7 ใช้ในการจับแถบสีไว้สำหรับการเลี้ยวซ้ายหรือเลี้ยวขวา ส่วน PA6 ใช้เมื่อเกิดการเลี้ยวแล้วจะหยุดการเลี้ยวเมื่อไหร่ โดยจะหยุดเมื่อตัวรับ MRD 750 จับสัญญาณแสงได้ ก็คือมีสัญญาณ พัลซ์เกิดขึ้น

ส่วนเซ็นเซอร์อีก 4 ชุดจะต่อเข้ากับพอร์ต B ของบอร์ด 8032 ใช้ในการเช็คสถานีว่าขณะนี้ได้ผ่านสถานีใดไปแล้ว และกำลังจะไปสถานีใดต่อไป จากที่มีเซ็นเซอร์ 4 ชุดจะมีจำนวนสถานีได้ทั้งหมด 16 สถานี แต่ในอนาคต เรายังสามารถเพิ่มจำนวนสถานีได้อีกเนื่องจากจำนวนขาบิทของพอร์ต B ยังเหลืออีก

บทที่ 4

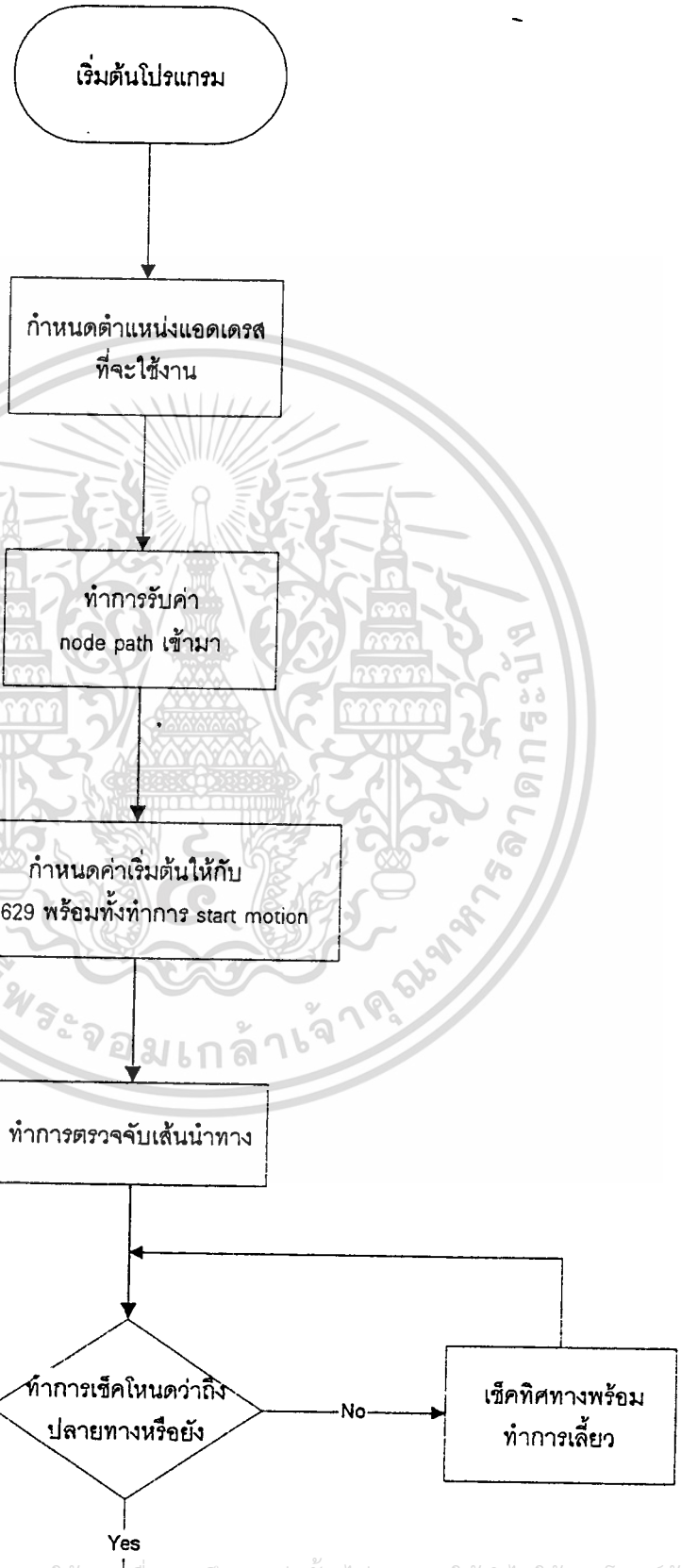
ลักษณะการทำงานของ AGV

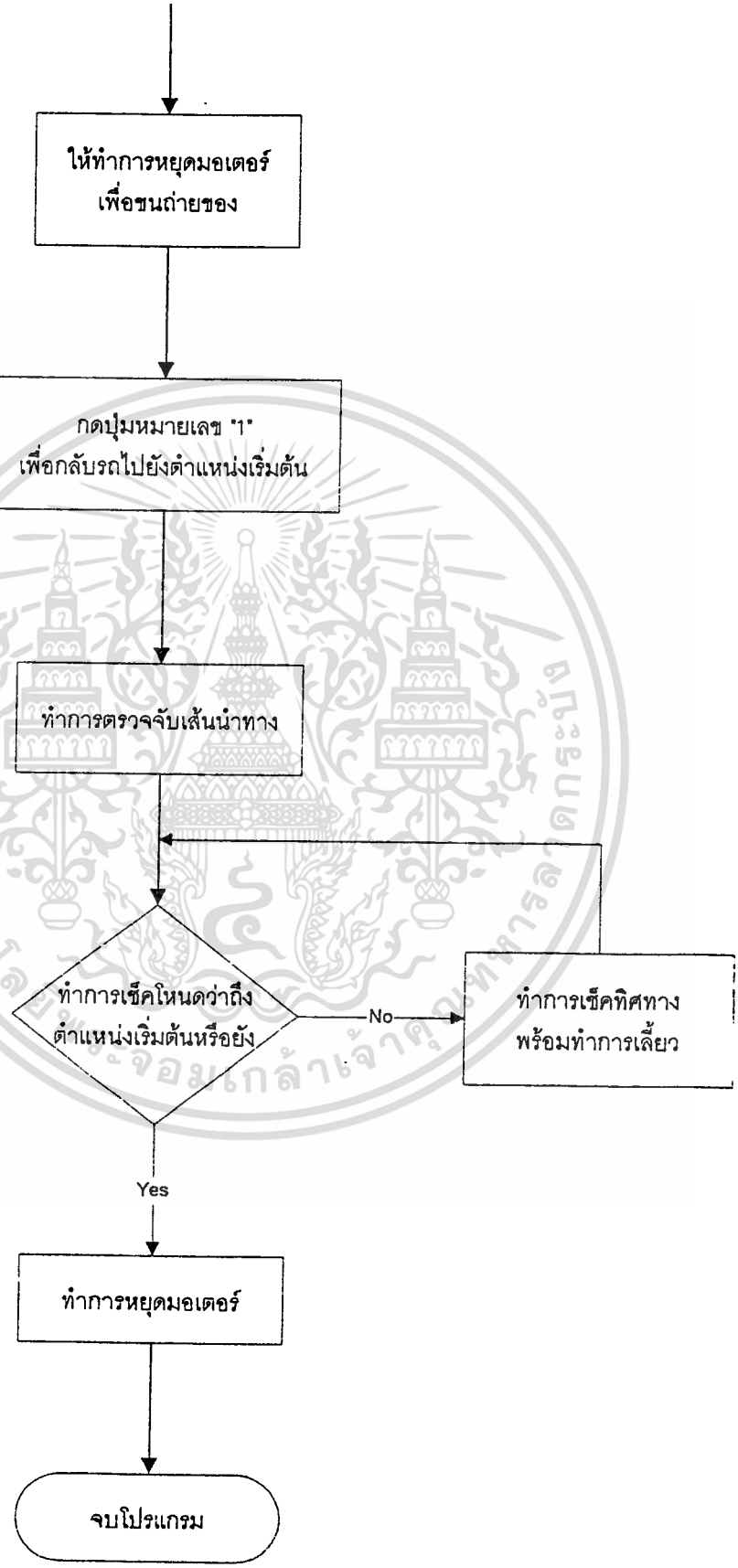
1) ใช้เซ็นเซอร์อินฟราเรดเป็นตัวตรวจจับเส้นนำทาง จุดแยกและสถานี
2) ป้อนค่าโหนดต่าง ๆ ทีละโหนดจนกว่าจะถึงตำแหน่งปลายทางตามแผนที่ที่มีให้
3) เคลื่อนที่ไปตามเส้นนำทางที่เป็นสีเงินสะท้อนแสงโดยถ้าเซ็นเซอร์พบเส้นนี้จะมีลอจิก"0" ถ้าไม่พบจะมีลอจิกเป็น "1"
4) จะเคลื่อนที่ไปทีละโหนดตามที่ได้มีการป้อนไว้ และจะหยุดที่ตำแหน่งปลายทางจนกว่าจะขนถ่ายของเสร็จ จากนั้นทำการกดปุ่มหมายเลข 1 รถจะหมุนตัวกลับ 180 องศา พร้อมทั้งเคลื่อนที่กลับไปตามเส้นทางเดิมและไปหยุดที่ตำแหน่งเริ่มต้น



รูปที่ 4.1 แสดงตัวยานขนส่ง

Flow Chart การทำงาน





อัลกอริทึมการกำหนดค่าให้ LM 629

1) ทำการจองตำแหน่งแอดเดรสในหน่วยความจำไว้ก่อน

2) ป้อนค่าฟิลเตอร์ลงไปในหน่วยความจำในตำแหน่งที่จองไว้

3) รีเซ็ตค่าต่าง ๆ ใน LM 629

4) ส่งค่าฟิลเตอร์จากหน่วยความจำไปยัง LM 629 พร้อมทั้งทำการอัฟเดตฟิลเตอร์ด้วย

5) ป้อนค่า trajectory ลงไปในหน่วยความจำในตำแหน่งที่จองไว้

6) ส่งค่า trajectory จากหน่วยความจำไปยัง LM 629

7) เริ่มทำการ start motion

****

อัลกอริทึมการตรวจจับเส้นนำทาง

1) รับค่าจากพอร์ต A เข้ามา พร้อมทั้งทำการคอมพลีเมนท์

2) and ด้วยค่า 00011111B

3) เช็คว่ามีบิทใดเซ็ตขึ้นมาบ้าง

- ถ้าบิท ACC.2 เซ็ตขึ้นมาแสดงว่ารถอยู่ตามเส้นให้โหลดค่าความเร็วของทั้ง 2 ล้อเท่ากัน
- ถ้าบิท ACC.1 เซ็ตขึ้นมาแสดงว่ารถเบี้ยวไปทางซ้ายเล็กน้อย ให้โหลดค่าความเร็วล้อซ้ายเร็วกว่าล้อขวา 20 %
- ถ้าบิท ACC.3 เซ็ตขึ้นมาแสดงว่า รถเบี้ยวไปทางขวาเล็กน้อย ให้โหลดค่าความเร็วล้อขวาเร็วกว่าล้อซ้าย 20 %
- ถ้าบิท ACC.4 เซ็ตขึ้นมาแสดงว่า รถเบี้ยวไปทางขวามากขึ้น ให้โหลดค่าความเร็วล้อขวาเร็วกว่าล้อซ้ายมากขึ้น
- ถ้าบิท ACC.0 เซ็ตขึ้นมาแสดงว่า รถเบี้ยวไปทางซ้ายมากขึ้น ให้โหลดค่าความเร็วล้อซ้ายเร็วกว่าล้อขวามากขึ้น
- ถ้าไม่มีบิทใดเซ็ตขึ้นมาเลย ก็จะทำซ้ำขั้นตอนเดิมไปเรื่อย ๆ จนกว่าจะมีบิทใดเซ็ตขึ้นมา

****

<u>อัลกอริทึมการเช็คทิศทาง(ตัดสินใจว่าจะวิ่งตรงหรือเลี้ยว)</u>

1) เมื่อกำหนดโหนดปลายทางขึ้น โปรแกรมจะกำหนดเส้นทางเดินไว้ และทำการเก็บค่าโหนดเหล่านั้นไว้ในหน่วยความจำ

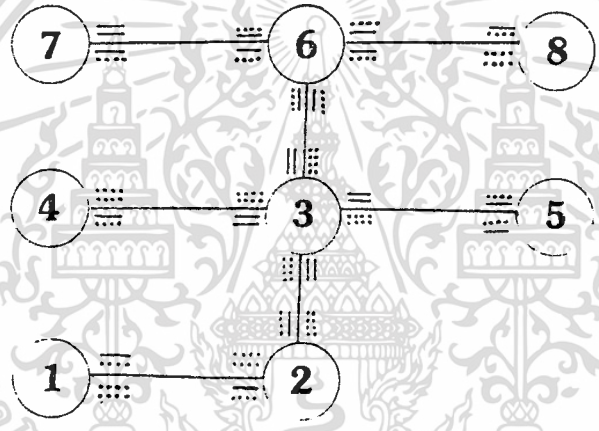
2) ทำการเก็บโหนดที่เซ็นเซอร์โหนดจับได้เป็นโหนดปัจจุบัน จากนั้นทำการโหลดค่าโหนดถัดไปจากหน่วยความจำที่เก็บไว้เข้ามา

3) ทำการเช็คโหนดปัจจุบันว่าเป็นสถานีหรือไม่

ถ้าเป็นสถานีก็ให้ทำการหยุดรถทันที

ถ้าไม่ใช่ก็ให้ทำการเช็คทิศทาง

4) เมื่อเจอแถบโหนดกลับหัวก็ให้เคลียร์บิทสถานีทิ้ง



รูปที่ 4.2 แสดงเส้นทางของโหนดต่างๆ รวมทั้งตำแหน่งเซ็นเซอร์จับโหนด

5) เมื่อเจอโหนดใหม่ให้

ถอยค่า โหนดปัจจุบัน เป็น โหนดก่อน

เก็บค่า โหนดใหม่ เป็น โหนดปัจจุบัน

โหลดค่า โหนดถัดไป จาก หน่วยความจำ

ทำการเก็บค่าโหนดทั้ง 3 นี้ ในรูปของไบท์ต่อโหนด เรียกว่า ไบท์บอกโหนด

6) ใช้โหนดปัจจุบันเป็นโหนดเปรียบเทียบเพื่อหาทิศที่ต้องการจะไปโดยเข้าไปเช็คในไบท์สถานะของโหนดในโปรแกรม

- เปรียบเทียบกับโหนดก่อนว่าโหนดก่อนอยู่ทางทิศอะไรของโหนดปัจจุบันจากนั้นเก็บค่าที่ได้ไว้ใน ไบท์บอกทิศก่อน

- เปรียบเทียบกับโหนดถัดไปว่าโหนดถัดไปอยู่ทางทิศอะไรของโหนดปัจจุบันจากนั้นเก็บค่าที่ได้ไว้ใน ไบท์บอกทิศถัดไป

ลักษณะไบท์บอกทิศจะมี 4 บิทโดยเป็นลอจิก " 1 " เพียงทิศเดียวเรียงดังนี้ N S E W

7) เข้าสู่ขั้นตอนการเปรียบเทียบโดยใช้ตารางที่

- เช็คไบท์บอกทิศก่อนว่าเป็นทิศอะไร

ถ้าเป็นทิศ	N	ก็โดดไปไปทำที่	รูทีนหนึ่ง
ถ้าเป็นทิศ	S	ก็โดดไปไปทำที่	รูทีนสอง
ถ้าเป็นทิศ	E	ก็โดดไปไปทำที่	รูทีนสาม
ถ้าเป็นทิศ	W	ก็โดดไปไปทำที่	รูทีนสี่

- รูทีนหนึ่ง ทำการเช็คไบท์บอกทิศถัดไป

ถ้าเป็นทิศ	S	ก็ให้ตรงไป
ถ้าเป็นทิศ	W	ก็ทำการเลี้ยวขวา
ถ้าเป็นทิศ	E	ก็ทำการเลี้ยวซ้าย

- รูทีนสอง ทำการเช็คไบท์บอกทิศถัดไป

ถ้าเป็นทิศ	N	ก็ให้ตรงไป
ถ้าเป็นทิศ	E	ก็ทำการเลี้ยวขวา
ถ้าเป็นทิศ	W	ก็ทำการเลี้ยวซ้าย

- รูทีนสาม ทำการเช็คไบท์บอกทิศถัดไป

ถ้าเป็นทิศ	W	ก็ให้ตรงไป
ถ้าเป็นทิศ	N	ก็ทำการเลี้ยวขวา
ถ้าเป็นทิศ	S	ก็ทำการเลี้ยวซ้าย

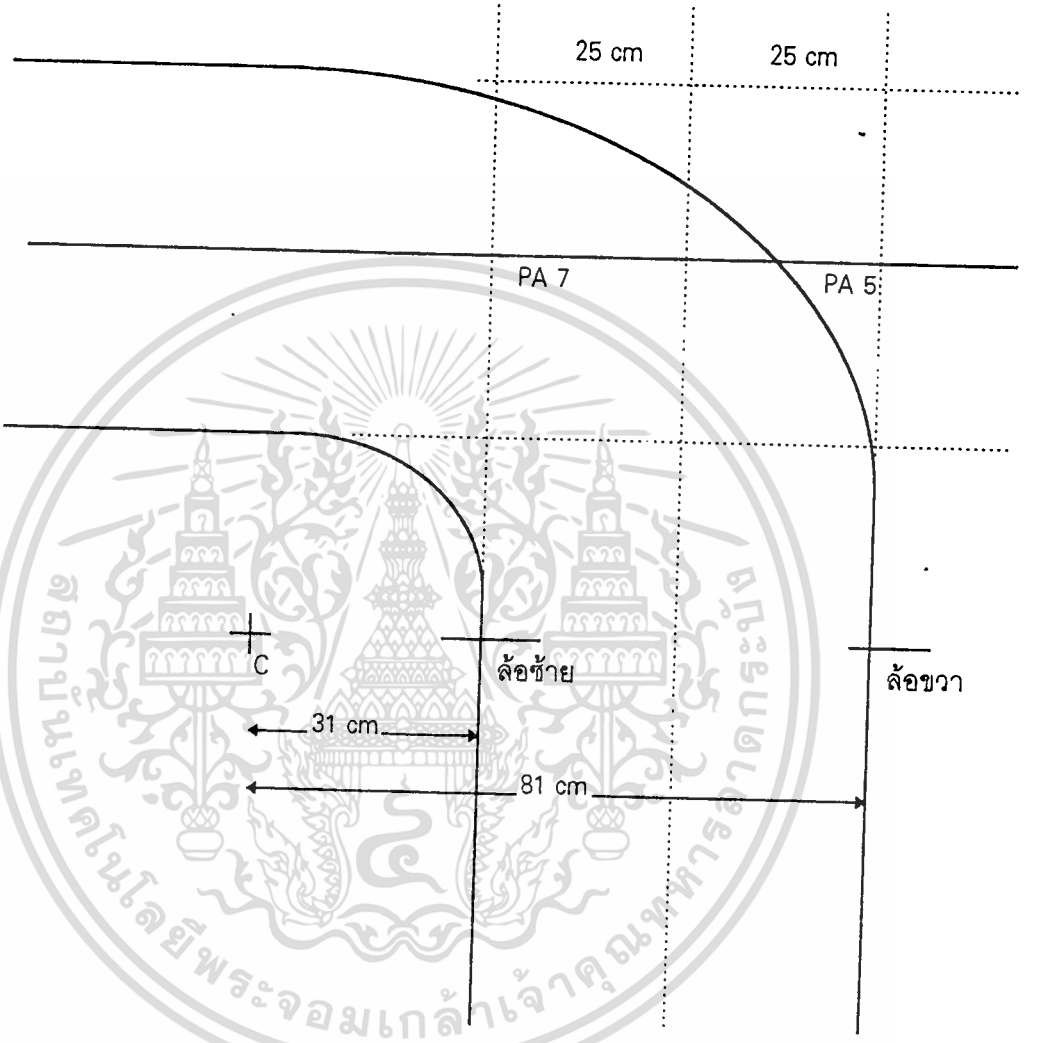
- รูทีนสี่ ทำการเช็คไบท์บอกทิศถัดไป

ถ้าเป็นทิศ	E	ก็ให้ตรงไป
ถ้าเป็นทิศ	S	ก็ทำการเลี้ยวขวา
ถ้าเป็นทิศ	N	ก็ทำการเลี้ยวซ้าย

****

อัลกอริทึมการเลี้ยว

1. คำนวณรัศมีการเลี้ยวจาก ระยะจากชิ้นส่วนต่างๆ ของ Robot ดังรูป



รูปที่ 4.3 แสดงรัศมีการเลี้ยวของ AGV

ระยะห่างของ 2 ล้อขับ = 50 cm

ระยะห่างจาก Node Sensor ถึง ล้อขับ = 56.5 cm

รัศมีของล้อขับ = 9 cm

2. ใช้วงเวียนลากเส้น โดยเปลี่ยนรัศมีไปจนกว่าเส้นที่วงเวียนลากไปนั้นสัมผัสกับแนวล้อ
3. จะได้ค่า รัศมีวงใน = 31 เซนติเมตร

   รัศมีวงนอก = 81 เซนติเมตร
4. คำนวณระยะทางที่แต่ละล้อต้องวิ่งไป โดยจะเห็นว่าต้องวิ่งไป 1/4 ของเส้นรอบวงแต่ละวง
5. <u>ตัวอย่างเช่น การเลี้ยวซ้าย</u>

   ล้อซ้ายต้องหมุนไป (1/4) * 2 * (22/7) * (รัศมีวงใน = 31 ซม.) = 48.7 ซม.

   คิดเป็นจำนวนรอบเท่ากับ 48.7 / [2* 22/7 * 9] = 0.861 รอบ

   ค่าจำนวนการนับในซอฟแวร์ มีค่าเท่ากับ

   4 * (พัลซ์จากเอนโคดเดอร์) * (จำนวนรอบที่แท้จริง)

   ล้อขวาต้องหมุนไป (1/4) * 2 * (22/7) * (รัศมีวงนอก = 81 ซม.) = 127.2 ซม.

   คิดเป็นจำนวนรอบเท่ากับ 127.2 / [2* 22/7 * 9] = 2.25 รอบ

   ค่าจำนวนการนับในซอฟแวร์ มีค่าเท่ากับ

   4 * (พัลซ์จากเอนโคดเดอร์) * (จำนวนรอบที่แท้จริง)

*** <u>*หมายเหตุ*</u> 1. การที่ต้องคูณด้วย 4 เข้าไป เป็นเพราะ LM 629 ทำการนับค่าความแตกต่างพัลซ์ Phase A และ Phase B ของเอนโคดเดอร์ย่อยลงไปอีก 4 ครั้ง

2. การเลี้ยวขวา ก็ทำนองเดียวกับการเลี้ยวซ้าย โดยมี

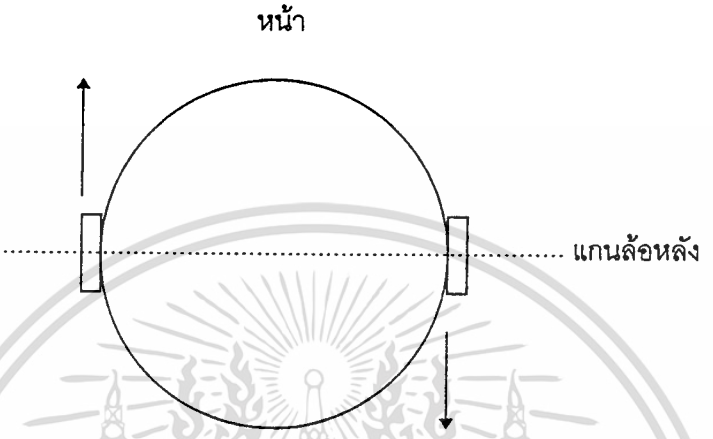
   รัศมีวงใน เป็นของล้อขวา

   รัศมีวงนอก เป็นของล้อซ้าย

****

อัลกอริทึมการหมุนกลับตัว 180 องศา

1. กำหนดให้หมุนกลับตัวทางขวา
2. ล้อทั้ง 2 ล้อ หมุนกลับทิศกัน ด้วยระยะเคลื่อนที่เท่ากัน



รูปที่ 4.4 แสดงการหมุนตัวกลับ 180 องศา

- จุดหมุน อยู่ กึ่งกลางของทั้ง 2 ล้อ
- รัศมีการหมุน = 25 เซนติเมตร
- ระยะทางที่แต่ละล้อหมุนไป เท่ากับ

  (1/2) * 2 * (22/7) * 25 = 78.54 cm
- จำนวนรอบที่แต่ละล้อต้องหมุน

  78.54 / [2 * 22/7 * 9] = 1.39 รอบ

3. โหลดค่า Trajectory เป็นการควบคุมแบบ Position Control โดย

   -ให้จำนวนการนับของล้อซ้ายเป็นบวกมีขนาดเท่ากับ

   4 * (พัลซ์ของเอนโคดเดอร์) * 1.39

   -ให้จำนวนการนับของล้อขวาเป็นลบมีขนาดเท่ากับ

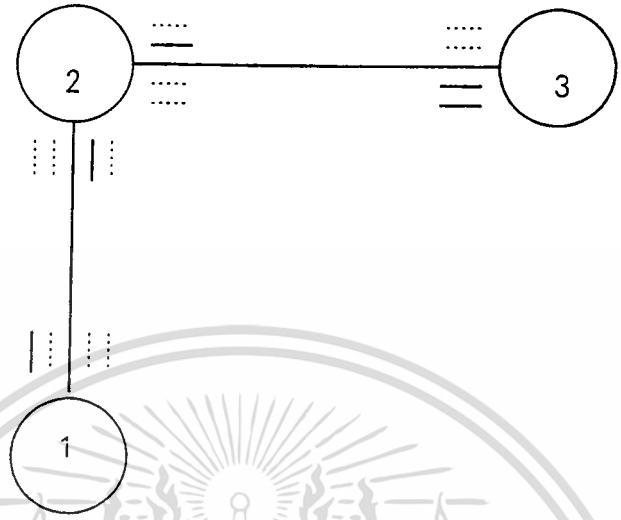
   4 * (พัลซ์ของเอนโคดเดอร์) * 1.39

****

บทที่ 5

ผลการทดลอง

ใช้แผนที่ ตามรูปดังนี้



รูปที่ 5.1 แสดงแผนที่ขนาด 3 โหนด

การทำงานเริ่มขึ้นเมื่อกดปุ่ม RUN โปรแกรม ตัวรถทำงานในโหมดความเร็วโดยตัวรถจะวิ่งจากโหนด 1 ไปยังโหนด 2 เซ็นเซอร์ตรวจจับเส้นทางจะคอยตรวจสอบว่ารถวิ่งตรงตามเส้นทางหรือไม่ ส่วนเซ็นเซอร์ตรวจจับโหนดจะตรวจสอบว่าขณะนี้ถึงโหนด 2 แล้วหรือยัง ถ้าถึงแล้วให้ทำการเช็คทิศทางว่าจะต้องเลี้ยวทางใด เมื่อเช็คเสร็จก็เข้าสู่โหมด Position ทำการเลี้ยวซ้ายเพื่อวิ่งต่อไปยังโหนด 3 ซึ่งเป็นตำแหน่งปลายทาง เมื่อเซ็นเซอร์ตรวจพบโหนด 3 ก็จะหยุดรถเพื่อขนถ่ายของ เมื่อขนถ่ายของเสร็จก็กดปุ่มหมายเลข 1 เพื่อหมุนรถกลับ 180 องศา แล้ววิ่งกลับไปยังตำแหน่งเริ่มต้น

ความเร็วของรถส่งของ (no load) = 180 cm/min

ความเร็วของรถส่งของ (loaded) = 180 cm/min

ค่าต่าง ๆ ในฟิลเตอร์

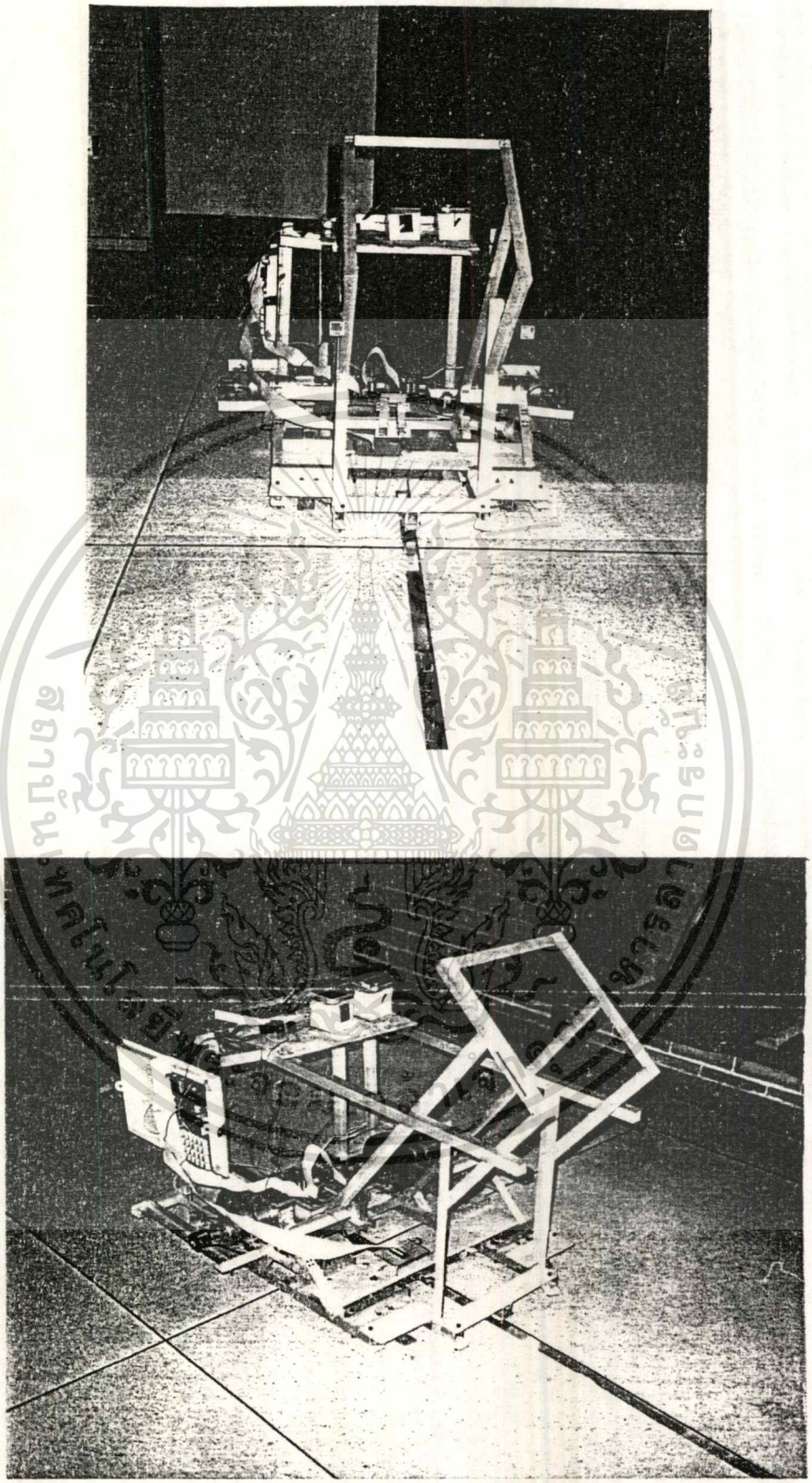
Kp = 000F H

Ki = 0001 H

Kd = 0000 H

น้ำหนักโหลดที่สามารถบรรทุกได้ = 30 kg

ระยะเวลาที่ AGV ทำงานได้สมบูรณ์ = 30 min.



รูปที่ 5.2 ยานขนส่งขณะปฏิบัติงาน

สรุปผลการทดลองและวิจารณ์

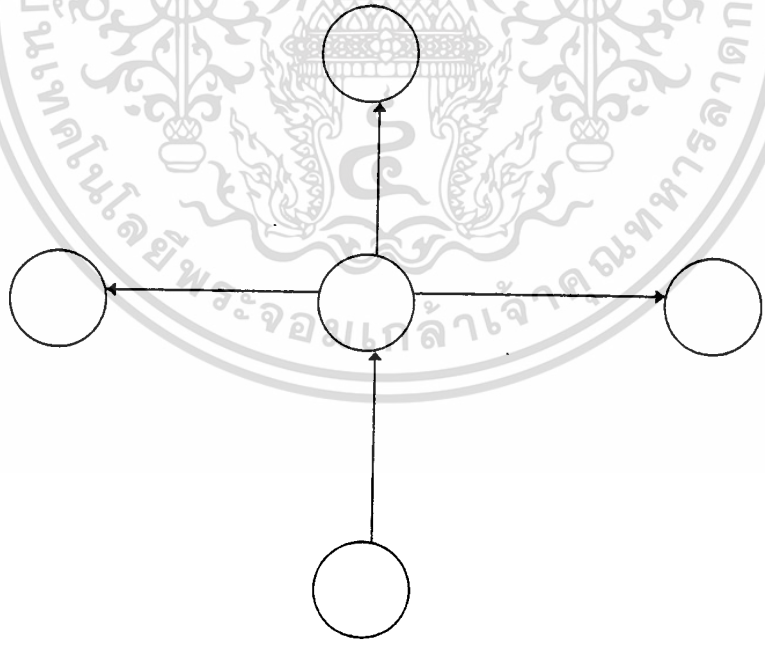
ตัว AGV จะทำงานได้ถูกต้องสมบูรณ์นั้น จะขึ้นกับว่าไฟที่แบตเตอรี่จ่ายให้เพียงพอหรือไม่ ก็คือ ในกรณีที่แบตเตอรี่ชาร์จเต็ม ระบบเซ็นเซอร์อินฟราเรดจะมีความสามารถในการตรวจจับหาเส้นนำทางได้ดี เป็นผลให้ช่วงเวลานั้น AGV จะทำงานได้ถูกต้องไม่มีข้อผิดพลาด ในกรณีที่แบตเตอรี่มีพลังงานจ่ายให้ไม่เพียงพอ ความสามารถในการตรวจจับเส้นนำทางของเซ็นเซอร์อินฟราเรดก็จะด้อยลง เป็นผลให้ AGV ทำงานผิดพลาดไม่ตรงตามคำสั่ง นอกจากนั้นความสมบูรณ์ของเส้นนำทาง , แถบพื้นบริเวณนอกเส้นนำทาง , สภาวะแสงภายนอก หรือแม้แต่วิธีการติดตั้งตำแหน่งเซ็นเซอร์ สิ่งเหล่านี้มีผลต่อการเคลื่อนที่ของ AGV

การเปลี่ยนแปลงเส้นทางเดินของ AGV นั้น ก็ทำได้โดยเพียงแค่เปลี่ยนทิศของโหนดแต่ละโหนดในโปรแกรมเท่านั้น ดังแสดงในรูปที่ 5.3 การกำหนดทิศจะกำหนดยังไงก็ได้ แต่เพื่อให้การทำงานง่ายขึ้นจะกำหนดให้โหนดแต่ละโหนดเป็นดังนี้

โหนด 1 ให้เป็น จุดเริ่มต้น

โหนด 2 ให้เป็น จุดแยก

โหนด 3 ให้เป็น จุดปลาย



รูปที่ 5.3 แสดงแผนที่ของ AGV ที่สมบูรณ์

ปัญหาที่เกิดขึ้น

1.) ETT บอร์ดทำงานค่อนข้างผิดผลาดง่าย เช่น

- ตำแหน่งอินเตอร์รัพท์เวกเตอร์ผิดไปจากคู่มือ
- ตำแหน่งแอดเดรสบางตำแหน่งที่ทางบอร์ดกำหนดให้ใช้ได้แต่พอเขียนลงไปกลับทำงานไม่ได้
- เวลาจะเลิกใช้งานแล้วควรเข้าโหมด Power Down Off ก่อน มิฉะนั้นเวลาสั่ง Run โปรแกรมบอร์ดจะไม่ทำการ Run โปรแกรมให้

2.) ธรรมดาขา pin สำหรับเสียบสายข้อมูลต้องมีค่าความต้านทานระหว่างขามาก ๆ แต่จากที่วัดได้มีค่าความต้านทานเพียง 123 โอห์มเท่านั้น ทำให้ไม่เกิดการรับส่งข้อมูล

3.) แผ่นวงจรที่นำไปให้ทางร้านกัดพริ้นท์ให้ลายทองแดงไม่ถึงกันทั้ง ๆ ที่ความจริงแล้วไม่ถึงกัน

4.) ต้องต่อ C แทนทาลัมระหว่างขา Vcc กับ Ground ของชิพ LM629 เพื่อให้ไฟที่จ่ายคงที่

5.) เซ็นเซอร์อินฟราเรดที่ใช้จับระยะได้น้อยเกินไป และถ้าเส้นนำทางไม่สมบูรณ์เพียงเล็กน้อย เซ็นเซอร์ก็จะตรวจจับไม่พบทำให้เกิดข้อผิดผลาดได้ เช่น ในกรณีที่ตรวจจับโหนดไม่พบก็จะทำการเลี้ยวหรือหยุดรถไม่ได้

6.) แบตเตอรี่ที่จ่ายไฟให้กับ ETT บอร์ด 8032 กับแผงเซ็นเซอร์ใช้ไฟมาก ทำให้ใช้งานได้ไม่นานนัก ต้องคอยตรวจสอบเพื่อนำกลับมาชาร์จใหม่อยู่เรื่อย ๆ

ข้อเสนอแนะ

1.) ระบบเซ็นเซอร์ควรใช้แบบที่ตรวจจับระยะได้มากขึ้น ตรวจพบเส้นนำทางได้แม้เส้นชำรุดไปบ้าง นอกจากนี้วิธีการติดตั้งตำแหน่งเซ็นเซอร์ก็ยังมีผลต่อการเคลื่อนที่ของตัวรถอีกด้วย

2.) ควรเปลี่ยนการควบคุมจาก ETT บอร์ด เป็นไมโครคอนโทรลเลอร์แทนเพื่อประหยัดราคาและพื้นที่ใช้งาน รวมทั้งการควบคุมก็จะทำได้ง่ายขึ้น

3.) ควรมีวงจรตรวจสอบแรงดันของแบตเตอรี่อยู่ตลอดเพื่อให้งานไม่ขาดตอน

เอกสารอ้างอิง

1) Dipl. - Ing. Arne buxbaum , Dipl. - Ing. Klaus Schierau , Professor ; Design of Control Systems for DC Drives ; Springer - Verlag Berlin Heidelberg 1990

2) ดร. สุธี ผู้เจริญชนะชัย ; รายงานเกี่ยวกับสมการมอเตอร์

3) ปรเมษฐ์ ประนยานันทน์ , ปิยพงศ์ เผ่าวณิช ; คู่มือการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS - 51 ; บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) ; พ. ศ. 2536

4) คู่มือบอร์ด ETT - 8032

5) Data Sheet MLED 81 ; Motorola Semiconductor

6) Data Sheet MRD 750 ; Motorola Semiconductor

7) Data Sheet LM 629 ; National Semiconductor

ภาคผนวก ก. ตัวอย่างโปรแกรมการทำงานขนาด 3 โหนด

```
;FILE NAME          MOBILE.ASM
;DISCRIPTION
;ASSEMBLER          SXA51/DOS
;DATE-M-Y           4-4-96
                ;   HEX   ; DATA BYTE
RESETCODE       EQU 00H   ;      0
PORT8           EQU 05H   ;      0
PORT12          EQU 06H   ;      0
DFH             EQU 02H   ;      0
SIP             EQU 03H   ;      0
LPEI            EQU 1BH   ;      2
PLES            EQU 1AH   ;      2
SBPA            EQU 20H   ;      4
SBPR            EQU 21H   ;      4
MSKI            EQU 1CH   ;      2
RSTI            EQU 1DH   ;      2
LFIL            EQU 1EH   ;    2 - 10
UDF             EQU 04H   ;      0
LTRJ            EQU 1FH   ;    2 - 14
STT             EQU 01H   ;      0
;RDSTAT         EQU NONE  ;      1
RDSIGS          EQU 0CH   ;      2
RDIP            EQU 09H   ;      4
RDDP            EQU 08H   ;      4
RDRP            EQU 0AH   ;      4
RDDV            EQU 07H   ;      4
RDRV            EQU 0BH   ;      2
RDSUM           EQU 0DH   ;      2

L_COMMAND       EQU 0E040H
L_DATA          EQU 0E041H
R_COMMAND       EQU 0E042H
R_DATA          EQU 0E043H
PORT_A            EQU 0E020H
PORT_B            EQU 0E021H
PORT_C            EQU 0E022H
P_CON           EQU 0E023H
                ORG 00000H
;*********     FILTER DATA ADDRESS  ********
BANK0:          DS  8
BANK1:          DS  8
BANK2:          DS  8
BANK3:          DS  8
OLD:            DS  1
STATUS:         DS  1
LFIL_DATA:      DS  10
L_LTRJ:         DS  14
R_LTRJ:         DS  14
L_V:            DS  4
R_V:            DS  4
A1:             DS  1
STACK:          DS  32
```

```
OUT_L         EQU   STATUS.0
OUT_R         EQU   STATUS.1
PRESS         EQU   STATUS.2
FINISHBIT     EQU   STATUS.3

              ORG   8000H ;
              LJMP  DRIVE_START

CHECK_LINE:   MOV     DPTR,#0E002H
              MOV     A,#06H
              MOVX    @DPTR,A
              MOV     DPTR,#PORT_A
              MOVX    A,@DPTR
              CPL     A
              ANL     A,#00011111B
              MOV     DPTR,#0E001H
              MOVX    @DPTR,A
              JB      ACC.2,DO_OOIOO
              JB      ACC.3,DO_OIOOO
              JB      ACC.1,DO_OOOIO
              JB      ACC.4,DO_IOOOO
              JB      ACC.0,DO_OOOOI
              ; COUNT_TIME
              LJMP    CHECK_LINE
DO_OOIOO:     LJMP    OOIOO
DO_OIOOO:     LJMP    OIOOO
DO_OOOIO:     LJMP    OOOIO
DO_IOOOO:     LJMP    IOOOO
DO_OOOOI:     LJMP    OOOOI

OOIOO:          MOV     R_V+0,#000H
                MOV     R_V+1,#001H
                MOV     R_V+2,#000H
                MOV     R_V+3,#000H
                LCALL   RR_V_SET
                MOV     L_V+0,#000H
                MOV     L_V+1,#001H
                MOV     L_V+2,#000H
                MOV     L_V+3,#000H
                LCALL   LL_V_SET
                CLR     OUT_L
                CLR     OUT_R
                RET

OIOOO:          MOV     R_V+0,#000H
                MOV     R_V+1,#001H
                MOV     R_V+2,#019H
                MOV     R_V+3,#099H
                LCALL   RR_V_SET
                MOV     L_V+0,#000H
                MOV     L_V+1,#000H
                MOV     L_V+2,#0E6H
                MOV     L_V+3,#066H
                LCALL   LL_V_SET
                SETB    OUT_R
                CLR     OUT_L
```

```
                RET
OOOIO:          MOV     R_V+0,#000H
                MOV     R_V+1,#000H
                MOV     R_V+2,#0E6H
                MOV     R_V+3,#066H
                LCALL   RR_V_SET
                MOV     L_V+0,#000H
                MOV     L_V+1,#001H
                MOV     L_V+2,#019H
                MOV     L_V+3,#099H
                LCALL   LL_V_SET
                SETB    OUT_L
                CLR     OUT_R
                RET

IOOOO:          MOV     R_V+0,#000H
                MOV     R_V+1,#000H
                MOV     R_V+2,#090H
                MOV     R_V+3,#000H
                LCALL   RR_V_SET
                MOV     L_V+0,#000H
                MOV     L_V+1,#000H
                MOV     L_V+2,#00FH
                MOV     L_V+3,#000H
                LCALL   LL_V_STOP
;                LCALL   LL_V_SET
                SETB    OUT_R
                CLR     OUT_L
                RET
OOOOI:          MOV     R_V+0,#000H
                MOV     R_V+1,#000H
                MOV     R_V+2,#00FH
                MOV     R_V+3,#000H
                LCALL   RR_V_STOP
;                LCALL   RR_V_SET
                MOV     L_V+0,#000H
                MOV     L_V+1,#000H
                MOV     L_V+2,#090H
                MOV     L_V+3,#000H
                LCALL   LL_V_SET
                SETB    OUT_L
                CLR     OUT_R
                RET
;************ TURN LEFT*********
TURN_LEFT:
                MOV     DPTR,#L_COMMAND     ;LEFT
                MOV     A,#DFH
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#R_COMMAND     ;RIGHT
                MOV     A,#DFH
                MOVX    @DPTR,A
                LCALL   R_BUSY

                MOV     DPTR,#L_COMMAND     ;LEFT
                MOV     A,#LTRJ
```

```
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,#00000000B          ;FWD,POSITION
MODE
                MOVX    @DPTR,A
                MOV     A,#00001010B    ; V,P
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA    ;V
                MOV     A,#00H
                MOVX    @DPTR,A
                MOV     A,#01H
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,#00H
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   L_BUSY

                MOV     DPTR,#L_DATA    ;P
                MOV     A,#00H
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,#20H
                MOVX    @DPTR,A
                MOV     A,#0D0H
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   L_BUSY

                MOV     DPTR,#R_COMMAND   ;RIGHT
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
                MOV     A,#00000000B          ;FWD,POSITION
MODE
                MOVX    @DPTR,A
                MOV     A,#00001010B    ; V,P
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA    ;V
                MOV     A,#00H
                MOVX    @DPTR,A
                MOV     A,#03H
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
```

```
                MOV     A,#50H
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   R_BUSY

                MOV     DPTR,#R_DATA    ;P
                MOV     A,#00H
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
                MOV     A,#55H
                MOVX    @DPTR,A
                MOV     A,#0F0H
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;********* TURN RIGHT***********
TURN_RIGHT:
                MOV     DPTR,#L_COMMAND     ;LEFT
                MOV     A,#DFH
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#R_COMMAND     ;RIGHT
                MOV     A,#DFH
                MOVX    @DPTR,A
                LCALL   R_BUSY

                MOV     DPTR,#L_COMMAND     ;LEFT
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,#00H          ;POSITION MODE
                MOVX    @DPTR,A
                MOV     A,#00001010B    ; V,P
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA    ;V
                MOV     A,#00H
                MOVX    @DPTR,A
                MOV     A,#03H
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,#50H
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   L_BUSY
```

```
MOV     DPTR,#L_DATA    ;P
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   L_BUSY
MOV     DPTR,#L_DATA
MOV     A,#55H
MOVX    @DPTR,A
MOV     A,#0F0H
MOVX    @DPTR,A
LCALL   L_BUSY
MOV     DPTR,#L_COMMAND
MOV     A,#STT
MOVX    @DPTR,A
LCALL   L_BUSY

MOV     DPTR,#R_COMMAND    ;RIGHT
MOV     A,#LTRJ
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
MOV     A,#00H             ;POSITION MODE
MOVX    @DPTR,A
MOV     A,#00001010B    ; V,P
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA    ;V
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#01H
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   R_BUSY

MOV     DPTR,#R_DATA    ;P
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
MOV     A,#20H
MOVX    @DPTR,A
MOV     A,#0D0H
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_COMMAND
MOV     A,#STT
MOVX    @DPTR,A
```

```
                LCALL   R_BUSY
                RET
;********** LTRJ COMPLETE ? *******
  COMPLETE: MOV     DPTR,#L_COMMAND
            MOVX    A,@DPTR
            JNB     ACC.2,COMPLETE
        COM: MOV    DPTR,#R_COMMAND
            MOVX    A,@DPTR
            JNB     ACC.2,COM
            RET

;********** CHECK TURN**********
CHECK_TURN: MOV     DPTR,#PORT_A
            MOVX    A,@DPTR
            CPL     A
            ANL     A,#10101110B

     C1:    CJNE    A,#10100010B,C2
            SJMP    DO
     C2:    CJNE    A,#10100100B,C3
            SJMP    DO
     C3:    CJNE    A,#10101000B,END_TURN
     DO:    JNB     PRESS,RIGHT
            LCALL   TURN_LEFT
            LCALL   COMPLETE
            SJMP    END_TURN
   RIGHT:   LCALL   TURN_RIGHT
            LCALL   COMPLETE
END_TURN:   RET

;*********** TURN 180
TURN_180:
            MOV     DPTR,#L_COMMAND     ;LEFT
            MOV     A,#DFH
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#R_COMMAND     ;RIGHT
            MOV     A,#DFH
            MOVX    @DPTR,A
            LCALL   R_BUSY

            MOV     DPTR,#L_COMMAND     ;LEFT
            MOV     A,#LTRJ
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA
            MOV     A,#00H          ;POSITION MODE
            MOVX    @DPTR,A
            MOV     A,#00001010B     ; V,P
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA    ;V
            MOV     A,#00H
            MOVX    @DPTR,A
            MOV     A,#02H
            MOVX    @DPTR,A
```

```
LCALL   L_BUSY
MOV     DPTR,#L_DATA
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   L_BUSY

MOV     DPTR,#L_DATA    ;P
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   L_BUSY
MOV     DPTR,#L_DATA
MOV     A,#36H
MOVX    @DPTR,A
MOV     A,#4CH
MOVX    @DPTR,A
LCALL   L_BUSY
MOV     DPTR,#L_COMMAND
MOV     A,#STT
MOVX    @DPTR,A
LCALL   L_BUSY

MOV     DPTR,#R_COMMAND    ;RIGHT
MOV     A,#LTRJ
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
MOV     A,#00H          ;POSITION MODE
MOVX    @DPTR,A
MOV     A,#00001010B    ; V,P
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA    ;V
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#02H
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
MOV     A,#00H
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   R_BUSY

MOV     DPTR,#R_DATA    ;P
MOV     A,#0FFH
MOVX    @DPTR,A
MOV     A,#0FFH
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
```

```
                MOV     A,#0C9H
                MOVX    @DPTR,A
                MOV     A,#0B4H
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   R_BUSY
                LCALL   COMPLETE
                RET
;*********  FIND START POINT ... FINISH
  START_POINT?:  MOV    DPTR,#PORT_A
                MOVX    A,@DPTR
                CPL     A
                ANL     A,#00000100B
                JNB     ACC.2,END_POINT?          ; CAR
BENDING

                    MOV   DPTR,#PORT_B
                    MOVX  A,@DPTR
                    CPL   A
                    ANL   A,#00001111B
                    CJNE  A,#01H,END_POINT?
                    SETB  FINISHBIT
   END_POINT?:      RET

;*********  FIND MARK TO STOP TEMP ***
 MARK?:             MOV    DPTR,#PORT_A
                    MOVX   A,@DPTR
                    CPL    A
                    ANL    A,#00001110B
                    JB    ACC.1,DO_MARK
                    JB    ACC.2,DO_MARK
                    JB    ACC.3,DO_MARK
                    LJMP   END_MARK?
DO_MARK:            MOV    DPTR,#PORT_B
                    MOVX   A,@DPTR
                    CPL    A
                    ANL    A,#00001111B
                    CJNE   A,#03H,END_MARK?

                    MOV     DPTR,#L_COMMAND    ;LEFT
                    MOV     A,#LTRJ
                    MOVX    @DPTR,A
                    LCALL   L_BUSY
                    MOV     DPTR,#L_DATA
                    MOV     A,#01H          ; STOP
                    MOVX    @DPTR,A
                    MOV     A,#00H
                    MOVX    @DPTR,A
                    LCALL   L_BUSY
                    MOV     DPTR,#L_COMMAND
                    MOV     A,#STT
                    MOVX    @DPTR,A
                    LCALL   L_BUSY
```

```
                MOV     DPTR,#R_COMMAND    ;RIGHT
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
                MOV     A,#01H             ; STOP
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   R_BUSY
   CHECK:       MOV     A,#27H             ; CHECK KEY 1
                LCALL   0030H
                CJNE    A,#01H,CHECK
                SETB    PRESS
                LCALL   TURN_180
 END_MARK?:     RET


;********     LM628 RESET ********
LL_RESET:       MOV     DPTR,#L_COMMAND
                MOV     A,#RESETCODE
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
RR_RESET:       MOV     DPTR,#R_COMMAND
                MOV     A,#RESETCODE
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;********     DEFINE HOME ******************
LL_DFH:         MOV     DPTR,#L_COMMAND
                MOV     A,#DFH
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
RR_DFH:         MOV     DPTR,#R_COMMAND
                MOV     A,#DFH
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;********     DEFINE 8 BIT ******************
LL_PORT8:       MOV     DPTR,#L_COMMAND
                MOV     A,#PORT8
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
RR_PORT8:       MOV     DPTR,#R_COMMAND
                MOV     A,#PORT8
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;********     FILTER CONTROL COMMAND *******
```

```
LL_LFIL:    MOV     DPTR,#L_COMMAND
            MOV     A,#LFIL
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA
            MOV     A,LFIL_DATA+0
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+1
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA        ;KP
            MOV     A,LFIL_DATA+2
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+3
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA        ;KI
            MOV     A,LFIL_DATA+4
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+5
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA        ;KD
            MOV     A,LFIL_DATA+6
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+7
            MOVX    @DPTR,A
            LCALL   L_BUSY
            MOV     DPTR,#L_DATA        ;II
            MOV     A,LFIL_DATA+8
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+9
            MOVX    @DPTR,A
            LCALL   L_BUSY
            RET
RR_LFIL:    MOV     DPTR,#R_COMMAND
            MOV     A,#LFIL
            MOVX    @DPTR,A
            LCALL   R_BUSY
            MOV     DPTR,#R_DATA
            MOV     A,LFIL_DATA+0
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+1
            MOVX    @DPTR,A
            LCALL   R_BUSY
            MOV     DPTR,#R_DATA        ;KP
            MOV     A,LFIL_DATA+2
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+3
            MOVX    @DPTR,A
            LCALL   R_BUSY
            MOV     DPTR,#R_DATA        ;KI
            MOV     A,LFIL_DATA+4
            MOVX    @DPTR,A
            MOV     A,LFIL_DATA+5
            MOVX    @DPTR,A
```

```
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA        ;KD
                MOV     A,LFIL_DATA+6
                MOVX    @DPTR,A
                MOV     A,LFIL_DATA+7
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA        ;II
                MOV     A,LFIL_DATA+8
                MOVX    @DPTR,A
                MOV     A,LFIL_DATA+9
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;******         UPDATE FILTER  *************
LL_UDF:         MOV     DPTR,#L_COMMAND
                MOV     A,#UDF
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
RR_UDF:         MOV     DPTR,#R_COMMAND
                MOV     A,#UDF
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;********     Trajectory Control Command *******
LV_LTRJ:        LCALL   L_BUSY
                MOV     DPTR,#L_COMMAND
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,L_LTRJ+0
                MOVX    @DPTR,A
                MOV     A,#00001000B
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA    ;V
                MOV     A,L_LTRJ+6
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+7
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,L_LTRJ+8
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+9
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
LL_LTRJ:        LCALL   L_BUSY
                MOV     DPTR,#L_COMMAND
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
```

```
                MOV     A,L_LTRJ+0
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+1
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     A,L_LTRJ+1
                JNB     ACC.5,LL_V
                MOV     DPTR,#L_DATA    ;A
                MOV     A,L_LTRJ+2
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+3
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,L_LTRJ+4
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+5
                MOVX    @DPTR,A
LL_V:           LCALL   L_BUSY
                MOV     DPTR,#L_DATA    ;V
                MOV     A,L_LTRJ+6
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+7
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,L_LTRJ+8
                MOVX    @DPTR,A
                MOV     A,L_LTRJ+9
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
RR_LTRJ:        LCALL   R_BUSY
                MOV     DPTR,#R_COMMAND
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
                MOV     A,R_LTRJ+0
                MOVX    @DPTR,A
                MOV     A,R_LTRJ+1
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     A,R_LTRJ+1
                JNB     ACC.5,RR_V
                MOV     DPTR,#R_DATA    ;A
                MOV     A,R_LTRJ+2
                MOVX    @DPTR,A
                MOV     A,R_LTRJ+3
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
                MOV     A,R_LTRJ+4
                MOVX    @DPTR,A
                MOV     A,R_LTRJ+5
                MOVX    @DPTR,A
```

```
                LCALL   R_BUSY
RR_V:           MOV     DPTR,#R_DATA    ;V
                MOV     A,R_LTRJ+6
                MOVX    @DPTR,A
                MOV     A,R_LTRJ+7
                MOVX    @DPTR,A
                LCALL   R_BUSY
                MOV     DPTR,#R_DATA
                MOV     A,R_LTRJ+8
                MOVX    @DPTR,A
                MOV     A,R_LTRJ+9
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;*******        START MOTION ********
LL_STT:         ;MOV     DPTR,#L_COMMAND
                ;MOVX    A,@DPTR
                ;JNB     ACC.2,LL_STT
                MOV     R0,#00H
                DJNZ    R0,$
                MOV     DPTR,#L_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   L_BUSY
                RET
RR_STT:         ;MOV     DPTR,#R_COMMAND
                ;MOVX    A,@DPTR
                ;JNB     ACC.2,RR_STT
                MOV     DPTR,#R_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   R_BUSY
                RET
;*******        BUSY *********
L_BUSY:         MOV     DPTR,#L_COMMAND
                MOVX    A,@DPTR
                JB      ACC.0,L_BUSY
                RET
R_BUSY:         MOV     DPTR,#R_COMMAND
                MOVX    A,@DPTR
                JB      ACC.0,R_BUSY
                RET

;*********    FILTER CONTROL COMMANDS ******
FILTER:         MOV     LFIL_DATA+00,#00000000B  ; CONTROL
WORD
                MOV     LFIL_DATA+01,#00001111B  ;
                MOV     LFIL_DATA+02,#000H       ; KP
                MOV     LFIL_DATA+03,#00FH       ;
                MOV     LFIL_DATA+04,#000H       ; KI
                MOV     LFIL_DATA+05,#001H       ;
                MOV     LFIL_DATA+06,#000H       ; KD
                MOV     LFIL_DATA+07,#000H       ;
                MOV     LFIL_DATA+08,#000H       ; IL
                MOV     LFIL_DATA+09,#00FH       ;
                RET
```

```
;********    TRAJECTORY CONTROL COMMANDS ****
LL_TRAJ:    MOV     L_LTRJ+00,#00011000B  ; CONTROL
WORD
            MOV     L_LTRJ+01,#00101000B  ;
            MOV     L_LTRJ+02,#000H       ; A
            MOV     L_LTRJ+03,#000H       ;
            MOV     L_LTRJ+04,#000H       ;
            MOV     L_LTRJ+05,#0F0H       ;
            MOV     L_LTRJ+06,#000H       ; V
            MOV     L_LTRJ+07,#000H       ;
            MOV     L_LTRJ+08,#000H       ;
            MOV     L_LTRJ+09,#000H       ;
            RET
RR_TRAJ:    MOV     R_LTRJ+00,#00011000B  ; CONTROL
WORD
            MOV     R_LTRJ+01,#00101000B  ;
            MOV     R_LTRJ+02,#000H       ; A
            MOV     R_LTRJ+03,#000H       ;
            MOV     R_LTRJ+04,#000H       ;
            MOV     R_LTRJ+05,#0F0H       ;
            MOV     R_LTRJ+06,#000H       ; V
            MOV     R_LTRJ+07,#000H       ;
            MOV     R_LTRJ+08,#000H       ;
            MOV     R_LTRJ+09,#000H       ;
            RET
;********   LM628 VELOCITY SETTING *****
;** INPUT = ?_V+0,1,2,3  0=HI_BYTE , 3=LO_BYTE
RR_V_SET:   MOV     R_LTRJ+00,#00011000B  ; CONTROL
WORD
            MOV     R_LTRJ+01,#00001000B  ;
            MOV     R_LTRJ+06,R_V+0       ; V
            MOV     R_LTRJ+07,R_V+1       ;
            MOV     R_LTRJ+08,R_V+2       ;
            MOV     R_LTRJ+09,R_V+3       ;
            ACALL   RR_LTRJ
            ACALL   RR_STT
            ;MOV     DPTR,#0E001H
            ;MOVX    A,@DPTR
            ;INC     A
            ;MOVX    @DPTR,A
            RET
LL_V_SET:   MOV     L_LTRJ+00,#00011000B  ; CONTROL
WORD
            MOV     L_LTRJ+01,#00001000B  ;
            MOV     L_LTRJ+06,L_V+0       ; V
            MOV     L_LTRJ+07,L_V+1       ;
            MOV     L_LTRJ+08,L_V+2       ;
            MOV     L_LTRJ+09,L_V+3       ;
            ACALL   LV_LTRJ
            ACALL   LL_STT
            RET
RR_V_STOP:  MOV     R_LTRJ+00,#00011001B  ; CONTROL
WORD
            MOV     R_LTRJ+01,#00000000B  ;
            ACALL   RR_LTRJ
            ACALL   RR_STT
```

```
                RET
LL_V_STOP:      MOV     L_LTRJ+00,#00011001B   ; CONTROL
WORD
                MOV     L_LTRJ+01,#00000000B   ;
                ACALL   LL_LTRJ
                ACALL   LL_STT
                RET
;******          MOTOR   INITIAL SET ACC AND V=0 ****
LL_INIT:        LCALL   FILTER
                LCALL   LL_RESET
                LCALL   LL_LFIL
                LCALL   LL_UDF
                LCALL   LL_TRAJ
                LCALL   LL_LTRJ
                LCALL   LL_STT
                RET
RR_INIT:        LCALL   FILTER
                LCALL   RR_RESET
                LCALL   RR_LFIL
                LCALL   RR_UDF
                LCALL   RR_TRAJ
                LCALL   RR_LTRJ
                LCALL   RR_STT
                RET

;************ DRIVE START************

DRIVE_START:    CLR     PRESS
                MOV     DPTR,#P_CON
                MOV     A,#10010010B
                MOVX    @DPTR,A
                LCALL   LL_INIT
                LCALL   RR_INIT
;************* CHECK LINE************
  CHECK_ALL:    CLR     FINISHBIT
                LCALL   CHECK_LINE
                LCALL   CHECK_TURN
                LCALL   MARK?
                LCALL   START_POINT?
                JB      FINISHBIT,FINISH
                SJMP    CHECK_ALL

  FINISH:       MOV     DPTR,#L_COMMAND    ;LEFT
                MOV     A,#LTRJ
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_DATA
                MOV     A,#01H          ; STOP
                MOVX    @DPTR,A
                MOV     A,#00H
                MOVX    @DPTR,A
                LCALL   L_BUSY
                MOV     DPTR,#L_COMMAND
                MOV     A,#STT
                MOVX    @DPTR,A
                LCALL   L_BUSY
```

```
MOV     DPTR,#R_COMMAND    ;RIGHT
MOV     A,#LTRJ
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_DATA
MOV     A,#01H          ; STOP
MOVX    @DPTR,A
MOV     A,#00H
MOVX    @DPTR,A
LCALL   R_BUSY
MOV     DPTR,#R_COMMAND
MOV     A,#STT
MOVX    @DPTR,A
LCALL   R_BUSY

END
```

ภาคผนวก ข. Data Sheet LM 628/629

National Semiconductor

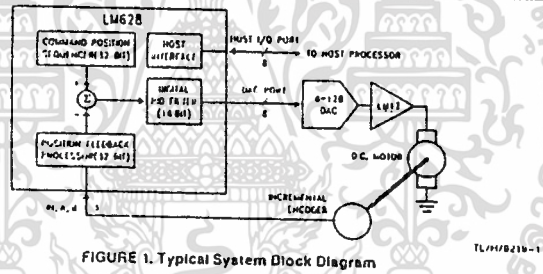
LM628/LM629 Precision Motion Controller

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, and are offered in both 6 MHz and 8 MHz maximum frequency versions. The suffixes -6 and -8, respectively, are used to designate version. 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

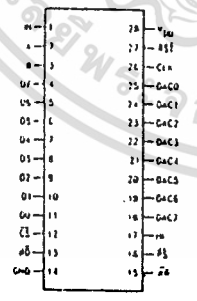


TL/H/6219-1

FIGURE 1. Typical System Block Diagram

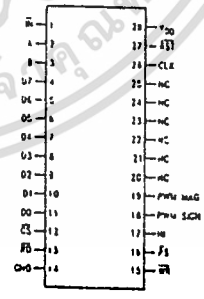
Connection Diagrams

LM628



TL/H/6219-2

LM629



TL/H/6219-3

Order Number LM628N-6, LM628N-8, LM629N-6 or LM629N-8
See NS Package Number 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 (Pin 14)	−0.3V to +7.0V
Ambient Storage Temperature	−65°C to +150°C
Lead Temperature (Soldering, 4 sec.)	260°C
Maximum Power Dissipation	550 mW
ESD Tolerance (C_{ZAP} = 120 pF, R_{ZAP} = 1.5k)	2000V

Operating Ratings

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

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

Symbol	Parameter	Conditions	Tested Limits Min	Tested Limits Max	Units
I_{DD}	Supply Current	Outputs Open		100	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	$0 \le V_{IN} \le V_{DD}$	−10	10	μA
OUTPUT VOLTAGES					
V_{OH}	Logic 1	I_{OH} = −1.6 mA	2.4		V
V_{OL}	Logic 0	I_{OH} = 1.6 mA		0.4	V
I_{OUT}	TRI-STATE® Output Leakage Current	$0 \le V_{OUT} \le V_{DD}$	−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	Tx	Tested Limits Min	Tested Limits Max	Units
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 or LM629N-6	T4	78		ns
LM628N-8 or LM629N-8	T4	57		ns
Clock Period				
LM628N-6 or LM629N-6	T5	166		ns
LM628N-8 or LM629N-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	Tag	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
AD 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 2)	ns
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
AD High to Hi-Z Time	T12		180	ns
Busy Bit Delay	T13		(Note 2)	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 2)	ns
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 stated Operating Ratings.

Note 2: 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.

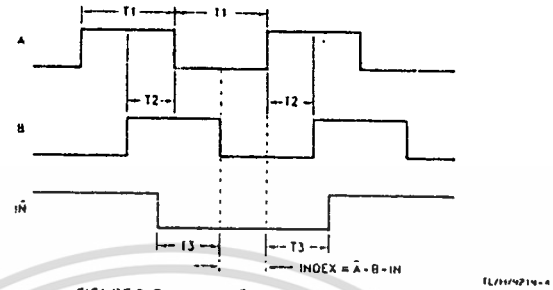


FIGURE 2. Quadrature Encoder Input Timing

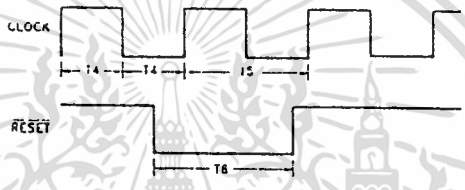


FIGURE 3. Clock and Reset Timing

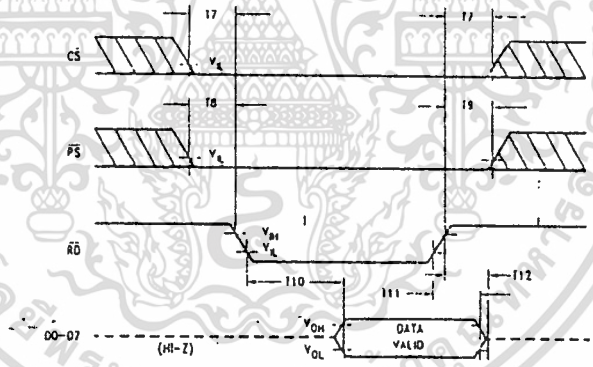


FIGURE 4. Status Byte Read Timing

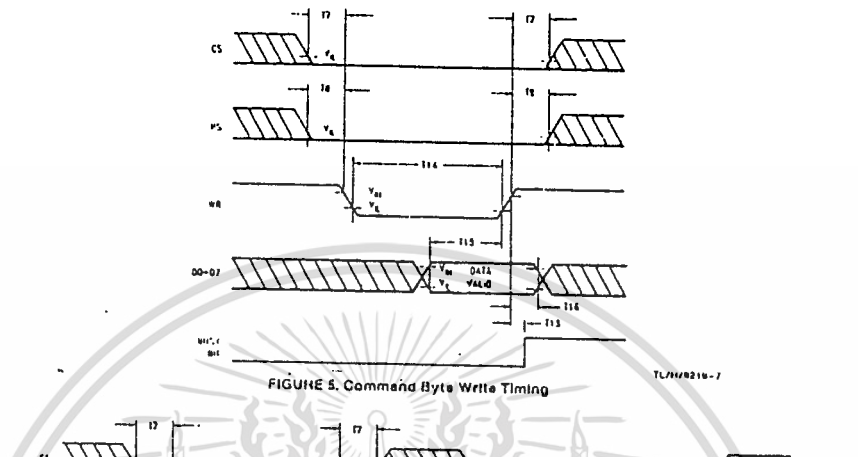


FIGURE 5. Command Byte Write Timing

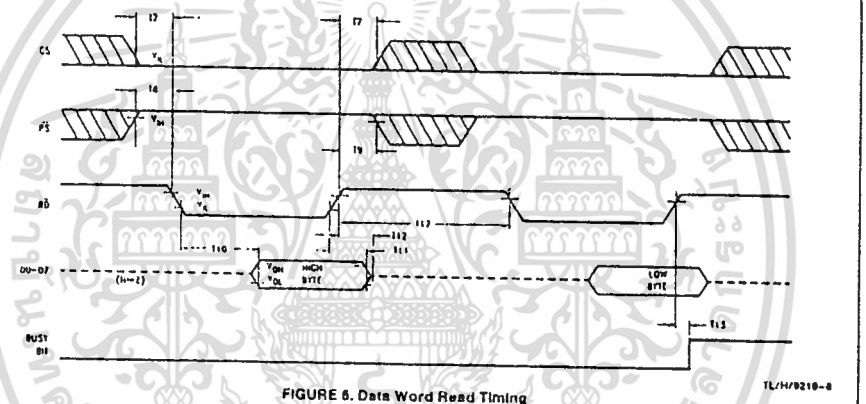


FIGURE 6. Data Word Read Timing

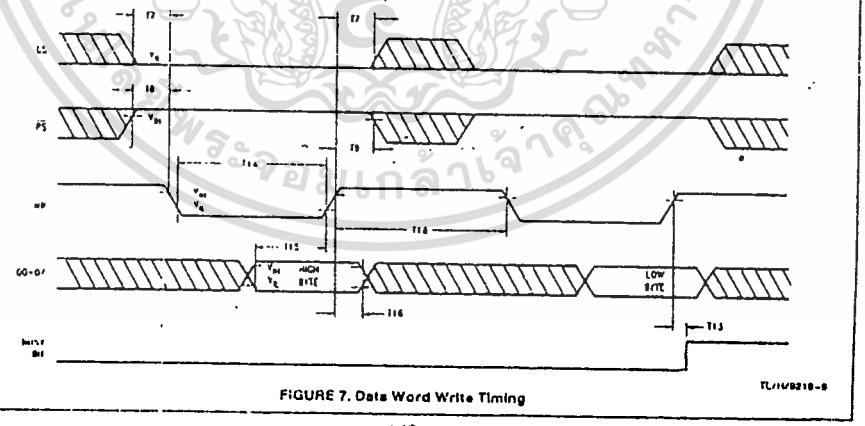


FIGURE 7. Data Word Write Timing

Pinout Description (See Connection Diagrams)

Pin 1, Index ($\overline{IR}$) 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, 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, 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, Chip Select ($\overline{CS}$) Input: Used to select the LM628 for writing and reading operations.

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

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

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

Pin 16, 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, 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 8-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 and a PWM magnitude signal on Pin 19. Pins 20 to 25 are not used in the LM629. *Figure 11* shows the PWM output signal format. Connect pin 25 to ground.

Pin 26, Clock (CLK) Input: Receives 6 MHz system clock.

Pin 27, 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/SBPH 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.

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 'b0'. If this does not occur, perform another reset and repeat the above steps.

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

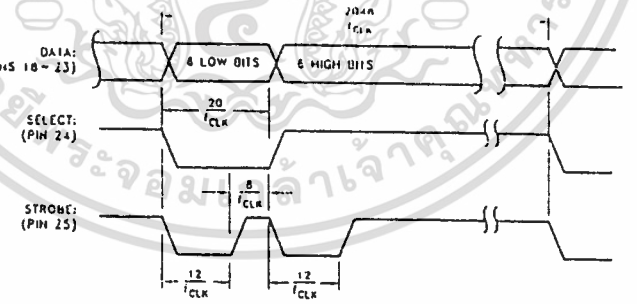


FIGURE 8. 12-Bit Multiplexed Output Timing

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 I 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.

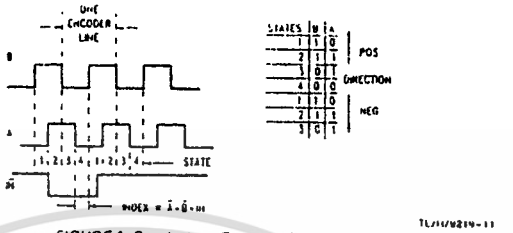
The optional index pulse output provided by some encoders assumes the logic-low state once per revolution. If the LM628 is so programmed by the user, it will record the absolute motor position in a dedicated register (the index register) at the time when all three encoder inputs are logic low.

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. When using the index input in this manner, the user should assure that the index input does not remain logic low during shaft rotation because LM628 internal interrupts are generated every time all three encoder inputs are logic low. These internal interrupts will cause the LM628 to malfunction if the velocity is faster than about 15,000 counts/second (when using a 8 MHz clock, or about 20,000 counts/second with an 8 MHz clock).

TABLE I. System Specifications Summary

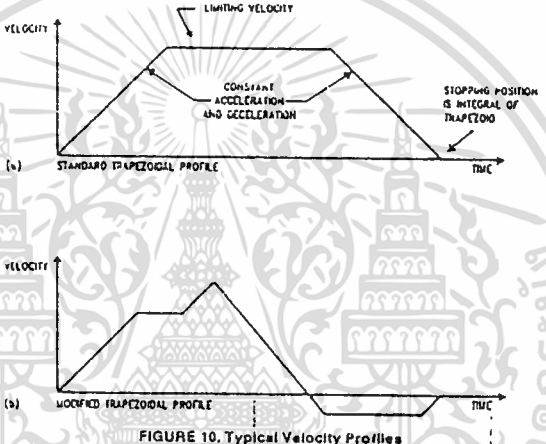
Position Range	−1,073,741,824 to 1,073,741,823 counts
Velocity Range	0 to $1,073,741,823/2^{16}$ counts/sample; ie, 0 to 16,383 counts/sample, with a resolution of $1/2^{16}$ counts/sample
Acceleration Range	0 to $1,073,741,823/2^{16}$ counts/sample/sample; ie, 0 to 16,383 counts/sample/sample, with 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
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: $2048/f_{CLK}$

Theory of Operation (Continued)



TL/H/9219-11

FIGURE 9. Quadrature Encoder Signals



TL/H/9219-12

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 condition goes undetected, and the impeding force on the motor is subsequently removed, 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:

Theory of Operation (Continued)

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:

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

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.

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

Theory of Operation (Continued)

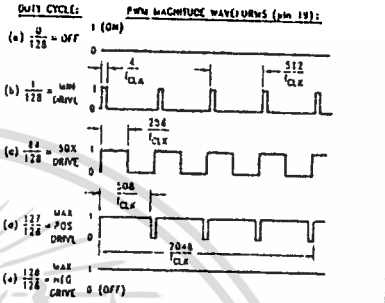
busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). Data transfers are accomplished via LM628-internal interrupts (which are not nestable); 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 *Figures 5* thru *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 8-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 *Figure 8* for details.

The LM628 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.



TL/H/9218-13

Note: Sign output (pin 18) not shown

FIGURE 11. PWM Output Signal Format

TABLE II. 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
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	1
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	00	2	1
RDSUM	Report	Read Integration Sum	0D	2	1

Note 1: Commands may be executed "On the Fly" during motion.

Note 2: Commands not applicable to execution during motion.

Note 3: Command may be executed during motion if acceleration parameter was not changed.

Note 4: 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 II 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

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.

Interrupt Control Commands (Continued)

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 III.

TABLE III. 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 III 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 III. 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

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

(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

Filter Control Commands (Continued)

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 IV.

TABLE IV. 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 V. 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
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 V. Derivative-Term Sampling Interval Selection Codes

Bit Position								Selected Derivative Sampling Interval
15	14	13	12	11	10	9	8	
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: 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.

Trajectory Control Commands (Continued)

The trajectory control parameters which are written to the LM823 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 VI.

TABLE VI. 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 [illegible]
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 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.

Before using LTRJ to issue a new acceleration value, a "motor off" command must first be executed (LTRJ command with bit 8 of the Trajectory Control Word set). This precaution is only necessary if the acceleration value is being changed.

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.

Data Reporting Commands (Continued)

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 VII.

TABLE VII. 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.

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 VIII.

TABLE VIII. 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.

Data Reporting Commands (Continued)

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 Table VII), 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:	02 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.

RDOP COMMAND: ReaD Desired Position

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

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:	C000C000 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:	0C Hex
Bytes Read:	Two
Data Range:	00000 Hex to $\frac{1}{2}$ 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.

Typical Applications (Continued)

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 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 appropriate 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. I, 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. *Figure 17* shows an LM621 brushless motor commutator interfaced to the LM629 PWM outputs and a discrete device switch-mode power amplifier for driving brushless DC 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 risetime 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 18* shows circuitry using the DS26LS31 and DS26LS32.

Typical Applications (Continued)

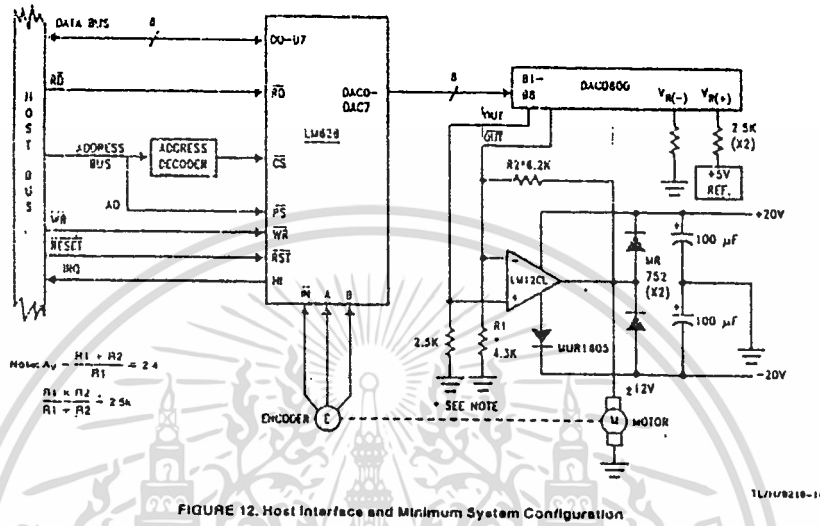


FIGURE 12. Host Interface and Minimum System Configuration

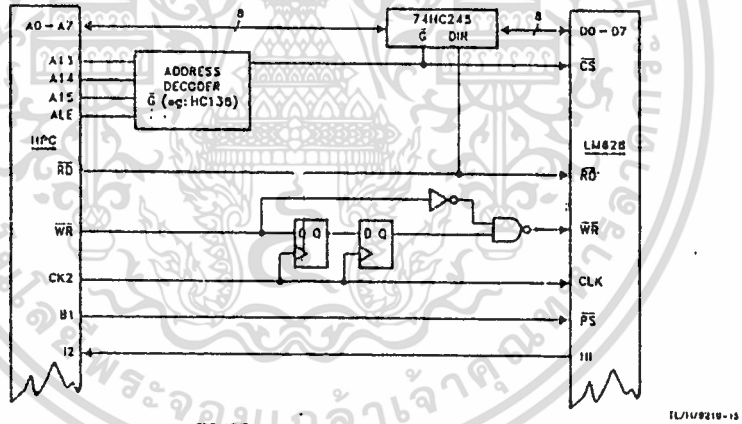
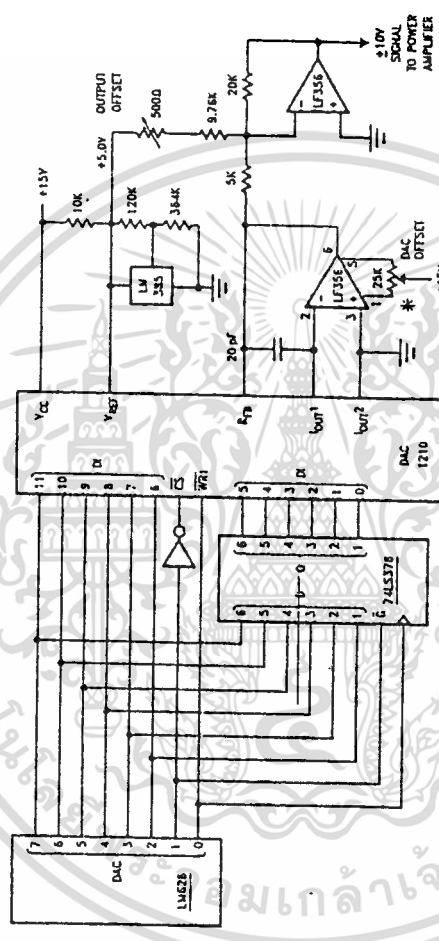


FIGURE 13. LM628 and HPC Interface

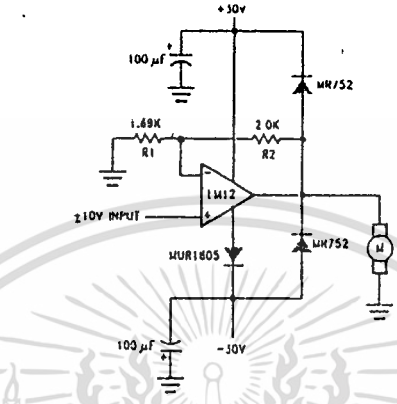
Typical Applications (Continued)



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

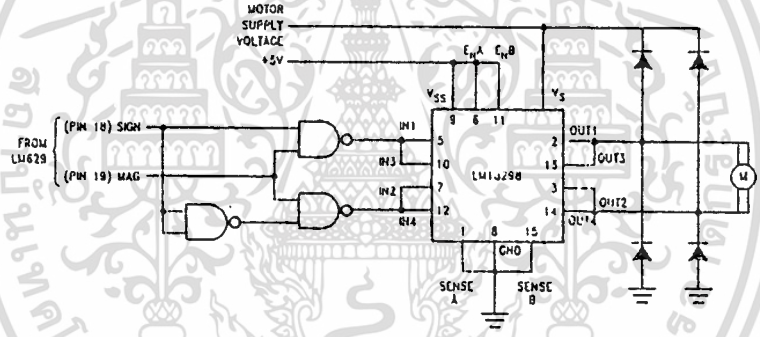
FIGURE 14. Interfacing a 12-Bit DAC and LM628

Typical Applications (Continued)



TL/H/9219–17

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



TL/H/9219–18

FIGURE 16. PWM Drive for Brush/Commutator Motors

Typical Applications (Continued)

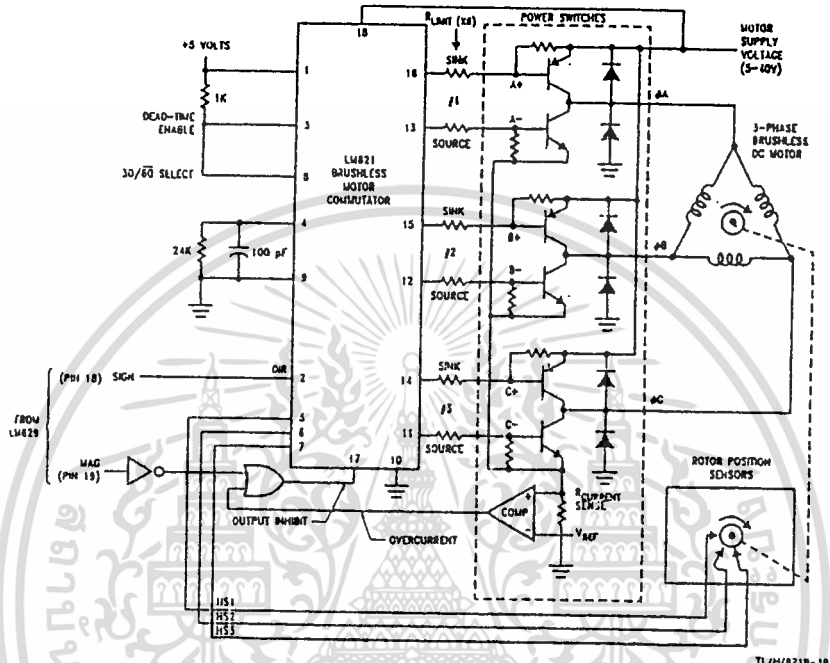


FIGURE 17. PWM Drive for Brushless Motors

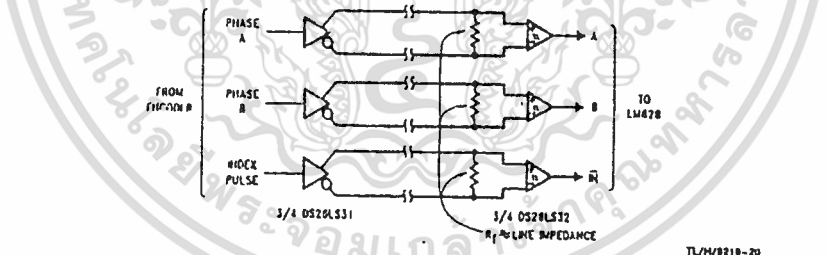


FIGURE 18. Typical Balanced-Line Encoder Input Circuit

ภาคผนวก ค. Data Sheet MLED 81

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

Infrared LED

MLED81

INFRARED
LED
940 nm

This device is designed for infrared remote control and other sensing applications, and can be used in conjunction with the MRD821 photodiode. It features high power output, using long-life gallium arsenide technology.

- Low Cost
- Popular T-1¾ Package
- Ideal Beam Angle for Most Remote Control Applications
- Uses Stable Long-Life LED Technology
- Clear Epoxy Package

CASE 279B-01

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Reverse Voltage	V_R	5	Volts
Forward Current — Continuous	I_F	100	mA
Forward Current — Peak Pulse	I_F	1	A
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	100 2.2	mW mW/°C
Ambient Operating Temperature Range	T_A	−30 to +70	°C
Storage Temperature	T_{stg}	−30 to +80	°C
Lead Soldering Temperature, 5 seconds max, 1/16 inch from case	—	260	°C

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Reverse Leakage Current (V_R = 3 V)	I_R	—	10	—	nA
Reverse Leakage Current (V_R = 5 V)	I_R	—	1	10	μA
Forward Voltage (I_F = 100 mA)	V_F	—	1.35	1.7	V
Temperature Coefficient of Forward Voltage	ΔV_F	—	−1.6	—	mV/K
Capacitance (f = 1 MHz)	C	—	25	—	pF

OPTICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
Peak Wavelength (I_F = 100 mA)	λ_P	—	940	—	nm
Spectral Half-Power Bandwidth	$\Delta\lambda$	—	50	—	nm
Total Power Output (I_F = 100 mA)	Φ_e	—	16	—	mW
Temperature Coefficient of Total Power Output	$\Delta\Phi_e$	—	−0.25	—	%/K
Axial Radiant Intensity (I_F = 100 mA)	I_e	10	15	—	mW/sr
Temperature Coefficient of Axial Radiant Intensity	ΔI_e	—	−0.25	—	%/K
Power Half-Angle	θ	—	±30	—	°

MLED81

TYPICAL CHARACTERISTICS

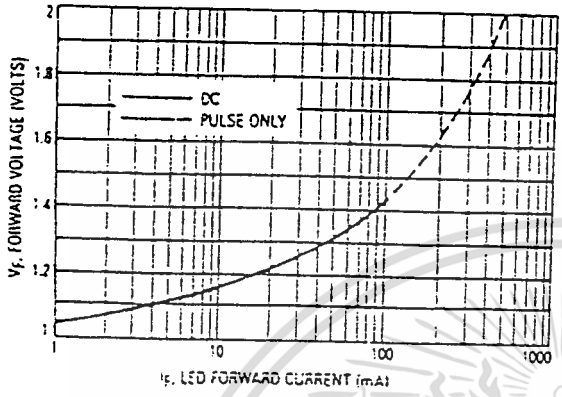


Figure 1. LED Forward Voltage versus Forward Current

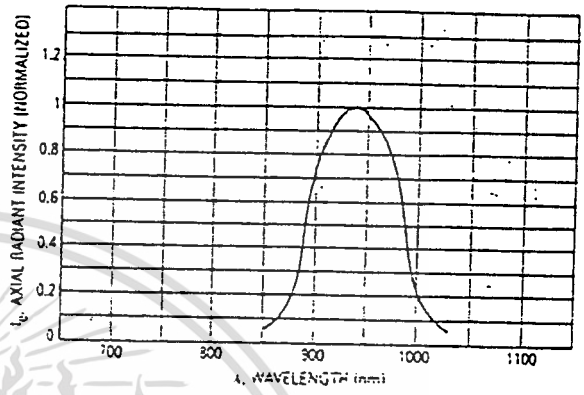


Figure 2. Relative Spectral Emission

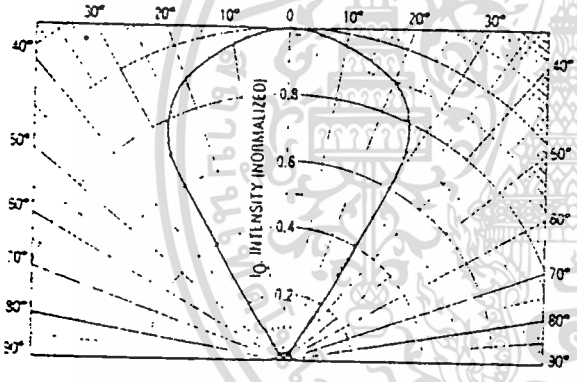


Figure 3. Spatial Radiation Pattern

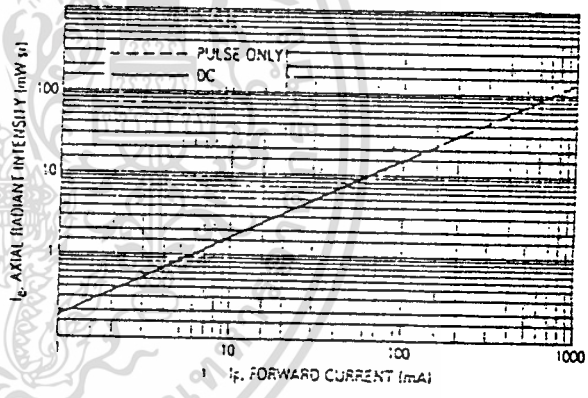


Figure 4. Intensity versus Forward Current

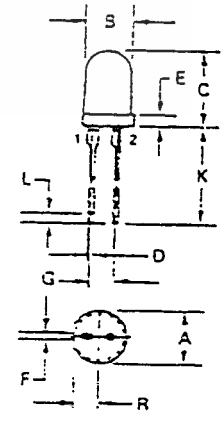
OUTLINE DIMENSIONS

CASE 2798-01

DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	5.62	5.56	0.217	0.235
B	4.80	5.20	0.189	0.205
C	8.13	9.14	0.320	0.360
D	0.51	0.71	0.020	0.028
E	1.15	1.39	0.045	0.055
F	0.51	0.76	0.020	0.030
G	2.29	2.79	0.090	0.110
K	25.40	25.57	1.00	1.06
L	0.18	1.82	0.007	0.072
R	2.42	2.79	0.095	0.110

NOTES.
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.

STYLE 1
PIN 1. CATHODE
2. ANODE



ภาคผนวก ง. Data Sheet MRD 750

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

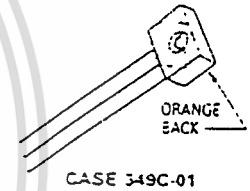
Photo Detector
Logic Output

MRD750

PHOTO DETECTOR
LOGIC OUTPUT

. incorporates a Schmitt Trigger which provides hysteresis for noise immunity and pulse shaping. The detector circuit is optimized for simplicity of operation and utilizes an open-collector output for application flexibility.

- Popular Low Cost Plastic Package
- High Coupling Efficiency
- Wide V_{CC} Range
- Ideally Suited for MLED71 Emitter
- Usable to 125 kHz



CASE 349C-01
PLASTIC

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

Rating	Symbol	Value	Unit
Output Voltage Range	V_O	0–16	Volts
Supply Voltage Range	V_{CC}	0–16	Volts
Output Current	I_O	50	mA
Device Dissipation Derate above 25°C (Note 1)	P_D	150 2	mW mW/°C
Maximum Operating Temperature	T_A	- 40 to + 85	°C
Storage Temperature Range	T_{stg}	- 40 to + 100	°C
Lead Soldering Temperature (5 seconds maximum; 1/16 inch from case) (Note 2)	T_L	260	°C

Notes: 1. Measured with device soldered into a typical PC board.
2. Heat sink should be applied to leads during soldering to prevent case temperature from exceeding 100°C.

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit
DEVICE (T_A = 25°C)					
Operating Voltage	V_{CC}	3	—	15	Volts
Supply Current with Output High (Figure 4 (I_F = 0, V_{CC} = 5 V)	$I_{CC(off)}$	—	1.3	5	mA
Output Current, High (I_F = 0, V_{CC} = V_O = 15 V, R_L = 270 Ω)	I_{OH}	—	—	100	μA

(continued)

MRD750

TYPICAL CHARACTERISTICS

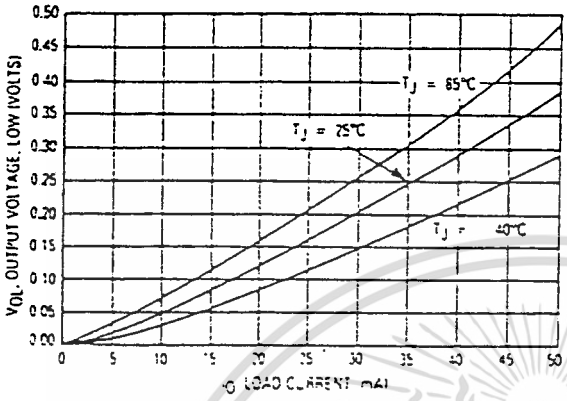


Figure 3. Output Voltage, Low versus Load Current

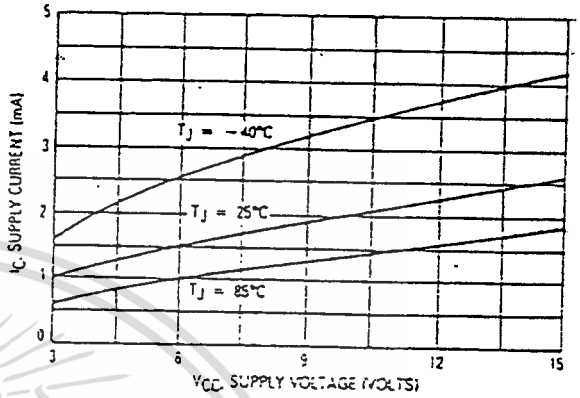


Figure 4. Supply Current versus Supply Voltage — Output High

TYPICAL COUPLED CHARACTERISTICS USING MLED71 EMITTER AND MRD750 DIGITAL OUTPUT DETECTOR

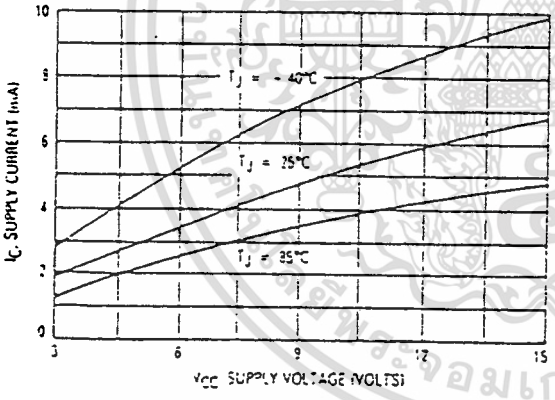


Figure 5. Supply Current versus Supply Voltage — Output Low

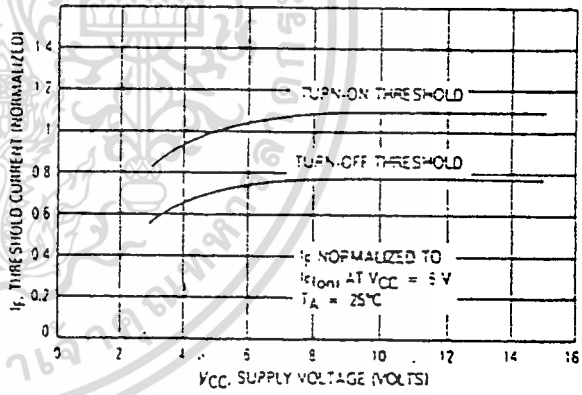


Figure 6. Threshold Current versus Supply Voltage

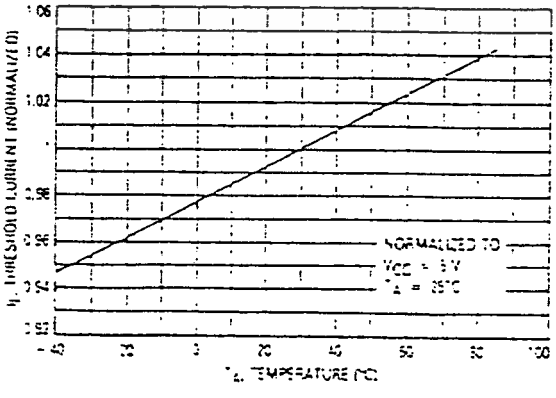


Figure 7. Threshold Current versus Temperature

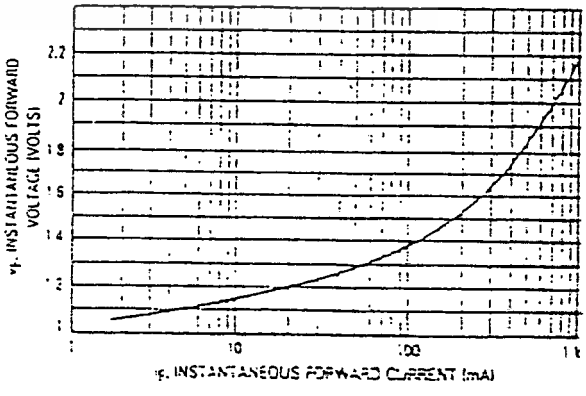


Figure 8. MLED71 Forward Characteristics

MRD750

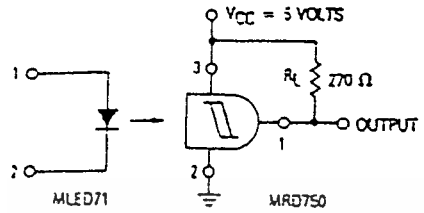


Figure 9. Test Circuit for Threshold Current Measurements

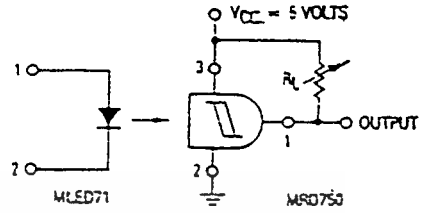


Figure 10. Test Circuit for Output Voltage versus Load Current Measurements

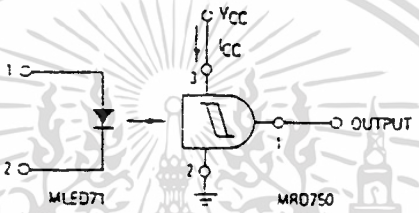
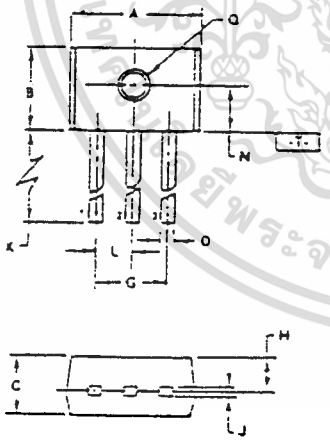


Figure 11. Test Circuit for Supply Current versus Supply Voltage Measurements



STYLE 3
PIN 1: OUTPUT
2: GROUND
3 VCC

NOTES
1 DIMENSIONS A, B AND C ARE DATUMS
2 POSITIONAL TOLERANCE FOR D DIMENSION
3 POSITIONAL TOLERANCE FOR Q DIAMETER
4 T IS A SEATING LANE
5 DIMENSIONING AND TOLERANCING PER ANSI 14.5, 1973

DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	9.43	9.66	0.371	0.380
B	2.79	3.30	0.110	0.130
C	4.23	4.78	0.166	0.188
D	0.43	0.56	0.017	0.022
G	2.54 BSC		0.100 BSC	
M	1.52 BSC		0.060 BSC	
J	0.23	0.36	0.009	0.014
K	12.70	—	0.500	—
L	7.17 BSC		0.260 BSC	
N	4.78 BSC		0.188 BSC	
Q	3.56	3.92	0.140	0.160

CASE 349C-01
PLASTIC